REJ09B0001-0100Z

RENESAS

16

R8C/Tiny Series

Software Manual

RENESAS 16-BIT SINGLE-CHIP MICROCOMPUTER

Software Manual

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

Rev. 1.00
Revision date: Jun. 19, 2003

RenesasTechnology
www.renesas.com

Keep safety first in your circuit designs!

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

Notes regarding these materials

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

Using This Manual

This manual is written for the R8C/Tiny series software. This manual can be used for all types of microcomputers having the R8C/Tiny series CPU core.
The reader of this manual is expected to have the basic knowledge of electric and logic circuits and microcomputers.
This manual consists of six chapters. The following lists the chapters and sections to be referred to when you want to know details on some specific subject.

- To understand the outline of the R8C/Tiny series and its features ..... **Chapter 1, "Overview"**
- To understand the operation of each addressing mode ..... **Chapter 2, "Addressing Modes"**
- To understand instruction functions (Syntax, operation, function, selectable src/dest (label), flag changes, description example, related instructions) ..... **Chapter 3, "Functions"**
- To understand instruction code and cycles ..... **Chapter 4, "Instruction Code/Number of Cycles"**
- To understand instruction interrupt ..... **Chapter 5, "Interrupt"**
- To understand calculation number of cycles ..... **Chapter 6, "Calculation Number of Cycles"**

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

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

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

M16C Family Documents

M16C family supports the following documents;

Type of documents	Contents
Short sheet	Overview of hardware
Data sheet	Overview of hardware, electrical characteristics
Hardware Manual	Hardware specifications (pin assignment, memory map, specifications of peripheral functions, electrical characteristics, timing chart)
Software Manual	Detailed description about operation of instruction (assembly language)
Application Note	Application example of peripheral function Sample program Method for creating programs using assembly and C languages

Table of Contents

Quick Reference in Alphabetic Order

Mnemonic	See page for function	See page for instruction code /number of cycles
ABS	39	138
ADC	40	138
ADCF	41	140
ADD	42	140
ADJNZ	44	146
AND	45	147
BAND	47	150
BCLR	48	150
BMCnd	49	152
BMEQ/Z	49	152
BMGE	49	152
BMGEU/C	49	152
BMGT	49	152
BMGTU	49	152
BMLE	49	152
BMLEU	49	152
BMLT	49	152
BMLTU/NC	49	152
BMN	49	152
BMNE/NZ	49	152
BMNO	49	152
BMO	49	152
BMPZ	49	152
BNAND	50	153
BNOR	51	154
BNOT	52	154
BNTST	53	155
BNXOR	54	156
BOR	55	156
BRK	56	157
BSET	57	157
BTST	58	158
BTSTC	59	159
BTSTS	60	160
BXOR	61	160
CMP	62	161
DADC	64	165
DADD	65	167
DEC	66	169
DIV	67	170

Mnemonic	See page for function	See page for instruction code /number of cycles
DIVU	68	171
DIVX	69	172
DSBB	70	173
DSUB	71	175
ENTER	72	177
EXITD	73	178
EXTS	74	178
FCLR	75	179
FSET	76	180
INC	77	180
INT	78	181
INTO	79	182
J*Cnd*	80	182
JEQ/Z	80	182
JGE	80	182
JGEU/C	80	182
JGT	80	182
JGTU	80	182
JLE	80	182
JLEU	80	182
JLT	80	182
JLTU/NC	80	182
JN	80	182
JNE/NZ	80	182
JNO	80	182
JO	80	182
JPZ	80	182
JMP	81	183
JMPI	82	185
JSR	83	187
JSRI	84	188
LDC	85	189
LDCTX	86	191
LDE	87	191
LDINTB	88	192
LDIPL	89	193
MOV	90	193
MOVA	92	200

Quick Reference in Alphabetic Order

Mnemonic	See page for function	See page for instruction code /number of cycles	Mnemonic	See page for function	See page for instruction code /number of cycles
MOV*Dir*	93	201	ROT	112	220
MOVHH	93	201	RTS	113	221
MOVHL	93	201	SBB	114	222
MOVLH	93	201	SBJNZ	115	224
MOVLL	93	201	SHA	116	225
MUL	94	203	SHL	117	228
MULU	95	205	SMOVB	118	230
NEG	96	207	SMOVF	119	231
NOP	97	207	SSTR	120	231
NOT	98	208	STC	121	232
OR	99	209	STCTX	122	233
POP	101	211	STE	123	233
POPC	102	213	STNZ	124	235
POPM	103	213	STZ	125	235
PUSH	104	214	STZX	126	236
PUSHA	105	216	SUB	127	236
PUSHC	106	216	TST	129	239
PUSHM	107	217	UND	130	241
REIT	108	217	WAIT	131	241
RMPA	109	218	XCHG	132	242
ROLC	110	218	XOR	133	243
RORC	111	219			

Quick Reference by Function

Function	Mnemonic	Content	See page for function	See page for instruction code /number of cycles
Transfer	MOV	Transfer	90	193
	MOVA	Transfer effective address	92	200
	MOVDir	Transfer 4-bit data	93	201
	POP	Restore register/memory	101	211
	POPM	Restore multiple registers	103	213
	PUSH	Save register/memory/immediate data	104	214
	PUSHA	Save effective address	105	216
	PUSHM	Save multiple registers	107	217
	LDE	Transfer from extended data area	87	191
	STE	Transfer to extended data area	123	233
	STNZ	Conditional transfer	124	235
	STZ	Conditional transfer	125	235
	STZX	Conditional transfer	126	236
	XCHG	Exchange	132	242
Bit manipulation	BAND	Logically AND bits	47	150
	BCLR	Clear bit	48	150
	BM*Cnd*	Conditional bit transfer	49	152
	BNAND	Logically AND inverted bits	50	153
	BNOR	Logically OR inverted bits	51	154
	BNOT	Invert bit	52	154
	BNTST	Test inverted bit	53	155
	BNXOR	Exclusive OR inverted bits	54	156
	BOR	Logically OR bits	55	156
	BSET	Set bit	57	157
	BTST	Test bit	58	158
	BTSTC	Test bit & clear	59	159
	BTSTS	Test bit & set	60	160
	BXOR	Exclusive OR bits	61	160
Shift	ROLC	Rotate left with carry	110	218
	RORC	Rotate right with carry	111	219
	ROT	Rotate	112	220
	SHA	Shift arithmetic	116	215
	SHL	Shift logical	117	228
Arithmetic	ABS	Absolute value	39	138
	ADC	Add with carry	40	138
	ADCF	Add carry flag	41	140
	ADD	Add without carry	42	140
	CMP	Compare	62	161
	DADC	Decimal add with carry	64	165

Quick Reference by Function

Function	Mnemonic	Content	See page for function	See page for instruction code /number of cycles
Arithmetic	DADD	Decimal add without carry	65	167
	DEC	Decrement	66	169
	DIV	Signed divide	67	170
	DIVU	Unsigned divide	68	171
	DIVX	Singed divide	69	172
	DSBB	Decimal subtract with borrow	70	173
	DSUB	Decimal subtract without borrow	71	175
	EXTS	Extend sign	74	178
	INC	Increment	77	180
	MUL	Signed multiply	94	203
	MULU	Unsigned multiply	95	205
	NEG	Two's complement	96	207
	RMPA	Calculate sum-of-products	109	218
	SBB	Subtract with borrow	114	222
	SUB	Subtract without borrow	127	236
Logical	AND	Logical AND	45	147
	NOT	Invert all bits	98	208
	OR	Logical OR	99	209
	TST	Test	129	239
	XOR	Exclusive OR	133	243
Jump	ADJNZ	Add & conditional jump	44	146
	SBJNZ	Subtract & conditional jump	115	224
	JCnd	Jump on condition	80	182
	JMP	Unconditional jump	81	184
	JMPI	Jump indirect	82	185
	JSR	Subroutine call	83	187
	JSRI	Indirect subroutine call	84	188
	RTS	Return from subroutine	113	221
String	SMOVB	Transfer string backward	118	230
	SMOVF	Transfer string forward	119	231
	SSTR	Store string	120	231
Other	BRK	Debug interrupt	56	157
	ENTER	Build stack frame	72	177
	EXITD	Deallocate stack frame	73	178
	FCLR	Clear flag register bit	75	179
	FSET	Set flag register bit	76	180
	INT	Interrupt by INT instruction	78	181
	INTO	Interrupt on overflow	79	182
	LDC	Transfer to control register	85	189
	LDCTX	Restore context	86	189
	LDINTB	Transfer to INTB register	88	192

Quick Reference by Function

Function	Mnemonic	Content	See page for function	See page for instruction code /number of cycles
Other	LDIPL	Set interrupt enable level	89	193
	NOP	No operation	97	207
	POPC	Restore control register	102	213
	PUSHC	Save control register	106	216
	REIT	Return from interrupt	108	216
	STC	Transfer from control register	121	232
	STCTX	Save context	122	233
	UND	Interrupt for undefined instruction	130	241
	WAIT	Wait	131	241

Quick Reference by Addressing (general instruction addressing)

Mnemonic	Addressing															See page for function	See page for instruction code /number of cycles
	R0L/R0	R0H/R1	R1L/R2	R1H/R3	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16	#IMM8	#IMM16	#IMM20	#IMM		
ABS	○	○	○	○	○	○	○	○	○	○	○					39	138
ADC	○	○	○	○	○	○	○	○	○	○	○	○	○			40	138
ADCF	○	○	○	○	○	○	○	○	○	○	○					41	140
ADD*1	○	○	○	○	○	○	○	○	○	○	○	○	○			42	140
ADJNZ*1	○	○	○	○	○	○	○	○	○	○	○				○	44	146
AND	○	○	○	○	○	○	○	○	○	○	○	○	○			45	147
CMP	○	○	○	○	○	○	○	○	○	○	○	○	○			62	161
DADC	○	○										○	○			64	165
DADD	○	○										○	○			65	167
DEC	○	○			○			○			○					66	169
DIV	○	○	○	○	○	○	○	○	○	○	○	○	○			67	170
DIVU	○	○	○	○	○	○	○	○	○	○	○	○	○			68	171
DIVX	○	○	○	○	○	○	○	○	○	○	○	○	○			69	172
DSBB	○	○										○	○			70	173
DSUB	○	○										○	○			71	175
ENTER												○				72	177
EXTS	○		○*2			○	○	○	○	○	○					74	178
INC	○*3	○*4			○			○			○					77	180
INT															○	78	181
JMPI*1	○	○	○	○	○	○	○	○		○	○					82	185
JSRI*1	○	○	○	○	○	○	○	○		○	○					83	187
LDC*1	○	○	○	○	○	○	○	○	○	○	○		○			85	189
LDE*1	○	○	○	○	○	○	○	○	○	○	○					87	191
LDINTB														○		88	192
LDIPL															○	89	193

*1 Has special instruction addressing.

*2 Only R1L can be selected.

*3 Only R0L can be selected.

*4 Only R0H can be selected.

Quick Reference by Addressing (general instruction addressing)

Mnemonic	Addressing															See page for function	See page for instruction code /number of cycles
	R0L/R0	R0H/R1	R1L/R2	R1H/R3	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16	#IMM8	#IMM16	#IMM20	#IMM		
MOV*1	○	○	○	○	○	○	○	○	○	○	○	○	○			90	193
MOVA	○	○	○	○	○		○	○	○	○	○					92	200
MOV*Dir*	○	○	○	○		○	○	○	○	○	○					93	201
MUL	○	○	○	○	○	○	○	○	○	○	○	○	○			94	203
MULU	○	○	○	○	○	○	○	○	○	○	○	○	○			95	205
NEG	○	○	○	○	○	○	○	○	○	○	○					96	207
NOT	○	○	○	○	○	○	○	○	○	○	○					98	208
OR	○	○	○	○	○	○	○	○	○	○	○	○	○			99	209
POP	○	○	○	○	○	○	○	○	○	○	○					101	211
POPM*1	○	○	○	○	○											103	213
PUSH	○	○	○	○	○	○	○	○	○	○	○					104	214
PUSHA							○	○	○	○	○					105	216
PUSHM*1	○	○	○	○	○											107	217
ROLC	○	○	○	○	○	○	○	○	○	○	○					110	218
RORC	○	○	○	○	○	○	○	○	○	○	○					111	219
ROT	○	○	○	○	○	○	○	○	○	○	○				○	112	220
SBB	○	○	○	○	○	○	○	○	○	○	○	○	○			114	222
SBJNZ*1	○	○	○	○	○	○	○	○	○	○	○				○	115	224
SHA*1	○	○	○	○	○	○	○	○	○	○	○				○	116	225
SHL*1	○	○	○	○	○	○	○	○	○	○	○				○	117	228
STC*1	○	○	○	○	○	○	○	○	○	○	○					121	232
STCTX*1											○					122	233
STE*1	○	○	○	○	○	○	○	○	○	○	○					123	233
STNZ	○	○						○			○	○				124	235
STZ	○	○						○			○	○				125	235

*1 Has special instruction addressing.

Quick Reference by Addressing (general instruction addressing)

Mnemonic	Addressing															See page for function	See page for instruction code /number of cycles
	R0L/R0	R0H/R1	R1L/R2	R1H/R3	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16	#IMM8	#IMM16	#IMM20	#IMM		
STZX	○	○						○			○	○				126	236
SUB	○	○	○	○	○	○	○	○	○	○	○	○	○			127	236
TST	○	○	○	○	○	○	○	○	○	○	○	○	○			129	239
XCHG	○	○	○	○	○	○	○	○	○	○	○					132	242
XOR	○	○	○	○	○	○	○	○	○	○	○	○	○			133	243

Quick Reference by Addressing (special instruction addressing)

Mnemonic	Addressing													See page for function	See page for instruction code /number of cycles
	dsp:20[A0]	dsp:20[A1]	abs20	R2R0/R3R1	A1A0	[A1A0]	dsp:8[SP]	label	SB/FB	ISP/USP	FLG	INTBL/INTBH	PC		
ADD*1										○				42	140
ADJNZ*1								○						44	146
JCnd								○						80	182
JMP			○					○						81	184
JMPI*1	○	○		○	○									82	185
JSR			○					○						83	187
JSRI*1	○	○		○	○									84	188
LDC*1									○	○	○	○		85	189
LDCTX			○											86	189
LDE*1	○		○			○								87	191
LDINTB												○*4		88	192
MOV*1							○							90	193
POPC									○	○	○	○		102	213
POPM*1									○					103	213
PUSHC									○	○	○	○		106	216
PUSHM*1									○					107	217
SBJNZ*1								○						115	224
SHA*1				○										116	225
SHL*1				○										117	228
STC*1				○	○				○	○	○	○	○	121	232
STCTX*1			○											122	233
STE*1	○		○			○								123	233

*1 Has general instruction addressing.

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

Quick Reference by Addressing (bit instruction addressing)

Mnemonic	Addressing										See page for function	See page for instruction code /number of cycles
	bit,Rn	bit,An	[An]	base:8[An]	bit,base:8[SB/FB]	base:16[An]	bit,base:16[SB]	bit,base:16	bit,base:11	U/I/O/B/S/Z/D/C		
BAND	○	○	○	○	○	○	○	○			47	150
BCLR	○	○	○	○	○	○	○	○	○		48	150
BM*Cnd*	○	○	○	○	○	○	○	○		○	49	152
BNAND	○	○	○	○	○	○	○	○			50	153
BNOR	○	○	○	○	○	○	○	○			21	154
BNOT	○	○	○	○	○	○	○	○	○		52	154
BNTST	○	○	○	○	○	○	○	○			53	155
BNXOR	○	○	○	○	○	○	○	○			54	156
BOR	○	○	○	○	○	○	○	○			55	156
BSET	○	○	○	○	○	○	○	○	○		57	157
BTST	○	○	○	○	○	○	○	○	○		58	158
BTSTC	○	○	○	○	○	○	○	○			59	159
BTSTS	○	○	○	○	○	○	○	○			60	160
BXOR	○	○	○	○	○	○	○	○			61	160
FCLR										○	75	179
FSET										○	76	180

Chapter 1

Overview

1.1 Features of R8C/Tiny Series

The R8C/Tiny series is single-chip microcomputer developed for built-in applications where the microcomputer is built into applications equipment.
The R8C/Tiny series support instructions suitable for the C language with frequently used instructions arranged in one- byte op-code. Therefore, it allows you for efficient program development with few memory capacity regardless of whether you are using the assembly language or C language. Furthermore, some instructions can be executed in clock cycle, making fast arithmetic processing possible.
Its instruction set consists of 89 discrete instructions matched to the R8C's abundant addressing modes. This powerful instruction set allows to perform register-register, register-memory, and memory-memory operations, as well as arithmetic/logic operations on bits and 4-bit data.
Some R8C/Tiny series models incorporate a multiplier, allowing for high-speed computation.

1.1.1 Features of R8C/Tiny series

●Register configuration

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

●Versatile instruction set

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

●Fast instruction execution time

Shortest 1-cycle instructions: 89 instructions include 20 1-cycle instructions.
(Approximately 75% of instructions execute in five cycles or under.)

1.1.2 Speed performance

Register-register transfer 0.1 µs
Register-memory transfer 0.1 µs
Register-register addition/subtraction 0.1 µs
8 bits x 8 bits register-register operation 0.2 µs
16 bits x 16 bits register-register operation 0.250 µs
16 bits / 8 bits register-register operation 0.904 µs
32 bits / 16 bits register-register operation 1.248 µs

•Conditions
-Products with built-in Multiplier
-Clock frequency 20 MHz

1.2 Address Space

Fig. 1.2.1 shows an address space.

Addresses 00000_{16} through $002FF_{16}$ make up an SFR (special function register) area. In individual models of the R8C/Tiny series, the SFR area extends from $002FF_{16}$ toward lower addresses.

Addresses from 00400_{16} on make up a memory area. In individual models of the R8C/Tiny series, a RAM area extends from address 00400_{16} toward higher addresses, and a ROM area extends from $0FFFF_{16}$ toward lower addresses. Addresses $0FFDC_{16}$ through $0FFFF_{16}$ make up a fixed vector area.

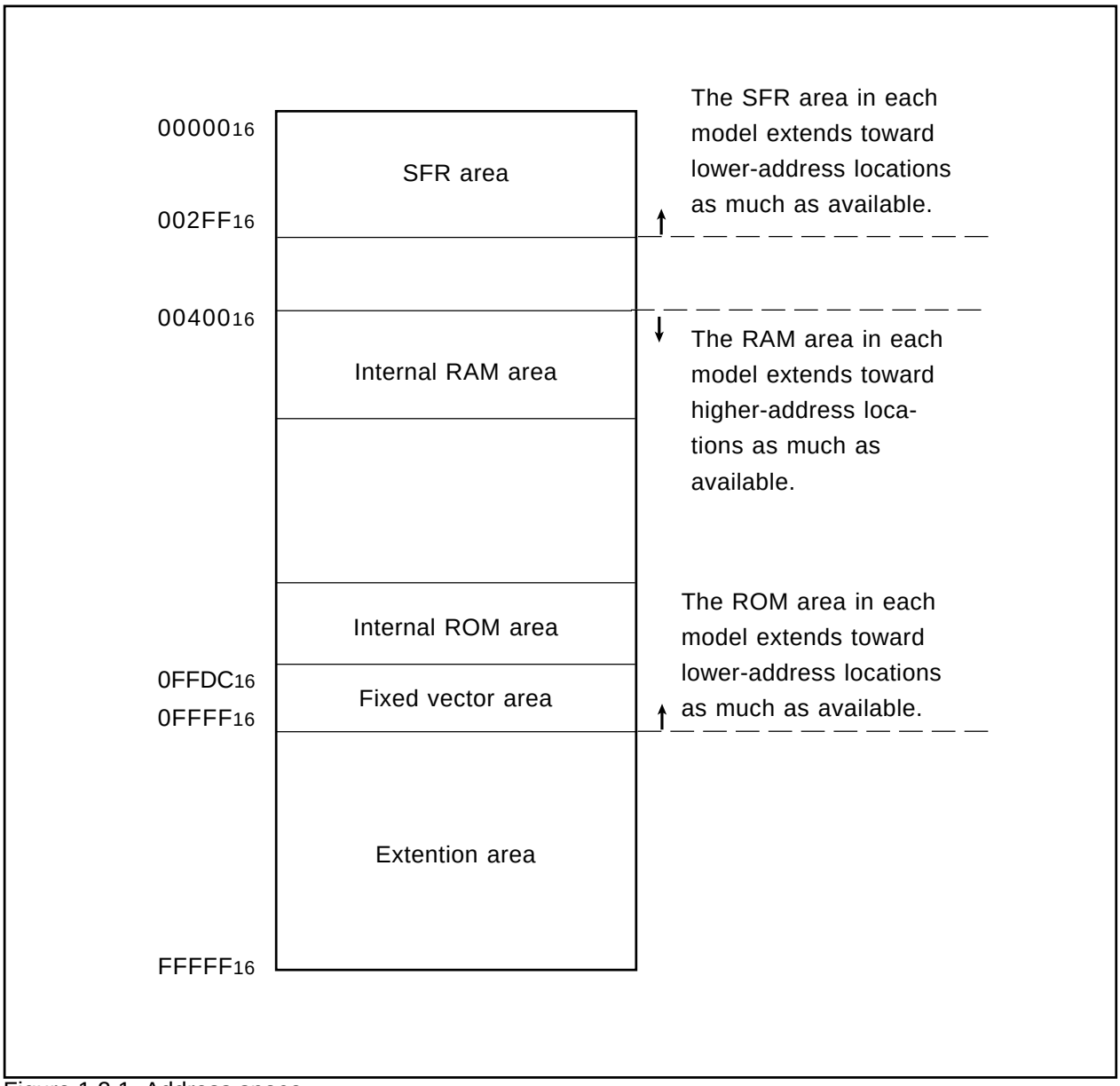


Figure 1.2.1 Address space

1.3 Register Configuration

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

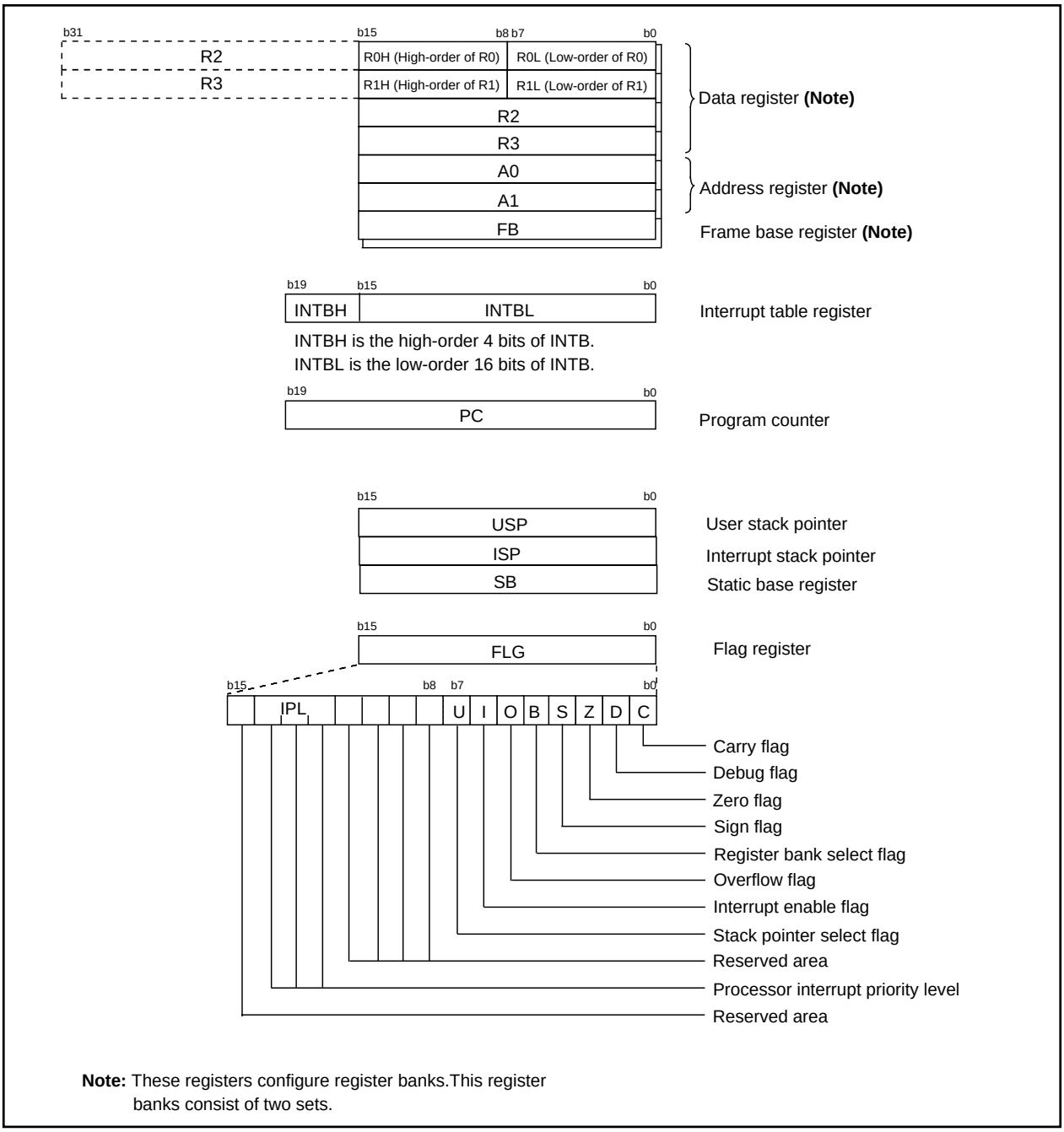


Figure 1.3.1 CPU register configuration

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

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

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

1.3.2 Address registers (A0 and A1)

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

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

1.3.3 Frame base register (FB)

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

1.3.4 Program counter (PC)

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

1.3.5 Interrupt table register (INTB)

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

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

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

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

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

1.3.7 Static base register (SB)

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

1.3.8 Flag register (FLG)

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

1.4 Flag Register (FLG)

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

1.4.1 Bit 0: Carry flag (C flag)

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

1.4.2 Bit 1: Debug flag (D flag)

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

1.4.3 Bit 2: Zero flag (Z flag)

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

1.4.4 Bit 3: Sign flag (S flag)

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

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

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

1.4.6 Bit 5: Overflow flag (O flag)

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

1.4.7 Bit 6: Interrupt enable flag (I flag)

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

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

When this flag is 0, the interrupt stack pointer (ISP) is selected; when the flag is 1, the user stack pointer (USP) is selected.
This flag is cleared to 0 when a hardware interrupt is acknowledged or an INT instruction of software interrupt numbers 0 to 31 is executed.

1.4.9 Bits 8-11: Reserved area

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

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

1.4.11 Bit 15: Reserved area

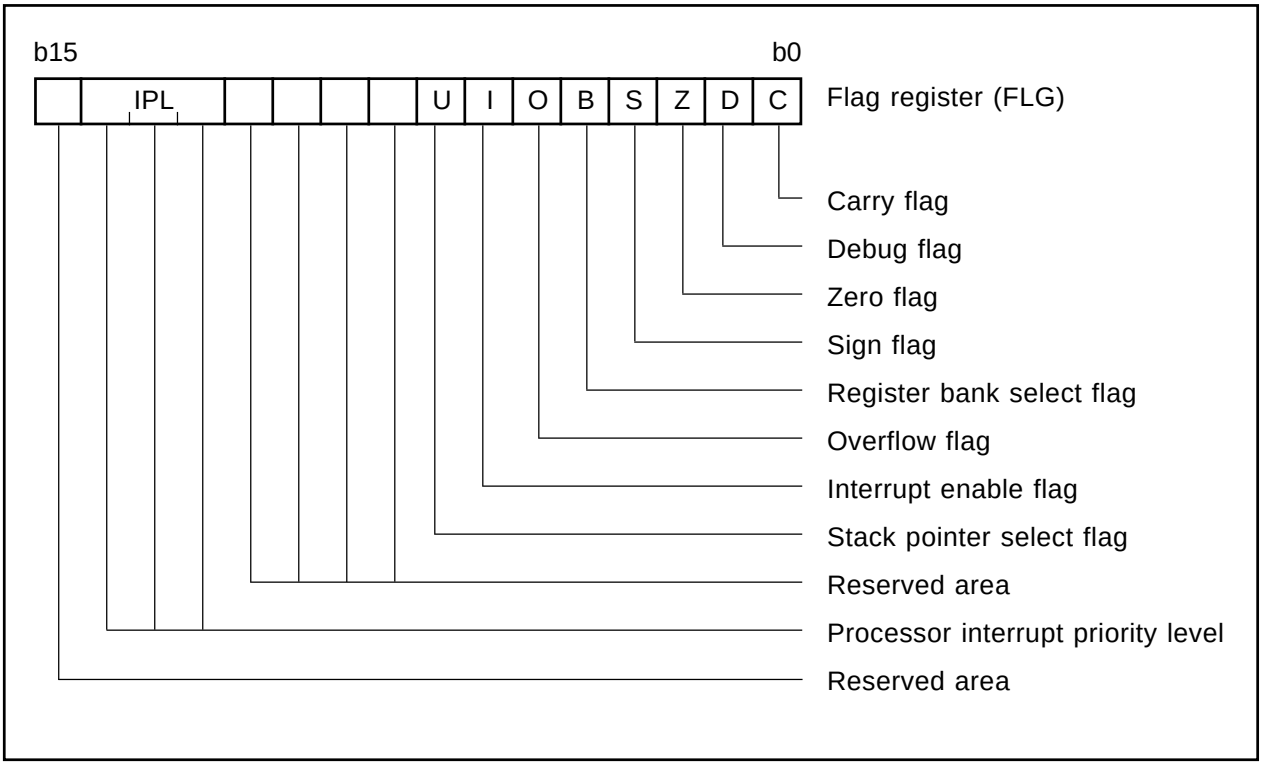


Figure 1.4.1 Configuration of flag register (FLG)

1.5 Register Bank

The R8C/Tiny has two register banks, each configured with data registers (R0, R1, R2, and R3), address registers (A0 and A1), and frame base register (FB). These two register banks are switched over by the register bank select flag (B flag) of the flag register (FLG).
Figure 1.5.1 shows a configuration of register banks.

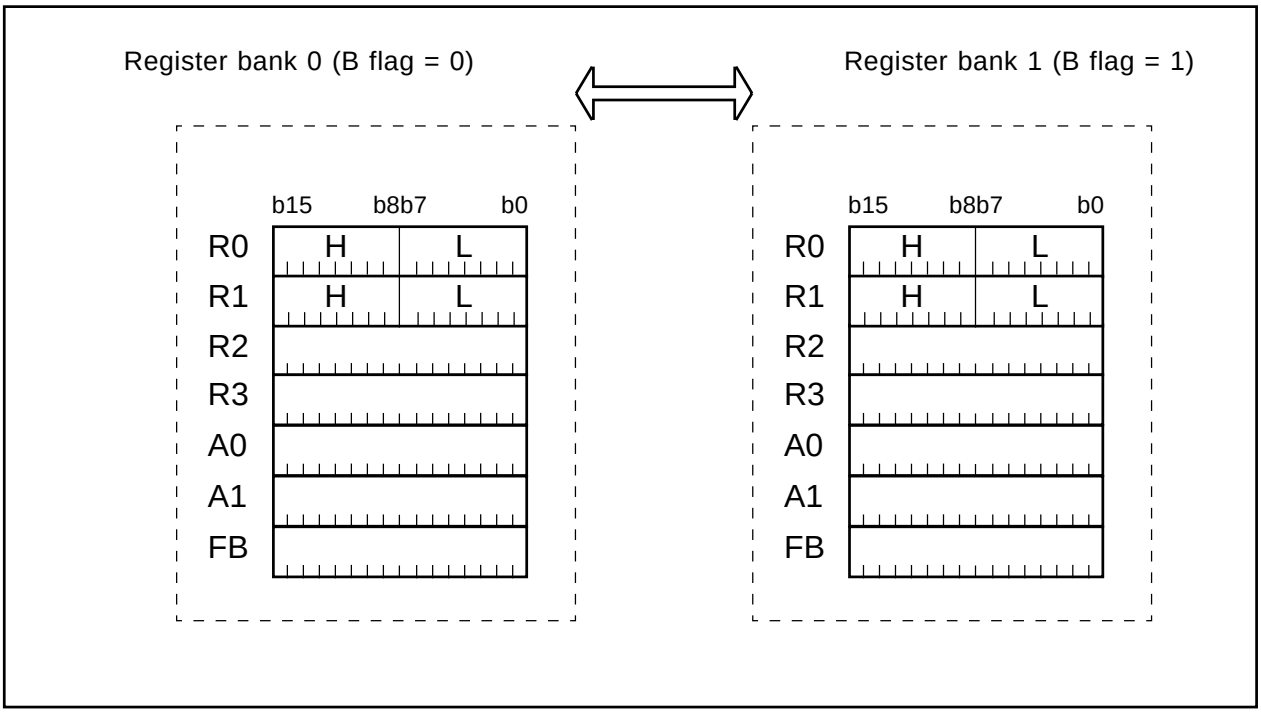


Figure 1.5.1 Configuration of register banks

1.6 Internal State after Reset is Cleared

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

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

1.7 Data Types

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

1.7.1 Integer

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

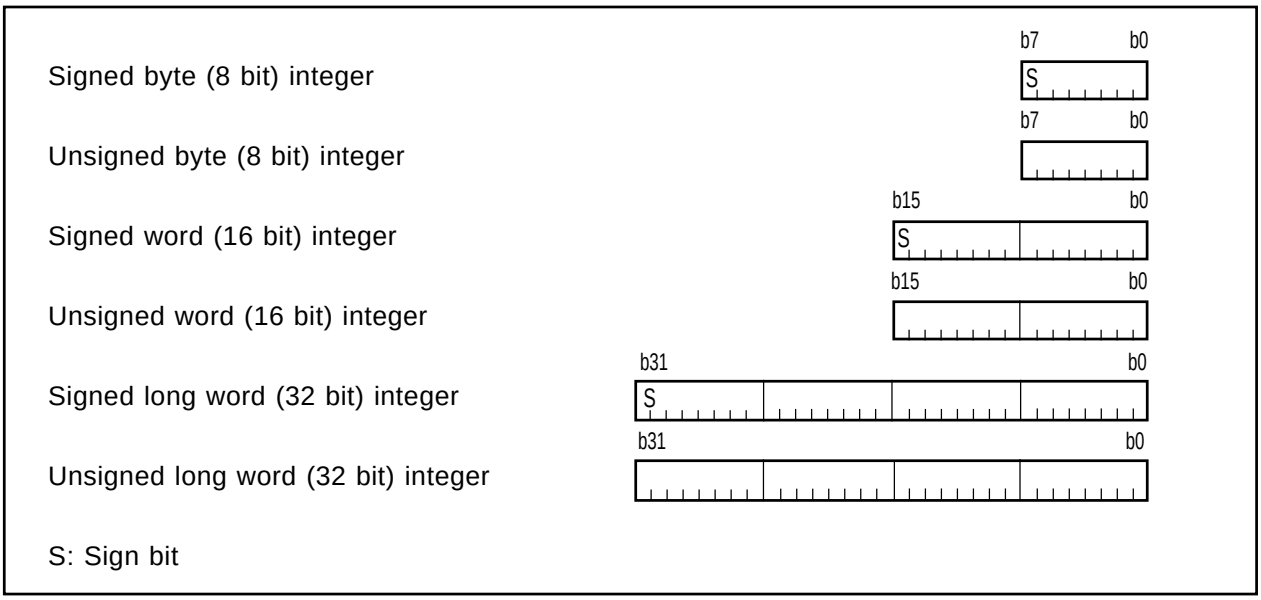


Figure 1.7.1 Integer data

1.7.2 Decimal

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

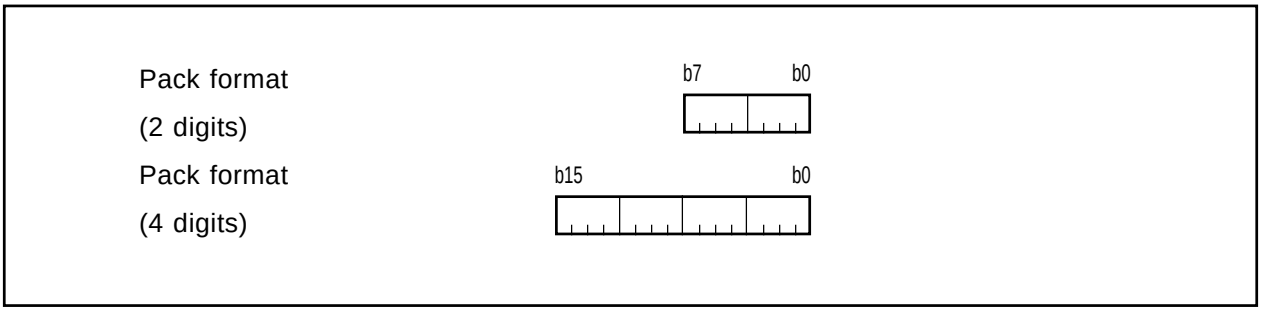


Figure 1.7.2 Decimal data

1.7.3 Bits

●Register bits

Figure 1.7.3 shows register bit specification.

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

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

Figure 1.7.3 Register bit specification

●Memory bits

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

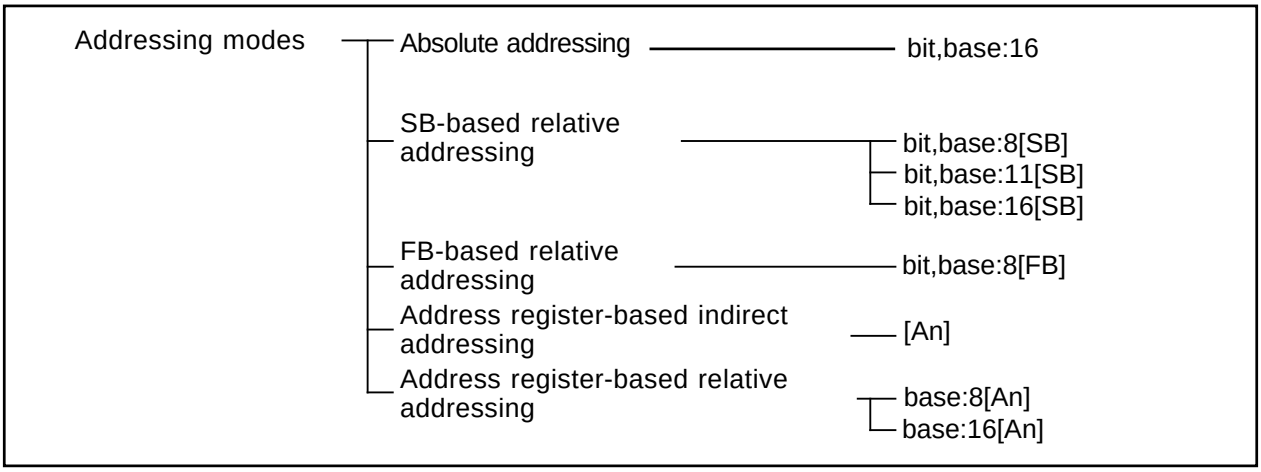


Figure 1.7.4 Addressing modes used for memory bit specification

Table 1.7.1 Bit-Specifying Address Range

Addressing	Specification range		Remarks
	Lower limit (address)	Upper limit (address)	
bit,base:16	00000_{16}	$01FFF_{16}$	
bit,base:8[SB]	[SB]	[SB]+$0001F_{16}$	The access range is 00000_{16} to $0FFFF_{16}$.
bit,base:11[SB]	[SB]	[SB]+$000FF_{16}$	The access range is 00000_{16} to $0FFFF_{16}$.
bit,base:16[SB]	[SB]	[SB]+$01FFF_{16}$	The access range is 00000_{16} to $0FFFF_{16}$.
bit,base:8[FB]	[FB]–00010_{16}	[FB]+$0000F_{16}$	The access range is 00000_{16} to $0FFFF_{16}$.
[An]	00000_{16}	$01FFF_{16}$	
base:8[An]	base:8	base:8+$01FFF_{16}$	The access range is 00000_{16} to $020FE_{16}$.
base:16[An]	base:16	base:16+$01FFF_{16}$	The access range is 00000_{16} to $0FFFF_{16}$.

(1) Bit specification by bit, base

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

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

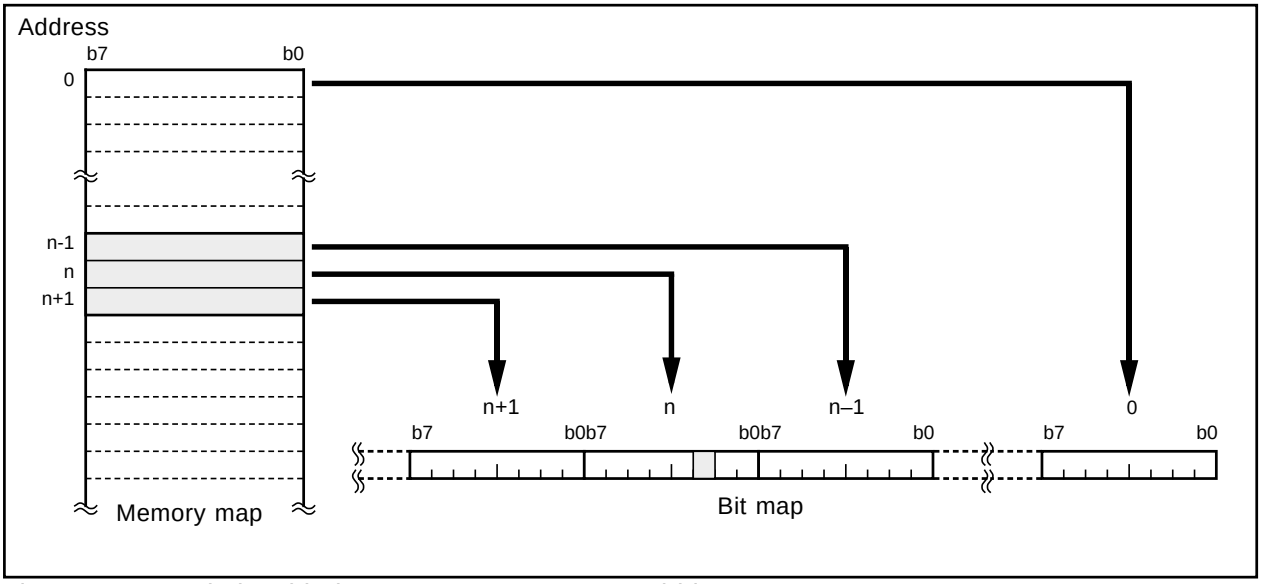


Figure 1.7.5 Relationship between memory map and bit map

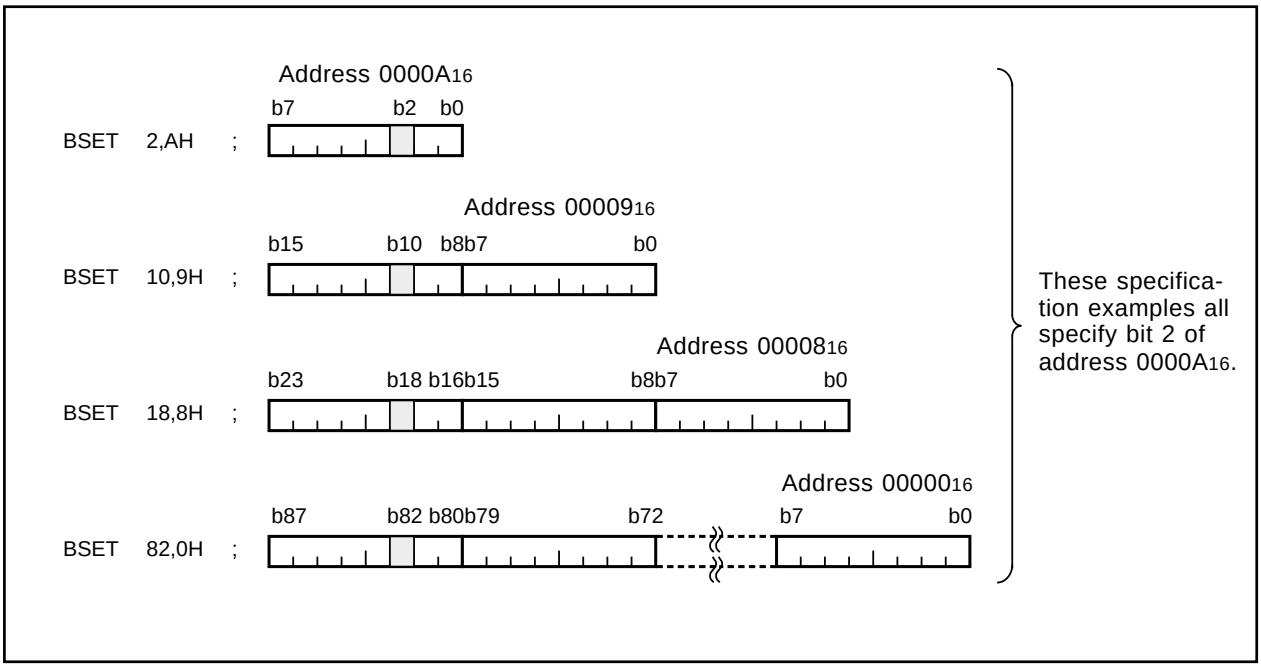


Figure 1.7.6 Examples of how to specify bit 2 of address 0000A16

(2) SB/FB relative bit specification

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

(3) Address register indirect/relative bit specification

For address register-based indirect addressing, use bit 0 of address 00000_{16} as the reference (= 0) and set your desired bit position to address register (**An**).
For address register-based relative addressing, use bit 0 of the address set to **base** as the reference (= 0) and set your desired bit position to address register (**An**).

1.7.4 String

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

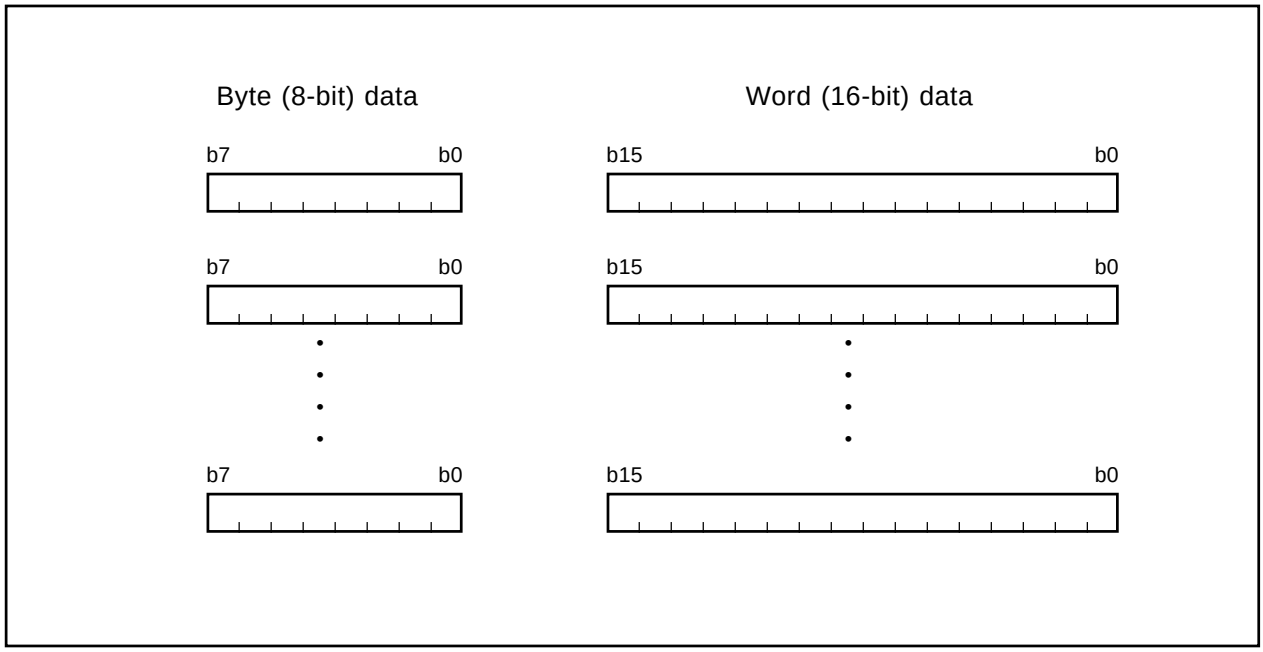


Figure 1.7.7 String data

1.8 Data Arrangement

1.8.1 Data Arrangement in Register

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

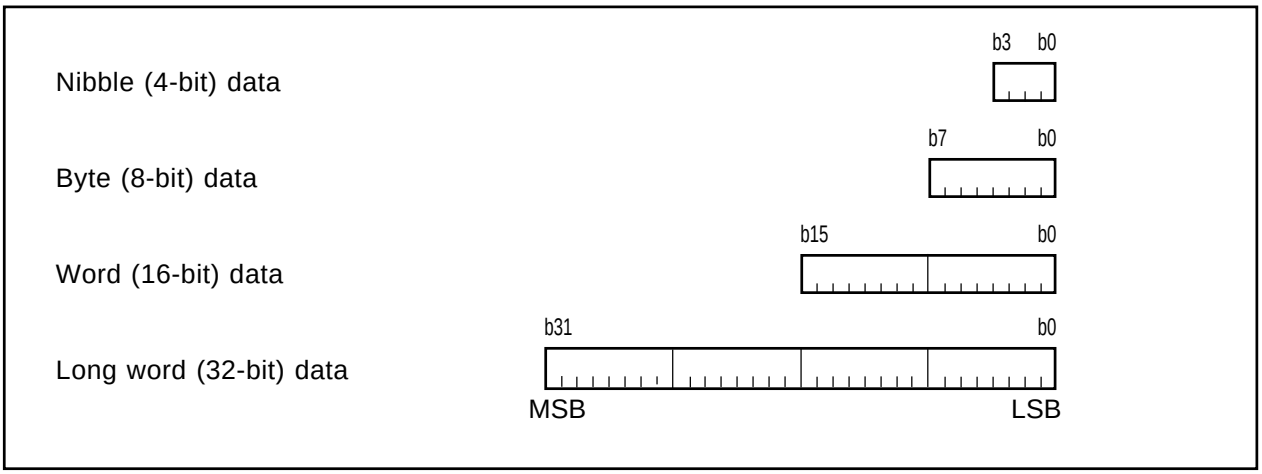


Figure 1.8.1 Data arrangement in register

1.8.2 Data Arrangement in Memory

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

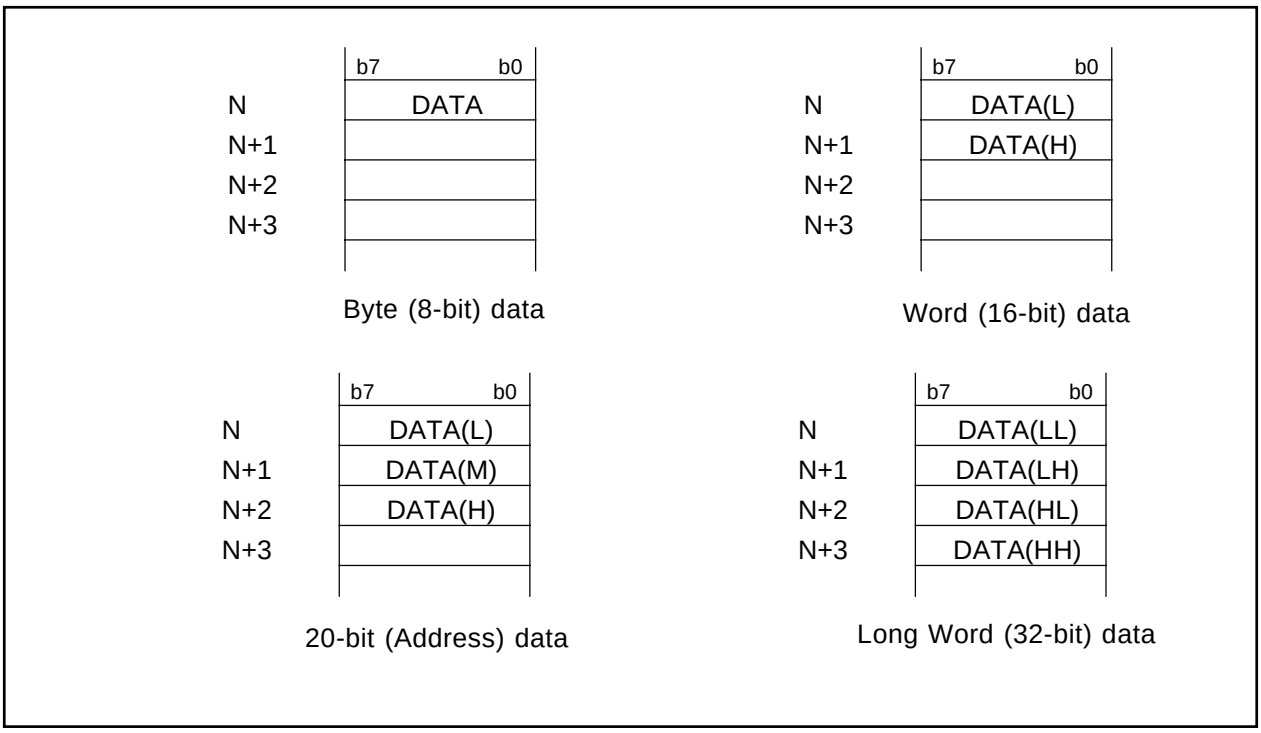


Figure 1.8.2 Data arrangement in memory

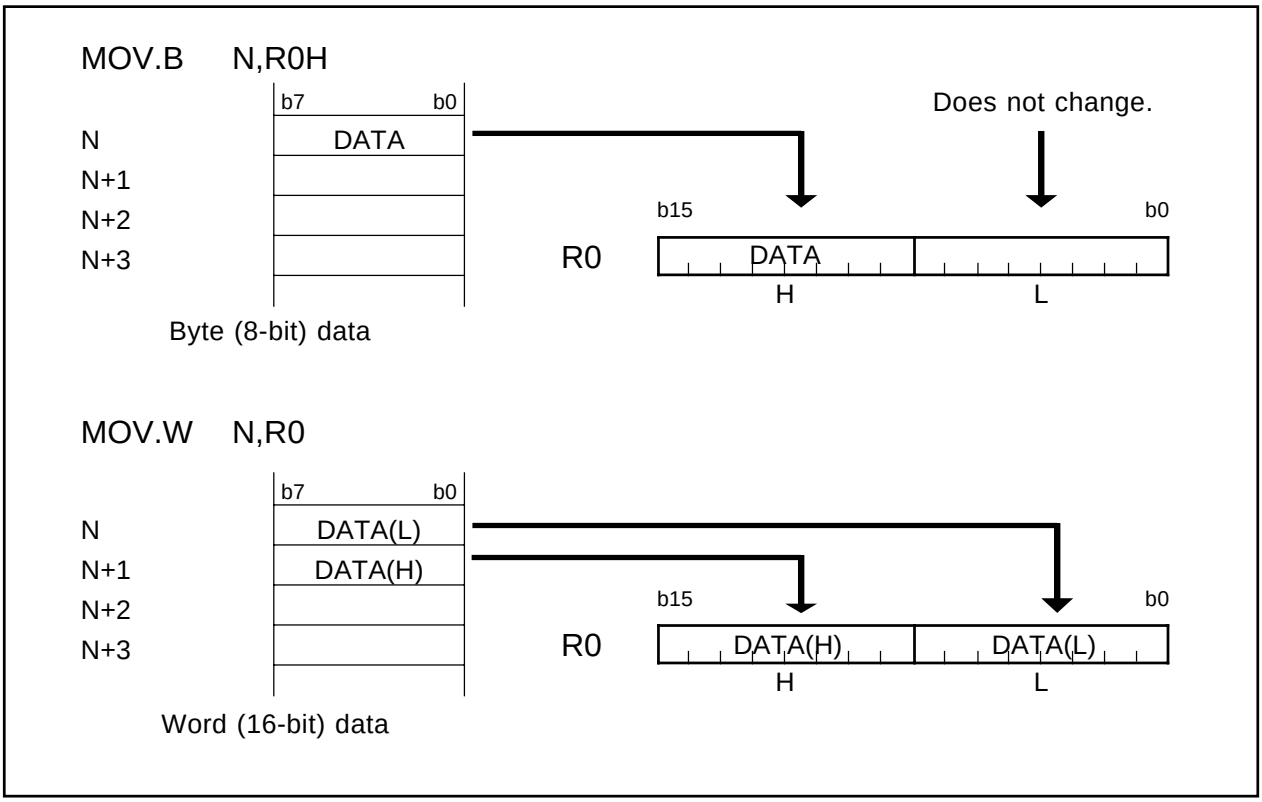


Figure 1.8.3 Examples of operation

1.9 Instruction Format

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

1.9.1 Generic format (:G)

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

1.9.2 Quick format (:Q)

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

1.9.3 Short format (:S)

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

1.9.4 Zero format (:Z)

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

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

1.10 Vector Table

There is an interrupt vector table as the vector table. In the interrupt vector table, there are the fixed vector table and the variable vector table.

1.10.1 Fixed Vector Table

The fixed vector table is an address-fixed vector table. The part of the interrupt vector table is allocated to addresses $0FFDC_{16}$ through $0FFFF_{16}$. Figure 1.10.1 shows a fixed vector table.

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

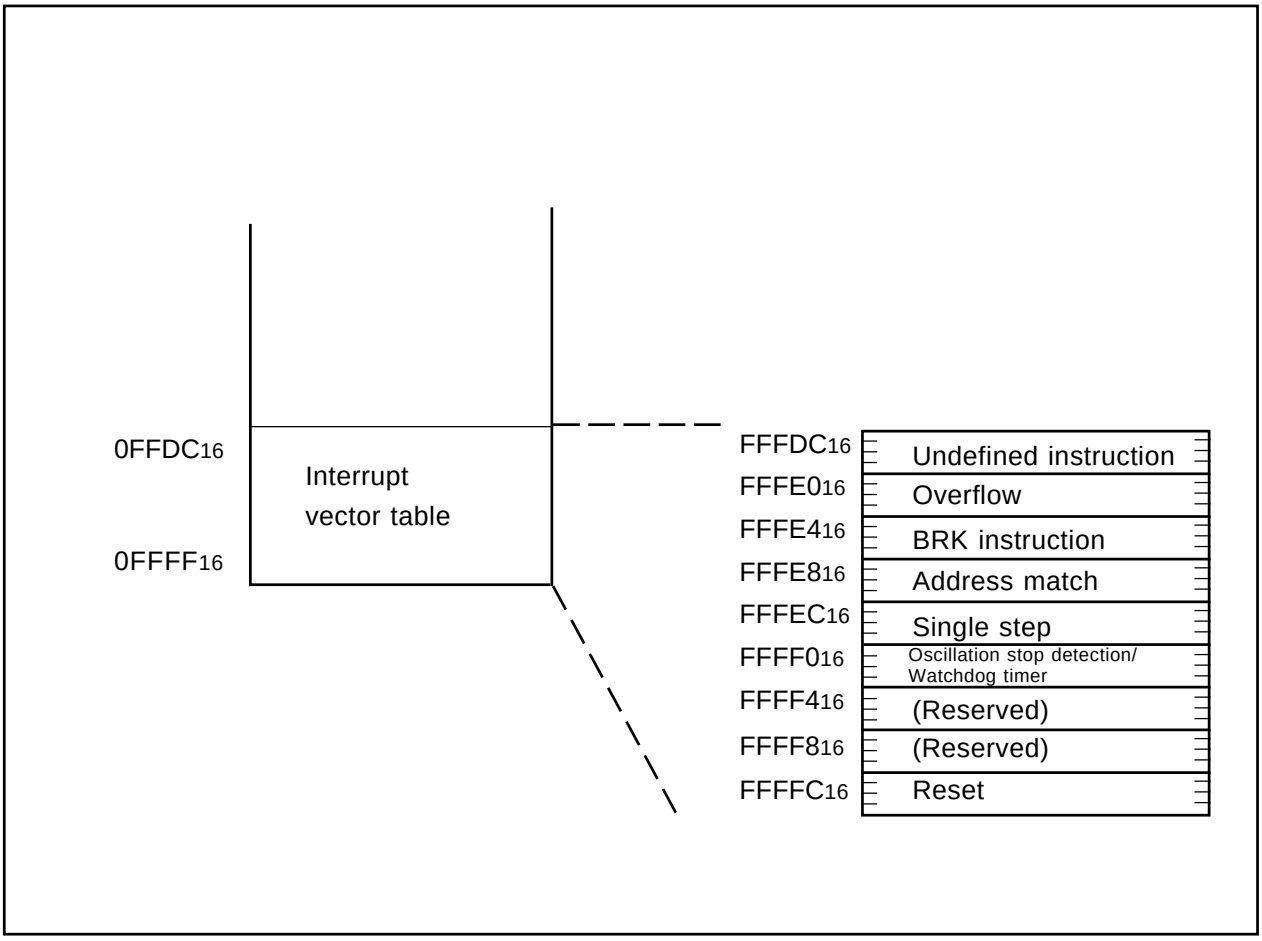


Figure 1.10.1 Fixed vector table

1.10.2 Variable Vector Table

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

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

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

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

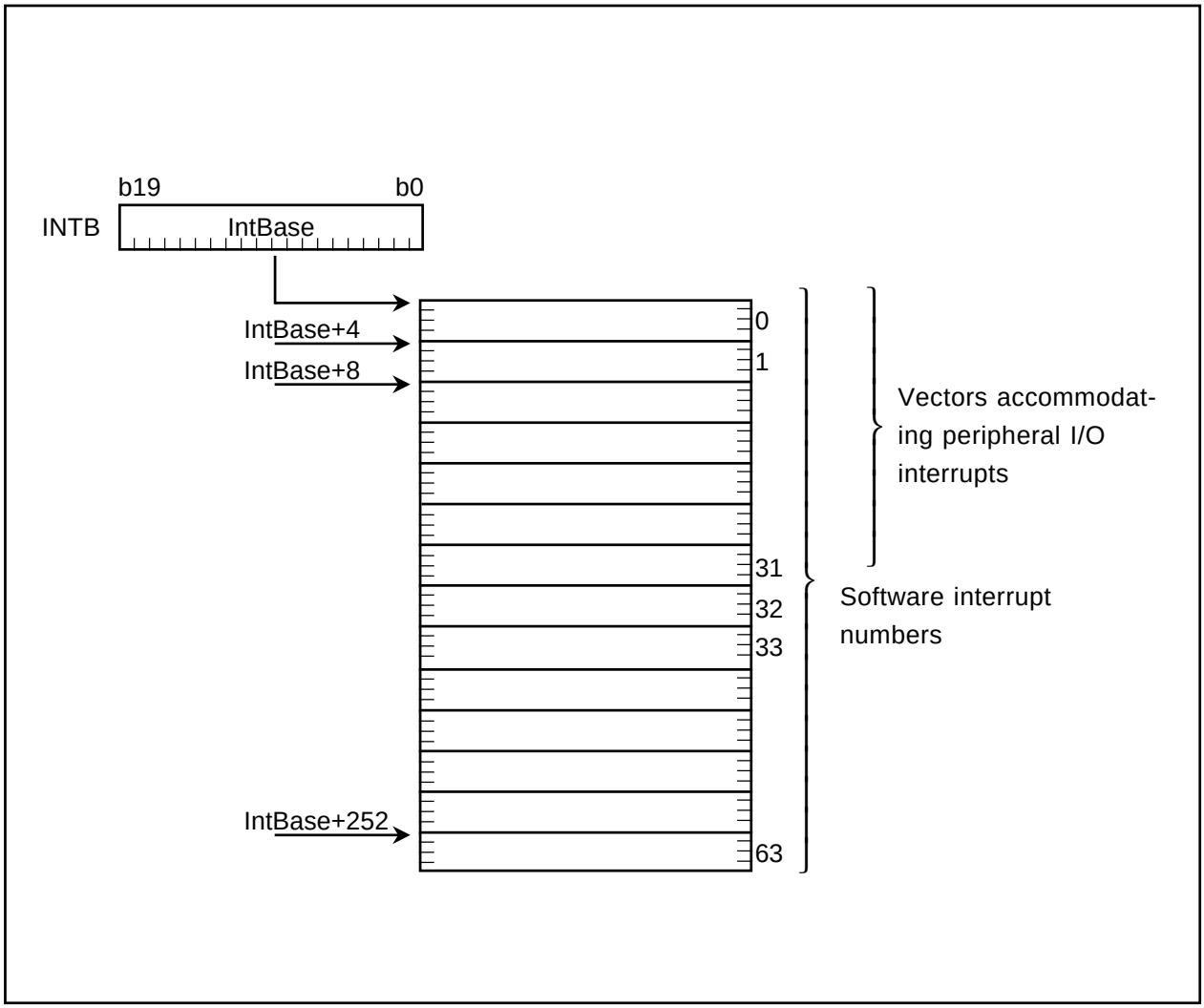


Figure 1.10.2 Variable vector table

Chapter 2

Addressing Modes

2.1 Addressing Modes

This section describes addressing mode-representing symbols and operations for each addressing mode. The R8C/Tiny series has three addressing modes outlined below.

2.1.1 General instruction addressing

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

The following lists the name of each general instruction addressing:

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

2.1.2 Special instruction addressing

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

The following lists the name of each specific instruction addressing:

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

2.1.3 Bit instruction addressing

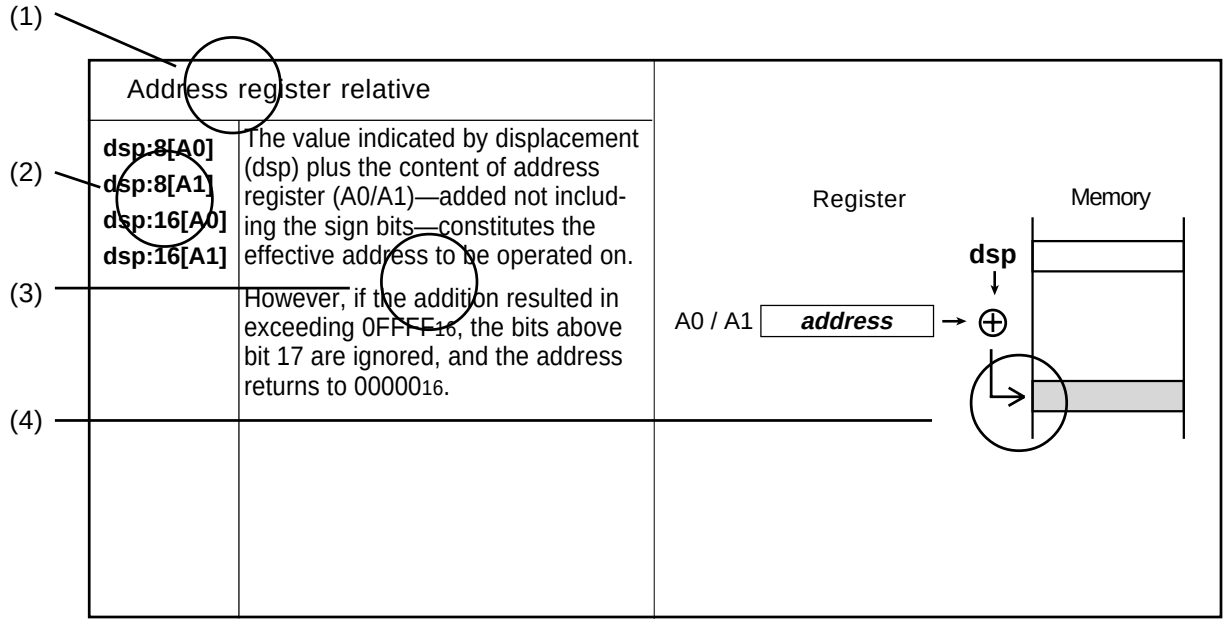
This addressing accesses an area from address 00000_{16} through address $0FFFF_{16}$.

The following lists the name of each bit instruction addressing:

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

2.2 Guide to This Chapter

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



(1) Name

Indicates the name of addressing.

(2) Symbol

Represents the addressing mode.

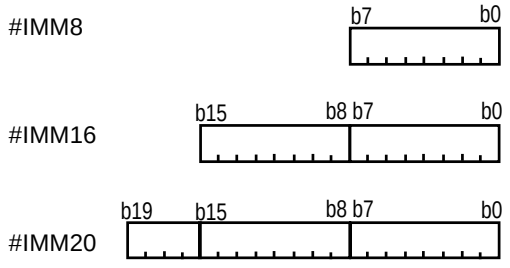
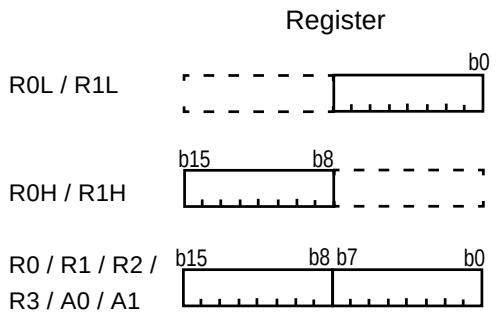
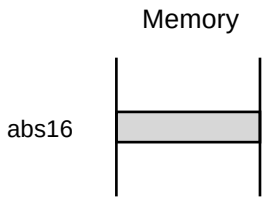
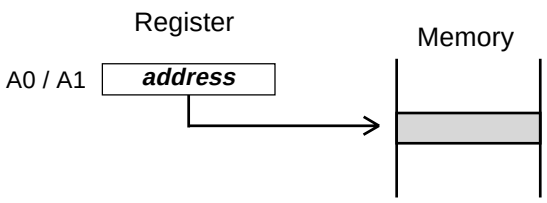
(3) Explanation

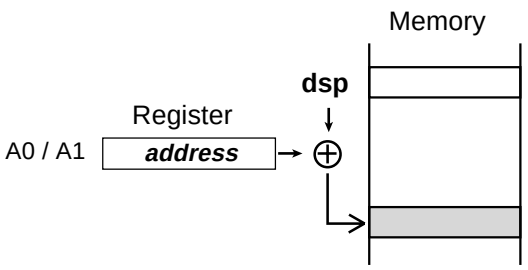
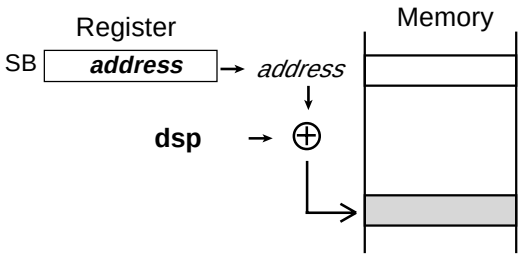
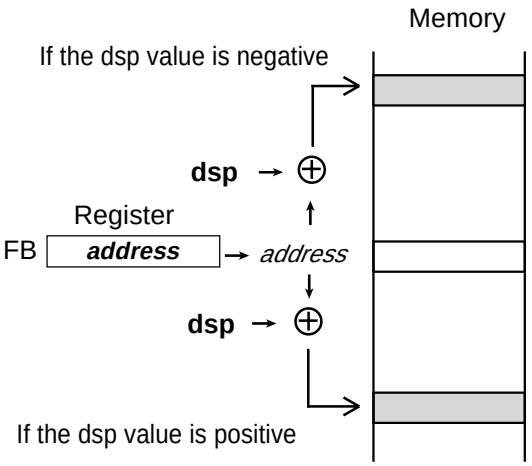
Describes the addressing operation and the effective address range.

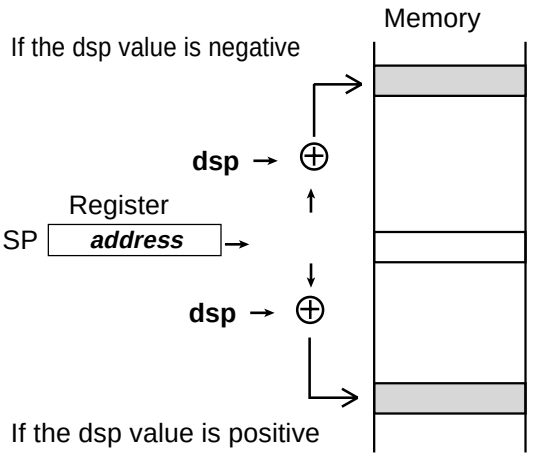
(4) Operation diagram

Diagrammatically explains the addressing operation.

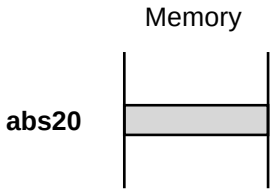
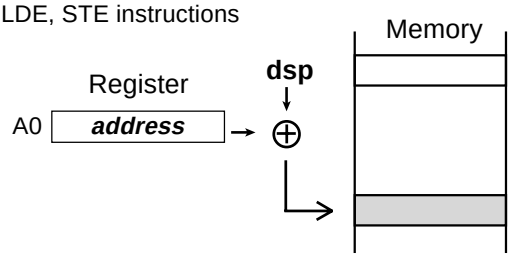
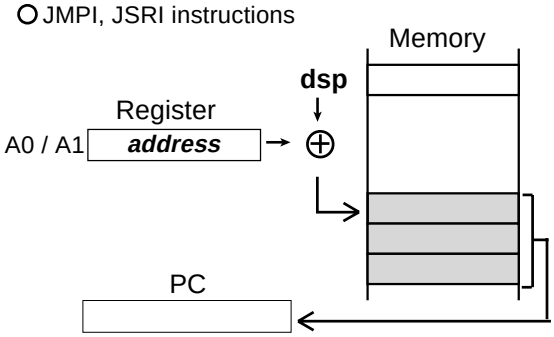
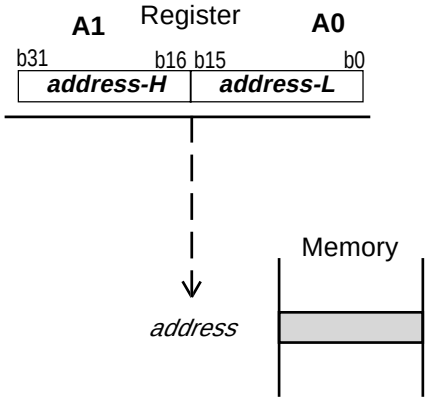
2.3 General Instruction Addressing

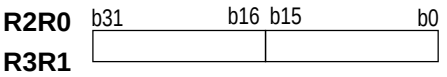
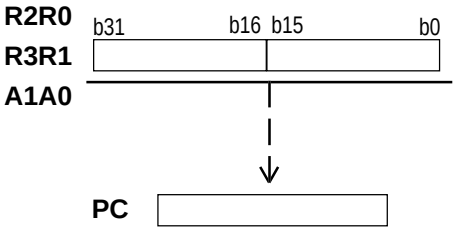
Immediate		
#IMM **#IMM8** **#IMM16** **#IMM20**	The immediate data indicated by #IMM is the object to be operated on.	
Register direct		
R0L **R0H** **R1L** **R1H** **R0** **R1** **R2** **R3** **A0** **A1**	The specified register is the object to be operated on.	
Absolute		
abs16	The value indicated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

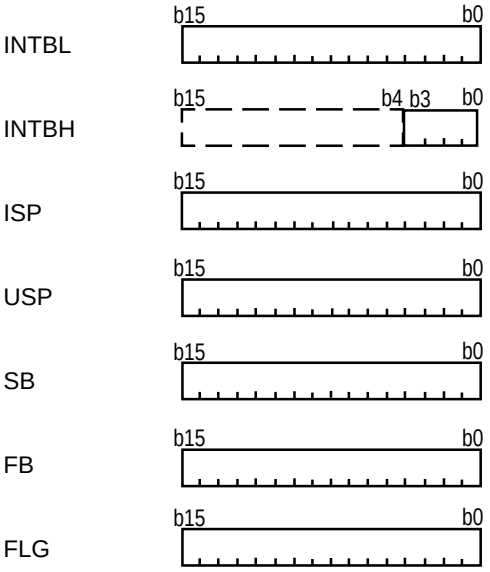
Address register relative		
dsp:8[A0] **dsp:8[A1]** **dsp:16[A0]** **dsp:16[A1]**	The value indicated by displacement (dsp) plus the content of address register (A0/A1)—added not including the sign bits—constitutes the effective address to be operated on. However, if the addition resulted in exceeding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

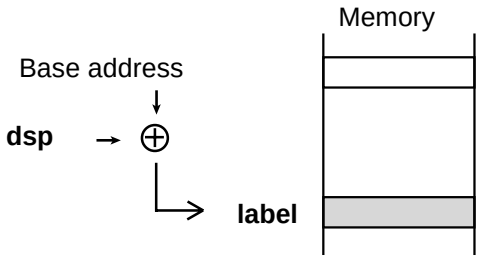
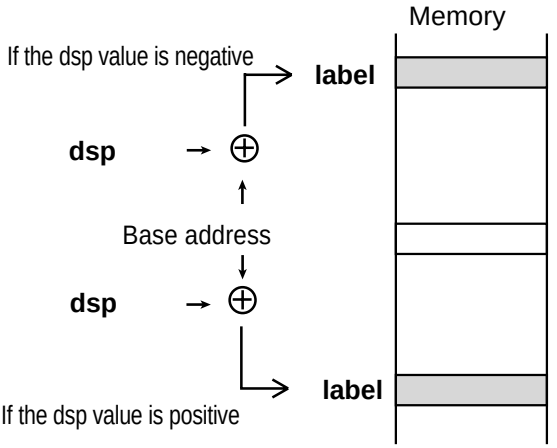
Stack pointer relative		
dsp:8[SP]	The address indicated by the content of stack pointer (SP) plus the value indicated by displacement (dsp)—added including the sign bits—constitutes the effective address to be operated on. The stack pointer (SP) here is the one indicated by the U flag. However, if the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

2.4 Special Instruction Addressing

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

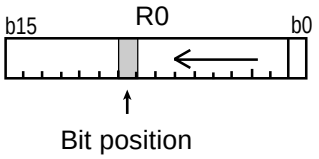
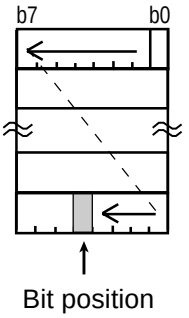
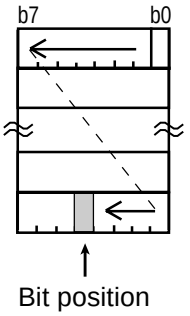
32-bit register direct		
R2R0 **R3R1** **A1A0**	The 32-bit concatenated register content of two specified registers is the object to be operated on. This addressing can be used in SHL, SHA, JMPI, and JSRI instructions. The following lists the register and instruction combinations that can be used. R2R0, R3R1 → SHL, SHA, JMPI, and JSRI instructions A1A0 → JMPI and JSRI instructions	○ SHL, SHA instructions  ○ JMPI, JSRI instructions 

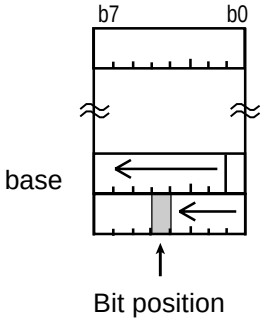
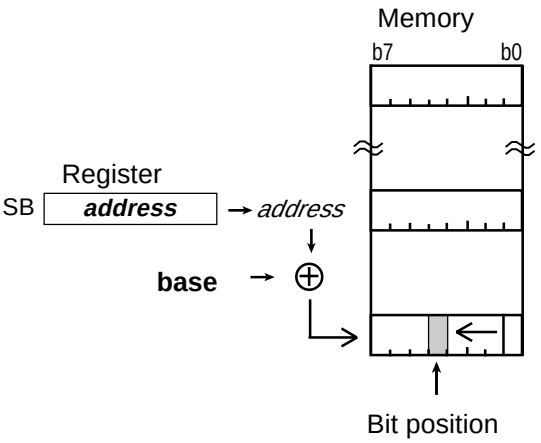
Control register direct		
INTBL **INTBH** **ISP** **SP** **SB** **FB** **FLG**	The specified control register is the object to be operated on. This addressing can be used in LDC, STC, PUSHC, and POPC instructions. If you specify SP, the stack pointer indicated by the U flag is the object to be operated on.	Register 

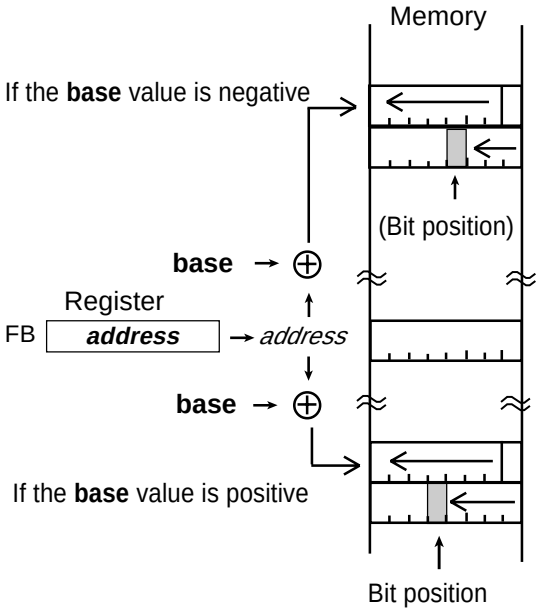
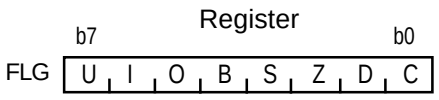
Program counter relative		
label	• If the jump length specifier (.length) is (.S)... the base address plus the value indicated by displacement (dsp)—added not including the sign bits—constitutes the effective address. This addressing can be used in JMP instruction.	 +0 ≦ dsp ≦ +7 *1 The base address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

2.5 Bit Instruction Addressing

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

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

Address register relative		
base:8[A0] **base:8[A1]** **base:16[A0]** **base:16[A1]**	The bit that is as much away from bit 0 at the address indicated by **base** as the number of bits indicated by address register (A0/A1) is the object to be operated on. However, if the address of the bit to be operated on exceeds $0FFFF_{16}$, the bits above bit 17 are ignored and the address returns to 00000_{16}. The address range that can be specified by address register (A0/A1) is 8,192 bytes from **base**.	
SB relative		
bit,base:8[SB] **bit,base:11[SB]** **bit,base:16[SB]**	The bit that is as much away from bit 0 at the address indicated by static base register (SB) plus the value indicated by **base** (added not including the sign bits) as the number of bits indicated by **bit** is the object to be operated on. However, if the address of the bit to be operated on exceeds $0FFFF_{16}$, the bits above bit 17 are ignored and the address returns to 00000_{16}. The address ranges that can be specified by bit,base: 8, bit,base: 11, and bit,base:16 respectively are 32 bytes, 256 bytes, and 8,192 bytes from the static base register (SB) value.	

FB relative		
bit,base:8[FB]	The bit that is as much away from bit 0 at the address indicated by frame base register (FB) plus the value indicated by **base** (added including the sign bit) as the number of bits indicated by **bit** is the object to be operated on. However, if the address of the bit to be operated on exceeds 00000_{16}-$0FFFF_{16}$, the bits above bit 17 are ignored and the address returns to 00000_{16} or $0FFFF_{16}$. The address range that can be specified by bit,base: 8 is 16 bytes toward lower addresses or 15 bytes toward higher addresses from the frame base register (FB) value.	
FLG direct		
U **I** **O** **B** **S** **Z** **D** **C**	The specified flag is the object to be operated on. This addressing can be used in FCLR and FSET instructions.	

Chapter 3

Functions

3.1 Guide to This Chapter

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

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

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

[Syntax]

(3) Page=193

MOV.size (:format) src,dest

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

B , W

(4) **[Operation]**

dest ← src

(5) **[Function]**

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

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

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

(7) **[Flag Change]**

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

Conditions

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

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

(8) **[Description Example]**

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

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

(1) Mnemonic

Indicates the mnemonic explained in this page.

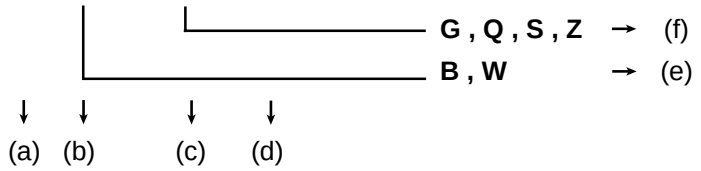
(2) Instruction code/Number of Cycles

Indicates the page in which instruction code/number of cycles is listed.
Refer to this page for instruction code and number of cycles.

(3) Syntax

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

MOV.size (: format) src , dest



(a) Mnemonic **MOV**
Describes the mnemonic.

(b) Size specifier **size**
Describes the data size in which data is handled. The following lists the data sizes that can be speci fied:

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

Some instructions do not have a size specifier.

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

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

Some instructions do not have an instruction format specifier.

(d) Operand **src, dest**
Describes the operand.

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

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

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

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

Page=193

[Syntax]

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

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

B , W

(4) **[Operation]**

dest ← src

(5) **[Function]**

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

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

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

(7) **[Flag Change]**

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

Conditions

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

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

(8) **[Description Example]**

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

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

(4) Operation

Explains the operation of the instruction using symbols.

(5) Function

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

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

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

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

(a) (b) (c) (d) (e)

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

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

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

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

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

(7) Flag change

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

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

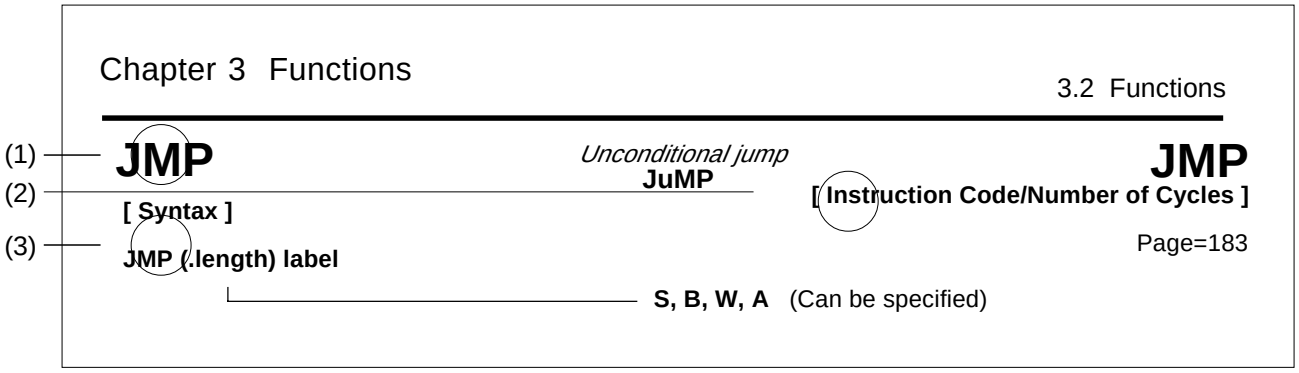
(8) Description example

Shows a description example for the instruction.

(9) Related instructions

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

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



(3) Syntax

Indicates the instruction syntax using a symbol.

JMP (.length) label

S, B, W, A → (d)

↓ (a) ↓ (b) ↓ (c)

(a) Mnemonic **JMP**

Describes the mnemonic.

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

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

The following lists the jump distances that can be specified:

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

(c) Operand **label**

Describes the operand.

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

ABS

Absolute value
ABSolute

ABS

[Syntax]

ABS.size dest

B , W

[Instruction Code/Number of Cycles]

Page=138

[Operation]

dest ← | dest |

[Function]

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

[Selectable dest]

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

[Flag Change]

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

Conditions

O : The flag is set (= 1) when dest before the operation is –128 (.B) or –32768 (.W); otherwise cleared (= 0).
S : The flag is set when the operation resulted in MSB = 1; otherwise cleared.
Z : The flag is set when the operation resulted in 0; otherwise cleared.
C : The flag is indeterminate.

[Description Example]

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

ADC

Add with carry
ADdition with Carry

ADC

[Syntax]

ADC.size src,dest

B , W

[Instruction Code/Number of Cycles]

Page=138

[Operation]

dest ← src + dest + C

[Function]

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

[Selectable src/dest]

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

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

[Flag Change]

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

Conditions

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

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

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

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

[Description Example]

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

[Related Instructions] ADCF,ADD,SBB,SUB

ADCF

Add carry flag
ADdition Carry Flag

ADCF

[Syntax]

ADCF.size dest

B , W

[Instruction Code/Number of Cycles]

Page=140

[Operation]

dest ← dest + C

[Function]

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

[Selectable dest]

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

[Flag Change]

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

Conditions

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

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

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

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

[Description Example]

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

[Related Instructions] ADC,ADD,SBB,SUB

ADD

Add without carry
ADDition

ADD

[Syntax]

ADD.size (:format) src,dest

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

[Instruction Code/Number of Cycles]

Page=140

[Operation]

dest ← dest + src

[Function]

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

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

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

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

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

[Flag Change]

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

Conditions

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

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

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

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

[Description Example]

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

[Related Instructions] ADC,ADCF,SBB,SUB

[src/dest Classified by Format]

G format

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

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

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

Q format

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

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

*3 The range of values that can be taken on is $-8 \leq$ #IMM $\leq +7$.

S format*4

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

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

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

ADJNZ

Add & conditional jump
ADdition then Jump on Not Zero

ADJNZ

[Syntax]

ADJNZ.size src,dest,label

B , W

[Instruction Code/Number of Cycles]

Page=146

[Operation]

dest ← dest + src

if dest ≠ 0 **then** jump label

[Function]

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

[Selectable src/dest/label]

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

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

*2 PC indicates the start address of the instruction.

[Flag Change]

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

[Description Example]

ADJNZ.W #–1,R0,label

[Related Instructions] SBJNZ

AND

Logically AND
AND

AND

[Syntax]

AND.size (:format) src,dest

G , S (Can be specified)
B , W

[Instruction Code/Number of Cycles]

Page=147

[Operation]

dest ← src ∧ dest

[Function]

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

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

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

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

[Flag Change]

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

Conditions

S : The flag is set when the operation resulted in MSB = 1; otherwise cleared.
Z : The flag is set when the operation resulted in 0; otherwise cleared.

[Description Example]

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

[Related Instructions] OR,XOR,TST

[src/dest Classified by Format]

G format

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

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

S format*2

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

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

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

BAND

Logically AND bits
Bit AND carry flag

BAND

[Syntax]

BAND src

[Instruction Code/Number of Cycles]

Page=150

[Operation]

C ← src ∧ C

[Function]

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

[Selectable src]

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

[Flag Change]

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

Conditions

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

[Description Example]

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

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

BCLR

Clear bit
Bit CLeaR

BCLR

[Syntax]

BCLR (:format) dest

G , S (Can be specified)

[Instruction Code/Number of Cycles]

Page=150

[Operation]

dest ← 0

[Function]

- This instruction stores 0 in *dest*.

[Selectable dest]

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

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

[Flag Change]

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

[Description Example]

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

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

BM*Cnd*

Conditional bit transfer
Bit Move Condition

BM*Cnd*

[Syntax]

BM*Cnd* **dest**

[Instruction Code/Number of Cycles]

Page=152

[Operation]

if true then dest ← 1
else dest ← 0

[Function]

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

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

[Selectable dest]

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

[Flag Change]

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

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

[Description Example]

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

[Related Instructions] J*Cnd*

BNAND

Logically AND inverted bits
Bit Not AND carry flag

BNAND

[Syntax]

BNAND src

[Instruction Code/Number of Cycles]

Page=153

[Operation]

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

[Function]

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

[Selectable src]

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

[Flag Change]

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

Condition

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

[Description Example]

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

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

BNOR

Logically OR inverted bits
Bit Not OR carry flag

BNOR

[Syntax]

BNOR src

[Instruction Code/Number of Cycles]

Page=154

[Operation]

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

[Function]

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

[Selectable src]

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

[Flag Change]

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

Condition

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

[Description Example]

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

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

BNOT

Invert bit
Bit NOT

BNOT

[Syntax]

BNOT(:format) dest

G , S (Can be specified)

[Instruction Code/Number of Cycles]

Page=154

[Operation]

dest ← $\overline{\text{dest}}$

[Function]

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

[Selectable dest]

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

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

[Flag Change]

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

[Description Example]

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

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

BNTST

Test inverted bit
Bit Not TeST

BNTST

[Syntax]

BNTST src

[Instruction Code/Number of Cycles]

Page=155

[Operation]

$Z \leftarrow \overline{src}$

$C \leftarrow \overline{src}$

[Function]

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

[Selectable src]

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

[Flag Change]

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

Conditions

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

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

[Description Example]

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

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

BNXOR

Exclusive OR inverted bits
Bit Not eXclusive OR carry flag

BNXOR

[Syntax]

BNXOR src

[Instruction Code/Number of Cycles]

Page=156

[Operation]

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

[Function]

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

[Selectable src]

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

[Flag Change]

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

Conditions

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

[Description Example]

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

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

BOR

Logically OR bits
Bit OR carry flag

BOR

[Syntax]

BOR src

[Instruction Code/Number of Cycles]

Page=156

[Operation]

C ← src ∨ C

[Function]

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

[Selectable src]

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

[Flag Change]

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

Conditions

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

[Description Example]

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

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

BRK

Debug interrupt
BReaK

BRK

[Syntax]

BRK

[Instruction Code/Number of Cycles]

Page=157

[Operation]

SP ← SP – 2
M(SP) ← (PC + 1)H, FLG
SP ← SP – 2
M(SP) ← (PC + 1)ML
PC ← M(FFFE4$_{16}$)

[Function]

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

[Flag Change][*1]

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

Conditions

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

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

[Description Example]

BRK

[Related Instructions] INT,INTO

BSET

Set bit
Bit SET

BSET

[Syntax]

BSET (:format) dest

G , S (Can be specified)

[Instruction Code/Number of Cycles]

Page=157

[Operation]

dest ← 1

[Function]

- This instruction stores 1 in *dest*.

[Selectable dest]

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

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

[Flag Change]

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

[Description Example]

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

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

BTST

Test bit
Bit TeST

BTST

[Syntax]

BTST (:format) src

G , S (Can be specified)

[Instruction Code/Number of Cycles]

Page=158

[Operation]

Z ← $\overline{\text{src}}$

C ← src

[Function]

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

[Selectable src]

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

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

[Flag Change]

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

Conditions

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

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

[Description Example]

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

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

BTSTC

Test bit & clear
Bit TeST & Clear

BTSTC

[Syntax]

BTSTC dest

[Instruction Code/Number of Cycles]

Page=159

[Operation]

Z ← $\overline{\text{dest}}$
C ← dest
dest ← 0

[Function]

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

[Selectable dest]

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

[Flag Change]

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

Conditions

Z : The flag is set when *dest* is 0; otherwise cleared.
C : The flag is set when *dest* is 1; otherwise cleared.

[Description Example]

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

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

BTSTS

Test bit & set
Bit TeST & Set

BTSTS

[Syntax]

BTSTS **dest**

[Instruction Code/Number of Cycles]

Page=160

[Operation]

Z ← $\overline{\text{dest}}$
C ← dest
dest ← 1

[Function]

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

[Selectable dest]

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

[Flag Change]

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

Conditions

Z : The flag is set when *dest* is 0; otherwise cleared.
C : The flag is set when *dest* is 1; otherwise cleared.

[Description Example]

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

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

BXOR

Exclusive OR bits
Bit eXclusive OR carry flag

BXOR

[Syntax]

BXOR src

[Instruction Code/Number of Cycles]

Page=160

[Operation]

C ← src ∀ C

[Function]

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

[Selectable src]

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

[Flag Change]

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

Conditions

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

[Description Example]

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

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

CMP

Compare
CoMPare

CMP

[Syntax]

CMP.size (:format) src,dest

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

[Instruction Code/Number of Cycles]

Page=161

[Operation]

dest – src

[Function]

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

[Selectable src/dest]

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

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

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

[Flag Change]

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

Conditions

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

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

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

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

[Description Example]

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

[src/dest Classified by Format]

G format

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

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

Q format

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

*2 The range of values that can be taken on is $-8 \leq$ #IMM $\leq +7$.

S format*3

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

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

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

DADC

Decimal add with carry
Decimal ADdition with Carry

DADC

[Syntax]

DADC.size src,dest

B , W

[Instruction Code/Number of Cycles]

Page=165

[Operation]

dest ← src + dest + C

[Function]

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

[Selectable src/dest]

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

[Flag Change]

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

Conditions

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

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

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

[Description Example]

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

[Related Instructions] DADD,DSUB,DSBB

DADD

Decimal add without carry
Decimal ADDition

DADD

[Syntax]

DADD.size src,dest

B , W

[Instruction Code/Number of Cycles]

Page=167

[Operation]

dest ← src + dest

[Function]

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

[Selectable src/dest]

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

[Flag Change]

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

Conditions

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

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

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

[Description Example]

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

[Related Instructions] DADC,DSUB,DSBB

DEC

Decrement
DECrement

DEC

[Syntax]

DEC.size dest

B , W

[Instruction Code/Number of Cycles]

Page=169

[Operation]

dest ← dest – 1

[Function]

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

[Selectable dest]

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

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

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

[Flag Change]

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

Conditions

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

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

[Description Example]

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

[Related Instructions] INC

DIV

Signed divide
DIVide

DIV

[Syntax]

DIV.size src

B , W

[Instruction Code/Number of Cycles]

Page=170

[Operation]

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

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

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

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

[Function]

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

[Selectable src]

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

[Flag Change]

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

Conditions

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

[Description Example]

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

[Related Instructions] DIVU,DIVX,MUL,MULU

DIVU

Unsigned divide
DIVide Unsigned

DIVU

[Syntax]

DIVU.size src

└ **B , W**

[Instruction Code/Number of Cycles]

Page=171

[Operation]

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

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

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

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

[Function]

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

[Selectable src]

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

[Flag Change]

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

Conditions

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

[Description Example]

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

[Related Instructions] DIV,DIVX,MUL,MULU

DIVX

Singed divide
DIVide eXtension

DIVX

[Syntax]

DIVX.size src

└ **B , W**

[Instruction Code/Number of Cycles]

Page=172

[Operation]

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

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

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

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

[Function]

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

[Selectable src]

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

[Flag Change]

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

Conditions

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

[Description Example]

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

[Related Instructions] DIV,DIVU,MUL,MULU

DSBB

Decimal subtract with borrow

Decimal SuBtract with Borrow

DSBB

[Syntax]

DSBB.size src,dest

B , W

[Instruction Code/Number of Cycles]

Page=173

[Operation]

dest ← dest – src – $\overline{C}$

[Function]

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

[Selectable src/dest]

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

[Flag Change]

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

Conditions

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

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

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

[Description Example]

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

[Related Instructions] DADC,DADD,DSUB

DSUB

Decimal subtract without borrow
Decimal SUBtract

DSUB

[Syntax]

DSUB.size src,dest

B , W

[Instruction Code/Number of Cycles]

Page=175

[Operation]

dest ← dest – src

[Function]

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

[Selectable src/dest]

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

[Flag Change]

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

Conditions

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

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

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

[Description Example]

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

[Related Instructions] DADC,DADD,DSBB

ENTER

Build stack frame
ENTER function

ENTER

[Syntax]

ENTER src

[Instruction Code/Number of Cycles]

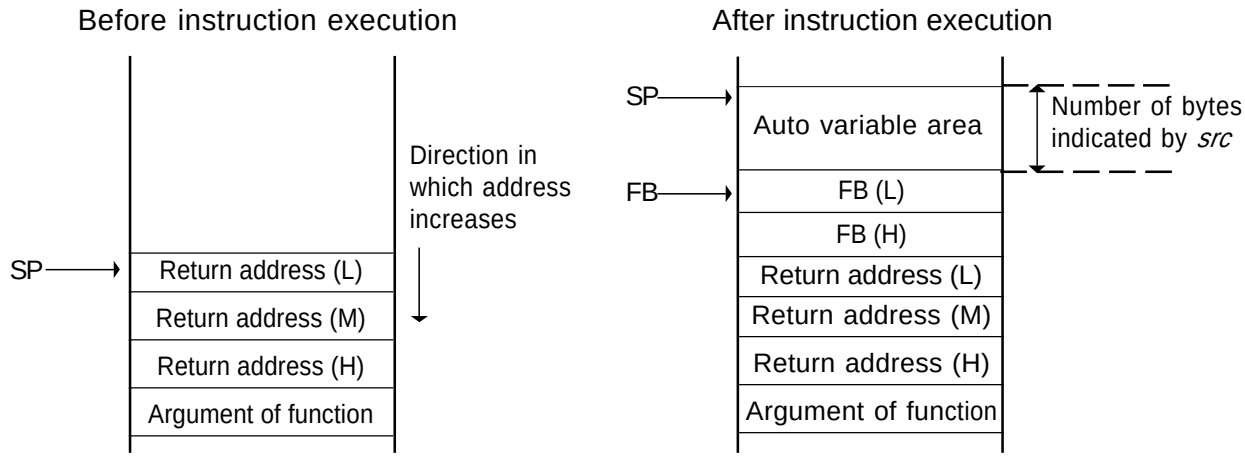
Page=177

[Operation]

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

[Function]

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



[Selectable src]

src
#IMM8

[Flag Change]

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

[Description Example]

```
ENTER    #3
```

[Related Instructions] EXITD

EXITD

Deallocate stack frame

EXIT and Deallocate stack frame

EXITD

[Syntax]

EXITD

[Instruction Code/Number of Cycles]

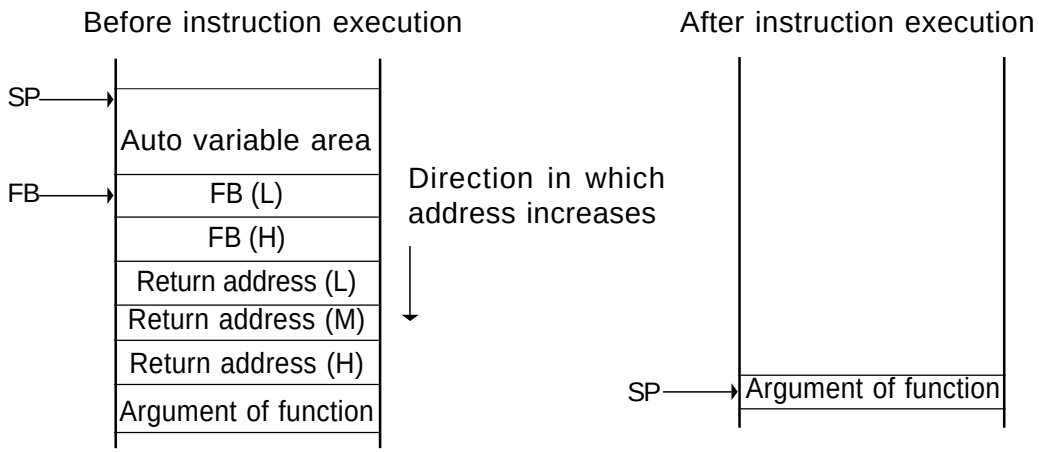
Page=178

[Operation]

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

[Function]

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



[Flag Change]

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

[Description Example]

EXITD

[Related Instructions] ENTER

EXTS

Extend sign
EXTend Sign

EXTS

[Syntax]

EXTS.size dest

B , W

[Instruction Code/Number of Cycles]

Page=178

[Operation]

dest ← EXT(dest)

[Function]

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

[Selectable dest]

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

[Flag Change]

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

Conditions

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

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

[Description Example]

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

FCLR

Clear flag register bit

Flag register CLeaR

FCLR

[Syntax]

FCLR dest

[Instruction Code/Number of Cycles]

Page=179

[Operation]

dest ← 0

[Function]

- This instruction stores 0 in *dest*.

[Selectable dest]

dest							
C	D	Z	S	B	O	I	U

[Flag Change]

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

*1 The selected flag is cleared to 0.

[Description Example]

```
FCLR    I
FCLR    S
```

[Related Instructions] FSET

FSET

Set flag register bit
Flag register SET

FSET

[Syntax]

FSET dest

[Instruction Code/Number of Cycles]

Page=180

[Operation]

dest ← 1

[Function]

- This instruction stores 1 in *dest*.

[Selectable dest]

dest							
C	D	Z	S	B	O	I	U

[Flag Change]

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

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

[Description Example]

```
FSET    I
FSET    S
```

[Related Instructions] FCLR

INC

Increment
INCrement

INC

[Syntax]

INC.size dest

B , W

[Instruction Code/Number of Cycles]

Page=180

[Operation]

dest ← dest + 1

[Function]

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

[Selectable dest]

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

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

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

[Flag Change]

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

Conditions

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

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

[Description Example]

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

[Related Instructions] DEC

INT

Interrupt by INT instruction
INTerrupt

INT

[Syntax]

INT **src**

[Instruction Code/Number of Cycles]

Page=181

[Operation]

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

[Function]

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

[Selectable src]

src
#IMM*1*2

*1 #IMM denotes a software interrupt number.

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

[Flag Change]

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

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

Conditions

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

I : The flag is cleared.

D : The flag is cleared.

[Description Example]

```
INT     #0
```

[Related Instructions] BRK,INTO

INTO

Interrupt on overflow

INTerrupt on Overflow

INTO

[Syntax]

INTO

[Instruction Code/Number of Cycles]

Page=182

[Operation]

SP ← SP – 2
M(SP) ← (PC + 1)H, FLG
SP ← SP – 2
M(SP) ← (PC + 1)ML
PC ← M(FFFE016)

[Function]

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

[Flag Change]

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

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

Conditions

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

[Description Example]

```
INTO
```

[Related Instructions] BRK,INT

J*Cnd*

Jump on condition
Jump on Condition

J*Cnd*

[Syntax]

J*Cnd* label

[Instruction Code/Number of Cycles]

Page=182

[Operation]

if true then jump label

[Function]

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

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

[Selectable label]

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

*1 PC indicates the start address of the instruction.

[Flag Change]

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

[Description Example]

```
JEQ     label
JNE     label
```

[Related Instructions] BM*Cnd*

JMP

Unconditional jump
JuMP

JMP

[Syntax]

JMP(.length) label

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

[Instruction Code/Number of Cycles]

Page=184

[Operation]

PC ← label

[Function]

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

[Selectable label]

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

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

[Flag Change]

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

[Description Example]

```
JMP     label
```

[Related Instructions] JMPI

JMPI

Jump indirect
JuMP Indirect

JMPI

[Syntax]

JMPI.length **src**

W , A

[Instruction Code/Number of Cycles]

Page=185

[Operation]

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

PC ← PC ± src

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

PC ← src

[Function]

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

[Selectable src]

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

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

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

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

[Flag Change]

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

[Description Example]

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

[Related Instructions] JMP

JSR

Subroutine call
Jump SubRoutine

JSR

[Syntax]

JSR(.length) label

W , A (Can be specified)

[Instruction Code/Number of Cycles]

Page=187

[Operation]

SP ← SP – 1
M(SP) ← (PC + n)H
SP ← SP – 2
M(SP) ← (PC + n)ML
PC ← label
*1 n denotes the number of instruction bytes.

[Function]

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

[Selectable label]

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

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

[Flag Change]

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

[Description Example]

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

[Related Instructions] JSRI

JSRI

Indirect subroutine call
Jump SubRoutine Indirect

JSRI

[Syntax]

JSRI.length src

W , A

[Instruction Code/Number of Cycles]

Page=188

[Operation]

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

SP ← SP – 1
M(SP) ← (PC + n)H
SP ← SP – 2
M(SP) ← (PC + n)ML
PC ← PC ± src
*1 n denotes the number of instruction bytes.

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

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

[Function]

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

[Selectable src]

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

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

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

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

[Flag Change]

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

[Description Example]

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

[Related Instructions] JSR

LDC

Transfer to control register
LoaD Control register

LDC

[Syntax]

LDC src,dest

[Instruction Code/Number of Cycles]

Page=189

[Operation]

dest ← src

[Function]

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

[Selectable src/dest]

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

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

[Flag Change]

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

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

[Description Example]

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

[Related Instructions] POPC,PUSHC,STC,LDINTB

LDCTX

Restore context
LoaD ConTeXt

LDCTX

[Syntax]

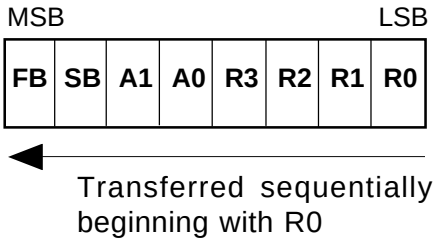
LDCTX abs16,abs20

[Instruction Code/Number of Cycles]

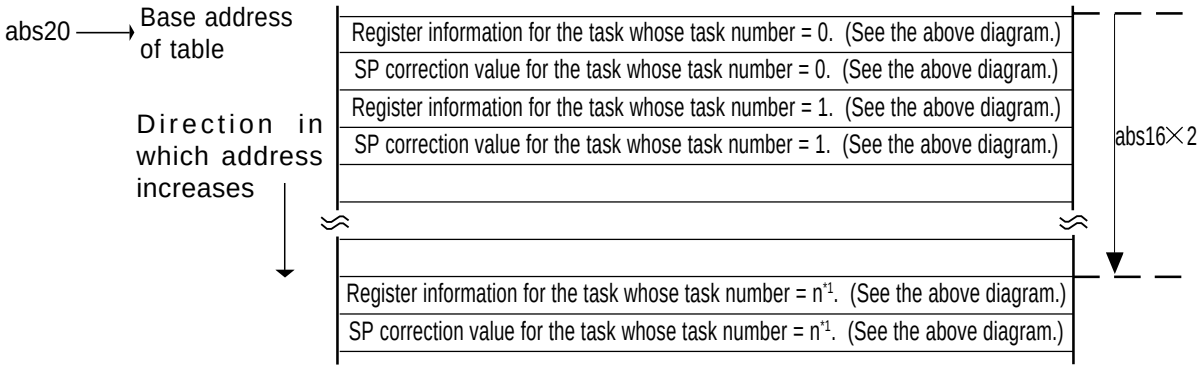
Page=189

[Function]

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



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



*1 n=0 to 255

[Flag Change]

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

[Description Example]

```
LDCTX    Ram,Rom_TBL
```

[Related Instructions] STCTX

LDE

Transfer from extended data area

LoaD from EXtra far data area

LDE

[Syntax]

LDE.size src,dest

B , W

[Instruction Code/Number of Cycles]

Page=191

[Operation]

dest ← src

[Function]

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

[Selectable src/dest]

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

[Flag Change]

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

Conditions

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

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

[Description Example]

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

[Related Instructions] STE,MOV,XCHG

LDINTB

Transfer to INTB register

LoaD INTB register

LDINTB

[Syntax]

LDINTB src

[Instruction Code/Number of Cycles]

Page=192

[Operation]

INTBHL ← src

[Function]

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

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

[Selectable src]

src
#IMM20

[Flag Change]

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

[Description Example]

```
LDINTB    #0F0000H
```

[Related Instructions] LDC,STC,PUSHC,POPC

LDIPL

Set interrupt enable level

LoaD Interrupt Permission Level

LDIPL

[Syntax]

LDIPL src

[Instruction Code/Number of Cycles]

Page=193

[Operation]

IPL ← src

[Function]

- This instruction transfers *src* to IPL.

[Selectable src]

src
#IMM*1

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

[Flag Change]

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

[Description Example]

```
LDIPL    #2
```

MOV

Transfer
MOVe

MOV

[Syntax]

MOV.size (:format) src,dest

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

B , W

[Instruction Code/Number of Cycles]

Page=193

[Operation]

dest ← src

[Function]

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

[Selectable src/dest]

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

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

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

[Flag Change]

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

Conditions

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

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

[Description Example]

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

[Related Instructions] LDE,STE,XCHG

[src/dest Classified by Format]

G format

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

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

Q format

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

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

S format

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

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

Z format

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

MOVA

Transfer effective address
MOVe effective Address

MOVA

[Syntax]

MOVA src,dest

[Instruction Code/Number of Cycles]

Page=200

[Operation]

dest ← EVA(src)

[Function]

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

[Selectable src/dest]

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

[Flag Change]

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

[Description Example]

```
MOVA    Ram:16[SB],A0
```

[Related Instructions] PUSHA

MOV*Dir*

Transfer 4-bit data
MOVe nibble

MOV*Dir*

[Syntax]

MOV*Dir* **src,dest**

[Instruction Code/Number of Cycles]

Page=201

[Operation]

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

[Function]

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

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

[Selectable src/dest]

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

[Flag Change]

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

[Description Example]

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

MUL

Signed multiply
MULtiple

MUL

[Syntax]

MUL.size src,dest

B , W

[Instruction Code/Number of Cycles]

Page=203

[Operation]

dest ← dest × src

[Function]

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

[Selectable src/dest]

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

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

[Flag Change]

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

[Description Example]

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

[Related Instructions] DIV,DIVU,DIVX,MULU

MULU

Unsigned multiply
MULtiple Unsigned

MULU

[Syntax]

MULU.size src,dest

└──── **B , W**

[Instruction Code/Number of Cycles]

Page=205

[Operation]

dest ← dest × src

[Function]

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

[Selectable src/dest]

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

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

[Flag Change]

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

[Description Example]

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

[Related Instructions] DIV,DIVU,DIVX,MUL

NEG

Two's complement
NEGate

NEG

[Syntax]

NEG.size dest

B , W

[Instruction Code/Number of Cycles]

Page=207

[Operation]

dest ← 0 – dest

[Function]

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

[Selectable dest]

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

[Flag Change]

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

Conditions

O : The flag is set when *dest* before the operation is –128 (.B) or –32768 (.W); otherwise cleared.
S : The flag is set when the operation resulted in MSB = 1; otherwise cleared.
Z : The flag is set when the operation resulted in 0; otherwise cleared.
C : The flag is set when the operation resulted in 0; otherwise cleared.

[Description Example]

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

[Related Instructions] NOT

NOP

No operation
No OPeration

NOP

[Syntax]

NOP

[Instruction Code/Number of Cycles]

Page=207

[Operation]

PC ← PC + 1

[Function]

- This instruction adds 1 to PC.

[Flag Change]

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

[Description Example]

NOP

NOT

Invert all bits
NOT

NOT

[Syntax]

NOT.size (:format) dest

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

[Instruction Code/Number of Cycles]

Page=208

[Operation]

dest ← $\overline{\text{dest}}$

[Function]

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

[Selectable dest]

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

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

[Flag Change]

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

Conditions

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

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

[Description Example]

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

[Related Instructions] NEG

OR

Logically OR
OR

OR

[Syntax]

OR.size (:format) src,dest

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

[Instruction Code/Number of Cycles]

Page=209

[Operation]

dest ← src ∨ dest

[Function]

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

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

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

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

[Flag Change]

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

Conditions

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

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

[Description Example]

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

[Related Instructions] AND,XOR,TST

[src/dest Classified by Format]

G format

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

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

S format*2

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

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

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

POP

Restore register/memory
POP

POP

[Syntax]

POP.size (:format) dest

G , S (Can be specified)
B , W

[Instruction Code/Number of Cycles]

Page=211

[Operation]

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

dest ← M(SP)
SP ← SP + 1

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

dest ← M(SP)
SP ← SP + 2

[Function]

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

[Selectable dest]

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

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

[Flag Change]

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

[Description Example]

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

[Related Instructions] PUSH,POPM,PUSHM

POPC

Restore control register
POP Control register

POPC

[Syntax]

POPC **dest**

[Instruction Code/Number of Cycles]

Page=213

[Operation]

dest ← M(SP)
SP*1 ← SP + 2

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

[Function]

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

[Selectable dest]

dest						
FB	SB	SP*2	ISP	FLG	INTBH	INTBL

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

[Flag Change]

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

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

[Description Example]

POPC SB

[Related Instructions] PUSHC,LDC,STC,LDINTB

POPM

Restore multiple registers
POP Multiple

POPM

[Syntax]

POPM **dest**

[Instruction Code/Number of Cycles]

Page=213

[Operation]

dest ← M(SP)

SP ← SP + N*1 × 2

*1 Number of registers to be restored

[Function]

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

FB	SB	A1	A0	R3	R2	R1	R0

Restored sequentially beginning with R0

[Selectable dest]

dest*2
R0 R1 R2 R3 A0 A1 SB FB

*2 You can choose multiple *dest*.

[Flag Change]

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

[Description Example]

```
POPM    R0,R1,A0,SB,FB
```

[Related Instructions] POP,PUSH,PUSHM

PUSH

Save register/memory/immediate data

PUSH

PUSH

[Syntax]

PUSH.size (:format) src

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

[Instruction Code/Number of Cycles]

Page=214

[Operation]

If the size specifier (.size) is (.B)	If the size specifier (.size) is (.W)
SP ← SP – 1	SP ← SP – 2
M(SP) ← src	M(SP) ← src

[Function]

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

[Selectable src]

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

*1 Can be selected in G and S formats.

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

[Flag Change]

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

[Description Example]

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

[Related Instructions] POP,POPM,PUSHM

PUSHA

Save effective address
PUSH effective Address

PUSHA

[Syntax]

PUSHA src

[Instruction Code/Number of Cycles]

Page=216

[Operation]

SP ← SP – 2
M(SP) ← EVA(src)

[Function]

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

[Selectable src]

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

[Flag Change]

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

[Description Example]

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

[Related Instructions] MOVA

PUSHC

Save control register
PUSH Control register

PUSHC

[Syntax]

PUSHC src

[Instruction Code/Number of Cycles]

Page=216

[Operation]

SP ← SP – 2

M(SP) ← src*1

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

[Function]

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

[Selectable src]

src
FB SB SP*2 ISP FLG INTBH INTBL

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

[Flag Change]

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

[Description Example]

```
PUSHC    SB
```

[Related Instructions] POPC,LDC,STC,LDINTB

PUSHM

Save multiple registers
PUSH Multiple

PUSHM

[Syntax]

PUSHM src

[Instruction Code/Number of Cycles]

Page=217

[Operation]

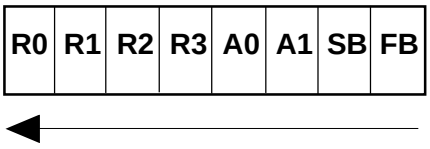
SP ← SP – N*1 × 2

M(SP) ← src

*1 Number of registers saved.

[Function]

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



Saved sequentially beginning with FB

[Selectable src]

src*2
R0 R1 R2 R3 A0 A1 SB FB

*2 You can choose multiple *src*.

[Flag Change]

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

[Description Example]

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

[Related Instructions] POP,PUSH,POPM

REIT

Return from interrupt

REturn from InTerrupt

REIT

[Syntax]

REIT

[Instruction Code/Number of Cycles]

Page=218

[Operation]

PC_{ML}	←	M(SP)
SP	←	SP + 2
PC_{H}, FLG	←	M(SP)
SP	←	SP + 2

[Function]

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

[Flag Change]

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

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

[Description Example]

```
REIT
```

RMPA

Calculate sum-of-products
Repeat MultiPle & Addition

RMPA

[Syntax]

RMPA.size

B , W

[Instruction Code/Number of Cycles]

Page=218

[Operation][*1]

Repeat

R2R0(R0)[*2] ← R2R0(R0)[*2] + M(A0) × M(A1)
A0 ← A0 + 2 (1)[*2]
A1 ← A1 + 2 (1)[*2]
R3 ← R3 – 1

Until R3 = 0

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

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

[Function]

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

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

[Flag Change]

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

Conditions

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

[Description Example]

RMPA.B

ROLC

Rotate left with carry
ROtate to Left with Carry

ROLC

[Syntax]

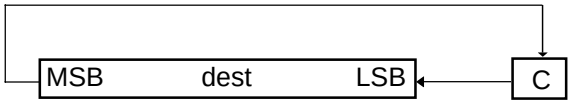
ROLC.size dest

B , W

[Instruction Code/Number of Cycles]

Page=218

[Operation]



[Function]

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

[Selectable dest]

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

[Flag Change]

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

Conditions

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

[Description Example]

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

[Related Instructions] RORC,ROT,SHA,SHL

RORC

Rotate right with carry
ROtate to Right with Carry

RORC

[Syntax]

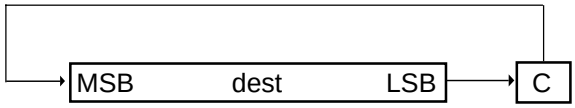
RORC.size dest

B , W

[Instruction Code/Number of Cycles]

Page=219

[Operation]



[Function]

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

[Selectable dest]

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

[Flag Change]

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

Conditions

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

[Description Example]

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

[Related Instructions] ROLC,ROT,SHA,SHL

ROT

Rotate
ROTate

ROT

[Syntax]

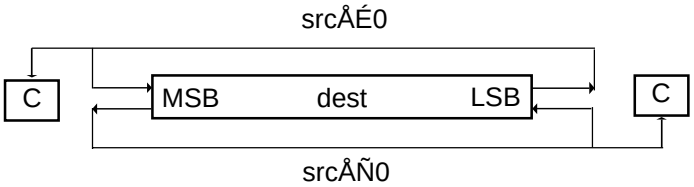
ROT.size src,dest

B , W

[Instruction Code/Number of Cycles]

Page=220

[Operation]



[Function]

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

[Selectable src/dest]

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

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

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

[Flag Change]

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

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

Conditions

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

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

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

[Description Example]

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

[Related Instructions] ROLC,RORC,SHA,SHL

RTS

Return from subroutine

ReTurn from Subroutine

RTS

[Syntax]

RTS

[Instruction Code/Number of Cycles]

Page=221

[Operation]

$PC_{ML} \leftarrow M(SP)$

$SP \leftarrow SP + 2$

$PC_{H} \leftarrow M(SP)$

$SP \leftarrow SP + 1$

[Function]

- This instruction causes control to return from a subroutine.

[Flag Change]

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

[Description Example]

RTS

SBB

Subtract with borrow

SuBtract with Borrow

SBB

[Syntax]

SBB.size src,dest

B , W

[Instruction Code/Number of Cycles]

Page=222

[Operation]

dest ← dest – src – $\overline{C}$

[Function]

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

[Selectable src/dest]

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

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

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
	–	–	○	–	○	○	–	○

Conditions

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

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

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

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

[Description Example]

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

[Related Instructions] ADC,ADCF,ADD,SUB

SBJNZ

Subtract & conditional jump
SuBtract then Jump on Not Zero

SBJNZ

[Syntax]

SBJNZ.size src,dest,label

B , W

[Instruction Code/Number of Cycles]

Page=224

[Operation]

dest ← dest – src

if dest ≠ 0 **then** jump label

[Function]

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

[Selectable src/dest/label]

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

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

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

[Flag Change]

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

[Description Example]

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

[Related Instructions] ADJNZ

SHA

Shift arithmetic
SHift Arithmetic

SHA

[Syntax]

SHA.size src,dest

B , W , L

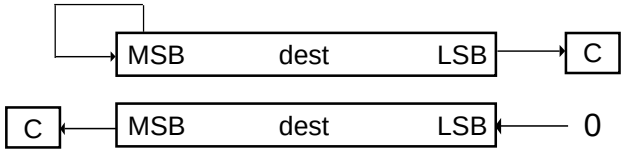
[Instruction Code/Number of Cycles]

Page=225

[Operation]

When *src* < 0

When *src* > 0



[Function]

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

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

[Selectable src/dest]

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

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

[Flag Change]

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

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

Conditions

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

[Description Example]

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

[Related Instructions] ROLC,RORC,ROT,SHL

SHL

Shift logical
SHift Logical

SHL

[Syntax]

SHL.size src,dest

B , W , L

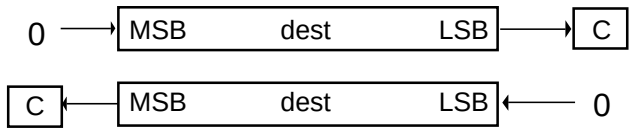
[Instruction Code/Number of Cycles]

Page=228

[Operation]

When *src* < 0

When *src* > 0



[Function]

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

[Selectable src/dest]

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

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

[Flag Change]

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

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

Conditions

S : The flag is set when the operation resulted in MSB = 1; otherwise cleared.
Z : The flag is set when the operation resulted in 0; otherwise cleared. However, the flag is indeterminate if you selected (.L) for the size specifier (.size).
C : The flag is set when the bit shifted out last is 1; otherwise cleared. However, the flag is indeterminate if you selected (.L) for the size specifier (.size).

[Description Example]

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

[Related Instructions] ROLC,RORC,ROT,SHA

SMOVB

Transfer string backward
String MOVe Backward

SMOVB

[Syntax]

SMOVB.size
└ **B , W**

[Instruction Code/Number of Cycles]

Page=230

[Operation][*1]

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

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

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

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

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

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

[Function]

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

[Flag Change]

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

[Description Example]

SMOVB.B

[Related Instructions] SMOVF,SSTR

SMOVF

Transfer string forward
String MOVe Forward

SMOVF

[Syntax]

SMOVF.size

└ **B , W**

[Instruction Code/Number of Cycles]

Page=231

[Operation][*1]

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

Repeat

M(A1) ← M(2^{16} × R1H + A0)
A0[*2*] ← A0 + 1
A1 ← A1 + 1
R3 ← R3 – 1

Until R3 = 0

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

Repeat

M(A1) ← M(2^{16} × R1H + A0)
A0[*2] ← A0 + 2
A1 ← A1 + 2
R3 ← R3 – 1

Until R3 = 0

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

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

[Function]

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

[Flag Change]

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

[Description Example]

SMOVF.W

[Related Instructions] SMOVB,SSTR

SSTR

Store string
String SToRe

SSTR

[Syntax]

SSTR.size
└──── **B , W**

[Instruction Code/Number of Cycles]

Page=231

[Operation][*1]

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

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

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

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

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

[Function]

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

[Flag Change]

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

[Description Example]

SSTR.B

[Related Instructions] SMOVB,SMOVF

STC

Transfer from control register

STore from Control register

STC

[Syntax]

STC src,dest

[Instruction Code/Number of Cycles]

Page=232

[Operation]

dest ← src

[Function]

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

[Selectable src/dest]

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

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

[Flag Change]

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

[Description Example]

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

[Related Instructions] POPC,PUSHC,LDC,LDINTB

STCTX

Save context
STore ConTeXt

STCTX

[Syntax]

STCTX abs16,abs20

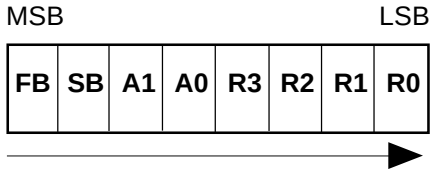
[Instruction Code/Number of Cycles]

Page=233

[Operation]

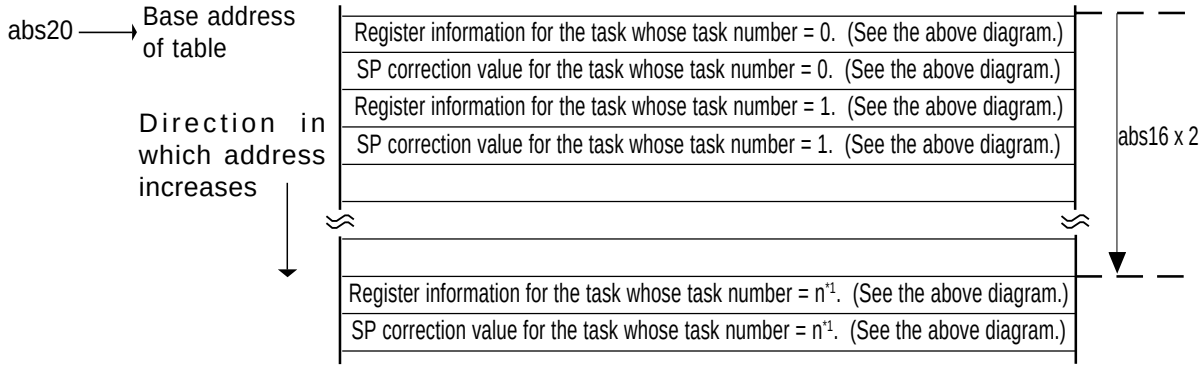
[Function]

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



Transferred sequentially beginning with FB

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



*1 n=0 to 255

[Flag Change]

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

[Description Example]

```
STCTX    Ram,Rom_TBL
```

[Related Instructions] LDCTX

STE

Transfer to extended data area

STore to EXtra far data area

STE

[Syntax]

STE.size **src,dest**

└ **B , W**

[Instruction Code/Number of Cycles]

Page=233

[Operation]

dest ← src

[Function]

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

[Selectable src/dest]

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

[Flag Change]

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

Conditions

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

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

[Description Example]

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

[Related Instructions] MOV,LDE,XCHG

STNZ

Conditional transfer
STore on Not Zero

STNZ

[Syntax]

STNZ **src,dest**

[Instruction Code/Number of Cycles]

Page=235

[Operation]

if Z = 0 **then** dest ← src

[Function]

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

[Selectable src/dest]

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

[Flag Change]

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

[Description Example]

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

[Related Instructions] STZ,STZX

STZ

Conditional transfer
STore on Zero

STZ

[Syntax]

STZ src,dest

[Instruction Code/Number of Cycles]

Page=235

[Operation]

if Z = 1 **then** dest ← src

[Function]

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

[Selectable src/dest]

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

[Flag Change]

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

[Description Example]

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

[Related Instructions] STNZ,STZX

STZX

Conditional transfer
STore on Zero eXtention

STZX

[Syntax]

STZX **src1,src2,dest**

[Instruction Code/Number of Cycles]

Page=236

[Operation]

If Z = 1 **then**

dest ← src1

else

dest ← src2

[Function]

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

[Selectable src/dest]

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

[Flag Change]

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

[Description Example]

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

[Related Instructions] STZ,STNZ

SUB

Subtract without borrow
SUBtract

SUB

[Syntax]

SUB.size (:format) src,dest

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

[Instruction Code/Number of Cycles]

Page=236

[Operation]

dest ← dest – src

[Function]

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

[Selectable src/dest]

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

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

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

[Flag Change]

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

Conditions

- O : The flag is set when a signed operation resulted in exceeding +32767 (.W) or –32768 (.W), or +127 (.B) or –128 (.B); otherwise cleared.
- S : The flag is set when the operation resulted in MSB = 1; otherwise cleared.
- Z : The flag is set when the operation resulted in 0; otherwise cleared.
- C : The flag is set when an unsigned operation resulted in any value equal to or greater than 0; otherwise cleared.

[Description Example]

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

[Related Instructions] ADC,ADCF,ADD,SBB

[src/dest Classified by Format]

G format

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

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

S format*2

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

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

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

TST

Test
TeST

TST

[Syntax]

TST.size **src,dest**

B , W

[Instruction Code/Number of Cycles]

Page=239

[Operation]

dest ∧ src

[Function]

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

[Selectable src/dest]

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

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

[Flag Change]

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

Conditions

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

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

[Description Example]

```
TST.B    #3,R0L
TST.B    A0,R0L        ; A0's 8 low-order bits and ROL are operated on.
TST.B    R0L,A0        ; R0L is zero-expanded and operated on with A0.
```

[Related Instructions] AND,OR,XOR

UND

Interrupt for undefined instruction

UNDefined instruction

UND

[Syntax]

UND

[Instruction Code/Number of Cycles]

Page=241

[Operation]

SP ← SP – 2
M(SP) ← (PC + 1)H, FLG
SP ← SP – 2
M(SP) ← (PC + 1)ML
PC ← M(FFFDC16)

[Function]

- This instruction generates an undefined instruction interrupt.
- The undefined instruction interrupt is a nonmaskable interrupt.

[Flag Change]

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

*1 The flags are saved to the stack area before the UND instruction is executed. After the interrupt, the flag status becomes as shown on the left.

Conditions

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

[Description Example]

UND

WAIT

Wait
WAIT

WAIT

[Syntax]

WAIT

[Instruction Code/Number of Cycles]

Page=241

[Operation]

[Function]

- This instruction halts program execution. Program execution is restarted when an interrupt of a higher priority level than IPL is acknowledged or a reset is generated.

[Flag Change]

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

[Description Example]

WAIT

XCHG

Exchange
eXCHanGe

XCHG

[Syntax]

XCHG.size src,dest

B , W

[Instruction Code/Number of Cycles]

Page=242

[Operation]

dest ←→ src

[Function]

- This instruction exchanges contents between *src* and *dest*.
- If *dest* is an A0 or A1 when the size specifier (.size) you selected is (.B), 16 bits of zero- expanded *src* data are placed in the A0 or A1 and the 8 low-order bits of the A0 or A1 are placed in *src*.

[Selectable src/dest]

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

[Flag Change]

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

[Description Example]

```
XCHG.B    R0L,A0        ; A0's 8 low-order bits and R0L's zero-expanded value are exchanged.
XCHG.W    R0,A1
XCHG.B    R0L,[A0]
```

[Related Instructions] MOV,LDE,STE

XOR

Exclusive OR
eXclusive OR

XOR

[Syntax]

XOR.size src,dest

└ **B , W**

[Instruction Code/Number of Cycles]

Page=243

[Operation]

dest ← dest ∀ src

[Function]

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

[Selectable src/dest]

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

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

[Flag Change]

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

Conditions

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

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

[Description Example]

```
XOR.B   A0,R0L      ; A0's 8 low-order bits and R0L are exclusive ORed.
XOR.B   R0L,A0      ; R0L is zero-expanded and exclusive ORed with A0.
XOR.B   #3,R0L
XOR.W   A0,A1
```

[Related Instructions] AND,OR,TST

Blank for page layout

Chapter 4

Instruction Code/Number of Cycles

4.1 Guide to This Chapter

This chapter describes instruction code and number of cycles for each op-code.

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

(1) **LDIPL**

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

(3)

b7							b0	b7				b0
0	1	1	1	1	1	0	1	1	0	1	0	IMM4

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

Bytes/Cycles	2/2

(1) **MOV**

(2) **(1) MOV.size:G #IMM, dest**

(3)

b7							b0	b7				b0
0	1	1	1	0	1	0	SIZE	1	1	0	0	DEST

dest code

dsp8	#IMM8
dsp16/abs16	#IMM16

.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0000	dsp:8[An]	dsp:8[A0]	1000
	R0H/R1	0001		dsp:8[A1]	1001
	R1L/R2	0010	dsp:8[SB/FB]	dsp:8[SB]	1010
	R1H/R3	0011		dsp:8[FB]	1011
An	A0	0100	dsp:16[An]	dsp:16[A0]	1100
	A1	0101		dsp:16[A1]	1101
[An]	[A0]	0110	dsp:16[SB]	dsp:16[SB]	1110
	[A1]	0111	abs16	abs16	1111

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

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

(1) Mnemonic

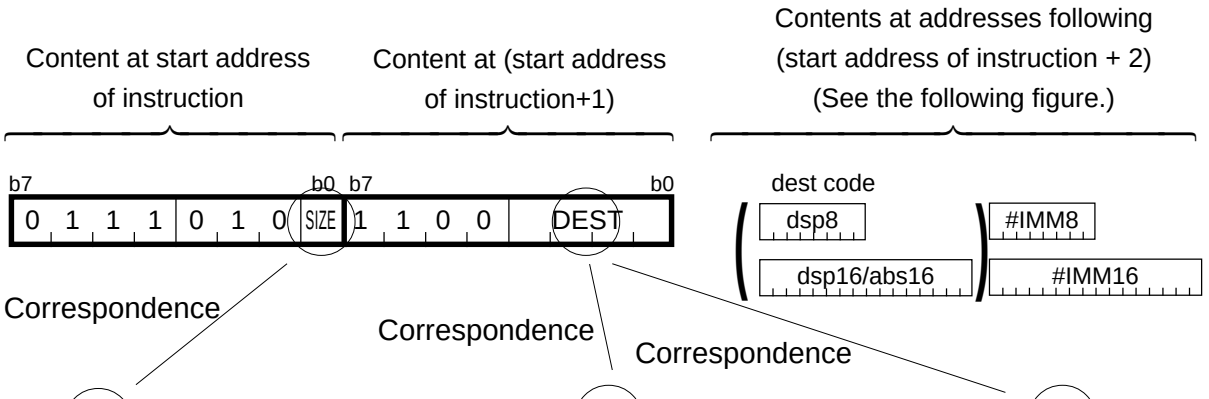
Shows the mnemonic explained in this page.

(2) Syntax

Shows an instruction syntax using symbols.

(3) Instruction code

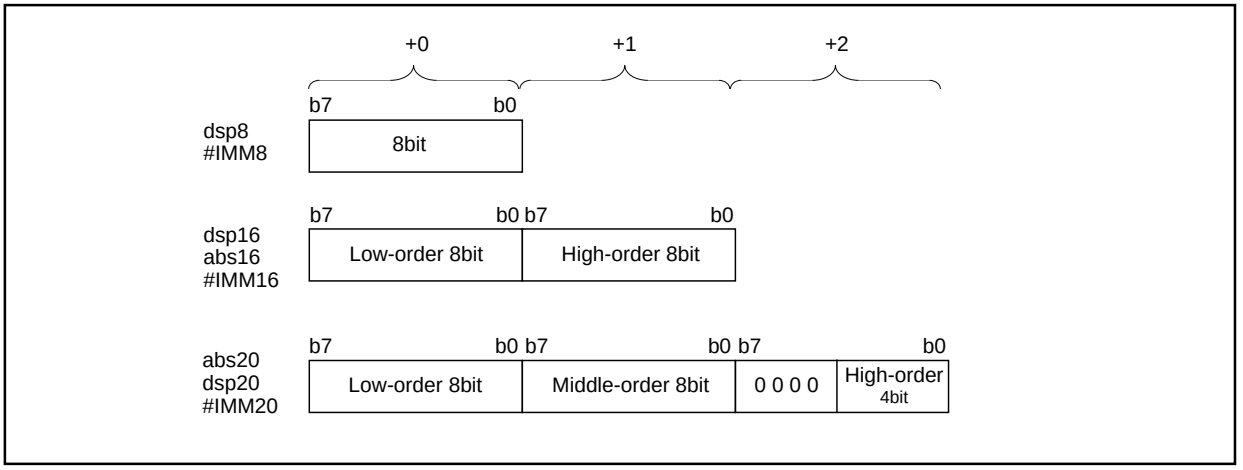
Shows instruction code. Entered in ( ) are omitted depending on src/dest you selected.



.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0000	dsp:8[An]	dsp:8[A0]	1000
	R0H/R1	0001		dsp:8[A1]	1001
	R1L/R2	0010	dsp:8[SB/FB]	dsp:8[SB]	1010
	R1H/R3	0011		dsp:8[FB]	1011
An	A0	0100	dsp:16[An]	dsp:16[A0]	1100
	A1	0101		dsp:16[A1]	1101
[An]	[A0]	0110	dsp:16[SB]	dsp:16[SB]	1110
	[A1]	0111	abs16	abs16	1111

Contents at addresses following (start address of instruction + 2) are arranged as follows:



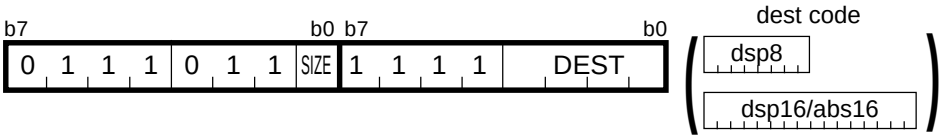
(4) Table of cycles

Shows the number of cycles required to execute this instruction and the number of instruction bytes. There is a chance that the number of cycles increases due to an effect of software wait.

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

ABS

(1) ABS.size dest



.size	SIZE
.B	0
.W	1

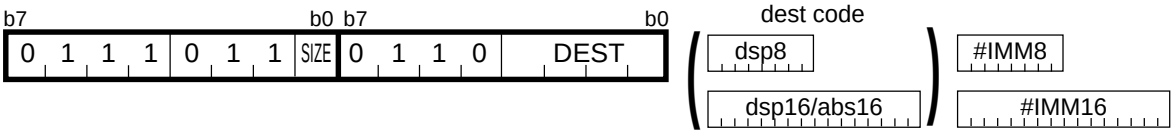
dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

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

ADC

(1) ADC.size #IMM, dest



.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

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

*1 If the size specifier (.size) is (.W), the number of bytes above is increased by 1.

ADC

(2) ADC.size src, dest

b7							b0	b7		b0
1	0	1	1	0	0	0	SIZE	SRC		DEST

src code: dsp8 / dsp16/abs16

dest code: dsp8 / dsp16/abs16

.size	SIZE
.B	0
.W	1

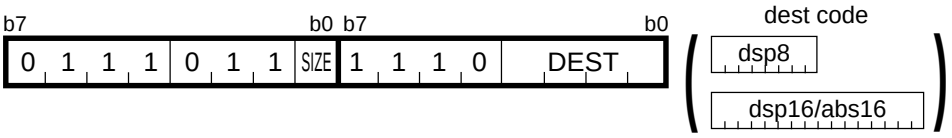
src/dest		SRC/DEST	src/dest		SRC/DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Rn	2/2	2/2	2/3	3/3	3/3	4/3	4/3	4/3
An	2/2	2/2	2/3	3/3	3/3	4/3	4/3	4/3
[An]	2/3	2/3	2/4	3/4	3/4	4/4	4/4	4/4
dsp:8[An]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	5/4
dsp:8[SB/FB]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	5/4
dsp:16[An]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4
dsp:16[SB]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4
abs16	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4

ADCF

(1) ADCF.size dest



.size	SIZE
.B	0
.W	1

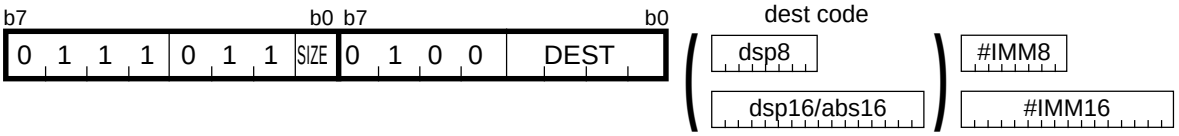
dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

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

ADD

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



.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

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

*1 If the size specifier (.size) is (.W), the number of bytes above is increased by 1.

ADD

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

b7							b0	b7				b0
1	1	0	0	1	0	0	SIZE	IMM4		DEST		

dest code

dsp8
dsp16/abs16

.size	SIZE
.B	0
.W	1

#IMM	IMM4	#IMM	IMM4
0	0 0 0 0	–8	1 0 0 0
+1	0 0 0 1	–7	1 0 0 1
+2	0 0 1 0	–6	1 0 1 0
+3	0 0 1 1	–5	1 0 1 1
+4	0 1 0 0	–4	1 1 0 0
+5	0 1 0 1	–3	1 1 0 1
+6	0 1 1 0	–2	1 1 1 0
+7	0 1 1 1	–1	1 1 1 1

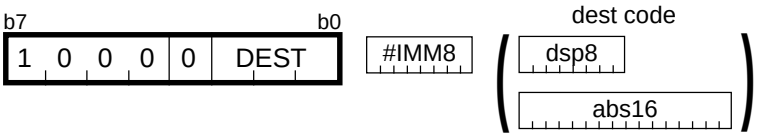
dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

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

ADD

(3) ADD.B:S #IMM8, dest



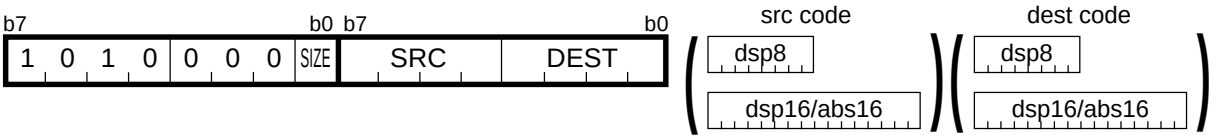
dest		DEST
Rn	R0H	0 1 1
	R0L	1 0 0
dsp:8[SB/FB]	dsp:8[SB]	1 0 1
	dsp:8[FB]	1 1 0
abs16	abs16	1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	2/1	3/3	4/3

ADD

(4) ADD.size:G src, dest



.size	SIZE
.B	0
.W	1

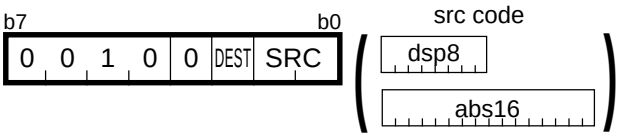
src/dest		SRC/DEST	src/dest		SRC/DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Rn	2/2	2/2	2/3	3/3	3/3	4/3	4/3	4/3
An	2/2	2/2	2/3	3/3	3/3	4/3	4/3	4/3
[An]	2/3	2/3	2/4	3/4	3/4	4/4	4/4	4/4
dsp:8[An]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	5/4
dsp:8[SB/FB]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	5/4
dsp:16[An]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4
dsp:16[SB]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4
abs16	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4

ADD

(5) ADD.B:S src, R0L/R0H



src		SRC
Rn	R0L/R0H	0 0
dsp:8[SB/FB]	dsp:8[SB]	0 1
	dsp:8[FB]	1 0
abs16	abs16	1 1

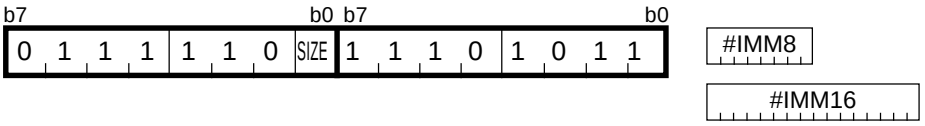
dest	DEST
R0L	0
R0H	1

[Number of Bytes/Number of Cycles]

src	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	1/2	2/3	3/3

ADD

(6) ADD.size:G #IMM, SP



.size	SIZE
.B	0
.W	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	3/2

*1 If the size specifier (.size) is (.W), the number of bytes above is increased by 1.

ADD

(7) ADD.size:Q #IMM, SP

b7							b0	b7							b0
0	1	1	1	1	1	0	1	1	0	1	1	IMM4			

*1 The instruction code is the same regardless of whether you selected (.B) or (.W) for the size specifier (.size).

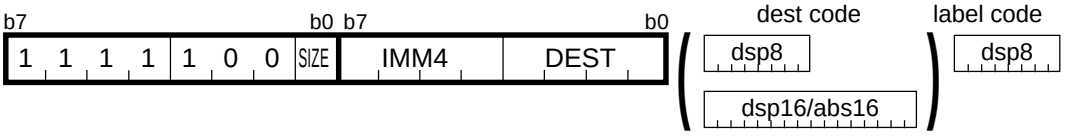
#IMM	IMM4	**#IMM**	IMM4
0	0 0 0 0	–8	1 0 0 0
+1	0 0 0 1	–7	1 0 0 1
+2	0 0 1 0	–6	1 0 1 0
+3	0 0 1 1	–5	1 0 1 1
+4	0 1 0 0	–4	1 1 0 0
+5	0 1 0 1	–3	1 1 0 1
+6	0 1 1 0	–2	1 1 1 0
+7	0 1 1 1	–1	1 1 1 1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/1

ADJNZ

(1) ADJNZ.size #IMM, dest, label



dsp8 (label code)= address indicated by label –(start address of instruction + 2)

.size	SIZE
.B	0
.W	1

#IMM	IMM4	#IMM	IMM4
0	0 0 0 0	–8	1 0 0 0
+1	0 0 0 1	–7	1 0 0 1
+2	0 0 1 0	–6	1 0 1 0
+3	0 0 1 1	–5	1 0 1 1
+4	0 1 0 0	–4	1 1 0 0
+5	0 1 0 1	–3	1 1 0 1
+6	0 1 1 0	–2	1 1 1 0
+7	0 1 1 1	–1	1 1 1 1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

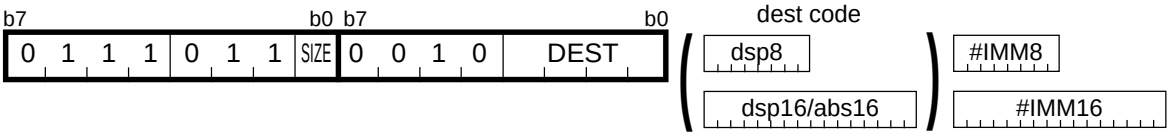
[Number of Bytes/Number of Cycles]

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

*1 If branched to label, the number of cycles above is increased by 4.

AND

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



.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

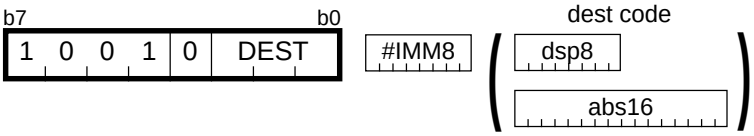
[Number of Bytes/Number of Cycles]

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

*1 If the size specifier (.size) is (.W), the number of bytes above is increased by 1.

AND

(2) AND.B:S #IMM8, dest



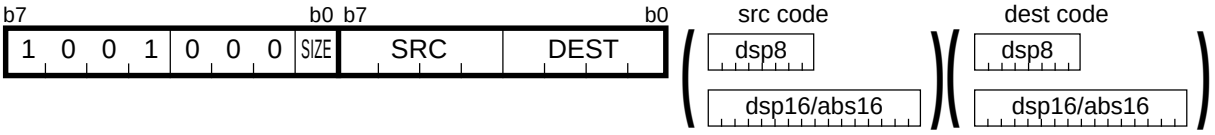
dest		DEST
Rn	R0H	0 1 1
	R0L	1 0 0
dsp:8[SB/FB]	dsp:8[SB]	1 0 1
	dsp:8[FB]	1 1 0
abs16	abs16	1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	2/1	3/3	4/3

AND

(3) AND.size:G src, dest



.size	SIZE
.B	0
.W	1

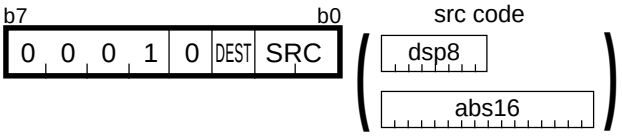
src/dest		SRC/DEST	src/dest		SRC/DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Rn	2/2	2/2	2/3	3/3	3/3	4/3	4/3	4/3
An	2/2	2/2	2/3	3/3	3/3	4/3	4/3	4/3
[An]	2/3	2/3	2/4	3/4	3/4	4/4	4/4	4/4
dsp:8[An]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	5/4
dsp:8[SB/FB]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	5/4
dsp:16[An]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4
dsp:16[SB]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4
abs16	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4

AND

(4) AND.B:S src, R0L/R0H



src		SRC
Rn	R0L/R0H	0 0
dsp:8[SB/FB]	dsp:8[SB]	0 1
	dsp:8[FB]	1 0
abs16	abs16	1 1

dest	DEST
R0L	0
R0H	1

[Number of Bytes/Number of Cycles]

src	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	1/2	2/3	3/3

BAND

(1) BAND src

b7							b0	b7				b0
0	1	1	1	1	1	1	0	0	1	0	0	SRC

src code
(dsp8 / dsp16)

src		SRC	src		SRC
bit,Rn	bit,R0	0 0 0 0	base:8[An]	base:8[A0]	1 0 0 0
	bit,R1	0 0 0 1		base:8[A1]	1 0 0 1
	bit,R2	0 0 1 0	bit,base:8 [SB/FB]	bit,base:8[SB]	1 0 1 0
	bit,R3	0 0 1 1		bit,base:8[FB]	1 0 1 1
bit,An	bit,A0	0 1 0 0	base:16[An]	base:16[A0]	1 1 0 0
	bit,A1	0 1 0 1		base:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	bit,base:16[SB]	bit,base:16[SB]	1 1 1 0
	[A1]	0 1 1 1	bit,base:16	bit,base:16	1 1 1 1

[Number of Bytes/Number of Cycles]

src	bit,Rn	bit,An	[An]	base:8 [An]	bit,base:8 [SB/FB]	base:16 [An]	bit,base:16 [SB]	bit,base:16
Bytes/Cycles	3/3	3/3	2/7	3/7	3/4	4/7	4/4	4/4

BCLR

(1) BCLR:G dest

b7							b0	b7				b0
0	1	1	1	1	1	1	0	1	0	0	0	DEST

dest code
(dsp8 / dsp16)

dest		DEST	dest		DEST
bit,Rn	bit,R0	0 0 0 0	base:8[An]	base:8[A0]	1 0 0 0
	bit,R1	0 0 0 1		base:8[A1]	1 0 0 1
	bit,R2	0 0 1 0	bit,base:8 [SB/FB]	bit,base:8[SB]	1 0 1 0
	bit,R3	0 0 1 1		bit,base:8[FB]	1 0 1 1
bit,An	bit,A0	0 1 0 0	base:16[An]	base:16[A0]	1 1 0 0
	bit,A1	0 1 0 1		base:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	bit,base:16[SB]	bit,base:16[SB]	1 1 1 0
	[A1]	0 1 1 1	bit,base:16	bit,base:16	1 1 1 1

[Number of Bytes/Number of Cycles]

dest	bit,Rn	bit,An	[An]	base:8 [An]	bit,base:8 [SB/FB]	base:16 [An]	bit,base:16 [SB]	bit,base:16
Bytes/Cycles	3/2	3/2	2/6	3/6	3/3	4/6	4/3	4/3

BCLR

(2) BCLR:S bit, base:11[SB]

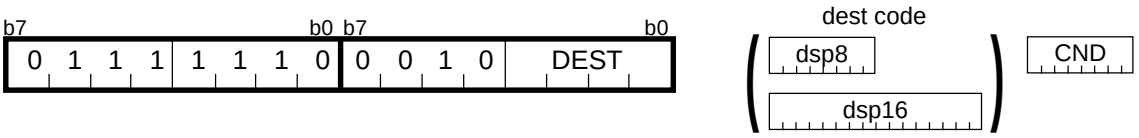
b7					b0	dest code
0	1	0	0	0	BIT	dsp8

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/3

BM*Cnd*

(1) BM*Cnd* dest



dest		DEST	dest		DEST
bit,Rn	bit,R0	0 0 0 0	base:8[An]	base:8[A0]	1 0 0 0
	bit,R1	0 0 0 1		base:8[A1]	1 0 0 1
	bit,R2	0 0 1 0	bit,base:8 [SB/FB]	bit,base:8[SB]	1 0 1 0
	bit,R3	0 0 1 1		bit,base:8[FB]	1 0 1 1
bit,An	bit,A0	0 1 0 0	base:16[An]	base:16[A0]	1 1 0 0
	bit,A1	0 1 0 1		base:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	bit,base:16[SB]	bit,base:16[SB]	1 1 1 0
	[A1]	0 1 1 1	bit,base:16	bit,base:16	1 1 1 1

Cnd	CND	***Cnd***	CND
GEU/C	0 0 0 0 0 0 0 0	LTU/NC	1 1 1 1 1 0 0 0
GTU	0 0 0 0 0 0 0 1	LEU	1 1 1 1 1 0 0 1
EQ/Z	0 0 0 0 0 0 1 0	NE/NZ	1 1 1 1 1 0 1 0
N	0 0 0 0 0 0 1 1	PZ	1 1 1 1 1 0 1 1
LE	0 0 0 0 0 1 0 0	GT	1 1 1 1 1 1 0 0
O	0 0 0 0 0 1 0 1	NO	1 1 1 1 1 1 0 1
GE	0 0 0 0 0 1 1 0	LT	1 1 1 1 1 1 1 0

[Number of Bytes/Number of Cycles]

dest	bit,Rn	bit,An	[An]	base:8 [An]	bit,base:8 [SB/FB]	base:16 [An]	bit,base:16 [SB]	bit,base:16
Bytes/Cycles	4/6	4/6	3/10	4/10	4/7	5/10	5/7	5/7

BM*Cnd*

(2) BM*Cnd* C

b7							b0	b7				b0
0	1	1	1	1	1	0	1	1	1	0	1	CND

Cnd	CND	*Cnd*	CND
GEU/C	0 0 0 0	PZ	0 1 1 1
GTU	0 0 0 1	LE	1 0 0 0
EQ/Z	0 0 1 0	O	1 0 0 1
N	0 0 1 1	GE	1 0 1 0
LTU/NC	0 1 0 0	GT	1 1 0 0
LEU	0 1 0 1	NO	1 1 0 1
NE/NZ	0 1 1 0	LT	1 1 1 0

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/1

*1 If the condition is true, the number of cycles above is increased by 1.

BNAND

(1) BNAND src

b7							b0	b7				b0
0	1	1	1	1	1	1	0	0	1	0	1	SRC

src code

src		SRC	src		SRC
bit,Rn	bit,R0	0 0 0 0	base:8[An]	base:8[A0]	1 0 0 0
	bit,R1	0 0 0 1		base:8[A1]	1 0 0 1
	bit,R2	0 0 1 0	bit,base:8 [SB/FB]	bit,base:8[SB]	1 0 1 0
	bit,R3	0 0 1 1		bit,base:8[FB]	1 0 1 1
bit,An	bit,A0	0 1 0 0	base:16[An]	base:16[A0]	1 1 0 0
	bit,A1	0 1 0 1		base:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	bit,base:16[SB]	bit,base:16[SB]	1 1 1 0
	[A1]	0 1 1 1	bit,base:16	bit,base:16	1 1 1 1

[Number of Bytes/Number of Cycles]

src	bit,Rn	bit,An	[An]	base:8 [An]	bit,base:8 [SB/FB]	base:16 [An]	bit,base:16 [SB]	bit,base:16
Bytes/Cycles	3/3	3/3	2/7	3/7	3/4	4/7	4/4	4/4

BNOR

(1) BNOR src

b7							b0	b7				b0
0	1	1	1	1	1	1	0	0	1	1	1	SRC

src code
(dsp8 / dsp16)

src		SRC	src		SRC
bit,Rn	bit,R0	0 0 0 0	base:8[An]	base:8[A0]	1 0 0 0
	bit,R1	0 0 0 1		base:8[A1]	1 0 0 1
	bit,R2	0 0 1 0	bit,base:8 [SB/FB]	bit,base:8[SB]	1 0 1 0
	bit,R3	0 0 1 1		bit,base:8[FB]	1 0 1 1
bit,An	bit,A0	0 1 0 0	base:16[An]	base:16[A0]	1 1 0 0
	bit,A1	0 1 0 1		base:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	bit,base:16[SB]	bit,base:16[SB]	1 1 1 0
	[A1]	0 1 1 1	bit,base:16	bit,base:16	1 1 1 1

[Number of Bytes/Number of Cycles]

src	bit,Rn	bit,An	[An]	base:8 [An]	bit,base:8 [SB/FB]	base:16 [An]	bit,base:16 [SB]	bit,base:16
Bytes/Cycles	3/3	3/3	2/7	3/7	3/4	4/7	4/4	4/4

BNOT

(1) BNOT:G dest

b7							b0	b7				b0
0	1	1	1	1	1	1	0	1	0	1	0	DEST

dest code
(dsp8 / dsp16)

dest		DEST	dest		DEST
bit,Rn	bit,R0	0 0 0 0	base:8[An]	base:8[A0]	1 0 0 0
	bit,R1	0 0 0 1		base:8[A1]	1 0 0 1
	bit,R2	0 0 1 0	bit,base:8 [SB/FB]	bit,base:8[SB]	1 0 1 0
	bit,R3	0 0 1 1		bit,base:8[FB]	1 0 1 1
bit,An	bit,A0	0 1 0 0	base:16[An]	base:16[A0]	1 1 0 0
	bit,A1	0 1 0 1		base:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	bit,base:16[SB]	bit,base:16[SB]	1 1 1 0
	[A1]	0 1 1 1	bit,base:16	bit,base:16	1 1 1 1

[Number of Bytes/Number of Cycles]

dest	bit,Rn	bit,An	[An]	base:8 [An]	bit,base:8 [SB/FB]	base:16 [An]	bit,base:16 [SB]	bit,base:16
Bytes/Cycles	3/2	3/2	2/6	3/6	3/3	4/6	4/3	4/3

BNOT

(2) BNOT:S bit, base:11[SB]

b7					b0
0 1 0 1				0	BIT

dest code

dsp8

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/3

BNTST

(1) BNTST src

b7							b0	b7				b0
0 1 1 1				1 1 1 0				0 0 1 1				SRC

src code

(dsp8 / dsp16)

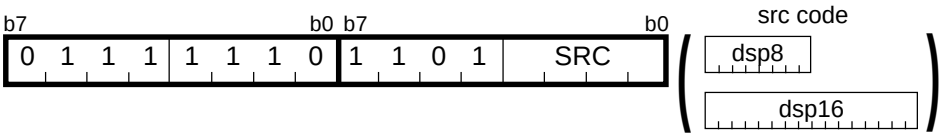
src		SRC	src		SRC
bit,Rn	bit,R0	0 0 0 0	base:8[An]	base:8[A0]	1 0 0 0
	bit,R1	0 0 0 1		base:8[A1]	1 0 0 1
	bit,R2	0 0 1 0	bit,base:8 [SB/FB]	bit,base:8[SB]	1 0 1 0
	bit,R3	0 0 1 1		bit,base:8[FB]	1 0 1 1
bit,An	bit,A0	0 1 0 0	base:16[An]	base:16[A0]	1 1 0 0
	bit,A1	0 1 0 1		base:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	bit,base:16[SB]	bit,base:16[SB]	1 1 1 0
	[A1]	0 1 1 1	bit,base:16	bit,base:16	1 1 1 1

[Number of Bytes/Number of Cycles]

src	bit,Rn	bit,An	[An]	base:8 [An]	bit,base:8 [SB/FB]	base:16 [An]	bit,base:16 [SB]	bit,base:16
Bytes/Cycles	3/3	3/3	2/7	3/7	3/4	4/7	4/4	4/4

BNXOR

(1) BNXOR src



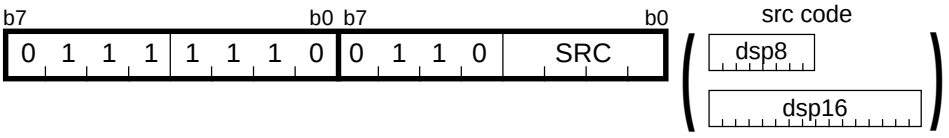
src		SRC	src		SRC
bit,Rn	bit,R0	0 0 0 0	base:8[An]	base:8[A0]	1 0 0 0
	bit,R1	0 0 0 1		base:8[A1]	1 0 0 1
	bit,R2	0 0 1 0	bit,base:8 [SB/FB]	bit,base:8[SB]	1 0 1 0
	bit,R3	0 0 1 1		bit,base:8[FB]	1 0 1 1
bit,An	bit,A0	0 1 0 0	base:16[An]	base:16[A0]	1 1 0 0
	bit,A1	0 1 0 1		base:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	bit,base:16[SB]	bit,base:16[SB]	1 1 1 0
	[A1]	0 1 1 1	bit,base:16	bit,base:16	1 1 1 1

[Number of Bytes/Number of Cycles]

src	bit,Rn	bit,An	[An]	base:8 [An]	bit,base:8 [SB/FB]	base:16 [An]	bit,base:16 [SB]	bit,base:16
Bytes/Cycles	3/3	3/3	2/7	3/7	3/4	4/7	4/4	4/4

BOR

(1) BOR src



src		SRC	src		SRC
bit,Rn	bit,R0	0 0 0 0	base:8[An]	base:8[A0]	1 0 0 0
	bit,R1	0 0 0 1		base:8[A1]	1 0 0 1
	bit,R2	0 0 1 0	bit,base:8 [SB/FB]	bit,base:8[SB]	1 0 1 0
	bit,R3	0 0 1 1		bit,base:8[FB]	1 0 1 1
bit,An	bit,A0	0 1 0 0	base:16[An]	base:16[A0]	1 1 0 0
	bit,A1	0 1 0 1		base:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	bit,base:16[SB]	bit,base:16[SB]	1 1 1 0
	[A1]	0 1 1 1	bit,base:16	bit,base:16	1 1 1 1

[Number of Bytes/Number of Cycles]

src	bit,Rn	bit,An	[An]	base:8 [An]	bit,base:8 [SB/FB]	base:16 [An]	bit,base:16 [SB]	bit,base:16
Bytes/Cycles	3/3	3/3	2/7	3/7	3/4	4/7	4/4	4/4

BRK

(1) BRK

b7							b0
0	0	0	0	0	0	0	0

[Number of Bytes/Number of Cycles]

Bytes/Cycles	1/27

*1 If you specify the target address of the BRK interrupt by use of the interrupt table register (INTB), the number of cycles shown in the table increases by two. At this time, set FF_{16} in addresses $FFFE4_{16}$ through $FFFE7_{16}$.

BSET

(1) BSET:G dest

b7							b0	b7				b0
0	1	1	1	1	1	1	0	1	0	0	1	DEST

dest code

(dsp8 / dsp16)

dest		DEST	dest		DEST
bit,Rn	bit,R0	0 0 0 0	base:8[An]	base:8[A0]	1 0 0 0
	bit,R1	0 0 0 1		base:8[A1]	1 0 0 1
	bit,R2	0 0 1 0	bit,base:8 [SB/FB]	bit,base:8[SB]	1 0 1 0
	bit,R3	0 0 1 1		bit,base:8[FB]	1 0 1 1
bit,An	bit,A0	0 1 0 0	base:16[An]	base:16[A0]	1 1 0 0
	bit,A1	0 1 0 1		base:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	bit,base:16[SB]	bit,base:16[SB]	1 1 1 0
	[A1]	0 1 1 1	bit,base:16	bit,base:16	1 1 1 1

[Number of Bytes/Number of Cycles]

dest	bit,Rn	bit,An	[An]	base:8 [An]	bit,base:8 [SB/FB]	base:16 [An]	bit,base:16 [SB]	bit,base:16
Bytes/Cycles	3/2	3/2	2/6	3/6	3/3	4/6	4/3	4/3

BSET

(2) BSET:S bit, base:11[SB]

b7					b0
0 1 0 0	1	BIT			

dest code: dsp8

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/3

BTST

(1) BTST:G src

b7							b0	b7							b0
0 1 1 1				1 1 1 0				1 0 1 1				SRC			

src code: (dsp8 / dsp16)

src		SRC	src		SRC
bit,Rn	bit,R0	0 0 0 0	base:8[An]	base:8[A0]	1 0 0 0
	bit,R1	0 0 0 1		base:8[A1]	1 0 0 1
	bit,R2	0 0 1 0	bit,base:8 [SB/FB]	bit,base:8[SB]	1 0 1 0
	bit,R3	0 0 1 1		bit,base:8[FB]	1 0 1 1
bit,An	bit,A0	0 1 0 0	base:16[An]	base:16[A0]	1 1 0 0
	bit,A1	0 1 0 1		base:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	bit,base:16[SB]	bit,base:16[SB]	1 1 1 0
	[A1]	0 1 1 1	bit,base:16	bit,base:16	1 1 1 1

[Number of Bytes/Number of Cycles]

src	bit,Rn	bit,An	[An]	base:8 [An]	bit,base:8 [SB/FB]	base:16 [An]	bit,base:16 [SB]	bit,base:16
Bytes/Cycles	3/2	3/2	2/6	3/6	3/3	4/6	4/3	4/3

BTST

(2) BTST:S bit, base:11[SB]

b7					b0
0 1 0 1	1	BIT			

src code: dsp8

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/3

BTSTC

(1) BTSTC dest

b7							b0	b7					b0
0 1 1 1				1 1 1 0				0 0 0 0				DEST	

dest code: (dsp8 / dsp16)

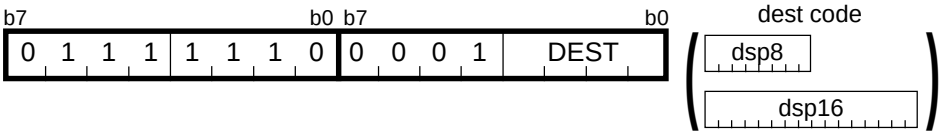
dest		DEST	dest		DEST
bit,Rn	bit,R0	0 0 0 0	base:8[An]	base:8[A0]	1 0 0 0
	bit,R1	0 0 0 1		base:8[A1]	1 0 0 1
	bit,R2	0 0 1 0	bit,base:8 [SB/FB]	bit,base:8[SB]	1 0 1 0
	bit,R3	0 0 1 1		bit,base:8[FB]	1 0 1 1
bit,An	bit,A0	0 1 0 0	base:16[An]	base:16[A0]	1 1 0 0
	bit,A1	0 1 0 1		base:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	bit,base:16[SB]	bit,base:16[SB]	1 1 1 0
	[A1]	0 1 1 1	bit,base:16	bit,base:16	1 1 1 1

[Number of Bytes/Number of Cycles]

dest	bit,Rn	bit,An	[An]	base:8 [An]	bit,base:8 [SB/FB]	base:16 [An]	bit,base:16 [SB]	bit,base:16
Bytes/Cycles	3/3	3/3	2/7	3/7	3/4	4/7	4/4	4/4

BTSTS

(1) BTSTS dest



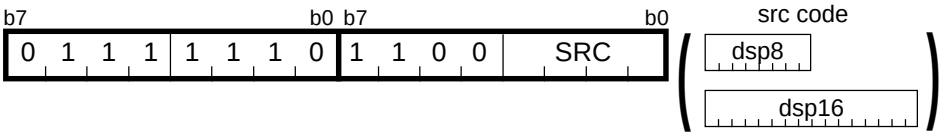
dest		DEST	dest		DEST
bit,Rn	bit,R0	0 0 0 0	base:8[An]	base:8[A0]	1 0 0 0
	bit,R1	0 0 0 1		base:8[A1]	1 0 0 1
	bit,R2	0 0 1 0	bit,base:8 [SB/FB]	bit,base:8[SB]	1 0 1 0
	bit,R3	0 0 1 1		bit,base:8[FB]	1 0 1 1
bit,An	bit,A0	0 1 0 0	base:16[An]	base:16[A0]	1 1 0 0
	bit,A1	0 1 0 1		base:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	bit,base:16[SB]	bit,base:16[SB]	1 1 1 0
	[A1]	0 1 1 1	bit,base:16	bit,base:16	1 1 1 1

[Number of Bytes/Number of Cycles]

dest	bit,Rn	bit,An	[An]	base:8 [An]	bit,base:8 [SB/FB]	base:16 [An]	bit,base:16 [SB]	bit,base:16
Bytes/Cycles	3/3	3/3	2/7	3/7	3/4	4/7	4/4	4/4

BXOR

(1) BXOR src



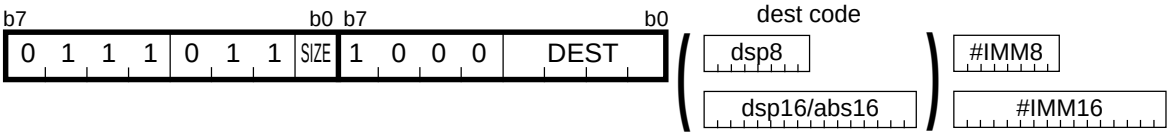
src		SRC	src		SRC
bit,Rn	bit,R0	0 0 0 0	base:8[An]	base:8[A0]	1 0 0 0
	bit,R1	0 0 0 1		base:8[A1]	1 0 0 1
	bit,R2	0 0 1 0	bit,base:8 [SB/FB]	bit,base:8[SB]	1 0 1 0
	bit,R3	0 0 1 1		bit,base:8[FB]	1 0 1 1
bit,An	bit,A0	0 1 0 0	base:16[An]	base:16[A0]	1 1 0 0
	bit,A1	0 1 0 1		base:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	bit,base:16[SB]	bit,base:16[SB]	1 1 1 0
	[A1]	0 1 1 1	bit,base:16	bit,base:16	1 1 1 1

[Number of Bytes/Number of Cycles]

src	bit,Rn	bit,An	[An]	base:8 [An]	bit,base:8 [SB/FB]	base:16 [An]	bit,base:16 [SB]	bit,base:16
Bytes/Cycles	3/3	3/3	2/7	3/7	3/4	4/7	4/4	4/4

CMP

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



.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

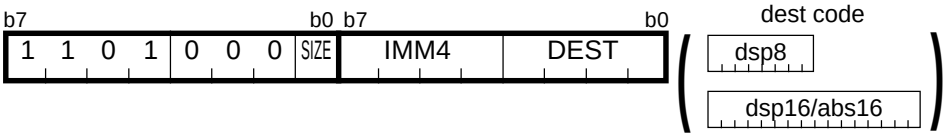
[Number of Bytes/Number of Cycles]

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

*1 If the size specifier (.size) is (.W), the number of bytes above is increased by 1.

CMP

(2) CMP.size:Q #IMM, dest



.size	SIZE
.B	0
.W	1

#IMM	IMM4	#IMM	IMM4
0	0 0 0 0	–8	1 0 0 0
+1	0 0 0 1	–7	1 0 0 1
+2	0 0 1 0	–6	1 0 1 0
+3	0 0 1 1	–5	1 0 1 1
+4	0 1 0 0	–4	1 1 0 0
+5	0 1 0 1	–3	1 1 0 1
+6	0 1 1 0	–2	1 1 1 0
+7	0 1 1 1	–1	1 1 1 1

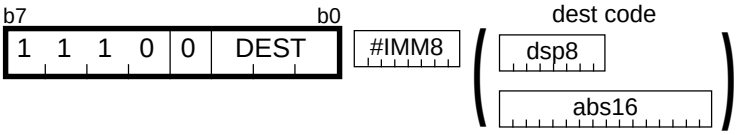
dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

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

CMP

(3) CMP.B:S #IMM8, dest



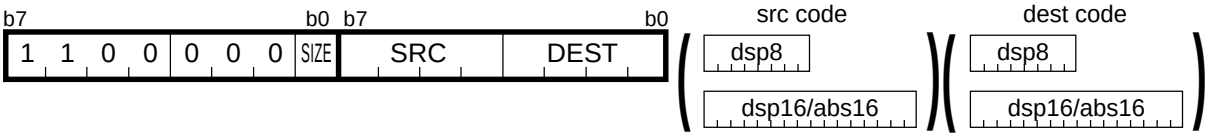
dest		DEST
Rn	R0H	0 1 1
	R0L	1 0 0
dsp:8[SB/FB]	dsp:8[SB]	1 0 1
	dsp:8[FB]	1 1 0
abs16	abs16	1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	2/1	3/3	4/3

CMP

(4) CMP.size:G src, dest



.size	SIZE
.B	0
.W	1

src/dest		SRC/DEST	src/dest		SRC/DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Rn	2/2	2/2	2/3	3/3	3/3	4/3	4/3	4/3
An	2/2	2/2	2/3	3/3	3/3	4/3	4/3	4/3
[An]	2/3	2/3	2/4	3/4	3/4	4/4	4/4	4/4
dsp:8[An]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	5/4
dsp:8[SB/FB]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	5/4
dsp:16[An]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4
dsp:16[SB]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4
abs16	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4

CMP

(5) CMP.B:S src, R0L/R0H

b7						b0
0	0	1	1	1	DEST	SRC

(src code: dsp8 / abs16)

src		SRC
Rn	R0L/R0H	0 0
dsp:8[SB/FB]	dsp:8[SB]	0 1
	dsp:8[FB]	1 0
abs16	abs16	1 1

dest	DEST
R0L	0
R0H	1

[Number of Bytes/Number of Cycles]

src	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	1/2	2/3	3/3

DADC

(1) DADC.B #IMM8, R0L

b7							b0	b7							b0
0	1	1	1	1	1	0	0	1	1	1	0	1	1	1	0

#IMM8

[Number of Bytes/Number of Cycles]

Bytes/Cycles	3/5

DADC

(2) DADC.W #IMM16, R0

b7							b0	b7							b0	
0	1	1	1	1	1	0	1	1	1	1	0	1	1	1	0	#IMM16

[Number of Bytes/Number of Cycles]

Bytes/Cycles	4/5

DADC

(3) DADC.B R0H, R0L

b7							b0	b7							b0
0	1	1	1	1	1	0	0	1	1	1	0	0	1	1	0

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/5

DADC

(4) DADC.W R1, R0

b7							b0	b7							b0
0	1	1	1	1	1	0	1	1	1	1	0	0	1	1	0

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/5

DADD

(1) DADD.B #IMM8, R0L

b7							b0	b7							b0	
0	1	1	1	1	1	0	0	1	1	1	0	1	1	0	0	#IMM8

[Number of Bytes/Number of Cycles]

Bytes/Cycles	3/5

DADD

(2) DADD.W #IMM16, R0

b7							b0	b7							b0	
0	1	1	1	1	1	0	1	1	1	1	0	1	1	0	0	#IMM16

[Number of Bytes/Number of Cycles]

Bytes/Cycles	4/5

DADD

(3) DADD.B R0H, R0L

b7							b0	b7							b0
0	1	1	1	1	1	0	0	1	1	1	0	0	1	0	0

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/5

DADD

(4) DADD.W R1, R0

b7							b0	b7							b0
0	1	1	1	1	1	0	1	1	1	1	0	0	1	0	0

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/5

DEC

(1) DEC.B dest

b7						b0
1	0	1	0	1	DEST	

(dest code: dsp8 / abs16)

dest		DEST
Rn	R0H	0 1 1
	R0L	1 0 0
dsp:8[SB/FB]	dsp:8[SB]	1 0 1
	dsp:8[FB]	1 1 0
abs16	abs16	1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	1/1	2/3	3/3

DEC

(2) DEC.W dest

b7							b0
1	1	1	1	DEST	0	1	0

dest	DEST
A0	0
A1	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	1/1

DIV

(1) DIV.size #IMM

b7							b0	b7							b0
0	1	1	1	1	1	0	SIZE	1	1	1	0	0	0	0	1

#IMM8
#IMM16

.size	SIZE
.B	0
.W	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	3/22

*1 If the size specifier (.size) is (.W), the number of bytes and cycles above are increased by 1 and 6, respectively.

*2 The number of cycles may decrease when an overflow occurs or depending on the value of the divisor or dividend.

DIV

(2) DIV.size src

b7							b0	b7							b0
0	1	1	1	0	1	1	SIZE	1	1	0	1	SRC			

src code: dsp8 / dsp16/abs16

.size	SIZE
.B	0
.W	1

src		SRC	src		SRC
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/22	2/22	2/24	3/24	3/24	4/24	4/24	4/24

*1 If the size specifier (.size) is (.W), the number of cycles above is increased by 6.

*2 The number of cycles may decrease when an overflow occurs or depending on the value of the divisor or dividend.

DIVU

(1) DIVU.size #IMM

b7							b0	b7							b0
0	1	1	1	1	1	0	SIZE	1	1	1	0	0	0	0	0

#IMM8

#IMM16

.size	SIZE
.B	0
.W	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	3/18

*2 The number of cycles may decrease when an overflow occurs or depending on the value of the divisor or dividend.

*3 If the size specifier (.size) is (.W), the number of bytes and cycles above are increased by 1 and 7, respectively.

DIVU

(2) DIVU.size src

b7							b0	b7							b0
0	1	1	1	0	1	1	SIZE	1	1	0	0	SRC			

src code: dsp8 / dsp16/abs16

.size	SIZE
.B	0
.W	1

src		SRC	src		SRC
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/18	2/18	2/20	3/20	3/20	4/20	4/20	4/20

*1 If the size specifier (.size) is (.W), the number of cycles above is increased by 7.

*2 The number of cycles may decrease when an overflow occurs or depending on the value of the divisor or dividend.

DIVX

(1) DIVX.size #IMM

b7							b0	b7							b0
0	1	1	1	1	1	0	SIZE	1	1	1	0	0	0	1	1

#IMM8

#IMM16

.size	SIZE
.B	0
.W	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	3/22

*2 The number of cycles may decrease when an overflow occurs or depending on the value of the divisor or dividend.

*3 If the size specifier (.size) is (.W), the number of bytes and cycles above are increased by 1 and 6, respectively.

DIVX

(2) DIVX.size src

b7							b0	b7							b0
0	1	1	1	0	1	1	SIZE	1	0	0	1	SRC			

src code: dsp8 / dsp16/abs16

.size	SIZE
.B	0
.W	1

src		SRC	src		SRC
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/22	2/22	2/24	3/24	3/24	4/24	4/24	4/24

*1 If the size specifier (.size) is (.W), the number of cycles above is increased by 6.

*2 The number of cycles may decrease when an overflow occurs or depending on the value of the divisor or dividend.

DSBB

(1) DSBB.B #IMM8, R0L

b7							b0	b7							b0
0	1	1	1	1	1	0	0	1	1	1	0	1	1	1	1

#IMM8

[Number of Bytes/Number of Cycles]

Bytes/Cycles	3/4

DSBB

(2) DSBB.W #IMM16, R0

b7							b0	b7							b0	
0	1	1	1	1	1	0	1	1	1	1	0	1	1	1	1	#IMM16

[Number of Bytes/Number of Cycles]

Bytes/Cycles	4/4

DSBB

(3) DSBB.B R0H, R0L

b7							b0	b7							b0
0	1	1	1	1	1	0	0	1	1	1	0	0	1	1	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/4

DSBB

(4) DSBB.W R1, R0

b7							b0	b7							b0
0	1	1	1	1	1	0	1	1	1	1	0	0	1	1	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/4

DSUB

(1) DSUB.B #IMM8, R0L

b7							b0	b7							b0	
0	1	1	1	1	1	0	0	1	1	1	0	1	1	0	1	#IMM8

[Number of Bytes/Number of Cycles]

Bytes/Cycles	3/4

DSUB

(2) DSUB.W #IMM16, R0

b7							b0	b7							b0	
0	1	1	1	1	1	0	1	1	1	1	0	1	1	0	1	#IMM16

[Number of Bytes/Number of Cycles]

Bytes/Cycles	4/4

DSUB

(3) DSUB.B R0H, R0L

b7							b0	b7							b0
0	1	1	1	1	1	0	0	1	1	1	0	0	1	0	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/4

DSUB

(4) DSUB.W R1, R0

b7							b0	b7							b0
0	1	1	1	1	1	0	1	1	1	1	0	0	1	0	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/4

ENTER

(1) ENTER #IMM8

b7							b0	b7							b0	
0	1	1	1	1	1	0	0	1	1	1	1	0	0	1	0	#IMM8

[Number of Bytes/Number of Cycles]

Bytes/Cycles	3/4

EXITD

(1) EXITD

b7							b0	b7							b0
0	1	1	1	1	1	0	1	1	1	1	1	0	0	1	0

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/9

EXTS

(1) EXTS.B dest

b7							b0	b7							b0
0	1	1	1	1	1	0	0	0	1	1	0	DEST			

(dest code: dsp8 / dsp16/abs16)

dest		DEST	dest		DEST
Rn	R0L	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	---	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	---	0 0 1 1		dsp:8[FB]	1 0 1 1
---	---	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	---	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

*1 Marked by **---** cannot be selected.

[Number of Bytes/Number of Cycles]

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

EXTS

(2) EXTS.W R0

b7							b0	b7							b0
0	1	1	1	1	1	0	0	1	1	1	1	0	0	1	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/3

FCLR

(1) FCLR dest

b7							b0	b7							b0
1	1	1	0	1	0	1	1	0	DEST			0	1	0	1

dest	DEST
C	0 0 0
D	0 0 1
Z	0 1 0
S	0 1 1
B	1 0 0
O	1 0 1
I	1 1 0
U	1 1 1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/2

FSET

(1) FSET dest

b7							b0	b7			b0
1	1	1	0	1	0	1	1	0	DEST	0 1 0 0	

dest	DEST
C	0 0 0
D	0 0 1
Z	0 1 0
S	0 1 1
B	1 0 0
O	1 0 1
I	1 1 0
U	1 1 1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/2

INC

(1) INC.B dest

b7						b0
1	0	1	0	0	DEST	

dest code

(dsp8 / abs16)

dest		DEST
Rn	R0H	0 1 1
	R0L	1 0 0
dsp:8[SB/FB]	dsp:8[SB]	1 0 1
	dsp:8[FB]	1 1 0
abs16	abs16	1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	1/1	2/3	3/3

INC

(2) INC.W dest

b7							b0
1	0	1	1	DEST	0	1	0

dest	DEST
A0	0
A1	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	1/1

INT

(1) INT #IMM

b7							b0
1	1	1	0	1	0	1	1

11	#IMM

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/19

INTO

(1) INTO

b7							b0
1	1	1	1	0	1	1	0

[Number of Bytes/Number of Cycles]

Bytes/Cycles	1/1

*1 If the O flag = 1, the number of cycles above is increased by 19.

J*Cnd*

(1) J*Cnd* label

b7						b0
0	1	1	0	1	CND	

label code
dsp8

dsp8 = address indicated by label – (start address of instruction + 1)

Cnd	CND	***Cnd***	CND
GEU/C	0 0 0	LTU/NC	1 0 0
GTU	0 0 1	LEU	1 0 1
EQ/Z	0 1 0	NE/NZ	1 1 0
N	0 1 1	PZ	1 1 1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/2

*2 If branched to label, the number of cycles above is increased by 2.

J*Cnd*

(2) J*Cnd* label

b7							b0	b7							b0	label code
0	1	1	1	1	1	0	1	1	1	0	0	CND				dsp8

dsp8 =address indicated by label – (start address of instruction + 2)

Cnd	CND	***Cnd***	CND
LE	1 0 0 0	GT	1 1 0 0
O	1 0 0 1	NO	1 1 0 1
GE	1 0 1 0	LT	1 1 1 0

[Number of Bytes/Number of Cycles]

Bytes/Cycles	3/2

*1 If branched to label, the number of cycles above is increased by 2.

JMP

(1) JMP.S label

b7							b0
0	1	1	0	0	dsp		

dsp = address indicated by label – (start address of instruction + 2)

[Number of Bytes/Number of Cycles]

Bytes/Cycles	1/5

JMP

(2) JMP.B label

b7							b0
1	1	1	1	1	1	1	0

label code

dsp8

dsp8 = address indicated by label – (start address of instruction + 1)

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/4

JMP

(3) JMP.W label

b7							b0
1	1	1	1	0	1	0	0

label code

dsp16

dsp16 = address indicated by label – (start address of instruction + 1)

[Number of Bytes/Number of Cycles]

Bytes/Cycles	3/4

JMP

(4) JMP.A label

b7							b0
1	1	1	1	1	1	0	0

label code

abs20

[Number of Bytes/Number of Cycles]

Bytes/Cycles	4/4

JMPI

(1) JMPI.W src

b7							b0	b7				b0
0	1	1	1	1	1	0	1	0	0	1	0	SRC

src code

(dsp8 / dsp16/abs16 / dsp20)

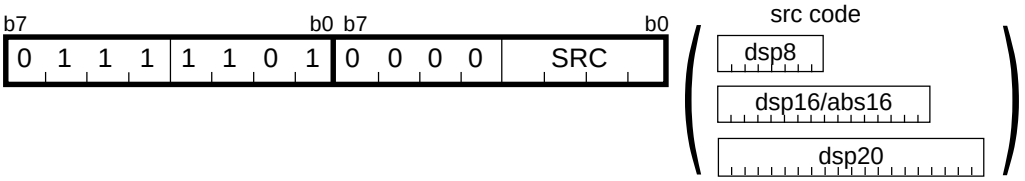
src		SRC	src		SRC
Rn	R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:20[An]	dsp:20[A0]	1 1 0 0
	A1	0 1 0 1		dsp:20[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:20[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/7	2/7	2/11	3/11	3/11	5/11	4/11	4/11

JMPI

(2) JMPI.A src



src		SRC	src		SRC
Rn	R2R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R3R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	---	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	---	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A1A0	0 1 0 0	dsp:20[An]	dsp:20[A0]	1 1 0 0
	---	0 1 0 1		dsp:20[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

*1 Marked by **---** cannot be selected.

[Number of Bytes/Number of Cycles]

src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:20[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/6	2/6	2/10	3/10	3/10	5/10	4/10	4/10

JSR

(1) JSR.W label

b7							b0
1	1	1	1	0	1	0	1

label code: dsp16

dsp16 = address indicated by label – (start address of instruction + 1)

[Number of Bytes/Number of Cycles]

Bytes/Cycles	3/8

JSR

(2) JSR.A label

b7							b0
1	1	1	1	1	1	0	1

label code: abs20

[Number of Bytes/Number of Cycles]

Bytes/Cycles	4/9

JSRI

(1) JSRI.W src

b7							b0	b7				b0
0	1	1	1	1	1	0	1	0	0	1	1	SRC

src code
- dsp8
- dsp16/abs16
- dsp20

src		SRC	src		SRC
Rn	R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:20[An]	dsp:20[A0]	1 1 0 0
	A1	0 1 0 1		dsp:20[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:20[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/11	2/11	2/15	3/15	3/15	5/15	4/15	4/15

JSRI

(2) JSRI.A src

b7							b0	b7				b0
0	1	1	1	1	1	0	1	0	0	0	1	SRC

src code
- dsp8
- dsp16/abs16
- dsp20

src		SRC	src		SRC
Rn	R2R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R3R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	---	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	---	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A1A0	0 1 0 0	dsp:20[An]	dsp:20[A0]	1 1 0 0
	---	0 1 0 1		dsp:20[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

*1 Marked by **---** cannot be selected.

[Number of Bytes/Number of Cycles]

src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:20[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/11	2/11	2/15	3/15	3/15	5/15	4/15	4/15

LDC

(1) LDC #IMM16, dest

b7							b0	b7			b0				
1	1	1	0	1	0	1	1	0	DEST			0	0	0	0

#IMM16

dest	DEST
---	0 0 0
INTBL	0 0 1
INTBH	0 1 0
FLG	0 1 1
ISP	1 0 0
SP	1 0 1
SB	1 1 0
FB	1 1 1

*1 Marked by **---** cannot be selected.

[Number of Bytes/Number of Cycles]

Bytes/Cycles	4/2

LDC

(2) LDC src, dest

b7							b0	b7			b0
0	1	1	1	1	0	1	0	1	DEST	SRC	

src code
(dsp8 / dsp16/abs16)

src		SRC	src		SRC
Rn	R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

dest	DEST
---	0 0 0
INTBL	0 0 1
INTBH	0 1 0
FLG	0 1 1
ISP	1 0 0
SP	1 0 1
SB	1 1 0
FB	1 1 1

*1 Marked by **---** cannot be selected.

[Number of Bytes/Number of Cycles]

src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/1	2/1	2/3	3/3	3/3	4/3	4/3	4/3

LDCTX

(1) LDCTX abs16, abs20

b7							b0	b7							b0		
0	1	1	1	1	1	0	0	1	1	1	1	0	0	0	0	abs16	abs20

[Number of Bytes/Number of Cycles]

Bytes/Cycles	7/11+2×m

*2 m denotes the number of transfers performed.

LDE

(1) LDE.size abs20, dest

b7							b0	b7				b0
0	1	1	1	0	1	0	SIZE	1	0	0	0	DEST

dest code: dsp8 / dsp16/abs16

src code: abs20

.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

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

LDE

(2) LDE.size dsp:20[A0], dest

b7							b0	b7				b0
0	1	1	1	0	1	0	SIZE	1	0	0	1	DEST

dest code: dsp8 / dsp16/abs16

src code: dsp20

.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

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

LDE

(3) LDE.size [A1A0], dest

b7							b0	b7				b0
0	1	1	1	0	1	0	SIZE	1	0	1	0	DEST

dest code: dsp8 / dsp16/abs16

.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

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

LDINTB

(1) LDINTB #IMM

b7							b0	b7							b0
1	1	1	0	1	0	1	1	0	0	1	0	0	0	0	0
0	0	0	0	#IMM1				0	0	0	0	0	0	0	0
1	1	1	0	1	0	1	1	0	0	0	1	0	0	0	0
#IMM2															

*1 #IMM1 indicates the 4 high-order bits of #IMM.
#IMM2 indicates the 16 low-order bits of #IMM.

[Number of Bytes/Number of Cycles]

Bytes/Cycles	8/4

LDIPL

(1) LDIPL #IMM

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

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/2

MOV

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

b7							b0	b7				b0
0	1	1	1	0	1	0	SIZE	1	1	0	0	DEST

dest code

(dsp8 / dsp16/abs16) #IMM8 / #IMM16

.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

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

*1 If the size specifier (.size) is (.W), the number of bytes above is increased by 1.

MOV

(2) MOV.size:Q #IMM, dest

b7							b0	b7							b0
1	1	0	1	1	0	0	SIZE	IMM4				DEST			

dest code

(dsp8 / dsp16/abs16)

.size	SIZE
.B	0
.W	1

#IMM	IMM4	#IMM	IMM4
0	0 0 0 0	–8	1 0 0 0
+1	0 0 0 1	–7	1 0 0 1
+2	0 0 1 0	–6	1 0 1 0
+3	0 0 1 1	–5	1 0 1 1
+4	0 1 0 0	–4	1 1 0 0
+5	0 1 0 1	–3	1 1 0 1
+6	0 1 1 0	–2	1 1 1 0
+7	0 1 1 1	–1	1 1 1 1

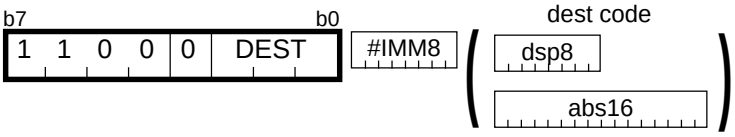
dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

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

MOV

(3) MOV.B:S #IMM8, dest



dest		DEST
Rn	R0H	0 1 1
	R0L	1 0 0
dsp:8[SB/FB]	dsp:8[SB]	1 0 1
	dsp:8[FB]	1 1 0
abs16	abs16	1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	2/1	3/2	4/2

MOV

(4) MOV.size:S #IMM, dest

b7							b0
1	SIZE	1	0	DEST	0	1	0

#IMM8

#IMM16

.size	SIZE
.B	1
.W	0

dest	DEST
A0	0
A1	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/1

*1 If the size specifier (.size) is (.W), the number of bytes and cycles above are increased by 1 and 1, respectively.

MOV

(5) MOV.B:Z #0, dest

b7					b0
1	0	1	1	0	DEST

dest code

(dsp8 / abs16)

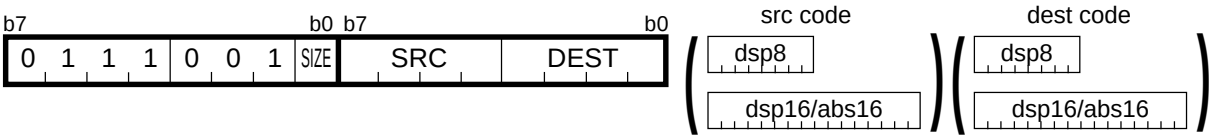
dest		DEST
Rn	R0H	0 1 1
	R0L	1 0 0
dsp:8[SB/FB]	dsp:8[SB]	1 0 1
	dsp:8[FB]	1 1 0
abs16	abs16	1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	1/1	2/2	3/2

MOV

(6) MOV.size:G src, dest



.size	SIZE
.B	0
.W	1

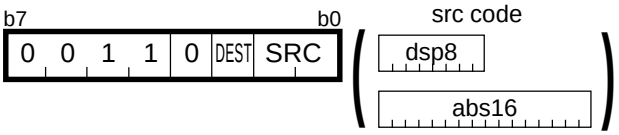
src/dest		SRC/DEST	src/dest		SRC/DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Rn	2/2	2/2	2/2	3/2	3/2	4/2	4/2	4/2
An	2/2	2/2	2/2	3/2	3/2	4/2	4/2	4/2
[An]	2/3	2/3	2/3	3/3	3/3	4/3	4/3	4/3
dsp:8[An]	3/3	3/3	3/3	4/3	4/3	5/3	5/3	5/3
dsp:8[SB/FB]	3/3	3/3	3/3	4/3	4/3	5/3	5/3	5/3
dsp:16[An]	4/3	4/3	4/3	5/3	5/3	6/3	6/3	6/3
dsp:16[SB]	4/3	4/3	4/3	5/3	5/3	6/3	6/3	6/3
abs16	4/3	4/3	4/3	5/3	5/3	6/3	6/3	6/3

MOV

(7) MOV.B:S src, dest



src		SRC
Rn	R0L/R0H	0 0
dsp:8[SB/FB]	dsp:8[SB]	0 1
	dsp:8[FB]	1 0
abs16	abs16	1 1

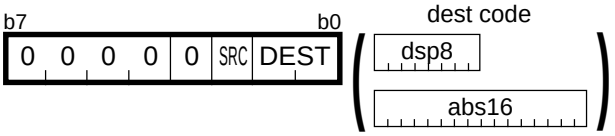
dest	DEST
A0	0
A1	1

[Number of Bytes/Number of Cycles]

src	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	1/2	2/3	3/3

MOV

(8) MOV.B:S R0L/R0H, dest



src	SRC
R0L	0
R0H	1

dest		DEST
dsp:8[SB/FB]	dsp:8[SB]	0 1
	dsp:8[FB]	1 0
abs16	abs16	1 1

[Number of Bytes/Number of Cycles]

dest	dsp:8[SB/FB]	abs16
Bytes/Cycles	2/2	3/2

MOV

(9) MOV.B:S src, R0L/R0H

b7						b0
0	0	0	0	1	DEST	SRC

src code: dsp8 / abs16

src		SRC
Rn	R0L/R0H	0 0
dsp:8[SB/FB]	dsp:8[SB]	0 1
	dsp:8[FB]	1 0
abs16	abs16	1 1

dest	DEST
R0L	0
R0H	1

[Number of Bytes/Number of Cycles]

src	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	1/2	2/3	3/3

MOV

(10) MOV.size:G dsp:8[SP], dest

b7							b0	b7					b0
0	1	1	1	0	1	0	SIZE	1	0	1	1	DEST	

dest code: dsp8 / dsp16/abs16

src code: dsp8

.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

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

MOV

(11) MOV.size:G src, dsp:8[SP]

b7							b0	b7				b0
0	1	1	1	0	1	0	SIZE	0	0	1	1	SRC

src code: dsp8 / dsp16/abs16

dest code: dsp8

.size	SIZE
.B	0
.W	1

src		SRC	src		SRC
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

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

MOVA

(1) MOVA src, dest

b7							b0	b7			b0
1	1	1	0	1	0	1	1	0	DEST	SRC	

src code: dsp8 / dsp16

src		SRC
dsp:8[An]	dsp:8[A0]	1 0 0 0
	dsp:8[A1]	1 0 0 1
dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	dsp:8[FB]	1 0 1 1
dsp:16[An]	dsp:16[A0]	1 1 0 0
	dsp:16[A1]	1 1 0 1
dsp:16[SB]	dsp:16[SB]	1 1 1 0
abs16	abs16	1 1 1 1

dest	DEST
R0	0 0 0
R1	0 0 1
R2	0 1 0
R3	0 1 1
A0	1 0 0
A1	1 0 1

[Number of Bytes/Number of Cycles]

src	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	3/2	3/2	4/2	4/2	4/2

MOV*Dir*

(1) MOV*Dir* R0L, dest

b7							b0	b7							b0
0	1	1	1	1	1	0	0	1	0	DIR		DEST			

dest code

(dsp8 / dsp16/abs16)

Dir	DIR
LL	0 0
LH	1 0
HL	0 1
HH	1 1

dest		DEST	dest		DEST
Rn	---	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H	0 0 1 1		dsp:8[FB]	1 0 1 1
An	---	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	---	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

*1 Marked by - - - cannot be selected.

[Number of Bytes/Number of Cycles]

dest	Rn	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
MOVHH, MOVLL	2/4	2/5	3/5	3/5	4/5	4/5	4/5
MOVHL, MOVLH	2/7	2/8	3/8	3/8	4/8	4/8	4/8

MOV*Dir*

(2) MOV*Dir* src, R0L

b7							b0	b7							b0
0	1	1	1	1	1	0	0	0	0	DIR		SRC			

dest code

(dsp8 / dsp16/abs16)

Dir	DIR
LL	0 0
LH	1 0
HL	0 1
HH	1 1

src		SRC	src		SRC
Rn	R0L	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H	0 0 1 1		dsp:8[FB]	1 0 1 1
An	---	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	---	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

*1 Marked by - - - cannot be selected.

[Number of Bytes/Number of Cycles]

src	Rn	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
MOVHH, MOVLL	2/3	2/5	3/5	3/5	4/5	4/5	4/5
MOVHL, MOVLH	2/6	2/8	3/8	3/8	4/8	4/8	4/8

MUL

(1) MUL.size #IMM, dest

b7							b0	b7				b0
0	1	1	1	1	1	0	SIZE	0	1	0	1	DEST

dest code

(dsp8 / dsp16/abs16) #IMM8 / #IMM16

.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	--- /R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/---	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	---	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	---	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

*1 Marked by - - - cannot be selected.

[Number of Bytes/Number of Cycles]

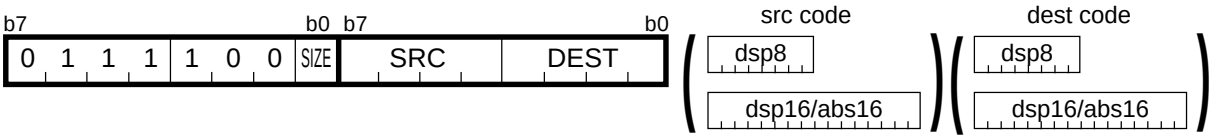
dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	3/4	3/4	3/5	4/5	4/5	5/5	5/5	5/5

*2 If dest is Rn or An while the size specifier (.size) is (.W), the number of bytes and cycles above are increased by 1 each.

*3 If dest is neither Rn nor An while the size specifier (.size) is (.W), the number of bytes and cycles above are increased by 1 and 2, respectively.

MUL

(2) MUL.size src, dest



.size	SIZE
.B	0
.W	1

src		SRC	src		SRC
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	--- /R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/---	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	---	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	---	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

*1 Marked by - - - cannot be selected.

[Number of Bytes/Number of Cycles]

src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Rn	2/4	2/4	2/5	3/5	3/5	4/5	4/5	4/5
An	2/4	2/5	2/5	3/5	3/5	4/5	4/5	4/5
[An]	2/6	2/6	2/6	3/6	3/6	4/6	4/6	4/6
dsp:8[An]	3/6	3/6	3/6	4/6	4/6	5/6	5/6	5/6
dsp:8[SB/FB]	3/6	3/6	3/6	4/6	4/6	5/6	5/6	5/6
dsp:16[An]	4/6	4/6	4/6	5/6	5/6	6/6	6/6	6/6
dsp:16[SB]	4/6	4/6	4/6	5/6	5/6	6/6	6/6	6/6
abs16	4/6	4/6	4/6	5/6	5/6	6/6	6/6	6/6

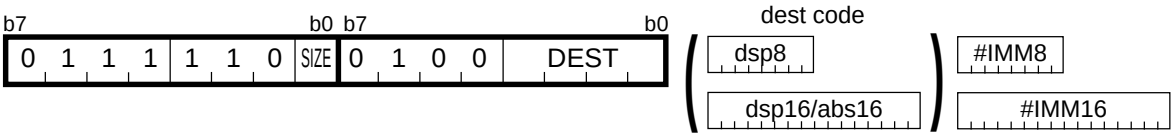
*2 If src is An and dest is Rn while the size specifier (.size) is (.W), the number of cycles above is increased by 1.

*3 If src is not An and dest is Rn or An while the size specifier (.size) is (.W), the number of cycles above is increased by 1.

*4 If dest is neither Rn nor An while the size specifier (.size) is (.W), the number of cycles above is increased by 2.

MULU

(1) MULU.size #IMM, dest



.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	--- /R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/---	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	---	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	---	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

*1 Marked by - - - cannot be selected.

[Numbera of Bytes/Number of Cycles]

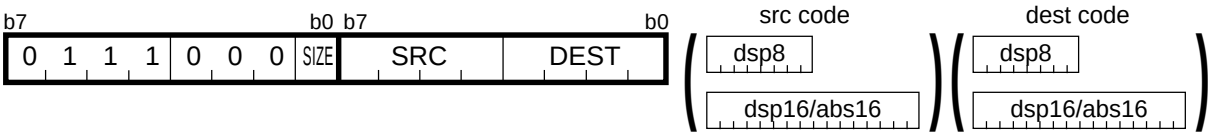
dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	3/4	3/4	3/5	4/5	4/5	5/5	5/5	5/5

*2 If dest is Rn or An while the size specifier (.size) is (.W), the number of bytes and cycles above are increased by 1 each.

*3 If dest is neither Rn nor An while the size specifier (.size) is (.W), the number of bytes and cycles above are increased by 1 and 2, respectively.

MULU

(2) MULU.size src, dest



.size	SIZE
.B	0
.W	1

src		SRC	src		SRC
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	--- /R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/---	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	---	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	---	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

*1 Marked by - - - cannot be selected.

[Number of Bytes/Number of Cycles]

src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Rn	2/4	2/4	2/5	3/5	3/5	4/5	4/5	4/5
An	2/4	2/5	2/5	3/5	3/5	4/5	4/5	4/5
[An]	2/6	2/6	2/6	3/6	3/6	4/6	4/6	4/6
dsp:8[An]	3/6	3/6	3/6	4/6	4/6	5/6	5/6	5/6
dsp:8[SB/FB]	3/6	3/6	3/6	4/6	4/6	5/6	5/6	5/6
dsp:16[An]	4/6	4/6	4/6	5/6	5/6	6/6	6/6	6/6
dsp:16[SB]	4/6	4/6	4/6	5/6	5/6	6/6	6/6	6/6
abs16	4/6	4/6	4/6	5/6	5/6	6/6	6/6	6/6

*2 If src is An and dest is Rn while the size specifier (.size) is (.W), the number of cycles above is increased by 1.

*3 If src is not An and dest is Rn or An while the size specifier (.size) is (.W), the number of cycles above is increased by 1.

*4 If dest is neither Rn nor An while the size specifier (.size) is (.W), the number of cycles above is increased by 2.

NEG

(1) NEG.size dest

b7							b0	b7							b0
0	1	1	1	0	1	0	SIZE	0	1	0	1	DEST			

dest code
(dsp8 / dsp16/abs16)

.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

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

NOP

(1) NOP

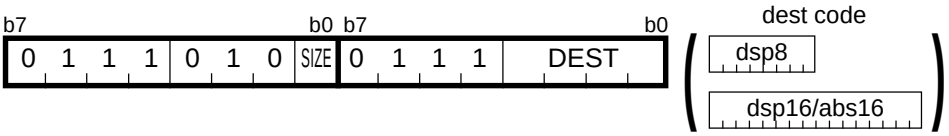
b7							b0
0	0	0	0	0	1	0	0

[Number of Bytes/Number of Cycles]

Bytes/Cycles	1/1

NOT

(1) NOT.size:G dest



.size	SIZE
.B	0
.W	1

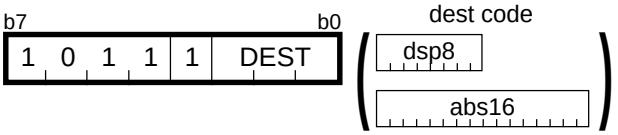
dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

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

NOT

(2) NOT.B:S dest



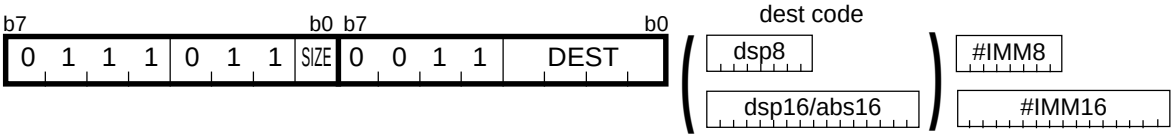
dest		DEST
Rn	R0H	0 1 1
	R0L	1 0 0
dsp:8[SB/FB]	dsp:8[SB]	1 0 1
	dsp:8[FB]	1 1 0
abs16	abs16	1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	1/1	2/3	3/3

OR

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



.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

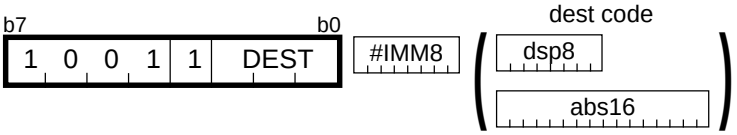
[Number of Bytes/Number of Cycles]

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

*1 If the size specifier (.size) is (.W), the number of bytes above is increased by 1.

OR

(2) OR.B:S #IMM8, dest



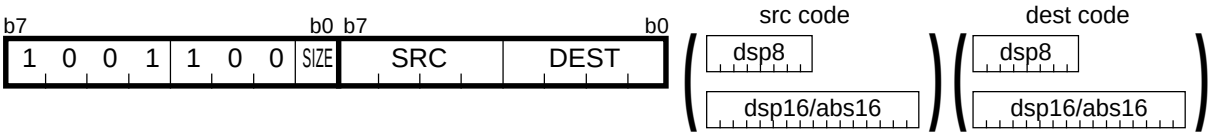
dest		DEST
Rn	R0H	0 1 1
	R0L	1 0 0
dsp:8[SB/FB]	dsp:8[SB]	1 0 1
	dsp:8[FB]	1 1 0
abs16	abs16	1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	2/1	3/3	4/3

OR

(3) OR.size:G src, dest



.size	SIZE
.B	0
.W	1

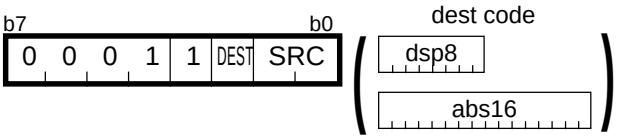
src/dest		SRC/DEST	src/dest		SRC/DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Rn	2/2	2/2	2/3	3/3	3/3	4/3	4/3	4/3
An	2/2	2/2	2/3	3/3	3/3	4/3	4/3	4/3
[An]	2/3	2/3	2/4	3/4	3/4	4/4	4/4	4/4
dsp:8[An]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	5/4
dsp:8[SB/FB]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	5/4
dsp:16[An]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4
dsp:16[SB]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4
abs16	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4

OR

(4) OR.B:S src, R0L/R0H



src		SRC
Rn	R0L/R0H	0 0
dsp:8[SB/FB]	dsp:8[SB]	0 1
	dsp:8[FB]	1 0
abs16	abs16	1 1

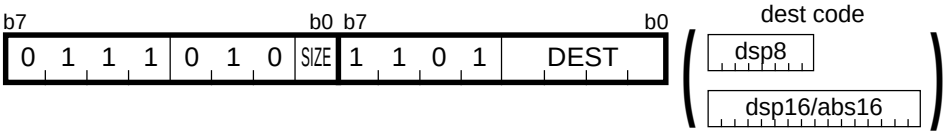
dest	DEST
R0L	0
R0H	1

[Number of Bytes/Number of Cycles]

src	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	1/2	2/3	3/3

POP

(1) POP.size:G dest



.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

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

POP

(2) POP.B:S dest

b7							b0
1	0	0	1	DEST	0	1	0

dest	DEST
R0L	0
R0H	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	1/3

POP

(3) POP.W:S dest

b7							b0
1	1	0	1	DEST	0	1	0

dest	DEST
A0	0
A1	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	1/3

POPC

(1) POPC dest

b7							b0	b7			b0
1	1	1	0	1	0	1	1	0	DEST	0 0 1 1	

dest	DEST	dest	DEST
---	0 0 0	ISP	1 0 0
INTBL	0 0 1	SP	1 0 1
INTBH	0 1 0	SB	1 1 0
FLG	0 1 1	FB	1 1 1

*1 Marked by - - - cannot be selected.

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/3

POPM

(1) POPM dest

b7							b0
1	1	1	0	1	1	0	1

DEST

dest							
FB	SB	A1	A0	R3	R2	R1	R0
DEST*2							

*2 The bit for a selected register is 1.
The bit for a non-selected register is 0.

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/3

*3 If two or more registers need to be restored, the number of required cycles is 2 x m (m: number of registers to be restored).

PUSH

(1) PUSH.size:G #IMM

b7							b0	b7							b0
0	1	1	1	1	1	0	SIZE	1	1	1	0	0	0	1	0

#IMM8

#IMM16

.size	SIZE
.B	0
.W	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	3/2

*1 If the size specifier (.size) is (.W), the number of bytes above is increased by 1.

PUSH

(2) PUSH.size:G src

b7							b0	b7					b0
0	1	1	1	0	1	0	SIZE	0	1	0	0	SRC	

src code

(dsp8 / dsp16/abs16)

.size	SIZE
.B	0
.W	1

src		SRC	src		SRC
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/2	2/2	2/4	3/4	3/4	4/4	4/4	4/4

PUSH

(3) PUSH.B:S src

b7							b0
1	0	0	0	SRC	0	1	0

src	SRC
R0L	0
R0H	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	1/2

PUSH

(4) PUSH.W:S src

b7							b0
1	1	0	0	SRC	0	1	0

src	SRC
A0	0
A1	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	1/2

PUSHA

(1) PUSHA src

b7							b0	b7							b0
0	1	1	1	1	1	0	1	1	0	0	1	SRC			

src code
(dsp8 / dsp16/abs16)

src		SRC
dsp:8[An]	dsp:8[A0]	1 0 0 0
	dsp:8[A1]	1 0 0 1
dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	dsp:8[FB]	1 0 1 1
dsp:16[An]	dsp:16[A0]	1 1 0 0
	dsp:16[A1]	1 1 0 1
dsp:16[SB]	dsp:16[SB]	1 1 1 0
abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

src	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs:16
Bytes/Cycles	3/2	3/2	4/2	4/2	4/2

PUSHC

(1) PUSHC src

b7							b0	b7							b0
1	1	1	0	1	0	1	1	0	SRC			0	0	1	0

src	SRC	src	SRC
---	0 0 0	ISP	1 0 0
INTBL	0 0 1	SP	1 0 1
INTBH	0 1 0	SB	1 1 0
FLG	0 1 1	FB	1 1 1

*1 Marked by - - - cannot be selected.

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/2

PUSHM

(1) PUSHM src

b7							b0	
1	1	1	0	1	1	0	0	SRC

src							
R0	R1	R2	R3	A0	A1	SB	FB
SRC*1							

*1 The bit for a selected register is 1.
The bit for a non-selected register is 0.

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/2×m

*2 m denotes the number of registers to be saved.

REIT

(1) REIT

b7							b0
1	1	1	1	1	0	1	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	1/6

RMPA

(1) RMPA.size

b7							b0	b7							b0
0	1	1	1	1	1	0	SIZE	1	1	1	1	0	0	0	1

.size	SIZE
.B	0
.W	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/4+7×m

*1 m denotes the number of operation performed.

*2 If the size specifier (.size) is (.W), the number of cycles is (6+9×m).

ROLC

(1) ROLC.size dest

b7							b0	b7							b0
0	1	1	1	0	1	1	SIZE	1	0	1	0	DEST			

dest code

(dsp8 / dsp16/abs16)

.size	SIZE
.B	0
.W	1

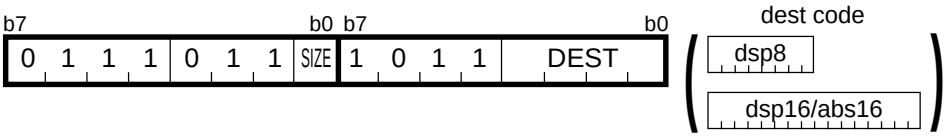
dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

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

RORC

(1) RORC.size dest



.size	SIZE
.B	0
.W	1

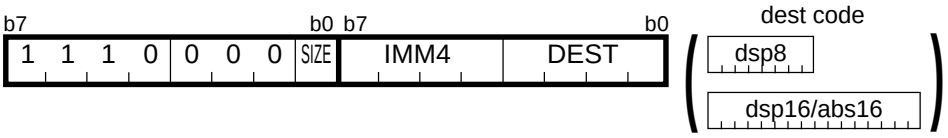
dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

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

ROT

(1) ROT.size #IMM, dest



.size	SIZE
.B	0
.W	1

#IMM	IMM4	#IMM	IMM4
+1	0 0 0 0	–1	1 0 0 0
+2	0 0 0 1	–2	1 0 0 1
+3	0 0 1 0	–3	1 0 1 0
+4	0 0 1 1	–4	1 0 1 1
+5	0 1 0 0	–5	1 1 0 0
+6	0 1 0 1	–6	1 1 0 1
+7	0 1 1 0	–7	1 1 1 0
+8	0 1 1 1	–8	1 1 1 1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/1+m	2/1+m	2/2+m	3/2+m	3/2+m	4/2+m	4/2+m	4/2+m

*1 m denotes the number of rotates performed.

ROT

(2) ROT.size R1H, dest

b7							b0	b7							b0
0	1	1	1	0	1	0	SIZE	0	1	1	0	DEST			

dest code: dsp8 / dsp16/abs16

.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/---	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	--- /R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

*1 Marked by - - - cannot be selected.

[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/2+m	2/2+m	2/3+m	3/3+m	3/3+m	4/3+m	4/3+m	4/3+m

*2 m denotes the number of rotates performed.

RTS

(1) RTS

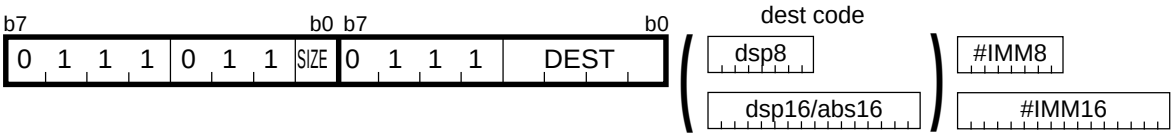
b7							b0
1	1	1	1	0	0	1	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	1/6

SBB

(1) SBB.size #IMM, dest



.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

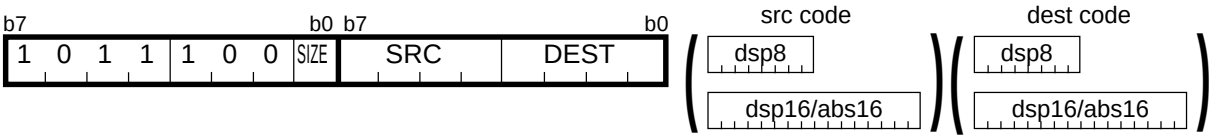
[Number of Bytes/Number of Cycles]

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

*1 If the size specifier (.size) is (.W), the number of bytes above is increased by 1.

SBB

(2) SBB.size src, dest



.size	SIZE
.B	0
.W	1

src/dest		SRC/DEST	src/dest		SRC/DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Rn	2/2	2/2	2/3	3/3	3/3	4/3	4/3	4/3
An	2/2	2/2	2/3	3/3	3/3	4/3	4/3	4/3
[An]	2/3	2/3	2/4	3/4	3/4	4/4	4/4	4/4
dsp:8[An]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	5/4
dsp:8[SB/FB]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	5/4
dsp:16[An]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4
dsp:16[SB]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4
abs16	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4

SBJNZ

(1) SBJNZ.size #IMM, dest, label

dsp8(label code) = address indicated by label – (start address of instruction + 2)

.size	SIZE
.B	0
.W	1

#IMM	IMM4	#IMM	IMM4
0	0 0 0 0	+8	1 0 0 0
–1	0 0 0 1	+7	1 0 0 1
–2	0 0 1 0	+6	1 0 1 0
–3	0 0 1 1	+5	1 0 1 1
–4	0 1 0 0	+4	1 1 0 0
–5	0 1 0 1	+3	1 1 0 1
–6	0 1 1 0	+2	1 1 1 0
–7	0 1 1 1	+1	1 1 1 1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

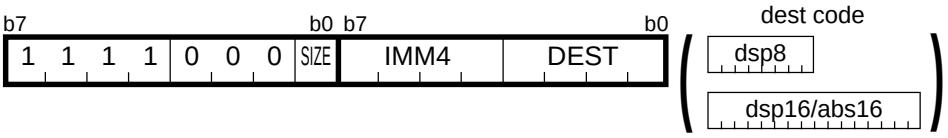
[Number of Bytes/Number of Cycles]

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

*1 If branched to label, the number of cycles above is increased by 4.

SHA

(1) SHA.size #IMM, dest



.size	SIZE
.B	0
.W	1

#IMM	IMM4	#IMM	IMM4
+1	0 0 0 0	–1	1 0 0 0
+2	0 0 0 1	–2	1 0 0 1
+3	0 0 1 0	–3	1 0 1 0
+4	0 0 1 1	–4	1 0 1 1
+5	0 1 0 0	–5	1 1 0 0
+6	0 1 0 1	–6	1 1 0 1
+7	0 1 1 0	–7	1 1 1 0
+8	0 1 1 1	–8	1 1 1 1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

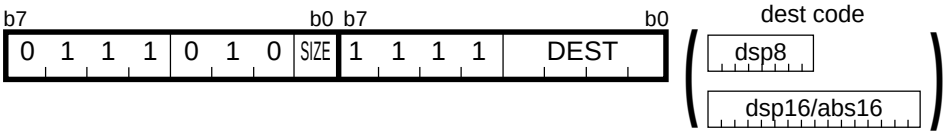
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/1+m	2/1+m	2/2+m	3/2+m	3/2+m	4/2+m	4/2+m	4/2+m

*1 m denotes the number of shifts performed.

SHA

(2) SHA.size R1H, dest



.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/---	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	--- /R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

*1 Marked by - - - cannot be selected.

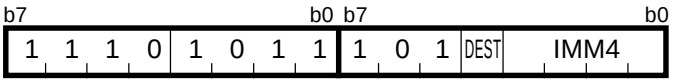
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/2+m	2/2+m	2/3+m	3/3+m	3/3+m	4/3+m	4/3+m	4/3+m

*2 m denotes the number of shifts performed.

SHA

(3) SHA.L #IMM, dest



#IMM	IMM4	#IMM	IMM4
+1	0 0 0 0	–1	1 0 0 0
+2	0 0 0 1	–2	1 0 0 1
+3	0 0 1 0	–3	1 0 1 0
+4	0 0 1 1	–4	1 0 1 1
+5	0 1 0 0	–5	1 1 0 0
+6	0 1 0 1	–6	1 1 0 1
+7	0 1 1 0	–7	1 1 1 0
+8	0 1 1 1	–8	1 1 1 1

dest	DEST
R2R0	0
R3R1	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/3+m

*2 m denotes the number of shifts performed.

SHA

(4) SHA.L R1H, dest

b7							b0	b7							b0
1	1	1	0	1	0	1	1	0	0	1	DEST	0	0	0	1

dest	DEST
R2R0	0
R3R1	1

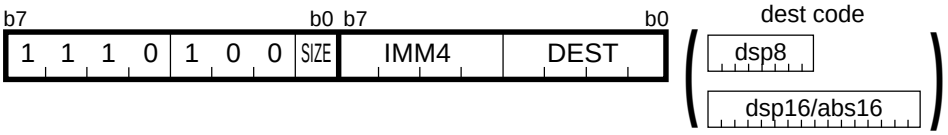
[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/4+m

*1 m denotes the number of shifts performed.

SHL

(1) SHL.size #IMM, dest



.size	SIZE
.B	0
.W	1

#IMM	IMM4	#IMM	IMM4
+1	0 0 0 0	–1	1 0 0 0
+2	0 0 0 1	–2	1 0 0 1
+3	0 0 1 0	–3	1 0 1 0
+4	0 0 1 1	–4	1 0 1 1
+5	0 1 0 0	–5	1 1 0 0
+6	0 1 0 1	–6	1 1 0 1
+7	0 1 1 0	–7	1 1 1 0
+8	0 1 1 1	–8	1 1 1 1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/1+m	2/1+m	2/2+m	3/2+m	3/2+m	4/2+m	4/2+m	4/2+m

*1 m denotes the number of shifts performed.

SHL

(2) SHL.size R1H, dest

b7							b0	b7				b0
0	1	1	1	0	1	0	SIZE	1	1	1	0	DEST

dest code: dsp8 / dsp16/abs16

.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/---	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	--- /R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

*1 Marked by - - - cannot be selected.

[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/2+m	2/2+m	2/3+m	3/3+m	3/3+m	4/3+m	4/3+m	4/3+m

*2 m denotes the number of shifts performed.

SHL

(3) SHL.L #IMM, dest

b7							b0	b7				b0
1	1	1	0	1	0	1	1	1	0	0	DEST	IMM4

#IMM	IMM4	#IMM	IMM4
+1	0 0 0 0	–1	1 0 0 0
+2	0 0 0 1	–2	1 0 0 1
+3	0 0 1 0	–3	1 0 1 0
+4	0 0 1 1	–4	1 0 1 1
+5	0 1 0 0	–5	1 1 0 0
+6	0 1 0 1	–6	1 1 0 1
+7	0 1 1 0	–7	1 1 1 0
+8	0 1 1 1	–8	1 1 1 1

dest	DEST
R2R0	0
R3R1	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/3+m

*2 m denotes the number of shifts performed.

SHL

(4) SHL.L R1H, dest

b7							b0	b7							b0
1	1	1	0	1	0	1	1	0	0	0	DEST	0	0	0	1

dest	DEST
R2R0	0
R3R1	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/4+m

*1 m denotes the number of shifts performed.

SMOVB

(1) SMOVB.size

b7							b0	b7							b0
0	1	1	1	1	1	0	SIZE	1	1	1	0	1	0	0	1

.size	SIZE
.B	0
.W	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/5+5×m

*2 m denotes the number of transfers performed.

SMOVF

(1) SMOVF.size

b7							b0	b7							b0
0	1	1	1	1	1	0	SIZE	1	1	1	0	1	0	0	0

.size	SIZE
.B	0
.W	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/5+5×m

*1 m denotes the number of transfers performed.

SSTR

(1) SSTR.size

b7							b0	b7							b0
0	1	1	1	1	1	0	SIZE	1	1	1	0	1	0	1	0

.size	SIZE
.B	0
.W	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/3+2×m

*1 m denotes the number of transfers performed.

STC

(1) STC src, dest

b7							b0	b7		b0
0	1	1	1	1	0	1	1	1	SRC	DEST

dest Code: dsp8 / dsp16/abs16

src	SRC
---	0 0 0
INTBL	0 0 1
INTBH	0 1 0
FLG	0 1 1
ISP	1 0 0
SP	1 0 1
SB	1 1 0
FB	1 1 1

dest		DEST	dest		DEST
Rn	R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

*1 Marked by - - - cannot be selected.

[Number of Bytes/Number of Cycles]

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

STC

(2) STC PC, dest

b7							b0	b7			b0
0	1	1	1	1	1	0	0	1	1	0	0	DEST

dest Code: dsp8 / dsp16/abs16

dest		DEST	dest		DEST
Rn	R2R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R3R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	---	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	---	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A1A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	---	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

*1 Marked by - - - cannot be selected.

[Number of Bytes/Number of Cycles]

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

STCTX

(1) STCTX abs16, abs20

b7							b0	b7							b0		
0	1	1	1	1	1	0	1	1	1	1	1	0	0	0	0	abs16	abs20

[Number of Bytes/Number of Cycles]

Bytes/Cycles	7/11+2×m

*1 m denotes the number of transfers performed.

STE

(1) STE.size src, abs20

b7							b0	b7					b0	src code	dest code
0	1	1	1	0	1	0	SIZE	0	0	0	0	SRC		dsp8 / dsp16/abs16	abs20

.size	SIZE
.B	0
.W	1

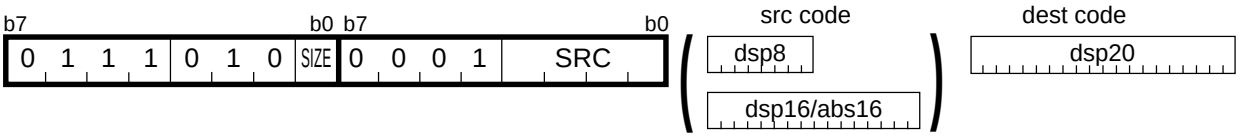
src		SRC	src		SRC
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

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

STE

(2) STE.size src, dsp:20[A0]



.size	SIZE
.B	0
.W	1

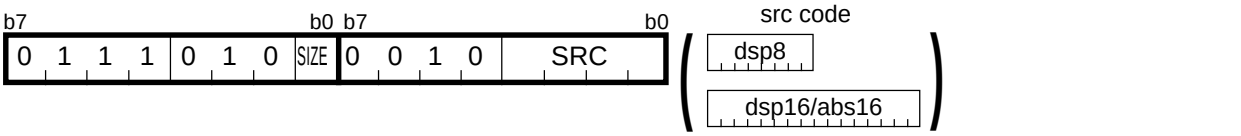
src		SRC	src		SRC
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

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

STE

(3) STE.size src, [A1A0]



.size	SIZE
.B	0
.W	1

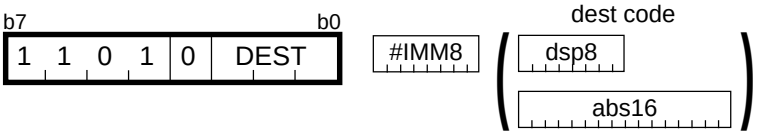
src		SRC	src		SRC
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/3	2/3	2/4	3/4	3/4	4/4	4/4	4/4

STNZ

(1) STNZ #IMM8, dest



dest		DEST
Rn	R0H	0 1 1
	R0L	1 0 0
dsp:8[SB/FB]	dsp:8[SB]	1 0 1
	dsp:8[FB]	1 1 0
abs16	abs16	1 1 1

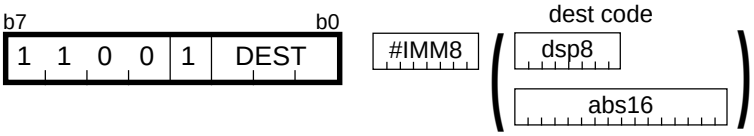
[Number of Bytes/Number of Cycles]

dest	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	2/1	3/2	4/2

*1 If the Z flag = 0, the number of cycles above is increased by 1.

STZ

(1) STZ #IMM8, dest



dest		DEST
Rn	R0H	0 1 1
	R0L	1 0 0
dsp:8[SB/FB]	dsp:8[SB]	1 0 1
	dsp:8[FB]	1 1 0
abs16	abs16	1 1 1

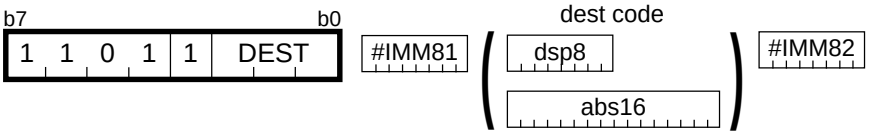
[Number of Bytes/Number of Cycles]

dest	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	2/1	3/2	4/2

*2 If the Z flag = 1, the number of cycles above is increased by 1.

STZX

(1) STZX #IMM81, #IMM82, dest



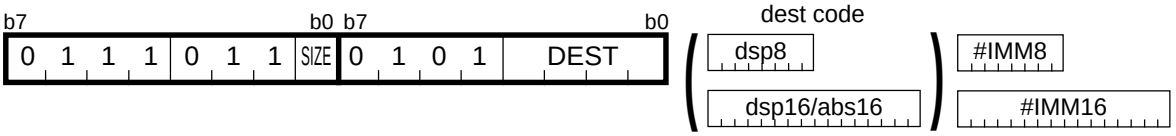
dest		DEST
Rn	R0H	0 1 1
	R0L	1 0 0
dsp:8[SB/FB]	dsp:8[SB]	1 0 1
	dsp:8[FB]	1 1 0
abs16	abs16	1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	3/2	4/3	5/3

SUB

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



.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

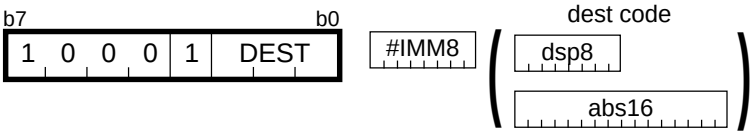
[Number of Bytes/Number of Cycles]

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

*1 If the size specifier (.size) is (.W), the number of bytes above is increased by 1.

SUB

(2) SUB.B:S #IMM8, dest



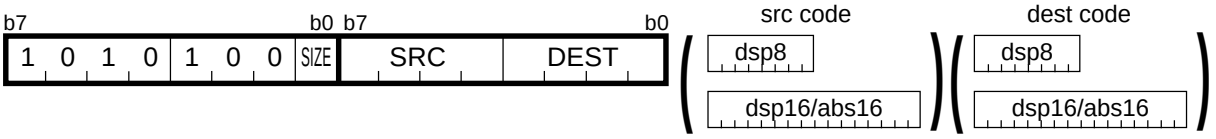
dest		DEST
Rn	R0H	0 1 1
	R0L	1 0 0
dsp:8[SB/FB]	dsp:8[SB]	1 0 1
	dsp:8[FB]	1 1 0
abs16	abs16	1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	2/1	3/3	4/3

SUB

(3) SUB.size:G src, dest



.size	SIZE
.B	0
.W	1

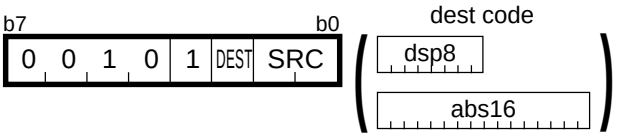
src/dest		SRC/DEST	src/dest		SRC/DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Rn	2/2	2/2	2/3	3/3	3/3	4/3	4/3	4/3
An	2/2	2/2	2/3	3/3	3/3	4/3	4/3	4/3
[An]	2/3	2/3	2/4	3/4	3/4	4/4	4/4	4/4
dsp:8[An]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	5/4
dsp:8[SB/FB]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	5/4
dsp:16[An]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4
dsp:16[SB]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4
abs16	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4

SUB

(4) SUB.B:S src, R0L/R0H



src		SRC
Rn	R0L/R0H	0 0
dsp:8[SB/FB]	dsp:8[SB]	0 1
	dsp:8[FB]	1 0
abs16	abs16	1 1

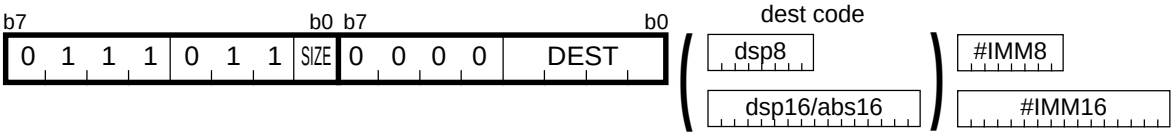
dest	DEST
R0L	0
R0H	1

[Number of Bytes/Number of Cycles]

src	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	1/2	2/3	3/3

TST

(1) TST.size #IMM, dest



.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

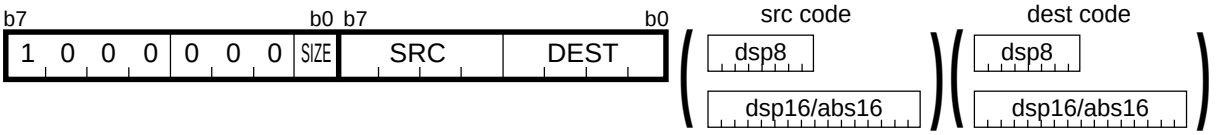
[Number of Bytes/Number of Cycles]

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

*1 If the size specifier (.size) is (.W), the number of bytes above is increased by 1.

TST

(2) TST.size src, dest



.size	SIZE
.B	0
.W	1

src/dest		SRC/DEST	src/dest		SRC/DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Rn	2/2	2/2	2/3	3/3	3/3	4/3	4/3	4/3
An	2/2	2/2	2/3	3/3	3/3	4/3	4/3	4/3
[An]	2/3	2/3	2/4	3/4	3/4	4/4	4/4	4/4
dsp:8[An]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	5/4
dsp:8[SB/FB]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	5/4
dsp:16[An]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4
dsp:16[SB]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4
abs16	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4

UND

(1) UND

b7							b0
1	1	1	1	1	1	1	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	1/20

WAIT

(1) WAIT

b7							b0	b7							b0
0	1	1	1	1	1	0	1	1	1	1	1	0	0	1	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/3

XCHG

(1) XCHG.size src, dest

b7							b0	b7							b0
0	1	1	1	1	0	1	SIZE	0	0	SRC		DEST			

dest code

dsp8
dsp16/abs16

.size	SIZE
.B	0
.W	1

src	SRC
R0L/R0	0 0
R0H/R1	0 1
R1L/R2	1 0
R1H/R3	1 1

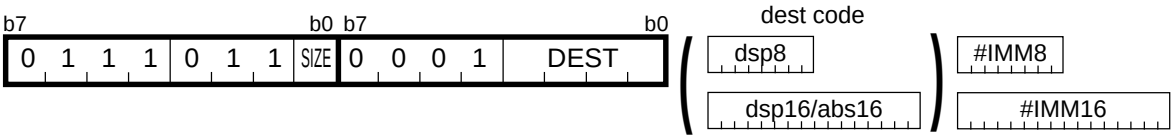
dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

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

XOR

(1) XOR.size #IMM, dest



.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

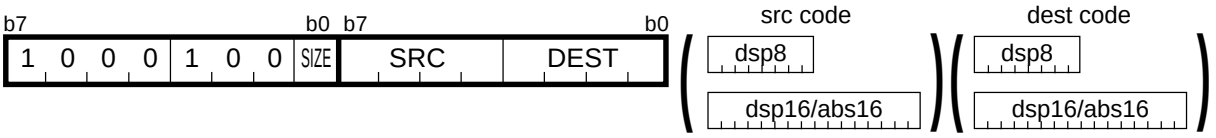
[Number of Bytes/Number of Cycles]

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

*1 If the size specifier (.size) is (.W), the number of bytes above is increased by 1.

XOR

(2) XOR.size src, dest



.size	SIZE
.B	0
.W	1

src/dest		SRC/DEST	src/dest		SRC/DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Rn	2/2	2/2	2/3	3/3	3/3	4/3	4/3	4/3
An	2/2	2/2	2/3	3/3	3/3	4/3	4/3	4/3
[An]	2/3	2/3	2/4	3/4	3/4	4/4	4/4	4/4
dsp:8[An]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	5/4
dsp:8[SB/FB]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	5/4
dsp:16[An]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4
dsp:16[SB]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4
abs16	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4

Chapter 5

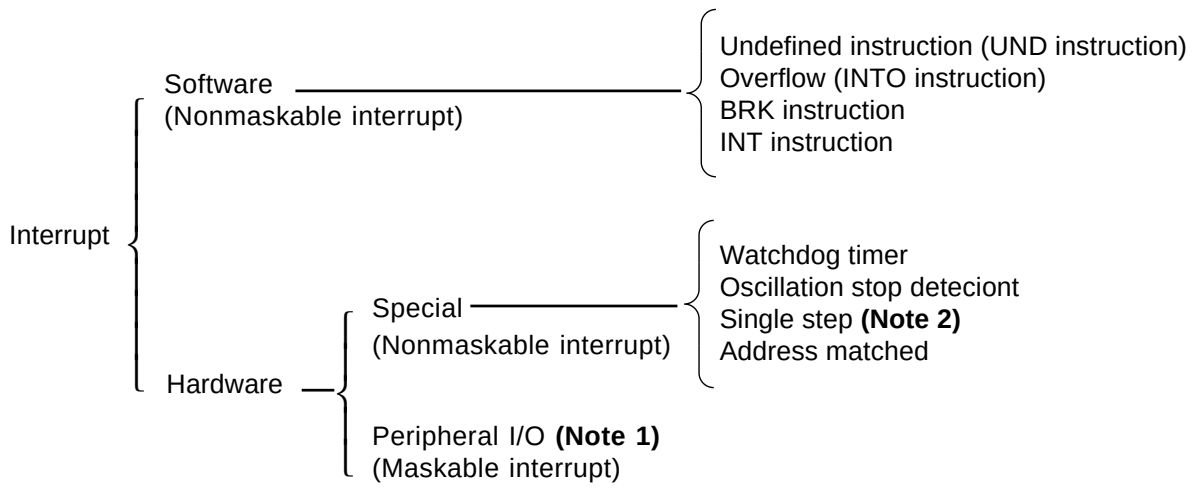
Interrupt

5.1 Outline of Interrupt

When an interrupt request is acknowledged, control branches to the interrupt routine that is set to an interrupt vector table. Each interrupt vector table must have had the start address of its corresponding interrupt routine set. For details about the interrupt vector table, refer to Section 1.10, “Vector Table”.

5.1.1 Types of Interrupts

Figure 5.1.1 lists the types of interrupts. Table 5.1.1 lists the source of interrupts (nonmaskable) and the fixed vector tables.



Note 1: Peripheral function interrupts are generated by the peripheral functions built into the microcomputer system.
Note 2: This is dedicated interrupt for development support tools. Do not use this interrupt.

Figure 5.1.1 Classification of interrupts

- Maskable interrupt: This type of interrupt **can** be controlled by using the I flag to enable (or disable) an interrupt or by changing the interrupt priority level.
- Nonmaskable interrupt: This type of interrupt **cannot** be controlled by using the I flag to enable (or disable) an interrupt or by changing the interrupt priority level.

Table 5.1.1 Interrupt Source (Nonmaskable) and Fixed Vector Table

Interrupt source	Vector table addresses Address (L) to address (H)	Remarks
Undefined instruction	$0FFDC_{16}$ to $0FFDF_{16}$	Interrupt generated by the UND instruction.
Overflow	$0FFE0_{16}$ to $0FFE3_{16}$	Interrupt generated by the INTO instruction.
BRK instruction	$0FFE4_{16}$ to $0FFE7_{16}$	Executed beginning from address indicated by vector in variable vector table if $0FFE7_{16}$ address contents are FF_{16}.
Address match	$0FFE8_{16}$ to $0FFEB_{16}$	Can be controlled by an interrupt enable bit.
Single step **(Note 1)**	$0FFEC_{16}$ to $0FFEF_{16}$	Do not use this interrupt.
Watchdog timer•Oscillation stop detection	$0FFF0_{16}$ to $0FFF3_{16}$	
(Reserved)	$0FFF4_{16}$ to $0FFF7_{16}$	
(Reserved)	$0FFF8_{16}$ to $0FFFB_{16}$	
Reset	$0FFFC_{16}$ to $0FFFF_{16}$	

Note 1: This is dedicated interrupt for development support tools. Do not use this interrupt.

5.1.2 Software Interrupts

Software interrupts are generated by some instruction that generates an interrupt request when executed. Software interrupts are nonmaskable interrupts.

●Undefined-instruction interrupt

This interrupt occurs when the UND instruction is executed.

●Overflow interrupt

This interrupt occurs if the INTO instruction is executed when the O flag is 1 (arithmetic result is overflow). The following lists the instructions that cause the O flag to change:

ABS, ADC, ADCF, ADD, CMP, DIV, DIVU, DIVX, NEG, RMPA, SBB, SHA, SUB

●BRK interrupt

This interrupt occurs when the BRK instruction is executed.

●INT instruction interrupt

This interrupt occurs when the INT instruction is executed. The software interrupt numbers which can be specified by INT instcution are 0 to 63. Note that software interrupt numbers 4 to 31 are assigned to peripheral function interrupts. This means that by executing the INT instruction, you can execute the same interrupt routine as used in peripheral function interrupts.

For software interrupt numbers 0 to 31, the U flag is saved when the INT instruction is executed and the U flag is cleared to 0 to choose the interrupt stack pointer (ISP) before executing the interrupt sequence. The previous U flag before the interrupt occurred is restored when control returns from the interrupt routine. For software interrupt numbers 32 to 63, when the instruction is executed, U flag does not change but uses selected SP at the time.

5.1.3 Hardware Interrupts

There are two types in hardware interrupts; special interrupts and peripherai function interrupts.

●Special interrupts

Special interrupts are nonmaskable interrupts.

(1) Watchdog timer interrupt

This interrupt is caused by the watchdog timer. Initialize the watchdog timer after the watchdog timer interrupt is generated. For details about the watchdog timer interrupt, refer to the R8C's Hardware Manual.

(2) Oscillation stop detection interrupt

This interrupt is caused by the oscillation stop detection interrupt. For details about the oscillation stop detection interrupt, refer to the R8C's Hardware Manual.

(3) Single-step interrupt

This interrupt is used exclusively for development support tools. Do not use this interrupt.

(4) Address-match interrupt

When any one of AIER0 bit or AIER1 bit of AIER register is "1" (address-match interrupt is enabled), the address-match interrupt is generated just before executing the instruction of the address shown by corresponding RMAD0 to RMAD1 registers.

●Peripheral function interrupts

These interrupts are generated by the peripheral functions built into the microcomputer system. Peripheral function interrupts are maskable interrupts.

The types of built-in peripheral functions vary with each R8C model, so do the types of interrupt causes. For details about peripheral function interrupts, refer to the R8C's Hardware Manual.

5.2 Interrupt Control

The following explains how to enable/disable maskable interrupts and set acknowledge priority. The explanation here does not apply to non-maskable interrupts.
Maskable interrupts are enabled and disabled by using the I flag, IPL, and ILVL2 bits to ILVL0 bits of each interrupt control register. Whether there is any interrupt requested is indicated by the IR bit of each interrupt control register.
For details about the memory allocation and the configuration of interrupt control registers, refer to the R8C's Hardware Manual.

5.2.1 I Flag

The I flag is used to disable/enable maskable interrupts. When the I flag is set to 1 (enabled), all maskable interrupts are enabled; when the I flag is cleared to 0 (disabed), they are disabled.
When the I flag is changed, the altered flag status is reflected in determining whether or not to accept an interrupt request at the following timing:

- If the flag is changed by an REIT instruction, the changed status takes effect beginning with that REIT instruction.
- If the flag is changed by an FCLR, FSET, POPC, or LDC instruction, the changed status takes effect beginning with the next instruction.

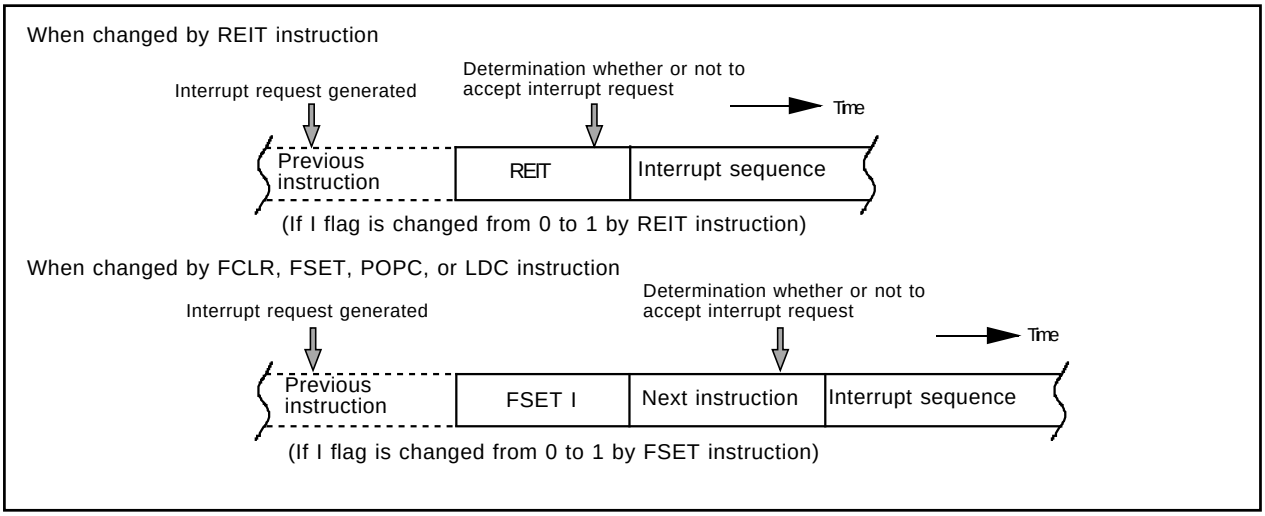


Figure 5.2.1 Timing at which changes of I flag are reflected in interrupt handling

5.2.2 IR Bit

The IR bit is set to 1 (interrupt request issued) when an interrupt request is generated. The IR bit is cleared to 0 (no interrupt request issued)after the interrupt request is acknowledged and the program brances to corresponding interrupt vector table.
The IR bit can be cleared to 0 by program. Do not set to 1.

5.2.3 ILVL2 to ILVL0 bis, IPL

Interrupt priority levels can be set by the ILVL2 to ILVL0 bits.
Table 5.2.1 shows how interrupt priority levels are set. Table 5.2.2 shows interrupt enable levels in relation to IPL.

The following lists the conditions under which an interrupt request is acknowledged:

- I flag = 1
- IR bit = 1
- Interrupt priority level > IPL

The I flag, ILVL2 to ILVL0 bits, and IPL are independent of each other, so they do not affect any other bit.

Table 5.2.1 Interrupt Priority Levels

ILVL2–ILVL0	Interrupt priority level	Priority order
000_2	Level 0(interrupt disabled)	———
001_2	Level 1	Low
010_2	Level 2	↓
011_2	Level 3	
100_2	Level 4	
101_2	Level 5	
110_2	Level 6	
111_2	Level 7	High

Table 5.2.2 Interrupt priority levels enabled by IPL

IPL	Enabled interrupt priority levels
000_2	Interrupt levels 1 and above are enabled.
001_2	Interrupt levels 2 and above are enabled.
010_2	Interrupt levels 3 and above are enabled.
011_2	Interrupt levels 4 and above are enabled.
100_2	Interrupt levels 5 and above are enabled.
101_2	Interrupt levels 6 and above are enabled.
110_2	Interrupt levels 7 and above are enabled.
111_2	All maskable interrupts are disabled.

When the IPL or the interrupt priority level of some interrupt is changed, the altered level is reflected in interrupt handling at the following timing:

- If the IPL is changed by an REIT instruction, the changed level takes effect beginning with the instruction that is executed two clock periods after the last clock of the REIT instruction.
- If the IPL is changed by a POPC, LDC, or LDIPL instruction, the changed level takes effect beginning with the instruction that is executed three clock periods after the last clock of the instruction used.
- If the interrupt priority level of a particular interrupt is changed by an instruction such as MOV, the changed level takes effect beginning with the instruction that is executed two clock or three clock periods after the last clock of the instruction used.

5.2.4 Changing Interrupt Control Register

(1) Each interrupt control register can only be modified while no interrupt requests corresponding to that register are generated. If interrupt requests managed by any interrupt control register are likely to occur, disable the interrupts before changing the interrupt control register.

(2) To modify any interrupt control register after disabling interrupts, be careful with the instructions used.

When Changing Other Than IR Bit

If an interrupt request corresponding to that register is generated while executing the instruction, the IR bit may not be set to “1” (interrupt requested), with the result that the interrupt request is ignored. If this presents a problem, use the following instructions to modify the register.

Instructions to use: AND, OR, BCLR, BSET

When Changing IR Bit

Even when the IR bit is cleared to “0” (interrupt not requested), it may not actually be cleared to “0” depending on the instruction used. Therefore, use the MOV instruction to set the IR bit to “0”.

(3) When disabling interrupts using the I flag, set the I flag according to the following sample programs. Refer to #2 for the change of interrupt control registers in the sample programs.

Sample programs 1 to 3 are to prevent the I flag from being set to “1” (interrupt enabled) before writing to the interrupt control registers for reasons of the internal bus or the instruction queue buffer.

Example 1: Use NOP instructions to prevent I flag being set to “1” before interrupt control register is changed

```
INT_SWITCH1:
    FCLR    I              ; Disable interrupts
    AND.B   #00H, 0056H    ; Set TXIC register to “00₁₆”
    NOP
    NOP
    FSET    I              ; Enable interrupts
```

Example 2: Use dummy read to have FSET instruction wait

```
INT_SWITCH2:
    FCLR    I              ; Disable interrupts
    AND.B   #00H, 0056H    ; Set TXIC register to “00₁₆”
    MOV.W   MEM, R0        ; Dummy read
    FSET    I              ; Enable interrupts
```

Example 3: Use POPC instruction to change I flag

```
INT_SWITCH3:
    PUSHC   FLG
    FCLR    I              ; Disable interrupts
    AND.B   #00H, 0056H    ; Set TXIC register to “00₁₆”
    POPC    FLG            ; Enable interrupts
```

5.3 Interrupt Sequence

An interrupt sequence — what are performed over a period from the instant an interrupt is accepted to the instant the interrupt routine is executed — is described here.

If an interrupt occurs during execution of an instruction, the processor determines its priority when the execution of the instruction is completed, and transfers control to the interrupt sequence from the next cycle. If an interrupt occurs during execution of either the SMOVB, SMOVF, SSTR or RMPA instruction, the processor temporarily suspends the instruction being executed, and transfers control to the interrupt sequence. Figure 5.3.1 shows the interrupt sequence executing time.

In the interrupt sequence, the processor carries out the following in sequence given:

(1) CPU gets the interrupt information (the interrupt number and interrupt request level) by reading address 00000_{16}. Then, the IR bit of corresponding interrupt is set to 0 (no interrupt request issued).

(2) Saves the FLG register as it was immediately before the start of interrupt sequence in the temporary register (Note 1) within the CPU.

(3) The I flag, the D flag, and the U flag of the FLG register are as follows:

- The I flag is set to 0 (interrupt disabled)
- The D flag is set to 0 (single-step interrupt is disabled)
- The U flag is set to 0 (ISP is specified)

However, the U flag does not change when the INT instruction of the software interrupt numbers 32-63 is executed.

(4) Saves the temporary register (Note 1) within the CPU in the stack area.

(5) Saves the PC in the stack area.

(6) Sets the interrupt priority level of the accepted instruction in the IPL.

(7) The first address of the interrupt routine set to the interrupt vector is set to the PC.

After the interrupt sequence is completed, the processor resumes executing instructions from the first address of the interrupt routine.

Note 1: This register cannot be utilized by the user.

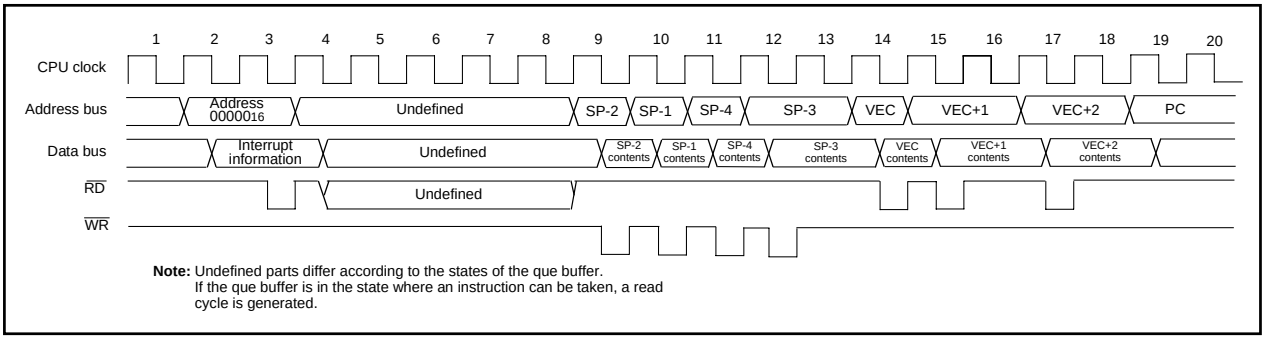
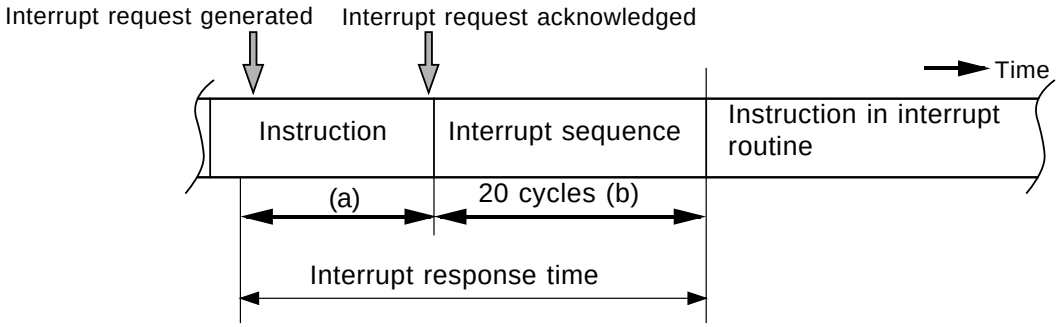


Figure 5.3.1 Interrupt sequence executing time

5.3.1 Interrupt Response Time

Figure 5.3.2 shows the interrupt resonse time. The interrupt response time means a period of time from when an interrupt request is generated till when the first instruction of the interrupt routine is executed. This period consists of time ((a) into Figure 5.3.1) from when an interrupt request is generated to when the instruction then under way is completed and time (20 cycles (b)) in which an interrupt sequence is executed.



(a) Time from when interrupt request is generated to when the instruction then under execution is completed. Time (a) varies with each instruction being executed. The DIVX instruction requires a maximum time that consists of 30 cycles (without wait state, cycle number in case the divisor is register).

(b) The address-match interrupt and the single-step interrupt are 21 cycles.

Figure 5.3.2 Interrupt response time

5.3.2 Changes of IPL When Interrupt Request Acknowledged

When an interrupt request of maskable instruction is acknowledged, the interrupt priority level of the acknowledged interrupt is set to the IPL.

When an software interrupt request or an special interrupt request is acknowledged, the value shown in Table 5.3.1 is set to the IPL. Table 5.3.1 shows the value of IPL when software interrupt and special interrupt request acknowledged.

Table 5.3.1 Value of IPL when software interrupt and special interrupt request acknowledged

Interrupt sources without interrupt priority levels	Value that is set to IPL
Watchdog timer, Oscillation stop detection	7
Software, Address-match, Single-step	Not changed

5.3.3 Saving Registers

In an interrupt sequence, the FLG register and the PC are saved to the stack area.

The order in which these contents are saved is as follows: First, the 4 high-order bits of the PC and 4 high-order bits (IPL) and 8 low-order bits of the FLG register for a total of 16 bits are saved to the stack area. Next, the 16 low-order bits of the PC are saved. Figure 5.3.3 shows the stack status before an interrupt request is acknowledged.

If there are any other registers you want to be saved, save them in program at the beginning of the interrupt routine. The PUSHM instruction allows you to save all registers except the SP by a single instruction.

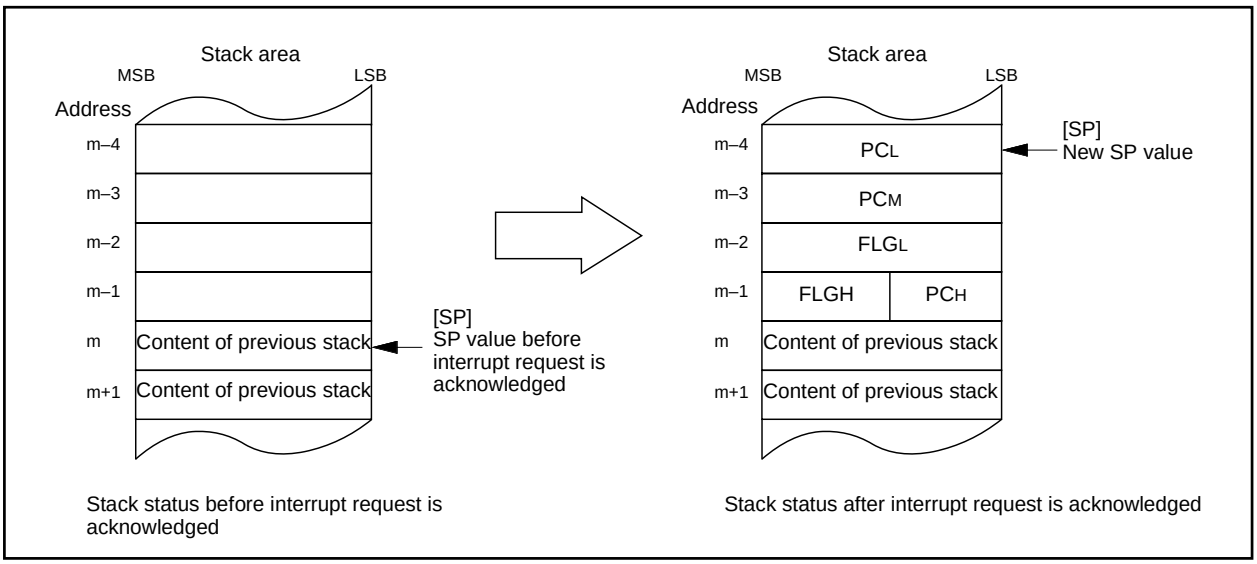


Figure 5.3.3 Stack status before and after an interrupt request is acknowledged

Register save operation performed in an interrupt sequence is executed in four operations 8 bits at a time. Figure 5.3.4 shows the operation to save registers.

Note 1: When the INT instruction for software interrupt numbers 32 to 63 is executed, SP is indicated by the U flag. The others are ISP.

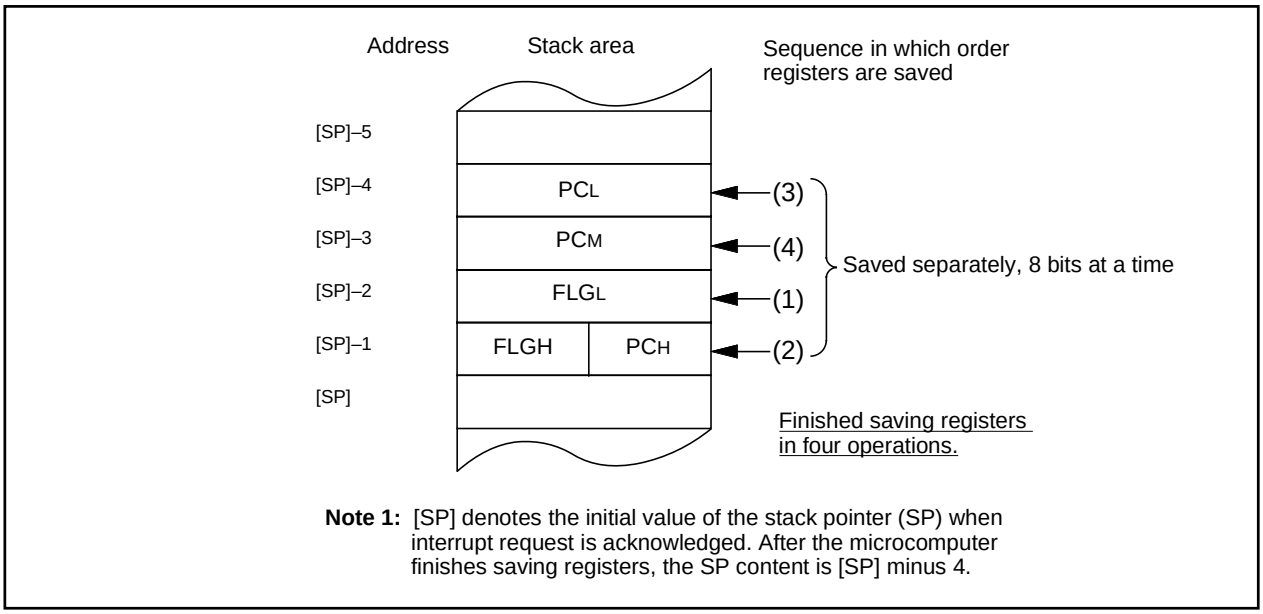


Figure 5.3.4 Operations to save registers

5.4 Return from Interrupt Routine

As you execute the REIT instruction at the end of the interrupt routine, the contents of the FLG register and the PC that have been saved to the stack area immediately preceding the interrupt sequence are automatically restored. Then control returns to the routine that was under execution before the interrupt request was acknowledged.

If there are any registers you saved via program in the interrupt routine, be sure to restore them using an instruction (e.g., POPM instruction) before executing the REIT instruction.

5.5 Interrupt Priority

When two or more interrupt requests occur while 1 instruction is executed, whichever interrupt request is acknowledged that has the highest priority.

The priority level of maskable interrupts (Peripheral function) can be selected arbitrarily by setting the ILVL2 to ILVL0 bits. If some maskable interrupts are assigned the same priority level, the priority between these interrupts is resolved by the priority that is set in hardware.

Special interrupts such as the watchdog timer interrupt have their priority levels set in hardware. Figure 5.5.1 lists the interrupt priority levels of hardware interrupts.

Software interrupts are not subjected to interrupt priority. They always causes control to branch to an interrupt routine when the relevant instruction is executed.

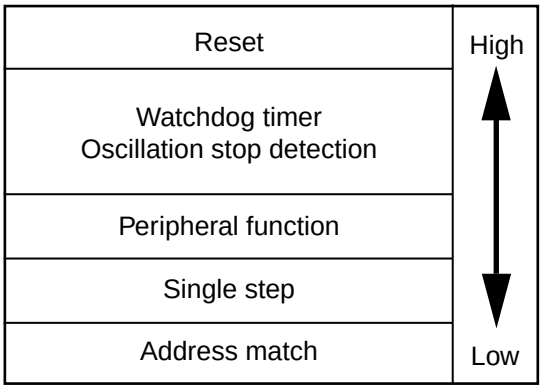


Figure 5.5.1 Interrupt priority levels of hardware interrupts

5.6 Multiple Interrupts

The following shows the internal bit states when control has branched to an interrupt routine:

- The interrupt enable flag (I flag) is cleared to 0 (interrupts disabled).
- The interrupt request bit for the acknowledged interrupt is cleared to 0.
- The processor interrupt priority level (IPL) equals the interrupt priority level of the acknowledged interrupt.

By setting the interrupt enable flag (I flag) (= 1) in the interrupt routine, you can reenable interrupts so that an interrupt request can be acknowledged that has higher priority than the processor interrupt priority level (IPL). Figure 5.6.1 shows how multiple interrupts are handled.
The interrupt requests that have not been acknowledged for their low interrupt priority level are kept pending. When the IPL is restored by an REIT instruction and interrupt priority is resolved against it, the pending interrupt request is acknowledged if the following condition is met:

Interrupt priority level of pending interrupt request > Restored processor interrupt priority level (IPL)

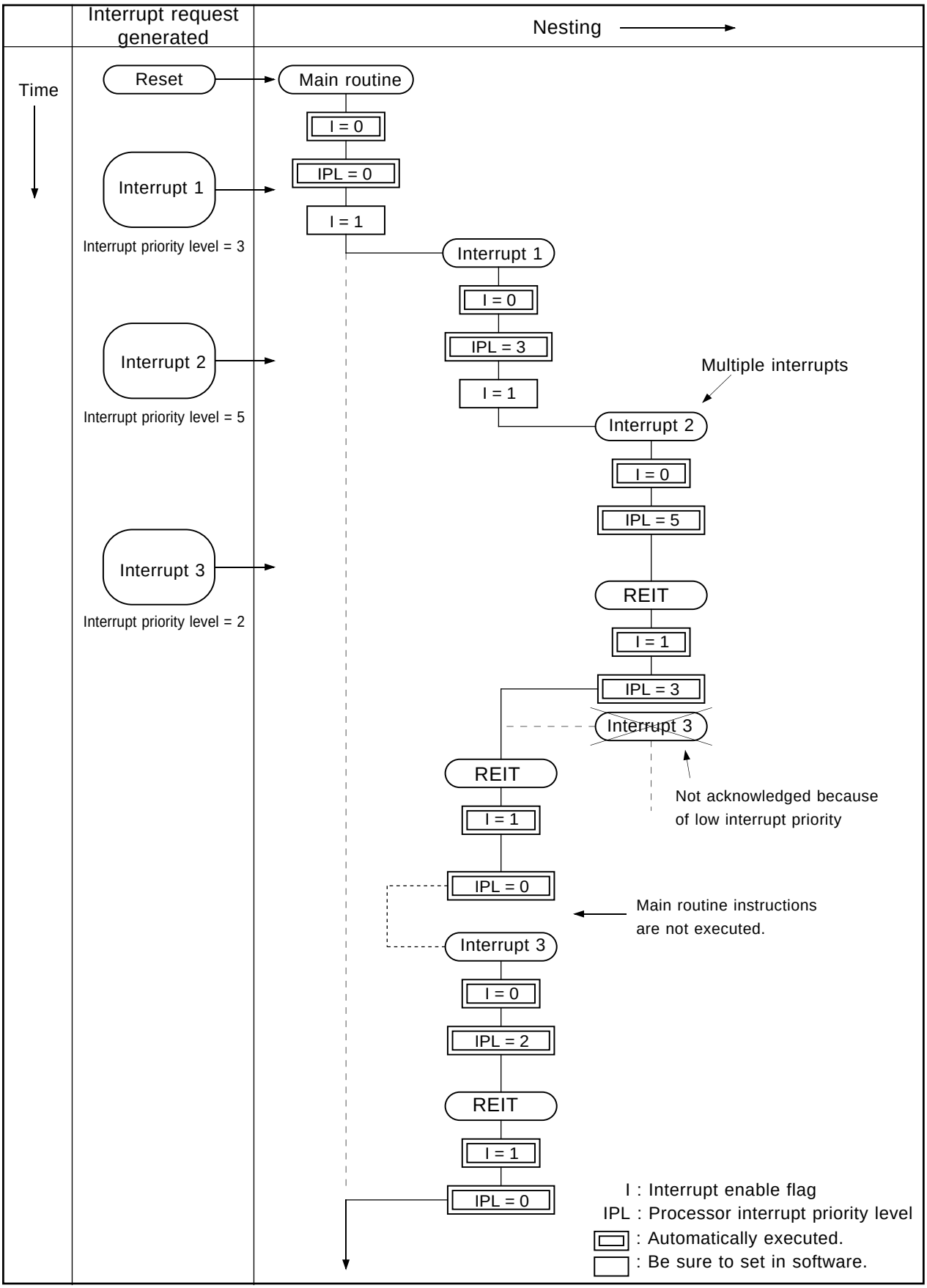


Figure 5.6.1 Multiple interrupts

5.7 Precautions for Interrupts

5.7.1 Reading Address 00000_{16}

Avoid reading the address 00000_{16} in a program. When a maskable interrupt request is accepted, the CPU reads interrupt information (interrupt number and interrupt request priority level) from the address 00000_{16} during the interrupt sequence. At this time, the IR bit for the accepted interrupt is set to “0”.
If the address 00000_{16} is read in a program, the IR bit for the interrupt which has the highest priority among the enabled interrupts is set to “0”. This may cause a problem that the interrupt is canceled, or an unexpected interrupt is generated.

5.7.2 SP Setting

Set any value in the SP before accepting an interrupt. The SP is set to “0000_{16}” after reset. Therefore, if an interrupt is accepted before setting any value in the SP, the program may go out of control.

5.7.3 Changing Interrupt Control Register

(1) Each interrupt control register can only be modified while no interrupt requests corresponding to that register are generated. If interrupt requests managed by any interrupt control register are likely to occur, disable the interrupts before changing the interrupt control register.

(2) To modify any interrupt control register after disabling interrupts, be careful with the instructions used.

When Changing Other Than IR Bit

If an interrupt request corresponding to that register is generated while executing the instruction, the IR bit may not be set to “1” (interrupt requested), with the result that the interrupt request is ignored. If this presents a problem, use the following instructions to modify the register.
Instructions to use: AND, OR, BCLR, BSET

When Changing IR Bit

Even when the IR bit is cleared to “0” (interrupt not requested), it may not actually be cleared to “0” depending on the instruction used. Therefore, use the MOV instruction to set the IR bit to “0”.

(3) When disabling interrupts using the I flag, set the I flag according to the following sample programs. Refer to #2 for the change of interrupt control registers in the sample programs.

Sample programs 1 to 3 are to prevent the I flag from being set to “1” (interrupt enabled) before writing to the interrupt control registers for reasons of the internal bus or the instruction queue buffer.

Example 1: Use NOP instructions to prevent I flag being set to "1" before interrupt control register is changed

```
INT_SWITCH1:
    FCLR    I              ; Disable interrupts
    AND.B   #00H, 0056H    ; Set TXIC register to "00₁₆"
    NOP
    NOP
    FSET    I              ; Enable interrupts
```

Example 2: Use dummy read to have FSET instruction wait

```
INT_SWITCH2:
    FCLR    I              ; Disable interrupts
    AND.B   #00H, 0056H    ; Set TXIC register to "00₁₆"
    MOV.W   MEM, R0        ; Dummy read
    FSET    I              ; Enable interrupts
```

Example 3: Use POPC instruction to change I flag

```
INT_SWITCH3:
    PUSHC   FLG
    FCLR    I              ; Disable interrupts
    AND.B   #00H, 0056H    ; Set TXIC register to "00₁₆"
    POPC    FLG            ; Enable interrupts
```

Chapter 6

Calculation Number of Cycles

6.1 Instruction queue buffer

6.1 Instruction Queue Buffer

The R8C/Tiny series have 4-stage (4-byte) instruction queue buffers. If the instruction queue buffer has a free space when the CPU can use the bus, instruction codes are taken into the instruction queue buffer. This is referred to as “prefetch”. The CPU reads (fetches) these instruction codes from the instruction queue buffer as it executes a program.

Explanation about the number of cycles in Chapter 4 assumes that all the necessary instruction codes are placed in the instruction queue buffer, and that 8-bit data is read or written to the memory without software wait. In the following cases, more cycles may be needed than the number of cycles shown in this manual:

- When not all of the instruction codes needed by the CPU are placed in the instruction queue buffer...
  Instruction codes are read in until all of the instruction codes required for program execution are available. Furthermore, the number of read cycles increases in the following cases:
  (1) The number of read cycles increases as many as the number of wait cycles incurred when reading instruction codes from an area in which software wait exists.
- When reading or writing data to an area in which software wait exists...
  The number of read or write cycles increases as many as the number of wait cycles incurred.
- When reading or writing 16-bit data from/to the SFR or the internal memory...
  The memory is accessed twice to read or write one 16-bit data. Therefore, the number of read or write cycles increases by one for each 16-bit data read or written.

Note that if prefetch and data access occur in the same timing, data access has priority. Also, if more than three bytes of instruction codes exist in the instruction queue buffer, the CPU assumes there is no free space in the instruction queue buffer and, therefore, does not prefetch instruction code.

Figures 6.1.1 shows an example when starting a read instruction (without software wait).

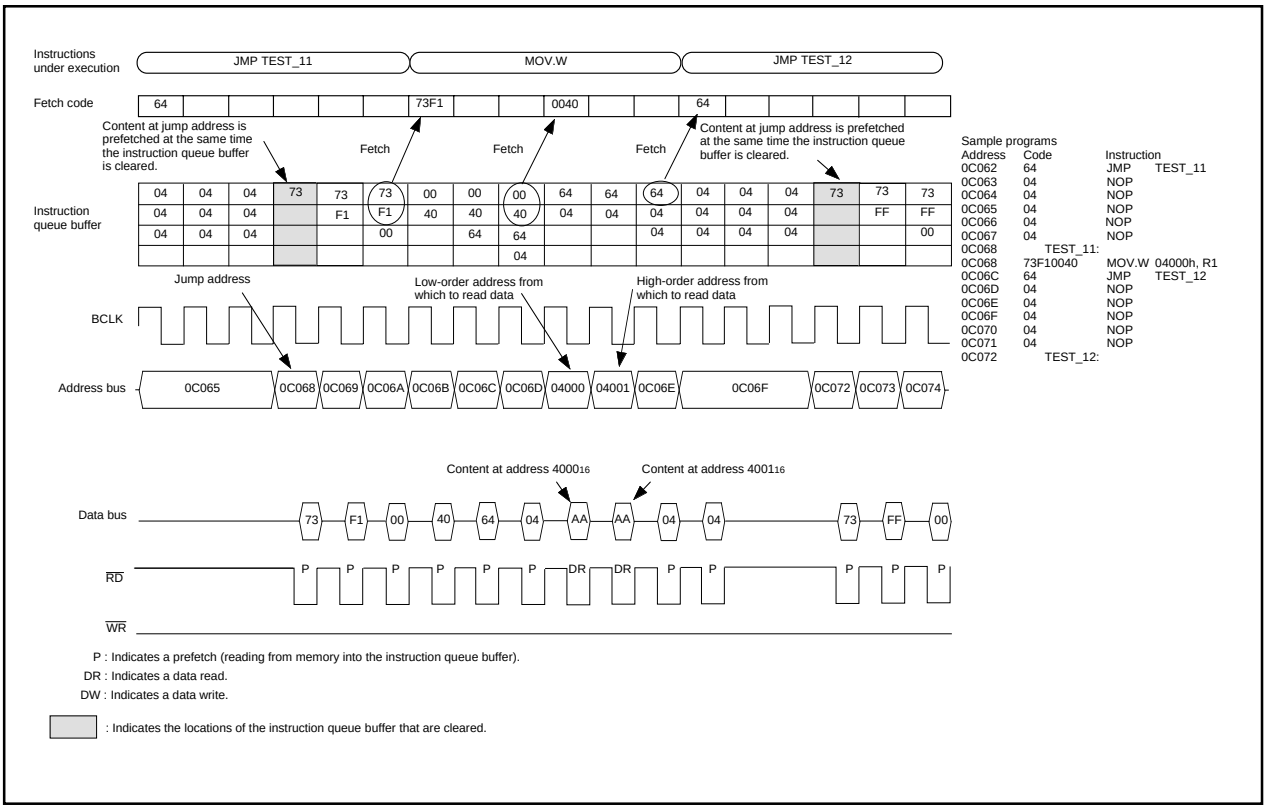


Figure 6.1.1 When starting a read instruction (without software wait state)

Q & A

Information in a Q&A form to be used to make the most of the R8C/Tiny series is given below.
Usually, one question and the answer to it are given on one page; the upper section is for the question, and the lower section is for the answer (if a pair of question and answer extends over two or more pages, a page number is given at the lower-right corner).
Functions closely connected with the contents of a page are shown at its upper-right corner.

Q

How do I distinguish between the static base register (SB) and the frame base register (FB)?

A

SB and FB function in the same manner, so you can use them as intended in programming in the assembly language. If you write a program in C, use FB as a stack frame base register.

Q

Is it possible to change the value of the interrupt table register (INTB) while a program is being executed?

A

Yes. But there can be a chance that the microcomputer runs away out of control if an interrupt request occurs in changing the value of INTB. So it is not recommended to frequently change the value of INTB while a program is being executed.

Q

What is the difference between the user stack pointer (USP) and the interrupt stack pointer (ISP)?, What are their roles?

A

You use USP when using the OS. When several tasks run, the OS secures stack areas to save registers of individual tasks. Also, stack areas have to be secured, task by task, to be used for handling interrupts that occur while tasks are being executed. If you use USP and ISP in such an instance, the stack for interrupts can be shared by these tasks; this allows you to efficiently use stack areas.

Q

How does the instruction code become if I use a bit instruction in absolute addressing ?

A

An explanation is given here by taking BSET bit,base:16 as an example.
This instruction is a 4-byte instruction. The 2 higher-order bytes of the instruction code indicate operation code, and the 2 lower-order bytes make up addressing mode to expresse bit,base:16.
The relation between the 2 lower-order bytes and bit,base:16 is as follows.
2 lower-order bytes = base:16 × 8 + bit

For example, in the case of BSET 2,0AH (setting bit 2 of address $000A_{16}$ to 1), the 2 lower-order bytes turn to A × 8 + 2 = 52H.
In the case of BSET 18,8H (setting the 18th bit from bit 0 of address 0008_{16} to 1), the 2 lower-order bytes turn to 8 × 8 + 18 = 52H, which is equivalent to BSET 2,AH.

The maximum value of base:16 × 8 + bit, FFFFH, indicates bit 7 of address $1FFF_{16}$. This is the maximum bit you can specify when using the bit instruction in absolute addressing.

Q

What is the difference between the DIV instruction and the DIVX instruction?

A

Either of the DIV instruction and the DIVX instruction is an instruction for signed division, the sign of the remainder is different.
The sign of the remainder left after the DIV instruction is the same as that of the dividend, on the contrary, the sign of the remainder of the DIVX instruction is the same as that of the divisor.
In general, the following relation among quotient, divisor, dividend, and remainder holds.
dividend = divisor × quotient + remainder
Since the sign of the remainder is different between these instructions, the quotient obtained either by dividing a positive integer by a negative integer or by dividing a negative integer by a positive integer using the DIV instruction is different from that obtained using the DIVX instruction.
For example, dividing 10 by –3 using the DIV instruction yields –3 and leaves +1, while doing the same using the DIVX instruction yields –4 and leaves –2.
Dividing –10 by +3 using the DIV instruction yields –3 and leaves –1, while doing the same using the DIVX instruction yields –4 and leaves +2.

Glossary

Technical terms used in this software manual are explained below. They are good in this manual only.

Term	Meaning	Related word
borrow	Tomove a digit to the next lower position.	carry
carry	Tomove a digit to the next higher position.	borrow
context	Registers that a program uses.	
decimal addition	An addition in terms of decimal system.	
displacement	The difference between the initial position and later position.	
effective address	An after-modification address to be actually used.	
extention area	For the R8C/Tiny series, the area from 10000_{16} through $FFFFF_{16}$.	
LSB	Abbreviation for Least Significant Biit The bit occupying the lowest-order position of a data item.	MSB

Term	Meaning	Related word
macro instruction	An instruction, written in a source language, to be expressed in a number of machine instructions when compiled into a machine code program.	
MSB	Abbreviation for Most Significant Bit The bit occupying the highest-order position of a data item.	LSB
operand	A part of instruction code that indicates the object on which an operation is performed.	operation code
operation	A generic term for move, comparison, bit processing, shift, rotation, arithmetic, logic, and branch.	
operation code	A part of instruction code that indicates what sort of operation the instruction performs.	operand
overflow	To exceed the maximum expressible value as a result of an operation.	
pack	To join data items. Used to mean to form two 4-bit data items into one 8-bit data item, to form two 8-bit data items into one 16-bit data item, etc.	unpack
SFR area	Abbreviation for Special Function Area. An area in which control bits of peripheral circuits embodied in a microcomputer and control registers are located.	

Term	Meaning	Related word
shift out	To move the content of a register either to the right or left until fully overflowed.	
sign bit	A bit that indicates either a positive or a negative (the highest-order bit).	
sign extension	To extend a data length in which the higher-order to be extended are made to have the same sign of the sign bit. For example, sign-extending FF_{16} results in $FFFF_{16}$, and sign-extending $0F_{16}$ results in $000F_{16}$.	
stack frame	An area for automatic variables the functions of the C language use.	
string	A sequence of characters.	
unpack	To restore combined items or packed information to the original form. Used to mean to separate 8-bit information into two parts — 4 lower-order bits and four higher-order bits, to separate 16-bit information into two parts — 8 lower-order bits and 8 higher-order bits, or the like.	pack
zero extension	To extend a data length by turning higher-order bits to 0's. For example, zero-extending FF_{16} to 16 bits results in $00FF_{16}$.	

Table of symbols

Symbols used in this software manual are explained below. They are good in this manual only.

Symbol	Meaning
←	Transposition from the right side to the left side
←→	Interchange between the right side and the left side
+	Addition
−	Subtraction
×	Multiplication
÷	Division
∧	Logical conjunction
∨	Logical disjunction
∀	Exclusive disjunction
¯	Logical negation
dsp16	16-bit displacement
dsp20	20-bit displacement
dsp8	8-bit displacement
EVA( )	An effective address indicated by what is enclosed in (Å@)
EXT( )	Sign extension
(H)	Higher-order byte of a register or memory
H4:	Four higher-order bits of an 8-bit register or 8-bit memory
| |	Absolute value
(L)	Lower-order byte of a register or memory
L4:	Four lower-order bits of an 8-bit register or 8-bit memory
LSB	Least Significant Bit
M( )	Content of memory indicated by what is enclosed in (Å@)
(M)	Middle-order byte of a register or memory
MSB	Most Significant Bit
PC_H	Higher-order byte of the program counter
PC_{ML}	Middle-order byte and lower-order byte of the program counter
FLG_H	Four higher-order bits of the flag register
FLG_L	Eight lower-order bits of the flag register

Index

REVISION HISTORY	R8C/Tiny Series SOFTWARE MANUAL

Rev.	Date	Description	
		Page	Summary
1.00	Jun./19/03		First Edition

RENESAS SEMICONDUCTORS
SOFTWARE MANUAL
R8C/Tiny Series Rev.1.00

Editioned by
Committee of editing of RENESAS Semiconductor Software Manual

---

R8C/Tiny Series
Software Manual

RENESAS

Renesas Technology Corp.
2-6-2, Ote-machi, Chiyoda-ku, Tokyo, 100-0004, Japan