RENESAS

User's Manual

32

RX Family RXv2 Instruction Set Architecture

User's Manual: Software

RENESAS 32-Bit MCU
RX Family

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

Renesas Electronics
www.renesas.com

Rev.1.00 Nov 2013

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
3. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.

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

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.

   Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
11. This document may not be reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

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

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

(2012.4)

Preface

This manual has been prepared to give the users a thorough understanding of the RXv2 instruction set architecture (RXv2) software as well as the ability to fully utilize the functions.

This manual contains detailed descriptions of CPU features and instruction sets, useful for a wide range of applications.

For information about RX Family hardware products and development support tools, also refer to the user's manuals or the operation manuals for the respective products.

Notation in This Manual

The following is a list of the elements of the notation used in this manual.

Classification	Notation	Meaning
Symbols	IMM	Immediate value
	SIMM	Immediate value for sign extension according to the processing size
	UIMM	Immediate value for zero extension according to the processing size
	src	Source of an instruction operand
	dest	Destination of an instruction operand
	dsp	Displacement of relative addressing
	pcdsp	Displacement of relative addressing of the program counter
	[ ]	Represents indirect addressing
	Rn	General-purpose register. R0 to R15 are specifiable unless stated otherwise.
	Rs	General-purpose register as a source. R0 to R15 are specifiable unless stated otherwise.
	Rs2	In the instructions where two general-purpose registers can be specified for operand, the first general-purpose register specified as a source is described as Rs and the second general-purpose register specified as a source is described as Rs2.
	Rd	General-purpose register as a destination. R0 to R15 are specifiable unless stated otherwise.
	Rd2	In the instructions where two general-purpose registers can be specified for operand, the first general-purpose register specified as a destination is described as Rd and the second general-purpose register specified as a destination is described as Rd2.
	Rb	General-purpose register specified as a base register. R0 to R15 are specifiable unless stated otherwise.
	Ri	General-purpose register as an index register. R0 to R15 are specifiable unless stated otherwise.
	Rx	Represents a control register. The PC, ISP, USP, INTB, EXTB, PSW, BPC, BPSW, FINTV, and FPSW are selectable, although the PC is only selectable as the src operand of MVFC and PUSHC instructions.
	flag	Represents a bit (U or I) or flag (O, S, Z, or C) in the PSW.
	Adest	Accumulator as a destination. A0 and A1 are specifiable.
	Asrc	Accumulator as a source. A0 and A1 are specifiable.
	tmp, tmp0, tmp1, tmp2, tmp3	Temporary registers
Values	000b	Binary number
	0000h	Hexadecimal number

Classification	Notation	Meaning
Bit length	#IMM:8 etc.	Represents the effective bit length for the operand symbol.
	:1	Indicates an effective length of one bit.
	:2	Indicates an effective length of two bits.
	:3	Indicates an effective length of three bits.
	:4	Indicates an effective length of four bits.
	:5	Indicates an effective length of five bits.
	:8	Indicates an effective length of eight bits.
	:16	Indicates an effective length of 16 bits.
	:24	Indicates an effective length of 24 bits.
	:32	Indicates an effective length of 32 bits.
Size specifiers	MOV.W etc.	Indicates the size that an instruction handles.
	.B	Byte (8 bits) is specified.
	.W	Word (16 bits) is specified.
	.L	Longword (32 bits) is specified.
Branch distance specifiers	BRA.A etc.	Indicates the length of the valid bits to represent the distance to the branch relative destination.
	.S	3-bit PC forward relative is specified. The range of valid values is 3 to 10.
	.B	8-bit PC relative is specified. The range of valid values is –128 to 127.
	.W	16-bit PC relative is specified. The range of valid values is –32768 to 32767.
	.A	24-bit PC relative is specified. The range of valid values is –8388608 to 8388607.
	.L	32-bit PC relative is specified. The range of valid values is –2147483648 to 2147483647.
Size extension specifiers added to memory operands	dsp:16[Rs].UB etc.	Indicates the size of a memory operand and the type of extension. If the specifier is omitted, the memory operand is handled as longword.
	.B	Byte (8 bits) is specified. The extension is sign extension.
	.UB	Byte (8 bits) is specified. The extension is zero extension.
	.W	Word (16 bits) is specified. The extension is sign extension.
	.UW	Word (16 bits) is specified. The extension is zero extension.
	.L	Longword (32 bits) is specified.

Classification	Notation	Meaning
Operations	(Operations in this manual are written in accord with C language syntax. The following is the notation in this manual.)	
	=	Assignment operator. The value on the right is assigned to the variable on the left.
	–	Indicates negation as a unary operator or a "difference" as a binary operator.
	+	Indicates "sum" as a binary operator.
	*	Indicates a pointer or a "product" as a binary operator.
	/	Indicates "quotient" as a binary operator.
	%	Indicates "remainder" as a binary operator.
	~	Indicates bit-wise "NOT" as a unary operator.
	&	Indicates bit-wise "AND" as a binary operator.
	|	Indicates bit-wise "OR" as a binary operator.
	^	Indicates bit-wise "Exclusive OR" as a binary operator.
	;	Indicates the end of a statement.
	{ }	Indicates the start and end of a complex sentence. Multiple statements can be put in { }.
	if (expression) statement 1 else statement 2	Indicates an if-statement. The expression is evaluated; statement 1 is executed if the result is true and statement 2 is executed if the result is false.
	for (statement 1; expression; statement 2) statement 3	Indicates a for-statement. After executing statement 1 and then evaluating the expression, statement 3 is executed if the result is true. After statement 3 is executed the first time, the expression is evaluated after executing statement 2.
	do statement while (expression);	Indicates a do-statement. As long as the expression is true, the statement is executed. Regardless of whether the expression is true or false, the statement is executed at least once.
	while (expression) statement	Indicates a while-statement. As long as the expression is true, the statement is executed.
Operations	==, !=	Comparison operators. "==" means "is equal to" and "!=" means "is not equal to".
	>, <	Comparison operators. ">" means "greater than" and "<" means "less than".
	>=, <=	Comparison operators. The condition includes "==" as well as ">" or "<".
	&&	Logical operator. Indicates the "AND" of the conditions to the left and right of the operator.
	||	Logical operator. Indicates the "OR" of the conditions to the left and right of the operator.
	<<, >>	Shift operators, respectively indicating leftward and rightward shifts.
	!	Logical operator, that is, inversion of the boolean value of a variable or expression.
Floating point number	NaN	Not a number
Floating-point standard	SNaN	Signaling NaN
	QNaN	Quiet NaN

Contents

List of RXv2 Instruction Set Architecture Instructions for RX Family

Quick Page Reference in Alphabetical Order (1 / 4)

Mnemonic		Function	Instruction Described in Detail (on Page)	Instruction Code Described in Detail (on Page)
ABS		Absolute value	58	211
ADC		Addition with carry	59	212
ADD		Addition without carry	60	213
AND		Logical AND	62	215
BCLR		Clearing a bit	64	217
B*Cnd*	BGEU	Relative conditional branch	65	219
	BC		65	219
	BEQ		65	219
	BZ		65	219
	BGTU		65	219
	BPZ		65	219
	BGE		65	219
	BGT		65	219
	BO		65	219
	BLTU		65	219
	BNC		65	219
	BNE		65	219
	BNZ		65	219
	BLEU		65	219
	BN		65	219
	BLE		65	219
	BLT		65	219
	BNO		65	219
BM*Cnd*	BMGEU	Conditional bit transfer	66	221
	BMC		66	221
	BMEQ		66	221
	BMZ		66	221
	BMGTU		66	221
	BMPZ		66	221
	BMGE		66	221
	BMGT		66	221
	BMO		66	221
	BMLTU		66	221
	BMNC		66	221
	BMNE		66	221
	BMNZ		66	221
	BMLEU		66	221
	BMN		66	221
	BMLE		66	221
	BMLT		66	221
	BMNO		66	221
BNOT		Inverting a bit	68	222

Quick Page Reference in Alphabetical Order (2 / 4)

Mnemonic	Function	Instruction Described in Detail (on Page)	Instruction Code Described in Detail (on Page)
BRA	Unconditional relative branch	69	224
BRK	Unconditional trap	70	225
BSET	Setting a bit	71	225
BSR	Relative subroutine branch	72	227
BTST	Testing a bit	73	228
CLRPSW	Clear a flag or bit in the PSW	74	230
CMP	Comparison	75	231
DIV	Signed division	76	233
DIVU	Unsigned division	78	235
EMACA	Extend multiply-accumulate to the accumulator	80	236
EMSBA	Extended multiply-subtract to the accumulator	81	236
EMUL	Signed multiplication	82	237
EMULA	Extended multiply to the accumulator	84	238
EMULU	Unsigned multiplication	85	238
FADD	Floating-point addition	87	240
FCMP	Floating-point comparison	90	241
FDIV	Floating-point division	93	242
FMUL	Floating-point multiplication	95	243
FSQRT	Floating-point square root	98	244
FSUB	Floating-point subtraction	98	245
FTOI	Floating point to integer conversion	103	246
FTOU	Floating point to integer conversion	106	246
INT	Software interrupt	109	247
ITOF	Integer to floating-point conversion	110	247
JMP	Unconditional jump	112	248
JSR	Jump to a subroutine	113	248
MACHI	Multiply-Accumulate the high-order word	114	249
MACLH	Multiply-Accumulate the lower-order word and higher-order word	115	249
MACLO	Multiply-Accumulate the low-order word	116	250
MAX	Selecting the highest value	117	250
MIN	Selecting the lowest value	118	252
MOV	Transferring data	119	253
MOVCO	Storing with LI flag clear	122	258
MOVLI	Loading with LI flag set	123	258
MOVU	Transfer unsigned data	124	259
MSBHI	Multiply-Subtract the higher-order word	126	260
MSBLH	Multiply-Subtract the lower-order word and higher-order word	127	260
MSBLO	Multiply-Subtract the lower-order word	128	261
MUL	Multiplication	129	261
MULHI	Multiply the high-order word	131	263
MULLH	Multiply the lower-order word and higher-order word	132	263
MULLO	Multiply the low-order word	133	264

Quick Page Reference in Alphabetical Order (3 / 4)

Mnemonic	Function	Instruction Described in Detail (on Page)	Instruction Code Described in Detail (on Page)
MVFACGU	Move the guard longword from the accumulator	134	264
MVFACHI	Move the high-order longword from accumulator	135	265
MVFACLO	Move the lower-order longword from the accumulator	136	265
MVFACMI	Move the middle-order longword from accumulator	137	266
MVFC	Transfer from a control register	138	266
MVTACGU	Move the guard longword to the accumulator	139	267
MVTACHI	Move the high-order longword to accumulator	140	267
MVTACLO	Move the low-order longword to accumulator	141	268
MVTC	Transfer to a control register	142	269
MVTIPL (privileged instruction)	Interrupt priority level setting	143	270
NEG	Two’s complementation	144	271
NOP	No operation	145	271
NOT	Logical complementation	146	272
OR	Logical OR	147	273
POP	Restoring data from stack to register	149	274
POPC	Restoring a control register	150	275
POPM	Restoring multiple registers from the stack	151	275
PUSH	Saving data on the stack	152	276
PUSHC	Saving a control register	153	277
PUSHM	Saving multiple registers	154	277
RACL	Round the accumulator longword	155	278
RACW	Round the accumulator word	157	278
RDACL	Round the accumulator longword	159	279
RDACW	Round the accumulator word	161	279
REVL	Endian conversion	163	280
REVW	Endian conversion	164	280
RMPA	Multiply-and-accumulate operation	165	281
ROLC	Rotation with carry to left	167	281
RORC	Rotation with carry to right	168	282
ROTL	Rotation to left	169	282
ROTR	Rotation to right	170	283
ROUND	Conversion from floating-point to integer	171	284
RTE (privileged instruction)	Return from the exception	174	284
RTFI (privileged instruction)	Return from the fast interrupt	175	285
RTS	Returning from a subroutine	176	285
RTSD	Releasing stack frame and returning from subroutine	177	285
SAT	Saturation of signed 32-bit data	179	286

Quick Page Reference in Alphabetical Order (4 / 4)

Mnemonic		Function	Instruction Described in Detail (on Page)	Instruction Code Described in Detail (on Page)
SATR		Saturation of signed 64-bit data for RMPA	180	286
SBB		Subtraction with borrow	181	287
SC*Cnd*	SCGEU	Condition setting	182	288
	SCC		182	288
	SCEQ		182	288
	SCZ		182	288
	SCGTU		182	288
	SCPZ		182	288
	SCGE		182	288
	SCGT		182	288
	SCO		182	288
	SCLTU		182	288
	SCNC		182	288
	SCNE		182	288
	SCNZ		182	288
	SCLEU		182	288
	SCN		182	288
	SCLE		182	288
	SCLT		182	288
	SCNO		182	288
SCMPU		String comparison	184	288
SETPSW		Setting a flag or bit in the PSW	185	289
SHAR		Arithmetic shift to the right	186	290
SHLL		Logical and arithmetic shift to the left	187	291
SHLR		Logical shift to the right	188	292
SMOVB		Transferring a string backward	189	293
SMOVF		Transferring a string forward	190	293
SMOVU		Transferring a string	191	293
SSTR		Storing a string	192	294
STNZ		Transfer with condition	193	294
STZ		Transfer with condition	194	295
SUB		Subtraction without borrow	195	296
SUNTIL		Searching for a string	196	297
SWHILE		Searching for a string	198	297
TST		Logical test	200	298
UTOF		Integer to floating-point conversion	201	299
WAIT (privileged instruction)		Waiting	203	300
XCHG		Exchanging values	204	300
XOR		Logical exclusive or	206	301

Section 1 CPU Programming Model

The RXv2 instruction set architecture (RXv2) has upward compatibility with the RXv1 instruction set architecture (RXv1).

- Adoption of variable-length instruction format

  As with RXv1, the RXv2 CPU has short formats for frequently used instructions, facilitating the development of efficient programs that take up less memory.
- Powerful instruction set

  The RXv2 supports 109 selected instructions. Moreover, DSP instructions and floating-point operation instructions are added, thus realizing high-speed arithmetic processing.
- Versatile addressing modes

  The RXv2 CPU has 11 versatile addressing modes, with register-register operations, register-memory operations, and bitwise operations included. Data transfer between memory locations is also possible.

1.1 Features

- Minimum instruction execution rate: One clock cycle
- Address space: 4-Gbyte linear addresses
- Register set of the CPU

  General purpose: Sixteen 32-bit registers

  Control: Ten 32-bit registers

  Accumulator: Two 72-bit registers
- Variable-length instruction format (lengths from one to eight bytes)
- 109 instructions/11 addressing modes

  Basic instructions: 75

  Floating-point operation instructions: 11

  DSP instructions: 23
- Processor modes

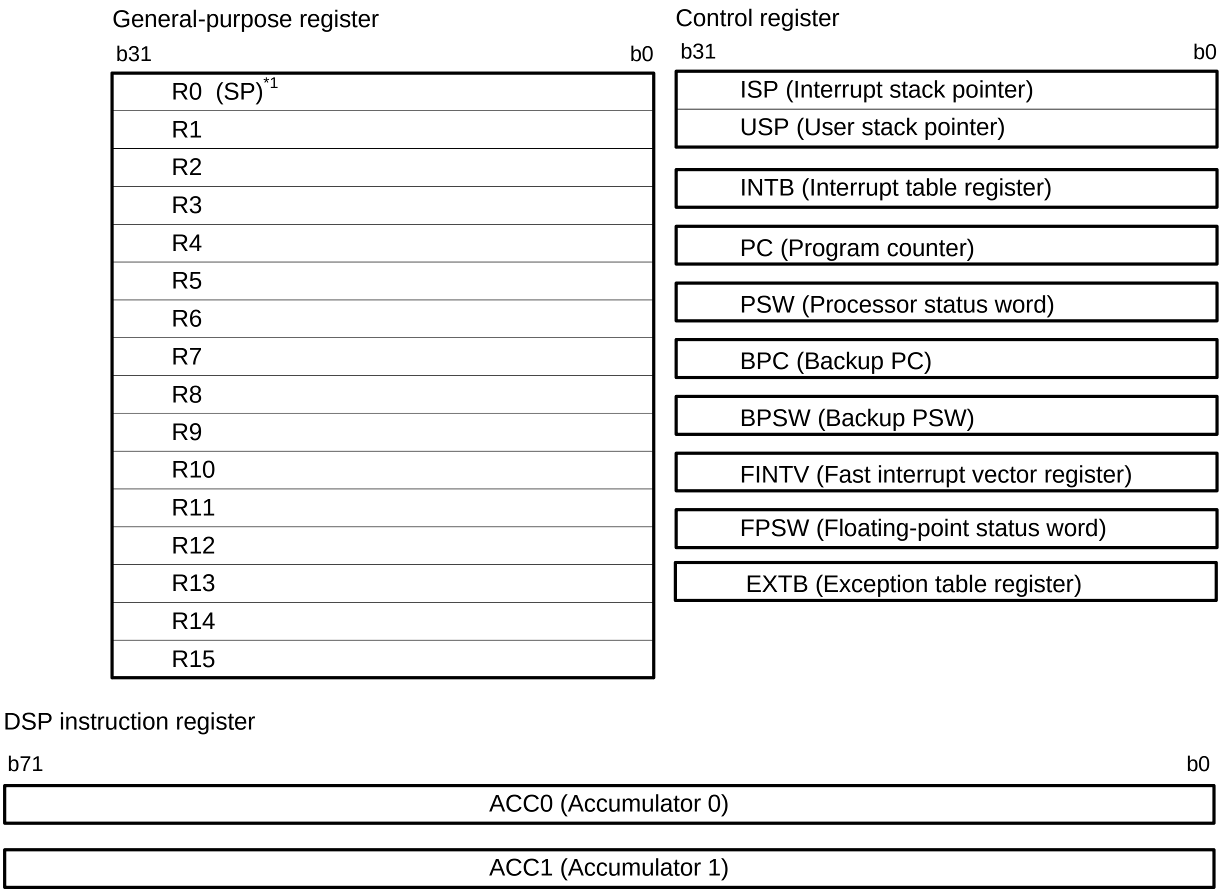
  Supervisor mode and user mode
- Vector tables

  Exception vector table and interrupt vector table
- Memory protection unit (as an optional function)
- Data arrangement

  Selectable as little endian or big endian

1.2 Register Set of the CPU

The RXv2 CPU has sixteen general-purpose registers, ten control registers, and two accumulator used for DSP instructions.



Note: 1. The stack pointer (SP) can be the interrupt stack pointer (ISP) or user stack pointer (USP), according to the value of the U bit in the PSW.

Figure 1.1 Register Set of the CPU

1.2.1 General-Purpose Registers (R0 to R15)

This CPU has sixteen 32-bit general-purpose registers (R0 to R15). R1 to R15 can be used as data register or address register.

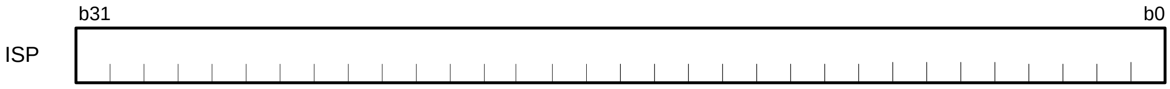
R0, a general-purpose register, also functions as the stack pointer (SP). The stack pointer is switched to operate as the interrupt stack pointer (ISP) or user stack pointer (USP) by the value of the stack pointer select bit (U) in the processor status word (PSW).

1.2.2 Control Registers

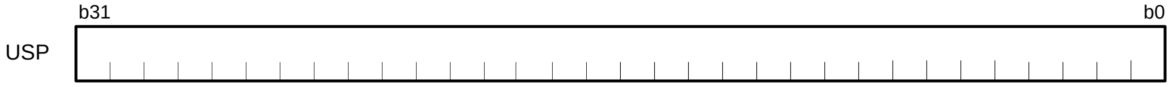
This CPU has the following ten control registers.

- Interrupt stack pointer (ISP)
- User stack pointer (USP)
- Interrupt table register (INTB)
- Program counter (PC)
- Processor status word (PSW)
- Backup PC (BPC)
- Backup PSW (BPSW)
- Fast interrupt vector register (FINTV)
- Floating-point status word (FPSW)
- Exception table register (EXTB)

1.2.2.1 Interrupt Stack Pointer (ISP)/User Stack Pointer (USP)



Value after reset: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0



Value after reset: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

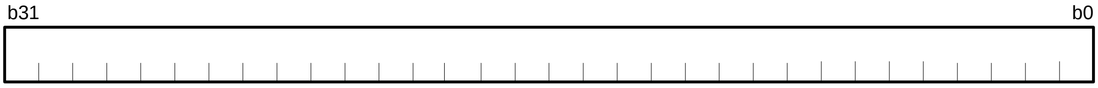
The stack pointer (SP) can be either of two types, the interrupt stack pointer (ISP) or the user stack pointer (USP). Whether the stack pointer operates as the ISP or USP depends on the value of the stack pointer select bit (U) in the processor status word (PSW).

1.2.2.2 Interrupt Table Register (INTB)

Value after reset: Undefined

The interrupt table register (INTB) specifies the address where the interrupt vector table starts.

1.2.2.3 Program Counter (PC)



Value after reset: Reset vector (Contents of addresses FFFFFFFCh to FFFFFFFFh)

The program counter (PC) indicates the address of the instruction being executed.

1.2.2.4 Processor Status Word (PSW)

	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	IPL[3:0]				—	—	—	PM	—	—	U	I
Value after reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	O	S	Z	C
Value after reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Symbol	Bit Name	Description	R/W
b0	C	Carry flag	0: No carry has occurred. 1: A carry has occurred.	R/W
b1	Z	Zero flag	0: Result is non-zero. 1: Result is 0.	R/W
b2	S	Sign flag	0: Result is a positive value or 0. 1: Result is a negative value.	R/W
b3	O	Overflow flag	0: No overflow has occurred. 1: An overflow has occurred.	R/W
b15 to b4	—	Reserved	These bits are read as 0. The write value should be 0.	R/W
b16	I*1	Interrupt enable bit	0: Interrupt disabled. 1: Interrupt enabled.	R/W
b17	U*1	Stack pointer select bit	0: Interrupt stack pointer (ISP) is selected. 1: User stack pointer (USP) is selected.	R/W
b19, b18	—	Reserved	These bits are read as 0. The write value should be 0.	R/W
b20	PM*1,*2,*3	Processor mode select bit	0: Supervisor mode is selected. 1: User mode is selected.	R/W
b23 to b21	—	Reserved	These bits are read as 0. The write value should be 0.	R/W
b27 to b24	IPL[3:0]*1	Processor interrupt priority level	b27 b24 0 0 0 0: Priority level 0 (lowest) 0 0 0 1: Priority level 1 0 0 1 0: Priority level 2 0 0 1 1: Priority level 3 0 1 0 0: Priority level 4 0 1 0 1: Priority level 5 0 1 1 0: Priority level 6 0 1 1 1: Priority level 7 1 0 0 0: Priority level 8 1 0 0 1: Priority level 9 1 0 1 0: Priority level 10 1 0 1 1: Priority level 11 1 1 0 0: Priority level 12 1 1 0 1: Priority level 13 1 1 1 0: Priority level 14 1 1 1 1: Priority level 15 (highest)	R/W
b31 to b28	—	Reserved	These bits are read as 0. The write value should be 0.	R/W

Notes: 1. In user mode, writing to the IPL[3:0], PM, U, and I bits by an MVTC or POPC instruction is ignored. Writing to the IPL[3:0] bits by an MVTIPL instruction generates a privileged instruction exception.

2. In supervisor mode, writing to the PM bit by an MVTC or POPC instruction is ignored, but writing to the other bits is possible.

3. Switching from supervisor mode to user mode requires execution of an RTE instruction after having set the PM bit in the PSW saved on the stack to 1 or executing an RTFI instruction after having set the PM bit in the backup PSW (BPSW) to 1.

The processor status word (PSW) indicates results of instruction execution or the state of the CPU.

C flag (Carry flag)

This flag retains the state of the bit after a carry, borrow, or shift-out has occurred.

Z flag (Zero flag)

This flag is set to 1 if the result of an operation is 0; otherwise its value is cleared to 0.

S flag (Sign flag)

This flag is set to 1 if the result of an operation is negative; otherwise its value is cleared to 0.

O flag (Overflow flag)

This flag is set to 1 if the result of an operation overflows; otherwise its value is cleared to 0.

I bit (Interrupt enable bit)

This bit enables interrupt requests. When an exception is accepted, the value of this bit becomes 0.

U bit (Stack pointer select bit)

This bit specifies the stack pointer as either the ISP or USP. When an exception request is accepted, this bit is set to 0. When the processor mode is switched from supervisor mode to user mode, this bit is set to 1.

PM bit (Processor mode select bit)

This bit specifies the operating mode of the processor. When an exception is accepted, the value of this bit becomes 0.

IPL[3:0] bits (Processor interrupt priority level)

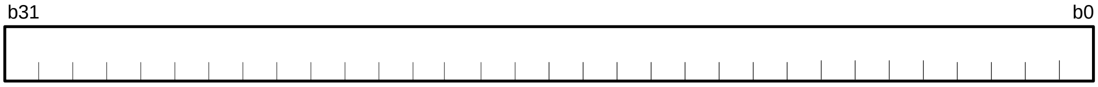
The IPL[3:0] bits specify the processor interrupt priority level as one of sixteen levels from zero to fifteen, where priority level zero is the lowest and priority level fifteen the highest. When the priority level of a requested interrupt is higher than the processor interrupt priority level, the interrupt is enabled. Setting the IPL[3:0] bits to level 15 (Fh) disables all interrupt requests. The IPL[3:0] bits are set to level 15 (Fh) when a non-maskable interrupt is generated. When interrupts in general are generated, the bits are set to the priority levels of accepted interrupts.

1.2.2.5 Backup PC (BPC)

Value after reset: Undefined

The backup PC (BPC) is provided to speed up response to interrupts. After a fast interrupt has been generated, the contents of the program counter (PC) are saved in the BPC.

1.2.2.6 Backup PSW (BPSW)



Value after reset: Undefined

The backup PSW (BPSW) is provided to speed up response to interrupts. After a fast interrupt has been generated, the contents of the processor status word (PSW) are saved in the BPSW. The allocation of bits in the BPSW corresponds to that in the PSW.

1.2.2.7 Fast Interrupt Vector Register (FINTV)

Value after reset: Undefined

The fast interrupt vector register (FINTV) is provided to speed up response to interrupts. The FINTV register specifies a branch destination address when a fast interrupt has been generated.

1.2.2.8 Floating-Point Status Word (FPSW)

	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	FS	FX	FU	FZ	FO	FV	—	—	—	—	—	—	—	—	—	—
Value after reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	EX	EU	EZ	EO	EV	—	DN	CE	CX	CU	CZ	CO	CV	RM[1:0]	
Value after reset:	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

Bit	Symbol	Bit Name	Description	R/W
b1, b0	RM[1:0]	Floating-point rounding-mode setting bits	b1 b0 0 0: Round to the nearest value 0 1: Round towards 0 1 0: Round towards +∞ 1 1: Round towards –∞	R/W
b2	CV	Invalid operation cause flag	0: No invalid operation has been encountered. 1: Invalid operation has been encountered.	R/(W)[*1]
b3	CO	Overflow cause flag	0: No overflow has occurred. 1: Overflow has occurred.	R/(W)[*1]
b4	CZ	Division-by-zero cause flag	0: No division-by-zero has occurred. 1: Division-by-zero has occurred.	R/(W)[*1]
b5	CU	Underflow cause flag	0: No underflow has occurred. 1: Underflow has occurred.	R/(W)[*1]
b6	CX	Inexact cause flag	0: No inexact exception has been generated. 1: Inexact exception has been generated.	R/(W)[*1]
b7	CE	Unimplemented processing cause flag	0: No unimplemented processing has been encountered. 1: Unimplemented processing has been encountered.	R/(W)[*1]
b8	DN	0 flush bit of denormalized number	0: A denormalized number is handled as a denormalized number. 1: A denormalized number is handled as 0.[*2]	R/W
b9	—	Reserved	This bit is read as 0. The write value should be 0.	R/W
b10	EV	Invalid operation exception enable bit	0: Invalid operation exception is masked. 1: Invalid operation exception is enabled.	R/W
b11	EO	Overflow exception enable bit	0: Overflow exception is masked. 1: Overflow exception is enabled.	R/W
b12	EZ	Division-by-zero exception enable bit	0: Division-by-zero exception is masked. 1: Division-by-zero exception is enabled.	R/W
b13	EU	Underflow exception enable bit	0: Underflow exception is masked. 1: Underflow exception is enabled.	R/W
b14	EX	Inexact exception enable bit	0: Inexact exception is masked. 1: Inexact exception is enabled.	R/W
b25 to b15	—	Reserved	These bits are read as 0. The write value should be 0.	R/W
b26	FV[*3]	Invalid operation flag	0: No invalid operation has been encountered. 1: Invalid operation has been encountered.[*8]	R/W

Bit	Symbol	Bit Name	Description	R/W
b27	FO[*4]	Overflow flag	0: No overflow has occurred. 1: Overflow has occurred.[*8]	R/W
b28	FZ[*5]	Division-by-zero flag	0: No division-by-zero has occurred. 1: Division-by-zero has occurred.[*8]	R/W
b29	FU[*6]	Underflow flag	0: No underflow has occurred. 1: Underflow has occurred.[*8]	R/W
b30	FX[*7]	Inexact flag	0: No inexact exception has been generated. 1: Inexact exception has been generated.[*8]	R/W
b31	FS	Floating-point error summary flag	This bit reflects the logical OR of the FU, FZ, FO, and FV flags.	R

Notes: 1. When 0 is written to the bit, the bit is set to 0; the bit remains the previous value when 1 is written.
2. Positive denormalized numbers are treated as +0, negative denormalized numbers as –0.
3. When the EV bit is set to 0, the FV flag is enabled.
4. When the EO bit is set to 0, the FO flag is enabled.
5. When the EZ bit is set to 0, the FZ flag is enabled.
6. When the EU bit is set to 0, the FU flag is enabled.
7. When the EX bit is set to 0, the FX flag is enabled.
8. Once the bit has been set to 1, this value is retained until it is cleared to 0 by software.

The floating-point status word (FPSW) indicates the results of floating-point operations. In products that do not support floating-point instructions, the value "00000000h" is always read out and writing to these bits does not affect operations.

When an exception handling enable bit (Ej) enables the exception handling (Ej = 1), the corresponding Cj flag indicates the cause. If the exception handling is masked (Ej = 0), check the Fj flag at the end of a series of processing. The Fj flag is the accumulation type flag (j = X, U, Z, O, or V).

RM[1:0] bits (Floating-point rounding-mode setting bits)

These bits specify the floating-point rounding-mode.

Explanation of Floating-Point Rounding Modes

- Rounding to the nearest value (the default behavior): An inexact result is rounded to the available value that is closest to the result which would be obtained with an infinite number of digits. If two available values are equally close, rounding is to the even alternative.
- Rounding towards 0: An inexact result is rounded to the smallest available absolute value; i.e., in the direction of zero (simple truncation).
- Rounding towards +∞: An inexact result is rounded to the nearest available value in the direction of positive infinity.
- Rounding towards –∞: An inexact result is rounded to the nearest available value in the direction of negative infinity.

(1) Rounding to the nearest value is specified as the default mode and returns the most accurate value.

(2) Modes such as rounding towards 0, rounding towards +∞, and rounding towards –∞ are used to ensure precision when interval arithmetic is employed.

CV flag (Invalid operation cause flag), CO flag (Overflow cause flag), CZ flag (Division-by-zero cause flag), CU flag (Underflow cause flag), CX flag (Inexact cause flag), and CE flag (Unimplemented processing cause flag)

Floating-point exceptions include the five specified in the IEEE754 standard, namely overflow, underflow, inexact, division-by-zero, and invalid operation. For a further floating-point exception that is generated upon detection of unimplemented processing, the corresponding flag (CE) is set to 1.

- The bit that has been set to 1 is cleared to 0 when the FPU instruction is executed.
- When 0 is written to the bit by the MVTC and POPC instructions, the bit is set to 0; the bit retains the previous value when 1 is written by the instruction.

DN bit (0 flush bit of denormalized number)

When this bit is set to 0, a denormalized number is handled as a denormalized number.
When this bit is set to 1, a denormalized number is handled as 0.

EV bit (Invalid operation exception enable bit), EO bit (Overflow exception enable bit), EZ bit (Division-by-zero exception enable bit), EU bit (Underflow exception enable bit), and EX bit (Inexact exception enable bit)

When any of five floating-point exceptions specified in the IEEE754 standard is generated by the FPU instruction, the bit decides whether the CPU will start handling the exception. When the bit is set to 0, the exception handling is masked; when the bit is set to 1, the exception handling is enabled.

FV flag (Invalid operation flag), FO flag (Overflow flag), FZ flag (Division-by-zero flag), FU flag (Underflow flag), and FX flag (Inexact flag)

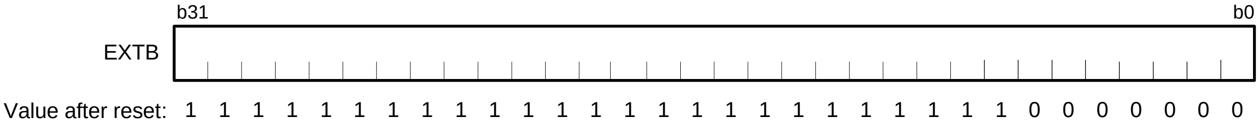
While the exception handling enable bit (Ej) is 0 (exception handling is masked), if any of five floating-point exceptions specified in the IEEE754 standard is generated, the corresponding bit is set to 1.

- When Ej is 1 (exception handling is enabled), the value of the flag remains.
- When the corresponding flag is set to 1, it remains 1 until it is cleared to 0 by software. (Accumulation flag)

FS flag (Floating-point error summary flag)

This bit reflects the logical OR of the FU, FZ, FO, and FV flags.

1.2.2.9 Exception Vector Table Register (EXTB)


b31
b0
EXTB
Value after reset: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0

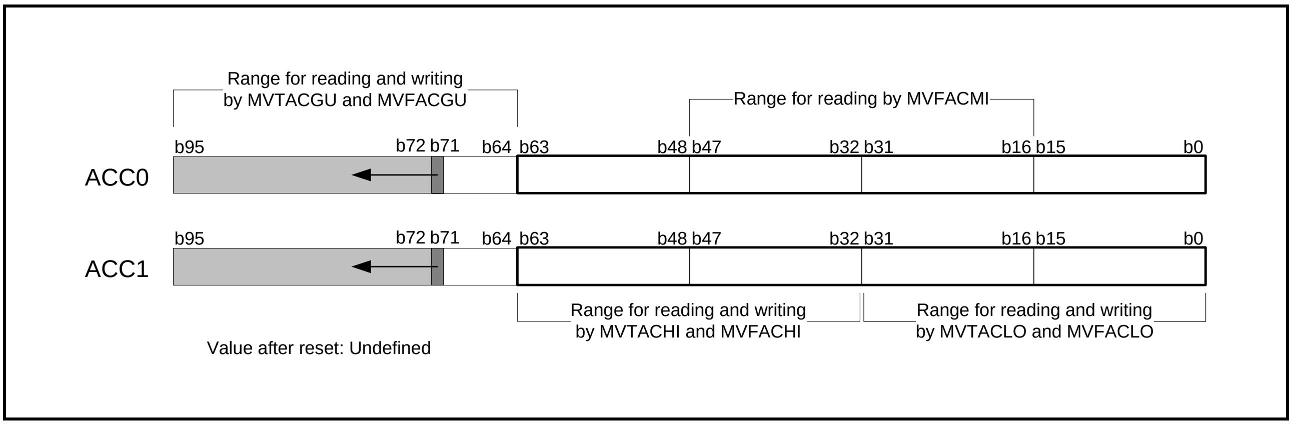
The exception table register (EXTB) specifies the address where the exception vector table starts.

1.2.3 Accumulator

The accumulator (ACC0 or ACC1) is a 72-bit register used for DSP instructions. The accumulator is handled as a 96-bit register for reading and writing. At this time, when bits 95 to 72 of the accumulator are read, the value where the value of bit 71 is sign extended is read. Writing to bits 95 to 72 of the accumulator is ignored. ACC0 is also used for the multiply and multiply-and-accumulate instructions; EMUL, EMULU, FMUL, MUL, and RMPA, in which case the prior value in ACC0 is modified by execution of the instruction.

Use the MVTACGU, MVTACHI, and MVTACLO instructions for writing to the accumulator. The MVTACGU, MVTACHI, and MVTACLO instructions write data to bits 95 to 64, the higher-order 32 bits (bits 63 to 32), and the lower-order 32 bits (bits 31 to 0), respectively.

Use the MVFACGU, MVFACHI, MVFACMI, and MVFACLO instructions for reading data from the accumulator. The MVFACGU, MVFACHI, MVFACMI, and MVFACLO instructions read data from the guard bits (bits 95 to 64), higher-order 32 bits (bits 63 to 32), the middle 32 bits (bits 47 to 16), and the lower-order 32 bits (bits 31 to 0), respectively.


Range for reading and writing by MVTACGU and MVFACGU
Range for reading by MVFACMI
b95
b72 b71
b64 b63
b48 b47
b32 b31
b16 b15
b0
ACC0
b95
b72 b71
b64 b63
b48 b47
b32 b31
b16 b15
b0
ACC1
Range for reading and writing by MVTACHI and MVFACHI
Range for reading and writing by MVTACLO and MVFACLO
Value after reset: Undefined

Note: The value of bit 71 is sign extended for bits 95 to 72 and the extended value is always read. Writing to this area is ignored.

1.3 Floating-Point Exceptions

Floating-point exceptions include the five specified in the IEEE754 standard, namely overflow, underflow, inexact, division-by-zero, and invalid operation, and a further floating-point exception that is generated on the detection of unimplemented processing. The following is an outline of the events that cause floating-point exceptions.

1.3.1 Overflow

An overflow occurs when the absolute value of the result of an arithmetic operation is greater than the range of values that can be represented in the floating-point format. Table 1.1 lists the results of operations when an overflow exception occurs.

Table 1.1 Operation Results When an Overflow Exception Has Occurred

		Operation Result (Value in the Destination Register)	
Floating-Point Rounding Mode	**Sign of Result**	**EO = 0**	**EO = 1**
Rounding towards –∞	+	+MAX	No change
	–	–∞	
Rounding towards +∞	+	+∞	
	–	–MAX	
Rounding towards 0	+	+MAX	
	–	–MAX	
Rounding to the nearest value	+	+∞	
	–	–∞	

Note: An inexact exception will be generated when an overflow error occurs while EO = 0.

1.3.2 Underflow

An underflow occurs when the absolute value of the result of an arithmetic operation is smaller than the range of normalized values that can be represented in the floating-point format. (However, this does not apply when the result is 0.) Table 1.2 lists the results of operations when an underflow exception occurs.

Table 1.2 Operation Results When an Underflow Exception Has Occurred

Operation Result (Value in the Destination Register)	
EU = 0	**EU = 1**
DN = 0: No change. (An unimplemented processing exception is generated.)	No change
DN = 1: The value of 0 is returned.	

1.3.3 Inexact

An inexact exception occurs when the result of a hypothetical calculation with infinite precision differs from the actual result of the operation. Table 1.3 lists the conditions leading to an inexact exception and the results of operations.

Table 1.3 Conditions Leading to an Inexact Exception and the Operation Results

	Operation Result (Value in the Destination Register)	
Occurrence Condition	**EX = 0**	**EX = 1**
An overflow exception has occurred while overflow exceptions are masked.	Refer to table 1.1, Operation Results When an Overflow Exception Has Occurred	No change
Rounding has been produced.	Value after rounding	

Notes: 1. An inexact exception will not be generated when an underflow error occurs.
2. An inexact exception will not be generated when an overflow exception occurs while overflow exceptions are enabled, regardless of the rounding generation.

1.3.4 Division-by-Zero

Dividing a non-zero finite number by zero produces a division-by-zero exception. Table 1.4 lists the results of operations that have led to a division-by-zero exception.

Table 1.4 Operation Results When a Division-by Zero Exception Has Occurred

	Operation Result (Value in the Destination Register)	
Dividend	**EZ = 0**	**EZ = 1**
Non-zero finite number	±∞ (the sign bit is the logical exclusive or of the sign bits of the divisor and dividend)	No change

Note that a division-by zero exception does not occur in the following situations.

Dividend	Result
0	An invalid operation exception is generated.
∞	No exception is generated. The result is ∞.
Denormalized number (DN = 0)	An unimplemented processing exception is generated.
QNaN	No exception is generated. The result is QNaN.
SNaN	An invalid operation exception is generated.

1.3.5 Invalid Operation

Executing an invalid operation produces an invalid exception. Table 1.5 lists the conditions leading to an invalid exception and the results of operations.

Table 1.5 Conditions Leading to an Invalid Exception and the Operation Results

	Operation Result (Value in the Destination Register)	
Occurrence Condition	**EV = 0**	**EV = 1**
Operation on SNaN operands	QNaN	No change
+∞+(–∞), +∞–(+∞), –∞–(–∞)		
0 × ∞		
0 ÷ 0, ∞ ÷ ∞		
Square root operation on numbers smaller than 0		
Overflow in integer conversion or attempting integer conversion of NaN or ∞ when executing FTOI or ROUND instruction	The return value is 7FFFFFFFh when the sign bit before conversion was 0 and 80000000h when the sign bit before conversion was 1.	
Overflow in integer conversion or attempting integer conversion of NaN or ∞ when executing FTOU instruction	The return value is FFFFFFFFh when the sign bit before conversion was 0 and 00000000h when the sign bit before conversion was 1.	
Comparison of SNaN operands	No destination	

Legend

NaN (Not a Number): Not a Number

SNaN (Signaling NaN): SNaN is a kind of NaN where the most significant bit in the mantissa part is 0.
Using an SNaN as a source operand in an operation generates an invalid operation. Using an SNaN as the initial value of a variable facilitates the detection of bugs in programs. Note that the hardware will not generate an SNaN.

QNaN (Quiet NaN): QNaN is a kind of NaN where the most significant bit in the mantissa part is 1.
Using a QNaN as a source operand in an operation (except in a comparison or format conversion) does not generate an invalid operation. Since a QNaN is propagated through operations, just checking the result without performing exception handling enables the debugging of programs. Note that hardware operations can generate a QNaN.

Table 1.6 lists the rules for generating QNaNs as the results of operations.

Table 1.6 Rules for Generating QNaNs

Source Operands	Operation Result (Value in the Destination Register)
An SNaN and a QNaN	The SNaN source operand converted into a QNaN
Two SNaNs	dest converted into a QNaN
Two QNaNs	dest
An SNaN and a real value	The SNaN source operand converted into a QNaN
A QNaN and a real value	The QNaN source operand
Neither source operand is an NaN and an invalid operation exception is generated	7FFFFFFFh

Note: The SNaN is converted into a QNaN while the most significant bit in the mantissa part is 1.

1.3.6 Unimplemented Processing

An unimplemented processing exception occurs when DN = 0 and a denormalized number is given as an operand, or when an underflow exception is generated as the result of an operation with DN = 0. An unimplemented processing exception will not occur with DN = 1.

There is no enable bit to mask an unimplemented processing exception, so this processing exception cannot be masked. The destination register remains as is.

1.4 Processor Mode

The RXv2 CPU supports two processor modes, supervisor and user. These processor modes and the memory protection function enable the realization of a hierarchical CPU resource protection and memory protection mechanism. Each processor mode imposes a level on rights of access to memory and the instructions that can be executed. Supervisor mode carries greater rights than user mode. The initial state after a reset is supervisor mode.

1.4.1 Supervisor Mode

In supervisor mode, all CPU resources are accessible and all instructions are available. However, writing to the processor mode select bit (PM) in the processor status word (PSW) by executing an MVTC or POPC instruction will be ignored. For details on how to write to the PM bit, refer to 1.2.2.4, Processor Status Word (PSW).

1.4.2 User Mode

In user mode, write access to the CPU resources listed below is restricted. The restriction applies to any instruction capable of write access.

- Some bits (bits IPL[3:0], PM, U, and I) in the processor status word (PSW)
- Interrupt stack pointer (ISP)
- Interrupt table register (INTB)
- Backup PSW (BPSW)
- Backup PC (BPC)
- Fast interrupt vector register (FINTV)
- Exception table register (EXTB)

1.4.3 Privileged Instruction

Privileged instructions can only be executed in supervisor mode. Executing a privileged instruction in user mode produces a privileged instruction exception. Privileged instructions include the RTFI, MVTIPL, RTE, and WAIT instructions.

1.4.4 Switching Between Processor Modes

Manipulating the processor mode select bit (PM) in the processor status word (PSW) switches the processor mode. However, rewriting the PM bit by executing an MVTC or POPC instruction is prohibited. Switch the processor mode by following the procedures described below.

(1) Switching from user mode to supervisor mode
After an exception has been generated, the PM bit in the PSW is set to 0 and the CPU switches to supervisor mode. The hardware pre-processing is executed in supervisor mode. The state of the processor mode before the exception was generated is retained in the PM bit in the copy of the PSW that is saved on the stack.

(2) Switching from supervisor mode to user mode
Executing an RTE instruction when the value of the copy of the PM bit in the PSW that has been preserved on the stack is "1" or an RTFI instruction when the value of the copy of the PM bit in the PSW that has been preserved in the backup PSW (BPSW) is "1" causes a transition to user mode. In the transition to user mode, the value of the stack pointer designation bit (the U bit in the PSW) becomes "1".

1.5 Data Types

The RXv2 CPU can handle four types of data: integer, floating-point, bit, and string.

1.5.1 Integer

An integer can be signed or unsigned. For signed integers, negative values are represented by two's complements.

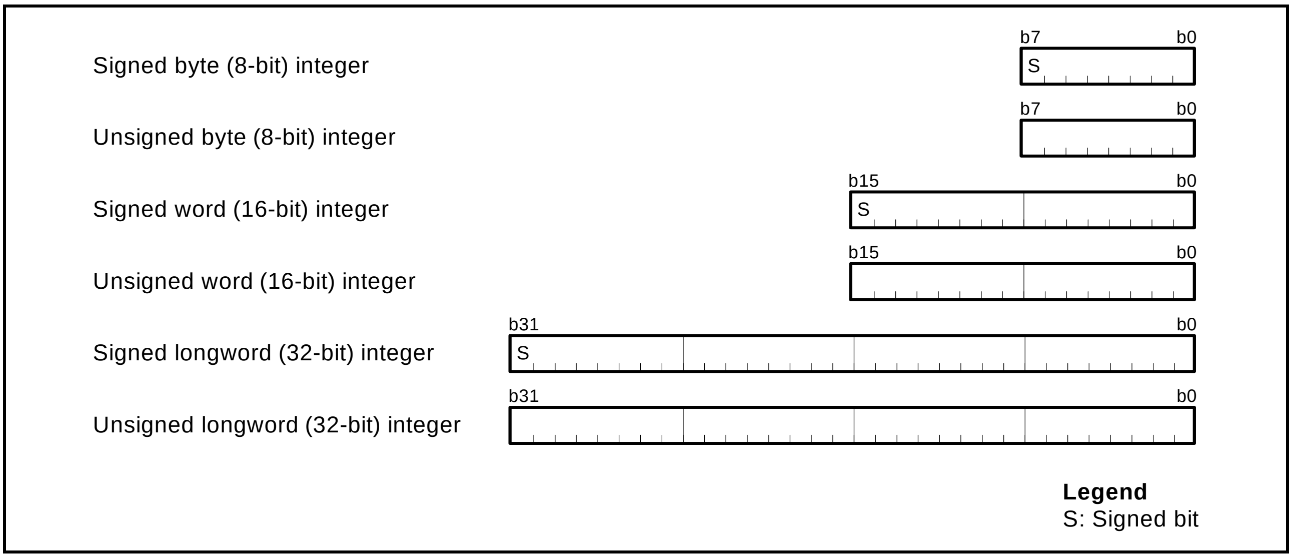

Signed byte (8-bit) integer
b7
b0
S
Unsigned byte (8-bit) integer
b7
b0
Signed word (16-bit) integer
b15
b0
S
Unsigned word (16-bit) integer
b15
b0
Signed longword (32-bit) integer
b31
b0
S
Unsigned longword (32-bit) integer
b31
b0
Legend
S: Signed bit

Figure 1.2 Integer

1.5.2 Floating-Point

Floating-point support is for the single-precision floating-point type specified in the IEEE754 standard; operands of this type can be used in eleven floating-point operation instructions: FADD, FCMP, FDIV, FMUL, FSUB, FTOI, ITOF, ROUND, FTOU, UTOF, and FSQRT.

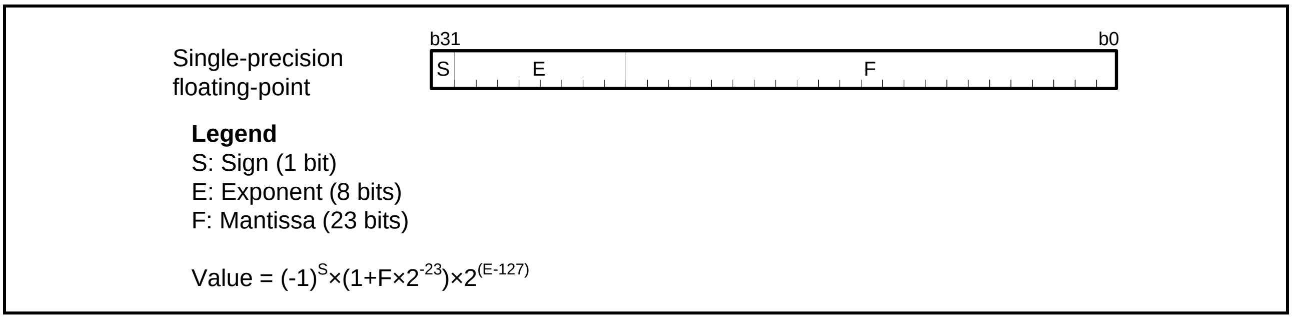

Single-precision floating-point
b31
b0
S
E
F
Legend
S: Sign (1 bit)
E: Exponent (8 bits)
F: Mantissa (23 bits)
Value = $(-1)^{S} \times (1+F \times 2^{-23}) \times 2^{(E-127)}$

Figure 1.3 Floating-Point

The floating-point format supports the values listed below.

- $0 < E < 255$ (normal numbers)
- $E = 0$ and $F = 0$ (signed zero)
- $E = 0$ and $F > 0$ (denormalized numbers)*
- $E = 255$ and $F = 0$ (infinity)
- $E = 255$ and $F > 0$ (NaN: Not-a-Number)

Note: * The number is treated as 0 when the DN bit in the FPSW is 1. When the DN bit is 0, an unimplemented processing exception is generated.

1.5.3 Bitwise Operations

Five bit-manipulation instructions are provided for bitwise operations: BCLR, BM*Cnd*, BNOT, BSET, and BTST.

A bit in a register is specified as the destination register and a bit number in the range from 31 to 0.

A bit in memory is specified as the destination address and a bit number from 7 to 0. The addressing modes available to specify addresses are register indirect and register relative.

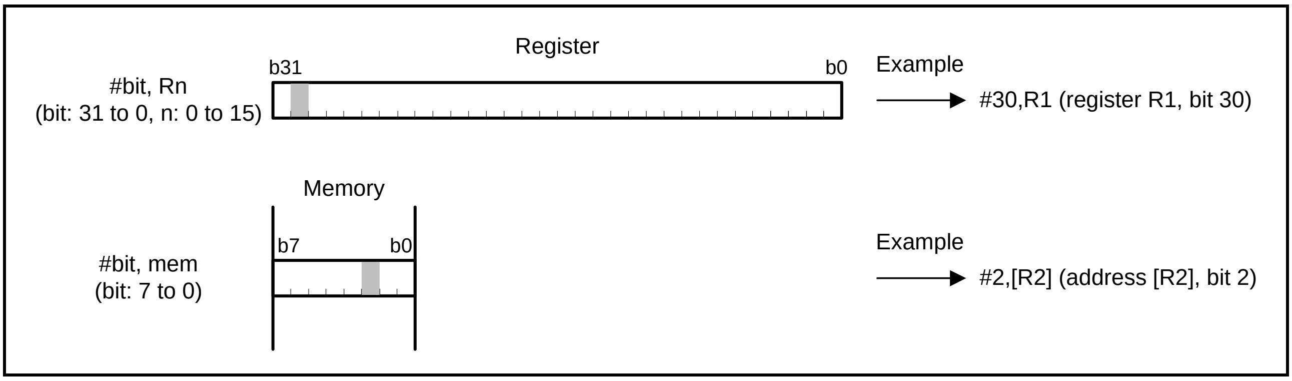

Register
#bit, Rn
(bit: 31 to 0, n: 0 to 15)
b31
b0
Example
#30,R1 (register R1, bit 30)
Memory
#bit, mem
(bit: 7 to 0)
b7
b0
Example
#2,[R2] (address [R2], bit 2)

Figure 1.4 Bit

1.5.4 Strings

The string data type consists of an arbitrary number of consecutive byte (8-bit), word (16-bit), or longword (32-bit) units. Seven string manipulation instructions are provided for use with strings: SCMPU, SMOVB, SMOVF, SMOVU, SSTR, SUNTIL, and SWHILE.

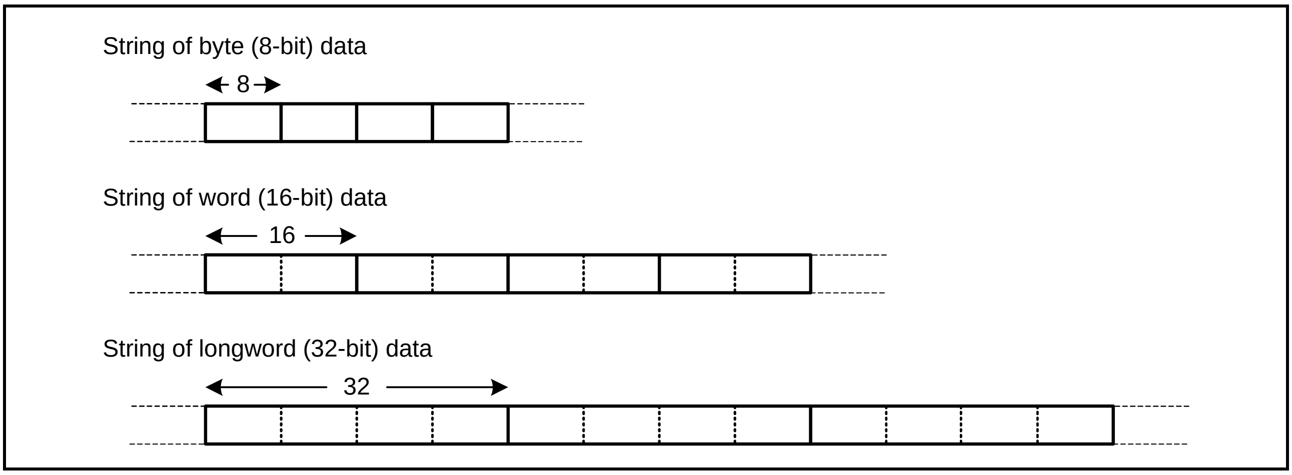

String of byte (8-bit) data
8
String of word (16-bit) data
16
String of longword (32-bit) data
32

Figure 1.5 String

1.6 Data Arrangement

1.6.1 Data Arrangement in Registers

Figure 1.6 shows the relation between the sizes of registers and bit numbers.

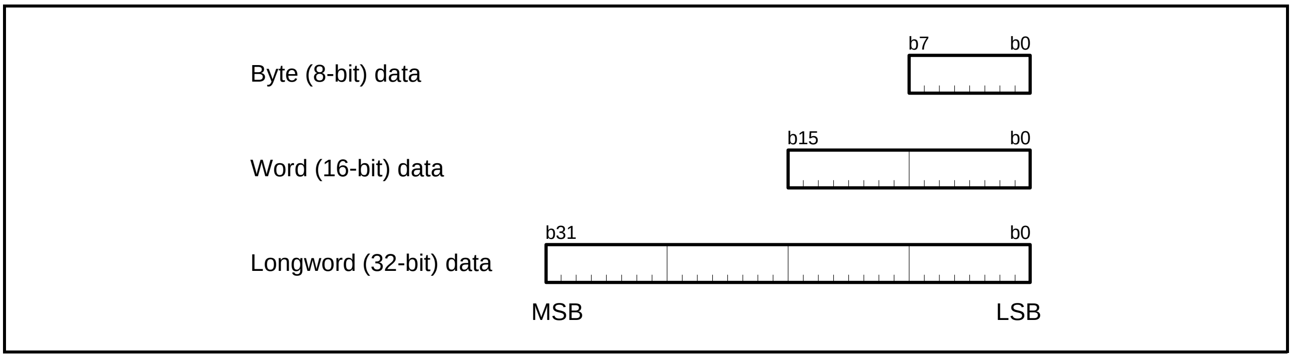
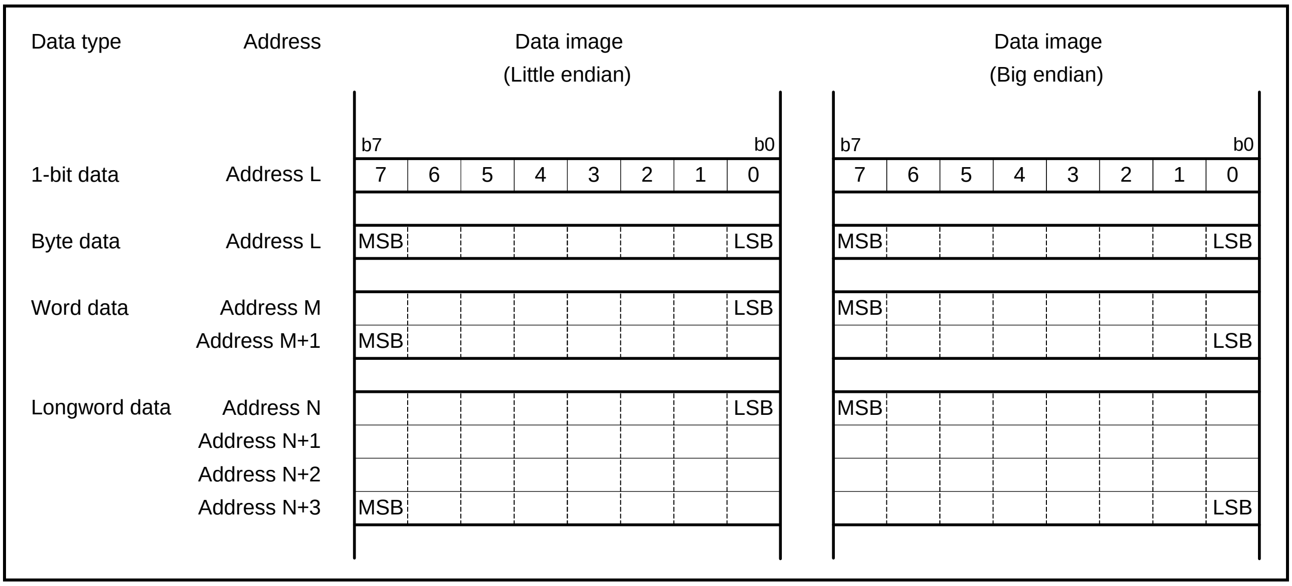


Figure 1.6 Data Arrangement in Registers

1.6.2 Data Arrangement in Memory

Data in memory have three sizes; byte (8-bit), word (16-bit), and longword (32-bit). The data arrangement is selectable as little endian or big endian. Figure 1.7 shows the arrangement of data in memory.



Data type	Address	Data image (Little endian)								Data image (Big endian)							
		b7							b0	b7							b0
1-bit data	Address L	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte data	Address L	MSB							LSB	MSB							LSB
Word data	Address M								LSB	MSB							
	Address M+1	MSB															LSB
Longword data	Address N								LSB	MSB							
	Address N+1																
	Address N+2																
	Address N+3	MSB															LSB

Figure 1.7 Data Arrangement in Memory

1.7 Vector Table

There are two types of vector table: exception and interrupt. Each vector in the vector table consists of four bytes and specifies the address where the corresponding exception handling routine starts.

1.7.1 Exception Vector Table

In the exception vector table, the individual vectors for the privileged instruction exception, access exception, undefined instruction exception, floating-point exception, non-maskable interrupt, and reset are allocated to the 128-byte area where the value indicated by the exception table register (EXTB) is used as the starting address (ExtBase). Note, however, that the reset vector is always allocated to FFFFFFFCh.

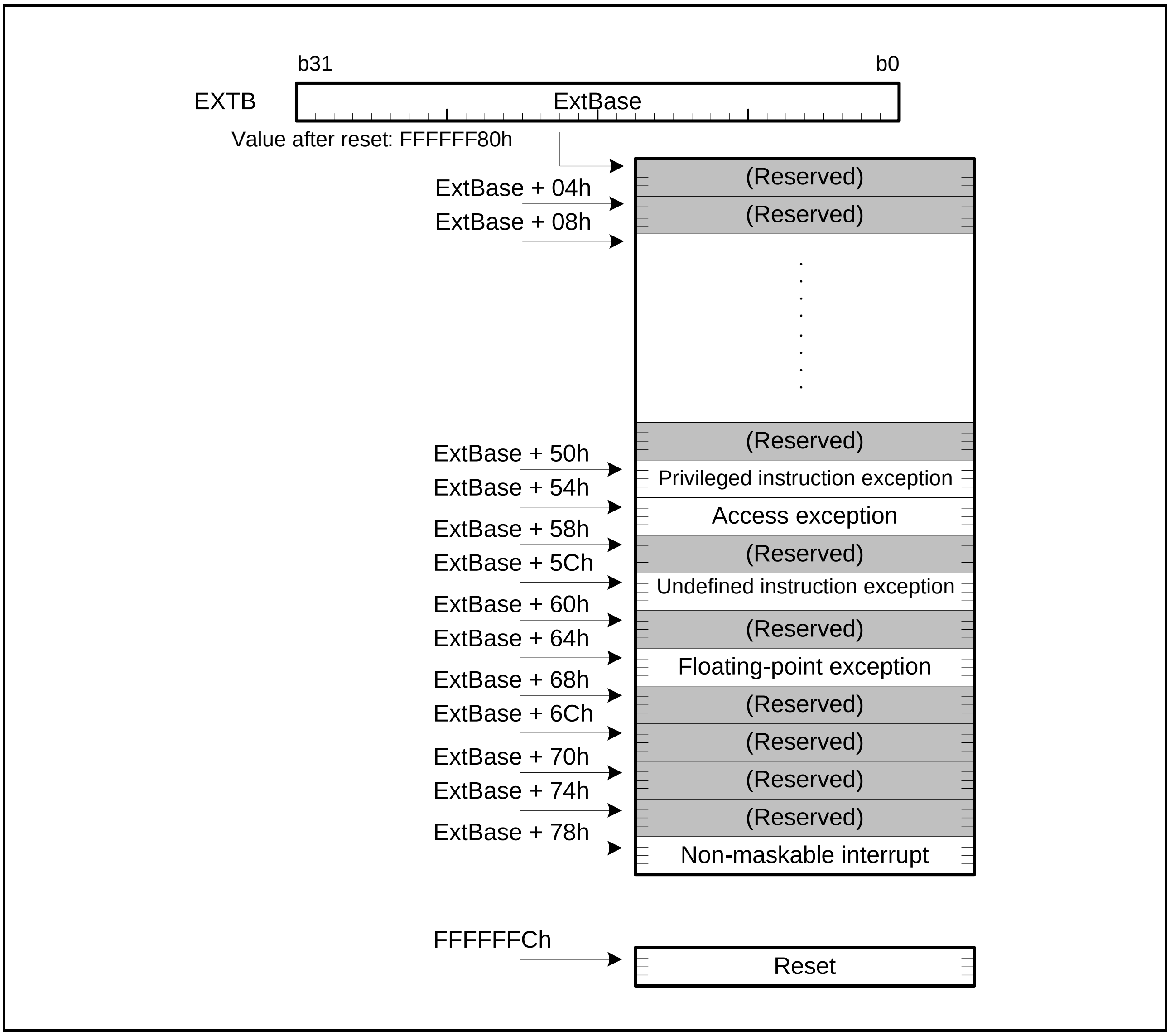


Figure 1.8 Exception Vector Table

1.7.2 Interrupt Vector Table

The address where the interrupt vector table is placed can be adjusted. The table is a 1,024-byte region that contains all vectors for unconditional traps and interrupts and starts at the address (IntBase) specified in the interrupt table register (INTB). Figure 1.9 shows the interrupt vector table.

Each vector in the interrupt vector table has a vector number from 0 to 255. Each of the INT instructions, which act as the sources of unconditional traps, is allocated to the vector that has the same number as that of the instruction itself (from 0 to 255). The BRK instruction is allocated to the vector with number 0. Furthermore, vector numbers within the set from 0 to 255 may also be allocated to other interrupt sources on a per-product basis.

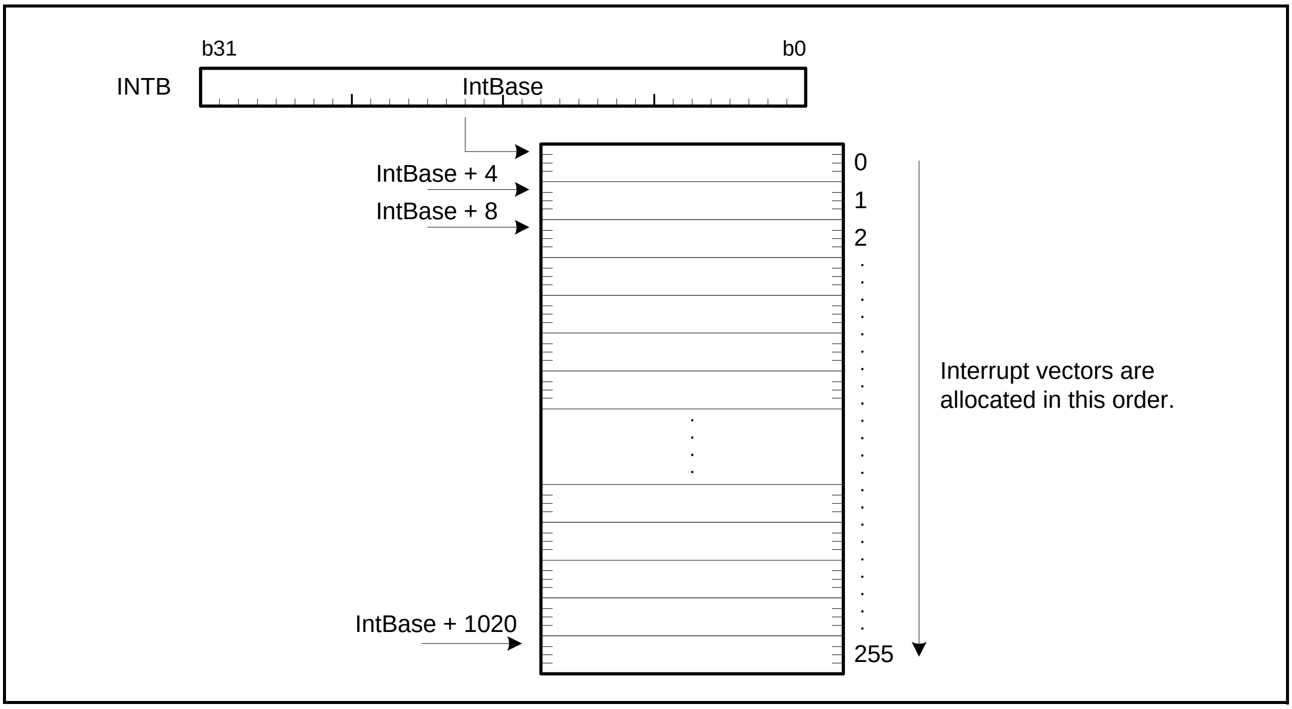


Figure 1.9 Interrupt Vector Table

1.8 Address Space

The address space of the RXv2 CPU is the 4 Gbyte range from address 0000 0000h to address FFFF FFFFh. Program and data regions taking up to a total of 4 Gbytes are linearly accessible. The address space of the RXv2 CPU is depicted in figure 1.10. For all regions, the designation may differ with the product and operating mode. For details, see the hardware manuals for the respective products.

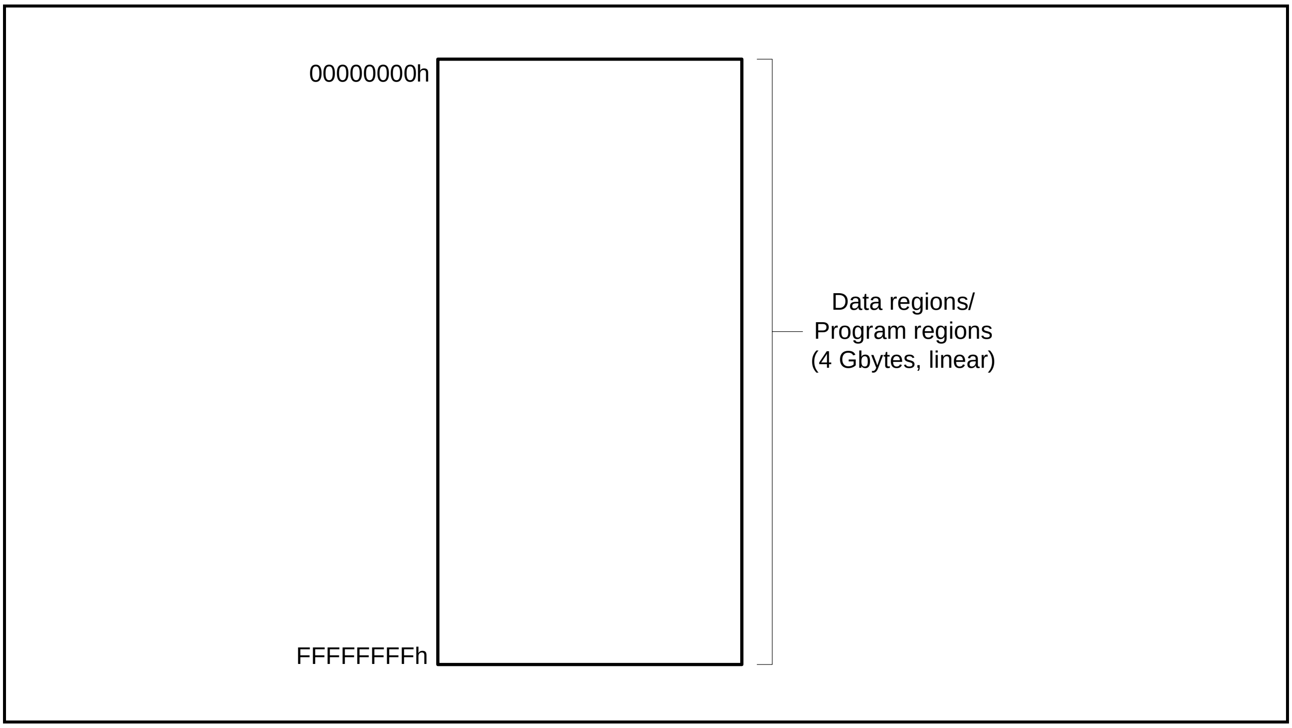


Figure 1.10 Address Space

Section 2 Addressing Modes

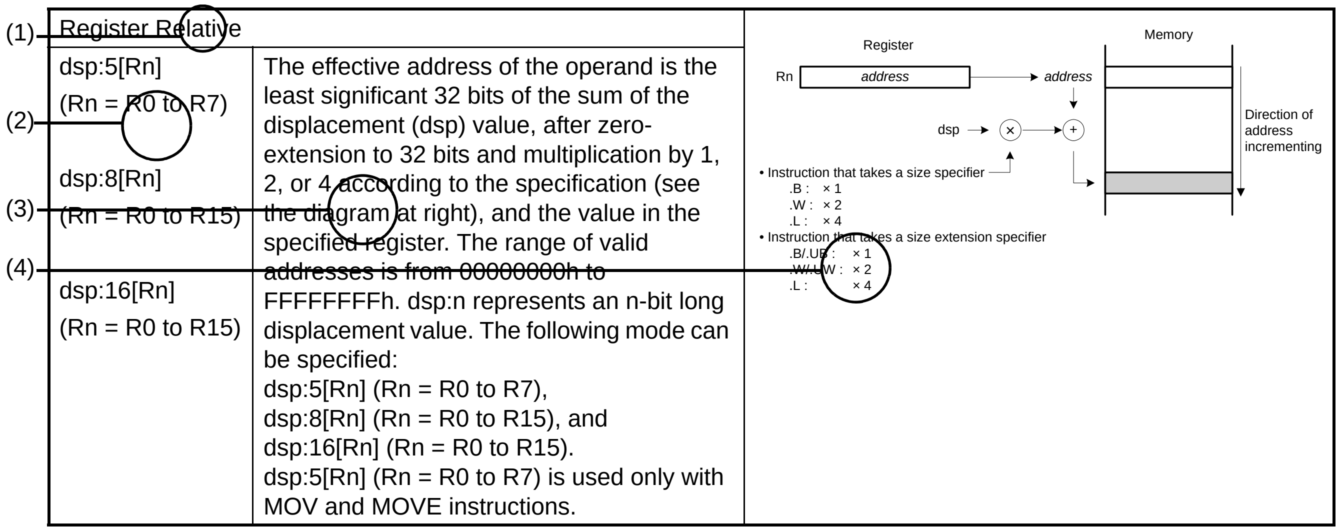
The following is a description of the notation and operations of each addressing mode.

There are eleven types of addressing mode.

- Immediate
- Register direct
- Register indirect
- Register relative
- Post-increment register indirect
- Pre-decrement register indirect
- Indexed register indirect
- Control register direct
- PSW direct
- Program counter relative
- Accumulator direct

2.1 Guide to This Section

The following sample shows how the information in this section is presented.



Register Relative		
dsp:5[Rn] (Rn = R0 to R7) dsp:8[Rn] (Rn = R0 to R15) dsp:16[Rn] (Rn = R0 to R15)	The effective address of the operand is the least significant 32 bits of the sum of the displacement (dsp) value, after zero-extension to 32 bits and multiplication by 1, 2, or 4 according to the specification (see the diagram at right), and the value in the specified register. The range of valid addresses is from 00000000h to FFFFFFFFh. dsp:n represents an n-bit long displacement value. The following mode can be specified: dsp:5[Rn] (Rn = R0 to R7), dsp:8[Rn] (Rn = R0 to R15), and dsp:16[Rn] (Rn = R0 to R15). dsp:5[Rn] (Rn = R0 to R7) is used only with MOV and MOVE instructions.	

(1)
(2)
(3)
(4)
Register
Memory
Rn
address
address
dsp
×
+
Direction of address incrementing
• Instruction that takes a size specifier
.B : × 1
.W : × 2
.L : × 4
• Instruction that takes a size extension specifier
.B/.UB : × 1
.W/.UW : × 2
.L : × 4

(1) Name

The name of the addressing mode is given here.

(2) Symbolic notation

This notation represents the addressing mode.
:8 or :16 represents the number of valid bits just before an instruction in this addressing mode is executed. This symbolic notation is added in the manual to represent the number of valid bits, and is not included in the actual program.

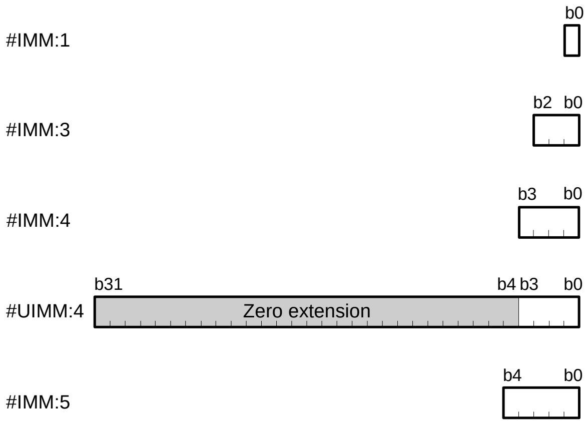
(3) Description

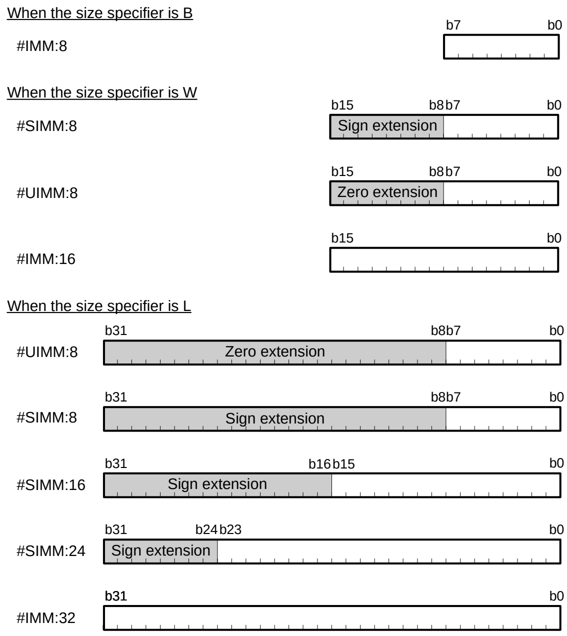
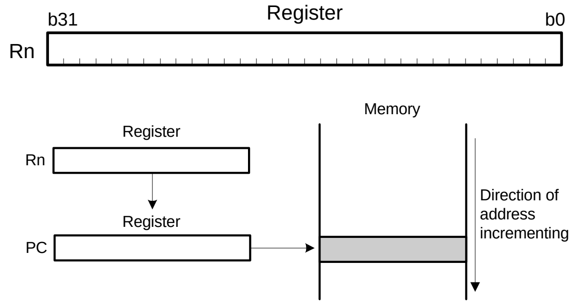
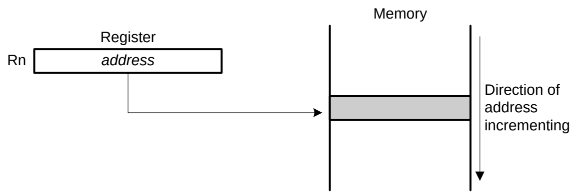
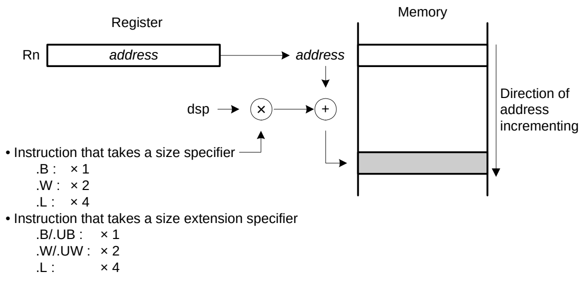
The operation and effective address range are described here.

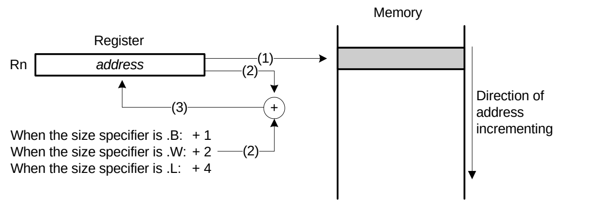
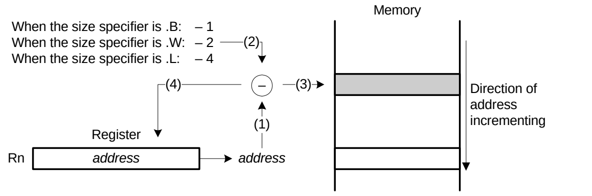
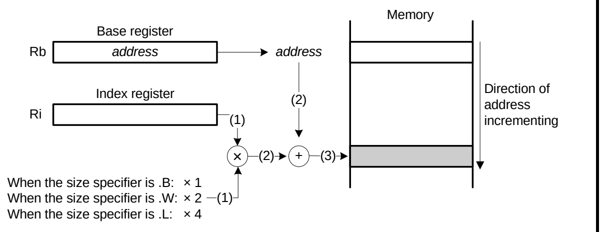
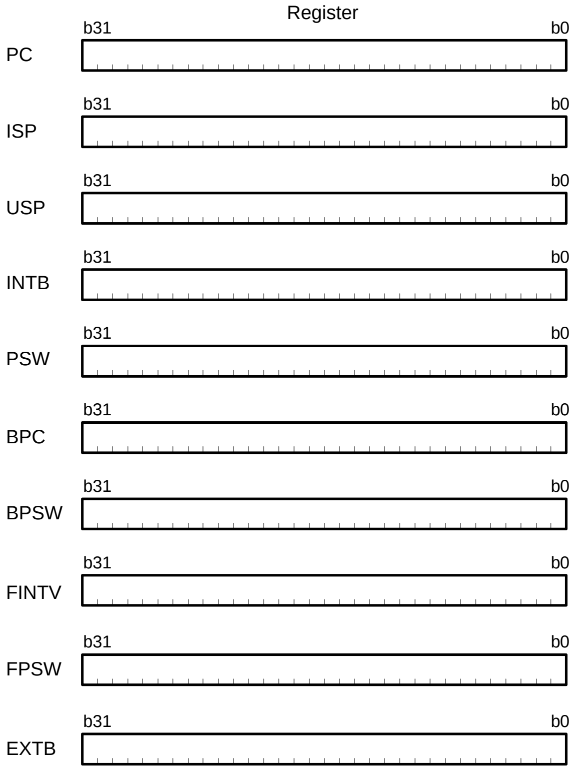
(4) Operation diagram

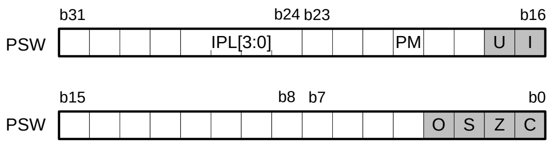
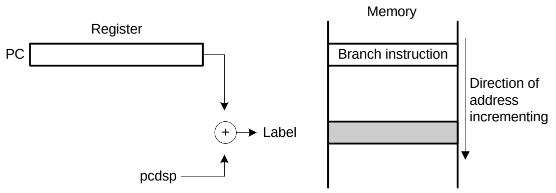
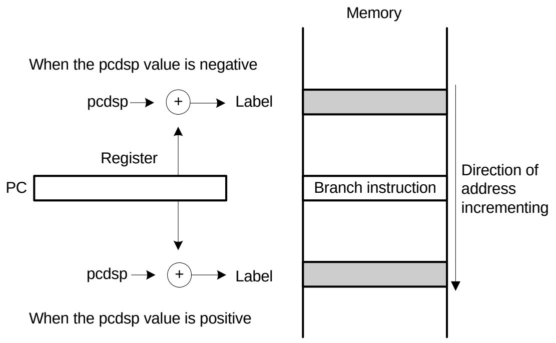
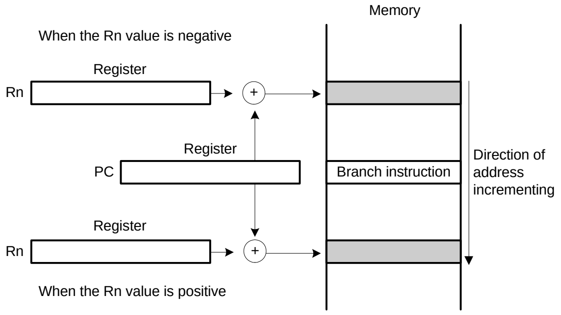
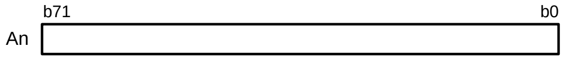
The operation of the addressing mode is illustrated here.

2.2 Addressing Modes

Immediate		
#IMM:1 #IMM:3 #IMM:4 #UIMM:4 #IMM:5	#IMM:1 The operand is the 1-bit immediate value indicated by #IMM. This addressing mode is used to specify the source for the RACW instruction. #IMM:3 The operand is the 3-bit immediate value indicated by #IMM. This addressing mode is used to specify the bit number for the bit manipulation instructions: BCLR, BM*Cnd*, BNOT, BSET, and BTST. #IMM:4 The operand is the 4-bit immediate value indicated by #IMM. This addressing mode is used to specify the interrupt priority level for the MVTIPL instruction. #UIMM:4 The operand is the 4-bit immediate value indicated by #UIMM after zero extension to 32 bits. This addressing mode is used to specify sources for ADD, AND, CMP, MOV, MUL, OR, and SUB instructions. #IMM:5 The operand is the 5-bit immediate value indicated by #IMM. This addressing mode is used in the following ways: - to specify the bit number for the bit-manipulation instructions: BCLR, BM*Cnd*, BNOT, BSET, and BTST; - to specify the number of bit places of shifting in certain arithmetic/logic instructions: SHAR, SHLL, and SHLR; and - to specify the number of bit places of rotation in certain arithmetic/logic instructions: ROTL and ROTR.	 b0 #IMM:1 b2 b0 #IMM:3 b3 b0 #IMM:4 b31 b4 b3 b0 #UIMM:4 Zero extension b4 b0 #IMM:5

Immediate		 When the size specifier is B b7 b0 #IMM:8 When the size specifier is W b15 b8 b7 b0 #SIMM:8 Sign extension b15 b8 b7 b0 #UIMM:8 Zero extension b15 b0 #IMM:16 When the size specifier is L b31 b8 b7 b0 #UIMM:8 Zero extension b31 b8 b7 b0 #SIMM:8 Sign extension b31 b16 b15 b0 #SIMM:16 Sign extension b31 b24 b23 b0 #SIMM:24 Sign extension b31 b0 #IMM:32
#IMM:8 #SIMM:8 #UIMM:8 #IMM:16 #SIMM:16 #SIMM:24 #IMM:32	The operand is the value specified by the immediate value. In addition, the operand will be the result of zero-extending or sign-extending the immediate value when it is specified by #UIMM or #SIMM. #IMM:n, #UIMM:n, and #SIMM:n represent n-bit long immediate values. For the range of IMM, refer to section 2.2.1, Ranges for Immediate Values.	
Register Direct		 b31 Register b0 Rn Memory Register Rn Register PC Direction of address incrementing
Rn (Rn = R0 to R15)	The operand is the specified register. In addition, the Rn value is transferred to the program counter (PC) when this addressing mode is used with JMP and JSR instructions. The range of valid addresses is from 00000000h to FFFFFFFFh. Rn (Rn = R0 to R15) can be specified.	
Register Indirect		 Memory Register Rn address Direction of address incrementing
[Rn] (Rn = R0 to R15)	The value in the specified register is the effective address of the operand. The range of valid addresses is from 00000000h to FFFFFFFFh. [Rn] (Rn = R0 to R15) can be specified.	
Register Relative		 Memory Register Rn address address dsp × + Direction of address incrementing • Instruction that takes a size specifier .B : × 1 .W : × 2 .L : × 4 • Instruction that takes a size extension specifier .B/.UB : × 1 .W/.UW : × 2 .L : × 4
dsp:5[Rn] (Rn = R0 to R7) dsp:8[Rn] (Rn = R0 to R15) dsp:16[Rn] (Rn = R0 to R15)	The effective address of the operand is the least significant 32 bits of the sum of the displacement (dsp) value, after zero-extension to 32 bits and multiplication by 1, 2, or 4 according to the specification (see the diagram at right), and the value in the specified register. The range of valid addresses is from 00000000h to FFFFFFFFh. dsp:n represents an n-bit long displacement value. The following mode can be specified: dsp:5[Rn] (Rn = R0 to R7), dsp:8[Rn] (Rn = R0 to R15), and dsp:16[Rn] (Rn = R0 to R15). dsp:5[Rn] (Rn = R0 to R7) is used only with MOV and MOVE instructions.	

Post-increment Register Indirect		 Memory Register Rn address (1) (2) (3) + When the size specifier is .B: + 1 When the size specifier is .W: + 2 When the size specifier is .L: + 4 (2) Direction of address incrementing
[Rn+] (Rn = R0 to R15)	The value in the specified register is the effective address of the operand. The range of valid addresses is from 00000000h to FFFFFFFFh. After the operation, 1, 2, or 4 is added to the value in the specified register according to the size specifier: .B, .W, or .L. This addressing mode is used with MOV and MOVU instructions.	
Pre-decrement Register Indirect		 Memory When the size specifier is .B: – 1 When the size specifier is .W: – 2 When the size specifier is .L: – 4 (2) (4) – (3) (1) Register Rn address address Direction of address incrementing
[–Rn] (Rn = R0 to R15)	According to the size specifier: .B, .W, or .L, 1, 2, or 4 is subtracted from the value in the specified register. The value after the operation is the effective address of the operand. The range of valid addresses is from 00000000h to FFFFFFFFh. This addressing mode is used with MOV and MOVU instructions.	
Indexed Register Indirect		 Memory Base register Rb address address Index register Ri (1) (2) × (2) + (3) When the size specifier is .B: × 1 When the size specifier is .W: × 2 When the size specifier is .L: × 4 (1) Direction of address incrementing
[Ri, Rb] (Ri = R0 to R15, Rb = R0 to R15)	The effective address of the operand is the least significant 32 bits of the sum of the value in the index register (Ri), multiplied by 1, 2, or 4 according to the size specifier: .B, .W, or .L, and the value in the base register (Rb). The range of valid addresses is from 00000000h to FFFFFFFFh. This addressing mode is used with MOV and MOVU instructions.	
Control Register Direct		 Register b31 b0 PC ISP USP INTB PSW BPC BPSW FINTV FPSW EXTB
PC ISP USP INTB PSW BPC BPSW FINTV FPSW EXTB	The operand is the specified control register. This addressing mode is used with MVFC, MVTC, POPC, and PUSHC instructions. The PC is only selectable as the src operand of MVFC and PUSHC instructions.	

PSW Direct		 b31 b24 b23 b16 PSW IPL[3:0] PM U I b15 b8 b7 b0 PSW O S Z C
C Z S O I U	The operand is the specified flag or bit. This addressing mode is used with CLRPSW and SETPSW instructions.	
Program Counter Relative		 Register PC pcdsp + Label Memory Branch instruction Direction of address incrementing
pcdsp:3	When the branch distance specifier is .S, the effective address is the least significant 32 bits of the unsigned sum of the value in the program counter (PC) and the displacement (pcdsp) value. The range of the branch is from 3 to 10. The range of valid addresses is from 00000000h to FFFFFFFFh. This addressing mode is used with B*Cnd* (where *Cnd*==EQ/Z or NE/NZ) and BRA instructions.	
pcdsp:8 pcdsp:16 pcdsp:24	When the branch distance specifier is .B, .W, or .A, the effective address is the signed sum of the value in the program counter (PC) and the displacement (pcdsp) value. The range of pcdsp depends on the branch distance specifier. For .B: –128 ≤ pcdsp:8 ≤ 127 For .W: –32768 ≤ pcdsp:16 ≤ 32767 For .A: –8388608 ≤ pcdsp:24 ≤ 8388607 The range of valid addresses is from 00000000h to FFFFFFFFh. When the branch distance specifier is .B, this addressing mode is used with B*Cnd* and BRA instructions. When the branch distance specifier is .W, this addressing mode is used with B*Cnd* (where *Cnd*==EQ/Z or NE/NZ), BRA, and BSR instructions. When the branch distance specifier is .A, this addressing mode is used with BRA and BSR instructions.	 Memory When the pcdsp value is negative pcdsp + Label Register PC Branch instruction Direction of address incrementing pcdsp + Label When the pcdsp value is positive
Rn (Rn = R0 to R15)	The effective address is the signed sum of the value in the program counter (PC) and the Rn value. The range of the Rn value is from –2147483648 to 2147483647. The range of valid addresses is from 00000000h to FFFFFFFFh. This addressing mode is used with BRA(.L) and BSR(.L) instructions.	 Memory When the Rn value is negative Register Rn + Register PC Branch instruction Direction of address incrementing Register Rn + When the Rn value is positive
Accumulator Direct		
A0, A1 (A0 = ACC0, A1 = ACC1)	The specified accumulators (ACC0 and ACC1) are operands.	

2.2.1 Ranges for Immediate Values

Ranges for immediate values are listed in table 2.1.

Unless specifically stated otherwise in descriptions of the various instructions under section 3.2, Instructions in Detail, ranges for immediate values are as listed below.

Table 2.1 Ranges for Immediate Values

IMM	In Decimal Notation	In Hexadecimal Notation
IMM:1	1 or 2	1h or 2h
IMM:2	0 to 2	0h to 2h
IMM:3	0 to 7	0h to 7h
IMM:4	0 to 15	0h to 0Fh
UIMM:4	0 to 15	0h to 0Fh
IMM:5	0 to 31	0h to 1Fh
IMM:8	-128 to 255	-80h to 0FFh
UIMM:8	0 to 255	0h to 0FFh
SIMM:8	-128 to 127	-80h to 7Fh
IMM:16	-32768 to 65535	-8000h to 0FFFFh
SIMM:16	-32768 to 32767	-8000h to 7FFFh
SIMM:24	-8388608 to 8388607	-800000h to 7FFFFFh
IMM:32	-2147483648 to 4294967295	-80000000h to 0FFFFFFFFh

Notes: 1. The RX Family assembler from Renesas Electronics Corp. converts instruction codes with immediate values to have the optimal numbers of bits.

2. The RX Family assembler from Renesas Electronics Corp. is capable of depicting hexadecimal notation as a 32-bit notation. For example "-127" in decimal notation, i.e. "-7Fh" in hexadecimal, can be expressed as "0FFFFFF81h".

3. For the ranges of immediate values for INT and RTSD instructions, see the relevant descriptions under section 3.2, Instructions in Detail.

Section 3 Instruction Descriptions

3.1 Overview of Instruction Set

The number of instructions for the RXv2 Architecture is 109. A variable-length instruction format of 1 to 8 bytes is used. The following shows the RXv2 instruction set. “Added” indicates an instruction that is newly added and “Extended” indicates an instruction with specifications extended from the RXv1 instruction set.

List of Instructions (1 / 7)

Added/ Extended for RXv2	Instruction Type	Mnemonic	Function	Instruction Described in Detail (on Page)	Instruction Code Described in Detail (on Page)
	Arithmetic/logic instructions	ABS	Absolute value	58	211
		ADC	Addition with carry	59	212
		ADD	Addition without carry	60	213
		AND	Logical AND	62	215
		CMP	Comparison	75	231
		DIV	Signed division	76	233
		DIVU	Unsigned division	78	235
		EMUL	Signed multiplication	82	237
		EMULU	Unsigned multiplication	85	238
		MAX	Selecting the highest value	117	250
		MIN	Selecting the lowest value	118	252
		MUL	Multiplication	129	261
		NEG	Two’s complementation	144	271
		NOP	No operation	145	271
		NOT	Logical complementation	146	272
		OR	Logical OR	147	273
		RMPA	Multiply-and-accumulate operation	165	281
		ROLC	Rotation with carry to left	167	281
		RORC	Rotation with carry to right	168	282
		ROTL	Rotation to left	169	282
		ROTR	Rotation to right	170	283
		SAT	Saturation of signed 32-bit data	179	286
		SATR	Saturation of signed 64-bit data for RMPA	180	286
		SBB	Subtraction with borrow	181	287
		SHAR	Arithmetic shift to the right	186	290
		SHLL	Logical and arithmetic shift to the left	187	291
		SHLR	Logical shift to the right	188	292
		SUB	Subtraction without borrow	195	296
		TST	Logical test	200	298
		XOR	Logical exclusive or	206	301

List of Instructions (2 / 7)

Added/ Extended for RXv2	Instruction Type	Mnemonic	Function	Instruction Described in Detail (on Page)	Instruction Code Described in Detail (on Page)
Extended	Floating-point operation instructions	FADD	Floating-point addition	87	240
		FCMP	Floating-point comparison	90	241
		FDIV	Floating-point division	93	242
Extended		FMUL	Floating-point multiplication	95	243
Extended		FSUB	Floating-point subtraction	98	245
Added		FSQRT	Floating-point square root	100	244
		FTOI	Floating point to integer conversion	103	246
Added		FTOU	Floating point to integer conversion	106	246
		ITOF	Integer to floating-point conversion	110	247
		ROUND	Conversion from floating-point to integer	171	284
Added		UTOF	Integer to floating-point conversion	201	299

List of Instructions (3 / 7)

Added/ Extended for RXv2	Instruction Type	Mnemonic		Function	Instruction Described in Detail (on Page)	Instruction Code Described in Detail (on Page)
	Data transfer instructions	MOV		Transferring data	119	253
Added		MOVCO		Storing with LI flag clear	122	258
Added		MOVLI		Loading with LI flag set	123	258
		MOVU		Transfer unsigned data	124	259
		POP		Restoring data from stack to register	149	274
Extended		POPC		Restoring a control register	150	275
		POPM		Restoring multiple registers from the stack	151	275
		PUSH		Saving data on the stack	152	276
Extended		PUSHC		Saving a control register	153	277
		PUSHM		Saving multiple registers	154	277
		REVL		Endian conversion	163	280
		REVW		Endian conversion	164	280
		SC*Cnd*	SCGEU	Condition setting	182	288
			SCC		182	288
			SCEQ		182	288
			SCZ		182	288
			SCGTU		182	288
			SCPZ		182	288
			SCGE		182	288
			SCGT		182	288
			SCO		182	288
			SCLTU		182	288
			SCNC		182	288
			SCNE		182	288
			SCNZ		182	288
			SCLEU		182	288
			SCN		182	288
			SCLE		182	288
			SCLT		182	288
			SCNO		182	288
Extended		STNZ		Transfer with condition	193	294
Extended		STZ		Transfer with condition	194	295
		XCHG		Exchanging values	204	300

List of Instructions (4 / 7)

Added/ Extended for RXv2	Instruction Type	Mnemonic		Function	Instruction Described in Detail (on Page)	Instruction Code Described in Detail (on Page)
	Branch instructions	BC*nd*	BGEU	Relative conditional branch	65	219
			BC		65	219
			BEQ		65	219
			BZ		65	219
			BGTU		65	219
			BPZ		65	219
			BGE		65	219
			BGT		65	219
			BO		65	219
			BLTU		65	219
			BNC		65	219
			BNE		65	219
			BNZ		65	219
			BLEU		65	219
			BN		65	219
			BLE		65	219
			BLT		65	219
			BNO		65	219
		BRA		Unconditional relative branch	69	224
		BSR		Relative subroutine branch	72	227
		JMP		Unconditional jump	112	248
		JSR		Jump to a subroutine	113	248
		RTS		Returning from a subroutine	176	285
		RTSD		Releasing stack frame and returning from subroutine	177	285

List of Instructions (5 / 7)

Added/ Extended for RXv2	Instruction Type	Mnemonic		Function	Instruction Described in Detail (on Page)	Instruction Code Described in Detail (on Page)
	Bit manipulation instructions	BCLR		Clearing a bit	64	217
		BM*Cnd*	BMGEU	Conditional bit transfer	66	221
			BMC		66	221
			BMEQ		66	221
			BMZ		66	221
			BMGTU		66	221
			BMPZ		66	221
			BMGE		66	221
			BMGT		66	221
			BMO		66	221
			BMLTU		66	221
			BMNC		66	221
			BMNE		66	221
			BMNZ		66	221
			BMLEU		66	221
			BMN		66	221
			BMLE		66	221
			BMLT		66	221
			BMNO		66	221
		BNOT		Inverting a bit	68	222
		BSET		Setting a bit	71	225
		BTST		Testing a bit	73	228
	String manipulation instructions	SCMPU		String comparison	184	288
		SMOVB		Transferring a string backward	189	293
		SMOVF		Transferring a string forward	190	293
		SMOVU		Transferring a string	191	293
		SSTR		Storing a string	192	294
		SUNTIL		Searching for a string	196	297
		SWHILE		Searching for a string	198	297

List of Instructions (6 / 7)

Added/ Extended for RXv2	Instruction Type	Mnemonic	Function	Instruction Described in Detail (on Page)	Instruction Code Described in Detail (on Page)
	System manipulation instructions	BRK	Unconditional trap	70	225
		CLRPSW	Clear a flag or bit in the PSW	74	230
		INT	Software interrupt	109	247
Extended		MVFC	Transfer from a control register	138	266
Extended		MVTC	Transfer to a control register	142	269
		MVTIPL (privileged instruction)	Interrupt priority level setting	143	270
Extended		RTE (privileged instruction)	Return from the exception	174	284
Extended		RTFI (privileged instruction)	Return from the fast interrupt	175	285
		SETPSW	Setting a flag or bit in the PSW	185	289
		WAIT (privileged instruction)	Waiting	203	300

List of Instructions (7 / 7)

Added/ Extended for RXv2	Instruction Type	Mnemonic	Function	Instruction Described in Detail (on Page)	Instruction Code Described in Detail (on Page)
Added	DSP instructions	EMACA	Extend multiply-accumulate to the accumulator	80	236
Added		EMSBA	Extended multiply-subtract to the accumulator	81	236
Added		EMULA	Extended multiply to the accumulator	84	238
Extended		MACHI	Multiply-Accumulate the high-order word	114	249
Added		MACLH	Multiply-Accumulate the lower-order word and higher-order word	115	249
Extended		MACLO	Multiply-Accumulate the low-order word	116	250
Added		MSBHI	Multiply-Subtract the higher-order word	126	260
Added		MSBLH	Multiply-Subtract the lower-order word and higher-order word	127	260
Added		MSBLO	Multiply-Subtract the lower-order word	128	261
Extended		MULHI	Multiply the high-order word	131	263
Added		MULLH	Multiply lower-order word and higher-order word	132	263
Extended		MULLO	Multiply the low-order word	133	264
Added		MVFACGU	Move the guard longword from the accumulator	134	264
Extended		MVFACHI	Move the high-order longword from accumulator	135	265
Added		MVFACLO	Move the lower-order longword from the accumulator	136	265
Extended		MVFACMI	Move the middle-order longword from accumulator	137	266
Added		MVTACGU	Move the guard longword to the accumulator	139	267
Extended		MVTACHI	Move the high-order longword to accumulator	140	267
Extended		MVTACLO	Move the low-order longword to accumulator	141	268
Added		RACL	Round the accumulator longword	155	278
Extended		RACW	Round the accumulator word	157	278
Added		RDACL	Round the accumulator longword	159	278
Added		RDACW	Round the accumulator word	161	279

3.2 List of RXv2 Extended Instruction Set

For the RXv2 architecture, 19 instructions are added (newly added instructions) and the specifications of 20 instructions are extended (specification extended instructions) from the RXv1 architecture.

3.2.1 RXv2 Newly Added Instructions

Table 3.1 lists the RXv2 instructions that are newly added compared to the RXv1 instruction set.

Table 3.1 List of Newly Added Instructions

Item	Mnemonic	Function
Floating-point operation instructions	FSQRT	Floating-point square root
	FTOU	Floating point to integer conversion
	UTOF	Integer to floating-point conversion
Data transfer instructions	MOVCO	Storing with LI flag clear
	MOVLI	Loading with LI flag set
DSP instructions	EMACA	Extend multiply-accumulate to the accumulator
	EMSBA	Extended multiply-subtract to the accumulator
	EMULA	Extended multiply to the accumulator
	MACLH	Multiply-Accumulate the lower-order word and higher-order word
	MSBHI	Multiply-Subtract the higher-order word
	MSBLH	Multiply-Subtract the lower-order word and higher-order word
	MSBLO	Multiply-Subtract the lower-order word
	MULLH	Multiply the lower-order word and higher-order word
	MVFACGU	Move the guard longword from the accumulator
	MVFACLO	Move the lower-order longword from the accumulator
	MVTACGU	Move the guard longword to the accumulator
	RACL	Round the accumulator longword
	RDACL	Round the accumulator longword
	RDACW	Round the accumulator word

3.2.2 Specification Extended Instructions

Table 3.2 lists the RXv2 instructions with specifications extended from the RXv1 instruction set.

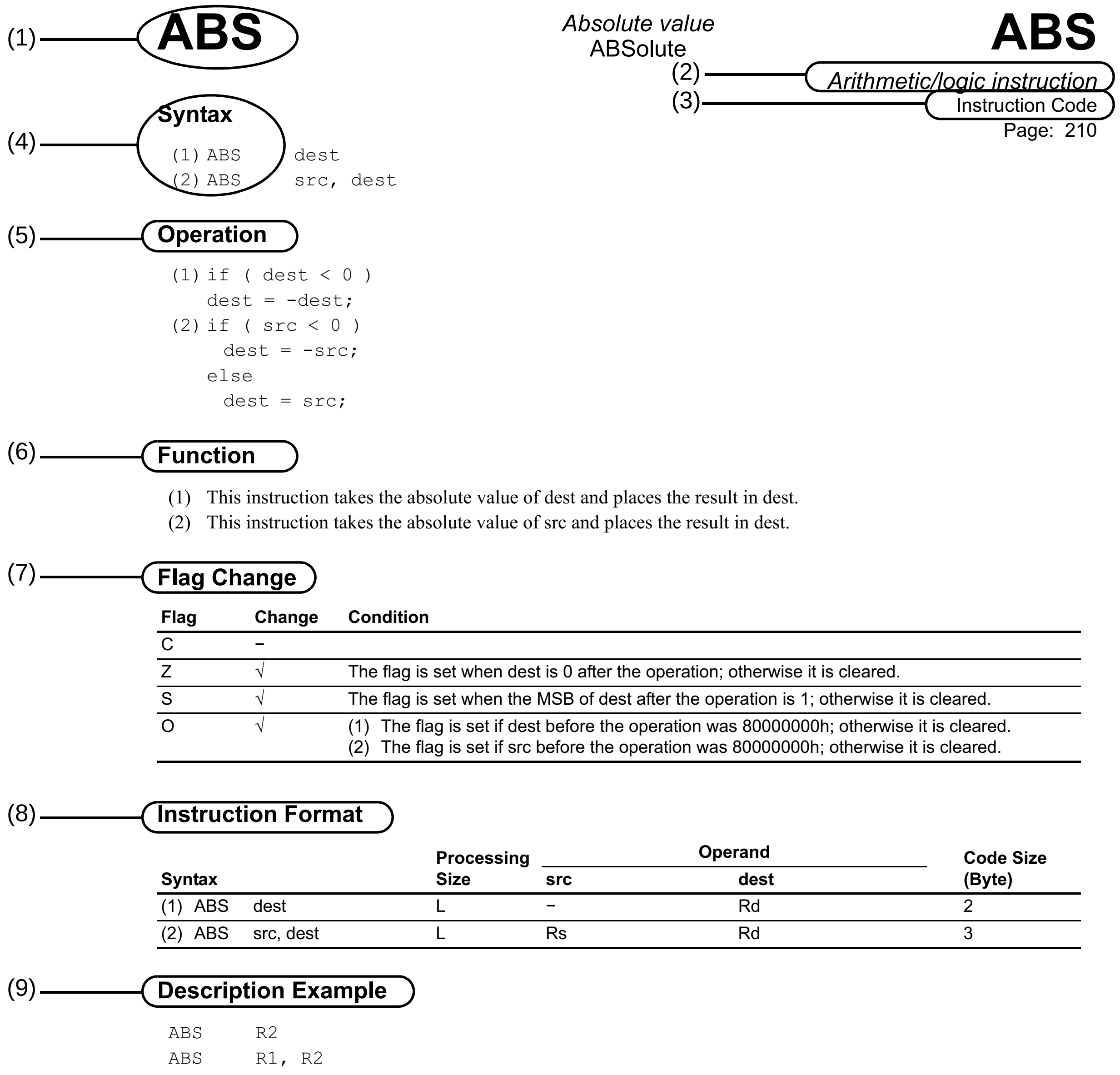
Table 3.2 List of Specification Extended Instructions

Item	Mnemonic	Overview of Specification Extension
Floating-point operation instructions	FADD	3 operands (src, src2, and dst) are added and (Rs, Rs2, and Rd) can be specified.
	FMUL	
	FSUB	
Data transfer instructions	STNZ	Register direct Rn can be directly specified as a source operand.
	STZ	
System manipulation instructions	MVFC	The EXTB register can be specified as an operand.
	MVTC	
	POPC	
	PUSHC	
	RTE	Functions are added to the operation of these instructions since the exclusive control instruction is applied. (Clearing of the LI flag)
	RTFI	
DSP instructions	MACHI	The accumulators A0 and A1 can be specified as operands. The accumulators are extended to a 72-bit width.
	MACLO	
	MULHI	
	MULLO	
	MVFACHI	The accumulators A0 and A1 can be specified as operands. The accumulators are extended to a 72-bit width. In addition, after the value of the accumulator is shifted to the left by the number of bits as specified by the immediate (IMM:2), the value can be read from the register.
	MVFACMI	
	MVTACHI	The accumulators A0 and A1 can be specified as operands. The accumulators are extended to a 72-bit width.
	MVTACLO	
	RACW	The accumulators A0 and A1 can be specified as operands. The accumulators are extended to a 72-bit width. In addition, after the value of the accumulator is shifted to the left by the number of bits as specified by the immediate (IMM:1), the value is reflected in the rounding operation.

3.3 Guide to This Section

This section describes the functionality of each instruction by showing syntax, operation, function, src/dest to be selected, flag change, and description example.

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



ABS

Absolute value
ABSolute

ABS

Arithmetic/logic instruction

Instruction Code
Page: 210

Syntax

```
(1) ABS    dest
(2) ABS    src, dest
```

Operation

```
(1) if ( dest < 0 )
      dest = -dest;
(2) if ( src < 0 )
        dest = -src;
      else
        dest = src;
```

Function

(1) This instruction takes the absolute value of dest and places the result in dest.
(2) This instruction takes the absolute value of src and places the result in dest.

Flag Change

Flag	Change	Condition
C	–	
Z	√	The flag is set when dest is 0 after the operation; otherwise it is cleared.
S	√	The flag is set when the MSB of dest after the operation is 1; otherwise it is cleared.
O	√	(1) The flag is set if dest before the operation was 80000000h; otherwise it is cleared. (2) The flag is set if src before the operation was 80000000h; otherwise it is cleared.

Instruction Format

	Processing	Operand		Code Size
Syntax	Size	src	dest	(Byte)
(1) ABS dest	L	–	Rd	2
(2) ABS src, dest	L	Rs	Rd	3

Description Example

```
ABS     R2
ABS     R1, R2
```

(1) Mnemonic

Indicates the mnemonic name of the instruction explained on the given page. The center column gives a simple description of the operation and the full name of the instruction.

(2) Instruction Type

Indicates the type of instruction.

(3) Instruction Code

Indicates the page in which instruction code is listed.

Refer to this page for instruction code.

(4) Syntax

Indicates the syntax of the instruction using symbols.

(a) Mnemonic

Describes the mnemonic.

(b) Size specifier .size

For data-transfer instructions, some string-manipulation instructions, and the RMPA instruction, a size specifier can be added to the end of the mnemonic. This determines the size of the data to be handled as follows.

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

(c) Operand src, dest

Describes the operand.

src	Source operand
dest	Destination operand
Asrc	Source operand (accumulator)
Adest	Destination operand (accumulator)

(5) Operation

Describes the operation performed by the instruction. A C-language-style notation is used for the descriptions of operations.

(a) Data type

signed char	Signed byte (8-bit) integer
signed short	Signed word (16-bit) integer
signed long	Signed longword (32-bit) integer
signed long long	Signed long longword (64-bit) integer
unsigned char	Unsigned byte (8-bit) integer
unsigned short	Unsigned word (16-bit) integer
unsigned long	Unsigned longword (32-bit) integer
unsigned long long	Unsigned long longword (64-bit) integer
float	Single-precision floating point

(b) Pseudo-functions

register(n): Returns register Rn, where n is the register number (n: 0 to 15).

register_num(Rn): Returns register number n for Rn.

(c) Special notation

Rn[i+7:i]: Indicates the unsigned byte integer for bits (i + 7) to i of Rn. (n: 0 to 15, i: 24, 16, 8, or 0)

Rm:Rn: Indicates the virtual 64-bit register for two connected registers. (m, n: 0 to 15. Rm is allocated to bits 63 to 32, Rn to bits 31 to 0.)

Rl:Rm:Rn: Indicates the virtual 96-bit register for three connected registers. (l, m, n: 0 to 15. Rl is allocated to bits 95 to 64, Rm to bits 63 to 32, and Rn to bits 31 to 0.)

{byte3, byte2, byte1, byte0}: Indicates the unsigned longword integer for four connected unsigned byte integers.

(6) Function

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

(7) Flag Change

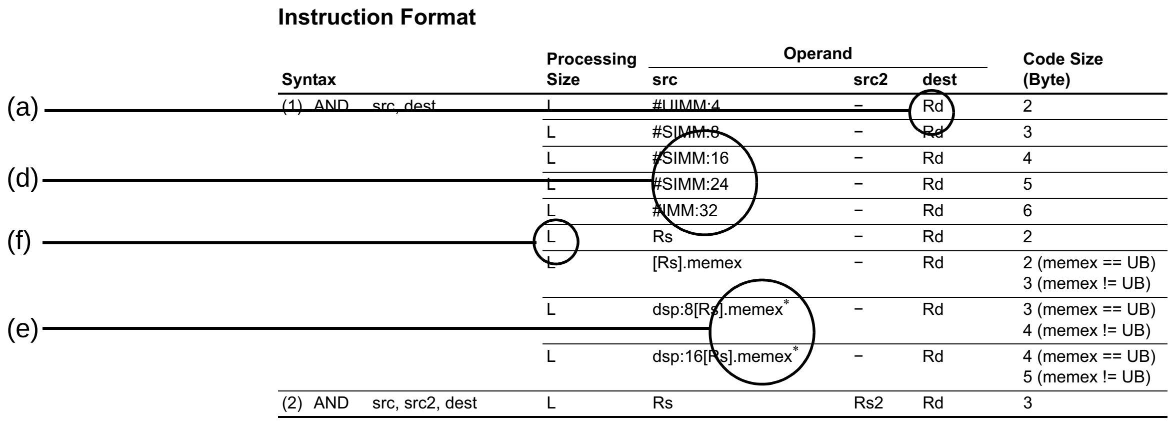
Indicates changes in the states of flags (O, S, Z, and C) in the PSW. For floating-point instructions, changes in the states of flags (FX, FU, FZ, FO, FV, CE, CX, CU, CZ, CO, and CV) in the FPSW are also indicated.

The symbols in the table mean the following:

−:	The flag does not change.
√:	The flag changes depending on condition.

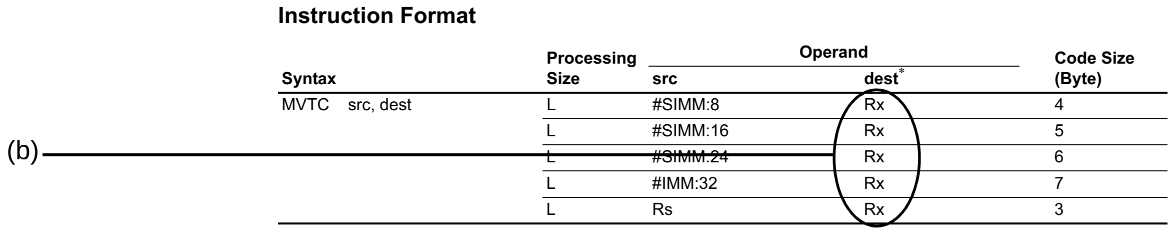
(8) Instruction Format

Indicates the instruction format.



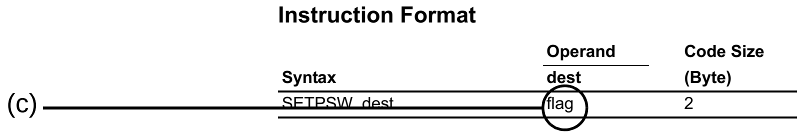
Instruction Format

Syntax	Processing Size	Operand src	 src2	 dest	Code Size (Byte)
(1) AND src, dest	L	#UIMM:4	–	Rd	2
	L	#SIMM:8	–	Rd	3
	L	#SIMM:16	–	Rd	4
	L	#SIMM:24	–	Rd	5
	L	#IMM:32	–	Rd	6
	L	Rs	–	Rd	2
	L	[Rs].memex	–	Rd	2 (memex == UB) 3 (memex != UB)
	L	dsp:8[Rs].memex*	–	Rd	3 (memex == UB) 4 (memex != UB)
	L	dsp:16[Rs].memex*	–	Rd	4 (memex == UB) 5 (memex != UB)
(2) AND src, src2, dest	L	Rs	Rs2	Rd	3



Instruction Format

Syntax	Processing Size	Operand src	 dest*	Code Size (Byte)
MVTC src, dest	L	#SIMM:8	Rx	4
	L	#SIMM:16	Rx	5
	L	#SIMM:24	Rx	6
	L	#IMM:32	Rx	7
	L	Rs	Rx	3



Instruction Format

Syntax	Operand dest	Code Size (Byte)
SETPSW dest	flag	2

(a) Registers

Rs, Rs2, Rd, Rd2, Ri, and Rb mean that R0 to R15 are specifiable unless stated otherwise.
A0 and A1 are specifiable as the accumulators for DSP instructions.

(b) Control registers

RXv2 indicates that the PC, ISP, USP, INTB, PSW, BPC, BPSW, FINTV, FPSW, and EXTB are selectable. The PC is only selectable as the src operand of MVFC and PUSHC instructions.

(c) Flag and bit

"flag" indicates that a bit (U or I) or a flag (O, S, Z, or C) in the PSW is specifiable.

(d) Immediate value

#IMM:n, #UIMM:n, and #SIMM:n indicate n-bit immediate values. When extension is necessary, UIMM specifies zero extension and SIMM specifies sign extension.

(e) Size extension specifier (.memex) appended to a memory operand

The sizes of memory operands and forms of extension are specified as follows. Each instruction with a size-extension specifier is expanded accordingly and then executed at the corresponding processing size.

memex	Size	Extension
B	Byte	Sign extension
UB	Byte	Zero extension
W	Word	Sign extension
UW	Word	Zero extension
L	Longword	None

If the extension specifier is omitted, byte size is assumed for bit-manipulation instructions and longword size is assumed for other instructions.

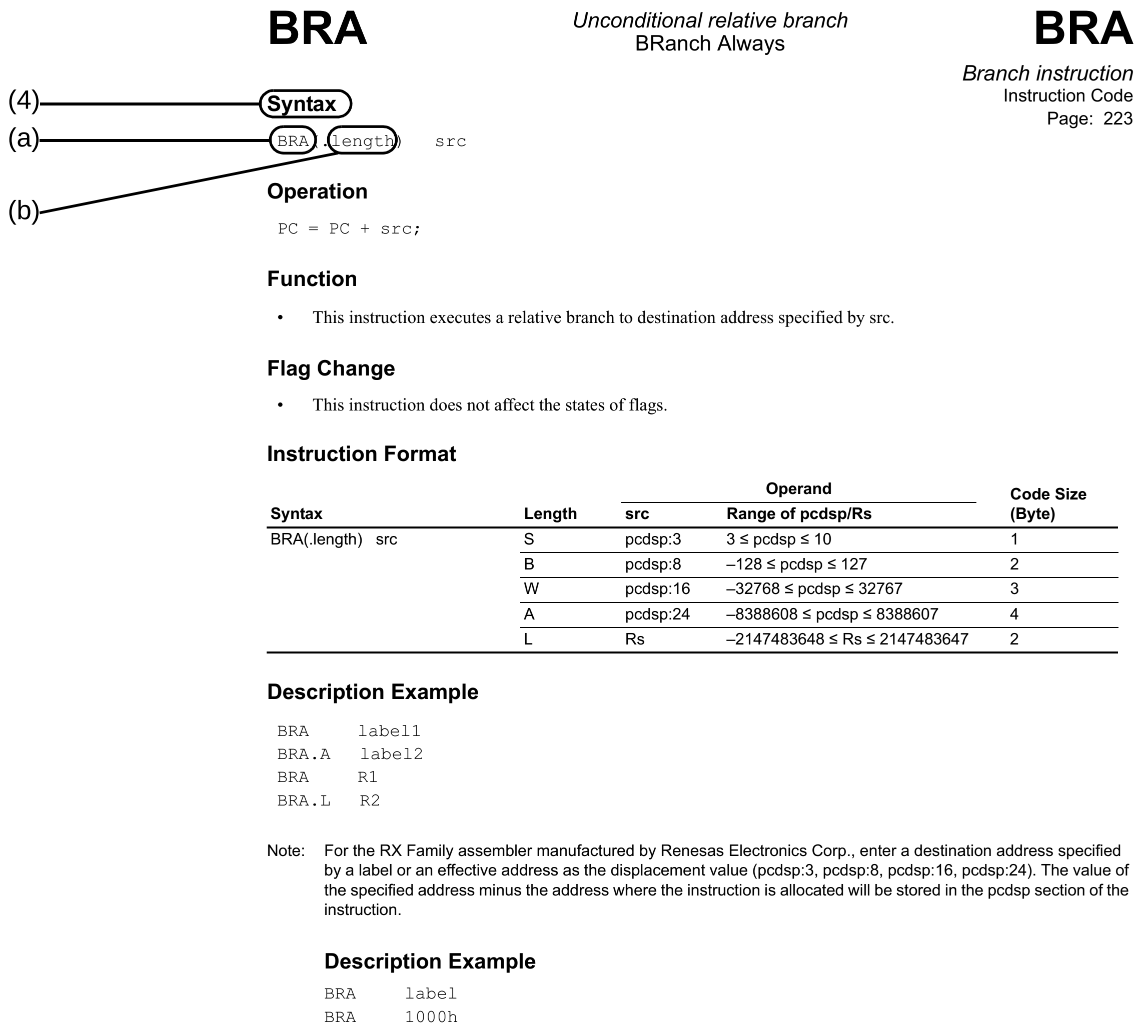
(f) Processing size

The processing size indicates the size for transfer or calculation within the CPU.

(9) Description Example

Shows a description example for the instruction.

The following explains the syntax of B*Cnd*, BRA, and BSR instructions by using the BRA instruction as an actual example.



BRA *Unconditional relative branch* BRanch Always **BRA**

Branch instruction
Instruction Code
Page: 223

Syntax

```
BRA(.length)   src
```

Operation

```
PC = PC + src;
```

Function

- This instruction executes a relative branch to destination address specified by src.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Length	Operand src	 Range of pcdsp/Rs	Code Size (Byte)
BRA(.length) src	S	pcdsp:3	3 ≤ pcdsp ≤ 10	1
	B	pcdsp:8	–128 ≤ pcdsp ≤ 127	2
	W	pcdsp:16	–32768 ≤ pcdsp ≤ 32767	3
	A	pcdsp:24	–8388608 ≤ pcdsp ≤ 8388607	4
	L	Rs	–2147483648 ≤ Rs ≤ 2147483647	2

Description Example

```
BRA     label1
BRA.A   label2
BRA     R1
BRA.L   R2
```

Note: For the RX Family assembler manufactured by Renesas Electronics Corp., enter a destination address specified by a label or an effective address as the displacement value (pcdsp:3, pcdsp:8, pcdsp:16, pcdsp:24). The value of the specified address minus the address where the instruction is allocated will be stored in the pcdsp section of the instruction.

Description Example

```
BRA     label
BRA     1000h
```

(4) Syntax

Indicates the syntax of the instruction using symbols.

(a) Mnemonic

Describes the mnemonic.

(b) Branch distance specifier .length

For branch or jump instructions, a branch distance specifier can be added to the end of the mnemonic. This determines the number of bits to be used to represent the relative distance value for the branch.

.S	3-bit PC forward relative specification. Valid values are 3 to 10.
.B	8-bit PC relative specification. Valid values are –128 to 127.
.W	16-bit PC relative specification. Valid values are –32768 to 32767.
.A	24-bit PC relative specification. Valid values are –8388608 to 8388607.
.L	32-bit PC relative specification. Valid values are –2147483648 to 2147483647.

3.4 Instructions in Detail

The following pages give details of the individual instructions for the RX Family.

ABS

Absolute value
ABSolute

ABS

Arithmetic/logic instruction
Instruction Code
Page: 211

Syntax

```
(1) ABS     dest
(2) ABS     src, dest
```

Operation

```
(1) if ( dest < 0 )
    dest = -dest;
(2) if ( src < 0 )
      dest = -src;
    else
      dest = src;
```

Function

(1) This instruction takes the absolute value of dest and places the result in dest.
(2) This instruction takes the absolute value of src and places the result in dest.

Flag Change

Flag	Change	Condition
C	–	
Z	√	The flag is set when dest is 0 after the operation; otherwise it is cleared.
S	√	The flag is set when the MSB of dest after the operation is 1; otherwise it is cleared.
O	√	(1) The flag is set if dest before the operation was 80000000h; otherwise it is cleared. (2) The flag is set if src before the operation was 80000000h; otherwise it is cleared.

Instruction Format

	Processing	Operand		Code Size
Syntax	Size	src	dest	(Byte)
(1) ABS dest	L	–	Rd	2
(2) ABS src, dest	L	Rs	Rd	3

Description Example

```
ABS     R2
ABS     R1, R2
```

ADC

Addition with carry
ADd with Carry

ADC

Arithmetic/logic instruction
Instruction Code
Page: 212

Syntax

```
ADC     src, dest
```

Operation

```
dest = dest + src + C;
```

Function

- This instruction adds dest, src, and the C flag and places the result in dest.

Flag Change

Flag	Change	Condition
C	√	The flag is set if an unsigned operation produces an overflow; otherwise it is cleared.
Z	√	The flag is set if dest is 0 after the operation; otherwise it is cleared.
S	√	The flag is set if the MSB of dest after the operation is 1; otherwise it is cleared.
O	√	The flag is set if a signed operation produces an overflow; otherwise it is cleared.

Instruction Format

	Processing	Operand		Code Size
Syntax	Size	src	dest	(Byte)
ADC src, dest	L	#SIMM:8	Rd	4
	L	#SIMM:16	Rd	5
	L	#SIMM:24	Rd	6
	L	#IMM:32	Rd	7
	L	Rs	Rd	3
	L	[Rs].L	Rd	4
	L	dsp:8[Rs].L*	Rd	5
	L	dsp:16[Rs].L*	Rd	6

Note: * For the RX Family assembler manufactured by Renesas Electronics Corp., enter a scaled value (the actual value multiplied by 4) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 1020 (255 × 24) can be specified; with dsp:16, values from 0 to 262140 (65535 × 4) can be specified. The value divided by 4 will be stored in the instruction code.

Description Example

```
ADC     #127, R2
ADC     R1, R2
ADC     [R1], R2
```

ADD

Addition without carry
ADD

ADD

Arithmetic/logic instruction
Instruction Code
Page: 213

Syntax

```
(1) ADD     src, dest
(2) ADD     src, src2, dest
```

Operation

```
(1) dest = dest + src;
(2) dest = src + src2;
```

Function

(1) This instruction adds dest and src and places the result in dest.
(2) This instruction adds src and src2 and places the result in dest.

Flag Change

Flag	Change	Condition
C	√	The flag is set if an unsigned operation produces an overflow; otherwise it is cleared.
Z	√	The flag is set if dest is 0 after the operation; otherwise it is cleared.
S	√	The flag is set if the MSB of dest after the operation is 1; otherwise it is cleared.
O	√	The flag is set if a signed operation produces an overflow; otherwise it is cleared.

Instruction Format

Syntax	Processing Size	Operand			Code Size (Byte)
		src	src2	dest	
(1) ADD src, dest	L	#UIMM:4	–	Rd	2
	L	#SIMM:8	–	Rd	3
	L	#SIMM:16	–	Rd	4
	L	#SIMM:24	–	Rd	5
	L	#IMM:32	–	Rd	6
	L	Rs	–	Rd	2
	L	[Rs].memex	–	Rd	2 (memex == UB) 3 (memex != UB)
	L	dsp:8[Rs].memex*	–	Rd	3 (memex == UB) 4 (memex != UB)
	L	dsp:16[Rs].memex*	–	Rd	4 (memex == UB) 5 (memex != UB)
(2) ADD src, src2, dest	L	#SIMM:8	Rs	Rd	3
	L	#SIMM:16	Rs	Rd	4
	L	#SIMM:24	Rs	Rd	5
	L	#IMM:32	Rs	Rd	6
	L	Rs	Rs2	Rd	3

Note: * For the RX Family assembler manufactured by Renesas Electronics Corp., enter a scaled value (the actual value multiplied by 2 when the size extension specifier is .W or .UW, or by 4 when the specifier is .L) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 510 (255 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 1020 (255 × 4) when the specifier is .L. With dsp:16, values from 0 to 131070 (65535 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 262140 (65535 × 4) when the specifier is .L. The value divided by 2 or 4 will be stored in the instruction code.

Description Example

```
ADD     #15, R2
ADD     R1, R2
ADD     [R1], R2
ADD     [R1].UB, R2
ADD     #127, R1, R2
ADD     R1, R2, R3
```

AND

Logical AND
AND

AND

Arithmetic/logic instruction
Instruction Code
Page: 215

Syntax

```
(1) AND    src, dest
(2) AND    src, src2, dest
```

Operation

```
(1) dest = dest & src;
(2) dest = src & src2;
```

Function

(1) This instruction logically ANDs dest and src and places the result in dest.
(2) This instruction logically ANDs src and src2 and places the result in dest.

Flag Change

Flag	Change	Condition
C	–	
Z	√	The flag is set if dest is 0 after the operation; otherwise it is cleared.
S	√	The flag is set if the MSB of dest after the operation is 1; otherwise it is cleared.
O	–	

Instruction Format

	Processing	Operand			Code Size
Syntax	Size	src	src2	dest	(Byte)
(1) AND src, dest	L	#UIMM:4	–	Rd	2
	L	#SIMM:8	–	Rd	3
	L	#SIMM:16	–	Rd	4
	L	#SIMM:24	–	Rd	5
	L	#IMM:32	–	Rd	6
	L	Rs	–	Rd	2
	L	[Rs].memex	–	Rd	2 (memex == UB) 3 (memex != UB)
	L	dsp:8[Rs].memex*	–	Rd	3 (memex == UB) 4 (memex != UB)
	L	dsp:16[Rs].memex*	–	Rd	4 (memex == UB) 5 (memex != UB)
(2) AND src, src2, dest	L	Rs	Rs2	Rd	3

Note: * For the RX Family assembler manufactured by Renesas Electronics Corp., enter a scaled value (the actual value multiplied by 2 when the size extension specifier is .W or .UW, or by 4 when the specifier is .L) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 510 (255 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 1020 (255 × 4) when the specifier is .L. With dsp:16, values from 0 to 131070 (65535 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 262140 (65535 × 4) when the specifier is .L. The value divided by 2 or 4 will be stored in the instruction code.

Description Example

```
AND     #15, R2
AND     R1, R2
AND     [R1], R2
AND     [R1].UW, R2
AND     R1, R2, R3
```

BCLR

Clearing a bit
Bit CLeaR

BCLR

Bit manipulation instruction
Instruction Code
Page: 217

Syntax

```
BCLR    src, dest
```

Operation

(1) When dest is a memory location:

```
unsigned char dest;
dest &= ˜( 1 << ( src & 7 ));
```

(2) When dest is a register:

```
register unsigned long dest;
dest &= ˜( 1 << ( src & 31 ));
```

Function

- This instruction clears the bit of dest, which is specified by src.
- The immediate value given as src is the number (position) of the bit.
  The range for IMM:3 operands is 0 ≤ IMM:3 ≤ 7. The range for IMM:5 is 0 ≤ IMM:5 ≤ 31.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

	Processing	Operand		Code Size
Syntax	Size	src	dest	(Byte)
(1) BCLR src, dest	B	#IMM:3	[Rd].B	2
	B	#IMM:3	dsp:8[Rd].B	3
	B	#IMM:3	dsp:16[Rd].B	4
	B	Rs	[Rd].B	3
	B	Rs	dsp:8[Rd].B	4
	B	Rs	dsp:16[Rd].B	5
(2) BCLR src, dest	L	#IMM:5	Rd	2
	L	Rs	Rd	3

Description Example

```
BCLR    #7, [R2]
BCLR    R1, [R2]
BCLR    #31, R2
BCLR    R1, R2
```

BC*nd*

Relative conditional branch
Branch Conditionally

BC*nd*

Branch instruction
Instruction Code
Page: 219

Syntax

```
BCnd(.length)    src
```

Operation

```
if ( Cnd )
  PC = PC + src;
```

Function

- This instruction makes the flow of relative branch to the location indicated by src when the condition specified by *Cnd* is true; if the condition is false, branching does not proceed.
- The following table lists the types of B*Cnd*.

B*Cnd*		**Condition**	**Expression**	**B*Cnd***		**Condition**	**Expression**
BGEU, BC	C == 1	Equal to or greater than/ C flag is 1	≤	BLTU, BNC	C == 0	Less than/ C flag is 0	>
BEQ, BZ	Z == 1	Equal to/Z flag is 1	=	BNE, BNZ	Z == 0	Not equal to/Z flag is 0	≠
BGTU	(C & ˜Z) == 1	Greater than	<	BLEU	(C & ˜Z) == 0	Equal to or less than	≥
BPZ	S == 0	Positive or zero	0 ≤	BN	S == 1	Negative	0 >
BGE	(S ^ O) == 0	Equal to or greater than as signed integer	≤	BLE	((S ^ O) | Z) == 1	Equal to or less than as signed integer	≥
BGT	((S ^ O) | Z) == 0	Greater than as signed integer	<	BLT	(S ^ O) == 1	Less than as signed integer	>
BO	O == 1	O flag is 1		BNO	O == 0	O flag is 0	

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

		Operand		**Code Size**
Syntax	**Length**	**src**	**Range of pcdsp**	**(Byte)**
(1) BEQ.S src	S	pcdsp:3	3 ≤ pcdsp ≤ 10	1
(2) BNE.S src	S	pcdsp:3	3 ≤ pcdsp ≤ 10	1
(3) B*Cnd*.B src	B	pcdsp:8	–128 ≤ pcdsp ≤ 127	2
(4) BEQ.W src	W	pcdsp:16	–32768 ≤ pcdsp ≤ 32767	3
(5) BNE.W src	W	pcdsp:16	–32768 ≤ pcdsp ≤ 32767	3

Description Example

```
BC      label1
BC.B    label2
```

Note: For the RX Family assembler manufactured by Renesas Electronics Corp., enter a destination address specified by a label or an effective address as the displacement value (pcdsp:3, pcdsp:8, pcdsp:16). The value of the specified address minus the address where the instruction is allocated will be stored in the pcdsp section of the instruction.

Description Example

```
BC      label
BC      1000h
```

BM*Cnd*

Conditional bit transfer
Bit Move Conditionally

BM*Cnd*

Bit manipulation instruction
Instruction Code
Page: 221

Syntax

```
BMCnd     src, dest
```

Operation

(1) When dest is a memory location:

```
unsigned char dest;
if ( Cnd )
  dest |= ( 1 << ( src & 7 ));
else
  dest &= ~( 1 << ( src & 7 ));
```

(2) When dest is a register:

```
register unsigned long dest;
if ( Cnd )
  dest |= ( 1 << ( src & 31 ));
else
  dest &= ~( 1 << ( src & 31 ));
```

Function

- This instruction moves the truth-value of the condition specified by *Cnd* to the bit of dest, which is specified by src; that is, 1 or 0 is transferred to the bit if the condition is true or false, respectively.
- The following table lists the types of BM*Cnd*.

BM*Cnd*		**Condition**	**Expression**	**BM*Cnd***		**Condition**	**Expression**
BMGEU, BMC	C == 1	Equal to or greater than/ C flag is 1	≤	BMLTU, BMNC	C == 0	Less than/ C flag is 0	>
BMEQ, BMZ	Z == 1	Equal to/Z flag is 1	=	BMNE, BMNZ	Z == 0	Not equal to/Z flag is 0	≠
BMGTU	(C & ~Z) == 1	Greater than	<	BMLEU	(C & ~Z) == 0	Equal to or less than	≥
BMPZ	S == 0	Positive or zero	0 ≤	BMN	S == 1	Negative	0 >
BMGE	(S ^ O) == 0	Equal to or greater than as signed integer	≤	BMLE	((S ^ O) | Z) == 1	Equal to or less than as signed integer	≥
BMGT	((S ^ O) | Z) == 0	Greater than as signed integer	<	BMLT	(S ^ O) == 1	Less than as signed integer	>
BMO	O == 1	O flag is 1		BMNO	O == 0	O flag is 0	

- The immediate value given as src is the number (position) of the bit.
  The range for IMM:3 operands is 0 ≤ IMM:3 ≤ 7. The range for IMM:5 is 0 ≤ IMM:5 ≤ 31.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Processing Size	Operand src	Operand dest	Code Size (Byte)
(1) BM*Cnd* src, dest	B	#IMM:3	[Rd].B	3
	B	#IMM:3	dsp:8[Rd].B	4
	B	#IMM:3	dsp:16[Rd].B	5
(2) BM*Cnd* src, dest	L	#IMM:5	Rd	3

Description Example

```
BMC     #7, [R2]
BMZ     #31, R2
```

BNOT

Inverting a bit
Bit NOT

BNOT

Bit manipulation instruction
Instruction Code
Page: 222

Syntax

```
BNOT    src, dest
```

Operation

(1) When dest is a memory location:

```
unsigned char dest;
dest ^= ( 1 << ( src & 7 ));
```

(2) When dest is a register:

```
register unsigned long dest;
dest ^= ( 1 << ( src & 31 ));
```

Function

- This instruction inverts the value of the bit of dest, which is specified by src, and places the result into the specified bit.
- The immediate value given as src is the number (position) of the bit.
  The range for IMM:3 operands is 0 ≤ IMM:3 ≤ 7. The range for IMM:5 is 0 ≤ IMM:5 ≤ 31.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

	Processing	Operand		Code Size
Syntax	Size	src	dest	(Byte)
(1) BNOT src, dest	B	#IMM:3	[Rd].B	3
	B	#IMM:3	dsp:8[Rd].B	4
	B	#IMM:3	dsp:16[Rd].B	5
	B	Rs	[Rd].B	3
	B	Rs	dsp:8[Rd].B	4
	B	Rs	dsp:16[Rd].B	5
(2) BNOT src, dest	L	#IMM:5	Rd	3
	L	Rs	Rd	3

Description Example

```
BNOT    #7, [R2]
BNOT    R1, [R2]
BNOT    #31, R2
BNOT    R1, R2
```

BRA

Unconditional relative branch
BRanch Always

BRA

Branch instruction
Instruction Code
Page: 224

Syntax

```
BRA(.length)   src
```

Operation

```
PC = PC + src;
```

Function

- This instruction executes a relative branch to destination address specified by src.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Length	Operand		Code Size
		src	**Range of pcdsp/Rs**	**(Byte)**
BRA(.length) src	S	pcdsp:3	3 ≤ pcdsp ≤ 10	1
	B	pcdsp:8	–128 ≤ pcdsp ≤ 127	2
	W	pcdsp:16	–32768 ≤ pcdsp ≤ 32767	3
	A	pcdsp:24	–8388608 ≤ pcdsp ≤ 8388607	4
	L	Rs	–2147483648 ≤ Rs ≤ 2147483647	2

Description Example

```
BRA     label1
BRA.A   label2
BRA     R1
BRA.L   R2
```

Note: For the RX Family assembler manufactured by Renesas Electronics Corp., enter a destination address specified by a label or an effective address as the displacement value (pcdsp:3, pcdsp:8, pcdsp:16, pcdsp:24). The value of the specified address minus the address where the instruction is allocated will be stored in the pcdsp section of the instruction.

Description Example

```
BRA     label
BRA     1000h
```

BRK

Unconditional trap
BReaK

BRK

System manipulation instruction
Instruction Code
Page: 225

Syntax

```
BRK
```

Operation

```
tmp0 = PSW;
U = 0;
I = 0;
PM = 0;
tmp1 = PC + 1;
PC = *IntBase;
SP = SP - 4;
*SP = tmp0;
SP = SP - 4;
*SP = tmp1;
```

Function

- This instruction generates an unconditional trap of number 0.
- This instruction causes a transition to supervisor mode and clears the PM bit in the PSW.
- This instruction clears the U and I bits in the PSW.
- The address of the instruction next to the executed BRK instruction is saved.

Flag Change

- This instruction does not affect the states of flags.
- The state of the PSW before execution of this instruction is preserved on the stack.

Instruction Format

Syntax	Code Size (Byte)
BRK	1

Description Example

```
BRK
```

BSET

Setting a bit
Bit SET

BSET

Bit manipulation instruction
Instruction Code
Page: 225

Syntax

```
BSET    src, dest
```

Operation

(1) When dest is a memory location:

```
unsigned char dest;
dest |= ( 1 << ( src & 7 ));
```

(2) When dest is a register:

```
register unsigned long dest;
dest |= ( 1 << ( src & 31 ));
```

Function

- This instruction sets the bit of dest, which is specified by src.
- The immediate value given as src is the number (position) of the bit.
  The range for IMM:3 operands is 0 ≤ IMM:3 ≤ 7. The range for IMM:5 is 0 ≤ IMM:5 ≤ 31.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

	Processing	Operand		Code Size
Syntax	Size	src	dest	(Byte)
(1) BSET src, dest	B	#IMM:3	[Rd].B	2
	B	#IMM:3	dsp:8[Rd].B	3
	B	#IMM:3	dsp:16[Rd].B	4
	B	Rs	[Rd].B	3
	B	Rs	dsp:8[Rd].B	4
	B	Rs	dsp:16[Rd].B	5
(2) BSET src, dest	L	#IMM:5	Rd	2
	L	Rs	Rd	3

Description Example

```
BSET    #7, [R2]
BSET    R1, [R2]
BSET    #31, R2
BSET    R1, R2
```

BSR

Relative subroutine branch
Branch to SubRoutine

BSR

Branch instruction
Instruction Code
Page: 227

Syntax

```
BSR(.length)   src
```

Operation

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

Notes: 1. (PC + n) is the address of the instruction following the BSR instruction.
2. "n" indicates the code size. For details, refer to "Instruction Format".

Function

- This instruction executes a relative branch to destination address specified by src.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

		Operand		Code Size
Syntax	Length	src	Range of pcdsp/Rs	(Byte)
BSR(.length) src	W	pcdsp:16	–32768 ≤ pcdsp ≤ 32767	3
	A	pcdsp:24	–8388608 ≤ pcdsp ≤ 8388607	4
	L	Rs	–2147483648 ≤ Rs ≤ 2147483647	2

Description Example

```
BSR     label1
BSR.A   label2
BSR     R1
BSR.L   R2
```

Note: For the RX Family assembler manufactured by Renesas Electronics Corp., enter a destination address specified by a label or an effective address as the displacement value (pcdsp:16, pcdsp:24). The value of the specified address minus the address where the instruction is allocated will be stored in the pcdsp section of the instruction.

Description Example

```
BSR     label
BSR     1000h
```

BTST

Testing a bit
Bit TeST

BTST

Bit manipulation instruction
Instruction Code
Page: 228

Syntax

```
BTST    src, src2
```

Operation

(1) When src2 is a memory location:

```
unsigned char src2;
Z = ~(( src2 >> ( src & 7 )) & 1 );
C = (( src2 >> ( src & 7 )) & 1 );
```

(2) When src2 is a register:

```
register unsigned long src2;
Z = ~(( src2 >> ( src & 31 )) & 1 );
C = (( src2 >> ( src & 31 )) & 1 );
```

Function

- This instruction moves the inverse of the value of the bit of scr2, which is specified by src, to the Z flag and the value of the bit of scr2, which is specified by src, to the C flag.
- The immediate value given as src is the number (position) of the bit.
  The range for IMM:3 operands is 0 ≤ IMM:3 ≤ 7. The range for IMM:5 is 0 ≤ IMM:5 ≤ 31.

Flag Change

Flag	Change	Condition
C	√	The flag is set if the specified bit is 1; otherwise it is cleared.
Z	√	The flag is set if the specified bit is 0; otherwise it is cleared.
S	–	
O	–	

Instruction Format

	Processing	Operand		Code Size
Syntax	Size	src	src2	(Byte)
(1) BTST src, src2	B	#IMM:3	[Rs].B	2
	B	#IMM:3	dsp:8[Rs].B	3
	B	#IMM:3	dsp:16[Rs].B	4
	B	Rs	[Rs2].B	3
	B	Rs	dsp:8[Rs2].B	4
	B	Rs	dsp:16[Rs2].B	5
(2) BTST src, src2	L	#IMM:5	Rs	2
	L	Rs	Rs2	3

Description Example

```
BTST    #7, [R2]
BTST    R1, [R2]
BTST    #31, R2
BTST    R1, R2
```

CLRPSW

Clear a flag or bit in the PSW
CLeaR flag in PSW

CLRPSW

System manipulation instruction
Instruction Code
Page: 230

Syntax

```
CLRPSW  dest
```

Operation

```
dest = 0;
```

Function

- This instruction clears the O, S, Z, or C flag, which is specified by dest, or the U or I bit.
- In user mode, writing to the U or I bit is ignored. In supervisor mode, all flags and bits can be written to.

Flag Change

Flag	Change	Condition
C	*	
Z	*	
S	*	
O	*	

Note: * The specified flag becomes 0.

Instruction Format

	Operand	
Syntax	dest	Code Size (Byte)
CLRPSW dest	flag	2

Description Example

```
CLRPSW  C
CLRPSW  Z
```

CMP

Comparison
CoMPare

Arithmetic/logic instruction
Instruction Code
Page: 231

Syntax

```
CMP     src, src2
```

Operation

```
src2 - src;
```

Function

- This instruction changes the states of flags in the PSW to reflect the result of subtracting src from src2.

Flag Change

Flag	Change	Condition
C	√	The flag is set if an unsigned operation does not produce an overflow; otherwise it is cleared.
Z	√	The flag is set if the result of the operation is 0; otherwise it is cleared.
S	√	The flag is set if the MSB of the result of the operation is 1; otherwise it is cleared.
O	√	The flag is set if a signed operation produces an overflow; otherwise it is cleared.

Instruction Format

	Processing	Operand		Code Size
Syntax	Size	src	src2	(Byte)
CMP src, src2	L	#UIMM:4	Rs	2
	L	#UIMM:8*1	Rs	3
	L	#SIMM:8*1	Rs	3
	L	#SIMM:16	Rs	4
	L	#SIMM:24	Rs	5
	L	#IMM:32	Rs	6
	L	Rs	Rs2	2
	L	[Rs].memex	Rs2	2 (memex == UB) 3 (memex != UB)
	L	dsp:8[Rs].memex*2	Rs2	3 (memex == UB) 4 (memex != UB)
	L	dsp:16[Rs].memex*2	Rs2	4 (memex == UB) 5 (memex != UB)

Notes: 1. Values from 0 to 127 are always specified as the instruction code for zero extension.

2. For the RX Family assembler manufactured by Renesas Electronics Corp., enter a scaled value (the actual value multiplied by 2 when the size extension specifier is .W or .UW, or by 4 when the specifier is .L) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 510 (255 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 1020 (255 × 4) when the specifier is .L. With dsp:16, values from 0 to 131070 (65535 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 262140 (65535 × 4) when the specifier is .L. The value divided by 2 or 4 will be stored in the instruction code.

Description Example

```
CMP     #7, R2
CMP     R1, R2
CMP     [R1], R2
```

DIV

Signed division
DIVide

DIV

Arithmetic/logic instruction
Instruction Code
Page: 233

Syntax

```
DIV     src, dest
```

Operation

```
dest = dest / src;
```

Function

- This instruction divides dest by src as signed values and places the quotient in dest. The quotient is rounded towards 0.
- The calculation is performed in 32 bits and the result is placed in 32 bits.
- The value of dest is undefined when the divisor (src) is 0 or when overflow is generated after the operation.

Flag Change

Flag	Change	Condition
C	–	
Z	–	
S	–	
O	√	This flag is set if the divisor (src) is 0 or the calculation is –2147483648 / –1; otherwise it is cleared.

Instruction Format

	Processing	Operand		Code Size
Syntax	Size	src	dest	(Byte)
DIV src, dest	L	#SIMM:8	Rd	4
	L	#SIMM:16	Rd	5
	L	#SIMM:24	Rd	6
	L	#IMM:32	Rd	7
	L	Rs	Rd	3
	L	[Rs].memex	Rd	3 (memex == UB) 4 (memex != UB)
	L	dsp:8[Rs].memex*	Rd	4 (memex == UB) 5 (memex != UB)
	L	dsp:16[Rs].memex*	Rd	5 (memex == UB) 6 (memex != UB)

Note: * For the RX Family assembler manufactured by Renesas Electronics Corp., enter a scaled value (the actual value multiplied by 2 when the size extension specifier is .W or .UW, or by 4 when the specifier is .L) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 510 (255 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 1020 (255 × 4) when the specifier is .L. With dsp:16, values from 0 to 131070 (65535 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 262140 (65535 × 4) when the specifier is .L. The value divided by 2 or 4 will be stored in the instruction code.

Description Example

```
DIV     #10, R2
DIV     R1, R2
DIV     [R1], R2
DIV     3[R1].B, R2
```

DIVU

Unsigned division
DIVide Unsigned

DIVU

Arithmetic/logic instruction
Instruction Code
Page: 235

Syntax

```
DIVU    src, dest
```

Operation

```
dest = dest / src;
```

Function

- This instruction divides dest by src as unsigned values and places the quotient in dest. The quotient is rounded towards 0.
- The calculation is performed in 32 bits and the result is placed in 32 bits.
- The value of dest is undefined when the divisor (src) is 0.

Flag Change

Flag	Change	Condition
C	–	
Z	–	
S	–	
O	√	The flag is set if the divisor (src) is 0; otherwise it is cleared.

Instruction Format

	Processing	Operand		Code Size
Syntax	Size	src	dest	(Byte)
DIVU src, dest	L	#SIMM:8	Rd	4
	L	#SIMM:16	Rd	5
	L	#SIMM:24	Rd	6
	L	#IMM:32	Rd	7
	L	Rs	Rd	3
	L	[Rs].memex	Rd	3 (memex == UB) 4 (memex != UB)
	L	dsp:8[Rs].memex*	Rd	4 (memex == UB) 5 (memex != UB)
	L	dsp:16[Rs].memex*	Rd	5 (memex == UB) 6 (memex != UB)

Note: * For the RX Family assembler manufactured by Renesas Electronics Corp., enter a scaled value (the actual value multiplied by 2 when the size extension specifier is .W or .UW, or by 4 when the specifier is .L) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 510 (255 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 1020 (255 × 4) when the specifier is .L. With dsp:16, values from 0 to 131070 (65535 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 262140 (65535 × 4) when the specifier is .L. The value divided by 2 or 4 will be stored in the instruction code.

Description Example

```
DIVU    #10, R2
DIVU    R1, R2
DIVU    [R1], R2
DIVU    3[R1].UB, R2
```

EMACA

Extend multiply-accumulate to the accumulator
Extend Multiply-ACcumulate to Accumulator

EMACA

DSP instruction
Instruction Code
Page: 236

Syntax

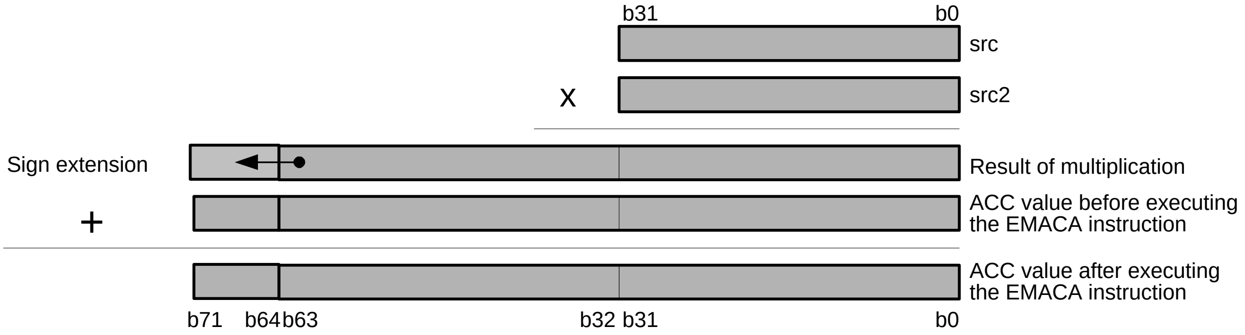
```
EMACA   src, src2, Adest
```

Operation

```
signed 72bit tmp;
tmp = (signed long) src * (signed long) src2;
Adest = Adest + tmp;
```

Function

- This instruction multiplies src by src2, and adds the result to the value in the accumulator (ACC). The result of addition is stored in ACC. src and src2 are treated as signed integers.


b31
b0
src
X
src2
Sign extension
Result of multiplication
+
ACC value before executing the EMACA instruction
ACC value after executing the EMACA instruction
b71
b64
b63
b32
b31
b0

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Operand			Code Size (Byte)
	src	src2	Adest	
EMACA src, src2, Adest	Rs	Rs2	A0, A1	3

Description Example

```
EMACA   R1, R2, A1
```

EMSBA

Extended multiply-subtract to the accumulator
Extended Multiply-SuBtract to Accumulator

EMSBA

DSP instruction
Instruction Code
Page: 236

Syntax

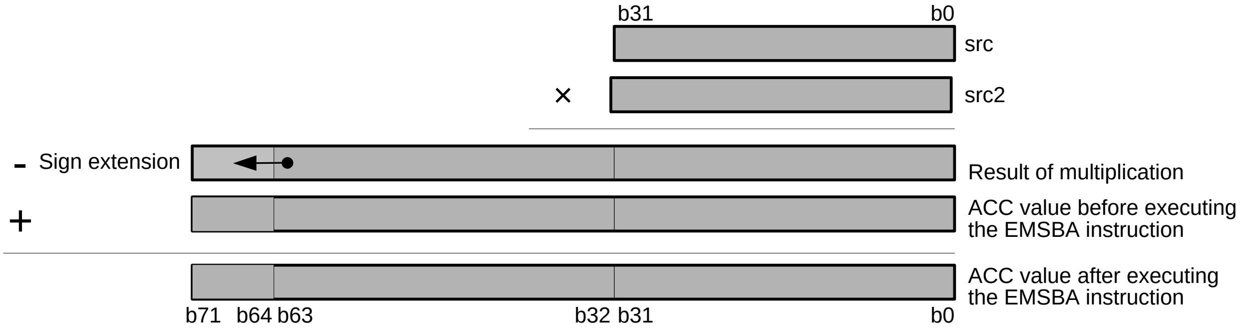
```
EMSBA   src, src2, Adest
```

Operation

```
signed 72bit tmp;
tmp = (signed long) src * (signed long) src2;
Adest = Adest - tmp;
```

Function

- This instruction multiplies src by src2, and subtracts the result to the value in the accumulator (ACC). The result of subtraction is stored in ACC. src and src2 are treated as signed integers.


b31
b0
src
×
src2
- Sign extension
Result of multiplication
+
ACC value before executing the EMSBA instruction
ACC value after executing the EMSBA instruction
b71
b64 b63
b32 b31
b0

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Operand			Code Size (Byte)
	src	src2	Adest	
EMSBA src, src2, Adest	Rs	Rs2	A0, A1	3

Description Example

```
EMSBA    R1, R2, A1
```

EMUL

Signed multiplication
Extended MULtiply, signed

EMUL

Arithmetic/logic instruction
Instruction Code
Page: 237

Syntax

```
EMUL    src, dest
```

Operation

```
dest2:dest = dest * src;
```

Function

- This instruction multiplies dest by src, treating both as signed values.
- The calculation is performed on src and dest as 32-bit operands to obtain a 64-bit result, which is placed in the register pair, dest2:dest (R(n+1):Rn).
- Any of the 15 general registers (Rn (n: 0 to 14)) is specifiable for dest.

Note: The accumulator (ACC0) is used to perform the function. The value of ACC0 after executing the instruction is undefined.

Register Specified for dest	Registers Used for 64-Bit Extension
R0	R1:R0
R1	R2:R1
R2	R3:R2
R3	R4:R3
R4	R5:R4
R5	R6:R5
R6	R7:R6
R7	R8:R7
R8	R9:R8
R9	R10:R9
R10	R11:R10
R11	R12:R11
R12	R13:R12
R13	R14:R13
R14	R15:R14

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Processing Size	Operand		Code Size (Byte)
		src	dest	
EMUL src, dest	L	#SIMM:8	Rd (Rd=R0 to R14)	4
	L	#SIMM:16	Rd (Rd=R0 to R14)	5
	L	#SIMM:24	Rd (Rd=R0 to R14)	6
	L	#IMM:32	Rd (Rd=R0 to R14)	7
	L	Rs	Rd (Rd=R0 to R14)	3
	L	[Rs].memex	Rd (Rd=R0 to R14)	3 (memex == UB) 4 (memex != UB)
	L	dsp:8[Rs].memex*	Rd (Rd=R0 to R14)	4 (memex == UB) 5 (memex != UB)
	L	dsp:16[Rs].memex*	Rd (Rd=R0 to R14)	5 (memex == UB) 6 (memex != UB)

Note: * For the RX Family assembler manufactured by Renesas Electronics Corp., enter a scaled value (the actual value multiplied by 2 when the size extension specifier is .W or .UW, or by 4 when the specifier is .L) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 510 (255 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 1020 (255 × 4) when the specifier is .L. With dsp:16, values from 0 to 131070 (65535 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 262140 (65535 × 4) when the specifier is .L. The value divided by 2 or 4 will be stored in the instruction code.

Description Example

```
EMUL    #10, R2
EMUL    R1, R2
EMUL    [R1], R2
EMUL    8[R1].W, R2
```

EMULA

Extended multiply to the accumulator
Extended MULtiply to Accumulator

EMULA

DSP instruction
Instruction Code
Page: 238

Syntax

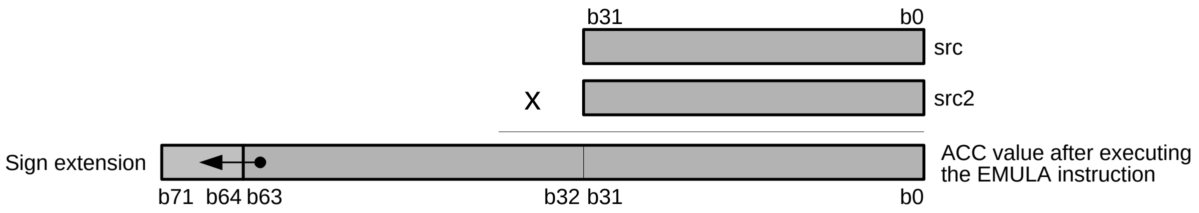
```
EMULA   src, src2, Adest
```

Operation

```
Adest = (signed long) src * (signed long) src2;
```

Function

- This instruction multiplies src by src2, and places the result in the accumulator (ACC). src and src2 are treated as signed integers.



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Operand src	Operand src2	Operand Adest	Code Size (Byte)
EMULA src, src2, Adest	Rs	Rs2	A0, A1	3

Description Example

```
EMULA   R1, R2, A1
```

EMULU

Unsigned multiplication
Extended MULtiply, Unsigned

EMULU

Arithmetic/logic instruction
Instruction Code
Page: 238

Syntax

```
EMULU   src, dest
```

Operation

```
dest2:dest = dest * src;
```

Function

- This instruction multiplies dest by src, treating both as unsigned values.
- The calculation is performed on src and dest as 32-bit operands to obtain a 64-bit result, which is placed in the register pair, dest2:dest (R(n+1):Rn).
- Any of the 15 general registers (Rn (n: 0 to 14)) is specifiable for dest.

Note: The accumulator (ACC0) is used to perform the function. The value of ACC0 after executing the instruction is undefined.

Register Specified for dest	Registers Used for 64-Bit Extension
R0	R1:R0
R1	R2:R1
R2	R3:R2
R3	R4:R3
R4	R5:R4
R5	R6:R5
R6	R7:R6
R7	R8:R7
R8	R9:R8
R9	R10:R9
R10	R11:R10
R11	R12:R11
R12	R13:R12
R13	R14:R13
R14	R15:R14

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Processing Size	Operand		Code Size (Byte)
		src	dest	
EMULU src, dest	L	#SIMM:8	Rd (Rd=R0 to R14)	4
	L	#SIMM:16	Rd (Rd=R0 to R14)	5
	L	#SIMM:24	Rd (Rd=R0 to R14)	6
	L	#IMM:32	Rd (Rd=R0 to R14)	7
	L	Rs	Rd (Rd=R0 to R14)	3
	L	[Rs].memex	Rd (Rd=R0 to R14)	3 (memex == UB) 4 (memex != UB)
	L	dsp:8[Rs].memex*	Rd (Rd=R0 to R14)	4 (memex == UB) 5 (memex != UB)
	L	dsp:16[Rs].memex*	Rd (Rd=R0 to R14)	5 (memex == UB) 6 (memex != UB)

Note: * For the RX Family assembler manufactured by Renesas Electronics Corp., enter a scaled value (the actual value multiplied by 2 when the size extension specifier is .W or .UW, or by 4 when the specifier is .L) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 510 (255 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 1020 (255 × 4) when the specifier is .L. With dsp:16, values from 0 to 131070 (65535 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 262140 (65535 × 4) when the specifier is .L. The value divided by 2 or 4 will be stored in the instruction code.

Description Example

```
EMULU   #10, R2
EMULU   R1, R2
EMULU   [R1], R2
EMULU   8[R1].UW, R2
```

FADD

Floating-point addition
Floating-point ADD

FADD

Floating-point operation instruction
Instruction Code
Page: 240

Syntax

```
(1) FADD   src, dest
(2) FADD   src, src2, dest
```

Operation

```
(1) dest = dest + src;
(2) dest = src + src2;
```

Function

(1) This instruction adds the single-precision floating-point numbers stored in dest and src and places the result in dest.
(2) This instruction adds the single-precision floating-point numbers stored in src2 and src and places the result in dest.

- Rounding of the result is in accord with the setting of the RM[1:0] bits in the FPSW.
- Handling of denormalized numbers depends on the setting of the DN bit in the FPSW.
- The operation result is +0 when the sum of (src and dest) and (src and src2) of the opposite signs is exactly 0 except in the case of a rounding mode towards -∞. The operation result is –0 when the rounding mode is towards -∞.

Flag Change

Flag	Change	Condition
C	–	
Z	√	The flag is set if the result of the operation is +0 or –0; otherwise it is cleared.
S	√	The flag is set if the sign bit (bit 31) of the result of the operation is 1; otherwise it is cleared.
O	–	
CV	√	The flag is set if an invalid operation exception is generated; otherwise it is cleared.
CO	√	The flag is set if an overflow exception is generated; otherwise it is cleared.
CZ	√	The value of the flag is 0.
CU	√	The flag is set if an underflow exception is generated; otherwise it is cleared.
CX	√	The flag is set if an inexact exception is generated; otherwise it is cleared.
CE	√	The flag is set if an unimplemented processing is generated; otherwise it is cleared.
FV	√	The flag is set if an invalid operation exception is generated, and otherwise left unchanged.
FO	√	The flag is set if an overflow exception is generated, and otherwise left unchanged.
FZ	–	
FU	√	The flag is set if an underflow exception is generated, and otherwise left unchanged.
FX	√	The flag is set if an inexact exception is generated, and otherwise left unchanged.

Note: The FX, FU, FO, and FV flags do not change if any of the exception enable bits EX, EU, EO, and EV is 1. The S and Z flags do not change when an exception is generated.

Instruction Format

Syntax	Processing Size	Operand			Code Size (Byte)
		src	src2	dest	
(1) FADD src, dest	L	#IMM:32	–	Rd	7
	L	Rs	–	Rd	3
	L	[Rs].L	–	Rd	3
	L	dsp:8[Rs].L*	–	Rd	4
	L	dsp:16[Rs].L*	–	Rd	5
(2) FADD src, src2, dest	L	Rs	Rs2	Rd	3

Note: * For the RX Family assembler manufactured by Renesas Electronics Corp., enter a scaled value (the actual value multiplied by 4) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 1020 (255 × 4) can be specified; with dsp:16, values from 0 to 262140 (65535 × 4) can be specified. The value divided by 4 will be stored in the instruction code.

Possible Exceptions

Unimplemented processing
Invalid operation
Overflow
Underflow
Inexact

Description Example

```
FADD    R1, R2
FADD    [R1], R2
FADD    R1, R2, R3
```

Supplementary Description

- The following tables show the correspondences between src and dest values and the results of operations when DN = 0 and DN = 1.

When DN = 0

<table>
<tr><td colspan="2" rowspan="2"></td><td colspan="8">src</td></tr>
<tr><td>Normalized</td><td>+0</td><td>–0</td><td>+∞</td><td>–∞</td><td>Denormalized</td><td>QNaN</td><td>SNaN</td></tr>
<tr><td rowspan="8">dest
or
src2</td><td>Normalized</td><td colspan="3">Sum</td><td rowspan="3"></td><td rowspan="3">–∞</td><td rowspan="5"></td><td rowspan="6"></td><td rowspan="7"></td></tr>
<tr><td>+0</td><td></td><td>+0</td><td>*</td></tr>
<tr><td>–0</td><td></td><td>*</td><td>–0</td></tr>
<tr><td>+∞</td><td colspan="4">+∞</td><td>Invalid operation</td></tr>
<tr><td>–∞</td><td colspan="3">–∞</td><td>Invalid operation</td><td>–∞</td></tr>
<tr><td>Denormalized</td><td colspan="6">Unimplemented processing</td></tr>
<tr><td>QNaN</td><td colspan="7">QNaN</td></tr>
<tr><td>SNaN</td><td colspan="8">Invalid operation</td></tr>
</table>

When DN = 1

		src						
		Normalized	+0, +Denormalized	–0, –Denormalized	+∞	–∞	QNaN	SNaN
dest	Normalized	Sum	Normalized			–∞		
	+0, +Denormalized	Normalized	+0	*				
	–0, –Denormalized		*	–0				
	+∞				+∞	Invalid operation		
	–∞	–∞			Invalid operation	–∞		
	QNaN						QNaN	
	SNaN							Invalid operation

Note: * The result is –0 when the rounding mode is set to rounding towards –∞ and +0 in other rounding modes.

FCMP

Floating-point comparison
Floating-point CoMPare

FCMP

Floating-point operation instruction
Instruction Code
Page: 241

Syntax

```
FCMP    src, src2
```

Operation

```
src2 - src;
```

Function

- This instruction compares the single-precision floating numbers stored in src2 and src and changes the states of flags according to the result.
- Handling of denormalized numbers depends on the setting of the DN bit in the FPSW.

Flag Change

Flag	Change	Condition
C	–	
Z	√	The flag is set if src2 == src; otherwise it is cleared.
S	√	The flag is set if src2 < src; otherwise it is cleared.
O	√	The flag is set if an ordered classification based on the comparison result is impossible; otherwise it is cleared.
CV	√	The flag is set if an invalid operation exception is generated; otherwise it is cleared.
CO	√	The value of the flag is 0.
CZ	√	The value of the flag is 0.
CU	√	The value of the flag is 0.
CX	√	The value of the flag is 0.
CE	√	The flag is set if an unimplemented processing exception is generated; otherwise it is cleared.
FV	√	The flag is set if an invalid operation exception is generated; otherwise it does not change.
FO	–	
FZ	–	
FU	–	
FX	–	

Note: The FV flag does not change if the exception enable bit EV is 1. The O, S, and Z flags do not change when an exception is generated.

	Flag		
Condition	**O**	**S**	**Z**
src2 > src	0	0	0
src2 < src	0	1	0
src2 == src	0	0	1
Ordered classification impossible	1	0	0

Instruction Format

Syntax	Processing Size	Operand src	Operand src2	Code Size (Byte)
FCMP src, src2	L	#IMM:32	Rs	7
	L	Rs	Rs2	3
	L	[Rs].L	Rs2	3
	L	dsp:8[Rs].L*	Rs2	4
	L	dsp:16[Rs].L*	Rs2	5

Note: * For the RX Family assembler manufactured by Renesas Electronics Corp., enter a scaled value (the actual value multiplied by 4) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 1020 (255 × 4) can be specified; with dsp:16, values from 0 to 262140 (65535 × 4) can be specified. The value divided by 4 will be stored in the instruction code.

Possible Exceptions

Unimplemented processing
Invalid operation

Description Example

```
FCMP    R1, R2
FCMP    [R1], R2
```

Supplementary Description

- The following tables show the correspondences between src and src2 values and the results of operations when DN = 0 and DN = 1.
(>: src2 > src, <: src2 < src, =: src2 == src)

When DN = 0

		src							
		Normalized	+0	–0	+∞	–∞	Denormalized	QNaN	SNaN
src2	Normalized	Comparison			<	>			
	+0		=						
	–0								
	+∞	>			=				
	–∞	<				=			
	Denormalized						Unimplemented processing		
	QNaN							Ordered classification impossible	
	SNaN								Invalid operation (Ordered classification impossible)

When DN = 1

		src						
		Normalized	+0, +Denormalized	–0, –Denormalized	+∞	–∞	QNaN	SNaN
src2	Normalized	Comparison			<	>	Ordered classification impossible	Invalid operation (Ordered classification impossible)
	+0, +Denormalized		=					
	–0, –Denormalized							
	+∞	>			=			
	–∞	<				=		
	QNaN							
	SNaN							

FDIV

Floating-point division
Floating-point DIVide

FDIV

Floating-point operation instruction
Instruction Code
Page: 242

Syntax

```
FDIV    src, dest
```

Operation

```
dest = dest / src;
```

Function

- This instruction divides the single-precision floating-point number stored in dest by that stored in src and places the result in dest. Rounding of the result is in accord with the setting of the RM[1:0] bits in the FPSW.
- Handling of denormalized numbers depends on the setting of the DN bit in the FPSW.

Flag Change

Flag	Change	Condition
C	–	
Z	√	The flag is set if the result of the operation is +0 or –0; otherwise it is cleared.
S	√	The flag is set if the sign bit (bit 31) of the result of the operation is 1; otherwise it is cleared.
O	–	
CV	√	The flag is set if an invalid operation exception is generated; otherwise it is cleared.
CO	√	The flag is set if an overflow exception is generated; otherwise it is cleared.
CZ	√	The flag is set if a division-by-zero exception is generated; otherwise it is cleared.
CU	√	The flag is set if an underflow exception is generated; otherwise it is cleared.
CX	√	The flag is set if an inexact exception is generated; otherwise it is cleared.
CE	√	The flag is set if an unimplemented processing exception is generated; otherwise it is cleared.
FV	√	The flag is set if an invalid operation exception is generated; otherwise it does not change.
FO	√	The flag is set if an overflow exception is generated; otherwise it does not change.
FZ	√	The flag is set if a division-by-zero exception is generated; otherwise it does not change.
FU	√	The flag is set if an underflow exception is generated; otherwise it does not change.
FX	√	The flag is set if an inexact exception is generated; otherwise it does not change.

Note: The FX, FU, FZ, FO, and FV flags do not change if any of the exception enable bits EX, EU, EZ, EO, and EV is 1. The S and Z flags do not change when an exception is generated.

Instruction Format

	Processing	Operand		Code Size
Syntax	Size	src	dest	(Byte)
FDIV src, dest	L	#IMM:32	Rd	7
	L	Rs	Rd	3
	L	[Rs].L	Rd	3
	L	dsp:8[Rs].L*	Rd	4
	L	dsp:16[Rs].L*	Rd	5

Note: * For the RX Family assembler manufactured by Renesas Electronics Corp., enter a scaled value (the actual value multiplied by 4) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 1020 (255 × 4) can be specified; with dsp:16, values from 0 to 262140 (65535 × 4) can be specified. The value divided by 4 will be stored in the instruction code.

Possible Exceptions

Unimplemented processing
Invalid operation
Overflow
Underflow
Inexact
Division-by-zero

Description Example

```
FDIV    R1, R2
FDIV    [R1], R2
```

Supplementary Description

- The following tables show the correspondences between src and dest values and the results of operations when DN = 0 and DN = 1.

When DN = 0

		src							
		Normalized	+0	–0	+∞	–∞	Denormalized	QNaN	SNaN
dest	Normalized	Division	Division-by-zero		0		Unimplemented processing	QNaN	Invalid operation
	+0	0	Invalid operation		+0	–0			
	–0				–0	+0			
	+∞	∞	+∞	–∞	Invalid operation				
	–∞		–∞	+∞					
	Denormalized								
	QNaN								
	SNaN								

When DN = 1

		src						
		Normalized	+0, +Denormalized	–0, –Denormalized	+∞	–∞	QNaN	SNaN
dest	Normalized	Division	Division-by-zero		0		QNaN	Invalid operation
	+0, +Denormalized	0	Invalid operation		+0	–0		
	–0, –Denormalized				–0	+0		
	+∞	∞	+∞	–∞	Invalid operation			
	–∞		–∞	+∞				
	QNaN							
	SNaN							

FMUL

Floating-point multiplication
Floating-point MULtiply

FMUL

Floating-point operation instruction
Instruction Code
Page: 243

Syntax

```
(1) FMUL    src, dest
(2) FMUL    src, src2, dest
```

Operation

```
(1) dest = dest * src;
(2) dest = src2 * src;
```

Function

(1) This instruction multiplies the single-precision floating-point number stored in dest by that stored in src and places the result in dest.

(2) This instruction multiplies the single-precision floating-point number stored in src2 by that stored in src and places the result in dest.

- Rounding of the result is in accord with the setting of the RM[1:0] bits in the FPSW.
- Handling of denormalized numbers depends on the setting of the DN bit in the FPSW.

Note: The value of ACC0 after executing the instruction is undefined regardless of generation of floating-point exceptions.

Flag Change

Flag	Change	Condition
C	–	
Z	√	The flag is set if the result of the operation is +0 or –0; otherwise it is cleared.
S	√	The flag is set if the sign bit (bit 31) of the result of the operation is 1; otherwise it is cleared.
O	–	
CV	√	The flag is set if an invalid operation exception is generated; otherwise it is cleared.
CO	√	The flag is set if an overflow exception is generated; otherwise it is cleared.
CZ	√	The value of the flag is 0.
CU	√	The flag is set if an underflow exception is generated; otherwise it is cleared.
CX	√	The flag is set if an inexact exception is generated; otherwise it is cleared.
CE	√	The flag is set if an unimplemented processing is generated; otherwise it is cleared.
FV	√	The flag is set if an invalid operation exception is generated, and otherwise left unchanged.
FO	√	The flag is set if an overflow exception is generated, and otherwise left unchanged.
FZ	–	
FU	√	The flag is set if an underflow exception is generated, and otherwise left unchanged.
FX	√	The flag is set if an inexact exception is generated, and otherwise left unchanged.

Note: The FX, FU, FO, and FV flags do not change if any of the exception enable bits EX, EU, EO, and EV is 1. The S and Z flags do not change when an exception is generated.

Instruction Format

Syntax	Processing Size	Operand			Code Size (Byte)
		src	src2	dest	
(1) FMUL src, dest	L	#IMM:32	–	Rd	7
	L	Rs	–	Rd	3
	L	[Rs].L	–	Rd	3
	L	dsp:8[Rs].L*	–	Rd	4
	L	dsp:16[Rs].L*	–	Rd	5
(2) FMUL src, src2, dest	L	Rs	Rs2	Rd	3

Note: * For the RX Family assembler manufactured by Renesas Electronics Corp., enter a scaled value (the actual value multiplied by 4) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 1020 (255 × 4) can be specified; with dsp:16, values from 0 to 262140 (65535 × 4) can be specified. The value divided by 4 will be stored in the instruction code.

Possible Exceptions

Unimplemented processing
Invalid operation
Overflow
Underflow
Inexact

Description Example

```
FMUL    R1, R2
FMUL    [R1], R2
FMUL    R1, R2, R3
```

Supplementary Description

- The following tables show the correspondences between src and dest values and the results of operations when DN = 0 and DN = 1.

When DN = 0

<table>
<tr><td colspan="2" rowspan="2"></td><td colspan="8">src</td></tr>
<tr><td>Normalized</td><td>+0</td><td>–0</td><td>+∞</td><td>–∞</td><td>Denormalized</td><td>QNaN</td><td>SNaN</td></tr>
<tr><td rowspan="8">dest or src2</td><td>Normalized</td><td colspan="3">Multiplication</td><td colspan="2">∞</td><td rowspan="5"></td><td rowspan="6"></td><td rowspan="7"></td></tr>
<tr><td>+0</td><td rowspan="2"></td><td>+0</td><td>–0</td><td colspan="2" rowspan="2">Invalid operation</td></tr>
<tr><td>–0</td><td>–0</td><td>+0</td></tr>
<tr><td>+∞</td><td rowspan="2">∞</td><td colspan="2" rowspan="2">Invalid operation</td><td>+∞</td><td>–∞</td></tr>
<tr><td>–∞</td><td>–∞</td><td>+∞</td></tr>
<tr><td>Denormalized</td><td colspan="6">Unimplemented processing</td></tr>
<tr><td>QNaN</td><td colspan="7">QNaN</td></tr>
<tr><td>SNaN</td><td colspan="8">Invalid operation</td></tr>
</table>

When DN = 1

		src						
		Normalized	+0, +Denormalized	–0, –Denormalized	+∞	–∞	QNaN	SNaN
dest or src2	Normalized	Multiplication			∞			
	+0, +Denormalized		+0	–0	Invalid operation			
	–0, –Denormalized		–0	+0				
	+∞	∞	Invalid operation		+∞	–∞		
	–∞				–∞	+∞		
	QNaN						QNaN	
	SNaN							Invalid operation

FSQRT

Floating-point square root
Floating-point SQuare RooT

FSQRT

Floating-point operation instruction
Instruction Code
Page: 244

Syntax

```
FSQRT    src, dest
```

Operation

```
dest = sqrt(src);
```

Function

- This instruction calculates the square root of the single-precision floating-point number stored in src and places the result in dest. Rounding of the result is in accord with the setting of the RM[1:0] bits in the FPSW.
- Handling of denormalized numbers depends on the setting of the DN bit in the FPSW.

Flag Change

Flag	Change	Condition
C	–	
Z	√	The flag is set if the result of the operation is +0 or –0; otherwise it is cleared.
S	√	The flag is set if the sign bit (bit 31) of the result of the operation is 1; otherwise it is cleared.
O	–	
CV	√	The flag is set if an invalid operation exception is generated; otherwise it is cleared.
CO	√	The value of the flag is 0.
CZ	√	The value of the flag is 0.
CU	√	The value of the flag is 0.
CX	√	The flag is set if an inexact exception is generated; otherwise it is cleared.
CE	√	The flag is set if an unimplemented processing is generated; otherwise it is cleared.
FV	√	The flag is set if an invalid operation exception is generated, and otherwise left unchanged.
FO	–	
FZ	–	
FU	–	
FX	√	The flag is set if an inexact exception is generated, and otherwise left unchanged.

Note: The FX, FU, FO, and FV flags do not change if any of the exception enable bits EX, EU, EO, and EV is 1. The S and Z flags do not change when an exception is generated.

Instruction Format

Syntax		Processing Size	Operand		Code Size (Byte)
			src	dest	
FSQRT	src, dest	L	#IMM:32	Rd	7
		L	Rs	Rd	3
		L	[Rs].L	Rd	3
		L	dsp:8[Rs].L*	Rd	4
		L	dsp:16[Rs].L*	Rd	5

Note: * For the RX Family assembler manufactured by Renesas Electronics Corp., enter a scaled value (the actual value multiplied by 4) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 1020 (255 × 4) can be specified; with dsp:16, values from 0 to 262140 (65535 × 4) can be specified. The value divided by 4 will be stored in the instruction code.

Possible Exceptions

Unimplemented processing
Invalid operation
Inexact

Description Example

```
FSQRT    R1, R2
FSQRT    [R1], R2
```

Supplementary Description

- The following tables show the correspondences between src values and the results of operations when DN = 0 and DN = 1.

When DN = 0

	src								
	+Normalized	–Normalized	+0	–0	+∞	–∞	Denormalized	QNaN	SNaN
Result	Square root	Invalid operation	+0	–0	+∞	Invalid operation	Unimplemented processing	QNaN	Invalid operation

When DN = 1

	src									
	+Normalized	–Normalized	+0	–0	+∞	–∞	+Denormalized	–Denormalized	QNaN	SNaN
Result	Square root	Invalid operation	+0	–0	+∞	Invalid operation	+0	–0	QNaN	Invalid operation

Rules for Generating QNaN When Invalid Operation is Generated

Source Operands	Operation Results
SNaN	The SNaN source operand converted into a QNaN
Other than above	7FFFFFFFh

Note: Corresponds to Table 1.6, Rules for Generating QNaNs.

FSUB

Floating-point subtraction
Floating-point SUBtract

FSUB

Floating-point operation instruction
Instruction Code
Page: 245

Syntax

```
(1) FSUB    src, dest
(2) FSUB    src, src2, dest
```

Operation

```
(1) dest = dest - src;
(2) dest = src2 - src;
```

Function

(1) This instruction subtracts the single-precision floating-point number stored in src from that stored in dest and places the result in dest.
(2) This instruction subtracts the single-precision floating-point number stored in src from that stored in src2 and places the result in dest.

- Rounding of the result is in accord with the setting of the RM[1:0] bits in the FPSW.
- Handling of denormalized numbers depends on the setting of the DN bit in the FPSW.
- The operation result is +0 when subtracting src from dest (src from src2) with both the same signs is exactly 0 except in the case of a rounding mode towards -∞. The operation result is -0 when the rounding mode is towards -∞.

Flag Change

Flag	Change	Condition
C	–	
Z	√	The flag is set if the result of the operation is +0 or –0; otherwise it is cleared.
S	√	The flag is set if the sign bit (bit 31) of the result of the operation is 1; otherwise it is cleared.
O	–	
CV	√	The flag is set if an invalid operation exception is generated; otherwise it is cleared.
CO	√	The flag is set if an overflow exception is generated; otherwise it is cleared.
CZ	√	The value of the flag is 0.
CU	√	The flag is set if an underflow exception is generated; otherwise it is cleared.
CX	√	The flag is set if an inexact exception is generated; otherwise it is cleared.
CE	√	The flag is set if an unimplemented processing is generated; otherwise it is cleared.
FV	√	The flag is set if an invalid operation exception is generated, and otherwise left unchanged.
FO	√	The flag is set if an overflow exception is generated, and otherwise left unchanged.
FZ	–	
FU	√	The flag is set if an underflow exception is generated, and otherwise left unchanged.
FX	√	The flag is set if an inexact exception is generated, and otherwise left unchanged.

Note: The FX, FU, FO, and FV flags do not change if any of the exception enable bits EX, EU, EO, and EV is 1. The S and Z flags do not change when an exception is generated.

Instruction Format

Syntax	Processing Size	Operand src	src2	dest	Code Size (Byte)
(1) FSUB src, dest	L	#IMM:32	–	Rd	7
	L	Rs	–	Rd	3
	L	[Rs].L	–	Rd	3
	L	dsp:8[Rs].L*	–	Rd	4
	L	dsp:16[Rs].L*	–	Rd	5
(2) FSUB src, src2, dest	L	Rs	Rs2	Rd	3

Note: * For the RX Family assembler manufactured by Renesas Electronics Corp., enter a scaled value (the actual value multiplied by 4) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 1020 (255 × 4) can be specified; with dsp:16, values from 0 to 262140 (65535 × 4) can be specified. The value divided by 4 will be stored in the instruction code.

Possible Exceptions

Unimplemented processing
Invalid operation
Overflow
Underflow
Inexact

Description Example

```
FSUB    R1, R2
FSUB    [R1], R2
FSUB    R1, R2, R3
```

Supplementary Description

- The following tables show the correspondences between src and dest values and the results of operations when DN = 0 and DN = 1.

When DN = 0

<table>
<tr><td colspan="2" rowspan="2"></td><td colspan="8">src</td></tr>
<tr><td>Normalized</td><td>+0</td><td>–0</td><td>+∞</td><td>–∞</td><td>Denormalized</td><td>QNaN</td><td>SNaN</td></tr>
<tr><td rowspan="8">dest</td><td>Normalized</td><td rowspan="3">Subtraction</td><td colspan="2"></td><td rowspan="3">–∞</td><td rowspan="4">+∞</td><td rowspan="6">Unimplemented processing</td><td rowspan="7">QNaN</td><td rowspan="8">Invalid operation</td></tr>
<tr><td>+0</td><td>*</td><td>+0</td></tr>
<tr><td>–0</td><td>–0</td><td>*</td></tr>
<tr><td>+∞</td><td colspan="3">+∞</td><td>Invalid operation</td></tr>
<tr><td>–∞</td><td colspan="4">–∞</td><td>Invalid operation</td></tr>
<tr><td>Denormalized</td><td colspan="5"></td></tr>
<tr><td>QNaN</td><td colspan="6"></td></tr>
<tr><td>SNaN</td><td colspan="7"></td></tr>
</table>

When DN = 1

		src						
		Normalized	+0, +Denormalized	–0, –Denormalized	+∞	–∞	QNaN	SNaN
dest	Normalized	Subtraction			–∞	+∞		
	+0, +Denormalized		*	+0				
	–0, –Denormalized		–0	*				
	+∞	+∞			Invalid operation			
	–∞	–∞				Invalid operation		
	QNaN						QNaN	
	SNaN							Invalid operation

Note: * The result is –0 when the rounding mode is set to rounding towards –∞ and +0 in other rounding modes.

FTOI

Floating point to integer conversion
Float TO Integer

FTOI

Floating-point operation instruction
Instruction Code
Page: 246

Syntax

```
FTOI    src, dest
```

Operation

```
dest = ( signed long ) src;
```

Function

- This instruction converts the single-precision floating-point number stored in src into a signed longword (32-bit) integer and places the result in dest.
- The result is always rounded towards 0, regardless of the setting of the RM[1:0] bits in the FPSW.

Flag Change

Flag	Change	Condition
C	–	
Z	√	The flag is set if the result of the operation is 0; otherwise it is cleared.
S	√	The flag is set if the sign bit (bit 31) of the result of the operation is 1; otherwise it is cleared.
O	–	
CV	√	The flag is set if an invalid operation exception is generated; otherwise it is cleared.
CO	√	The value of the flag is 0.
CZ	√	The value of the flag is 0.
CU	√	The value of the flag is 0.
CX	√	The flag is set if an inexact exception is generated; otherwise it is cleared.
CE	√	The flag is set if an unimplemented processing exception is generated; otherwise it is cleared.
FV	√	The flag is set if an invalid operation exception is generated; otherwise it does not change.
FO	–	
FZ	–	
FU	–	
FX	√	The flag is set if an inexact exception is generated; otherwise it does not change.

Note: The FX and FV flags do not change if any of the exception enable bits EX and EV is 1. The S and Z flags do not change when an exception is generated.

Instruction Format

	Processing	Operand		Code Size
Syntax	Size	src	dest	(Byte)
FTOI src, dest	L	Rs	Rd	3
	L	[Rs].L	Rd	3
	L	dsp:8[Rs].L*	Rd	4
	L	dsp:16[Rs].L*	Rd	5

Note: * For the RX Family assembler manufactured by Renesas Electronics Corp., enter a scaled value (the actual value multiplied by 4) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 1020 (255 × 4) can be specified; with dsp:16, values from 0 to 262140 (65535 × 4) can be specified. The value divided by 4 will be stored in the instruction code.

Possible Exceptions

Unimplemented processing
Invalid operation
Inexact

Description Example

```
FTOI    R1, R2
FTOI    [R1], R2
```

Supplementary Description

- The following tables show the correspondences between src and dest values and the results of operations when DN = 0 and DN = 1.

When DN = 0

src Value (exponent is shown without bias)		dest	Exception
src ≥ 0	+∞	When an invalid operation exception is generated with the EV bit = 1: No change	Invalid operation exception
	127 ≥ Exponent ≥ 31	Other cases: 7FFFFFFFh	
	30 ≥ Exponent ≥ –126	00000000h to 7FFFFF80h	None[*1]
	+Denormalized number	No change	Unimplemented processing exception
	+0	00000000h	None
src < 0	–0		
	–Denormalized number	No change	Unimplemented processing exception
	30 ≥ Exponent ≥ –126	00000000h to 80000080h	None[*1]
	127 ≥ Exponent ≥ 31	When an invalid operation exception is generated with the EV bit = 1: No change	Invalid operation exception[*2]
	–∞	Other cases: 80000000h	
NaN	QNaN	When an invalid operation exception is generated with the EV bit = 1: No change Other cases:	Invalid operation exception
	SNaN	Sign bit = 0: 7FFFFFFFh Sign bit = 1: 80000000h	

Notes: 1. An inexact exception occurs when the result is rounded.
2. No invalid operation exception occurs when src = CF000000h.

When DN = 1

src Value (exponent is shown without bias)		dest	Exception
src ≥ 0	+∞	When an invalid operation exception is generated with the EV bit = 1: No change	Invalid operation exception
	127 ≥ Exponent ≥ 31	Other cases: 7FFFFFFFh	
	30 ≥ Exponent ≥ –126	00000000h to 7FFFFF80h	None*1
	+0, +Denormalized number	00000000h	None
src < 0	–0, –Denormalized number		
	30 ≥ Exponent ≥ –126	00000000h to 80000080h	None*1
	127 ≥ Exponent ≥ 31	When an invalid operation exception is generated with the EV bit = 1: No change	Invalid operation exception*2
	–∞	Other cases: 80000000h	
NaN	QNaN	When an invalid operation exception is generated with the EV bit = 1: No change Other cases:	Invalid operation exception
	SNaN	Sign bit = 0: 7FFFFFFFh Sign bit = 1: 80000000h	

Notes: 1. An inexact exception occurs when the result is rounded.

2. No invalid operation exception occurs when src = CF000000h.

FTOU

Floating point to integer conversion
Float TO Unsigned integer

FTOU

Floating-point operation instruction
Instruction Code
Page: 246

Syntax

```
FTOU    src, dest
```

Operation

```
dest = ( unsigned long ) src;
```

Function

- This instruction converts the single-precision floating-point number stored in src into an unsigned longword (32-bit) integer and places the result in dest.
- The result is always rounded towards 0, regardless of the setting of the RM[1:0] bits in the FPSW.

Flag Change

Flag	Change	Condition
C	–	
Z	√	The flag is set if the result of the operation is 0; otherwise it is cleared.
S	√	The flag is set if bit 31 of the result of the operation is 1; otherwise it is cleared.
O	–	
CV	√	The flag is set if an invalid operation exception is generated; otherwise it is cleared.
CO	√	The value of the flag is 0.
CZ	√	The value of the flag is 0.
CU	√	The value of the flag is 0.
CX	√	The flag is set if an inexact exception is generated; otherwise it is cleared.
CE	√	The flag is set if an unimplemented processing is generated; otherwise it is cleared.
FV	√	The flag is set if an invalid operation exception is generated, and otherwise left unchanged.
FO	–	
FZ	–	
FU	–	
FX	√	The flag is set if an inexact exception is generated, and otherwise left unchanged.

Note: The FX and FV flags do not change if any of the exception enable bits EX and EV is 1. The S and Z flags do not change when an exception is generated.

Instruction Format

	Processing	Operand		Code Size
Syntax	Size	src	dest	(Byte)
FTOU src, dest	L	Rs	Rd	3
	L	[Rs].L	Rd	3
	L	dsp:8[Rs].L*	Rd	4
	L	dsp:16[Rs].L*	Rd	5

Note: * For the RX Family assembler manufactured by Renesas Electronics Corp., enter a scaled value (the actual value multiplied by 4) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 1020 (255 × 4) can be specified; with dsp:16, values from 0 to 262140 (65535 × 4) can be specified. The value divided by 4 will be stored in the instruction code.

Possible Exceptions

Unimplemented processing
Invalid operation
Inexact

Description Example

```
FTOU    R1, R2
FTOU    [R1], R2
```

Supplementary Description

- The following tables show the correspondences between src and dest values and the results of operations when DN = 0 and DN = 1.

When DN = 0

src Value (exponent is shown without bias)		dest	Exception
src ≥ 0	+∞	When an invalid operation exception is generated with the EV bit = 1: No change Other cases: 7FFFFFFFh	Invalid operation
	127 ≥ Exponent ≥ 32		
	31 ≥ Exponent ≥ –126	00000000h to 7FFFFF80h	None[*1]
	+Denormalized number	No change	Unimplemented processing
	+0	00000000h	None
src < 0	–0		
	–Denormalized number	No change	Unimplemented processing
	-Normalized number, -∞	When an invalid operation exception is generated with the EV bit = 1: No change Other cases: 00000000h	Invalid operation
NaN	QNaN	When an invalid operation exception is generated with the EV bit = 1: No change Other cases: Most significant bit = 0: FFFFFFFFh Most significant bit = 1: 00000000h	Invalid operation
	SNaN		

Note: 1.An inexact exception occurs when the result is rounded.

When DN = 1

src Value (exponent is shown without bias)		dest	Exception
src ≥ 0	+∞	When an invalid operation exception is generated with the EV bit = 1: No change Other cases: FFFFFFFFh	Invalid operation
	127 ≥ Exponent ≥ 32		
	31 ≥ Exponent ≥ –126	00000000h to FFFFFF00h	None[*1]
	+0, +Denormalized number	00000000h	None
src < 0	–0		
	-Normalized number, -∞	When an invalid operation exception is generated with the EV bit = 1: No change Other cases: 00000000h	Invalid operation
NaN	QNaN	When an invalid operation exception is generated with the EV bit = 1: No change Other cases: Sign bit = 0: FFFFFFFFh Sign bit = 1: 00000000h	Invalid operation
	SNaN		

Notes: 1. An inexact exception occurs when the result is rounded.

INT

Software interrupt
INTerrupt

INT

System manipulation instruction
Instruction Code
Page: 247

Syntax

```
INT     src
```

Operation

```
tmp0 = PSW;
U = 0;
I = 0;
PM = 0;
tmp1 = PC + 3;
PC = *(IntBase + src * 4);
SP = SP - 4;
*SP = tmp0;
SP = SP - 4;
*SP = tmp1;
```

Function

- This instruction generates the unconditional trap which corresponds to the number specified as src.
- The INT instruction number (src) is in the range $0 \leq \text{src} \leq 255$.
- This instruction causes a transition to supervisor mode, and clears the PM bit in the PSW to 0.
- This instruction clears the U and I bits in the PSW to 0.

Flag Change

- This instruction does not affect the states of flags.
- The state of the PSW before execution of this instruction is preserved on the stack.

Instruction Format

	Operand	Code Size
Syntax	src	(Byte)
INT src	#IMM:8	3

Description Example

```
INT     #0
```

ITOF

Integer to floating-point conversion
Integer TO Floating-point

ITOF

Floating-point operation instruction
Instruction Code
Page: 247

Syntax

```
ITOF    src, dest
```

Operation

```
dest = ( float ) src;
```

Function

- This instruction converts the signed longword (32-bit) integer stored in src into a single-precision floating-point number and places the result in dest. Rounding of the result is in accord with the setting of the RM[1:0] bits in the FPSW. 00000000h is handled as +0 regardless of the rounding mode.

Flag Change

Flag	Change	Condition
C	–	
Z	√	The flag is set if the result of the operation is +0; otherwise it is cleared.
S	√	The flag is set if the sign bit (bit 31) of the result of the operation is 1; otherwise it is cleared.
O	–	
CV	√	The value of the flag is 0.
CO	√	The value of the flag is 0.
CZ	√	The value of the flag is 0.
CU	√	The value of the flag is 0.
CX	√	The flag is set if an inexact exception is generated; otherwise it is cleared.
CE	√	The value of the flag is 0.
FV	–	
FO	–	
FZ	–	
FU	–	
FX	√	The flag is set if an inexact exception is generated; otherwise it does not change.

Note: The FX flag does not change if the exception enable bit EX is 1. The S and Z flags do not change when an exception is generated.

Instruction Format

Syntax	Processing Size	Operand		Code Size (Byte)
		src	dest	
ITOF src, dest	L	Rs	Rd	3
	L	[Rs].memex	Rd	3 (memex == UB) 4 (memex != UB)
	L	dsp:8[Rs].memex*	Rd	4 (memex == UB) 5 (memex != UB)
	L	dsp:16[Rs].memex*	Rd	5 (memex == UB) 6 (memex != UB)

Note: * For the RX Family assembler manufactured by Renesas Electronics Corp., enter a scaled value (the actual value multiplied by 2 when the size extension specifier is .W or .UW, or by 4 when the specifier is .L) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 510 (255 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 1020 (255 × 4) when the specifier is .L. With dsp:16, values from 0 to 131070 (65535 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 262140 (65535 × 4) when the specifier is .L. The value divided by 2 or 4 will be stored in the instruction code.

Possible Exceptions

Inexact

Description Example

```
ITOF    R1, R2
ITOF    [R1], R2
ITOF    16[R1].L, R2
```

JMP

Unconditional jump
JuMP

JMP

Branch instruction
Instruction Code
Page: 248

Syntax

```
JMP     src
```

Operation

```
PC = src;
```

Function

- This instruction branches to the instruction specified by src.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

	Operand	Code Size
Syntax	src	(Byte)
JMP src	Rs	2

Description Example

```
JMP     R1
```

JSR

Jump to a subroutine
Jump SubRoutine

JSR

Branch instruction
Instruction Code
Page: 248

Syntax

```
JSR      src
```

Operation

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

Note: * (PC + 2) is the address of the instruction following the JSR instruction.

Function

- This instruction causes the flow of execution to branch to the subroutine specified by src.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

	Operand	Code Size
Syntax	src	(Byte)
JSR src	Rs	2

Description Example

```
JSR      R1
```

MACHI

Multiply-Accumulate the high-order word
Multiply-ACcumulate HIgh-order word

MACHI

DSP instruction
Instruction Code
Page: 249

Syntax

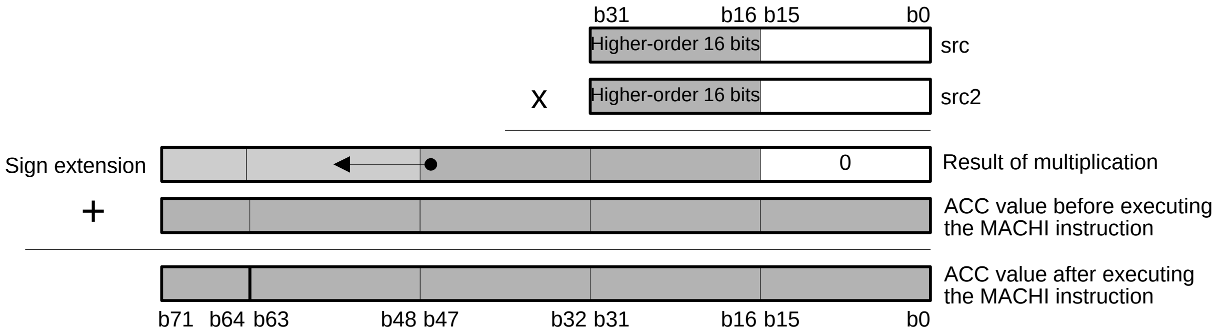
```
MACHI   src, src2, Adest
```

Operation

```
signed short tmp1, tmp2;
signed 72bit tmp3;
tmp1 = (signed short) (src >> 16);
tmp2 = (signed short) (src2 >> 16);
tmp3 = (signed long) tmp1 * (signed long) tmp2;
Adest = Adest + (tmp3 << 16);
```

Function

- This instruction multiplies the higher-order 16 bits of src by the higher-order 16 bits of src2, and adds the result to the value in the accumulator (ACC). The addition is performed with the least significant bit of the result of multiplication corresponding to bit 16 of ACC. The result of addition is stored in ACC. The higher-order 16 bits of src and the higher-order 16 bits of src2 are treated as signed integers.



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Operand			Code Size
	src	src2	Adest	(Byte)
MACHI src, src2, Adest	Rs	Rs2	A0, A1	3

Description Example

```
MACHI   R1, R2, A1
```

MACLH

Multiply-Accumulate the lower-order word and higher-order word
Multiply-ACcumulate Low-order word and High-order word

MACLH

DSP instruction
Instruction Code
Page: 249

Syntax

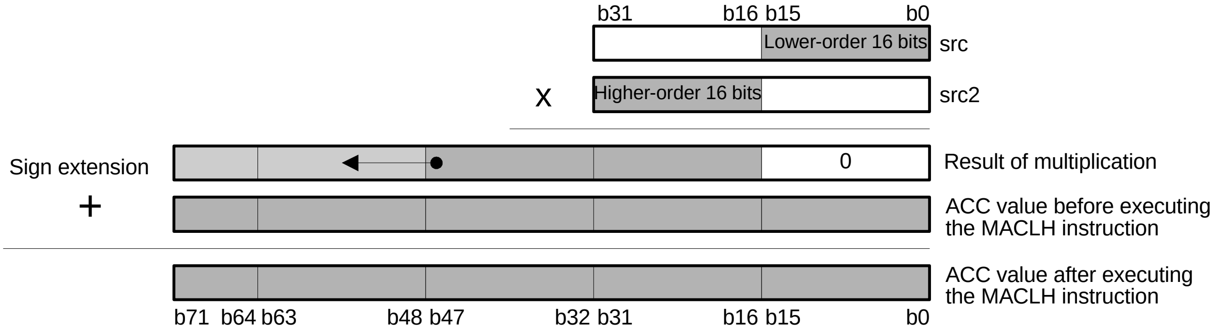
```
MACLH   src, src2, Adest
```

Operation

```
signed short tmp1, tmp2;
signed 72bit tmp3;
tmp1 = (signed short) src;
tmp2 = (signed short) (src2 >> 16);
tmp3 = (signed long) tmp1 * (signed long) tmp2;
Adest = Adest + (tmp3 << 16);
```

Function

- This instruction multiplies the lower-order 16 bits of src by the higher-order 16 bits of src2, and adds the result to the value in the accumulator (ACC). The addition is performed with the least significant bit of the result of multiplication corresponding to bit 16 of ACC. The result of addition is stored in ACC. The lower-order 16 bits of src and the higher-order 16 bits of src2 are treated as signed integers.



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Operand			Code Size
	src	src2	Adest	(Byte)
MACLH src, src2, Adest	Rs	Rs2	A0, A1	3

Description Example

```
MACLH   R1, R2, A1
```

MACLO

Multiply-Accumulate the low-order word
Multiply-ACcumulate LOw-order word

MACLO

DSP instruction
Instruction Code
Page: 250

Syntax

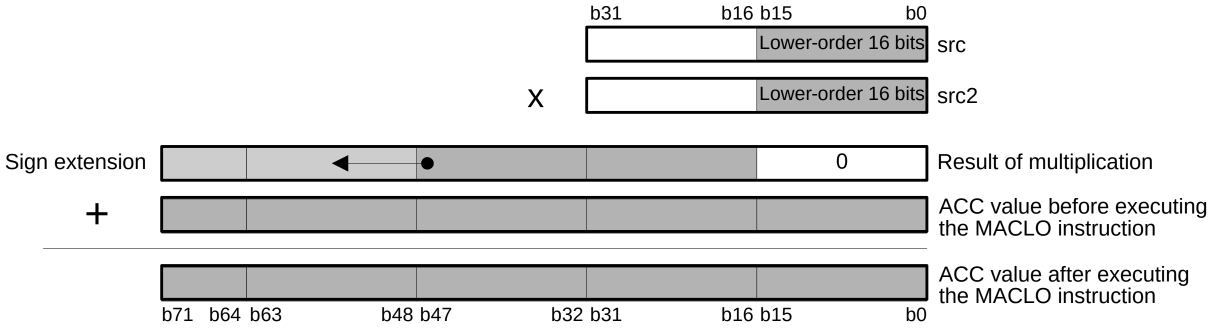
```
MACLO   src, src2, Adest
```

Operation

```
signed short tmp1, tmp2;
signed 72bit tmp3;
tmp1 = (signed short) src;
tmp2 = (signed short) src2;
tmp3 = (signed long) tmp1 * (signed long) tmp2;
Adest = Adest + (tmp3 << 16);
```

Function

- This instruction multiplies the lower-order 16 bits of src by the lower-order 16 bits of src2, and adds the result to the value in the accumulator (ACC). The addition is performed with the least significant bit of the result of multiplication corresponding to bit 16 of ACC. The result of addition is stored in ACC. The lower-order 16 bits of src and the lower-order 16 bits of src2 are treated as signed integers.



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Operand			Code Size
	src	src2	Adest	(Byte)
MACLO src, src2, Adest	Rs	Rs2	A0, A1	3

Description Example

```
MACLO   R1, R2, A1
```

MAX

Selecting the highest value
MAXimum value select

MAX

Arithmetic/logic instruction
Instruction Code
Page: 250

Syntax

```
MAX     src, dest
```

Operation

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

Function

- This instruction compares src and dest as signed values and places whichever is greater in dest.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

	Processing	Operand		Code Size
Syntax	**Size**	**src**	**dest**	**(Byte)**
MAX src, dest	L	#SIMM:8	Rd	4
	L	#SIMM:16	Rd	5
	L	#SIMM:24	Rd	6
	L	#IMM:32	Rd	7
	L	Rs	Rd	3
	L	[Rs].memex	Rd	3 (memex == UB) 4 (memex != UB)
	L	dsp:8[Rs].memex*	Rd	4 (memex == UB) 5 (memex != UB)
	L	dsp:16[Rs].memex*	Rd	5 (memex == UB) 6 (memex != UB)

Note: * For the RX Family assembler manufactured by Renesas Electronics Corp., enter a scaled value (the actual value multiplied by 2 when the size extension specifier is .W or .UW, or by 4 when the specifier is .L) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 510 (255 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 1020 (255 × 4) when the specifier is .L. With dsp:16, values from 0 to 131070 (65535 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 262140 (65535 × 4) when the specifier is .L. The value divided by 2 or 4 will be stored in the instruction code.

Description Example

```
MAX     #10, R2
MAX     R1, R2
MAX     [R1], R2
MAX     3[R1].B, R2
```

MIN

Selecting the lowest value
MINimum value select

MIN

Arithmetic/logic instruction
Instruction Code
Page: 252

Syntax

```
MIN     src, dest
```

Operation

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

Function

- This instruction compares src and dest as signed values and places whichever is smaller in dest.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

	Processing	Operand		Code Size
Syntax	**Size**	**src**	**dest**	**(Byte)**
MIN src, dest	L	#SIMM:8	Rd	4
	L	#SIMM:16	Rd	5
	L	#SIMM:24	Rd	6
	L	#IMM:32	Rd	7
	L	Rs	Rd	3
	L	[Rs].memex	Rd	3 (memex == UB) 4 (memex != UB)
	L	dsp:8[Rs].memex*	Rd	4 (memex == UB) 5 (memex != UB)
	L	dsp:16[Rs].memex*	Rd	5 (memex == UB) 6 (memex != UB)

Note: * For the RX Family assembler manufactured by Renesas Electronics Corp., enter a scaled value (the actual value multiplied by 2 when the size extension specifier is .W or .UW, or by 4 when the specifier is .L) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 510 (255 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 1020 (255 × 4) when the specifier is .L. With dsp:16, values from 0 to 131070 (65535 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 262140 (65535 × 4) when the specifier is .L. The value divided by 2 or 4 will be stored in the instruction code.

Description Example

```
MIN     #10, R2
MIN     R1, R2
MIN     [R1], R2
MIN     3[R1].B, R2
```

MOV

Transferring data
MOVe

MOV

Data transfer instruction
Instruction Code
Page: 253

Syntax

```
MOV.size   src, dest
```

Operation

```
dest = src;
```

Function

- This instruction transfers src to dest as listed in the following table.

src	dest	Function
Immediate value	Register	Transfers the immediate value to the register. When the immediate value is specified in less than 32 bits, it is transferred to the register after being zero-extended if specified as #UIMM and sign-extended if specified as #SIMM.
Immediate value	Memory location	Transfers the immediate value to the memory location in the specified size. When the immediate value is specified with a width in bits smaller than the specified size, it is transferred to the memory location after being zero-extended if specified as #UIMM and sign-extended if specified as #SIMM.
Register	Register	Transfers the data in the source register (src) to the destination register (dest). When the size specifier is .B, the data is transferred to the register (dest) after the byte of data in the LSB of the register (src) has been sign-extended to form a longword of data. When the size specifier is .W, the data is transferred to the register (dest) after the word of data from the LSB end of the register (src) has bee sign-extended to form a longword of data.
Register	Memory location	Transfers the data in the register to the memory location. When the size specifier is .B, the byte of data in the LSB of the register is transferred. When the size specifier is .W, the word of data from the LSB end of the register is transferred.
Memory location	Register	Transfers the data at the memory location to the register. When the size specifier is .B or .W, the data at the memory location are sign-extended to form a longword, which is transferred to the register.
Memory location	Memory location	Transfers the data with the specified size at the source memory location (src) to the specified size at the destination memory location (dest).

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

		Processing	Operand		Code Size
Syntax	Size	Size	src	dest	(Byte)
MOV.size src, dest	Store (short format)				
	B/W/L	size	Rs (Rs = R0 to R7)	dsp:5[Rd]*1 (Rd = R0 to R7)	2
	Load (short format)				
	B/W/L	L	dsp:5[Rs]*1 (Rs = R0 to R7)	Rd (Rd = R0 to R7)	2
	Set immediate value to register (short format)				
	L	L	#UIMM:4	Rd	2

		Processing	Operand		Code Size
Syntax	Size	Size	src	dest	(Byte)
MOV.size src, dest	Set immediate value to memory location (short format)				
	B	B	#IMM:8	dsp:5[Rd]*1 (Rd = R0 to R7)	3
	W/L	size	#UIMM:8	dsp:5[Rd]*1 (Rd = R0 to R7)	3
	Set immediate value to register				
	L	L	#UIMM:8*2	Rd	3
	L	L	#SIMM:8*2	Rd	3
	L	L	#SIMM:16	Rd	4
	L	L	#SIMM:24	Rd	5
	L	L	#IMM:32	Rd	6
	Data transfer between registers (sign extension)				
	B/W	L	Rs	Rd	2
	Data transfer between registers (no sign extension)				
	L	L	Rs	Rd	2
	Set immediate value to memory location				
	B	B	#IMM:8	[Rd]	3
	B	B	#IMM:8	dsp:8[Rd]*1	4
	B	B	#IMM:8	dsp:16[Rd]*1	5
	W	W	#SIMM:8	[Rd]	3
	W	W	#SIMM:8	dsp:8[Rd]*1	4
	W	W	#SIMM:8	dsp:16[Rd]*1	5
	W	W	#IMM:16	[Rd]	4
	W	W	#IMM:16	dsp:8[Rd]*1	5
	W	W	#IMM:16	dsp:16[Rd]*1	6
	L	L	#SIMM:8	[Rd]	3
	L	L	#SIMM:8	dsp:8[Rd]*1	4
	L	L	#SIMM:8	dsp:16 [Rd]*1	5
	L	L	#SIMM:16	[Rd]	4
	L	L	#SIMM:16	dsp:8[Rd]*1	5
	L	L	#SIMM:16	dsp:16 [Rd]*1	6
	L	L	#SIMM:24	[Rd]	5
	L	L	#SIMM:24	dsp:8[Rd]*1	6
	L	L	#SIMM:24	dsp:16 [Rd]*1	7
	L	L	#IMM:32	[Rd]	6
	L	L	#IMM:32	dsp:8[Rd]*1	7
	L	L	#IMM:32	dsp:16 [Rd]*1	8
	Load				
	B/W/L	L	[Rs]	Rd	2
	B/W/L	L	dsp:8[Rs]*1	Rd	3
	B/W/L	L	dsp:16[Rs]*1	Rd	4
	B/W/L	L	[Ri, Rb]	Rd	3
	Store				
	B/W/L	size	Rs	[Rd]	2
	B/W/L	size	Rs	dsp:8[Rd]*1	3
	B/W/L	size	Rs	dsp:16[Rd]*1	4
	B/W/L	size	Rs	[Ri, Rb]	3

Syntax	Size	Processing Size	Operand src	Operand dest	Code Size (Byte)
MOV.size src, dest	Data transfer between memory locations				
	B/W/L	size	[Rs]	[Rd]	2
	B/W/L	size	[Rs]	dsp:8[Rd]*1	3
	B/W/L	size	[Rs]	dsp:16[Rd]*1	4
	B/W/L	size	dsp:8[Rs]*1	[Rd]	3
	B/W/L	size	dsp:8[Rs]*1	dsp:8[Rd]*1	4
	B/W/L	size	dsp:8[Rs]*1	dsp:16[Rd]*1	5
	B/W/L	size	dsp:16[Rs]*1	[Rd]	4
	B/W/L	size	dsp:16[Rs]*1	dsp:8[Rd]*1	5
	B/W/L	size	dsp:16[Rs]*1	dsp:16[Rd]*1	6
	Store with post-increment*3				
	B/W/L	size	Rs	[Rd+]	3
	Store with pre-decrement*3				
	B/W/L	size	Rs	[–Rd]	3
	Load with post-increment*4				
	B/W/L	L	[Rs+]	Rd	3
	Load with pre-decrement*4				
	B/W/L	L	[–Rs]	Rd	3

Notes: 1. For the RX Family assembler manufactured by Renesas Electronics Corp., enter a scaled value (the actual value multiplied by 2 when the size extension specifier is .W, or by 4 when the specifier is .L) as the displacement value (dsp:5, dsp:8, dsp:16). With dsp:5, values from 0 to 62 (31 × 2) can be specified when the size specifier is .W, or values from 0 to 124 (31 × 4) when the specifier is .L. With dsp:8, values from 0 to 510 (255 × 2) can be specified when the size specifier is .W, or values from 0 to 1020 (255 × 4) when the specifier is .L. With dsp:16, values from 0 to 131070 (65535 × 2) can be specified when the size specifier is .W, or values from 0 to 262140 (65535 × 4) when the specifier is .L. The value divided by 2 or 4 will be stored in the instruction code.

2. For values from 0 to 127, an instruction code for zero extension is always selected.

3. In cases of store with post-increment and store with pre-decrement, if the same register is specified for Rs and Rd, the value before updating the address is transferred as the source.

4. In cases of load with post-increment and load with pre-decrement, if the same register is specified for Rs and Rd, the data transferred from the memory location are saved in Rd.

Description Example

```
MOV.L   #0, R2
MOV.L   #128:8, R2
MOV.L   #-128:8, R2
MOV.L   R1, R2
MOV.L   #0, [R2]
MOV.W   [R1], R2
MOV.W   R1, [R2]
MOV.W   [R1, R2], R3
MOV.W   R1, [R2, R3]
MOV.W   [R1], [R2]
MOV.B   R1, [R2+]
MOV.B   [R1+], R2
MOV.B   R1, [-R2]
MOV.B   [-R1], R2
```

MOVCO

Storing with LI flag clear
MOVe-COnditional

Data transfer instruction
Instruction Code
Page: 258

Syntax

```
MOVCO    src, dest
```

Operation

```
if (LI == 1) {dest=src;src=0;}
else { src=1; }
LI = 0;
```

Function

When the LI flag is 1, data in src (register) is stored in dest (memory) and the LI flag and src are cleared to 0.
When the LI flag is 0, data is not stored in src. Instead, 1 is set to src.

The LI flag is in the inside of CPU. The bit can be accessed only by MOVCO, MOVLI, RTE or RTFI instruction.
Customer can not access the LI flag directly.

Before executing the MOVCO instruction, execute the MOVLI instruction for the same address.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

	Processing	Operand		Code Size
Syntax	Size	src	dest	(Byte)
MOVCO src, dest	L	Rs	[Rd]	3

Description Example

```
MOVCO    R1, [R2]
```

MOVLI

Loading with LI flag set
MOVe LInked

MOVLI

Data transfer instruction
Instruction Code
Page: 258

Syntax

```
MOVLI    src, dest
```

Operation

```
LI = 1;
dest = src;
```

Function

This instruction transfers the longword data in src (memory) to dest (register).
This instruction sets the LI flag along with the normal load operation.

The LI flag is cleared when the conditions below are satisfied.
When an MOVCO instruction is executed
When an RTE or RTFI instruction is executed.

The LI flag is in the inside of CPU. The bit can be accessed only by MOVCO, MOVLI, RTE or RTFI instruction.
Customer can not access the LI flag directly.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

	Processing	Operand		Code Size
Syntax	**Size**	**src**	**dest**	**(Byte)**
MOVLI src, dest	L	[Rs]	Rd	3

Description Example

```
MOVLI    [R1], R2
```

MOVU

Transfer unsigned data
MOVe Unsigned data

MOVU

Data transfer instruction
Instruction Code
Page: 259

Syntax

```
MOVU.size  src, dest
```

Operation

```
dest = src;
```

Function

- This instruction transfers src to dest as listed in the following table.

src	dest	Function
Register	Register	Transfers the byte or word of data from the LSB in the source register (src) to the destination register (dest), after zero-extension to form a longword data.
Memory location	Register	Transfers the byte or word of data at the memory location to the register, after zero-extension to form a longword data.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Size	Processing Size	Operand		Code Size (Byte)
			src	dest	
MOVU.size src, dest	Load (short format)				
	B/W	L	dsp:5[Rs][*1] (Rs = R0 to R7)	Rd (Rd = R0 to R7)	2
	Data transfer between registers (zero extension)				
	B/W	L	Rs	Rd	2
	Load				
	B/W	L	[Rs]	Rd	2
	B/W	L	dsp:8[Rs][*1]	Rd	3
	B/W	L	dsp:16[Rs][*1]	Rd	4
	B/W	L	[Ri, Rb]	Rd	3
	Load with post-increment[*2]				
	B/W	L	[Rs+]	Rd	3
	Load with pre-decrement[*2]				
	B/W	L	[–Rs]	Rd	3

Notes: 1. For the RX Family assembler manufactured by Renesas Electronics Corp., enter a scaled value (the actual value multiplied by 2 when the size extension specifier is .W) as the displacement value (dsp:5, dsp:8, dsp:16). With dsp:5, values from 0 to 62 (31 × 2) can be specified when the size specifier is .W. With dsp:8, values from 0 to 510 (255 × 2) can be specified when the size specifier is .W. With dsp:16, values from 0 to 131070 (65535 × 2) can be specified when the size specifier is .W. The value divided by 2 will be stored in the instruction code.

2. In cases of load with post-increment and load with pre-decrement, if the same register is specified for Rs and Rd, the data transferred from the memory location are saved in Rd.

Description Example

```
MOVU.W  2[R1], R2
MOVU.W  R1, R2
MOVU.B  [R1+], R2
MOVU.B  [-R1], R2
```

MSBHI

Multiply-Subtract the higher-order word
Multiply-SuBtract HIgh-order word

MSBHI

DSP instruction
Instruction Code
Page: 260

Syntax

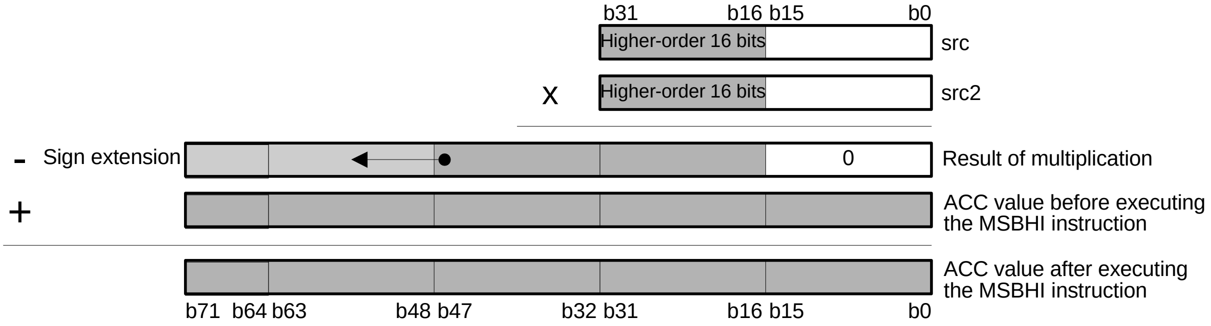
```
MSBHI   src, src2, Adest
```

Operation

```
signed short tmp1, tmp2;
signed 72bit tmp3;
tmp1 = (signed short) (src >> 16);
tmp2 = (signed short) (src2 >> 16);
tmp3 = (signed long) tmp1 * (signed long) tmp2;
Adest = Adest - (tmp3 << 16);
```

Function

- This instruction multiplies the higher-order 16 bits of src by the higher-order 16 bits of src2, and subtracts the result from the value in the accumulator (ACC). The subtraction is performed with the least significant bit of the result of multiplication corresponding to bit 16 of ACC. The result of subtraction is stored in ACC. The higher-order 16 bits of src and the higher-order 16 bits of src2 are treated as signed integers.



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Operand src	 src2	 Adest	Code Size (Byte)
MSBHI src, src2, Adest	Rs	Rs2	A0, A1	3

Description Example

```
MSBHI   R1, R2, A1
```

MSBLH

Multiply-Subtract the lower-order word
Multiply-SuBtract Low-order word and
High-order word

MSBLH

DSP instruction
Instruction Code
Page: 260

Syntax

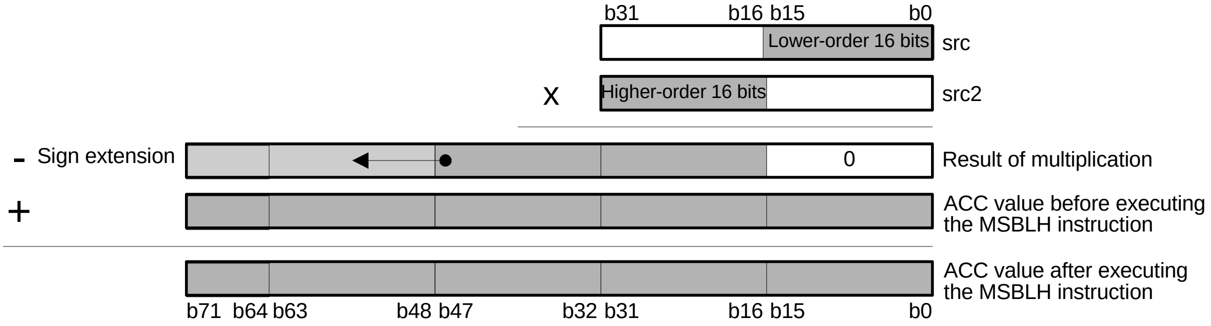
```
MSBLH   src, src2, Adest
```

Operation

```
signed short tmp1, tmp2;
signed 72bit tmp3;
tmp1 = (signed short) src;
tmp2 = (signed short) (src2 >> 16);
tmp3 = (signed long) tmp1 * (signed long) tmp2;
Adest = Adest - (tmp3 << 16);
```

Function

- This instruction multiplies the lower-order 16 bits of src by the higher-order 16 bits of src2, and subtracts the result from the value in the accumulator (ACC). The subtraction is performed with the least significant bit of the result of multiplication corresponding to bit 16 of ACC. The result of subtraction is stored in ACC. The lower-order 16 bits of src and the higher-order 16 bits of src2 are treated as signed integers.



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Operand			Code Size
	src	src2	Adest	(Byte)
MSBLH src, src2, Adest	Rs	Rs2	A0, A1	3

Description Example

```
MSBLH   R1, R2, A1
```

MSBLO

Multiply-Subtract the lower-order word
Multiply-SuBtract LOw-order word

MSBLO

DSP instruction
Instruction Code
Page: 261

Syntax

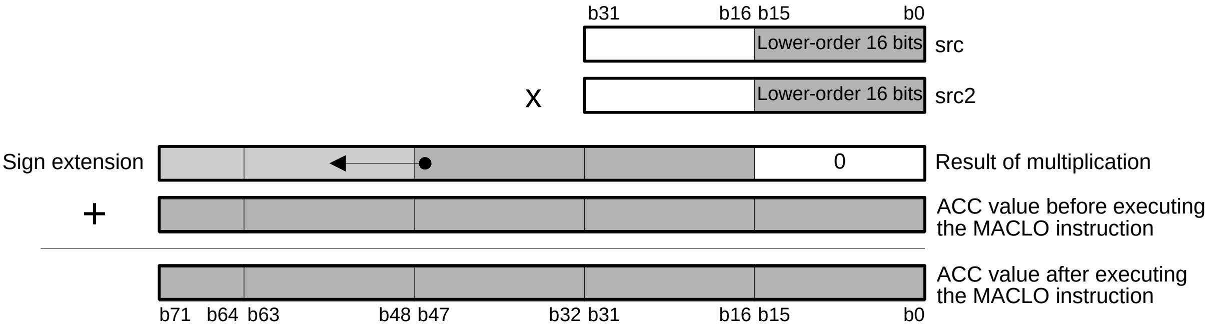
```
MSBLO   src, src2, Adest
```

Operation

```
signed short tmp1, tmp2;
signed 72bit tmp3;
tmp1 = (signed short) src;
tmp2 = (signed short) src2;
tmp3 = (signed long) tmp1 * (signed long) tmp2;
Adest = Adest - (tmp3 << 16);
```

Function

- This instruction multiplies the lower-order 16 bits of src by the lower-order 16 bits of src2, and subtracts the result from the value in the accumulator (ACC). The subtraction is performed with the least significant bit of the result of multiplication corresponding to bit 16 of ACC. The result of subtraction is stored in ACC. The lower-order 16 bits of src and the lower-order 16 bits of src2 are treated as signed integers.



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Operand src	Operand src2	Operand Adest	Code Size (Byte)
MSBLO src, src2, Adest	Rs	Rs2	A0, A1	3

Description Example

```
MSBLO   R1, R2, A1
```

MUL

Multiplication
MULtiply

MUL

Arithmetic/logic instruction
Instruction Code
Page: 261

Syntax

```
(1) MUL     src, dest
(2) MUL     src, src2, dest
```

Operation

```
(1) dest = src * dest;
(2) dest = src * src2;
```

Function

(1) This instruction multiplies src and dest and places the result in dest.
- The calculation is performed in 32 bits and the lower-order 32 bits of the result are placed.
- The operation result will be the same whether a singed or unsigned multiply is executed.

(2) This instruction multiplies src and src2 and places the result in dest.
- The calculation is performed in 32 bits and the lower-order 32 bits of the result are placed.
- The operation result will be the same whether a singed or unsigned multiply is executed.

Note: The accumulator (ACC0) is used to perform the function. The value of ACC0 after executing the instruction is undefined.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

	Processing	Operand			Code Size
Syntax	Size	src	src2	dest	(Byte)
(1) MUL src, dest	L	#UIMM:4	–	Rd	2
	L	#SIMM:8	–	Rd	3
	L	#SIMM:16	–	Rd	4
	L	#SIMM:24	–	Rd	5
	L	#IMM:32	–	Rd	6
	L	Rs	–	Rd	2
	L	[Rs].memex	–	Rd	2 (memex == UB) 3 (memex != UB)
	L	dsp:8[Rs].memex*	–	Rd	3 (memex == UB) 4 (memex != UB)
	L	dsp:16[Rs].memex*	–	Rd	4 (memex == UB) 5 (memex != UB)
(2) MUL src, src2, dest	L	Rs	Rs2	Rd	3

Note: * For the RX Family assembler manufactured by Renesas Electronics Corp., enter a scaled value (the actual value multiplied by 2 when the size extension specifier is .W or .UW, or by 4 when the specifier is .L) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 510 (255 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 1020 (255 × 4) when the specifier is .L. With dsp:16, values from 0 to 131070 (65535 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 262140 (65535 × 4) when the specifier .L. The value divided by 2 or 4 will be stored in the instruction code.

Description Example

```
MUL     #10, R2
MUL     R1, R2
MUL     [R1], R2
MUL     4[R1].W, R2
MUL     R1, R2, R3
```

MULHI

Multiply the high-order word
MULtiply HIgh-order word

MULHI

DSP instruction
Instruction Code
Page: 263

Syntax

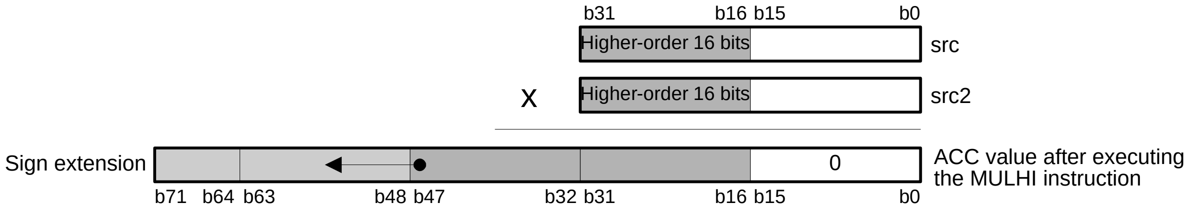
```
MULHI   src, src2, Adest
```

Operation

```
signed short tmp1, tmp2;
signed 72bit tmp3;
tmp1 = (signed short) (src >> 16);
tmp2 = (signed short) (src2 >> 16);
tmp3 = (signed long) tmp1 * (signed long) tmp2;
Adest = (tmp3 << 16);
```

Function

- This instruction multiplies the higher-order 16 bits of src by the higher-order 16 bits of src2, and stores the result in the accumulator (ACC). When the result is stored, the least significant bit of the result corresponds to bit 16 of ACC, and the section corresponding to bits 71 to 48 of ACC is sign-extended. Moreover, bits 15 to 0 of ACC are cleared to 0. The higher-order 16 bits of src and the higher-order 16 bits of src2 are treated as signed integers.



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Operand src	 src2	 Adest	Code Size (Byte)
MULHI src, src2, Adest	Rs	Rs2	A0, A1	3

Description Example

```
MULHI   R1, R2, A1
```

MULLH

Multiply the lower-order word and higher-order word
Multiply Low-order word and High-order word

MULLH

DSP instruction
Instruction Code
Page: 263

Syntax

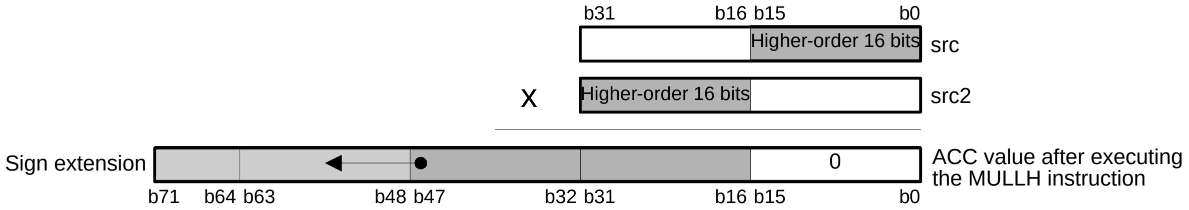
```
MULLH   src, src2, Adest
```

Operation

```
signed short tmp1, tmp2;
signed 72bit tmp3;
tmp1 = (signed short) src;
tmp2 = (signed short) (src2 >> 16);
tmp3 = (signed long) tmp1 * (signed long) tmp2;
Adest = (tmp3 << 16);
```

Function

- This instruction multiplies the lower-order 16 bits of src by the higher-order 16 bits of src2, and stores the result in the accumulator (ACC). When the result is stored, the least significant bit of the result corresponds to bit 16 of ACC, and the section corresponding to bits 71 to 48 of ACC is sign-extended. Moreover, bits 15 to 0 of ACC are cleared to 0. The lower-order 16 bits of src and the higher-order 16 bits of src2 are treated as signed integers.



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

	Operand			Code Size
Syntax	src	src2	Adest	(Byte)
MULLH src, src2, Adest	Rs	Rs2	A0, A1	3

Description Example

```
MULLH   R1, R2, A1
```

MULLO

Multiply the low-order word
MULtiply LOw-order word

MULLO

DSP instruction
Instruction Code
Page: 264

Syntax

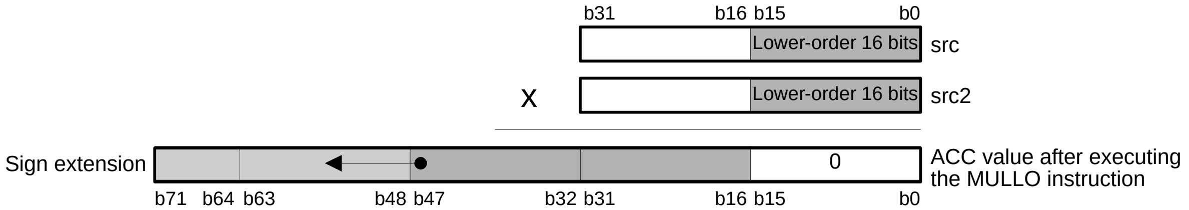
```
MULLO   src, src2, Adest
```

Operation

```
signed short tmp1, tmp2;
signed 72bit tmp3;
tmp1 = (signed short) src;
tmp2 = (signed short) src2;
tmp3 = (signed long) tmp1 * (signed long) tmp2;
Adest = (tmp3 << 16);
```

Function

- This instruction multiplies the lower-order 16 bits of src by the lower-order 16 bits of src2, and stores the result in the accumulator (ACC). When the result is stored, the least significant bit of the result corresponds to bit 16 of ACC, and the section corresponding to bits 71 to 48 of ACC is sign-extended. Moreover, bits 15 to 0 of ACC are cleared to 0. The lower-order 16 bits of src and the lower-order 16 bits of src2 are treated as signed integers.



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Operand			Code Size
	src	src2	Adest	(Byte)
MULLO src, src2, Adest	Rs	Rs2	A0, A1	3

Description Example

```
MULLO   R1, R2, A1
```

MVFACGU

Move the guard longword from the accumulator
MoVe From ACcumulator GUard longword

MVFACGU

DSP instruction
Instruction Code
Page: 264

Syntax

```
MVFACGU  src, Asrc, dest
```

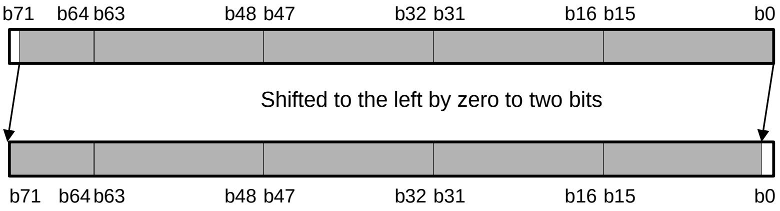
Operation

```
signed 72bit tmp;
tmp = (signed 72bit) Asrc << src;
dest = (signed long) (tmp >> 64);
```

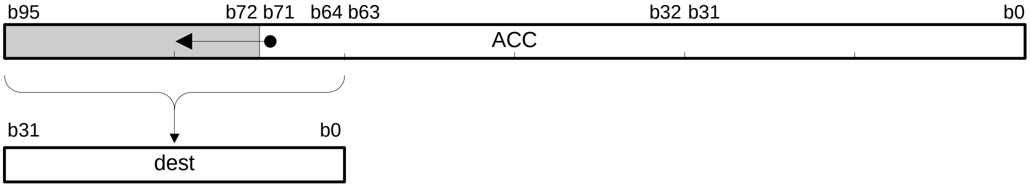
Function

- The MVFACGU instruction is executed according to the following procedures.

  Processing 1:
  The value of the accumulator is shifted to the left by zero to two bits as specified by src.



  Processing 2:
  This instruction moves the higher-order 32 bits of the accumulator (ACC) to dest.



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Operand			Code Size (Byte)
	src	Asrc	dest	
MVFACGU src, Asrc, dest	#IMM:2 (IMM:2 = 0 to 2)	A0, A1	Rd	3

Description Example

```
MVFACGU  #1, A1, R1
```

MVFACHI

Move the high-order longword from accumulator
MoVe From ACcumulator HIgh-order longword

MVFACHI

DSP instruction
Instruction Code
Page: 265

Syntax

```
MVFACHI  src, Asrc, dest
```

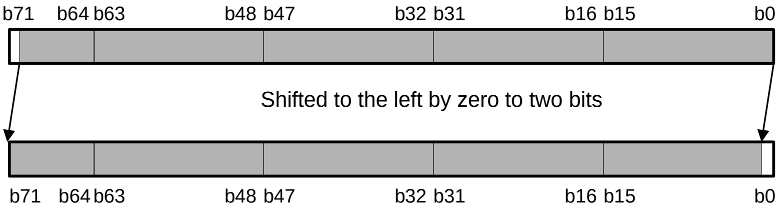
Operation

```
signed 72bit tmp;
tmp = (signed 72bit) Asrc << src;
dest = (signed long) (tmp >> 32);
```

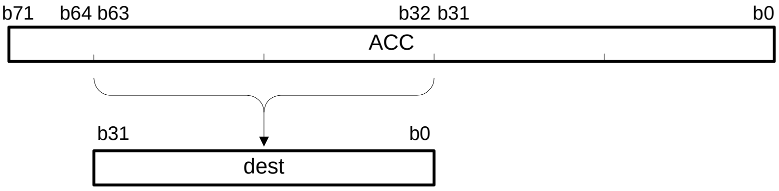
Function

- The MVFACHI instruction is executed according to the following procedures.

Processing 1:
The value of the accumulator is shifted to the left by zero to two bits as specified by src.



Processing 2:
This instruction moves the contents of bits 63 to 32 of the accumulator (ACC) to dest.



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Operand src	 Asrc	 dest	Code Size (Byte)
MVFACHI src, Asrc, dest	#IMM:2 (IMM:2 = 0 to 2)	A0, A1	Rd	3

Description Example

```
MVFACHI  #1, A1, R1
```

MVFACLO

Move the lower-order longword from the accumulator
MoVe From ACcumulator LOw-order longword

MVFACLO

DSP instruction
Instruction Code
Page: 265

Syntax

```
MVFACLO  src, Asrc, dest
```

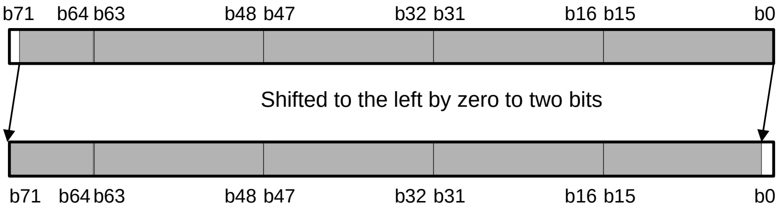
Operation

```
signed 72bit tmp;
tmp = (signed 72bit) Asrc << src;
dest = (signed long) tmp;
```

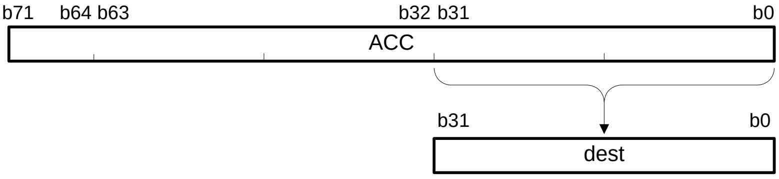
Function

- The MVFACLO instruction is executed according to the following procedures.

Processing 1:
The value of the accumulator is shifted to the left by zero to two bits as specified by src.



Processing 2:
This instruction moves the contents of bits 31 to 0 of the accumulator (ACC) to dest.



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Operand			Code Size (Byte)
	src	Asrc	dest	
MVFACLO src, Asrc, dest	#IMM:2 (IMM:2 = 0 to 2)	A0, A1	Rd	3

Description Example

```
MVFACLO  #1, A1, R1
```

MVFACMI

Move the middle-order longword from the accumulator

MoVe From ACcumulator MIddle-order longword

MVFACMI

DSP instruction

Instruction Code

Page: 266

Syntax

```
MVFACMI  src, Asrc, dest
```

Operation

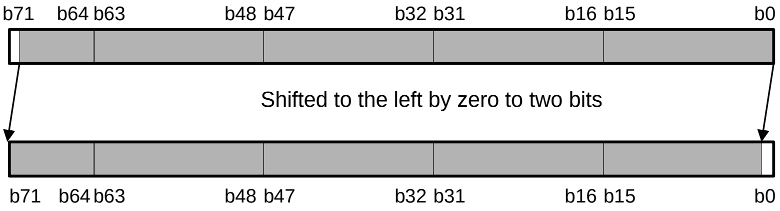
```
signed 72bit tmp;
tmp = (signed 72bit) Asrc << src;
dest = (signed long) (Asrc >> 16);
```

Function

- The MVFACMI instruction is executed according to the following procedures.

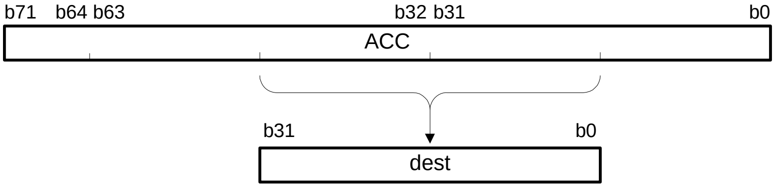
Processing 1:

The value of the accumulator is shifted to the left by zero to two bits as specified by src.



Processing 2:

This instruction moves the contents of bits 47 to 16 of the accumulator (ACC) to dest.



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Operand			Code Size (Byte)
	src	Asrc	dest	
MVFACMI src, Asrc, dest	#IMM:2 (IMM:2 = 0 to 2)	A0, A1	Rd	3

Description Example

```
MVFACMI  #1, A1, R1
```

MVFC

Transfer from a control register
MoVe From Control register

MVFC

System manipulation instruction
Instruction Code
Page: 266

Syntax

```
MVFC    src, dest
```

Operation

```
dest = src;
```

Function

- This instruction transfers src to dest.
- When the PC is specified as src, this instruction transfers its own address to dest.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

		Operand		
Syntax	**Processing Size**	**src***	**dest**	**Code Size (Byte)**
MVFC src, dest	L	Rx	Rd	3

Note: * Selectable src: Registers PC, ISP, USP, INTB, EXTB, PSW, BPC, BPSW, FINTV, and FPSW

Description Example

```
MVFC    USP, R1
```

MVTACGU *Move the guard longword to the accumulator* MoVe To ACcumulator GUard longword MVTACG

DSP instruction
Instruction Code
Page: 267

Syntax

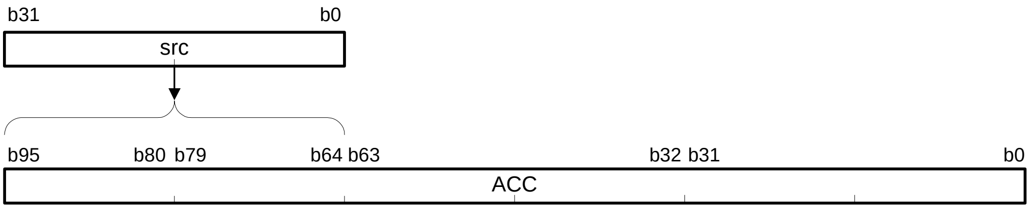
```
MVTACGU  src, Adest
```

Operation

```
Adest = (Adest & 00FFFFFFFFFFFFFFFFh) | ((signed 72bit) src << 64);
```

Function

- This instruction moves the contents of src to the most significant 32 bits (bits 95 to 64) of the accumulator (ACC).



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Operand		Code Size (Byte)
	src	Adest	
MVTACGU src, Adest	Rs	A0, A1	3

Description Example

```
MVTACGU  R1, A1
```

MVTACHI

Move the high-order longword to the accumulator
MoVe To ACcumulator HIgh-order longword

MVTACHI

DSP instruction
Instruction Code
Page: 267

Syntax

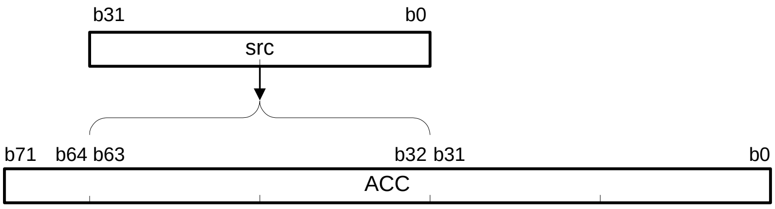
```
MVTACHI  src, Adest
```

Operation

```
Adest = (Adest & FF00000000FFFFFFFFh) | ((signed 72bit) src << 32);
```

Function

- This instruction moves the contents of src to the higher-order 32 bits (bits 63 to 32) of the accumulator (ACC).



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

	Operand		Code Size
Syntax	src	Adest	(Byte)
MVTACHI src, Adest	Rs	A0, A1	3

Description Example

```
MVTACHI  R1, A1
```

MVTACLO

Move the low-order longword to the accumulator
MoVe To ACcumulator LOw-order longword

MVTACLO

DSP instruction
Instruction Code
Page: 268

Syntax

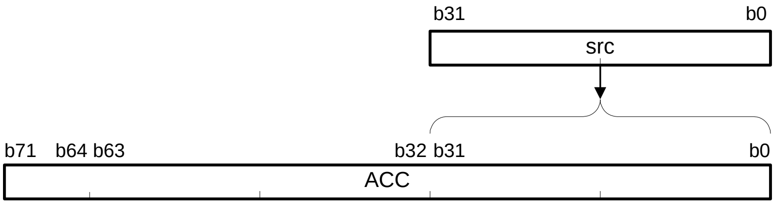
```
MVTACLO  src, Adest
```

Operation

```
Adest = (Adest & FFFFFFFFFF00000000h) | (unsigned 72bit) src;
```

Function

- This instruction moves the contents of src to the lower-order 32 bits (bits 31 to 0) of the accumulator (ACC).



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

	Operand		Code Size
Syntax	src	Adest	(Byte)
MVTACLO src, Adest	Rs	A0, A1	3

Description Example

```
MVTACLO  R1, A1
```

MVTC

Transfer to a control register
MoVe To Control register

MVTC

System manipulation instruction
Instruction Code
Page: 269

Syntax

```
MVTC    src, dest
```

Operation

```
dest = src;
```

Function

- This instruction transfers src to dest.
- In user mode, writing to the ISP, INTB, EXTB, BPC, BPSW, and FINTV, and the IPL[3:0], PM, U, and I bits in the PSW is ignored. In supervisor mode, writing to the PM bit in the PSW is ignored.

Flag Change

Flag	Change	Condition
C	*	
Z	*	
S	*	
O	*	

Note: * The flag changes only when dest is the PSW.

Instruction Format

	Processing	Operand		Code Size
Syntax	Size	src	dest*	(Byte)
MVTC src, dest	L	#SIMM:8	Rx	4
	L	#SIMM:16	Rx	5
	L	#SIMM:24	Rx	6
	L	#IMM:32	Rx	7
	L	Rs	Rx	3

Note: * Selectable dest: Registers ISP, USP, INTB, EXTB, PSW, BPC, BPSW, FINTV, and FPSW
Note that the PC cannot be specified as dest.

Description Example

```
MVTC    #0FFFFF000h, INTB
MVTC    R1, USP
```

MVTIPL

Interrupt priority level setting
MoVe To Interrupt Priority Level

MVTIPL

System manipulation instruction
Instruction Code
Page: 270

Syntax

```
MVTIPL  src
```

Operation

```
IPL = src;
```

Function

- This instruction transfers src to the IPL[3:0] bits in the PSW.
- This instruction is a privileged instruction. Attempting to execute this instruction in user mode generates a privileged instruction exception.
- The value of src is an unsigned integer in the range 0 ≤ src ≤ 15.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

	Operand	Code Size
Syntax	src	(Byte)
MVTIPL src	#IMM:4	3

Description Example

```
MVTIPL  #2
```

NEG

Two's complementation
NEGate

NEG

Arithmetic/logic instruction
Instruction Code
Page: 271

Syntax

```
(1) NEG     dest
(2) NEG     src, dest
```

Operation

```
(1) dest = -dest;
(2) dest = -src;
```

Function

(1) This instruction arithmetically inverts (takes the two's complement of) dest and places the result in dest.
(2) This instruction arithmetically inverts (takes the two's complement of) src and places the result in dest.

Flag Change

Flag	Change	Condition
C	√	The flag is set if dest is 0 after the operation; otherwise it is cleared.
Z	√	The flag is set if dest is 0 after the operation; otherwise it is cleared.
S	√	The flag is set if the MSB of dest after the operation is 1; otherwise it is cleared.
O	√	(1) The flag is set if dest before the operation was 80000000h; otherwise it is cleared. (2) The flag is set if src before the operation was 80000000h; otherwise it is cleared.

Instruction Format

	Processing	Operand		Code Size
Syntax	Size	src	dest	(Byte)
(1) NEG dest	L	–	Rd	2
(2) NEG src, dest	L	Rs	Rd	3

Description Example

```
NEG     R1
NEG     R1, R2
```

NOP

No operation
No OPeration

NOP

Arithmetic/logic instruction
Instruction Code
Page: 271

Syntax

```
NOP
```

Operation

```
/* No operation */
```

Function

- This instruction executes no process. The operation will be continued from the next instruction.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Code Size (Byte)
NOP	1

Description Example

```
NOP
```

NOT

Logical complementation
NOT

Arithmetic/logic instruction
Instruction Code
Page: 272

Syntax

```
(1) NOT    dest
(2) NOT    src, dest
```

Operation

```
(1) dest = ~dest;
(2) dest = ~src;
```

Function

(1) This instruction logically inverts dest and places the result in dest.
(2) This instruction logically inverts src and places the result in dest.

Flag Change

Flag	Change	Condition
C	–	
Z	√	The flag is set if dest is 0 after the operation; otherwise it is cleared.
S	√	The flag is set if the MSB of dest after the operation is 1; otherwise it is cleared.
O	–	

Instruction Format

		Processing	Operand		Code Size
Syntax		Size	src	dest	(Byte)
(1) NOT	dest	L	–	Rd	2
(2) NOT	src, dest	L	Rs	Rd	3

Description Example

```
NOT    R1
NOT    R1, R2
```

OR

Logical OR
OR

OR

Arithmetic/logic instruction
Instruction Code
Page: 273

Syntax

```
(1) OR      src, dest
(2) OR      src, src2, dest
```

Operation

```
(1) dest = dest | src;
(2) dest = src | src2;
```

Function

(1) This instruction takes the logical OR of dest and src and places the result in dest.
(2) This instruction takes the logical OR of src and src2 and places the result in dest.

Flag Change

Flag	Change	Condition
C	–	
Z	√	The flag is set if dest is 0 after the operation; otherwise it is cleared.
S	√	The flag is set if the MSB of dest after the operation is 1; otherwise it is cleared.
O	–	

Instruction Format

		Processing	Operand			Code Size
Syntax		Size	src	src2	dest	(Byte)
(1) OR	src, dest	L	#UIMM:4	–	Rd	2
		L	#SIMM:8	–	Rd	3
		L	#SIMM:16	–	Rd	4
		L	#SIMM:24	–	Rd	5
		L	#IMM:32	–	Rd	6
		L	Rs	–	Rd	2
		L	[Rs].memex	–	Rd	2 (memex == UB) 3 (memex != UB)
		L	dsp:8[Rs].memex*	–	Rd	3 (memex == UB) 4 (memex != UB)
		L	dsp:16[Rs].memex*	–	Rd	4 (memex == UB) 5 (memex != UB)
(2) OR	src, src2, dest	L	Rs	Rs2	Rd	3

Note: * For the RX Family assembler manufactured by Renesas Electronics Corp., enter a scaled value (the actual value multiplied by 2 when the size extension specifier is .W or .UW, or by 4 when the specifier is .L) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 510 (255 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 1020 (255 × 4) when the specifier is .L. With dsp:16, values from 0 to 131070 (65535 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 262140 (65535 × 4) when the specifier is .L. The value divided by 2 or 4 will be stored in the instruction code.

Description Example

```
OR      #8, R1
OR      R1, R2
OR      [R1], R2
OR      8[R1].L, R2
OR      R1, R2, R3
```

POP

Restoring data from stack to register
POP data from the stack

POP

Data transfer instruction
Instruction Code
Page: 274

Syntax

```
POP     dest
```

Operation

```
tmp = *SP;
SP = SP + 4;
dest = tmp;
```

Function

- This instruction restores data from the stack and transfers it to dest.
- The stack pointer in use is specified by the U bit in the PSW.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

	Processing	Operand	Code Size
Syntax	**Size**	**dest**	**(Byte)**
POP dest	L	Rd	2

Description Example

```
POP     R1
```

POPC

Restoring a control register
POP Control register

POPC

Data transfer instruction
Instruction Code
Page: 275

Syntax

```
POPC    dest
```

Operation

```
tmp = *SP;
SP = SP + 4;
dest = tmp;
```

Function

- This instruction restores data from the stack and transfers it to the control register specified as dest.
- The stack pointer in use is specified by the U bit in the PSW.
- In user mode, writing to the ISP, INTB, EXTB, BPC, BPSW, and FINTV, and the IPL[3:0], PM, U, and I bits in the PSW is ignored. In supervisor mode, writing to the PM bit in the PSW is ignored.

Flag Change

Flag	Change	Condition
C	*	
Z	*	
S	*	
O	*	

Note: * The flag changes only when dest is the PSW.

Instruction Format

	Processing	Operand	Code Size
Syntax	Size	dest*	(Byte)
POPC dest	L	Rx	2

Note: * Selectable dest: Registers ISP, USP, INTB, EXTB, PSW, BPC, BPSW, FINTV, and FPSW
Note that the PC cannot be specified as dest

Description Example

```
POPC    PSW
```

POPM

Restoring multiple registers from the stack
POP Multiple registers

POPM

Data transfer instruction
Instruction Code
Page: 275

Syntax

```
POPM    dest-dest2
```

Operation

```
signed char i;
for ( i = register_num(dest); i <= register_num(dest2); i++ ) {
  tmp = *SP;
  SP = SP + 4;
  register(i) = tmp;
}
```

Function

- This instruction restores values from the stack to the block of registers in the range specified by dest and dest2.
- The range is specified by first and last register numbers. Note that the condition (first register number < last register number) must be satisfied.
- R0 cannot be specified.
- The stack pointer in use is specified by the U bit in the PSW.
- Registers are restored from the stack in the following order:

R15	R14	R13	R12	....	R2	R1

Restoration is in sequence from R1.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

	Processing	Operand		Code Size
Syntax	Size	dest	dest2	(Byte)
POPM dest-dest2	L	Rd (Rd = R1 to R14)	Rd2 (Rd2 = R2 to R15)	2

Description Example

```
POPM    R1-R3
POPM    R4-R8
```

PUSH

Saving data on the stack
PUSH data onto the stack

PUSH

Data transfer instruction
Instruction Code
Page: 276

Syntax

```
PUSH.size  src
```

Operation

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

Note: * SP is decremented by 4 even when the size specifier (.size) is .B or .W. The higher-order 24 and 16 bits in the respective cases (.B and .W) are undefined.

Function

- This instruction pushes src onto the stack.
- When src is in register and the size specifier for the PUSH instruction is .B or .W, the byte or word of data from the LSB in the register are saved respectively.
- The transfer to the stack is processed in longwords. When the size specifier is .B or .W, the higher-order 24 or 16 bits are undefined respectively.
- The stack pointer in use is specified by the U bit in the PSW.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

		Processing	Operand	Code Size
Syntax	**Size**	**Size**	**src**	**(Byte)**
PUSH.size src	B/W/L	L	Rs	2
	B/W/L	L	[Rs]	2
	B/W/L	L	dsp:8[Rs]*	3
	B/W/L	L	dsp:16[Rs]*	4

Note: * For the RX Family assembler manufactured by Renesas Electronics Corp., enter a scaled value (the actual value multiplied by 2 when the size extension specifier is .W, or by 4 when the specifier is .L) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 510 (255 × 2) can be specified when the size specifier is .W, or values from 0 to 1020 (255 × 4) when the specifier is .L. With dsp:16, values from 0 to 131070 (65535 × 2) can be specified when the size specifier is .W, or values from 0 to 262140 (65535 × 4) when the specifier is .L. The value divided by 2 or 4 will be stored in the instruction code.

Description Example

```
PUSH.B   R1
PUSH.L   [R1]
```

PUSHC

Saving a control register
PUSH Control register

Data transfer instruction
Instruction Code
Page: 277

Syntax

```
PUSHC   src
```

Operation

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

Function

- This instruction pushes the control register specified by src onto the stack.
- The stack pointer in use is specified by the U bit in the PSW.
- When the PC is specified as src, this instruction pushes its own address onto the stack.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Processing Size	Operand src*	Code Size (Byte)
PUSHC src	L	Rx	2

Note: * Selectable src: Registers PC, ISP, USP, INTB, EXTB, PSW, BPC, BPSW, FINTV, and FPSW

Description Example

```
PUSHC   PSW
```

PUSHM

Saving multiple registers
PUSH Multiple registers

PUSHM

Data transfer instruction
Instruction Code
Page: 277

Syntax

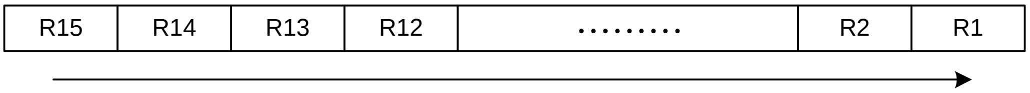
```
PUSHM   src-src2
```

Operation

```
signed char i;
for ( i = register_num(src2); i >= register_num(src); i-- ) {
  tmp = register(i);
  SP = SP - 4;
  *SP = tmp;
}
```

Function

- This instruction saves values to the stack from the block of registers in the range specified by src and src2.
- The range is specified by first and last register numbers. Note that the condition (first register number < last register number) must be satisfied.
- R0 cannot be specified.
- The stack pointer in use is specified by the U bit in the PSW.
- Registers are saved in the stack in the following order:



Saving is in sequence from R15.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Processing Size	Operand		Code Size (Byte)
		src	src2	
PUSHM src-src2	L	Rs (Rs = R1 to R14)	Rs2 (Rs2 = R2 to R15)	2

Description Example

```
PUSHM   R1-R3
PUSHM   R4-R8
```

RACL

Round the accumulator longword
Round ACcumulator Longword

RACL

DSP instruction
Instruction Code
Page: 278

Syntax

```
RACL    src, Adest
```

Operation

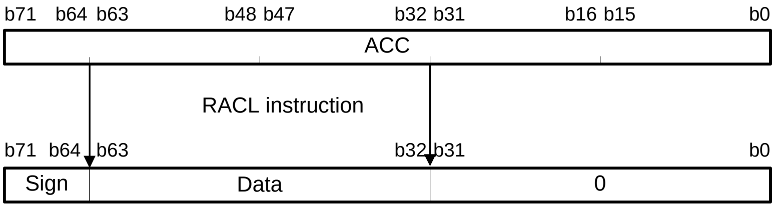
```
signed 72bit tmp;
signed 73bit tmp73;

tmp = (signed 72bit) Adest << src;
tmp73 = (signed 73bit) tmp + 0000000000080000000h;

if (tmp73 > (signed 73bit) 0007FFFFFFF00000000h)
    Adest = 007FFFFFFF00000000h;
else if (tmp73 < (signed 73bit) 1FF8000000000000000h)
    Adest = FF8000000000000000h;
else
    Adest = tmp & FFFFFFFFFF00000000h;
```

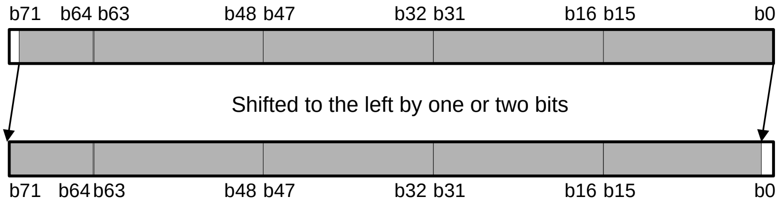
Function

- This instruction rounds the value of the accumulator into a longword and stores the result in the accumulator.

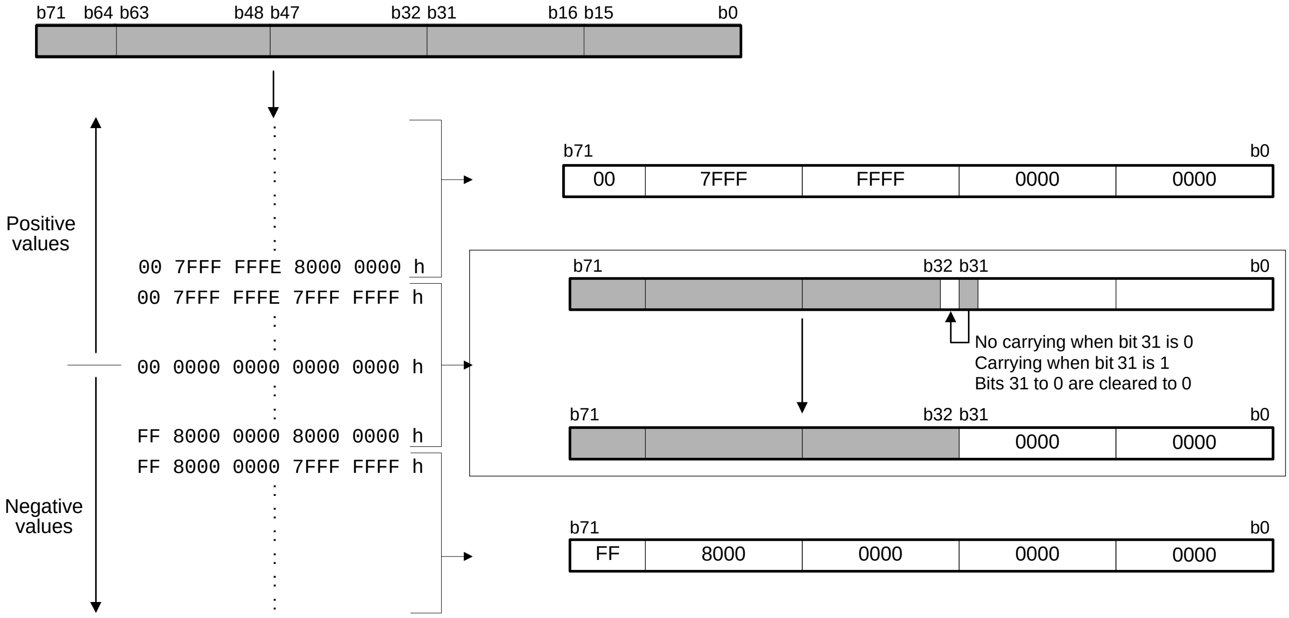


- The RACL instruction is executed according to the following procedures.

  Processing 1:
  The value of the accumulator is shifted to the left by one or two bits as specified by src.



Processing 2:
The value of the accumulator changes according to the value of 64 bits after the contents have been shifted to the left by one or two bits.



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

	Operand		Code Size
Syntax	src	Adest	(Byte)
RACL src, Adest	#IMM:1 (IMM:1 = 1, 2)	A0, A1	3

Description Example

```
RACL   #1, A1
RACL   #2, A0
```

RACW

Round the accumulator word
Round ACcumulator Word

RACW

DSP instruction
Instruction Code
Page: 278

Syntax

```
RACW    src, Adest
```

Operation

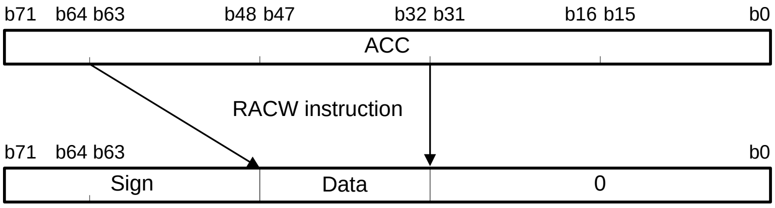
```
signed 72bit tmp;
signed 73bit tmp73;

tmp = (signed 72bit) Adest << src;
tmp73 = (signed 73bit) tmp + 0000000000080000000h;

if (tmp73 > (signed 73bit) 00000007FFF00000000h)
    Adest = 0000007FFF00000000h;
else if (tmp73 < (signed 73bit) 1FFFFFF800000000000h)
    Adest = FFFFFF800000000000h;
else
    Adest = tmp & FFFFFFFFFF00000000h;
```

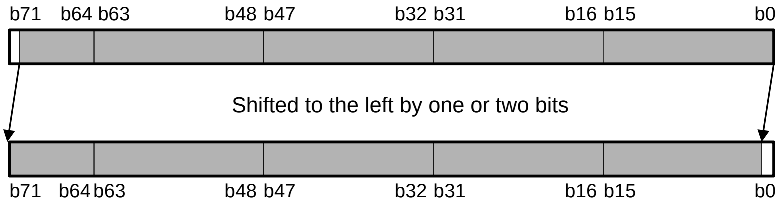
Function

- This instruction rounds the value of the accumulator into a word and stores the result in the accumulator.

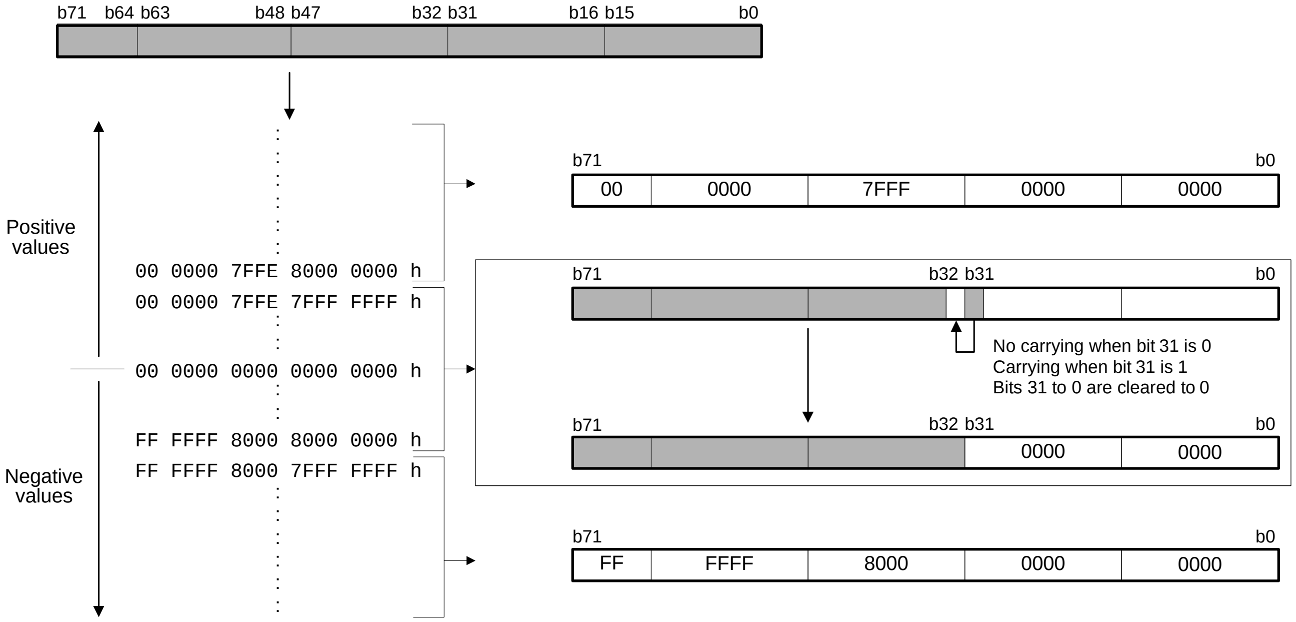


- The RACW instruction is executed according to the following procedures.

  Processing 1:
  The value of the accumulator is shifted to the left by one or two bits as specified by src.


b71 b64 b63 b48 b47 b32 b31 b16 b15 b0
Shifted to the left by one or two bits
b71 b64 b63 b48 b47 b32 b31 b16 b15 b0

Processing 2:
The value of the accumulator changes according to the value of 64 bits after the contents have been shifted to the left by one or two bits.



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

	Operand		Code Size
Syntax	**src**	**Adest**	**(Byte)**
RACW src, Adest	#IMM:1 (IMM:1 = 1, 2)	A0, A1	3

Description Example

```
RACW    #1, A1
RACW    #2, A0
```

RDACL

Round the accumulator longword
Round Down ACcumulator Longword

RDACL

DSP instruction
Instruction Code
Page: 279

Syntax

```
RDACL    src, Adest
```

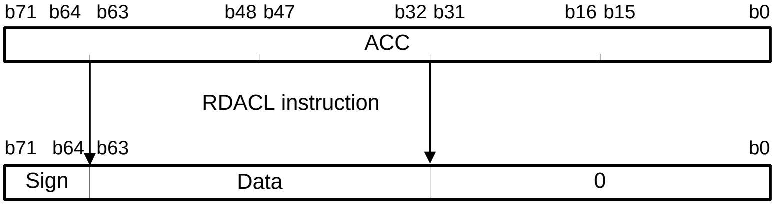
Operation

```
signed 72bit tmp;
tmp = (signed 72bit) Adest << src;

if (tmp > (signed 72bit) 007FFFFFFF00000000h)
    Adest = 007FFFFFFF00000000h;
else if (tmp < (signed 72bit) FF8000000000000000h)
    Adest = FF8000000000000000h;
else
    Adest = tmp & FFFFFFFFFF00000000h;
```

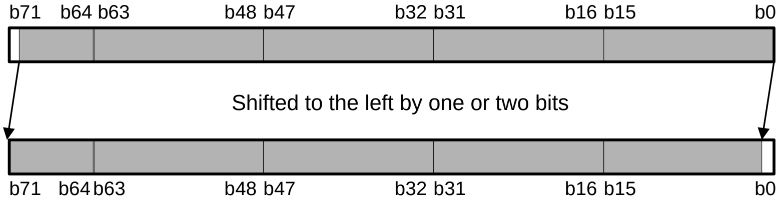
Function

- This instruction rounds the value of the accumulator into a longword and stores the result in the accumulator.

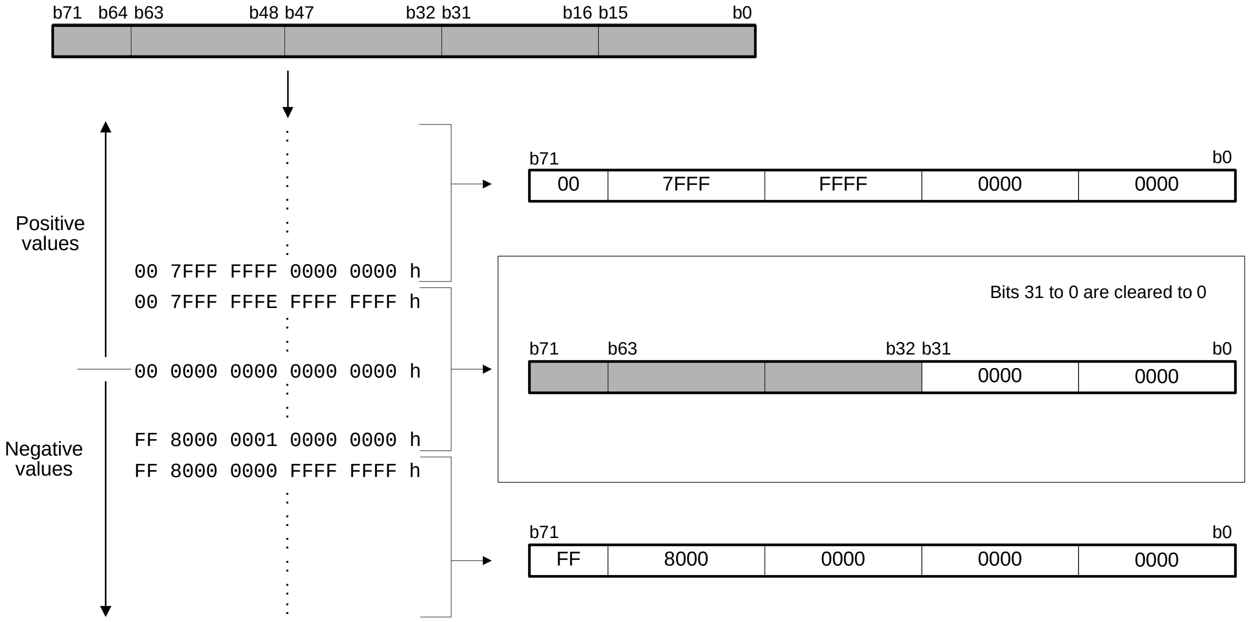


- The RDACL instruction is executed according to the following procedures.

  Processing 1:
  The value of the accumulator is shifted to the left by one or two bits as specified by src.



Processing 2:
The value of the accumulator changes according to the value of 64 bits after the contents have been shifted to the left by one or two bits.



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

	Operand		Code Size
Syntax	**src**	**Adest**	**(Byte)**
RDACL src, Adest	#IMM:1 (IMM:1 = 1, 2)	A0, A1	3

Description Example

```
RDACL    #1, A1
RDACL    #2, A0
```

RDACW

Round the accumulator word
Round Down ACcumulator Word

RDACW

DSP instruction
Instruction Code
Page: 279

Syntax

```
RDACW    src, Adest
```

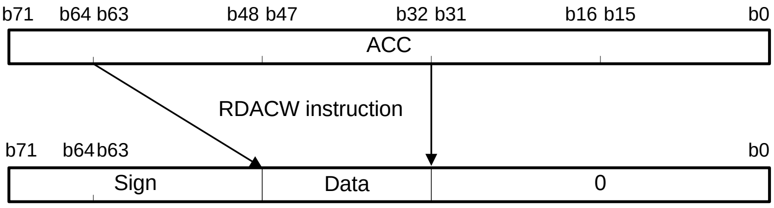
Operation

```
signed 72bit tmp;
tmp = (signed 72bit) Adest << src;

if (tmp > (signed 72bit) 0000007FFF00000000h)
    Adest = 0000007FFF00000000h;
else if (tmp < (signed 72bit) FFFFFF800000000000h)
    Adest = FFFFFF800000000000h;
else
    Adest = tmp & FFFFFFFFFF00000000h;
```

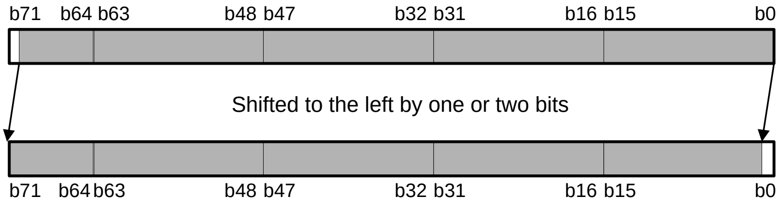
Function

- This instruction rounds the value of the accumulator into a word and stores the result in the accumulator.

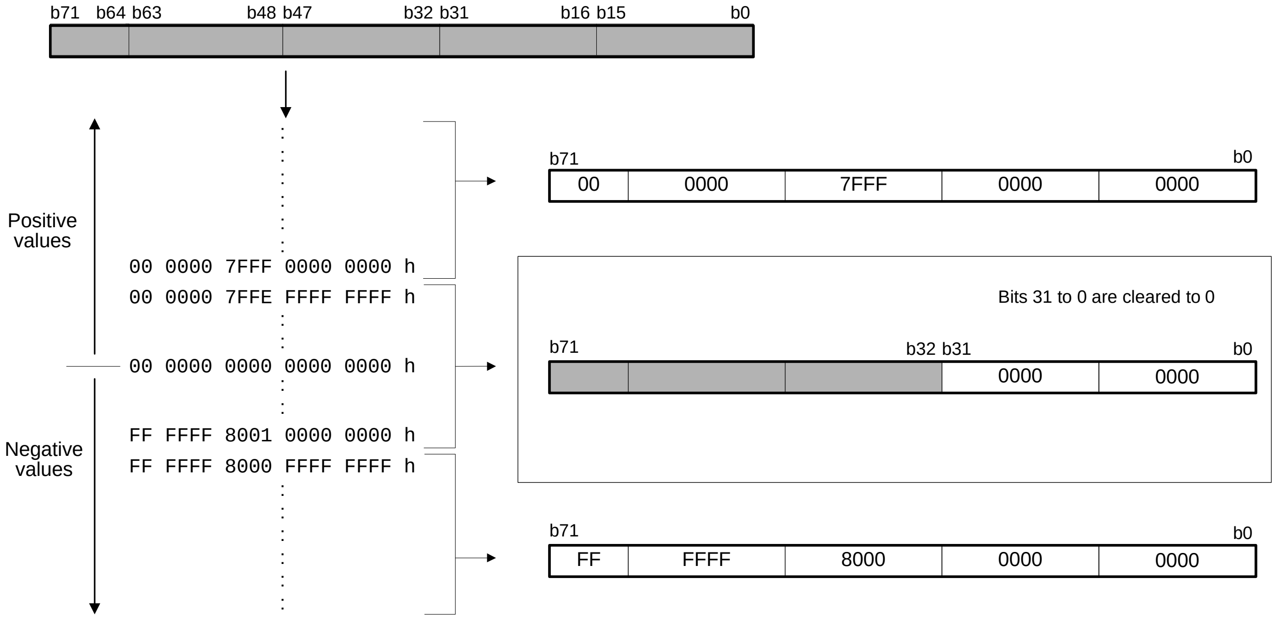


- The RDACW instruction is executed according to the following procedures.

  Processing 1:
  The value of the accumulator is shifted to the left by one or two bits as specified by src.


b71 b64 b63 b48 b47 b32 b31 b16 b15 b0
Shifted to the left by one or two bits
b71 b64 b63 b48 b47 b32 b31 b16 b15 b0

Processing 2:
The value of the accumulator changes according to the value of 64 bits after the contents have been shifted to the left by one or two bits.



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Operand src	Operand Adest	Code Size (Byte)
RDACW src, Adest	#IMM:1 (IMM:1 = 1, 2)	A0, A1	3

Description Example

```
RDACW   #1, A1
RDACW   #2, A1
```

REVL

Endian conversion
REVerse Longword data

REVL

Data transfer instruction
Instruction Code
Page: 280

Syntax

```
REVL    src, dest
```

Operation

```
Rd = { Rs[7:0], Rs[15:8], Rs[23:16], Rs[31:24] }
```

Function

- This instruction converts the endian byte order within a 32-bit datum, which is specified by src, and saves the result in dest.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

	Operand		Code Size
Syntax	**src**	**dest**	**(Byte)**
REVL src, dest	Rs	Rd	3

Description Example

```
REVL    R1, R2
```

REVW

Endian conversion
REVerse Word data

REVW

Data transfer instruction
Instruction Code
Page: 280

Syntax

```
REVW    src, dest
```

Operation

```
Rd = { Rs[23:16], Rs[31:24], Rs[7:0], Rs[15:8] }
```

Function

- This instruction converts the endian byte order within the higher- and lower-order 16-bit data, which are specified by src, and saves the result in dest.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

	Operand		Code Size
Syntax	**src**	**dest**	**(Byte)**
REVW src, dest	Rs	Rd	3

Description Example

```
REVW    R1, R2
```

RMPA

Multiply-and-accumulate operation
Repeated MultiPly and Accumulate

RMPA

Arithmetic/logic instruction
Instruction Code
Page: 281

Syntax

```
RMPA.size
```

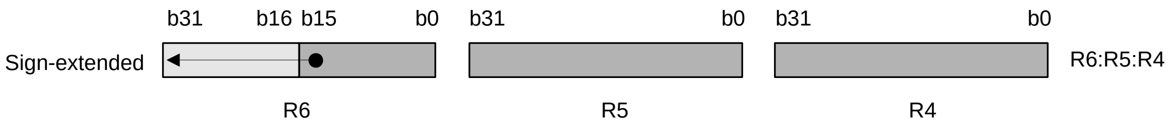
Operation

```
while ( R3 != 0 ) {
  R6:R5:R4 = R6:R5:R4 + *R1 * *R2;
  R1 = R1 + n;
  R2 = R2 + n;
  R3 = R3 - 1;
}
```

Notes: 1. If this instruction is executed with R3 set to 0, it is ignored and has no effect on registers and flags.
2. When the size specifier (.size) is .B, .W, or .L, n is 1, 2, or 4, respectively.

Function

- This instruction performs a multiply-and-accumulate operation with the multiplicand addresses specified by R1, the multiplier addresses specified by R2, and the number of multiply-and-accumulate operations specified by R3. The operands and result are handled as signed values, and the result is placed in R6:R5:R4 as an 80-bit datum. Note that the higher-order 16 bits of R6 are set to the value obtained by sign-extending the lower-order 16 bits of R6.
- The greatest value that is specifiable in R3 is 00010000h.



- The data in R1 and R2 are undefined when instruction execution is completed.
- Specify the initial value in R6:R5:R4 before executing the instruction. Furthermore, be sure to set R6 to FFFFFFFFh when R5:R4 is negative or to 00000000h if R5:R4 is positive.
- An interrupt request during execution of this instruction will be accepted, so processing of the instruction will be suspended. That is, execution of the instruction will continue on return from the interrupt processing routine. However, be sure to save the contents of the R1, R2, R3, R4, R5, R6, and PSW when an interrupt is generated and restore them when execution is returned from the interrupt routine.
- In execution of the instruction, the data may be prefetched from the multiplicand addresses specified by R1 and the multiplier addresses specified by R2, with R3 as the upper limit. For details of the data size to be prefetched, refer to the hardware manual of each product.

Note: The accumulator (ACC0) is used to perform the function. The value of ACC0 after executing the instruction is undefined.

Flag Change

Flag	Change	Condition
C	–	
Z	–	
S	√	The flag is set if the MSB of R6 is 1; otherwise it is cleared.
O	√	The flag is set if the R6:R5:R4 data is greater than 2^{63} –1 or smaller than $–2^{63}$; otherwise it is cleared.

Instruction Format

Syntax	Size	Processing Size	Code Size (Byte)
RMPA.size	B/W/L	size	2

Description Example

```
RMPA.W
```

ROLC

Rotation with carry to left
ROtate Left with Carry

ROLC

Arithmetic/logic instruction
Instruction Code
Page: 281

Syntax

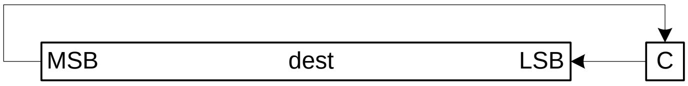
```
ROLC    dest
```

Operation

```
dest <<= 1;
if ( C == 0 ) { dest &= FFFFFFFEh; }
else { dest |= 00000001h; }
```

Function

- This instruction treats dest and the C flag as a unit, rotating the whole one bit to the left.



Flag Change

Flag	Change	Condition
C	√	The flag is set if the shifted-out bit is 1; otherwise it is cleared.
Z	√	The flag is set if dest is 0 after the operation; otherwise it is cleared.
S	√	The flag is set if the MSB of dest after the operation is 1; otherwise it is cleared.
O	–	

Instruction Format

Syntax	Processing Size	Operand dest	Code Size (Byte)
ROLC dest	L	Rd	2

Description Example

```
ROLC    R1
```

RORC

Rotation with carry to right
ROtate Right with Carry

RORC

Arithmetic/logic instruction
Instruction Code
Page: 282

Syntax

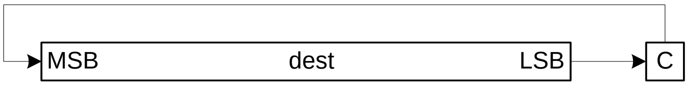
```
RORC    dest
```

Operation

```
dest >>= 1;
if ( C == 0 ) { dest &= 7FFFFFFFh; }
else { dest |= 80000000h; }
```

Function

- This instruction treats dest and the C flag as a unit, rotating the whole one bit to the right.



Flag Change

Flag	Change	Condition
C	√	The flag is set if the shifted-out bit is 1; otherwise it is cleared.
Z	√	The flag is set if dest is 0 after the operation; otherwise it is cleared.
S	√	The flag is set if the MSB of dest after the operation is 1; otherwise it is cleared.
O	–	

Instruction Format

	Processing	Operand	Code Size
Syntax	Size	dest	(Byte)
RORC dest	L	Rd	2

Description Example

```
RORC    R1
```

ROTL

Rotation to left
ROTate Left

ROTL

Arithmetic/logic instruction
Instruction Code
Page: 282

Syntax

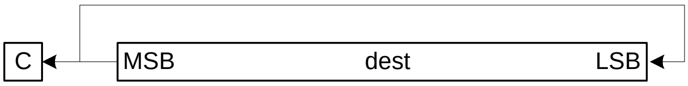
```
ROTL    src, dest
```

Operation

```
unsigned long tmp0, tmp1;
tmp0 = src & 31;
tmp1 = dest << tmp0;
dest = (( unsigned long ) dest >> ( 32 - tmp0 )) | tmp1;
```

Function

- This instruction rotates dest leftward by the number of bit positions specified by src and saves the value in dest. Bits overflowing from the MSB are transferred to the LSB and to the C flag.
- src is an unsigned integer in the range of 0 ≤ src ≤ 31.
- When src is in register, only five bits in the LSB are valid.



Flag Change

Flag	Change	Condition
C	√	After the operation, this flag will have the same LSB value as dest. In addition, when src is 0, this flag will have the same LSB value as dest.
Z	√	The flag is set if dest is 0 after the operation; otherwise it is cleared.
S	√	The flag is set if the MSB of dest after the operation is 1; otherwise it is cleared.
O	–	

Instruction Format

	Processing	Operand		Code Size
Syntax	Size	src	dest	(Byte)
ROTL src, dest	L	#IMM:5	Rd	3
	L	Rs	Rd	3

Description Example

```
ROTL    #1, R1
ROTL    R1, R2
```

ROTR

Rotation to right
ROTate Right

ROTR

Arithmetic/logic instruction
Instruction Code
Page: 283

Syntax

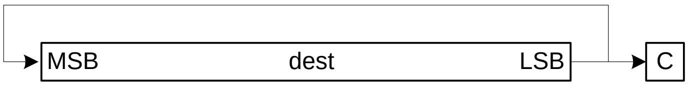
```
ROTR    src, dest
```

Operation

```
unsigned long tmp0, tmp1;
tmp0 = src & 31;
tmp1 = ( unsigned long ) dest >> tmp0;
dest = ( dest << ( 32 - tmp0 )) | tmp1;
```

Function

- This instruction rotates dest rightward by the number of bit positions specified by src and saves the value in dest. Bits overflowing from the LSB are transferred to the MSB and to the C flag.
- src is an unsigned integer in the range of $0 \leq \text{src} \leq 31$.
- When src is in register, only five bits in the LSB are valid.



Flag Change

Flag	Change	Condition
C	√	After the operation, this flag will have the same MSB value as dest. In addition, when src is 0, this flag will have the same MSB value as dest.
Z	√	The flag is set if dest is 0 after the operation; otherwise it is cleared.
S	√	The flag is set if the MSB of dest after the operation is 1; otherwise it is cleared.
O	–	

Instruction Format

	Processing	Operand		Code Size
Syntax	Size	src	dest	(Byte)
ROTR src, dest	L	#IMM:5	Rd	3
	L	Rs	Rd	3

Description Example

```
ROTR    #1, R1
ROTR    R1, R2
```

ROUND

Conversion from floating-point to integer
ROUND floating-point to integer

ROUND

Floating-point operation instruction
Instruction Code
Page: 284

Syntax

```
ROUND   src, dest
```

Operation

```
dest = ( signed long ) src;
```

Function

- This instruction converts the single-precision floating-point number stored in src into a signed longword (32-bit) integer and places the result in dest. The result is rounded according to the setting of the RM[1:0] bits in the FPSW.

Bits RM[1:0]	Rounding Mode
00b	Round to the nearest value
01b	Round towards 0
10b	Round towards +∞
11b	Round towards –∞

Flag Change

Flag	Change	Condition
C	–	
Z	√	The flag is set if the result of the operation is 0; otherwise it is cleared.
S	√	The flag is set if the sign bit (bit 31) of the result of the operation is 1; otherwise it is cleared.
O	–	
CV	√	The flag is set if an invalid operation exception is generated; otherwise it is cleared.
CO	√	The value of the flag is 0.
CZ	√	The value of the flag is 0.
CU	√	The value of the flag is 0.
CX	√	The flag is set if an inexact exception is generated; otherwise it is cleared.
CE	√	The flag is set if an unimplemented processing exception is generated; otherwise it is cleared.
FV	√	The flag is set if an invalid operation exception is generated; otherwise it does not change.
FO	–	
FZ	–	
FU	–	
FX	√	The flag is set if an inexact exception is generated; otherwise it does not change.

Note: The FX and FV flags do not change if any of the exception enable bits EX and EV is 1. The S and Z flags do not change when an exception is generated.

Instruction Format

Syntax	Processing Size	Operand		Code Size (Byte)
		src	dest	
ROUND src, dest	L	Rs	Rd	3
	L	[Rs].L	Rd	3
	L	dsp:8[Rs].L*	Rd	4
	L	dsp:16[Rs].L*	Rd	5

Note: * For the RX Family assembler manufactured by Renesas Electronics Corp., enter a scaled value (the actual value multiplied by 4) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 1020 (255 × 4) can be specified; with dsp:16, values from 0 to 262140 (65535 × 4) can be specified. The value divided by 4 will be stored in the instruction code.

Possible Exceptions

Unimplemented processing
Invalid operation
Inexact

Description Example

```
ROUND   R1, R2
ROUND   [R1], R2
```

Supplementary Description

- The following tables show the correspondences between src and dest values and the results of operations when DN = 0 and DN = 1.

When DN = 0

src Value (exponent is shown without bias)		dest	Exception
src ≥ 0	+∞	When an invalid operation exception is generated with the EV bit = 1: No change	Invalid operation exception
	127 ≥ Exponent ≥ 31	Other cases: 7FFFFFFFh	
	30 ≥ Exponent ≥ –126	00000000h to 7FFFFF80h	None*1
	+Denormalized number	No change	Unimplemented processing exception
	+0	00000000h	None
src < 0	–0		
	–Denormalized number	No change	Unimplemented processing exception
	30 ≥ Exponent ≥ –126	00000000h to 80000080h	None*1
	127 ≥ Exponent ≥ 31	When an invalid operation exception is generated with the EV bit = 1: No change	Invalid operation exception*2
	–∞	Other cases: 80000000h	
NaN	QNaN	When an invalid operation exception is generated with the EV bit = 1: No change Other cases:	Invalid operation exception
	SNaN	Sign bit = 0: 7FFFFFFFh Sign bit = 1: 80000000h	

Notes: 1. An inexact exception occurs when the result is rounded.
2. No invalid operation exception occurs when src = CF000000h.

When DN = 1

src Value (exponent is shown without bias)		dest	Exception
src ≥ 0	+∞	When an invalid operation exception is generated with the EV bit = 1: No change	Invalid operation exception
	127 ≥ Exponent ≥ 31	Other cases: 7FFFFFFFh	
	30 ≥ Exponent ≥ –126	00000000h to 7FFFFF80h	None*1
	+0, +Denormalized number	00000000h	None
src < 0	–0, –Denormalized number		
	30 ≥ Exponent ≥ –126	00000000h to 80000080h	None*1
	127 ≥ Exponent ≥ 31	When an invalid operation exception is generated with the EV bit = 1: No change	Invalid operation exception*2
	–∞	Other cases: 80000000h	
NaN	QNaN	When an invalid operation exception is generated with the EV bit = 1: No change Other cases:	Invalid operation exception
	SNaN	Sign bit = 0: 7FFFFFFFh Sign bit = 1: 80000000h	

Notes: 1. An inexact exception occurs when the result is rounded.
2. No invalid operation exception occurs when src = CF000000h.

RTE

Return from the exception
ReTurn from Exception

RTE

System manipulation instruction
Instruction Code
Page: 284

Syntax

```
RTE
```

Operation

```
PC = *SP;
SP = SP + 4;
tmp = *SP;
SP = SP + 4;
PSW = tmp;
LI = 0:
```

Function

- This instruction returns execution from the exception handling routine by restoring the PC and PSW contents that were preserved when the exception was accepted.
- This instruction is a privileged instruction. Attempting to execute this instruction in user mode generates a privileged instruction exception.
- If returning is accompanied by a transition to user mode, the U bit in the PSW becomes 1.

Flag Change

Flag	Change	Condition
C	*	
Z	*	
S	*	
O	*	

Note: * The flags become the corresponding values on the stack.

Instruction Format

Syntax	Code Size (Byte)
RTE	2

Description Example

```
RTE
```

RTFI

Return from the fast interrupt
ReTurn from Fast Interrupt

RTFI

System manipulation instruction
Instruction Code
Page: 285

Syntax

```
RTFI
```

Operation

```
PSW = BPSW;
PC = BPC;
LI = 0:
```

Function

- This instruction returns execution from the fast-interrupt handler by restoring the PC and PSW contents that were saved in the BPC and BPSW when the fast interrupt request was accepted.
- This instruction is a privileged instruction. Attempting to execute this instruction in user mode generates a privileged instruction exception.
- If returning is accompanied by a transition to user mode, the U bit in the PSW becomes 1.
- The data in the BPC and BPSW are undefined when instruction execution is completed.

Flag Change

Flag	Change	Condition
C	*	
Z	*	
S	*	
O	*	

Note: * The flags become the corresponding values from the BPSW.

Instruction Format

Syntax	Code Size (Byte)
RTFI	2

Description Example

```
RTFI
```

RTS

Returning from a subroutine
ReTurn from Subroutine

RTS

Branch instruction
Instruction Code
Page: 285

Syntax

```
RTS
```

Operation

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

Function

- This instruction returns the flow of execution from a subroutine.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Code Size (Byte)
RTS	1

Description Example

```
RTS
```

RTSD

Releasing stack frame and returning from subroutine
ReTurn from Subroutine and Deallocate stack frame

RTSD

Branch instruction
Instruction Code
Page: 285

Syntax

```
(1) RTSD    src
(2) RTSD    src, dest-dest2
```

Operation

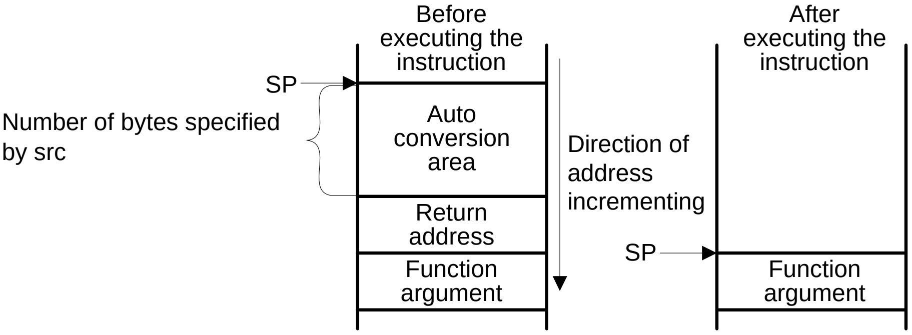
```
(1) SP = SP + src;
    PC = *SP;
    SP = SP + 4;

(2) signed char i;
    SP = SP + ( src - ( register_num(dest2) - register_num(dest) +1 ) * 4 );
    for ( i = register_num(dest); i <= register_num(dest2); i++ ) {
      tmp = *SP;
      SP = SP + 4;
      register(i) = tmp;
    }
    PC = *SP;
    SP = SP + 4;
```

Function

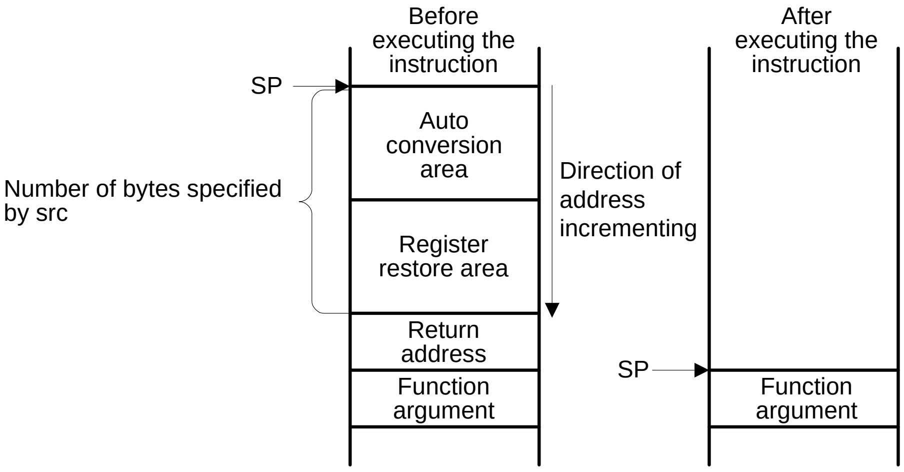
(1) This instruction returns the flow of execution from a subroutine after deallocating the stack frame for the subroutine.

- Specify src to be the size of the stack frame (auto conversion area).


Before executing the instruction
SP
Number of bytes specified by src
Auto conversion area
Return address
Function argument
Direction of address incrementing
After executing the instruction
SP
Function argument

(2) This instruction returns the flow of execution from a subroutine after deallocating the stack frame for the subroutine and also restoring register values from the stack area.

- Specify src to be the total size of the stack frame (auto conversion area and register restore area).


Before executing the instruction
After executing the instruction
SP
Number of bytes specified by src
Auto conversion area
Register restore area
Return address
Function argument
Direction of address incrementing
SP
Function argument

- This instruction restores values for the block of registers in the range specified by dest and dest2 from the stack.
- The range is specified by first and last register numbers. Note that the condition (first register number ≤ last register number) must be satisfied.
- R0 cannot be specified.
- The stack pointer in use is specified by the U bit in the PSW.
- Registers are restored from the stack in the following order:

R15	R14	R13	R12	··········	R2	R1

Restoration is in sequence from R1.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax		Operand			Code Size (Byte)
		src	dest	dest2	
(1) RTSD	src	#UIMM:8*	–	–	2
(2) RTSD	src, dest-dest2	#UIMM:8*	Rd (Rd=R1 to R15)	Rd2 (Rd2=R1 to R15)	3

Note: * For the RX Family assembler manufactured by Renesas Electronics Corp., enter a scaled value (the actual value multiplied by 4) as the immediate value. With UIMM:8, values from 0 to 1020 (255 × 4) can be specified. The value divided by 4 will be stored in the instruction code.

Description Example

```
RTSD    #4
RTSD    #16, R5-R7
```

SAT

Saturation of signed 32-bit data
SATurate signed 32-bit data

SAT

Arithmetic/logic instruction
Instruction Code
Page: 286

Syntax

```
SAT     dest
```

Operation

```
if ( O == 1 && S == 1 )
   dest = 7FFFFFFFh;
else if ( O == 1 && S == 0 )
   dest = 80000000h;
```

Function

- This instruction performs a 32-bit signed saturation operation.
- When the O flag is 1 and the S flag is 1, the result of the operation is 7FFFFFFFh and it is placed in dest.
  When the O flag is 1 and the S flag is 0, the result of the operation is 80000000h and it is placed in dest. In other cases, the dest value does not change.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

	Processing	Operand	Code Size
Syntax	Size	dest	(Byte)
SAT dest	L	Rd	2

Description Example

```
SAT     R1
```

SATR

Saturation of signed 64-bit data for RMPA
SATuRate signed 64-bit data for RMPA

SATR

Arithmetic/logic instruction
Instruction Code
Page: 286

Syntax

```
SATR
```

Operation

```
if ( O == 1 && S == 0 )
   R6:R5:R4 = 000000007FFFFFFFFFFFFFFFh;
else if ( O == 1 && S == 1 )
   R6:R5:R4 = FFFFFFFF8000000000000000h;
```

Function

- This instruction performs a 64-bit signed saturation operation.
- When the O flag is 1 and the S flag is 0, the result of the operation is 000000007FFFFFFFFFFFFFFFh and it is placed in R6:R5:R4. When the O flag is 1 and the S flag is 1, the result of the operation is FFFFFFFF8000000000000000h and it is place in R6:R5:R4. In other cases, the R6:R5:R4 value does not change.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Code Size (Byte)
SATR	2

Description Example

```
SATR
```

SBB

Subtraction with borrow
SuBtract with Borrow

SBB

Arithmetic/logic instruction
Instruction Code
Page: 287

Syntax

```
SBB      src, dest
```

Operation

```
dest = dest - src - !C;
```

Function

- This instruction subtracts src and the inverse of the C flag (borrow) from dest and places the result in dest.

Flag Change

Flag	Change	Condition
C	√	The flag is set if an unsigned operation produces no overflow; otherwise it is cleared.
Z	√	The flag is set if dest is 0 after the operation; otherwise it is cleared.
S	√	The flag is set if the MSB of dest after the operation is 1; otherwise it is cleared.
O	√	The flag is set if a signed operation produces an overflow; otherwise it is cleared.

Instruction Format

	Processing	Operand		Code Size
Syntax	**Size**	**src**	**dest**	**(Byte)**
SBB src, dest	L	Rs	Rd	3
	L	[Rs].L	Rd	4
	L	dsp:8[Rs].L*	Rd	5
	L	dsp:16[Rs].L*	Rd	6

Note: * For the RX Family assembler manufactured by Renesas Electronics Corp., enter a scaled value (the actual value multiplied by 4) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 1020 (255 × 4) can be specified; with dsp:16, values from 0 to 262140 (65535 × 4) can be specified. The value divided by 4 will be stored in the instruction code.

Description Example

```
SBB      R1, R2
SBB      [R1], R2
```

SC*Cnd*

Condition setting
Store Condition Conditionally

SC*Cnd*

Data transfer instruction
Instruction Code
Page: 288

Syntax

```
SCCnd.size   dest
```

Operation

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

Function

- This instruction moves the truth-value of the condition specified by *Cnd* to dest; that is, 1 or 0 is stored to dest if the condition is true or false, respectively.
- The following table lists the types of SC*Cnd*.

SC*Cnd*	Condition	Expression	SC*Cnd*	Condition	Expression
SCGEU, C == 1 SCC	Equal to or greater than/ C flag is 1	≤	SCLTU, C == 0 SCNC	Less than/ C flag is 0	>
SCEQ, Z == 1 SCZ	Equal to/ Z flag is 1	=	SCNE, Z == 0 SCNZ	Not equal to/ Z flag is 0	≠
SCGTU (C & ~Z) == 1	Greater than	<	SCLEU (C & ~Z) == 0	Equal to or less than	≥
SCPZ S == 0	Positive or zero	0 ≤	SCN S == 1	Negative	0 >
SCGE (S ^ O) == 0	Equal to or greater than as signed integer	≤	SCLE ((S ^ O) | Z) == 1	Equal to or less than as signed integer	≥
SCGT ((S ^ O) | Z) == 0	Greater than as signed integer	<	SCLT (S ^ O) == 1	Less than as signed integer	>
SCO O == 1	O flag is 1		SCNO O == 0	O flag is 0	

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

		Processing	Operand	Code Size
Syntax	**Size**	**Size**	**dest**	**(Byte)**
SC*Cnd*.size dest	L	L	Rd	3
	B/W/L	size	[Rd]	3
	B/W/L	size	dsp:8[Rd]*	4
	B/W/L	size	dsp:16[Rd]*	5

Note: * For the RX Family assembler manufactured by Renesas Electronics Corp., enter a scaled value (the actual value multiplied by 2 when the size extension specifier is .W, or by 4 when the specifier is .L) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 510 (255 × 2) can be specified when the size specifier is .W, or values from 0 to 1020 (255 × 4) when the specifier is .L. With dsp:16, values from 0 to 131070 (65535 × 2) can be specified when the size specifier is .W, or values from 0 to 262140 (65535 × 4) when the specifier is .L. The value divided by 2 or 4 will be stored in the instruction code.

Description Example

```
SCC.L   R2
SCNE.W  [R2]
```

SCMPU

String comparison
String CoMPare Until not equal

SCMPU

String manipulation instruction
Instruction Code
Page: 288

Syntax

```
SCMPU
```

Operation

```
unsigned char *R2, *R1, tmp0, tmp1;
unsigned long R3;
while ( R3 != 0 ) {
    tmp0 = *R1++;
    tmp1 = *R2++;
    R3--;
    if ( tmp0 != tmp1 || tmp0 == '\0' ) {
        break;
    }
}
```

Note: If this instruction is executed with R3 set to 0, it is ignored and has no effect on registers and flags.

Function

- This instruction compares strings in successively higher addresses specified by R1, which indicates the source address for comparison, and R2, which indicates the destination address for comparison, until the values do not match or the null character "\0" (= 00h) is detected, with the number of bytes specified by R3 as the upper limit.
- In execution of the instruction, the data may be prefetched from the source address for comparison specified by R1 and the destination address for comparison specified by R2, with R3 as the upper limit. For details of the data size to be prefetched, refer to the hardware manual of each product.
- The contents of R1 and R2 are undefined upon completion of the instruction.
- An interrupt request during execution of this instruction will be accepted, so processing of the instruction will be suspended. That is, execution of the instruction will continue on return from the interrupt processing routine. However, be sure to save the contents of the R1, R2, R3, and PSW when an interrupt is generated and restore them when execution is returned from the interrupt routine.

Flag Change

Flag	Change	Condition
C	√	This flag is set if the operation of (*R1 – *R2) as unsigned integers produces a value greater than or equal to 0; otherwise it is cleared.
Z	√	This flag is set if the two strings have matched; otherwise it is cleared.
S	–	
O	–	

Instruction Format

Syntax	Processing Size	Code Size (Byte)
SCMPU	B	2

Description Example

```
SCMPU
```

SETPSW

Setting a flag or bit in the PSW
SET flag of PSW

SETPSW

System manipulation instruction
Instruction Code
Page: 289

Syntax

```
SETPSW  dest
```

Operation

```
dest = 1;
```

Function

- This instruction clears the O, S, Z, or C flag, which is specified by dest, or the U or I bit.
- In user mode, writing to the U or I bit in the PSW will be ignored. In supervisor mode, all flags and bits can be written to.

Flag Change

Flag	Change	Condition
C	*	
Z	*	
S	*	
O	*	

Note: * The specified flag is set to 1.

Instruction Format

	Operand	Code Size
Syntax	dest	(Byte)
SETPSW dest	flag	2

Description Example

```
SETPSW  C
SETPSW  Z
```

SHAR

Arithmetic shift to the right
SHift Arithmetic Right

SHAR

Arithmetic/logic instruction
Instruction Code
Page: 290

Syntax

```
(1) SHAR    src, dest
(2) SHAR    src, src2, dest
```

Operation

```
(1) dest = ( signed long ) dest >> ( src & 31 );
(2) dest = ( signed long ) src2 >> ( src & 31 );
```

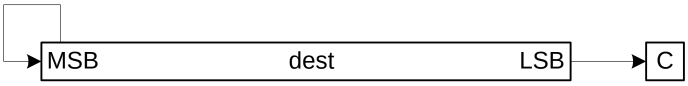
Function

(1) This instruction arithmetically shifts dest to the right by the number of bit positions specified by src and saves the value in dest.

- Bits overflowing from the LSB are transferred to the C flag.
- src is an unsigned in the range of 0 ≤ src ≤ 31.
- When src is in register, only five bits in the LSB are valid.

(2) After this instruction transfers src2 to dest, it arithmetically shifts dest to the right by the number of bit positions specified by src and saves the value in dest.

- Bits overflowing from the LSB are transferred to the C flag.
- src is an unsigned integer in the range of 0 ≤ src ≤ 31.



Flag Change

Flag	Change	Condition
C	√	The flag is set if the shifted-out bit is 1; otherwise it is cleared. However, when src is 0, this flag is also cleared.
Z	√	The flag is set if dest is 0 after the operation; otherwise it is cleared.
S	√	The flag is set if the MSB of dest after the operation is 1; otherwise it is cleared.
O	√	The flag is cleared to 0.

Instruction Format

Syntax	Processing	Operand			Code Size
	Size	src	src2	dest	(Byte)
(1) SHAR src, dest	L	#IMM:5	–	Rd	2
	L	Rs	–	Rd	3
(2) SHAR src, src2, dest	L	#IMM:5	Rs	Rd	3

Description Example

```
SHAR    #3, R2
SHAR    R1, R2
SHAR    #3, R1, R2
```

SHLL

Logical and arithmetic shift to the left
SHift Logical and arithmetic Left

SHLL

Arithmetic/logic instruction
Instruction Code
Page: 291

Syntax

```
(1) SHLL    src, dest
(2) SHLL    src, src2, dest
```

Operation

```
(1) dest = dest << ( src & 31 );
(2) dest = src2 << ( src & 31 );
```

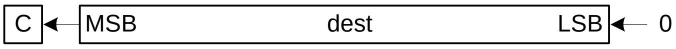
Function

(1) This instruction arithmetically shifts dest to the left by the number of bit positions specified by src and saves the value in dest.

- Bits overflowing from the MSB are transferred to the C flag.
- When src is in register, only five bits in the LSB are valid.
- src is an unsigned integer in the range of 0 ≤ src ≤ 31.

(2) After this instruction transfers src2 to dest, it arithmetically shifts dest to the left by the number of bit positions specified by src and saves the value in dest.

- Bits overflowing from the MSB are transferred to the C flag.
- src is an unsigned integer in the range of 0 ≤ src ≤ 31.



Flag Change

Flag	Change	Condition
C	√	The flag is set if the shifted-out bit is 1; otherwise it is cleared. However, when src is 0, this flag is also cleared.
Z	√	The flag is set if dest is 0 after the operation; otherwise it is cleared.
S	√	The flag is set if the MSB of dest after the operation is 1; otherwise it is cleared.
O	√	This bit is cleared to 0 when the MSB of the result of the operation is equal to all bit values that have been shifted out (i.e. the shift operation has not changed the sign); otherwise it is set to 1. However, when scr is 0, this flag is also cleared.

Instruction Format

	Processing	Operand			Code Size
Syntax	**Size**	**src**	**src2**	**dest**	**(Byte)**
(1) SHLL src, dest	L	#IMM:5	–	Rd	2
	L	Rs	–	Rd	3
(2) SHLL src, src2, dest	L	#IMM:5	Rs	Rd	3

Description Example

```
SHLL    #3, R2
SHLL    R1, R2
SHLL    #3, R1, R2
```

SHLR

Logical shift to the right
SHift Logical Right

SHLR

Arithmetic/logic instruction
Instruction Code
Page: 292

Syntax

```
(1) SHLR    src, dest
(2) SHLR    src, src2, dest
```

Operation

```
(1) dest = ( unsigned long ) dest >> ( src & 31 );
(2) dest = ( unsigned long ) src2 >> ( src & 31 );
```

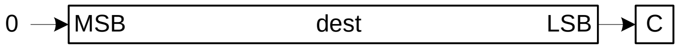
Function

(1) This instruction logically shifts dest to the right by the number of bit positions specified by src and saves the value in dest.

- Bits overflowing from the LSB are transferred to the C flag.
- src is an unsigned integer in the range of 0 ≤ src ≤ 31.
- When src is in register, only five bits in the LSB are valid.

(2) After this instruction transfers src2 to dest, it logically shifts dest to the right by the number of bit positions specified by src and saves the value in dest.

- Bits overflowing from the LSB are transferred to the C flag.
- src is an unsigned integer in the range of 0 ≤ src ≤ 31.



Flag Change

Flag	Change	Condition
C	√	The flag is set if the shifted-out bit is 1; otherwise it is cleared. However, when src is 0, this flag is also cleared.
Z	√	The flag is set if dest is 0 after the operation; otherwise it is cleared.
S	√	The flag is set if the MSB of dest after the operation is 1; otherwise it is cleared.
O	–	

Instruction Format

	Processing	Operand			Code Size
Syntax	Size	src	src2	dest	(Byte)
(1) SHLR src, dest	L	#IMM:5	–	Rd	2
	L	Rs	–	Rd	3
(2) SHLR src, src2, dest	L	#IMM:5	Rs	Rd	3

Description Example

```
SHLR    #3, R2
SHLR    R1, R2
SHLR    #3, R1, R2
```

SMOVB

Transferring a string backward
Strings MOVe Backward

SMOVB

String manipulation instruction
Instruction Code
Page: 293

Syntax

```
SMOVB
```

Operation

```
unsigned char *R1, *R2;
unsigned long R3;
while ( R3 != 0 ) {
  *R1-- = *R2--;
  R3 = R3 - 1;
}
```

Note: If this instruction is executed with R3 set to 0, it is ignored and has no effect on registers and flags.

Function

- This instruction transfers a string consisting of the number of bytes specified by R3 from the source address specified by R2 to the destination address specified by R1, with transfer proceeding in the direction of decreasing addresses.
- In execution of the instruction, data may be prefetched from the source address specified by R2, with R3 as the upper limit. For details of the data size to be prefetched, refer to the hardware manual of each product.
- The destination address specified by R1 should not be included in the range of data to be prefetched, which starts from the source address specified by R2.
- On completion of instruction execution, R1 and R2 indicate the next addresses in sequence from those for the last transfer.
- An interrupt request during execution of this instruction will be accepted, so processing of the instruction will be suspended. That is, execution of the instruction will continue on return from the interrupt processing routine. However, be sure to save the contents of the R1, R2, R3, and PSW when an interrupt is generated and restore them when execution is returned from the interrupt routine.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Processing Size	Code Size (Byte)
SMOVB	B	2

Description Example

```
SMOVB
```

SMOVF

Transferring a string forward
Strings MOVe Forward

SMOVF

String manipulation instruction
Instruction Code
Page: 293

Syntax

```
SMOVF
```

Operation

```
unsigned char *R1, *R2;
unsigned long R3;
while ( R3 != 0 ) {
  *R1++ = *R2++;
  R3 = R3 - 1;
}
```

Note: If this instruction is executed with R3 set to 0, it is ignored and has no effect on registers and flags.

Function

- This instruction transfers a string consisting of the number of bytes specified by R3 from the source address specified by R2 to the destination address specified by R1, with transfer proceeding in the direction of increasing addresses.
- In execution of the instruction, data may be prefetched from the source address specified by R2, with R3 as the upper limit. For details of the data size to be prefetched, refer to the hardware manual of each product.
- The destination address specified by R1 should not be included in the range of data to be prefetched, which starts from the source address specified by R2.
- On completion of instruction execution, R1 and R2 indicate the next addresses in sequence from those for the last transfer.
- An interrupt request during execution of this instruction will be accepted, so processing of the instruction will be suspended. That is, execution of the instruction will continue on return from the interrupt processing routine. However, be sure to save the contents of the R1, R2, R3, and PSW when an interrupt is generated and restore them when execution is returned from the interrupt routine.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Processing Size	Code Size (Byte)
SMOVF	B	2

Description Example

```
SMOVF
```

SMOVU

Transferring a string
Strings MOVe while Unequal to zero

SMOVU

String manipulation instruction
Instruction Code
Page: 293

Syntax

```
SMOVU
```

Operation

```
unsigned char *R1, *R2, tmp;
unsigned long R3;
while ( R3 != 0 ) {
  tmp = *R2++;
  *R1++ = tmp;
  R3--;
  if ( tmp == '\0' ) {
    break;
  }
}
```

Note: If this instruction is executed with R3 set to 0, it is ignored and has no effect on registers and flags.

Function

- This instruction transfers strings successively from the source address specified by R2 to the higher destination addresses specified by R1 until the null character "\0" (= 00h) is detected, with the number of bytes specified by R3 as the upper limit. String transfer is completed after the null character has been transferred.
- In execution of the instruction, data may be prefetched from the source address specified by R2, with R3 as the upper limit. For details of the data size to be prefetched, refer to the hardware manual of each product.
- The destination address specified by R1 should not be included in the range of data to be prefetched, which starts from the source address specified by R2.
- The contents of R1 and R2 are undefined upon completion of the instruction.
- An interrupt request during execution of this instruction will be accepted, so processing of the instruction will be suspended. That is, execution of the instruction will continue on return from the interrupt processing routine. However, be sure to save the contents of the R1, R2, R3, and PSW when an interrupt is generated and restore them when execution is returned from the interrupt routine.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Processing Size	Code Size (Byte)
SMOVU	B	2

Description Example

```
SMOVU
```

SSTR

Storing a string
String SToRe

SSTR

String manipulation instruction
Instruction Code
Page: 294

Syntax

```
SSTR.size
```

Operation

```
unsigned { char | short | long } *R1, R2;
unsigned long R3;
while ( R3 != 0 ) {
  *R1++ = R2;
  R3 = R3 - 1;
}
```

Notes:
1. If this instruction is executed with R3 set to 0, it is ignored and has no effect on registers and flags.
2. R1++: Incrementation is by the value corresponding to the size specifier (.size), i.e. by 1 for .B, 2 for .W, and 4 for .L.
3. R2: How much of the value in R2 is stored depends on the size specifier (.size): the byte from the LSB end of R2 is stored for .B, the word from the LSB end of R2 is stored for .W, and the longword in R2 is stored for .L.

Function

- This instruction stores the contents of R2 successively proceeding in the direction of increasing addresses specified by R1 up to the number specified by R3.
- On completion of instruction execution, R1 indicates the next address in sequence from that for the last transfer.
- An interrupt request during execution of this instruction will be accepted, so processing of the instruction will be suspended. That is, execution of the instruction will continue on return from the interrupt processing routine. However, be sure to save the contents of the R1, R2, R3, and PSW when an interrupt is generated and restore them when execution is returned from the interrupt routine.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Size	Processing Size	Code Size (Byte)
SSTR.size	B/W/L	size	2

Description Example

```
SSTR.W
```

STNZ

Transfer with condition
STore on Not Zero

STNZ

Data transfer instruction
Instruction Code
Page: 294

Syntax

```
STNZ    src, dest
```

Operation

```
if ( Z == 0 )
   dest = src;
```

Function

- This instruction moves src to dest when the Z flag is 0. dest does not change when the Z flag is 1.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Processing Size	Operand		Code Size (Byte)
		src	dest	
STNZ src, dest	L	#SIMM:8	Rd	4
	L	#SIMM:16	Rd	5
	L	#SIMM:24	Rd	6
	L	#IMM:32	Rd	7
	L	Rs	Rd	3

Description Example

```
STNZ    #1, R2
STNZ    R1, R2
```

STZ

Transfer with condition
STore on Zero

STZ

Data transfer instruction
Instruction Code
Page: 295

Syntax

```
STZ     src, dest
```

Operation

```
if ( Z == 1 )
   dest = src;
```

Function

- This instruction moves src to dest when the Z flag is 1. dest does not change when the Z flag is 0.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

	Processing	Operand		Code Size
Syntax	**Size**	**src**	**dest**	**(Byte)**
STZ src, dest	L	#SIMM:8	Rd	4
	L	#SIMM:16	Rd	5
	L	#SIMM:24	Rd	6
	L	#IMM:32	Rd	7
	L	Rs	Rd	3

Description Example

```
STZ     #1, R2
STZ     R1, R2
```

SUB

Subtraction without borrow
SUBtract

Arithmetic/logic instruction
Instruction Code
Page: 296

Syntax

```
(1) SUB     src, dest
(2) SUB     src, src2, dest
```

Operation

```
(1) dest = dest - src;
(2) dest = src2 - src;
```

Function

(1) This instruction subtracts src from dest and places the result in dest.
(2) This instruction subtracts src from src2 and places the result in dest.

Flag Change

Flag	Change	Condition
C	√	The flag is set if an unsigned operation produces no overflow; otherwise it is cleared.
Z	√	The flag is set if dest is 0 after the operation; otherwise it is cleared.
S	√	The flag is set if the MSB of dest after the operation is 1; otherwise it is cleared.
O	√	The flag is set if a signed operation produces an overflow; otherwise it is cleared.

Instruction Format

		Processing	Operand			Code Size
Syntax		Size	src	src2	dest	(Byte)
(1) SUB	src, dest	L	#UIMM:4	–	Rd	2
		L	Rs	–	Rd	2
		L	[Rs].memex	–	Rd	2 (memex == UB) 3 (memex != UB)
		L	dsp:8[Rs].memex*	–	Rd	3 (memex == UB) 4 (memex != UB)
		L	dsp:16[Rs].memex*	–	Rd	4 (memex == UB) 5 (memex != UB)
(2) SUB	src, src2, dest	L	Rs	Rs2	Rd	3

Note: * For the RX Family assembler manufactured by Renesas Electronics Corp., enter a scaled value (the actual value multiplied by 2 when the size extension specifier is .W or .UW, or by 4 when the specifier is .L) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 510 (255 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 1020 (255 × 4) when the specifier is .L. With dsp:16, values from 0 to 131070 (65535 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 262140 (65535 × 4) when the specifier is .L. The value divided by 2 or 4 will be stored in the instruction code.

Description Example

```
SUB     #15, R2
SUB     R1, R2
SUB     [R1], R2
SUB     1[R1].B, R2
SUB     R1, R2, R3
```

SUNTIL

Searching for a string
Search UNTIL equal string

SUNTIL

String manipulation instruction
Instruction Code
Page: 297

Syntax

```
SUNTIL.size
```

Operation

```
unsigned { char | short | long } *R1;
unsigned long R2, R3, tmp;
while ( R3 != 0 ) {
  tmp = ( unsigned long ) *R1++;
  R3--;
  if ( tmp == R2 ) {
      break;
  }
}
```

Notes: 1. If this instruction is executed with R3 set to 0, it is ignored and has no effect on registers and flags.
2. R1++: Incrementation is by the value corresponding to the size specifier (.size), i.e. by 1 for .B, 2 for .W, and 4 for .L.

Function

- This instruction searches a string for comparison from the first address specified by R1 for a match with the value specified in R2, with the search proceeding in the direction of increasing addresses and the number specified by R3 as the upper limit on the number of comparisons. When the size specifier (.size) is .B or .W, the byte or word data on the memory is compared with the value in R2 after being zero-extended to form a longword of data.
- In execution of the instruction, data may be prefetched from the destination address for comparison specified by R1, with R3 as the upper limit. For details of the data size to be prefetched, refer to the hardware manual of each product.
- Flags change according to the results of the operation "*R1 – R2".
- The value in R1 upon completion of instruction execution indicates the next address where the data matched. Unless there was a match within the limit, the value in R1 is the next address in sequence from that for the last comparison.
- The value in R3 on completion of instruction execution is the initial value minus the number of comparisons.
- An interrupt request during execution of this instruction will be accepted, so processing of the instruction will be suspended. That is, execution of the instruction will continue on return from the interrupt processing routine. However, be sure to save the contents of the R1, R2, R3, and PSW when an interrupt is generated and restore them when execution is returned from the interrupt routine.

Flag Change

Flag	Change	Condition
C	√	The flag is set if a comparison operation as unsigned integers results in any value equal to or greater than 0; otherwise it is cleared.
Z	√	The flag is set if matched data is found; otherwise it is cleared.
S	–	
O	–	

Instruction Format

Syntax	Size	Processing Size	Code Size (Byte)
SUNTIL.size	B/W/L	L	2

Description Example

```
SUNTIL.W
```

SWHILE

Searching for a string
Search WHILE unequal string

SWHILE

String manipulation instruction
Instruction Code
Page: 297

Syntax

```
SWHILE.size
```

Operation

```
unsigned { char | short | long } *R1;
unsigned long R2, R3, tmp;
while ( R3 != 0 ) {
    tmp = ( unsigned long ) *R1++;
    R3--;
    if ( tmp != R2 ) {
        break;
    }
}
```

Notes: 1. If this instruction is executed with R3 set to 0, it is ignored and has no effect on registers and flags.
2. R1++: Incrementation is by the value corresponding to the size specifier (.size), i.e. by 1 for .B, 2 for .W, and 4 for .L.

Function

- This instruction searches a string for comparison from the first address specified by R1 for an unmatch with the value specified in R2, with the search proceeding in the direction of increasing addresses and the number specified by R3 as the upper limit on the number of comparisons. When the size specifier (.size) is. B or .W, the byte or word data on the memory is compared with the value in R2 after being zero-extended to form a longword of data.
- In execution of the instruction, data may be prefetched from the destination address for comparison specified by R1, with R3 as the upper limit. For details of the data size to be prefetched, refer to the hardware manual of each product.
- Flags change according to the results of the operation "*R1 – R2".
- The value in R1 upon completion of instruction execution indicates the next addresses where the data did not match. If all the data contents match, the value in R1 is the next address in sequence from that for the last comparison.
- The value in R3 on completion of instruction execution is the initial value minus the number of comparisons.
- An interrupt request during execution of this instruction will be accepted, so processing of the instruction will be suspended. That is, execution of the instruction will continue on return from the interrupt processing routine. However, be sure to save the contents of the R1, R2, R3, and PSW when an interrupt is generated and restore them when execution is returned from the interrupt routine.

Flag Change

Flag	Change	Condition
C	√	The flag is set if a comparison operation as unsigned integers results in any value equal to or greater than 0; otherwise it is cleared.
Z	√	The flag is set if all the data contents match; otherwise it is cleared.
S	–	
O	–	

Instruction Format

Syntax	Size	Processing Size	Code Size (Byte)
SWHILE.size	B/W/L	L	2

Description Example

```
SWHILE.W
```

TST

Logical test
TeST logical

TST

Arithmetic/logic instruction
Instruction Code
Page: 298

Syntax

```
TST     src, src2
```

Operation

```
src2 & src;
```

Function

- This instruction changes the flag states in the PSW according to the result of logical AND of src2 and src.

Flag Change

Flag	Change	Condition
C	–	
Z	√	The flag is set if the result of the operation is 0; otherwise it is cleared.
S	√	The flag is set if the MSB of the result of the operation is 1; otherwise it is cleared.
O	–	

Instruction Format

	Processing	Operand		Code Size
Syntax	Size	src	src2	(Byte)
TST src, src2	L	#SIMM:8	Rs	4
	L	#SIMM:16	Rs	5
	L	#SIMM:24	Rs	6
	L	#IMM:32	Rs	7
	L	Rs	Rs2	3
	L	[Rs].memex	Rs2	3 (memex == UB) 4 (memex != UB)
	L	dsp:8[Rs].memex*	Rs2	4 (memex == UB) 5 (memex != UB)
	L	dsp:16[Rs].memex*	Rs2	5 (memex == UB) 6 (memex != UB)

Note: * For the RX Family assembler manufactured by Renesas Electronics Corp., enter a scaled value (the actual value multiplied by 2 when the size extension specifier is .W or .UW, or by 4 when the specifier is .L) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 510 (255 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 1020 (255 × 4) when the specifier is .L. With dsp:16, values from 0 to 131070 (65535 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 262140 (65535 × 4) when the specifier is .L. The value divided by 2 or 4 will be stored in the instruction code.

Description Example

```
TST     #7, R2
TST     R1, R2
TST     [R1], R2
TST     1[R1].UB, R2
```

UTOF

Integer to floating-point conversion
Integer TO Floating-point

UTOF

Floating-point operation instruction
Instruction Code
Page: 299

Syntax

```
UTOF   src, dest
```

Operation

```
dest = ( float ) (unsigned long ) src;
```

Function

- This instruction converts the signed longword (32-bit) integer stored in src into a single-precision floating-point number and places the result in dest. Rounding of the result is in accord with the setting of the RM[1:0] bits in the FPSW. 00000000h is handled as +0 regardless of the rounding mode.

Flag Change

Flag	Change	Condition
C	–	
Z	√	The flag is set if the result of the operation is 0; otherwise it is cleared.
S	√	The value of the flag is 0.
O	–	
CV	√	The value of the flag is 0.
CO	√	The value of the flag is 0.
CZ	√	The value of the flag is 0.
CU	√	The value of the flag is 0.
CX	√	The flag is set if an inexact exception is generated; otherwise it is cleared.
CE	√	The value of the flag is 0.
FV	–	
FO	–	
FZ	–	
FU	–	
FX	√	The flag is set if an inexact exception is generated, and otherwise left unchanged.

Note: The FX flag does not change if the exception enable bit EX is 1. The S and Z flags do not change when an exception is generated.

Instruction Format

Syntax	Processing Size	Operand src	 dest	Code Size (Byte)
UTOF src, dest	L	Rs	Rd	3
	L	[Rs].memex	Rd	3 (memex == UB) 4 (memex != UB)
	L	dsp:8[Rs].memex*	Rd	4 (memex == UB) 5 (memex != UB)
	L	dsp:16[Rs].memex*	Rd	5 (memex == UB) 6 (memex != UB)

Note: * For the RX Family assembler manufactured by Renesas Electronics Corp., enter a scaled value (the actual value multiplied by 2 when the size extension specifier is .W or .UW, or by 4 when the specifier is .L) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 510 (255 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 1020 (255 × 4) when the specifier is .L. With dsp:16, values from 0 to 131070 (65535 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 262140 (65535 × 4) when the specifier is .L. The value divided by 2 or 4 will be stored in the instruction code.

Possible Exceptions

Inexact

Description Example

```
UTOF    R1, R2
UTOF    [R1], R2
UTOF    16[R1].L, R2
```

WAIT

Waiting
WAIT

WAIT

System manipulation instruction
Instruction Code
Page: 300

Syntax

```
WAIT
```

Operation

Function

- This instruction stops program execution. Program execution is then restarted by acceptance of a non-maskable interrupt, interrupt, or generation of a reset.
- This instruction is a privileged instruction. Attempting to execute this instruction in user mode generates a privileged instruction exception.
- The I bit in the PSW becomes 1.
- The address of the PC saved at the generation of an interrupt is the one next to the WAIT instruction.

Note: For the power-down state when the execution of the program is stopped, refer to the hardware manual of each product.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Code Size (Byte)
WAIT	2

Description Example

```
WAIT
```

XCHG

Exchanging values
eXCHanGe

XCHG

Data transfer instruction
Instruction Code
Page: 300

Syntax

```
XCHG    src, dest
```

Operation

```
tmp = src;
src = dest;
dest = tmp;
```

Function

- This instruction exchanges the contents of src and dest as listed in the following table.

src	dest	Function
Register	Register	Exchanges the data in the source register (src) and the destination register (dest).
Memory location	Register	Exchanges the data at the memory location and the register. When the size extension specifier (.size) is .B or .UB, the byte of data in the LSB of the register is exchanged with the data at the memory location. When the size extension specifier (.size) is .W or .UW, the word of data in the LSB of the register is exchanged with the data at the memory location. When the size extension specifier is other than .L, the data at the memory location is transferred to the register after being extended with the specified type of extension to form a longword of data.

- This instruction may be used for the exclusive control. For details, refer to the hardware manual of each product.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

	Processing	Operand		Code Size
Syntax	Size	src	dest	(Byte)
XCHG src, dest	L	Rs	Rd	3
	L	[Rs].memex	Rd	3 (memex == UB) 4 (memex != UB)
	L	dsp:8[Rs].memex*	Rd	4 (memex == UB) 5 (memex != UB)
	L	dsp:16[Rs].memex*	Rd	5 (memex == UB) 6 (memex != UB)

Note: * For the RX Family assembler manufactured by Renesas Electronics Corp., enter a scaled value (the actual value multiplied by 2 when the size extension specifier is .W or .UW, or by 4 when the specifier is .L) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 510 (255 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 1020 (255 × 4) when the specifier is .L. With dsp:16, values from 0 to 131070 (65535 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 262140 (65535 × 4) when the specifier .L. The value divided by 2 or 4 will be stored in the instruction code.

Description Example

```
XCHG    R1, R2
XCHG    [R1].W, R2
```

XOR

Logical exclusive or
eXclusive OR logical

XOR

Arithmetic/logic instruction
Instruction Code
Page: 301

Syntax

```
XOR     src, dest
```

Operation

```
dest = dest ^ src;
```

Function

- This instruction exclusive-ORs dest and src and places the result in dest.

Flag Change

Flag	Change	Condition
C	–	
Z	√	The flag is set if dest is 0 after the operation; otherwise it is cleared.
S	√	The flag is set if the MSB of dest after the operation is 1; otherwise it is cleared.
O	–	

Instruction Format

Syntax	Processing Size	Operand src	Operand dest	Code Size (Byte)
XOR src, dest	L	#SIMM:8	Rd	4
	L	#SIMM:16	Rd	5
	L	#SIMM:24	Rd	6
	L	#IMM:32	Rd	7
	L	Rs	Rd	3
	L	[Rs].memex	Rd	3 (memex == UB) 4 (memex != UB)
	L	dsp:8[Rs].memex*	Rd	4 (memex == UB) 5 (memex != UB)
	L	dsp:16[Rs].memex*	Rd	5 (memex == UB) 6 (memex != UB)

Note: * For the RX Family assembler manufactured by Renesas Electronics Corp., enter a scaled value (the actual value multiplied by 2 when the size extension specifier is .W or .UW, or by 4 when the specifier is .L) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 510 (255 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 1020 (255 × 4) when the specifier is .L. With dsp:16, values from 0 to 131070 (65535 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 262140 (65535 × 4) when the specifier is .L. The value divided by 2 or 4 will be stored in the instruction code.

Description Example

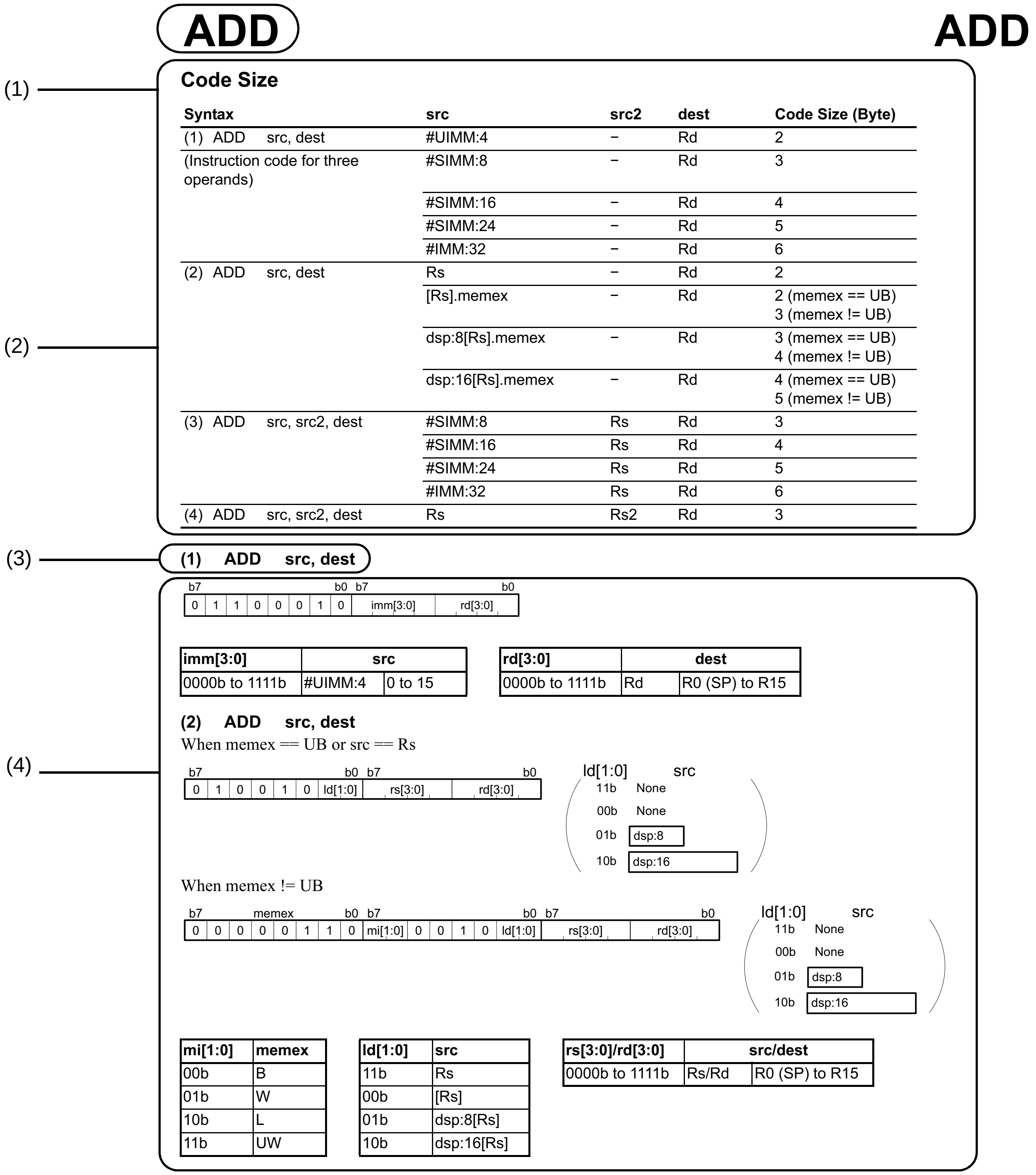
```
XOR     #8, R1
XOR     R1, R2
XOR     [R1], R2
XOR     16[R1].L, R2
```

Section 4 Instruction Code

4.1 Guide to This Section

This section describes instruction codes by showing the respective opcodes.

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



ADD

ADD

Code Size

Syntax	src	src2	dest	Code Size (Byte)
(1) ADD src, dest	#UIMM:4	–	Rd	2
(Instruction code for three operands)	#SIMM:8	–	Rd	3
	#SIMM:16	–	Rd	4
	#SIMM:24	–	Rd	5
	#IMM:32	–	Rd	6
(2) ADD src, dest	Rs	–	Rd	2
	[Rs].memex	–	Rd	2 (memex == UB) 3 (memex != UB)
	dsp:8[Rs].memex	–	Rd	3 (memex == UB) 4 (memex != UB)
	dsp:16[Rs].memex	–	Rd	4 (memex == UB) 5 (memex != UB)
(3) ADD src, src2, dest	#SIMM:8	Rs	Rd	3
	#SIMM:16	Rs	Rd	4
	#SIMM:24	Rs	Rd	5
	#IMM:32	Rs	Rd	6
(4) ADD src, src2, dest	Rs	Rs2	Rd	3

(1) ADD src, dest

b7							b0	b7	b0
0	1	1	0	0	0	1	0	imm[3:0]	rd[3:0]

imm[3:0]	src	
0000b to 1111b	#UIMM:4	0 to 15

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) ADD src, dest

When memex == UB or src == Rs

b7						b0	b7	b0
0	1	0	0	1	0	ld[1:0]	rs[3:0]	rd[3:0]

ld[1:0]	src
11b	None
00b	None
01b	dsp:8
10b	dsp:16

When memex != UB

b7			memex				b0	b7					b0	b7	b0
0	0	0	0	0	1	1	0	mi[1:0]	0	0	1	0	ld[1:0]	rs[3:0]	rd[3:0]

ld[1:0]	src
11b	None
00b	None
01b	dsp:8
10b	dsp:16

mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

(1) Mnemonic

Indicates the mnemonic name of the instruction explained on the given page.

(2) List of Code Size

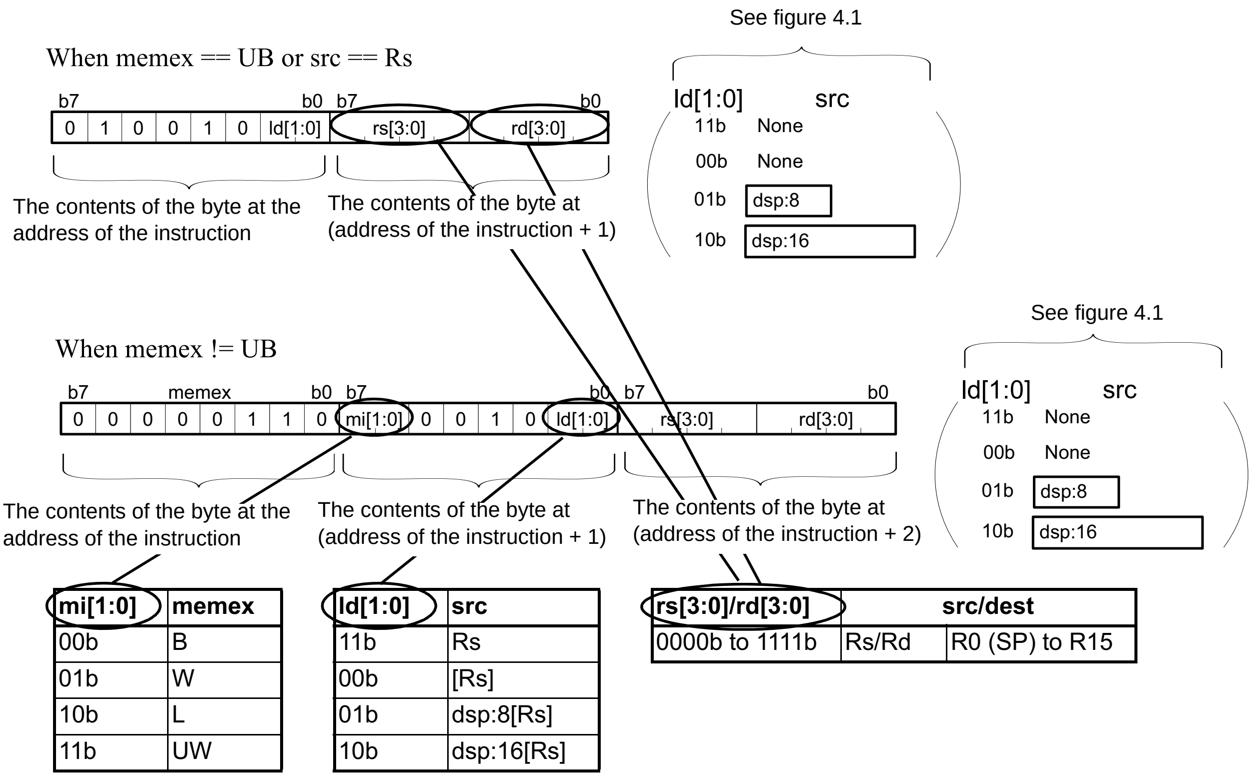
Indicates the number of bytes the instruction requires. An individual RXv2 CPU instruction takes up from one to eight bytes.

(3) Syntax

Indicates the syntax of the instruction using symbols.

(4) Instruction Code

Indicates the instruction code. The code in parentheses may be selected or omitted depending on src/dest to be selected.


When memex == UB or src == Rs
See figure 4.1
b7
b0
b7
b0
0 1 0 0 1 0 ld[1:0] rs[3:0] rd[3:0]
The contents of the byte at the address of the instruction
The contents of the byte at (address of the instruction + 1)
ld[1:0] src
11b None
00b None
01b dsp:8
10b dsp:16
When memex != UB
See figure 4.1
b7
memex
b0
b7
b0
b7
b0
0 0 0 0 0 1 1 0 mi[1:0] 0 0 1 0 ld[1:0] rs[3:0] rd[3:0]
The contents of the byte at the address of the instruction
The contents of the byte at (address of the instruction + 1)
The contents of the byte at (address of the instruction + 2)
ld[1:0] src
11b None
00b None
01b dsp:8
10b dsp:16

mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

The contents of the operand, that is the byte at (address of the instruction +2) or (following address of the instruction +3) in the previous page, are arranged as shown in figure 4.1.

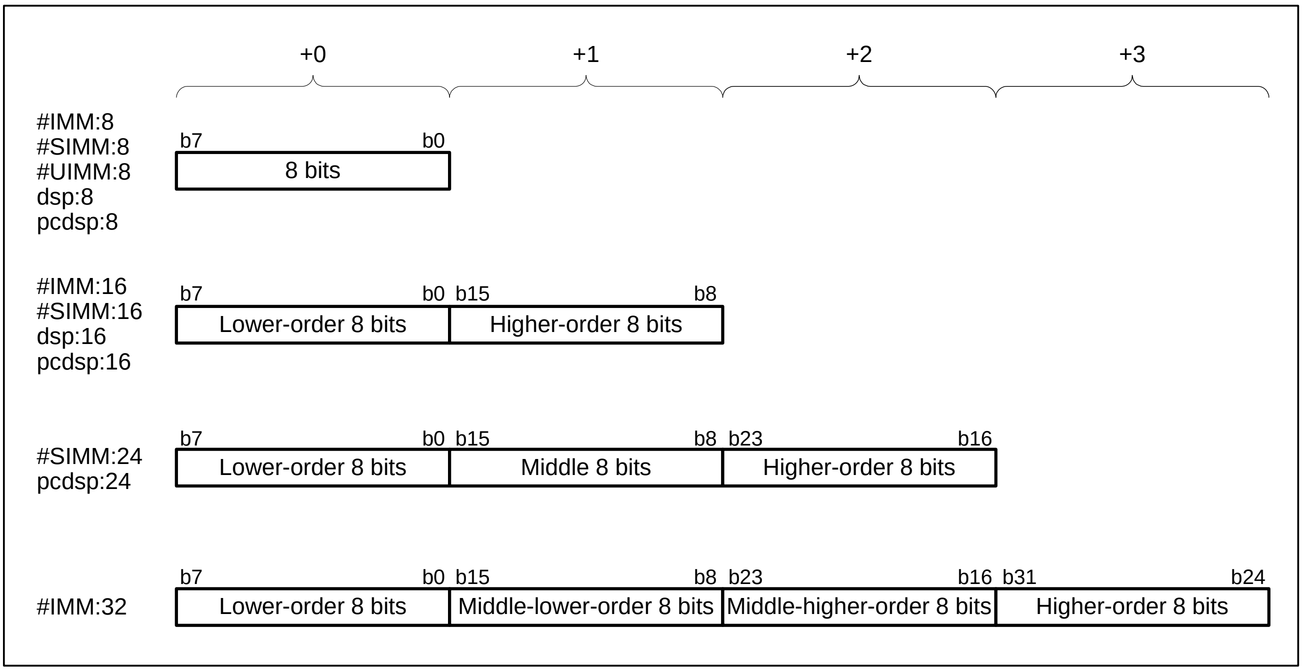


Figure 4.1 Immediate (IMM) and Displacement (dsp) Values

The abbreviations such as for rs, rd, ld, and mi represent the following.

rs:	Source register
rs2:	Second source register
rd:	Destination register
rd2:	Second destination register
ri:	Index register
rb:	Base register
li:	Length of immediate
ld:	Length of displacement
lds:	Length of source displacement
ldd:	Length of destination displacement
mi:	Memory extension size infix
imm:	Immediate
dsp:	Displacement
cd:	Condition code
cr:	Control register
cb:	Control bit
sz:	Size specifier
ad:	Addressing

4.2 Instruction Code Described in Detail

The following pages give details of the instruction codes for the RXv2 CPU.

ABS ABS

Code Size

Syntax	src	dest	Code Size (Byte)
(1) ABS dest	–	Rd	2
(2) ABS src, dest	Rs	Rd	3

(1) ABS dest

b7							b0	b7				b0
0	1	1	1	1	1	1	0	0	0	1	0	rd[3:0]

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) ABS src, dest

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	0	0	0	0	0	1	1	1	1	rs[3:0]	rd[3:0]

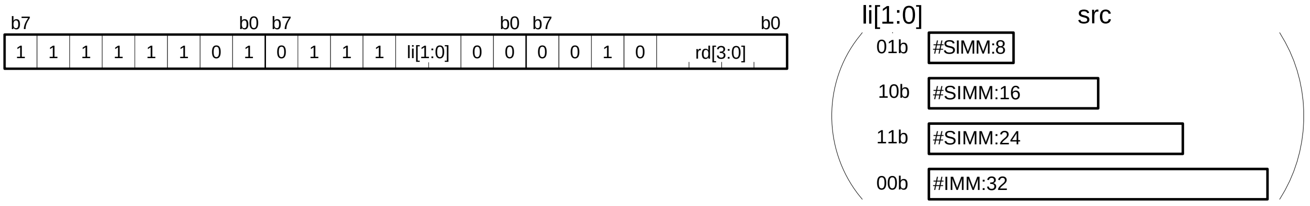
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

ADC ADC

Code Size

Syntax	src	dest	Code Size (Byte)
(1) ADC src, dest	#SIMM:8	Rd	4
	#SIMM:16	Rd	5
	#SIMM:24	Rd	6
	#IMM:32	Rd	7
(2) ADC src, dest	Rs	Rd	3
(3) ADC src, dest	[Rs].L	Rd	4
	dsp:8[Rs].L	Rd	5
	dsp:16[Rs].L	Rd	6

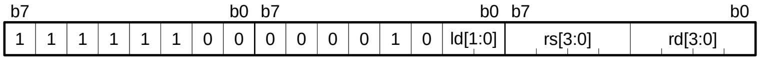
(1) ADC src, dest



li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

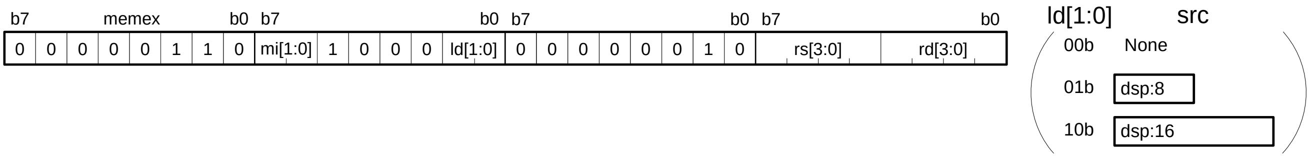
(2) ADC src, dest



ld[1:0]	src
11b	Rs

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

(3) ADC src, dest


b7
memex
b0
b7
b0
b7
b0
b7
b0
0 0 0 0 0 1 1 0 mi[1:0] 1 0 0 0 ld[1:0] 0 0 0 0 0 0 1 0 rs[3:0] rd[3:0]
ld[1:0]
src
00b
None
01b
dsp:8
10b
dsp:16

mi[1:0]	memex
10b	L

ld[1:0]	src
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

ADD ADD

Code Size

Syntax	src	src2	dest	Code Size (Byte)
(1) ADD src, dest	#UIMM:4	−	Rd	2
(Instruction code for three operands)	#SIMM:8	−	Rd	3
	#SIMM:16	−	Rd	4
	#SIMM:24	−	Rd	5
	#IMM:32	−	Rd	6
(2) ADD src, dest	Rs	−	Rd	2
	[Rs].memex	−	Rd	2 (memex == UB) 3 (memex != UB)
	dsp:8[Rs].memex	−	Rd	3 (memex == UB) 4 (memex != UB)
	dsp:16[Rs].memex	−	Rd	4 (memex == UB) 5 (memex != UB)
(3) ADD src, src2, dest	#SIMM:8	Rs	Rd	3
	#SIMM:16	Rs	Rd	4
	#SIMM:24	Rs	Rd	5
	#IMM:32	Rs	Rd	6
(4) ADD src, src2, dest	Rs	Rs2	Rd	3

(1) ADD src, dest

b7							b0	b7	b0
0	1	1	0	0	0	1	0	imm[3:0]	rd[3:0]

imm[3:0]	src	
0000b to 1111b	#UIMM:4	0 to 15

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) ADD src, dest

When memex == UB or src == Rs

b7							b0	b7	b0
0	1	0	0	1	0	ld[1:0]		rs[3:0]	rd[3:0]

ld[1:0]	src
11b	None
00b	None
01b	dsp:8
10b	dsp:16

When memex != UB

b7			memex				b0	b7					b0	b7	b0
0	0	0	0	0	1	1	0	mi[1:0]		0	0	1	0	ld[1:0]	rs[3:0]	rd[3:0]

ld[1:0]	src
11b	None
00b	None
01b	dsp:8
10b	dsp:16

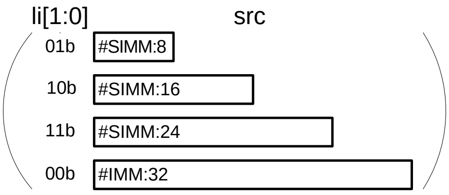
mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

(3) ADD src, src2, dest

b7							b0	b7		b0
0	1	1	1	0	0	li[1:0]		rs2[3:0]		rd[3:0]



li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

rs2[3:0]/rd[3:0]	src2/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

(4) ADD src, src2, dest

b7							b0	b7					b0	b7		b0
1	1	1	1	1	1	1	1	0	0	1	0	rd[3:0]		rs[3:0]		rs2[3:0]

rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b to 1111b	Rs/Rs2/Rd	R0 (SP) to R15

AND AND

Code Size

Syntax			src	src2	dest	Code Size (Byte)
(1)	AND	src, dest	#UIMM:4	−	Rd	2
(2)	AND	src, dest	#SIMM:8	−	Rd	3
			#SIMM:16	−	Rd	4
			#SIMM:24	−	Rd	5
			#IMM:32	−	Rd	6
(3)	AND	src, dest	Rs	−	Rd	2
			[Rs].memex	−	Rd	2 (memex == UB) 3 (memex != UB)
			dsp:8[Rs].memex	−	Rd	3 (memex == UB) 4 (memex != UB)
			dsp:16[Rs].memex	−	Rd	4 (memex == UB) 5 (memex != UB)
(4)	AND	src, src2, dest	Rs	Rs2	Rd	3

(1) AND src, dest

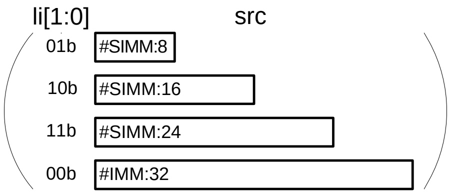
b7							b0	b7	b0
0	1	1	0	0	1	0	0	imm[3:0]	rd[3:0]

imm[3:0]	src	
0000b to 1111b	#UIMM:4	0 to 15

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) AND src, dest

b7						b0	b7				b0
0	1	1	1	0	1	li[1:0]	0	0	1	0	rd[3:0]

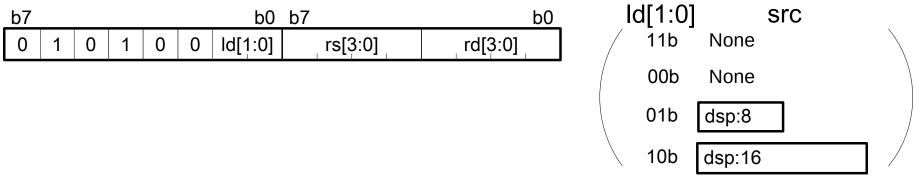


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

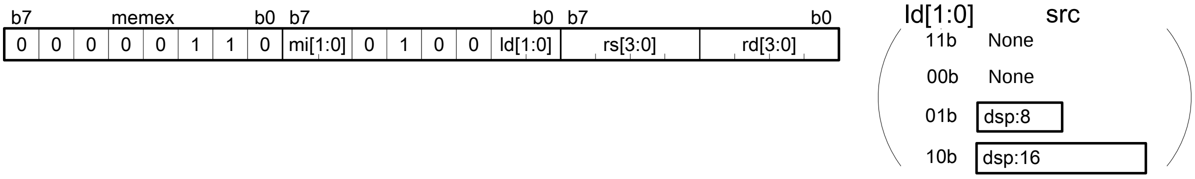
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(3) AND src, dest

When memex == UB or src == Rs



When memex != UB

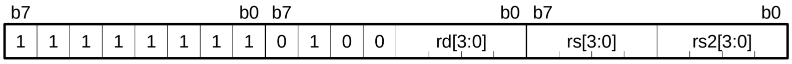


mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

(4) AND src, src2, dest



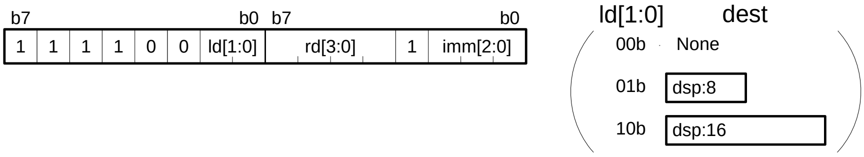
rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b to 1111b	Rs/Rs2/Rd	R0 (SP) to R15

BCLR BCLR

Code Size

Syntax	src	dest	Code Size (Byte)
(1) BCLR src, dest	#IMM:3	[Rd].B	2
	#IMM:3	dsp:8[Rd].B	3
	#IMM:3	dsp:16[Rd].B	4
(2) BCLR src, dest	Rs	[Rd].B	3
	Rs	dsp:8[Rd].B	4
	Rs	dsp:16[Rd].B	5
(3) BCLR src, dest	#IMM:5	Rd	2
(4) BCLR src, dest	Rs	Rd	3

(1) BCLR src, dest

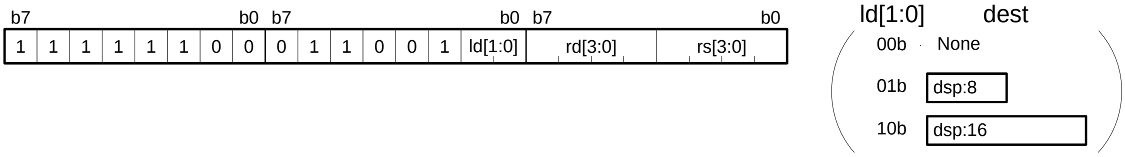


ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

imm[2:0]	src	
000b to 111b	#IMM:3	0 to 7

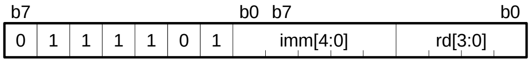
(2) BCLR src, dest



ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

(3) BCLR src, dest



imm[4:0]	src	
00000b to 11111b	#IMM:5	0 to 31

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(4) BCLR src, dest

b7							b0	b7						b0	b7	b0
1	1	1	1	1	1	0	0	0	1	1	0	0	1	ld[1:0]	rd[3:0]	rs[3:0]

ld[1:0]	dest
11b	Rd

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

BC*nd* BC*nd*

Code Size

Syntax	src	Code Size (Byte)
(1) BC*nd*.S src	pcdsp:3	1
(2) BC*nd*.B src	pcdsp:8	2
(3) BC*nd*.W src	pcdsp:16	3

(1) BC*nd*.S src

b7					b0
0	0	0	1	cd	dsp[2:0]*

Note: * dsp[2:0] specifies pcdsp:3 = src.

cd	BC*nd*
0b	BEQ, BZ
1b	BNE, BNZ

dsp[2:0]	Branch Distance
011b	3
100b	4
101b	5
110b	6
111b	7
000b	8
001b	9
010b	10

(2) BC*nd*.B src

b7				b0	src
0	0	1	0	cd[3:0]	pcdsp:8*

Note: * Address indicated by pcdsp:8 = src minus the address of the instruction

cd[3:0]	BC*nd*	cd[3:0]	BC*nd*
0000b	BEQ, BZ	1000b	BGE
0001b	BNE, BNZ	1001b	BLT
0010b	BGEU, BC	1010b	BGT
0011b	BLTU, BNC	1011b	BLE
0100b	BGTU	1100b	BO
0101b	BLEU	1101b	BNO
0110b	BPZ	1110b	BRA.B
0111b	BN	1111b	Reserved

(3) B*Cnd*.W src

b7							b0
0	0	1	1	1	0	1	cd

src
pcdsp:16*

Note: * Address indicated by pcdsp:16 = src minus the address of the instruction

cd	**B*Cnd***
0b	BEQ, BZ
1b	BNE, BNZ

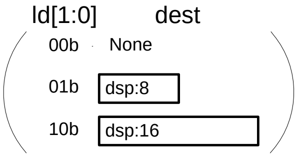
BM*Cnd* BM*Cnd*

Code Size

Syntax	src	dest	Code Size (Byte)
(1) BM*Cnd* src, dest	#IMM:3	[Rd].B	3
	#IMM:3	dsp:8[Rd].B	4
	#IMM:3	dsp:16[Rd].B	5
(2) BM*Cnd* src, dest	#IMM:5	Rd	3

(1) BM*Cnd* src, dest

b7							b0	b7					b0	b7	b0
1	1	1	1	1	1	0	0	1	1	1	imm[2:0]	ld[1:0]	rd[3:0]	cd[3:0]	



imm[2:0]	src	
000b to 111b	#IMM:3	0 to 7

ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

cd[3:0]	BM*Cnd*	cd[3:0]	BM*Cnd*
0000b	BMEQ, BMZ	1000b	BMGE
0001b	BMNE, BMNZ	1001b	BMLT
0010b	BMGEU, BMC	1010b	BMGT
0011b	BMLTU, BMNC	1011b	BMLE
0100b	BMGTU	1100b	BMO
0101b	BMLEU	1101b	BMNO
0110b	BMPZ	1110b	Reserved
0111b	BMN	1111b	Reserved

(2) BM*Cnd* src, dest

b7							b0	b7			b0	b7	b0
1	1	1	1	1	1	0	1	1	1	1	imm[4:0]	cd[3:0]	rd[3:0]

imm[4:0]	src	
00000b to 11111b	#IMM:5	0 to 31

cd[3:0]	BM*Cnd*	cd[3:0]	BM*Cnd*
0000b	BMEQ, BMZ	1000b	BMGE
0001b	BMNE, BMNZ	1001b	BMLT
0010b	BMGEU, BMC	1010b	BMGT
0011b	BMLTU, BMNC	1011b	BMLE
0100b	BMGTU	1100b	BMO
0101b	BMLEU	1101b	BMNO
0110b	BMPZ	1110b	Reserved
0111b	BMN	1111b	Reserved

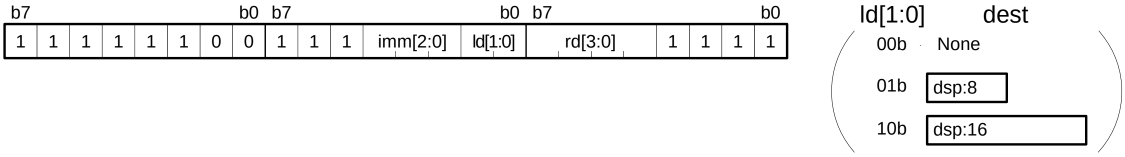
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

BNOT BNOT

Code Size

Syntax	src	dest	Code Size (Byte)
(1) BNOT src, dest	#IMM:3	[Rd].B	3
	#IMM:3	dsp:8[Rd].B	4
	#IMM:3	dsp:16[Rd].B	5
(2) BNOT src, dest	Rs	[Rd].B	3
	Rs	dsp:8[Rd].B	4
	Rs	dsp:16[Rd].B	5
(3) BNOT src, dest	#IMM:5	Rd	3
(4) BNOT src, dest	Rs	Rd	3

(1) BNOT src, dest

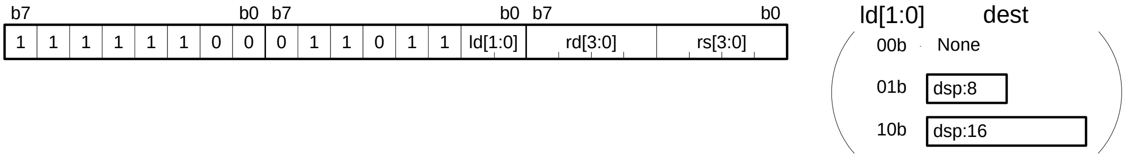


imm[2:0]	src	
000b to 111b	#IMM:3	0 to 7

ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) BNOT src, dest



ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

(3) BNOT src, dest

b7							b0	b7							b0	b7							b0
1	1	1	1	1	1	0	1	1	1	1	imm[4:0]					1	1	1	1	rd[3:0]			

imm[4:0]	src	
00000b to 11111b	#IMM:5	0 to 31

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(4) BNOT src, dest

b7							b0	b7							b0	b7							b0
1	1	1	1	1	1	0	0	0	1	1	0	1	1	ld[1:0]		rd[3:0]				rs[3:0]			

ld[1:0]	dest
11b	Rd

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

BRA BRA

Code Size

Syntax	src	Code Size (Byte)
(1) BRA.S src	pcdsp:3	1
(2) BRA.B src	pcdsp:8	2
(3) BRA.W src	pcdsp:16	3
(4) BRA.A src	pcdsp:24	4
(5) BRA.L src	Rs	2

(1) BRA.S src

b7							b0
0	0	0	0	1	dsp[2:0]*		

Note: * dsp[2:0] specifies pcdsp:3 = src.

dsp[2:0]	Branch Distance
011b	3
100b	4
101b	5
110b	6
111b	7
000b	8
001b	9
010b	10

(2) BRA.B src

b7							b0	src
0	0	1	0	1	1	1	0	pcdsp:8*

Note: * Address indicated by pcdsp:8 = src minus the address of the instruction

(3) BRA.W src

b7							b0	src
0	0	1	1	1	0	0	0	pcdsp:16*

Note: * Address indicated by pcdsp:16 = src minus the address of the instruction

(4) BRA.A src

b7							b0	src
0	0	0	0	0	1	0	0	pcdsp:24*

Note: * Address indicated by pcdsp:24 = src minus the address of the instruction

(5) BRA.L src

b7							b0	b7				b0
0	1	1	1	1	1	1	1	0	1	0	0	rs[3:0]

rs[3:0]	src	
0000b to 1111b	Rs	R0 (SP) to R15

BRK BRK

Code Size

Syntax	Code Size (Byte)
(1) BRK	1

(1) BRK

b7							b0
0	0	0	0	0	0	0	0

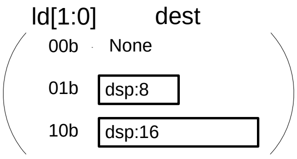
BSET BSET

Code Size

Syntax	src	dest	Code Size (Byte)
(1) BSET src, dest	#IMM:3	[Rd].B	2
	#IMM:3	dsp:8[Rd].B	3
	#IMM:3	dsp:16[Rd].B	4
(2) BSET src, dest	Rs	[Rd].B	3
	Rs	dsp:8[Rd].B	4
	Rs	dsp:16[Rd].B	5
(3) BSET src, dest	#IMM:5	Rd	2
(4) BSET src, dest	Rs	Rd	3

(1) BSET src, dest

b7						b0	b7		b0
1	1	1	1	0	0	ld[1:0]	rd[3:0]	0	imm[2:0]



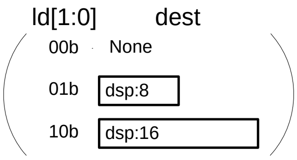
ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

imm[2:0]	src	
000b to 111b	#IMM:3	0 to 7

(2) BSET src, dest

b7							b0	b7							b0	b7							b0
1	1	1	1	1	1	0	0	0	1	1	0	0	0	ld[1:0]		rd[3:0]				rs[3:0]			



ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

(3) BSET src, dest

b7							b0	b7							b0
0	1	1	1	1	0	0	imm[4:0]					rd[3:0]			

imm[4:0]	src	
00000b to 11111b	#IMM:5	0 to 31

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(4) BSET src, dest

b7							b0	b7							b0	b7							b0
1	1	1	1	1	1	0	0	0	1	1	0	0	0	ld[1:0]		rd[3:0]				rs[3:0]			

ld[1:0]	dest
11b	Rd

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

BSR BSR

Code Size

Syntax	src	Code Size (Byte)
(1) BSR.W src	pcdsp:16	3
(2) BSR.A src	pcdsp:24	4
(3) BSR.L src	Rs	2

(1) BSR.W src

b7							b0	src
0	0	1	1	1	0	0	1	pcdsp:16*

Note: * Address indicated by pcdsp:16 = src minus the address of the instruction

(2) BSR.A src

b7							b0	src
0	0	0	0	0	1	0	1	pcdsp:24*

Note: * Address indicated by pcdsp:24 = src minus the address of the instruction

(3) BSR.L src

b7							b0	b7							b0
0	1	1	1	1	1	1	1	0	1	0	1	rs[3:0]			

rs[3:0]	src	
0000b to 1111b	Rs	R0 (SP) to R15

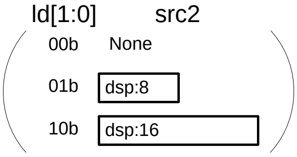
BTST BTST

Code Size

Syntax	src	src2	Code Size (Byte)
(1) BTST src, src2	#IMM:3	[Rs].B	2
	#IMM:3	dsp:8[Rs].B	3
	#IMM:3	dsp:16[Rs].B	4
(2) BTST src, src2	Rs	[Rs2].B	3
	Rs	dsp:8[Rs2].B	4
	Rs	dsp:16[Rs2].B	5
(3) BTST src, src2	#IMM:5	Rs	2
(4) BTST src, src2	Rs	Rs2	3

(1) BTST src, src2

b7						b0	b7		b0
1	1	1	1	0	1	ld[1:0]	rs[3:0]	0	imm[2:0]



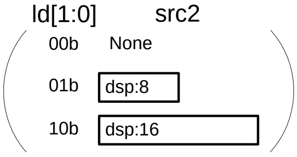
ld[1:0]	src2
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]	src2	
0000b to 1111b	Rs	R0 (SP) to R15

imm[2:0]	src	
000b to 111b	#IMM:3	0 to 7

(2) BTST src, src2

b7							b0	b7						b0	b7	b0
1	1	1	1	1	1	0	0	0	1	1	0	1	0	ld[1:0]	rs2[3:0]	rs[3:0]



ld[1:0]	src2
00b	[Rs2]
01b	dsp:8[Rs2]
10b	dsp:16[Rs2]

rs[3:0]/rs2[3:0]	src/src2	
0000b to 1111b	Rs/Rs2	R0 (SP) to R15

(3) BTST src, src2

b7							b0 b7	b0
0	1	1	1	1	1	0	imm[4:0]	rs[3:0]

imm[4:0]	src	
00000b to 11111b	#IMM:5	0 to 31

rs[3:0]	src2	
0000b to 1111b	Rs	R0 (SP) to R15

(4) BTST src, src2

b7							b0	b7							b0	b7							b0
1	1	1	1	1	1	0	0	0	1	1	0	1	0	ld[1:0]		rs2[3:0]				rs[3:0]			

ld[1:0]	src2
11b	Rs2

rs[3:0]/rs2[3:0]	src/src2	
0000b to 1111b	Rs/Rs2	R0 (SP) to R15

CLRPSW CLRPSW

Code Size

Syntax	dest	Code Size (Byte)
(1) CLRPSW dest	flag	2

(1) CLRPSW dest

b7							b0	b7				b0
0	1	1	1	1	1	1	1	1	0	1	1	cb[3:0]

cb[3:0]	dest	
0000b	flag	C
0001b		Z
0010b		S
0011b		O
0100b		Reserved
0101b		Reserved
0110b		Reserved
0111b		Reserved
1000b		I
1001b		U
1010b		Reserved
1011b		Reserved
1100b		Reserved
1101b		Reserved
1110b		Reserved
1111b		Reserved

CMP CMP

Code Size

Syntax			src	src2	Code Size (Byte)
(1)	CMP	src, src2	#UIMM:4	Rs	2
(2)	CMP	src, src2	#UIMM:8	Rs	3
(3)	CMP	src, src2	#SIMM:8	Rs	3
			#SIMM:16	Rs	4
			#SIMM:24	Rs	5
			#IMM:32	Rs	6
(4)	CMP	src, src2	Rs	Rs2	2
			[Rs].memex	Rs2	2 (memex == UB) 3 (memex != UB)
			dsp:8[Rs].memex	Rs2	3 (memex == UB) 4 (memex != UB)
			dsp:16[Rs].memex	Rs2	4 (memex == UB) 5 (memex != UB)

(1) CMP src, src2

b7							b0	b7	b0
0	1	1	0	0	0	0	1	imm[3:0]	rs2[3:0]

imm[3:0]	src	
0000b to 1111b	#UIMM:4	0 to 15

rs2[3:0]	src2	
0000b to 1111b	Rs	R0 (SP) to R15

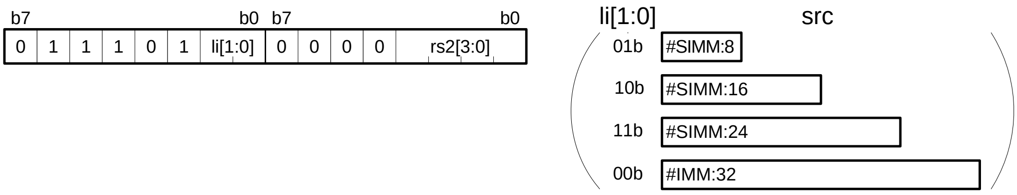
(2) CMP src, src2

b7							b0	b7				b0
0	1	1	1	0	1	0	1	0	1	0	1	rs2[3:0]

src
#UIMM:8

rs2[3:0]	src2	
0000b to 1111b	Rs	R0 (SP) to R15

(3) CMP src, src2

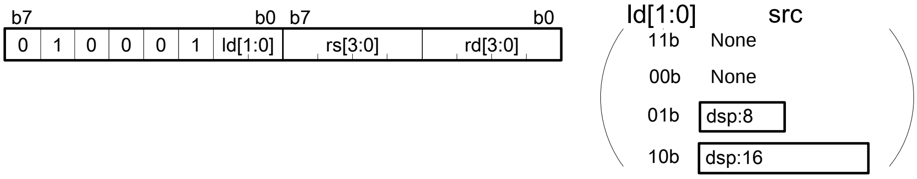


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

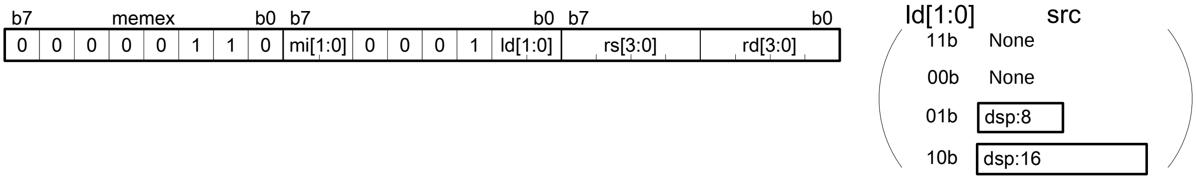
rs2[3:0]	src2	
0000b to 1111b	Rs	R0 (SP) to R15

(4) CMP src, src2

When memex == UB or src == Rs



When memex != UB



mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

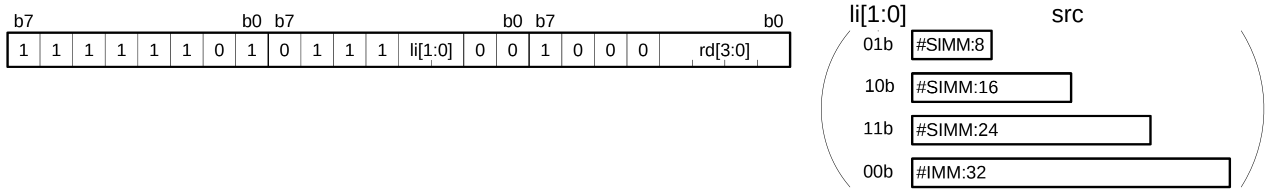
rs[3:0]/rs2[3:0]	src/src2	
0000b to 1111b	Rs/Rs2	R0 (SP) to R15

DIV DIV

Code Size

Syntax	src	dest	Code Size (Byte)
(1) DIV src, dest	#SIMM:8	Rd	4
	#SIMM:16	Rd	5
	#SIMM:24	Rd	6
	#IMM:32	Rd	7
(2) DIV src, dest	Rs	Rd	3
	[Rs].memex	Rd	3 (memex == UB) 4 (memex != UB)
	dsp:8[Rs].memex	Rd	4 (memex == UB) 5 (memex != UB)
	dsp:16[Rs].memex	Rd	5 (memex == UB) 6 (memex != UB)

(1) DIV src, dest

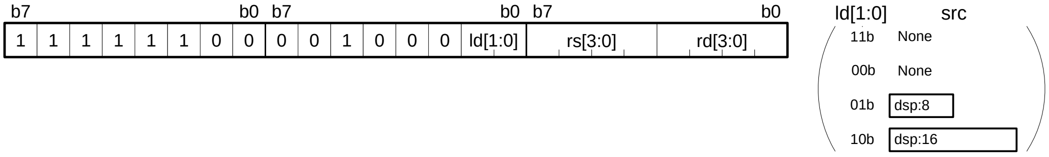


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

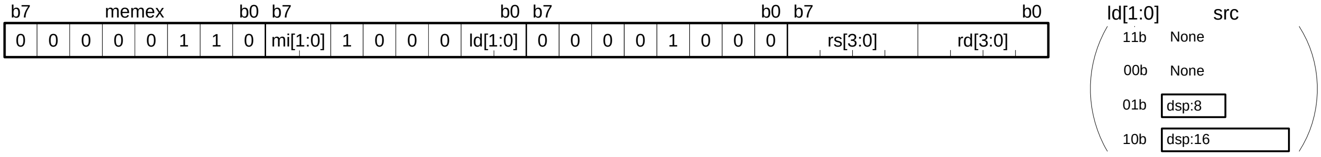
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) DIV src, dest

When memex == UB or src == Rs



When memex != UB



mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

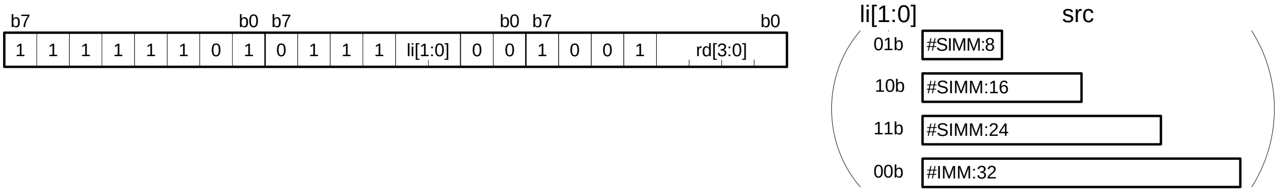
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

DIVU DIVU

Code Size

Syntax	src	dest	Code Size (Byte)
(1) DIVU src, dest	#SIMM:8	Rd	4
	#SIMM:16	Rd	5
	#SIMM:24	Rd	6
	#IMM:32	Rd	7
(2) DIVU src, dest	Rs	Rd	3
	[Rs].memex	Rd	3 (memex == UB) 4 (memex != UB)
	dsp:8[Rs].memex	Rd	4 (memex == UB) 5 (memex != UB)
	dsp:16[Rs].memex	Rd	5 (memex == UB) 6 (memex != UB)

(1) DIVU src, dest

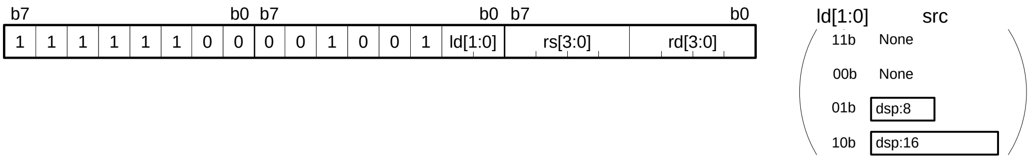


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

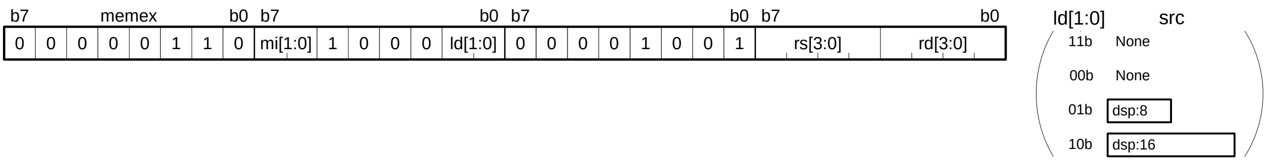
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) DIVU src, dest

When memex == UB or src == Rs



When memex != UB


b7 memex b0 b7 b0 b7 b0 b7 b0
0 0 0 0 0 1 1 0 mi[1:0] 1 0 0 0 ld[1:0] 0 0 0 0 1 0 0 1 rs[3:0] rd[3:0]
ld[1:0] src
11b None
00b None
01b dsp:8
10b dsp:16

mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

EMACA EMACA

Code Size

Syntax	src	dest2	Adest	Code Size (Byte)
(1) EMACA src, src2, Adest	Rs	Rs2	A0, A1	3

(1) EMACA src, src2, Adest

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	1	0	0	0	0	a	1	1	1	rs[3:0]	rs2[3:0]

a	Adest
0b	A0
1b	A1

rs[3:0]/rs2[3:0]	src/src2	
0000b to 1111b	Rs/Rs2	R0 (SP) to R15

EMSBA EMSBA

Code Size

Syntax	src	dest2	Adest	Code Size (Byte)
(1) EMSBA src, src2, Adest	Rs	Rs2	A0, A1	3

(1) EMSBA src, src2, Adest

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	1	0	1	0	0	a	1	1	1	rs[3:0]	rs2[3:0]

a	Adest
0b	A0
1b	A1

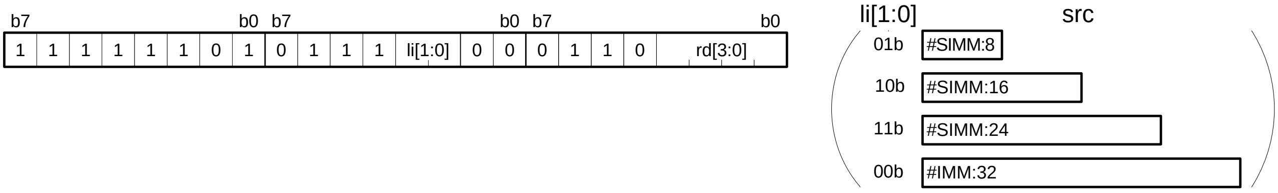
rs[3:0]/rs2[3:0]	src/src2	
0000b to 1111b	Rs/Rs2	R0 (SP) to R15

EMUL EMUL

Code Size

Syntax	src	dest	Code Size (Byte)
(1) EMUL src, dest	#SIMM:8	Rd	4
	#SIMM:16	Rd	5
	#SIMM:24	Rd	6
	#IMM:32	Rd	7
(2) EMUL src, dest	Rs	Rd	3
	[Rs].memex	Rd	3 (memex == UB) 4 (memex != UB)
	dsp:8[Rs].memex	Rd	4 (memex == UB) 5 (memex != UB)
	dsp:16[Rs].memex	Rd	5 (memex == UB) 6 (memex != UB)

(1) EMUL src, dest

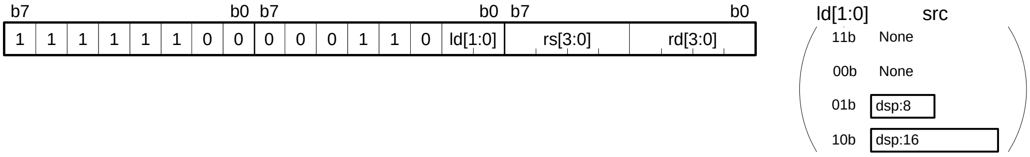


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

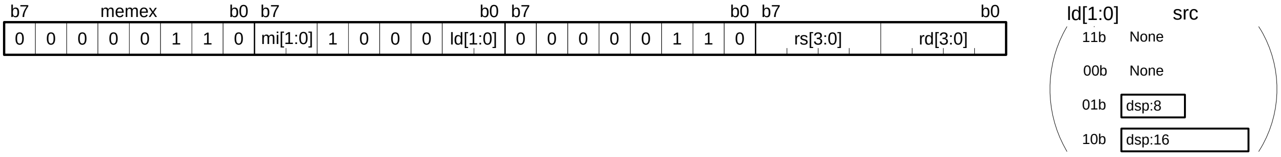
rd[3:0]	dest	
0000b to 1110b	Rd	R0 (SP) to R14

(2) EMUL src, dest

When memex == UB or src == Rs



When memex != UB



mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]	src	
0000b to 1111b	Rs	R0 (SP) to R15

rd[3:0]	dest	
0000b to 1110b	Rd	R0 (SP) to R14

EMULA EMULA

Code Size

Syntax	src	dest2	Adest	Code Size (Byte)
(1) EMULA src, src2, Adest	Rs	Rs2	A0, A1	3

(1) EMULA src, src2, Adest

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	1	0	0	0	0	a	0	1	1	rs[3:0]	rs2[3:0]

a	Adest
0b	A0
1b	A1

rs[3:0]/rs2[3:0]	src/src2	
0000b to 1111b	Rs/Rs2	R0 (SP) to R15

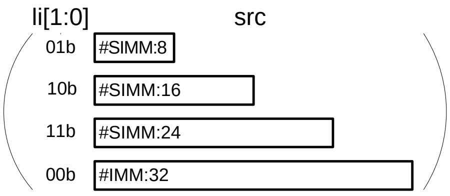
EMULU EMULU

Code Size

Syntax	src	dest	Code Size (Byte)
(1) EMULU src, dest	#SIMM:8	Rd	4
	#SIMM:16	Rd	5
	#SIMM:24	Rd	6
	#IMM:32	Rd	7
(2) EMULU src, dest	Rs	Rd	3
	[Rs].memex	Rd	3 (memex == UB) 4 (memex != UB)
	dsp:8[Rs].memex	Rd	4 (memex == UB) 5 (memex != UB)
	dsp:16[Rs].memex	Rd	5 (memex == UB) 6 (memex != UB)

(1) EMULU src, dest

b7							b0	b7						b0	b7				b0
1	1	1	1	1	1	0	1	0	1	1	1	li[1:0]	0	0	0	1	1	1	rd[3:0]



li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

rd[3:0]	dest	
0000b to 1110b	Rd	R0 (SP) to R14

(2) EMULU src, dest

When memex == UB or src == Rs

b7							b0	b7							b0	b7			b0
1	1	1	1	1	1	0	0	0	0	0	1	1	1	ld[1:0]		rs[3:0]		rd[3:0]	

ld[1:0]	src
11b	None
00b	None
01b	dsp:8
10b	dsp:16

When memex != UB

b7		memex					b0	b7							b0	b7							b0	b7			b0
0	0	0	0	0	1	1	0	mi[1:0]		1	0	0	0	ld[1:0]		0	0	0	0	0	1	1	1	rs[3:0]		rd[3:0]	

ld[1:0]	src
11b	None
00b	None
01b	dsp:8
10b	dsp:16

mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]	src	
0000b to 1111b	Rs	R0 (SP) to R15

rd[3:0]	dest	
0000b to 1110b	Rd	R0 (SP) to R14

FADD FADD

Code Size

Syntax	src	src2	dest	Code Size (Byte)
(1) FADD src, dest	#IMM:32	—	Rd	7
(2) FADD src, dest	Rs	—	Rd	3
	[Rs].L	—	Rd	3
	dsp:8[Rs].L	—	Rd	4
	dsp:16[Rs].L	—	Rd	5
(3) FADD src, src2, dest	Rs	Rs2	Rd	3

(1) FADD src, dest

b7							b0	b7							b0	b7				b0
1	1	1	1	1	1	0	1	0	1	1	1	0	0	1	0	0	0	1	0	rd[3:0]

src
#IMM:32

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) FADD src, dest

b7							b0	b7						b0	b7	b0
1	1	1	1	1	1	0	0	1	0	0	0	1	0	ld[1:0]	rs[3:0]	rd[3:0]

ld[1:0]	src
11b	None
00b	None
01b	dsp:8
10b	dsp:16

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

(3) FADD src, src2, dest

b7							b0	b7				b0	b7	b0
1	1	1	1	1	1	1	1	1	0	1	0	rd[3:0]	rs[3:0]	rs2[3:0]

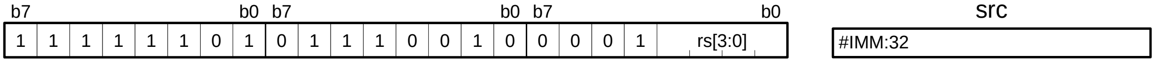
rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b to 1111b	Rs/Rs2/Rd	R0 (SP) to R15

FCMP FCMP

Code Size

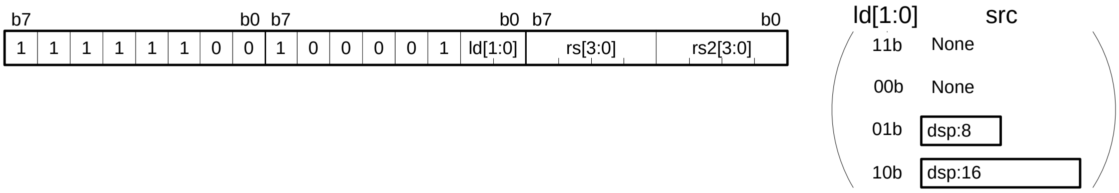
Syntax	src	src2	Code Size (Byte)
(1) FCMP src, src2	#IMM:32	Rs	7
(2) FCMP src, src2	Rs	Rs2	3
	[Rs].L	Rs2	3
	dsp:8[Rs].L	Rs2	4
	dsp:16[Rs].L	Rs2	5

(1) FCMP src, src2



rs[3:0]	src2	
0000b to 1111b	Rs	R0 (SP) to R15

(2) FCMP src, src2



ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rs2[3:0]	src/src2	
0000b to 1111b	Rs/Rs2	R0 (SP) to R15

FDIV FDIV

Code Size

Syntax	src	dest	Code Size (Byte)
(1) FDIV src, dest	#IMM:32	Rd	7
(2) FDIV src, dest	Rs	Rd	3
	[Rs].L	Rd	3
	dsp:8[Rs].L	Rd	4
	dsp:16[Rs].L	Rd	5

(1) FDIV src, dest

b7							b0	b7							b0	b7			b0
1	1	1	1	1	1	0	1	0	1	1	1	0	0	1	0	0	1	0	0	rd[3:0]

src
#IMM:32

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) FDIV src, dest

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	0	1	0	0	1	0	0	ld[1:0]		rs[3:0]	rd[3:0]

ld[1:0]	src
11b	None
00b	None
01b	dsp:8
10b	dsp:16

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

FMUL FMUL

Code Size

Syntax	src	src2	dest	Code Size (Byte)
(1) FMUL src, dest	#IMM:32	—	Rd	7
(2) FMUL src, dest	Rs	—	Rd	3
	[Rs].L	—	Rd	3
	dsp:8[Rs].L	—	Rd	4
	dsp:16[Rs].L	—	Rd	5
(3) FMUL src, src2, dest	Rs	Rs2	Rd	3

(1) FMUL src, dest

b7							b0	b7							b0	b7				b0
1	1	1	1	1	1	0	1	0	1	1	1	0	0	1	0	0	0	1	1	rd[3:0]

src
#IMM:32

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) FMUL src, dest

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	0	1	0	0	0	1	1	ld[1:0]		rs[3:0]	rd[3:0]

ld[1:0]	src
11b	None
00b	None
01b	dsp:8
10b	dsp:16

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

(3) FMUL src, src2, dest

b7							b0	b7					b0	b7	b0
1	1	1	1	1	1	1	1	1	0	1	1	rd[3:0]		rs[3:0]	rs2[3:0]

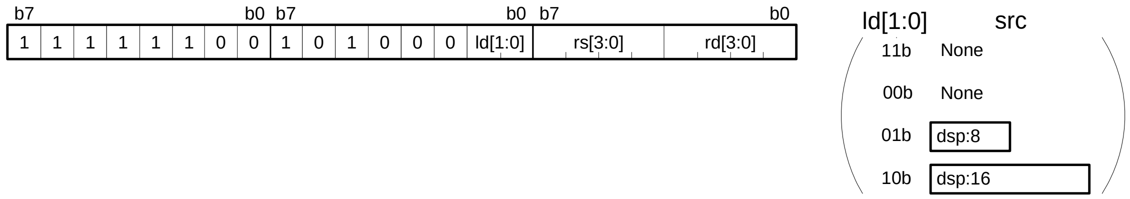
rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b to 1111b	Rs/Rs2/Rd	R0 (SP) to R15

FSQRT FSQRT

Code Size

Syntax	src	dest	Code Size (Byte)
FSQRT src, dest	Rs	Rd	3
	[Rs].L	Rd	3
	dsp:8[Rs].L	Rd	4
	dsp:16[Rs].L	Rd	5

(1) FSQRT src, dest



ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

FSUB FSUB

Code Size

Syntax	src	src2	dest	Code Size (Byte)
(1) FSUB src, dest	#IMM:32	—	Rd	7
(2) FSUB src, dest	Rs	—	Rd	3
	[Rs].L	—	Rd	3
	dsp:8[Rs].L	—	Rd	4
	dsp:16[Rs].L	—	Rd	5
(3) FSUB src, src2, dest	Rs	Rs2	Rd	3

(1) FSUB src, dest

b7							b0	b7							b0	b7			b0
1	1	1	1	1	1	0	1	0	1	1	1	0	0	1	0	0	0	0	0	rd[3:0]

src

#IMM:32

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) FSUB src, dest

b7							b0	b7						b0	b7	b0
1	1	1	1	1	1	0	0	1	0	0	0	0	0	ld[1:0]	rs[3:0]	rd[3:0]

ld[1:0]	src
11b	None
00b	None
01b	dsp:8
10b	dsp:16

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

(3) FSUB src, src2, dest

b7							b0	b7				b0	b7	b0
1	1	1	1	1	1	1	1	1	0	0	0	rd[3:0]	rs[3:0]	rs2[3:0]

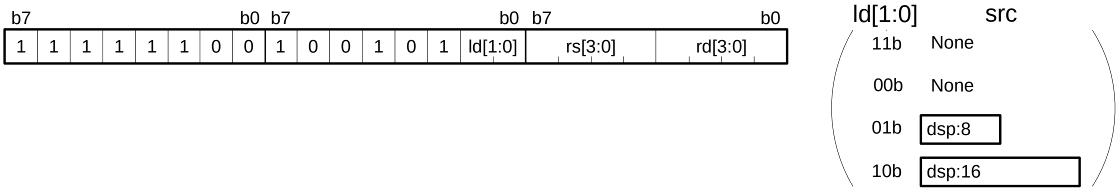
rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b to 1111b	Rs/Rs2/Rd	R0 (SP) to R15

FTOI FTOI

Code Size

Syntax	src	dest	Code Size (Byte)
(1) FTOI src, dest	Rs	Rd	3
	[Rs].L	Rd	3
	dsp:8[Rs].L	Rd	4
	dsp:16[Rs].L	Rd	5

(1) FTOI src, dest



ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

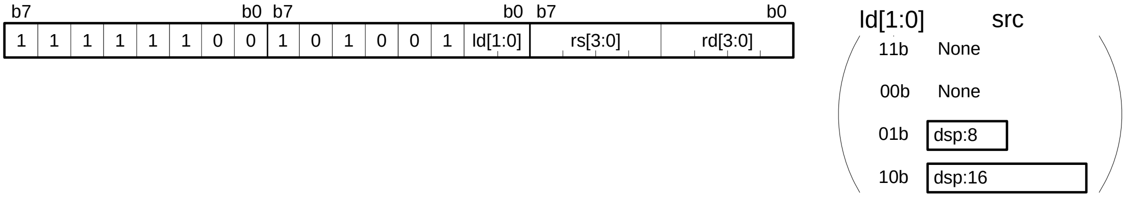
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

FTOU FTOU

Code Size

Syntax	src	dest	Code Size (Byte)
FTOU src, dest	Rs	Rd	3
	[Rs].L	Rd	3
	dsp:8[Rs].L	Rd	4
	dsp:16[Rs].L	Rd	5

(1) FTOU src, dest



ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

INT INT

Code Size

Syntax	src	Code Size (Byte)
(1) INT src	#IMM:8	3

(1) INT src

b7							b0	b7							b0	src
0	1	1	1	0	1	0	1	0	1	1	0	0	0	0	0	#IMM:8

ITOF ITOF

Code Size

Syntax	src	dest	Code Size (Byte)
(1) ITOF src, dest	Rs	Rd	3
	[Rs].memex	Rd	3 (memex == UB) 4 (memex != UB)
	dsp:8[Rs].memex	Rd	4 (memex == UB) 5 (memex != UB)
	dsp:16[Rs].memex	Rd	5 (memex == UB) 6 (memex != UB)

(1) ITOF src, dest

When memex == UB or src == Rs

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	0	0	1	0	0	0	1	ld[1:0]		rs[3:0]	rd[3:0]

ld[1:0]	src
11b	None
00b	None
01b	dsp:8
10b	dsp:16

When memex != UB

b7							b0	b7							b0	b7							b0	b7	b0
			memex																						
0	0	0	0	0	1	1	0	mi[1:0]		1	0	0	0	ld[1:0]		0	0	0	1	0	0	0	1	rs[3:0]	rd[3:0]

ld[1:0]	src
11b	None
00b	None
01b	dsp:8
10b	dsp:16

mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

JMP JMP

Code Size

Syntax	src	Code Size (Byte)
(1) JMP src	Rs	2

(1) JMP src

b7							b0	b7				b0
0	1	1	1	1	1	1	1	0	0	0	0	rs[3:0]

rs[3:0]	src	
0000b to 1111b	Rs	R0 (SP) to R15

JSR JSR

Code Size

Syntax	src	Code Size (Byte)
(1) JSR src	Rs	2

(1) JSR src

b7							b0	b7				b0
0	1	1	1	1	1	1	1	0	0	0	1	rs[3:0]

rs[3:0]	src	
0000b to 1111b	Rs	R0 (SP) to R15

MACHI MACHI

Code Size

Syntax	src	src2	Adest	Code Size (Byte)
(1) MACHI src, src2, Adest	Rs	Rs2	A0, A1	3

(1) MACHI src, src2, Adest

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	1	0	0	0	0	a	1	0	0	rs[3:0]	rs2[3:0]

a	Adest
0b	A0
1b	A1

rs[3:0]/rs2[3:0]	src/src2	
0000b to 1111b	Rs/Rs2	R0 (SP) to R15

MACLH MACLH

Code Size

Syntax	src	src2	Adest	Code Size (Byte)
(1) MACLH src, src2, Adest	Rs	Rs2	A0, A1	3

(1) MACLH src, src2, Adest

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	1	0	0	0	0	a	1	1	0	rs[3:0]	rs2[3:0]

a	Adest
0b	A0
1b	A1

rs[3:0]/rs2[3:0]	src/src2	
0000b to 1111b	Rs/Rs2	R0 (SP) to R15

MACLO MACLO

Code Size

Syntax	src	src2	Adest	Code Size (Byte)
(1) MACLO src, src2, Adest	Rs	Rs2	A0, A1	3

(1) MACLO src, src2, Adest

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	1	0	0	0	0	a	1	0	1	rs[3:0]	rs2[3:0]

a	Adest
0b	A0
1b	A1

rs[3:0]/rs2[3:0]	src/src2	
0000b to 1111b	Rs/Rs2	R0 (SP) to R15

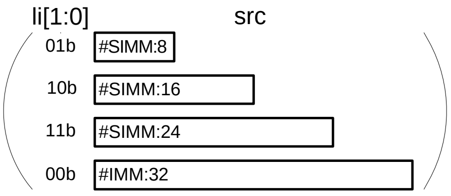
MAX MAX

Code Size

Syntax	src	dest	Code Size (Byte)
(1) MAX src, dest	#SIMM:8	Rd	4
	#SIMM:16	Rd	5
	#SIMM:24	Rd	6
	#IMM:32	Rd	7
(2) MAX src, dest	Rs	Rd	3
	[Rs].memex	Rd	3 (memex == UB) 4 (memex != UB)
	dsp:8[Rs].memex	Rd	4 (memex == UB) 5 (memex != UB)
	dsp:16[Rs].memex	Rd	5 (memex == UB) 6 (memex != UB)

(1) MAX src, dest

b7							b0	b7						b0	b7				b0
1	1	1	1	1	1	0	1	0	1	1	1	li[1:0]	0	0	0	1	0	0	rd[3:0]

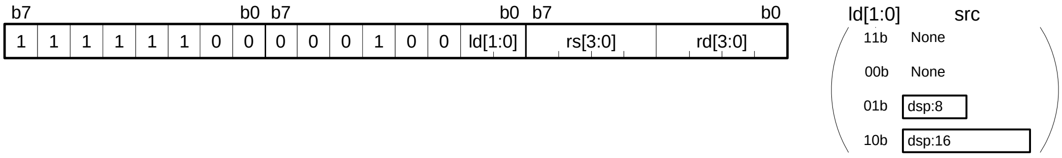


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

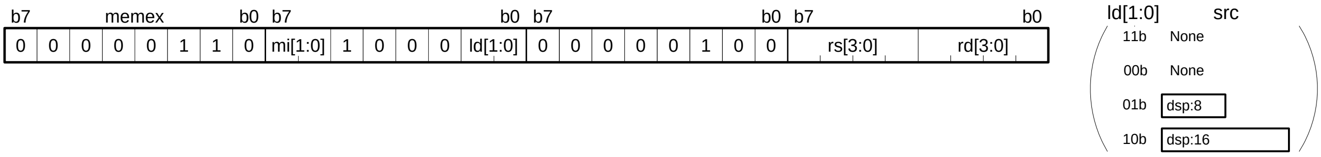
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) MAX src, dest

When memex == UB or src == Rs



When memex != UB



mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

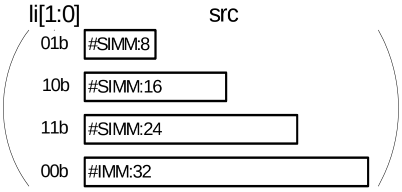
MIN MIN

Code Size

Syntax	src	dest	Code Size (Byte)
(1) MIN src, dest	#SIMM:8	Rd	4
	#SIMM:16	Rd	5
	#SIMM:24	Rd	6
	#IMM:32	Rd	7
(2) MIN src, dest	Rs	Rd	3
	[Rs].memex	Rd	3 (memex == UB) 4 (memex != UB)
	dsp:8[Rs].memex	Rd	4 (memex == UB) 5 (memex != UB)
	dsp:16[Rs].memex	Rd	5 (memex == UB) 6 (memex != UB)

(1) MIN src, dest

b7							b0	b7							b0	b7							b0
1	1	1	1	1	1	0	1	0	1	1	1	li[1:0]		0	0	0	1	0	1	rd[3:0]			



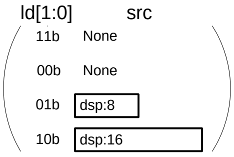
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) MIN src, dest

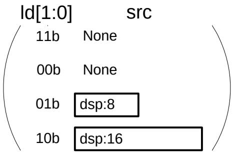
When memex == UB or src == Rs

b7							b0	b7							b0	b7							b0
1	1	1	1	1	1	0	0	0	0	0	1	0	1	ld[1:0]		rs[3:0]				rd[3:0]			



When memex != UB

b7			memex				b0	b7							b0	b7							b0	b7							b0
0	0	0	0	0	1	1	0	mi[1:0]		1	0	0	0	ld[1:0]		0	0	0	0	0	1	0	1	rs[3:0]				rd[3:0]			



mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

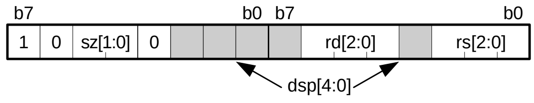
MOV MOV

Code Size

Syntax	Size	Processing Size	src	dest	Code Size (Byte)
(1) MOV.size src, dest	B/W/L	size	Rs (Rs = R0 to R7)	dsp:5[Rd] (Rd = R0 to R7)	2
(2) MOV.size src, dest	B/W/L	L	dsp:5[Rs] (Rs = R0 to R7)	Rd (Rd = R0 to R7)	2
(3) MOV.size src, dest	L	L	#UIMM:4	Rd	2
(4) MOV.size src, dest	B	B	#IMM:8	dsp:5[Rd] (Rd = R0 to R7)	3
	W/L	size	#UIMM:8	dsp:5[Rd] (Rd = R0 to R7)	3
(5) MOV.size src, dest	L	L	#UIMM:8	Rd	3
(6) MOV.size src, dest	L	L	#SIMM:8	Rd	3
	L	L	#SIMM:16	Rd	4
	L	L	#SIMM:24	Rd	5
	L	L	#IMM:32	Rd	6
(7) MOV.size src, dest	B/W	L	Rs	Rd	2
	L	L	Rs	Rd	2
(8) MOV.size src, dest	B	B	#IMM:8	[Rd]	3
	B	B	#IMM:8	dsp:8[Rd]	4
	B	B	#IMM:8	dsp:16[Rd]	5
	W	W	#SIMM:8	[Rd]	3
	W	W	#SIMM:8	dsp:8[Rd]	4
	W	W	#SIMM:8	dsp:16[Rd]	5
	W	W	#IMM:16	[Rd]	4
	W	W	#IMM:16	dsp:8[Rd]	5
	W	W	#IMM:16	dsp:16[Rd]	6
	L	L	#SIMM:8	[Rd]	3
	L	L	#SIMM:8	dsp:8[Rd]	4
	L	L	#SIMM:8	dsp:16 [Rd]	5
	L	L	#SIMM:16	[Rd]	4
	L	L	#SIMM:16	dsp:8[Rd]	5
	L	L	#SIMM:16	dsp:16 [Rd]	6
	L	L	#SIMM:24	[Rd]	5
	L	L	#SIMM:24	dsp:8[Rd]	6
	L	L	#SIMM:24	dsp:16 [Rd]	7
	L	L	#IMM:32	[Rd]	6
	L	L	#IMM:32	dsp:8[Rd]	7
	L	L	#IMM:32	dsp:16 [Rd]	8
(9) MOV.size src, dest	B/W/L	L	[Rs]	Rd	2
	B/W/L	L	dsp:8[Rs]	Rd	3
	B/W/L	L	dsp:16[Rs]	Rd	4
(10) MOV.size src, dest	B/W/L	L	[Ri, Rb]	Rd	3
(11) MOV.size src, dest	B/W/L	size	Rs	[Rd]	2
	B/W/L	size	Rs	dsp:8[Rd]	3
	B/W/L	size	Rs	dsp:16[Rd]	4

Syntax	Size	Processing Size	src	dest	Code Size (Byte)
(12) MOV.size src, dest	B/W/L	size	Rs	[Ri, Rb]	3
(13) MOV.size src, dest	B/W/L	size	[Rs]	[Rd]	2
	B/W/L	size	[Rs]	dsp:8[Rd]	3
	B/W/L	size	[Rs]	dsp:16[Rd]	4
	B/W/L	size	dsp:8[Rs]	[Rd]	3
	B/W/L	size	dsp:8[Rs]	dsp:8[Rd]	4
	B/W/L	size	dsp:8[Rs]	dsp:16[Rd]	5
	B/W/L	size	dsp:16[Rs]	[Rd]	4
	B/W/L	size	dsp:16[Rs]	dsp:8[Rd]	5
	B/W/L	size	dsp:16[Rs]	dsp:16[Rd]	6
(14) MOV.size src, dest	B/W/L	size	Rs	[Rd+]	3
	B/W/L	size	Rs	[–Rd]	3
(15) MOV.size src, dest	B/W/L	L	[Rs+]	Rd	3
	B/W/L	L	[–Rs]	Rd	3

(1) MOV.size src, dest

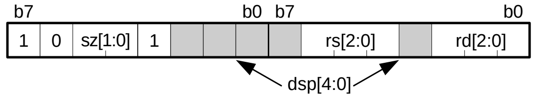


sz[1:0]	Size
00b	B
01b	W
10b	L

dsp[4:0]	dsp:5
00000b to 11111b	0 to 31

rs[2:0]/rd[2:0]	src/dest	
000b to 111b	Rs/Rd	R0 (SP) to R7

(2) MOV.size src, dest

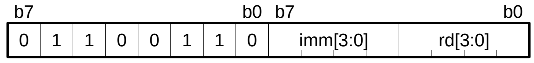


sz[1:0]	Size
00b	B
01b	W
10b	L

dsp[4:0]	dsp:5
00000b to 11111b	0 to 31

rs[2:0]/rd[2:0]	src/dest	
000b to 111b	Rs/Rd	R0 (SP) to R7

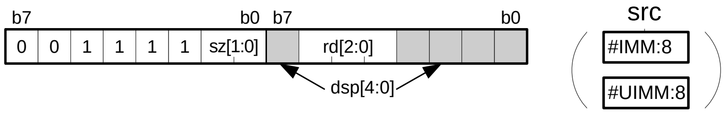
(3) MOV.size src, dest



imm[3:0]	src	
0000b to 1111b	#UIMM:4	0 to 15

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(4) MOV.size src, dest

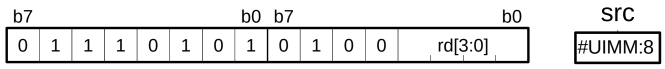


sz[1:0]	Size
00b	B
01b	W
10b	L

dsp[4:0]	dsp:5
00000b to 11111b	0 to 31

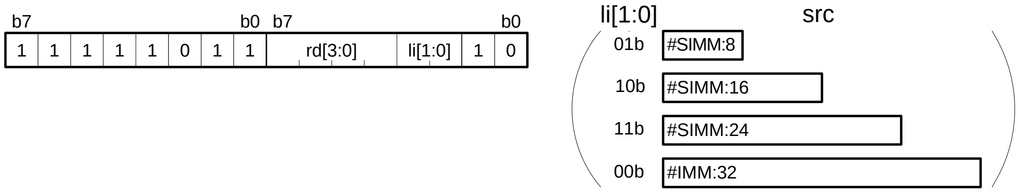
rd[2:0]	dest	
000b to 111b	Rd	R0 (SP) to R7

(5) MOV.size src, dest



rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

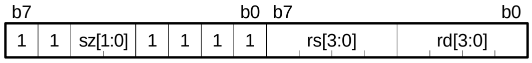
(6) MOV.size src, dest



li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

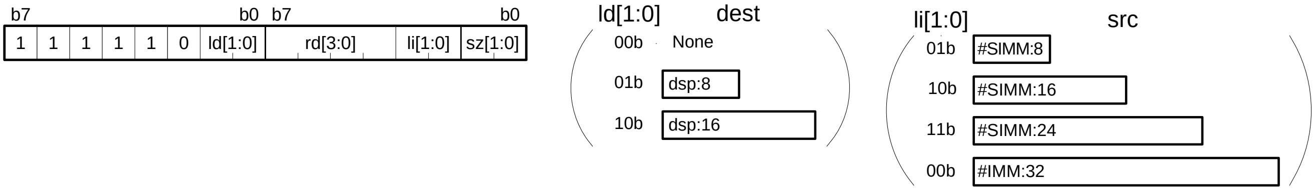
(7) MOV.size src, dest



sz[1:0]	Size
00b	B
01b	W
10b	L

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

(8) MOV.size src, dest



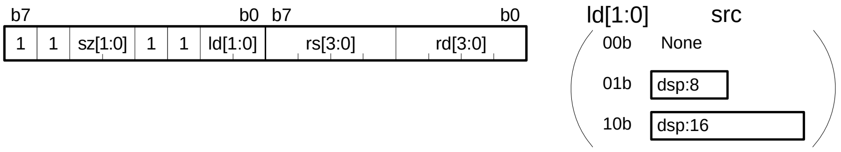
ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

sz[1:0]	Size
00b	B
01b	W
10b	L

(9) MOV.size src, dest

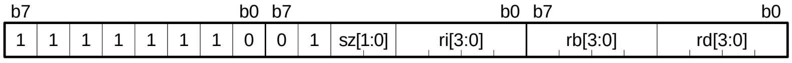


sz[1:0]	Size
00b	B
01b	W
10b	L

ld[1:0]	src
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

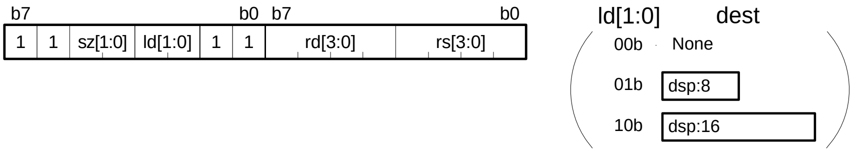
(10) MOV.size src, dest



sz[1:0]	Size
00b	B
01b	W
10b	L

ri[3:0]/rb[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Ri/Rb/Rd	R0 (SP) to R15

(11) MOV.size src, dest



sz[1:0]	Size
00b	B
01b	W
10b	L

ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

(12) MOV.size src, dest

b7							b0	b7							b0	b7							b0
1	1	1	1	1	1	1	0	0	0	sz[1:0]		ri[3:0]				rb[3:0]				rs[3:0]			

sz[1:0]	Size
00b	B
01b	W
10b	L

rs[3:0]/ri[3:0]/rb[3:0]	src/dest	
0000b to 1111b	Rs/Ri/Rb	R0 (SP) to R15

(13) MOV.size src, dest

b7							b0	b7							b0
1	1	sz[1:0]		ldd[1:0]		lds[1:0]		rs[3:0]				rd[3:0]			

lds[1:0]	src
00b	None
01b	dsp:8
10b	dsp:16

ldd[1:0]	dest
00b	None
01b	dsp:8
10b	dsp:16

sz[1:0]	Size
00b	B
01b	W
10b	L

lds[1:0]/ldd[1:0]	src/dest
00b	[Rs]/[Rd]
01b	dsp:8[Rs]/dsp:8[Rd]
10b	dsp:16[Rs]/dsp:16[Rd]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

(14) MOV.size src, dest

b7							b0	b7							b0	b7							b0
1	1	1	1	1	1	0	1	0	0	1	0	ad[1:0]		sz[1:0]		rd[3:0]				rs[3:0]			

ad[1:0]	Addressing
00b	Rs, [Rd+]
01b	Rs, [-Rd]

sz[1:0]	Size
00b	B
01b	W
10b	L

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

(15) MOV.size src, dest

b7							b0	b7							b0	b7							b0
1	1	1	1	1	1	0	1	0	0	1	0	ad[1:0]		sz[1:0]		rs[3:0]				rd[3:0]			

ad[1:0]	Addressing
10b	[Rs+], Rd
11b	[-Rs], Rd

sz[1:0]	Size
00b	B
01b	W
10b	L

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

MOVCO MOVCO

Code Size

Syntax	Size	Processing Size	src	dest	Code Size (Byte)
(1) MOVCO src, dest	L	L	Rs	[Rd]	3

(1) MOVCO src, dest

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	1	0	0	1	0	0	1	1	1	rd[3:0]	rs[3:0]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

MOVLI MOVLI

Code Size

Syntax	Size	Processing Size	src	dest	Code Size (Byte)
(1) MOVLI src, dest	L	L	[Rs]	Rd	3

(1) MOVLI src, dest

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	1	0	0	1	0	1	1	1	1	rs[3:0]	rd[3:0]

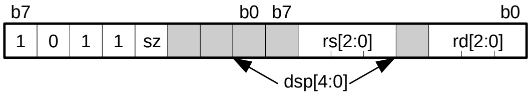
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

MOVU MOVU

Code Size

Syntax	Size	Processing Size	src	dest	Code Size (Byte)
(1) MOVU.size src, dest	B/W	L	dsp:5[Rs] (Rs = R0 to R7)	Rd (Rd = R0 to R7)	2
(2) MOVU.size src, dest	B/W	L	Rs	Rd	2
	B/W	L	[Rs]	Rd	2
	B/W	L	dsp:8[Rs]	Rd	3
	B/W	L	dsp:16[Rs]	Rd	4
(3) MOVU.size src, dest	B/W	L	[Ri, Rb]	Rd	3
(4) MOVU.size src, dest	B/W	L	[Rs+]	Rd	3
	B/W	L	[–Rs]	Rd	3

(1) MOVU.size src, dest

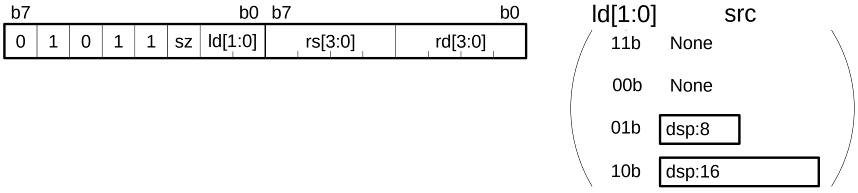


sz	Size
0b	B
1b	W

dsp[4:0]	dsp:5
00000b to 11111b	0 to 31

rs[2:0]/rd[2:0]	src/dest	
000b to 111b	Rs/Rd	R0 (SP) to R7

(2) MOVU.size src, dest

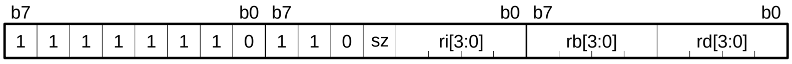


sz	Size
0b	B
1b	W

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

(3) MOVU.size src, dest



sz	Size
0b	B
1b	W

ri[3:0]/rb[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Ri/Rb/Rd	R0 (SP) to R15

(4) MOVU.size src, dest

b7							b0	b7						b0	b7	b0
1	1	1	1	1	1	0	1	0	0	1	1	ad[1:0]	0	sz	rs[3:0]	rd[3:0]

ad[1:0]	Addressing
10b	[Rs+], Rd
11b	[-Rs], Rd

sz	Size
0b	B
1b	W

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

MSBHI MSBHI

Code Size

Syntax	src	src2	Adest	Code Size (Byte)
(1) MSBHI src, src2, Adest	Rs	Rs2	A0, A1	3

(1) MSBHI src, src2, Adest

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	1	0	1	0	0	a	1	0	0	rs[3:0]	rs2[3:0]

a	Adest
0b	A0
1b	A1

rs[3:0]/rs2[3:0]	src/src2	
0000b to 1111b	Rs/Rs2	R0 (SP) to R15

MSBLH MSBLH

Code Size

Syntax	src	src2	Adest	Code Size (Byte)
(1) MSBLH src, src2, Adest	Rs	Rs2	A0, A1	3

(1) MSBLH src, src2, Adest

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	1	0	1	0	0	a	1	1	0	rs[3:0]	rs2[3:0]

a	Adest
0b	A0
1b	A1

rs[3:0]/rs2[3:0]	src/src2	
0000b to 1111b	Rs/Rs2	R0 (SP) to R15

MSBLO MSBLO

Code Size

Syntax	src	src2	Adest	Code Size (Byte)
(1) MSBLO src, src2, Adest	Rs	Rs2	A0, A1	3

(1) MSBLO src, src2, Adest

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	1	0	1	0	0	a	1	0	1	rs[3:0]	rs2[3:0]

a	Adest
0b	A0
1b	A1

rs[3:0]/rs2[3:0]	src/src2	
0000b to 1111b	Rs/Rs2	R0 (SP) to R15

MUL MUL

Code Size

Syntax	src	src2	dest	Code Size (Byte)
(1) MUL src, dest	#UIMM:4	–	Rd	2
(2) MUL src, dest	#SIMM:8	–	Rd	3
	#SIMM:16	–	Rd	4
	#SIMM:24	–	Rd	5
	#IMM:32	–	Rd	6
(3) MUL src, dest	Rs	–	Rd	2
	[Rs].memex	–	Rd	2 (memex == UB) 3 (memex != UB)
	dsp:8[Rs].memex	–	Rd	3 (memex == UB) 4 (memex != UB)
	dsp:16[Rs].memex	–	Rd	4 (memex == UB) 5 (memex != UB)
(4) MUL src, src2, dest	Rs	Rs2	Rd	3

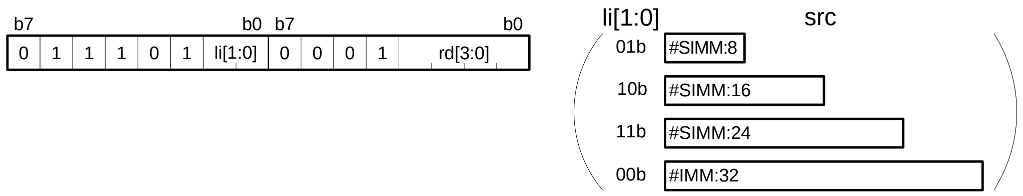
(1) MUL src, dest

b7							b0	b7	b0
0	1	1	0	0	0	1	1	imm[3:0]	rd[3:0]

imm[3:0]	src	
0000b to 1111b	#UIMM:4	0 to 15

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) MUL src, dest

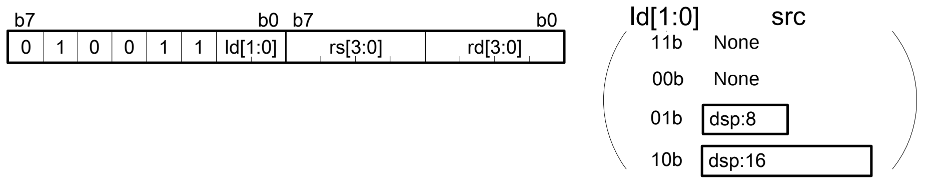


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

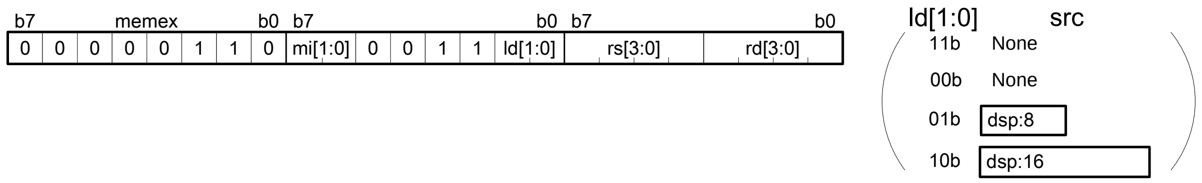
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(3) MUL src, dest

When memex == UB or src == Rs



When memex != UB

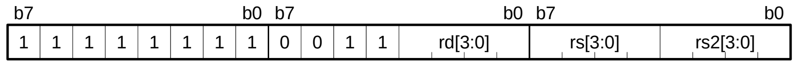


mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

(4) MUL src, src2, dest



rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b to 1111b	Rs/Rs2/Rd	R0 (SP) to R15

MULHI MULHI

Code Size

Syntax	src	src2	Adest	Code Size (Byte)
(1) MULHI src, src2, Adest	Rs	Rs2	A0, A1	3

(1) MULHI src, src2, Adest

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	1	0	0	0	0	a	0	0	0	rs[3:0]	rs2[3:0]

a	Adest
0b	A0
1b	A1

rs[3:0]/rs2[3:0]	src/src2	
0000b to 1111b	Rs/Rs2	R0 (SP) to R15

MULLH MULLH

Code Size

Syntax	src	src2	Adest	Code Size (Byte)
(1) MULLH src, src2, Adest	Rs	Rs2	A0, A1	3

(1) MULLH src, src2, Adest

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	1	0	0	0	0	a	0	1	0	rs[3:0]	rs2[3:0]

a	Adest
0b	A0
1b	A1

rs[3:0]/rs2[3:0]	src/src2	
0000b to 1111b	Rs/Rs2	R0 (SP) to R15

MULLO MULLO

Code Size

Syntax	src	src2	Adest	Code Size (Byte)
(1) MULLO src, src2, Adest	Rs	Rs2	A0, A1	3

(1) MULLO src, src2, Adest

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	1	0	0	0	0	a	0	0	1	rs[3:0]	rs2[3:0]

a	Adest
0b	A0
1b	A1

rs[3:0]/rs2[3:0]	src/src2	
0000b to 1111b	Rs/Rs2	R0 (SP) to R15

MVFACGU MVFACGU

Code Size

Syntax	src	Asrc	dest	Code Size (Byte)
(1) MVFACGU src, Asrc, dest	#IMM:2	A0, A1	Rd	3

(1) MVFACGU src, Asrc, dest

b7							b0	b7							b0	b7				b0
1	1	1	1	1	1	0	1	0	0	0	1	1	1	1	imm[1]	a	imm[0]	1	1	rd[3:0]

a	Asrc
0b	A0
1b	A1

imm[1:0]	src	
00	#IMM:2	2
01		—
10		0
11		1

dest	
Rd	R0 (SP) to R15

MVFACHI MVFACHI

Code Size

Syntax	src	Asrc	dest	Code Size (Byte)
(1) MVFACHI src, Asrc, dest	#IMM:2	A0, A1	Rd	3

(1) MVFACHI src, Asrc, dest

b7							b0	b7							b0	b7							b0
1	1	1	1	1	1	0	1	0	0	0	1	1	1	1	imm [1]	a	imm [0]	0	0	rd[3:0]			

a	Asrc
0b	A0
1b	A1

imm[1:0]	src	
00	#IMM:2	2
01		—
10		0
11		1

dest	
Rd	R0 (SP) to R15

MVFACLO MVFACLO

Code Size

Syntax	src	Asrc	dest	Code Size (Byte)
(1) MVFACLO src, Asrc, dest	#IMM:2	A0, A1	Rd	3

(1) MVFACLO src, Asrc, dest

b7							b0	b7							b0	b7							b0
1	1	1	1	1	1	0	1	0	0	0	1	1	1	1	imm [1]	a	imm [0]	0	1	rd[3:0]			

a	Asrc
0b	A0
1b	A1

imm[1:0]	src	
00	#IMM:2	2
01		—
10		0
11		1

dest	
Rd	R0 (SP) to R15

MVFACMI MVFACMI

Code Size

Syntax	src	Asrc	dest	Code Size (Byte)
(1) MVFACMI src, Asrc, dest	#IMM:2	A0, A1	Rd	3

(1) MVFACMI src, Asrc, dest

b7							b0	b7							b0	b7							b0
1	1	1	1	1	1	0	1	0	0	0	1	1	1	1	imm[1]	a	imm[0]	1	0	rd[3:0]			

a	Asrc
0b	A0
1b	A1

imm[1:0]	src	
00	#IMM:2	2
01		—
10		0
11		1

dest	
Rd	R0 (SP) to R15

MVFC MVFC

Code Size

Syntax	src	dest	Code Size (Byte)
(1) MVFC src, dest	Rx	Rd	3

(1) MVFC src, dest

b7							b0	b7							b0	b7							b0
1	1	1	1	1	1	0	1	0	1	1	0	1	0	1	0	cr[3:0]				rd[3:0]			

cr[3:0]	src	
0000b	Rx	PSW
0001b		PC
0010b		USP
0011b		FPSW
0100b		Reserved
0101b		Reserved
0110b		Reserved
0111b		Reserved
1000b		BPSW
1001b		BPC
1010b		ISP
1011b		FINTV
1100b		INTB
1101b		EXTB
1110b to 1111b		Reserved

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

MVTACGU MVTACGU

Code Size

Syntax	src	Adest	Code Size (Byte)
(1) MVTACGU src, Adest	Rs	A0, A1	3

(1) MVTACGU src, Adest

b7							b0	b7							b0	b7				b0
1	1	1	1	1	1	0	1	0	0	0	1	0	1	1	1	a	0	1	1	rs[3:0]

a	Adest
0b	A0
1b	A1

rs[3:0]	src	
0000b to 1111b	Rs	R0 (SP) to R15

MVTACHI MVTACHI

Code Size

Syntax	src	Adest	Code Size (Byte)
(1) MVTACGU src, Adest	Rs	A0, A1	3

(1) MVTACHI src, Adest

b7							b0	b7							b0	b7				b0
1	1	1	1	1	1	0	1	0	0	0	1	0	1	1	1	a	0	0	0	rs[3:0]

a	Adest
0b	A0
1b	A1

rs[3:0]	src	
0000b to 1111b	Rs	R0 (SP) to R15

MVTACLO MVTACLO

Code Size

Syntax	src	Adest	Code Size (Byte)
(1) MVTACLO src, Adest	Rs	A0, A1	3

(1) MVTACLO src, Adest

b7							b0	b7							b0	b7							b0
1	1	1	1	1	1	0	1	0	0	0	1	0	1	1	1	a	0	0	1	rs[3:0]			

a	Adest
0b	A0
1b	A1

rs[3:0]	src	
0000b to 1111b	Rs	R0 (SP) to R15

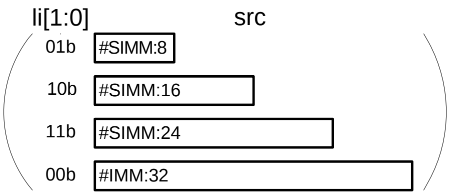
MVTC MVTC

Code Size

Syntax	src	dest	Code Size (Byte)
(1) MVTC src, dest	#SIMM:8	Rx	4
	#SIMM:16	Rx	5
	#SIMM:24	Rx	6
	#IMM:32	Rx	7
(2) MVTC src, dest	Rs	Rx	3

(1) MVTC src, dest

b7							b0	b7						b0	b7				b0
1	1	1	1	1	1	0	1	0	1	1	1	li[1:0]	1	1	0	0	0	0	cr[3:0]



li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

cr[3:0]	dest	
0000b	Rx	PSW
0001b		Reserved
0010b		USP
0011b		FPSW
0100b		Reserved
0101b		Reserved
0110b		Reserved
0111b		Reserved
1000b		BPSW
1001b		BPC
1010b		ISP
1011b		FINTV
1100b		INTB
1101b		EXTB
1110b to 1111b		Reserved

(2) MVTC src, dest

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	1	0	1	1	0	1	0	0	0	rs[3:0]	cr[3:0]

cr[3:0]	dest	
0000b	Rx	PSW
0001b		Reserved
0010b		USP
0011b		FPSW
0100b		Reserved
0101b		Reserved
0110b		Reserved
0111b		Reserved
1000b		BPSW
1001b		BPC
1010b		ISP
1011b		FINTV
1100b		INTB
1101b		EXTB
1110b to 1111b		Reserved

rs[3:0]	src	
0000b to 1111b	Rs	R0 (SP) to R15

MVTIPL MVTIPL

Code Size

Syntax	src	Code Size (Byte)
(1) MVTIPL src	#IMM:4	3

(1) MVTIPL src

b7							b0	b7							b0	b7				b0
0	1	1	1	0	1	0	1	0	1	1	1	0	0	0	0	0	0	0	0	imm[3:0]

imm[3:0]	#IMM:4
0000b to 1111b	0 to 15

NEG NEG

Code Size

Syntax	src	dest	Code Size (Byte)
(1) NEG dest	–	Rd	2
(2) NEG src, dest	Rs	Rd	3

(1) NEG dest

b7							b0	b7				b0
0	1	1	1	1	1	1	0	0	0	0	1	rd[3:0]

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) NEG src, dest

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	0	0	0	0	0	0	1	1	1	rs[3:0]	rd[3:0]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

NOP NOP

Code Size

Syntax	Code Size (Byte)
(1) NOP	1

(1) NOP

b7							b0
0	0	0	0	0	0	1	1

NOT NOT

Code Size

Syntax	src	dest	Code Size (Byte)
(1) NOT dest	–	Rd	2
(2) NOT src, dest	Rs	Rd	3

(1) NOT dest

b7							b0	b7				b0
0	1	1	1	1	1	1	0	0	0	0	0	rd[3:0]

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) NOT src, dest

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	0	0	0	1	1	1	0	1	1	rs[3:0]	rd[3:0]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

OR OR

Code Size

Syntax		src	src2	dest	Code Size (Byte)
(1) OR	src, dest	#UIMM:4	−	Rd	2
(2) OR	src, dest	#SIMM:8	−	Rd	3
		#SIMM:16	−	Rd	4
		#SIMM:24	−	Rd	5
		#IMM:32	−	Rd	6
(3) OR	src, dest	Rs	−	Rd	2
		[Rs].memex	−	Rd	2 (memex == UB) 3 (memex != UB)
		dsp:8[Rs].memex	−	Rd	3 (memex == UB) 4 (memex != UB)
		dsp:16[Rs].memex	−	Rd	4 (memex == UB) 5 (memex != UB)
(4) OR	src, src2, dest	Rs	Rs2	Rd	3

(1) OR src, dest

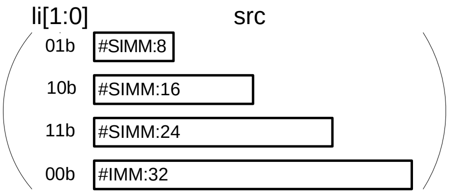
b7							b0	b7		b0
0	1	1	0	0	1	0	1	imm[3:0]	rd[3:0]	

imm[3:0]	src	
0000b to 1111b	#UIMM:4	0 to 15

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) OR src, dest

b7							b0	b7					b0
0	1	1	1	0	1	li[1:0]		0	0	1	1	rd[3:0]	



li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(3) OR src, dest

When memex == UB or src == Rs

b7						b0	b7	b0
0	1	0	1	0	1	ld[1:0]	rs[3:0]	rd[3:0]

ld[1:0]	src
11b	None
00b	None
01b	dsp:8
10b	dsp:16

When memex != UB

b7			memex				b0	b7					b0	b7	b0
0	0	0	0	0	1	1	0	mi[1:0]	0	1	0	1	ld[1:0]	rs[3:0]	rd[3:0]

ld[1:0]	src
11b	None
00b	None
01b	dsp:8
10b	dsp:16

mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

(4) OR src, src2, dest

b7							b0	b7				b0	b7	b0
1	1	1	1	1	1	1	1	0	1	0	1	rd[3:0]	rs[3:0]	rs2[3:0]

rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b to 1111b	Rs/Rs2/Rd	R0 (SP) to R15

POP POP

Code Size

Syntax	dest	Code Size (Byte)
(1) POP dest	Rd	2

(1) POP dest

b7							b0	b7				b0
0	1	1	1	1	1	1	0	1	0	1	1	rd[3:0]

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

POPC POPC

Code Size

Syntax	dest	Code Size (Byte)
(1) POPC dest	Rx	2

(1) POPC dest

b7							b0	b7							b0
0	1	1	1	1	1	1	0	1	1	1	0	cr[3:0]			

cr[3:0]	dest	
0000b	Rx	PSW
0001b		Reserved
0010b		USP
0011b		FPSW
0100b		Reserved
0101b		Reserved
0110b		Reserved
0111b		Reserved
1000b		BPSW
1001b		BPC
1010b		ISP
1011b		FINTV
1100b		INTB
1101b		EXTB
1110b to 1111b		Reserved

POPM POPM

Code Size

Syntax	dest	dest2	Code Size (Byte)
(1) POPM dest-dest2	Rd	Rd2	2

(1) POPM dest-dest2

b7							b0	b7							b0
0	1	1	0	1	1	1	1	rd[3:0]				rd2[3:0]			

rd[3:0]	dest	
0001b to 1110b	Rd	R1 to R14

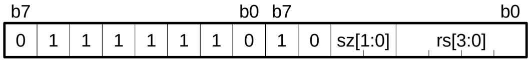
rd2[3:0]	dest2	
0010b to 1111b	Rd2	R2 to R15

PUSH PUSH

Code Size

Syntax	src	Code Size (Byte)
(1) PUSH.size src	Rs	2
(2) PUSH.size src	[Rs]	2
	dsp:8[Rs]	3
	dsp:16[Rs]	4

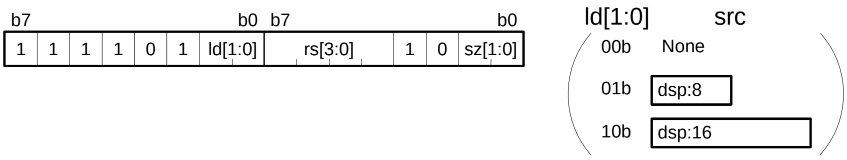
(1) PUSH.size src



sz[1:0]	Size
00b	B
01b	W
10b	L

rs[3:0]	src	
0000b to 1111b	Rs	R0 (SP) to R15

(2) PUSH.size src



ld[1:0]	src
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]	src	
0000b to 1111b	Rs	R0 (SP) to R15

sz[1:0]	Size
00b	B
01b	W
10b	L

PUSHC PUSHC

Code Size

Syntax	src	Code Size (Byte)
(1) PUSHC src	Rx	2

(1) PUSHC src

b7							b0	b7							b0
0	1	1	1	1	1	1	0	1	1	0	0	cr[3:0]			

cr[3:0]	src	
0000b	Rx	PSW
0001b		PC
0010b		USP
0011b		FPSW
0100b		Reserved
0101b		Reserved
0110b		Reserved
0111b		Reserved
1000b		BPSW
1001b		BPC
1010b		ISP
1011b		FINTV
1100b		INTB
1101b		EXTB
1110b to 1111b		Reserved

PUSHM PUSHM

Code Size

Syntax	src	src2	Code Size (Byte)
(1) PUSHM src-src2	Rs	Rs2	2

(1) PUSHM src-src2

b7							b0	b7							b0
0	1	1	0	1	1	1	0	rs[3:0]				rs2[3:0]			

rs[3:0]	src	
0001b to 1110b	Rs	R1 to R14

rs2[3:0]	src2	
0010b to 1111b	Rs2	R2 to R15

RACL RACL

Code Size

Syntax	src	Adest	Code Size (Byte)
(1) RACL src, Adest	#IMM:1 (IMM:1 = 1, 2)	A0, A1	3

(1) RACL src, Adest

b7							b0	b7							b0	b7							b0
1	1	1	1	1	1	0	1	0	0	0	1	1	0	0	1	a	0	0	imm	0	0	0	0

a	Adest
0b	A0
1b	A1

imm	src/src2	
0b,1b	#IMM:1	1, 2

RACW RACW

Code Size

Syntax	src	Adest	Code Size (Byte)
(1) RACW src, Adest	#IMM:1	A0, A1	3

(1) RACW src, Adest

b7							b0	b7							b0	b7							b0
1	1	1	1	1	1	0	1	0	0	0	1	1	0	0	0	0	0	0	imm	0	0	0	0

a	Adest
0b	A0
1b	A1

imm	src/src2	
0b, 1b	#IMM:1	1, 2

RDACL RDACL

Code Size

Syntax	src	Adest	Code Size (Byte)
(1) RDACL src, Adest	#IMM:1 (IMM:1 = 1, 2)	A0, A1	3

(1) RDACL src, Adest

b7							b0	b7							b0	b7							b0
1	1	1	1	1	1	0	1	0	0	0	1	1	0	0	1	a	1	0	imm	0	0	0	0

a	Adest
0b	A0
1b	A1

imm	src/src2	
0b,1b	#IMM:1	1, 2

RDACW RDACW

Code Size

Syntax	src	Adest	Code Size (Byte)
(1) RDACW src, Adest	#IMM:1 (IMM:1 = 1, 2)	A0, A1	3

(1) RDACW src, Adest

b7							b0	b7							b0	b7							b0
1	1	1	1	1	1	0	1	0	0	0	1	1	0	0	0	a	1	0	imm	0	0	0	0

a	Adest
0b	A0
1b	A1

imm	src/src2	
0b, 1b	#IMM:1	1, 2

REVL REVL

Code Size

Syntax	src	dest	Code Size (Byte)
(1) REVL src, dest	Rs	Rd	3

(1) REVL src, dest

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	1	0	1	1	0	0	1	1	1	rs[3:0]	rd[3:0]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

REVW REVW

Code Size

Syntax	src	dest	Code Size (Byte)
(1) REVW src, dest	Rs	Rd	3

(1) REVW src, dest

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	1	0	1	1	0	0	1	0	1	rs[3:0]	rd[3:0]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

RMPA RMPA

Code Size

Syntax	Size	Code Size (Byte)
(1) RMPA.size	B	2
	W	2
	L	2

(1) RMPA.size

b7							b0	b7							b0
0	1	1	1	1	1	1	1	1	0	0	0	1	1	sz[1:0]	

sz[1:0]	Size
00b	B
01b	W
10b	L

ROLC ROLC

Code Size

Syntax	dest	Code Size (Byte)
(1) ROLC dest	Rd	2

(1) ROLC dest

b7							b0	b7							b0
0	1	1	1	1	1	1	0	0	1	0	1	rd[3:0]			

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

RORC RORC

Code Size

Syntax	dest	Code Size (Byte)
(1) RORC dest	Rd	2

(1) RORC dest

b7							b0	b7				b0
0	1	1	1	1	1	1	0	0	1	0	0	rd[3:0]

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

ROTL ROTL

Code Size

Syntax	src	dest	Code Size (Byte)
(1) ROTL src, dest	#IMM:5	Rd	3
(2) ROTL src, dest	Rs	Rd	3

(1) ROTL src, dest

b7							b0	b7							b0 b7	b0
1	1	1	1	1	1	0	1	0	1	1	0	1	1	1	imm[4:0]	rd[3:0]

imm[4:0]	src	
00000b to 11111b	#IMM:5	0 to 31

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) ROTL src, dest

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	1	0	1	1	0	0	1	1	0	rs[3:0]	rd[3:0]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

ROTR ROTR

Code Size

Syntax	src	dest	Code Size (Byte)
(1) ROTR src, dest	#IMM:5	Rd	3
(2) ROTR src, dest	Rs	Rd	3

(1) ROTR src, dest

b7							b0	b7							b0 b7	b0
1	1	1	1	1	1	0	1	0	1	1	0	1	1	0	imm[4:0]	rd[3:0]

imm[4:0]	src	
00000b to 11111b	#IMM:5	0 to 31

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) ROTR src, dest

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	1	0	1	1	0	0	1	0	0	rs[3:0]	rd[3:0]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

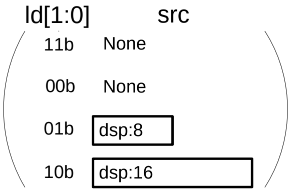
ROUND ROUND

Code Size

Syntax	src	dest	Code Size (Byte)
(1) ROUND src, dest	Rs	Rd	3
	[Rs].L	Rd	3
	dsp:8[Rs].L	Rd	4
	dsp:16[Rs].L	Rd	5

(1) ROUND src, dest

b7							b0	b7						b0	b7	b0
1	1	1	1	1	1	0	0	1	0	0	1	1	0	ld[1:0]	rs[3:0]	rd[3:0]



ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

RTE RTE

Code Size

Syntax	Code Size (Byte)
(1) RTE	2

(1) RTE

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

RTFI RTFI

Code Size

Syntax	Code Size (Byte)
(1) RTFI	2

(1) RTFI

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

RTS RTS

Code Size

Syntax	Code Size (Byte)
(1) RTS	1

(1) RTS

b7							b0
0	0	0	0	0	0	1	0

RTSD RTSD

Code Size

Syntax	src	dest	dest2	Code Size (Byte)
(1) RTSD src	#UIMM:8	–	–	2
(2) RTSD src, dest-dest2	#UIMM:8	Rd	Rd2	3

(1) RTSD src

b7							b0	src
0	1	1	0	0	1	1	1	#UIMM:8

(2) RTSD src, dest-dest2

b7							b0	b7	b0	src
0	0	1	1	1	1	1	1	rd[3:0]	rd2[3:0]	#UIMM:8

rd[3:0]/rd2[3:0]	dest/dest2	
0001b to 1111b	Rd/Rd2	R1 to R15

SAT SAT

Code Size

Syntax	dest	Code Size (Byte)
(1) SAT dest	Rd	2

(1) SAT dest

b7							b0	b7							b0
0	1	1	1	1	1	1	0	0	0	1	1	rd[3:0]			

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

SATR SATR

Code Size

Syntax	Code Size (Byte)
(1) SATR	2

(1) SATR

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

SBB SBB

Code Size

Syntax	src	dest	Code Size (Byte)
(1) SBB src, dest	Rs	Rd	3
(2) SBB src, dest	[Rs].L	Rd	4
	dsp:8[Rs].L	Rd	5
	dsp:16[Rs].L	Rd	6

(1) SBB src, dest

b7							b0	b7						b0	b7	b0
1	1	1	1	1	1	0	0	0	0	0	0	0	0	ld[1:0]	rs[3:0]	rd[3:0]

ld[1:0]	src
11b	Rs

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

(2) SBB src, dest

b7			memex				b0	b7						b0	b7							b0	b7	b0
0	0	0	0	0	1	1	0	1	0	1	0	0	0	ld[1:0]	0	0	0	0	0	0	0	0	rs[3:0]	rd[3:0]

ld[1:0]	src
00b	None
01b	dsp:8
10b	dsp:16

ld[1:0]	src
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

SCCnd SCCnd

Code Size

Syntax	Size	dest	Code Size (Byte)
(1) SCC*nd*.size dest	L	Rd	3
	B/W/L	[Rd]	3
	B/W/L	dsp:8[Rd]	4
	B/W/L	dsp:16[Rd]	5

(1) SCC*nd*.size dest

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	0	1	1	0	1	sz[1:0]	ld[1:0]			rd[3:0]	cd[3:0]

ld[1:0]	dest
11b	None
00b	None
01b	dsp:8
10b	dsp:16

sz[1:0]	Size
00b	B
01b	W
10b	L

ld[1:0]	dest
11b	Rd
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

cd[3:0]	SCC*nd*	cd[3:0]	SCC*nd*
0000b	SCEQ, SCZ	1000b	SCGE
0001b	SCNE, SCNZ	1001b	SCLT
0010b	SCGEU, SCC	1010b	SCGT
0011b	SCLTU, SCNC	1011b	SCLE
0100b	SCGTU	1100b	SCO
0101b	SCLEU	1101b	SCNO
0110b	SCPZ	1110b	Reserved
0111b	SCN	1111b	Reserved

SCMPU SCMPU

Code Size

Syntax	Code Size (Byte)
(1) SCMPU	2

(1) SCMPU

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

SETPSW SETPSW

Code Size

Syntax	dest	Code Size (Byte)
(1) SETPSW dest	flag	2

(1) SETPSW dest

b7							b0	b7							b0
0	1	1	1	1	1	1	1	1	0	1	0	cb[3:0]			

cb[3:0]	dest	
0000b	flag	C
0001b		Z
0010b		S
0011b		O
0100b		Reserved
0101b		Reserved
0110b		Reserved
0111b		Reserved
1000b		I
1001b		U
1010b		Reserved
1011b		Reserved
1100b		Reserved
1101b		Reserved
1110b		Reserved
1111b		Reserved

SHAR SHAR

Code Size

Syntax	src	src2	dest	Code Size (Byte)
(1) SHAR src, dest	#IMM:5	–	Rd	2
(2) SHAR src, dest	Rs	–	Rd	3
(3) SHAR src, src2, dest	#IMM:5	Rs	Rd	3

(1) SHAR src, dest

b7 b0 b7 b0

0	1	1	0	1	0	1	imm[4:0]	rd[3:0]

imm[4:0]	src	
00000b to 11111b	#IMM:5	0 to 31

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) SHAR src, dest

b7 b0 b7 b0 b7 b0

1	1	1	1	1	1	0	1	0	1	1	0	0	0	0	1	rs[3:0]	rd[3:0]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

(3) SHAR src, src2, dest

b7 b0 b7 b0 b7 b0

1	1	1	1	1	1	0	1	1	0	1	imm[4:0]	rs2[3:0]	rd[3:0]

imm[4:0]	src	
00000b to 11111b	#IMM:5	0 to 31

rs2[3:0]/rd[3:0]	src2/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

SHLL SHLL

Code Size

Syntax	src	src2	dest	Code Size (Byte)
(1) SHLL src, dest	#IMM:5	–	Rd	2
(2) SHLL src, dest	Rs	–	Rd	3
(3) SHLL src, src2, dest	#IMM:5	Rs	Rd	3

(1) SHLL src, dest

b7							b0 b7	b0
0	1	1	0	1	1	0	imm[4:0]	rd[3:0]

imm[4:0]	src	
00000b to 11111b	#IMM:5	0 to 31

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) SHLL src, dest

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	1	0	1	1	0	0	0	1	0	rs[3:0]	rd[3:0]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

(3) SHLL src, src2, dest

b7							b0	b7			b0	b7	b0
1	1	1	1	1	1	0	1	1	1	0	imm[4:0]	rs2[3:0]	rd[3:0]

imm[4:0]	src	
00000b to 11111b	#IMM:5	0 to 31

rs2[3:0]/rd[3:0]	src2/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

SHLR SHLR

Code Size

Syntax	src	src2	dest	Code Size (Byte)
(1) SHLR src, dest	#IMM:5	−	Rd	2
(2) SHLR src, dest	Rs	−	Rd	3
(3) SHLR src, src2, dest	#IMM:5	Rs	Rd	3

(1) SHLR src, dest

b7							b0 b7	b0
0	1	1	0	1	0	0	imm[4:0]	rd[3:0]

imm[4:0]	src	
00000b to 11111b	#IMM:5	0 to 31

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) SHLR src, dest

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	1	0	1	1	0	0	0	0	0	rs[3:0]	rd[3:0]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

(3) SHLR src, src2, dest

b7							b0	b7			b0	b7	b0
1	1	1	1	1	1	0	1	1	0	0	imm[4:0]	rs2[3:0]	rd[3:0]

imm[4:0]	src	
00000b to 11111b	#IMM:5	0 to 31

rs2[3:0]/rd[3:0]	src2/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

SMOVB SMOVB

Code Size

Syntax	Code Size (Byte)
(1) SMOVB	2

(1) SMOVB

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

SMOVF SMOVF

Code Size

Syntax	Code Size (Byte)
(1) SMOVF	2

(1) SMOVF

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

SMOVU SMOVU

Code Size

Syntax	Code Size (Byte)
(1) SMOVU	2

(1) SMOVU

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

SSTR SSTR

Code Size

Syntax	Size	Processing Size	Code Size (Byte)
(1) SSTR.size	B	B	2
	W	W	2
	L	L	2

(1) SSTR.size

b7							b0	b7						b0
0	1	1	1	1	1	1	1	1	0	0	0	1	0	sz[1:0]

sz[1:0]	Size
00b	B
01b	W
10b	L

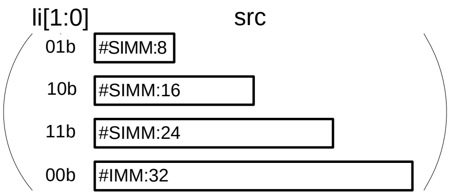
STNZ STNZ

Code Size

Syntax	src	dest	Code Size (Byte)
(1) STNZ src, dest	#SIMM:8	Rd	4
	#SIMM:16	Rd	5
	#SIMM:24	Rd	6
	#IMM:32	Rd	7
(2) STNZ src, dest	Rs	Rd	3

(1) STNZ src, dest

b7							b0	b7						b0	b7				b0
1	1	1	1	1	1	0	1	0	1	1	1	li[1:0]	0	0	1	1	1	1	rd[3:0]



li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) STNZ src, dest

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	0	0	1	0	0	1	0	1	1	rs[3:0]	rd[3:0]

rs[3:0/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

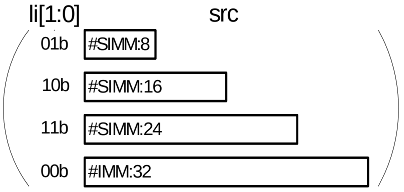
STZ STZ

Code Size

Syntax	src	dest	Code Size (Byte)
(1) STZ src, dest	#SIMM:8	Rd	4
	#SIMM:16	Rd	5
	#SIMM:24	Rd	6
	#IMM:32	Rd	7
(2) STZ src, dest	Rs	Rd	3

(1) STZ src, dest

b7							b0	b7						b0	b7				b0
1	1	1	1	1	1	0	1	0	1	1	1	li[1:0]	0	0	1	1	1	0	rd[3:0]



li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) STZ src, dest

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	0	0	1	0	0	1	0	1	1	rs[3:0]	rd[3:0]

rs[3:0/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

SUB SUB

Code Size

Syntax	src	src2	dest	Code Size (Byte)
(1) SUB src, dest	#UIMM:4	−	Rd	2
(2) SUB src, dest	Rs	−	Rd	2
	[Rs].memex	−	Rd	2 (memex == UB) 3 (memex != UB)
	dsp:8[Rs].memex	−	Rd	3 (memex == UB) 4 (memex != UB)
	dsp:16[Rs].memex	−	Rd	4 (memex == UB) 5 (memex != UB)
(3) SUB src, src2, dest	Rs	Rs2	Rd	3

(1) SUB src, dest

b7							b0	b7	b0
0	1	1	0	0	0	0	0	imm[3:0]	rd[3:0]

imm[3:0]	src	
0000b to 1111b	#UIMM:4	0 to 15

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) SUB src, dest

When memex == UB or src == Rs

b7							b0	b7	b0
0	1	0	0	0	0	ld[1:0]		rs[3:0]	rd[3:0]

ld[1:0]	src
11b	None
00b	None
01b	dsp:8
10b	dsp:16

When memex != UB

b7		memex					b0	b7							b0	b7	b0
0	0	0	0	0	1	1	0	mi[1:0]		0	0	0	0	ld[1:0]		rs[3:0]	rd[3:0]

ld[1:0]	src
11b	None
00b	None
01b	dsp:8
10b	dsp:16

mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

(3) SUB src, src2, dest

b7							b0	b7					b0	b7	b0
1	1	1	1	1	1	1	1	0	0	0	0	rd[3:0]		rs[3:0]	rs2[3:0]

rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b to 1111b	Rs/Rs2/Rd	R0 (SP) to R15

SUNTIL SUNTIL

Code Size

Syntax	Size	Processing Size	Code Size (Byte)
(1) SUNTIL.size	B	B	2
	W	W	2
	L	L	2

(1) SUNTIL.size

b7							b0	b7						b0
0	1	1	1	1	1	1	1	1	0	0	0	0	0	sz[1:0]

sz[1:0]	Size
00b	B
01b	W
10b	L

SWHILE SWHILE

Code Size

Syntax	Size	Processing Size	Code Size (Byte)
(1) SWHILE.size	B	B	2
	W	W	2
	L	L	2

(1) SWHILE.size

b7							b0	b7						b0
0	1	1	1	1	1	1	1	1	0	0	0	0	1	sz[1:0]

sz[1:0]	Size
00b	B
01b	W
10b	L

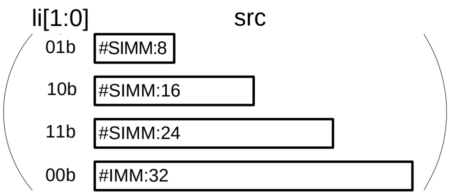
TST TST

Code Size

Syntax	src	src2	Code Size (Byte)
(1) TST src, src2	#SIMM:8	Rs	4
	#SIMM:16	Rs	5
	#SIMM:24	Rs	6
	#IMM:32	Rs	7
(2) TST src, src2	Rs	Rs2	3
	[Rs].memex	Rs2	3 (memex == UB) 4 (memex != UB)
	dsp:8[Rs].memex	Rs2	4 (memex == UB) 5 (memex != UB)
	dsp:16[Rs].memex	Rs2	5 (memex == UB) 6 (memex != UB)

(1) TST src, src2

b7							b0	b7					b0	b7					b0
1	1	1	1	1	1	0	1	0	1	1	1	li[1:0]	0	0	1	1	0	0	rs2[3:0]



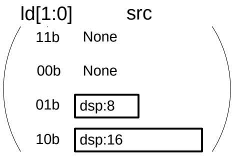
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

rs2[3:0]	src2	
0000b to 1111b	Rs	R0 (SP) to R15

(2) TST src, src2

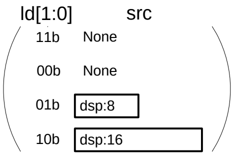
When memex == UB or src == Rs

b7							b0	b7						b0	b7	b0
1	1	1	1	1	1	0	0	0	0	1	1	0	0	ld[1:0]	rs[3:0]	rs2[3:0]



When memex != UB

b7		memex					b0	b7					b0	b7							b0	b7	b0
0	0	0	0	0	1	1	0	mi[1:0]	1	0	0	0	ld[1:0]	0	0	0	0	1	1	0	0	rs[3:0]	rs2[3:0]



mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rs2[3:0]	src/src2	
0000b to 1111b	Rs/Rs2	R0 (SP) to R15

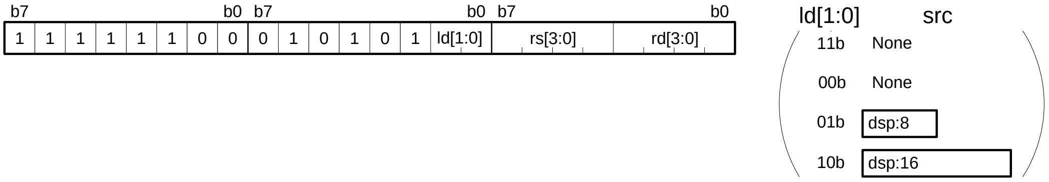
UTOF UTOF

Code Size

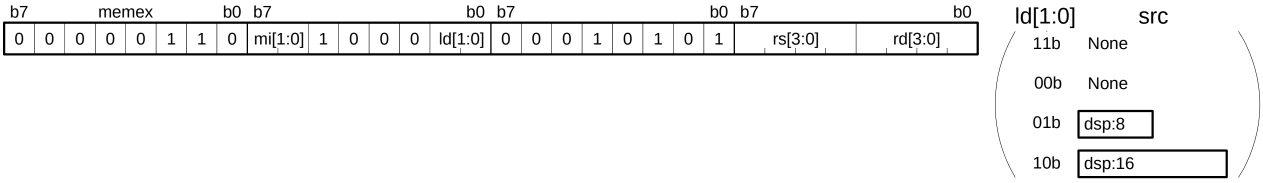
Syntax	src	dest	Code Size (Byte)
(1) UTOF src, dest	Rs	Rd	3
	[Rs].memex	Rd	3 (memex == UB) 4 (memex != UB)
	dsp:8[Rs].memex	Rd	4 (memex == UB) 5 (memex != UB)
	dsp:16[Rs].memex	Rd	5 (memex == UB) 6 (memex != UB)

(1) UTOF src, dest

When memex == UB or src == Rs



When memex != UB



mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

WAIT WAIT

Code Size

Syntax	Code Size (Byte)
(1) WAIT	2

(1) WAIT

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

XCHG XCHG

Code Size

Syntax	src	dest	Code Size (Byte)
(1) XCHG src, dest	Rs	Rd	3
	[Rs].memex	Rd	3 (memex == UB) 4 (memex != UB)
	dsp:8[Rs].memex	Rd	4 (memex == UB) 5 (memex != UB)
	dsp:16[Rs].memex	Rd	5 (memex == UB) 6 (memex != UB)

(1) XCHG src, dest

When memex == UB or src == Rs

b7							b0	b7							b0	b7		b0
1	1	1	1	1	1	0	0	0	1	0	0	0	0	ld[1:0]		rs[3:0]	rd[3:0]	

ld[1:0]	src
11b	None
00b	None
01b	dsp:8
10b	dsp:16

When memex != UB

b7			memex				b0	b7							b0	b7							b0	b7		b0
0	0	0	0	0	1	1	0	mi[1:0]		1	0	0	0	ld[1:0]		0	0	0	1	0	0	0	0	rs[3:0]	rd[3:0]	

ld[1:0]	src
11b	None
00b	None
01b	dsp:8
10b	dsp:16

mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

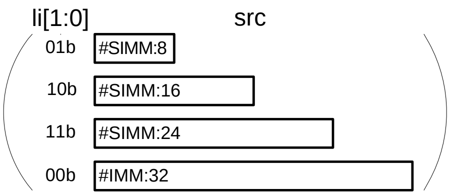
XOR XOR

Code Size

Syntax	src	dest	Code Size (Byte)
(1) XOR src, dest	#SIMM:8	Rd	4
	#SIMM:16	Rd	5
	#SIMM:24	Rd	6
	#IMM:32	Rd	7
(2) XOR src, dest	Rs	Rd	3
	[Rs].memex	Rd	3 (memex == UB) 4 (memex != UB)
	dsp:8[Rs].memex	Rd	4 (memex == UB) 5 (memex != UB)
	dsp:16[Rs].memex	Rd	5 (memex == UB) 6 (memex != UB)

(1) XOR src, dest

b7							b0	b7						b0	b7				b0
1	1	1	1	1	1	0	1	0	1	1	1	li[1:0]	0	0	1	1	0	1	rd[3:0]



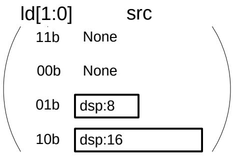
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) XOR src, dest

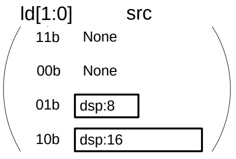
When memex == UB or src == Rs

b7							b0	b7						b0	b7	b0
1	1	1	1	1	1	0	0	0	0	1	1	0	1	ld[1:0]	rs[3:0]	rd[3:0]



When memex != UB

b7			memex				b0	b7					b0	b7							b0	b7	b0
0	0	0	0	0	1	1	0	mi[1:0]	1	0	0	0	ld[1:0]	0	0	0	0	1	1	0	1	rs[3:0]	rd[3:0]



mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

Section 5 EXCEPTIONS

5.1 Types of Exception

During the execution of a program by the CPU, the occurrence of certain events may necessitate suspending execution of the main flow of the program and starting the execution of another flow. Such events are called exceptions.

Figure 5.1 shows the types of exception.

The occurrence of an exception causes the processor mode to switch to supervisor mode.

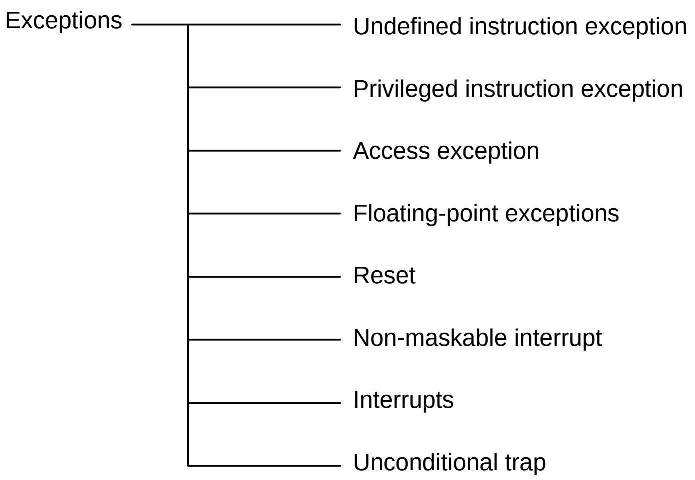

Exceptions
Undefined instruction exception
Privileged instruction exception
Access exception
Floating-point exceptions
Reset
Non-maskable interrupt
Interrupts
Unconditional trap

Figure 5.1 Types of Exception

5.1.1 Undefined Instruction Exception

An undefined instruction exception occurs when execution of an undefined instruction (an instruction not implemented) is detected.

5.1.2 Privileged Instruction Exception

A privileged instruction exception occurs when execution of a privileged instruction is detected while operation is in user mode. Privileged instructions can only be executed in supervisor mode.

5.1.3 Access Exception

When it detects an error in memory access, the CPU generates an access exception. Detection of memory protection errors for memory protection units generates exceptions of two types: instruction-access exceptions and operand-access exceptions.

5.1.4 Floating-Point Exceptions

Floating-point exceptions include the five specified in the IEEE754 standard, namely overflow, underflow, inexact, division-by-zero, and invalid operation, and a further floating-point exception that is generated on the detection of unimplemented processing. The exception processing of floating-point exceptions is masked when the EX, EU, EZ, EO, or EV bit in FPSW is 0.

5.1.5 Reset

A reset through input of the reset signal to the CPU causes the exception handling. This has the highest priority of any exception and is always accepted.

5.1.6 Non-Maskable Interrupt

The non-maskable interrupt is generated by input of the non-maskable interrupt signal to the CPU and is only used when the occurrence of a fatal fault has been detected in the system. Never end the exception handling routine for the non-maskable interrupt with an attempt to return to the program that was being executed at the time of interrupt generation.

5.1.7 Interrupts

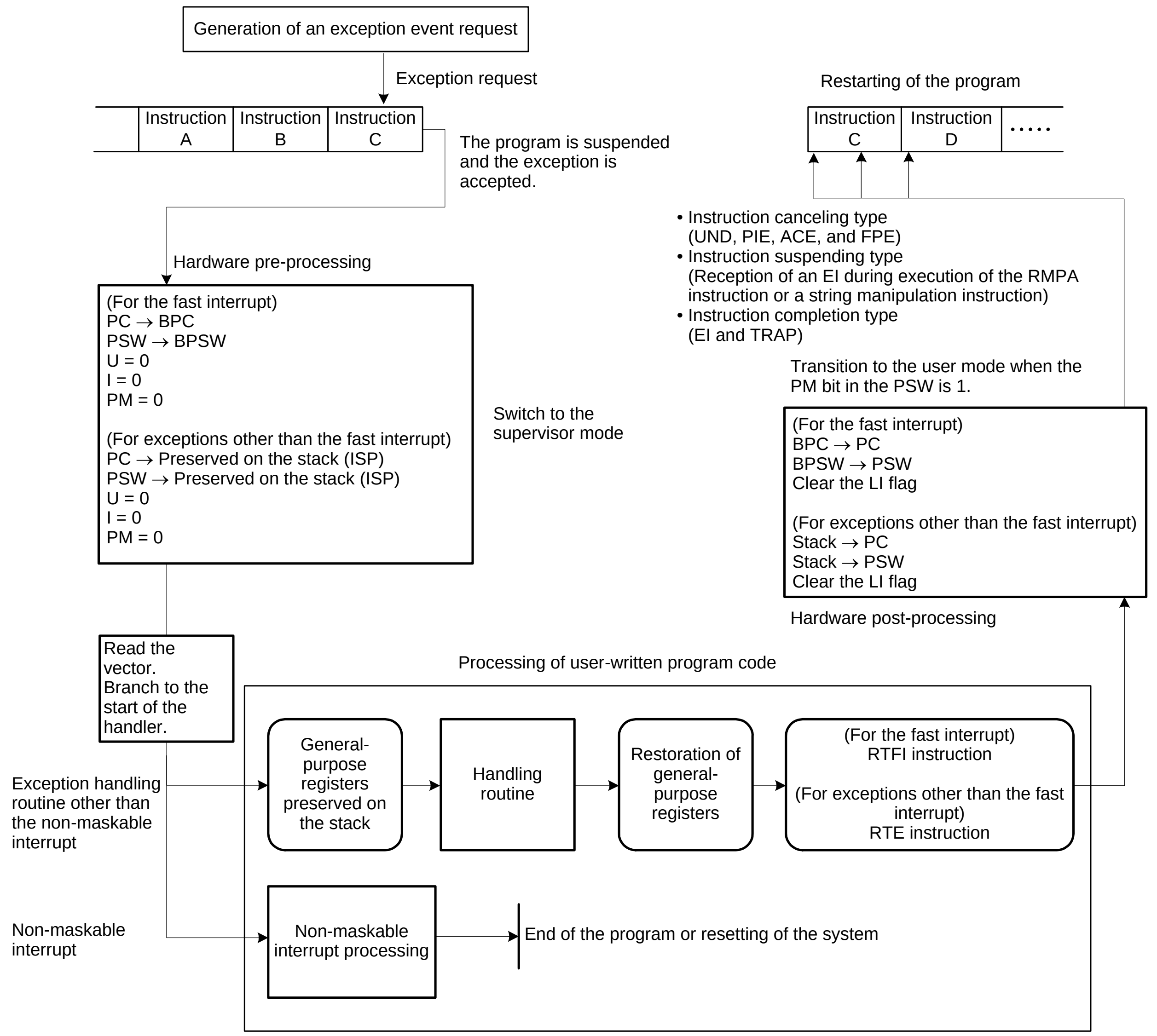
Interrupts are generated by the input of interrupt signals to the CPU. The interrupt with the highest priority can be selected for handling as a fast interrupt. In the case of the fast interrupt, hardware pre-processing and hardware post-processing are handled fast. The priority level of the fast interrupt is fifteen (the highest). The exception processing of interrupts is masked when the I bit in PSW is 0.

5.1.8 Unconditional Trap

An unconditional trap is generated when the INT or BRK instruction is executed.

5.2 Exception Handling Procedure

For exception handling, part of the processing is handled automatically by hardware and part is handled by a program (the exception handling routine) that has been written by the user. Figure 5.2 shows the handling procedure when an exception other than a reset is accepted.


Generation of an exception event request
Exception request
Instruction A
Instruction B
Instruction C
The program is suspended and the exception is accepted.
Restarting of the program
Instruction C
Instruction D
.....
• Instruction canceling type (UND, PIE, ACE, and FPE)
• Instruction suspending type (Reception of an EI during execution of the RMPA instruction or a string manipulation instruction)
• Instruction completion type (EI and TRAP)
Hardware pre-processing
(For the fast interrupt)
PC → BPC
PSW → BPSW
U = 0
I = 0
PM = 0
(For exceptions other than the fast interrupt)
PC → Preserved on the stack (ISP)
PSW → Preserved on the stack (ISP)
U = 0
I = 0
PM = 0
Switch to the supervisor mode
Transition to the user mode when the PM bit in the PSW is 1.
(For the fast interrupt)
BPC → PC
BPSW → PSW
Clear the LI flag
(For exceptions other than the fast interrupt)
Stack → PC
Stack → PSW
Clear the LI flag
Hardware post-processing
Read the vector. Branch to the start of the handler.
Processing of user-written program code
Exception handling routine other than the non-maskable interrupt
General-purpose registers preserved on the stack
Handling routine
Restoration of general-purpose registers
(For the fast interrupt)
RTFI instruction
(For exceptions other than the fast interrupt)
RTE instruction
Non-maskable interrupt
Non-maskable interrupt processing
End of the program or resetting of the system

Legend

UND:	Undefined instruction exception
PIE:	Privileged instruction exception
ACE:	Access exception
FPE:	Floating-point exception
EI:	Interrupt
TRAP:	Unconditional trap

Figure 5.2 Outline of the Exception Handling Procedure

When an exception is accepted, hardware processing by the RXv2 CPU is followed by vector table access to acquire the address of the branch destination. A vector address is allocated to each exception. The branch destination address of the exception handling routine for the given exception is written to each vector address.

Hardware pre-processing by the RXv2 CPU handles saving of the contents of the program counter (PC) and processor status word (PSW). In the case of the fast interrupt, the contents are saved in the backup PC (BPC) and the backup PSW (BPSW), respectively. In the case of other exceptions, the contents are preserved in the stack area. General purpose registers and control registers other than the PC and PSW that are to be used within the exception handling routine must be preserved on the stack by user program code at the start of the exception handling routine.

On completion of processing by most exception handling routine, registers preserved under program control are restored and the RTE instruction is executed to restore execution from the exception handling routine to the original program. For return from the fast interrupt, the RTFI instruction is used instead. In the case of the non-maskable interrupt, however, end the program or reset the system without returning to the original program.

Hardware post-processing by the RXv2 CPU handles restoration of the pre-exception contents of the PC and PSW. In the case of the fast interrupt, the contents of the BPC and BPSW are restored to the PC and PSW, respectively. In the case of other exceptions, the contents are restored from the stack area to the PC and PSW.

5.3 Acceptance of Exceptions

When an exception occurs, the CPU suspends the execution of the program and processing branches to the start of the exception handling routine.

5.3.1 Timing of Acceptance and Saved PC Value

Table 5.1 lists the timing of acceptance and program counter (PC) value to be saved for each type of exception event.

Table 5.1 Timing of Acceptance and Saved PC Value

Exception		Type of Handling	Timing of Acceptance	Value Saved in the BPC/ on the Stack
Undefined instruction exception		Instruction canceling type	During instruction execution	PC value of the instruction that is generated by the exception
Privileged instruction exception		Instruction canceling type	During instruction execution	PC value of the instruction that is generated by the exception
Access exception		Instruction canceling type	During instruction execution	PC value of the instruction that is generated by the exception
Floating-point exceptions		Instruction canceling type	During instruction execution	PC value of the instruction that is generated by the exception
Reset		Program abandonment type	Any machine cycle	None
Non-maskable interrupt	During execution of the RMPA, SCMPU, SMOVB, SMOVF, SMOVU, SSTR, SUNTIL, and SWHILE instructions	Instruction suspending type	During instruction execution	PC value of the instruction being executed
	Other than the above	Instruction completion type	At the next break between instructions	PC value of the next instruction
Interrupts	During execution of the RMPA, SCMPU, SMOVB, SMOVF, SMOVU, SSTR, SUNTIL, and SWHILE instructions	Instruction suspending type	During instruction execution	PC value of the instruction being executed
	Other than the above	Instruction completion type	At the next break between instructions	PC value of the next instruction
Unconditional trap		Instruction completion type	At the next break between instructions	PC value of the next instruction

5.3.2 Vector and Site for Preserving the PC and PSW

The vector for each type of exception and the site for preserving the contents of the program counter (PC) and processor status word (PSW) are listed in table 5.2.

Table 5.2 Vector and Site for Preserving the PC and PSW

Exception		Vector	Site for Preserving the PC and PSW
Undefined instruction exception		Exception vector table	Stack
Privileged instruction exception		Exception vector table	Stack
Access exception		Exception vector table	Stack
Floating-point exceptions		Exception vector table	Stack
Reset		Exception vector table	Nowhere
Non-maskable interrupt		Exception vector table	Stack
Interrupts	Fast interrupt	FINTV	BPC and BPSW
	Other than the above	Interrupt vector table	Stack
Unconditional trap		Interrupt vector table	Stack

5.4 Hardware Processing for Accepting and Returning from Exceptions

This section describes the hardware processing for accepting and returning from an exception other than a reset.

(1) Hardware pre-processing for accepting an exception

(a) Preserving the PSW

(For the fast interrupt)

PSW → BPSW

(For other exceptions)

PSW → Stack area

Note: The FPSW is not preserved by hardware pre-processing. Therefore, if this is used within the exception handling routine for floating-point instructions, the user should ensure that it is preserved in the stack area from within the exception handling routine.

(b) Updating of the PM, U, and I bits in the PSW

I: Cleared to 0

U: Cleared to 0

PM: Cleared to 0

(c) Preserving the PC

(For the fast interrupt)

PC → BPC

(For other exceptions)

PC → Stack area

(d) Set the branch-destination address of the exception handling routine in the PC

Processing is shifted to the exception handling routine by acquiring the vector corresponding to the exception and branching accordingly.

(2) Hardware post-processing for executing RTE and RTFI instructions

(a) Restoring the PSW

(For the fast interrupt)

BPSW → PSW

(For other exceptions)

Stack area → PSW

(b) Restoring the PC

(For the fast interrupt)

BPC → PC

(For other exceptions)

Stack area → PC

(c) Clearing the LI flag

5.5 Hardware Pre-processing

The sequences of hardware pre-processing from reception of each exception request to execution of the associated exception handling routine are explained below.

5.5.1 Undefined Instruction Exception

(1) The value of the processor status word (PSW) is saved on the stack (ISP).

(2) The processor mode select bit (PM), the stack pointer select bit (U), and the interrupt enable bit (I) in the PSW are cleared to 0.

(3) The value of the program counter (PC) is saved on the stack (ISP).

(4) The vector is fetched from the value of EXTB + address 0000 005Ch.

(5) The PC is set to the fetched address and processing branches to the start of the exception handling routine.

5.5.2 Privileged Instruction Exception

(1) The value of the processor status word (PSW) is saved on the stack (ISP).

(2) The processor mode select bit (PM), the stack pointer select bit (U), and the interrupt enable bit (I) in the PSW are cleared to 0.

(3) The value of the program counter (PC) is saved on the stack (ISP).

(4) The vector is fetched from the value of EXTB + address 0000 0050h.

(5) The PC is set to the fetched address and processing branches to the start of the exception handling routine.

5.5.3 Access Exception

(1) The value of the processor status word (PSW) is saved on the stack (ISP).

(2) The processor mode select bit (PM), the stack pointer select bit (U), and the interrupt enable bit (I) in the PSW are cleared to 0.

(3) The value of the program counter (PC) is saved on the stack (ISP).

(4) The vector is fetched from the value of EXTB + address 0000 0054h.

(5) The PC is set to the fetched address and processing branches to the start of the exception handling routine.

5.5.4 Floating-Point Exceptions

(1) The value of the processor status word (PSW) is saved on the stack (ISP).

(2) The processor mode select bit (PM), the stack pointer select bit (U), and the interrupt enable bit (I) in the PSW are cleared to 0.

(3) The value of the program counter (PC) is saved on the stack (ISP).

(4) The vector is fetched from the value of EXTB + address 0000 0064h.

(5) The PC is set to the fetched address and processing branches to the start of the exception handling routine.

5.5.5 Reset

(1) The control registers are initialized.

(2) The address of the processing routine is fetched from the vector address, FFFFFFFCh.

(3) The PC is set to the fetched address.

5.5.6 Non-Maskable Interrupt

(1) The value of the processor status word (PSW) is saved on the stack (ISP).

(2) The processor mode select bit (PM), the stack pointer select bit (U), and the interrupt enable bit (I) in the PSW are cleared to 0.

(3) If the interrupt was generated during the execution of an RMPA, SCMPU, SMOVB, SMOVF, SMOVU, SSTR, SUNTIL, or SWHILE instruction, the value of the program counter (PC) for that instruction is saved on the stack (ISP). For other instructions, the PC value of the next instruction is saved.

(4) The processor interrupt priority level bits (IPL[3:0]) in the PSW are set to Fh.

(5) The vector is fetched from the value of EXTB + address 0000 0078h.

(6) The PC is set to the fetched address and processing branches to the start of the exception handling routine.

5.5.7 Interrupts

(1) The value of the processor status word (PSW) is saved on the stack (ISP) or, for the fast interrupt, in the backup PSW (BPSW).

(2) The processor mode select bit (PM), the stack pointer select bit (U), and the interrupt enable bit (I) in the PSW are cleared to 0.

(3) If the interrupt was generated during the execution of an RMPA, SCMPU, SMOVB, SMOVF, SMOVU, SSTR, SUNTIL, or SWHILE instruction, the value of the program counter (PC) for that instruction is saved. For other instructions, the PC value of the next instruction is saved. Saving of the PC is in the backup PC (BPC) for fast interrupts and on the stack for other interrupts.

(4) The processor interrupt priority level bits (IPL[3:0]) in the PSW indicate the interrupt priority level of the interrupt.

(5) The vector for an interrupt source other than the fast interrupt is fetched from the interrupt vector table. For the fast interrupt, the address is fetched from the fast interrupt vector register (FINTV).

(6) The PC is set to the fetched address and processing branches to the start of the exception handling routine.

5.5.8 Unconditional Trap

(1) The value of the processor status word (PSW) is saved on the stack (ISP).

(2) The processor mode select bit (PM), the stack pointer select bit (U), and the interrupt enable bit (I) in the PSW are cleared to 0.

(3) The value of the program counter (PC) is saved on the stack (ISP).

(4) For the INT instruction, the value at the vector corresponding to the INT instruction number is fetched from the interrupt vector table.
For the BRK instruction, the value at the vector from the start address is fetched from the interrupt vector table.

(5) The PC is set to the fetched address and processing branches to the start of the exception handling routine.

5.6 Return from Exception Handling Routines

Executing the instructions listed in table 5.3 at the end of the corresponding exception handling routines restores the values of the program counter (PC) and processor status word (PSW) that were saved on the stack or in control registers (BPC and BPSW) immediately before the exception handling sequence.

Table 5.3 Return from Exception Handling Routines

Exception		Instruction for Return
Undefined instruction exception		RTE
Privileged instruction exception		RTE
Access exception		RTE
Floating-point exceptions*		RTE
Reset		Return is impossible
Non-maskable interrupt		Disabled
Interrupts	Fast interrupt	RTFI
	Other than the above	RTE
Unconditional trap		RTE

5.7 Order of Priority for Exceptions

The order of priority for exceptions is given in table 5.4. When multiple exceptions are generated at the same time, the exception with the highest priority is accepted first.

Table 5.4 Order of Priority for Exceptions

Order of Priority		Exception
High	1	Reset
↑	2	Non-maskable interrupt
	3	Interrupts
	4	Instruction access exception
	5	Undefined instruction exception Privileged instruction exception
	6	Unconditional trap
	7	Operand access exception
Low	8	Floating-point exceptions*

Index

REVISION HISTORY RX Family RXv2 Instruction Set Architecture Software

		Description	
Rev.	**Date**	**Page**	**Summary**
1.00	Nov 12, 2013	-	First edition issued

RX Family RXv2 Instruction Set Architecture User’s Manual: Software

Publication Date: Rev.1.00 Nov 12, 2013

Published by: Renesas Electronics Corporation

SALES OFFICES

Renesas Electronics Corporation

http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700, Fax: +44-1628-651-804

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 1.3

RX Family RXv2 Instruction Set Architecture

RENESAS

Renesas Electronics Corporation