User's Manual

32

RX Family RXv1 Instruction Set Architecture

User's Manual: Software

RENESAS 32-Bit MCU
RX Family

Renesas Electronics
www.renesas.com

Rev.1.30 Dec 2019

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 or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving 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, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. 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 shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended 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; industrial robots; etc.
   “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.
   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or 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 damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, 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 and impractical, you are responsible for evaluating 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. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall 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 shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, 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.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

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

(Rev.4.0-1 November 2017)

How to Use This Manual

This manual is designed to provide users with an understanding of RXv1 instruction set architecture (RXv1). The manual contains detailed descriptions of CPU features and instruction sets. The manual is intended for users who are designing application systems using this CPU. Target users are expected to understand the fundamentals of microcomputers.

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, src2	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, 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.
	ACC	Accumulator
	tmp, tmp0, tmp1, tmp2, tmp3 etc.	Temporary registers
Values	000b	Binary number
	0000h	Hexadecimal number
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.

Classification	Notation	Meaning
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.
Operations	(Operations in this manual are written in accordance 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.

Classification	Notation	Meaning
Operations	{ }	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.
	==, !=	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.
Floating-point datum	NaN	Not a number
Floating-point representation	SNaN	Signaling NaN
	QNaN	Quiet NaN

Contents

List of RXv1 Instruction Set 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	51	169
ADC		Add with carry	52	170
ADD		Add without carry	53	171
AND		Logical AND	55	173
BCLR		Clear a bit	57	175
B*Cnd*	BGEU	Conditional relative branch	58	177
	BC		58	177
	BEQ		58	177
	BZ		58	177
	BGTU		58	177
	BPZ		58	177
	BGE		58	177
	BGT		58	177
	BO		58	177
	BLTU		58	177
	BNC		58	177
	BNE		58	177
	BNZ		58	177
	BLEU		58	177
	BN		58	177
	BLE		58	177
	BLT		58	177
	BNO		58	177
BM*Cnd*	BMGEU	Conditional bit transfer	59	179
	BMC		59	179
	BMEQ		59	179
	BMZ		59	179
	BMGTU		59	179
	BMPZ		59	179
	BMGE		59	179
	BMGT		59	179
	BMO		59	179
	BMLTU		59	179
	BMNC		59	179
	BMNE		59	179
	BMNZ		59	179
	BMLEU		59	179
	BMN		59	179
	BMLE		59	179
	BMLT		59	179
	BMNO		59	179
BNOT		Not a bit	61	180
BRA		Unconditional relative branch	62	181

Quick Page Reference in Alphabetical Order (2 / 4)

Mnemonic	Function	Instruction Described in Detail (on Page)	Instruction Code Described in Detail (on Page)
BRK	Unconditional trap	63	182
BSET	Set a bit	64	182
BSR	Relative branch to subroutine	65	184
BTST	Test a bit	66	185
CLRPSW	Clear a flag or bit in the PSW	67	186
CMP	Compare	68	187
DIV	Divide signed	69	188
DIVU	Divide unsigned	70	190
EMUL	Extended multiply signed	71	191
EMULU	Extended multiply unsigned	73	192
FADD*1	Add floating-point	75	194
FCMP*1	Comparefloating-point	77	195
FDIV*1	Divide floating-point	79	196
FMUL*1	Multiply floating-point	81	197
FSUB*1	Subtractfloating-point	83	198
FTOI*1	Convert floating-point to signed integer	85	199
INT	Software interrupt	88	200
ITOF*1	Convert signed integer to floating-point	89	200
JMP	Unconditional jump	91	201
JSR	Jump to subroutine	92	201
MACHI	Multiply-Accumulate the upper words	93	202
MACLO	Multiply-Accumulate the lower words	94	202
MAX	Maximum of two signed integers	95	203
MIN	Minimum of two signed integers	96	204
MOV	Move	97	205
MOVU	Move unsigned	100	210
MUL	Multiply	102	211
MULHI	Multiply the upper words	104	213
MULLO	Multiply the lower words	105	213
MVFACHI	Move data from the upper longword of the accumulator	106	213
MVFACMI	Move data from the middle-order longword of the accumulator	107	214
MVFC	Move data from a control register	108	214
MVTACHI	Move data to the upper longword of the accumulator	109	215
MVTACLO	Move data to the lower longword of the accumulator	110	215
MVTC	Move data to a control register	111	216
MVTIPL (privileged instruction)*2	Move data to IPL	112	217
NEG	Negate (two's complement)	113	218
NOP	No operation	114	218
NOT	Logical NOT (one's complement)	115	219
OR	Logical OR	116	220
POP	Pop register from stack	117	221
POPC	Pop a control register from stack	118	222
POPM	Pop multiple registers from stack	119	222
PUSH	Push register on stack	120	223

Quick Page Reference in Alphabetical Order (3 / 4)

Mnemonic		Function	Instruction Described in Detail (on Page)	Instruction Code Described in Detail (on Page)
PUSHC		Push a control register on stack	121	224
PUSHM		Push multiple registers on stack	122	224
RACW		Round the accumulator word	123	225
REVL		Reverse endian within longword	125	225
REVW		Reverse endian within word	126	225
RMPA		Repeat multiply-accumulate	127	226
ROLC		Rotate left with carry	129	226
RORC		Rotate right with carry	130	227
ROTL		Rotate left	131	227
ROTR		Rotate right	132	228
ROUND[*1]		Round floating-point to signed integer	133	229
RTE (privileged instruction)		Return from exception	136	229
RTFI (privileged instruction)		Return from fast interrupt	137	230
RTS		Return from subroutine	138	230
RTSD		Return from subroutine after deallocating stack frame	139	230
SAT		Saturate	141	231
SATR		Saturate for RMPA	142	231
SBB		Subtract with borrow	143	232
SC*Cnd*	SCGEU	Store condition	144	233
	SCC		144	233
	SCEQ		144	233
	SCZ		144	233
	SCGTU		144	233
	SCPZ		144	233
	SCGE		144	233
	SCGT		144	233
	SCO		144	233
	SCLTU		144	233
	SCNC		144	233
	SCNE		144	233
	SCNZ		144	233
	SCLEU		144	233
	SCN		144	233
	SCLE		144	233
	SCLT		144	233
	SCNO		144	233
SCMPU		String compare until not equal	145	233
SETPSW		Set a flag or bit in the PSW	146	234
SHAR		Arithmetic shift right	147	235
SHLL		Logical shift left	148	236
SHLR		Logical shift right	149	237
SMOVB		String move backward	150	238
SMOVF		String move forward	151	238
SMOVU		String move until zero detected	152	238

Quick Page Reference in Alphabetical Order (4 / 4)

Mnemonic	Function	Instruction Described in Detail (on Page)	Instruction Code Described in Detail (on Page)
SSTR	String store	153	239
STNZ	Store on not zero	154	239
STZ	Store on zero	155	240
SUB	Subtract without borrow	156	241
SUNTIL	String search until equal	157	242
SWHILE	String search while equal	159	242
TST	Test logical	161	243
WAIT (privileged instruction)	Wait	162	244
XCHG	Exchange	163	244
XOR	Logical Exclusive OR	164	245

Note: 1. The floating-point arithmetic instructions are optional functions. Whether or not the product has the floating-point arithmetic instructions will depend on the product. For details, refer to the user's manual: hardware for each product.

Note: 2. Products of the RX610 Group do not support the MVTIPL instruction.

1. CPU Programming Model

The RXv1 instruction set architecture (RXv1) has the following features.

- Adoption of variable-length instruction format

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

  The RXv1 supports 90 selected instructions. DSP instructions and floating-point arithmetic instructions realize high-speed arithmetic processing.
- Versatile addressing modes

  The RXv1 CPU has 10 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: Nine 32-bit registers

  Accumulator: One 64-bit register
- Variable-length instruction format (lengths from one to eight bytes)
- 90 instructions/10 addressing modes

  Basic instructions: 73

  Floating-point arithmetic instructions: 8 (as an optional function*)

  DSP instructions: 9
- Processor modes

  Supervisor mode and user mode
- Vector tables

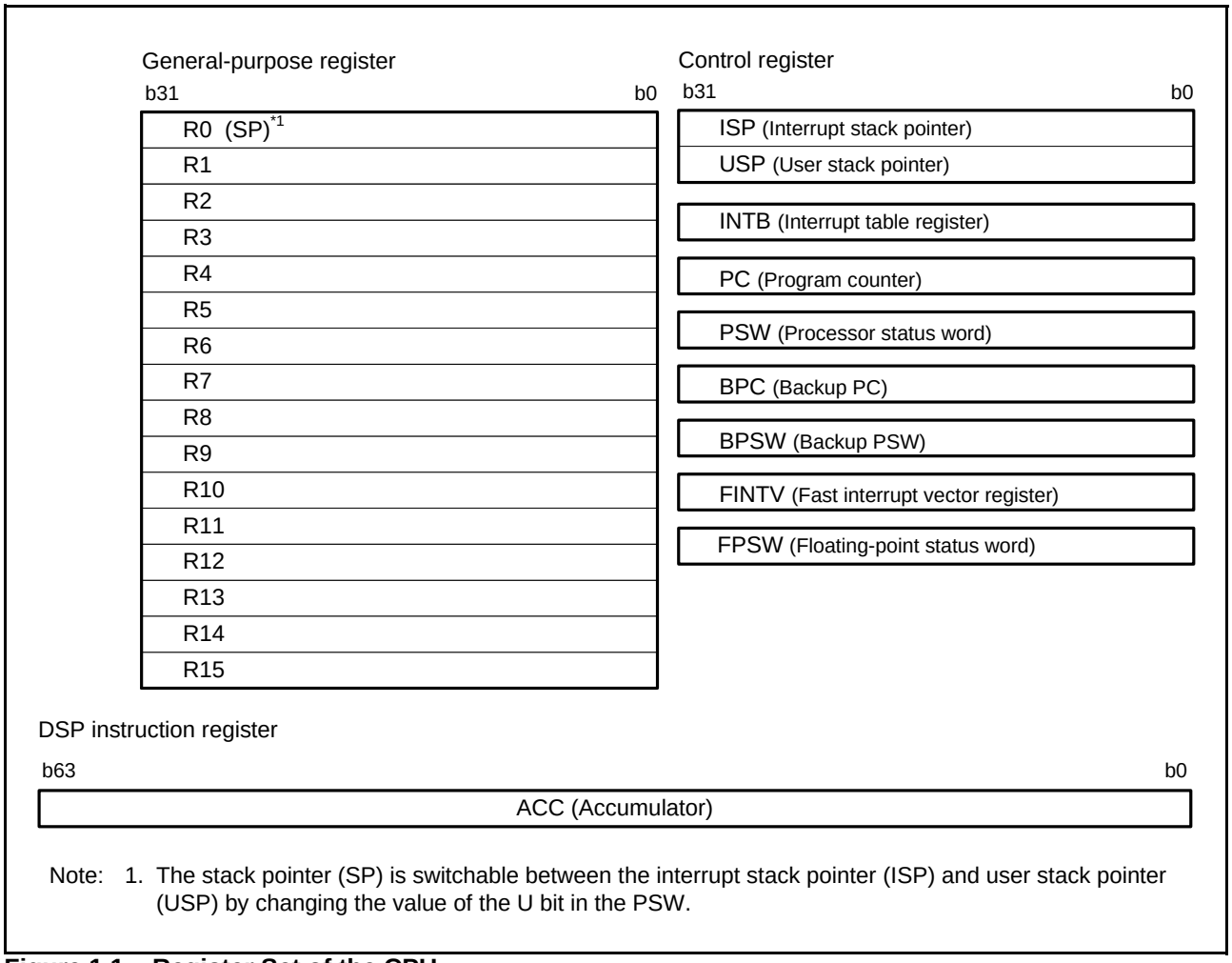
  Fixed vector table and relocatable vector table
- Memory protection unit (as an optional function)
- Data arrangement

  Selectable as little endian or big endian

Note: * The floating-point arithmetic instructions are optional functions. Whether or not the product has the floating-point arithmetic instructions will depend on the product. For details, refer to the user's manual: hardware for each product.

1.2 Register Set of the CPU

The RXv1 CPU has sixteen general-purpose registers, nine control registers, and one accumulator used for DSP instructions.



Note: 1. The stack pointer (SP) is switchable between the interrupt stack pointer (ISP) and user stack pointer (USP) by changing 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). R0 to R15 can be used as data register or address register.

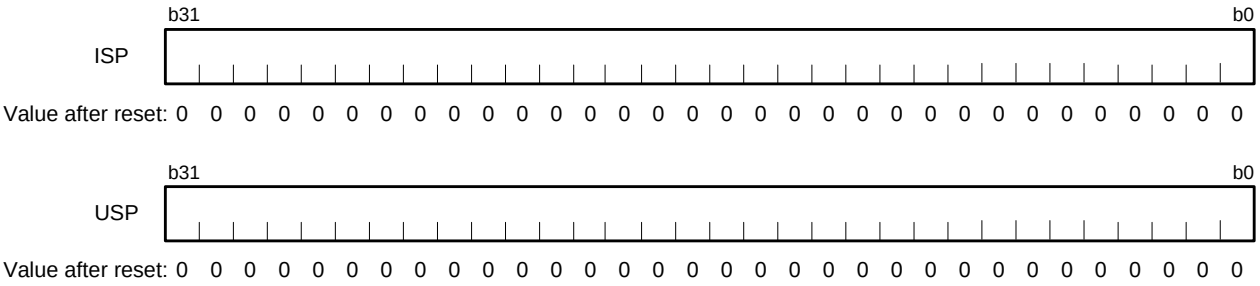
R0, a general-purpose register, also functions as the stack pointer (SP). The stack pointer is switched to operate as the interrupt stack pointer (ISP) or user stack pointer (USP) by the value of the stack pointer select bit (U) in the processor status word (PSW).

1.2.2 Control Registers

This CPU has the following nine control registers.

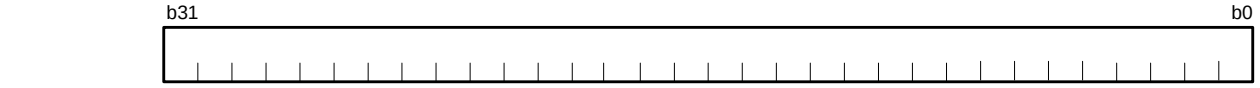
- Interrupt stack pointer (ISP)
- User stack pointer (USP)
- Interrupt table register (INTB)
- Program counter (PC)
- Processor status word (PSW)
- Backup PC (BPC)
- Backup PSW (BPSW)
- Fast interrupt vector register (FINTV)
- Floating-point status word (FPSW)

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



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).

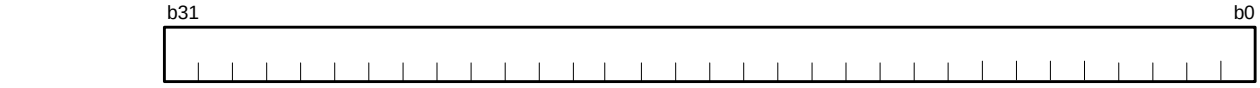
1.2.2.2 Interrupt Table Register (INTB)



Value after reset: Undefined

The interrupt table register (INTB) specifies the address where the relocatable 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, *4	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

Note: 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.

Note: 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.

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

Note: 4. Bit 27, the IPL[3] bit, is reserved in products of the RX610 group. Writing to this bit is ignored. The bit is read as 0.

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)

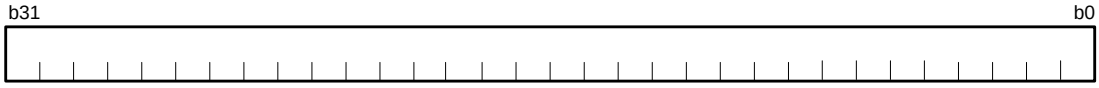
The IPL[3:0] bits specify the processor interrupt priority level as one of sixteen levels from zero to fifteen, where priority level zero is the lowest and priority level fifteen the highest. When the priority level of a requested interrupt is higher than the processor interrupt priority level, the interrupt 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.

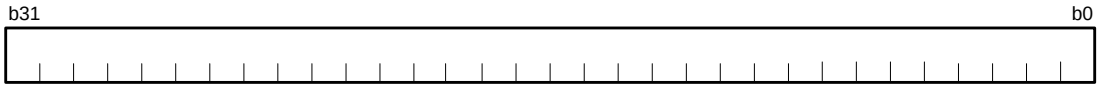
1.2.2.6 Backup PSW (BPSW)



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
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

Bit	Symbol	Bit Name	Description	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

Note: 1. When 0 is written to the bit, the setting of the bit will be 0; the bit retains the previous value in response to the writing of 1.

Note: 2. Positive denormalized numbers are treated as +0, negative denormalized numbers as –0.

Note: 3. When the EV bit is set to 0, the FV flag is enabled.

Note: 4. When the EO bit is set to 0, the FO flag is enabled.

Note: 5. When the EZ bit is set to 0, the FZ flag is enabled.

Note: 6. When the EU bit is set to 0, the FU flag is enabled.

Note: 7. When the EX bit is set to 0, the FX flag is enabled.

Note: 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 arithmetic operations. In products that do not support the floating-point instruction, 00000000h is read and the writing is ignored.

When the corresponding exception handling enable bits (Ej) are set to enable processing of the exceptions (Ej = 1), the Cj flags can be used by the exception handling routine to identify the source of that exception. If handling of an exception is masked (Ej = 0), the Fj flag can be used to check for the generation of the exception at the end of a sequence of processing. The Fj flags operate in an accumulative fashion (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 of a hypothetical calculation with infinite precision. 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.

- If an exception or processing that is not implemented is not encountered in the execution of a floating-point arithmetic instruction, the corresponding flags become 0.
- 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 floating-point arithmetic 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)

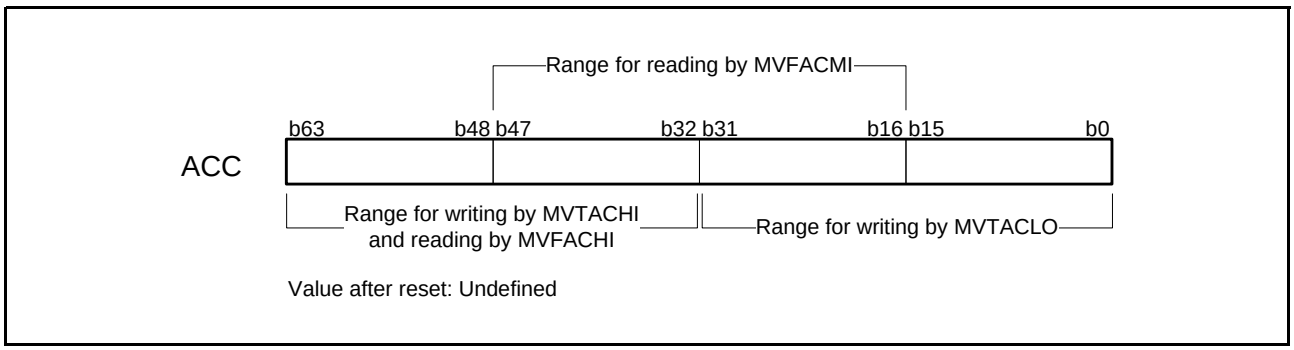
This bit reflects the logical OR of the FU, FZ, FO, and FV flags.

1.2.3 Accumulator

The accumulator (ACC) is a 64-bit register used for DSP instructions. ACC is also used for the multiply and multiply-and-accumulate instructions; EMUL, EMULU, FMUL, MUL, and RMPA, in which case the prior value in ACC is modified by execution of the instruction.

Use the MVTACHI and MVTACLO instructions for writing to the accumulator. The MVTACHI and MVTACLO instructions write data to the upper 32 bits (bits 63 to 32) and the lower 32 bits (bits 31 to 0), respectively.

Use the MVFACHI and MVFACMI instructions for reading data from the accumulator. The MVFACHI and MVFACMI instructions read data from the upper 32 bits (bits 63 to 32) and the middle 32 bits (bits 47 to 16), respectively.



1.3 Floating-Point Exceptions

Floating-point exceptions are generated when any of the five exceptions specified in the IEEE754 standard, namely overflow, underflow, inexact, division-by-zero, or invalid operation, or an attempts to use processing that is not implemented, is detected upon execution of a floating-point arithmetic instruction. Exception handling by the CPU only proceeds when any among the EX, EU, EZ, EO, or EV bits in the FPSW, which corresponding to the five types of exception, is set to 1.

The following is an outline of the events that cause floating-point exceptions.

Note: Floating-point exceptions do not occur on the products which do not support the floating-point arithmetic instructions.

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	

Note: 1. An inexact exception will not be generated when an underflow error occurs.

Note: 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. However, if the dividend is one of those listed in Table 1.5, the operation is not treated as division by zero.

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

Table 1.5 Dividends and Operations that are not Treated as Division by Zero

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 1.6 lists the conditions leading to an invalid exception and the results of operations.

Table 1.6 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
$+\infty + (-\infty)$, $+\infty - (+\infty)$, $-\infty - (-\infty)$		
$0 \times \infty$		
$0 \div 0$, $\infty \div \infty$		
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.	
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 fraction 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 fraction 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 1.7 lists the rules for generating QNaNs as the results of operations.

Table 1.7 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 fraction 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 RXv1 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 section 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)

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 PSW that is saved on the stack.

(2) Switching from supervisor mode to user mode
Executing an RTE instruction when the value of the PM bit in the PSW that has been saved on the stack is “1” or an RTFI instruction when the value of the PM bit in the PSW that has been saved 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 RXv1 CPU can handle four types of data: integer, floating-point number, 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.

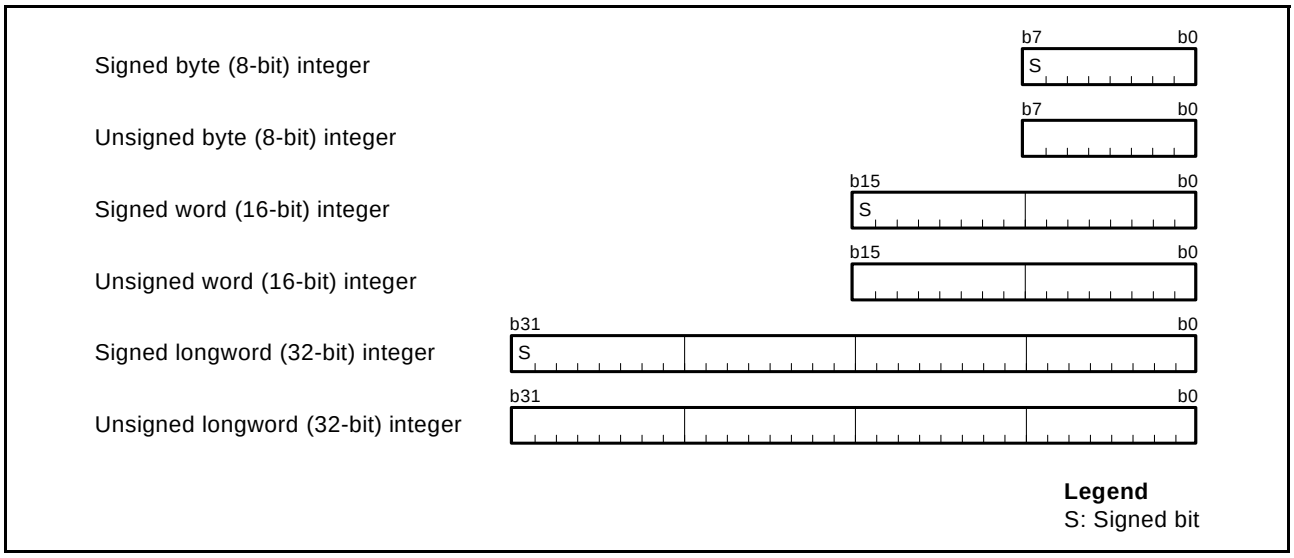


Figure 1.2 Integer

1.5.2 Floating-Point Number

The floating-point number is compliant with that specified in the IEEE754 standard; operands of this type can be used in eight floating-point arithmetic instructions: FADD, FCMP, FDIV, FMUL, FSUB, FTOI, ITOF, and ROUND.

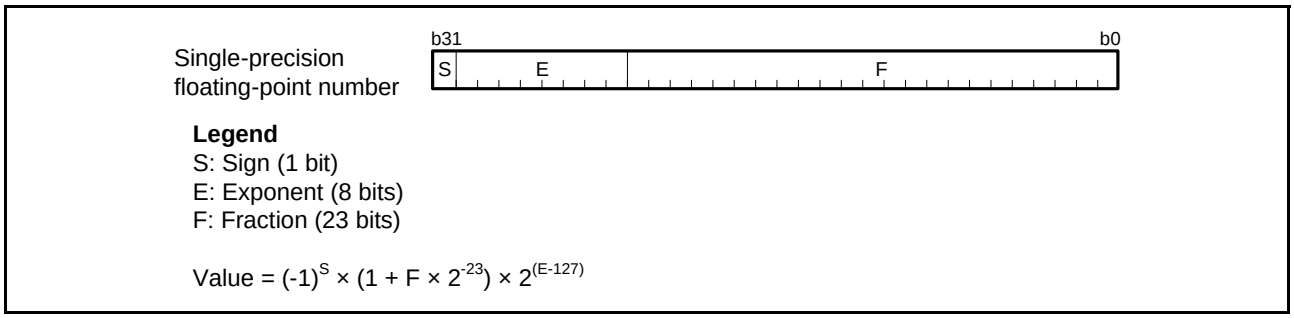


Figure 1.3 Floating-Point Number

The floating-point number can represent 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 Bit

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.

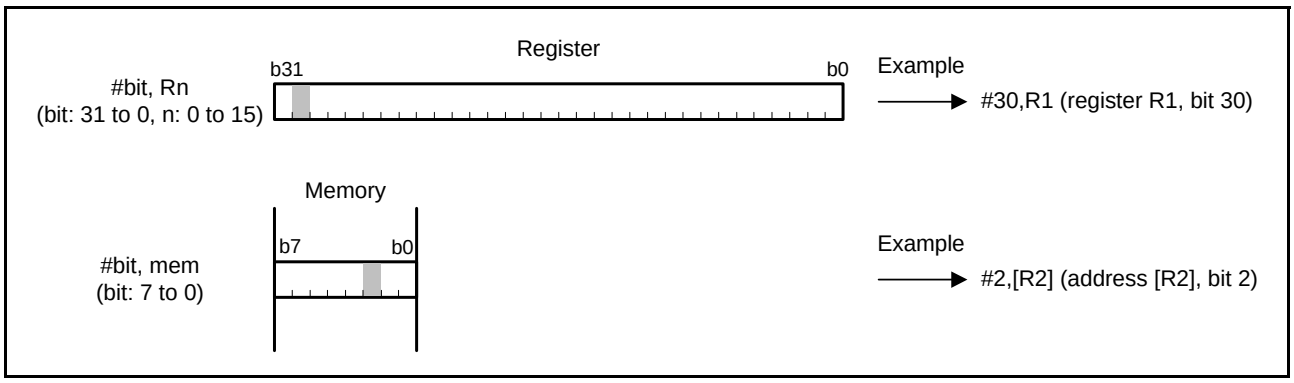


Figure 1.4 Bit

1.5.4 String

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.

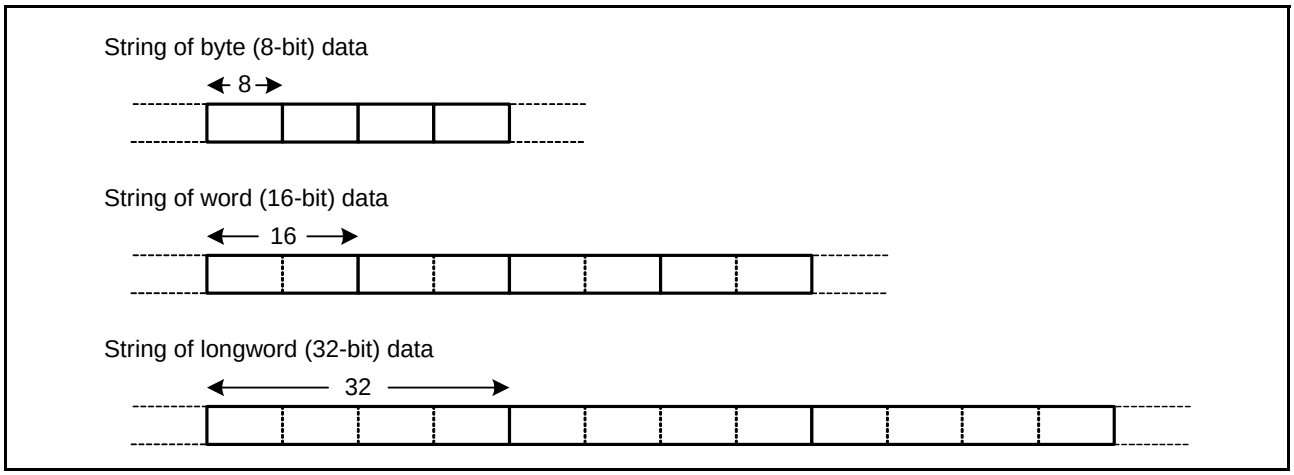


Figure 1.5 String

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.

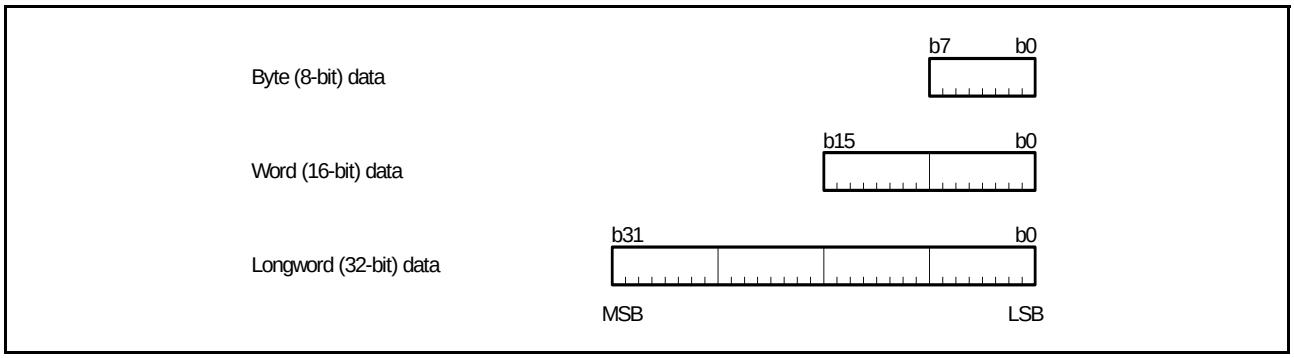


Figure 1.6 Data Arrangement in Registers

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.

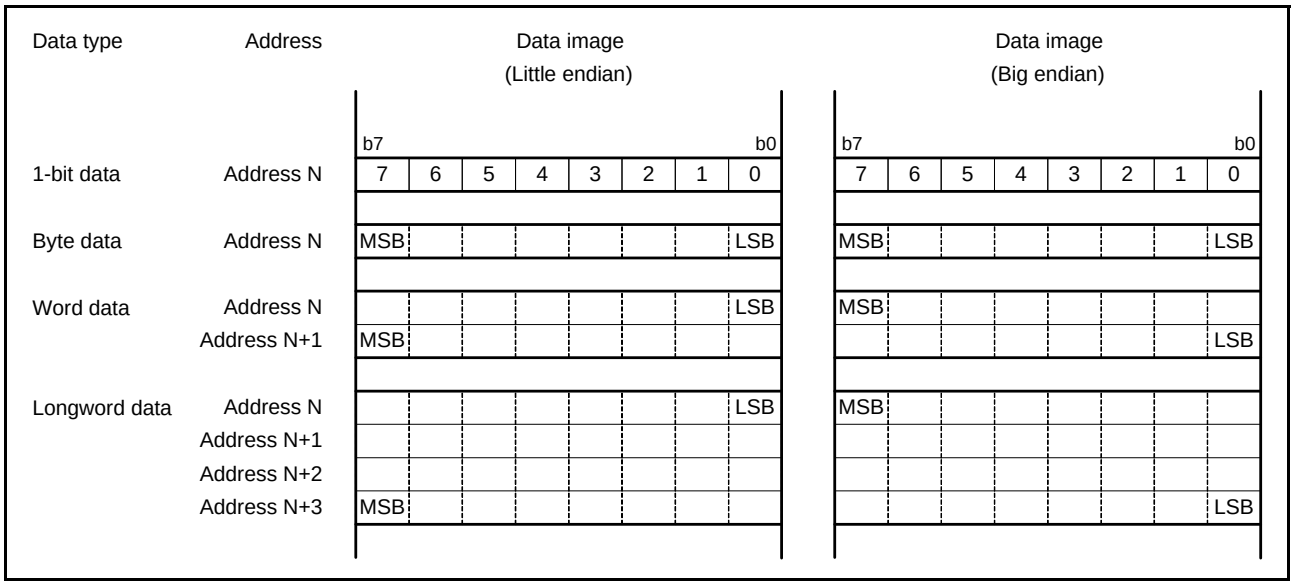


Figure 1.7 Data Arrangement in Memory

1.7 Vector Table

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

1.7.1 Fixed Vector Table

The fixed vector table is allocated to a fixed address range. The individual vectors for the privileged instruction exception, access exception, undefined instruction exception, floating-point exception, non-maskable interrupt, and reset are allocated to addresses in the range from FFFFFF80h to FFFFFFFFh. Figure 1.8 shows the fixed vector table.

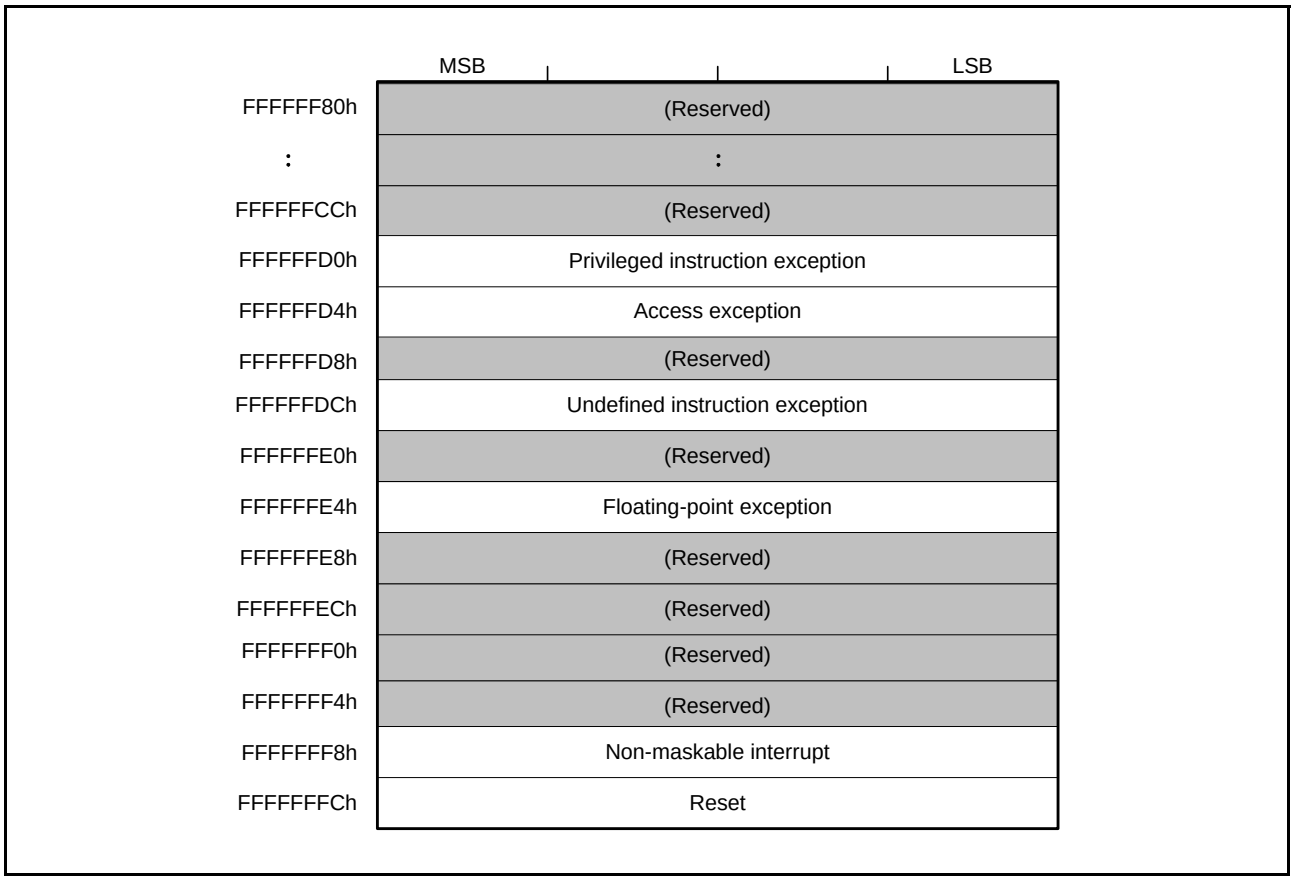


Figure 1.8 Fixed Vector Table

1.7.2 Relocatable Vector Table

The address where the relocatable 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 relocatable vector table.

Each vector in the relocatable 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.

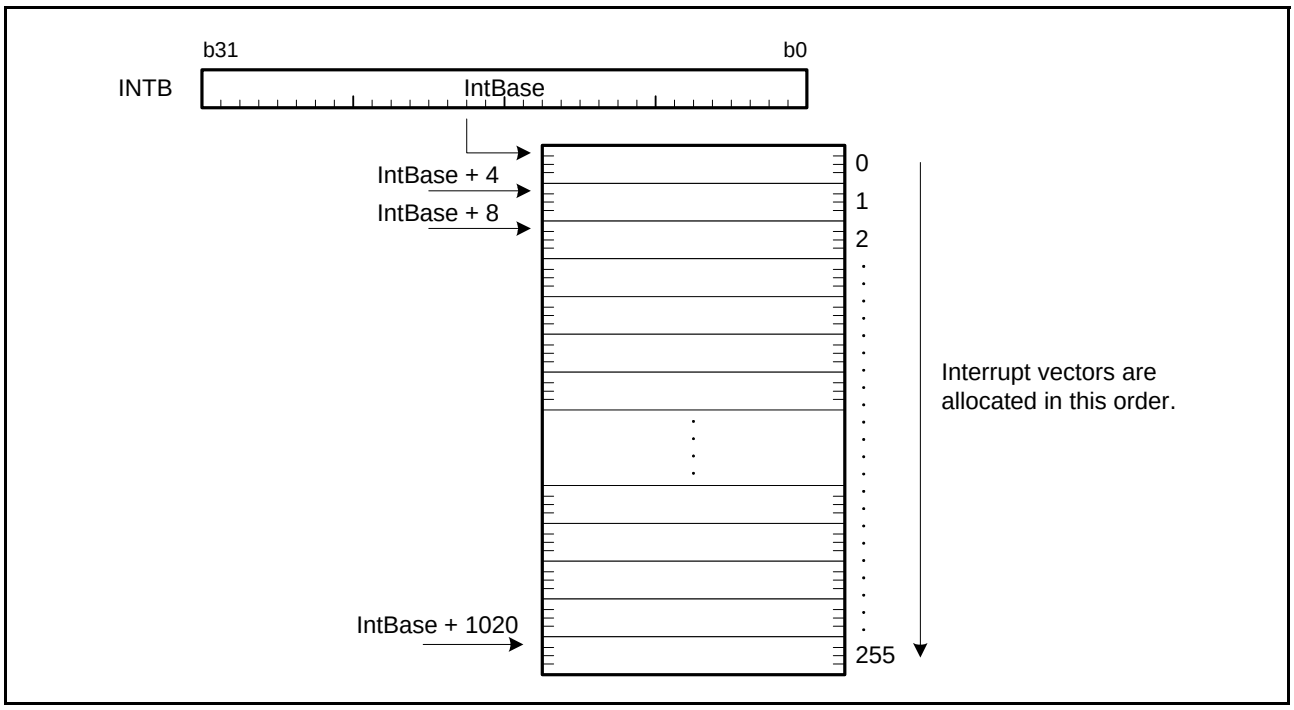


Figure 1.9 Relocatable Vector Table

1.8 Address Space

The address space of the RXv1 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 RXv1 CPU is depicted in Figure 1.10. For all regions, the designation may differ with the product and operating mode. For details, refer to the user's manual: hardware for each product.

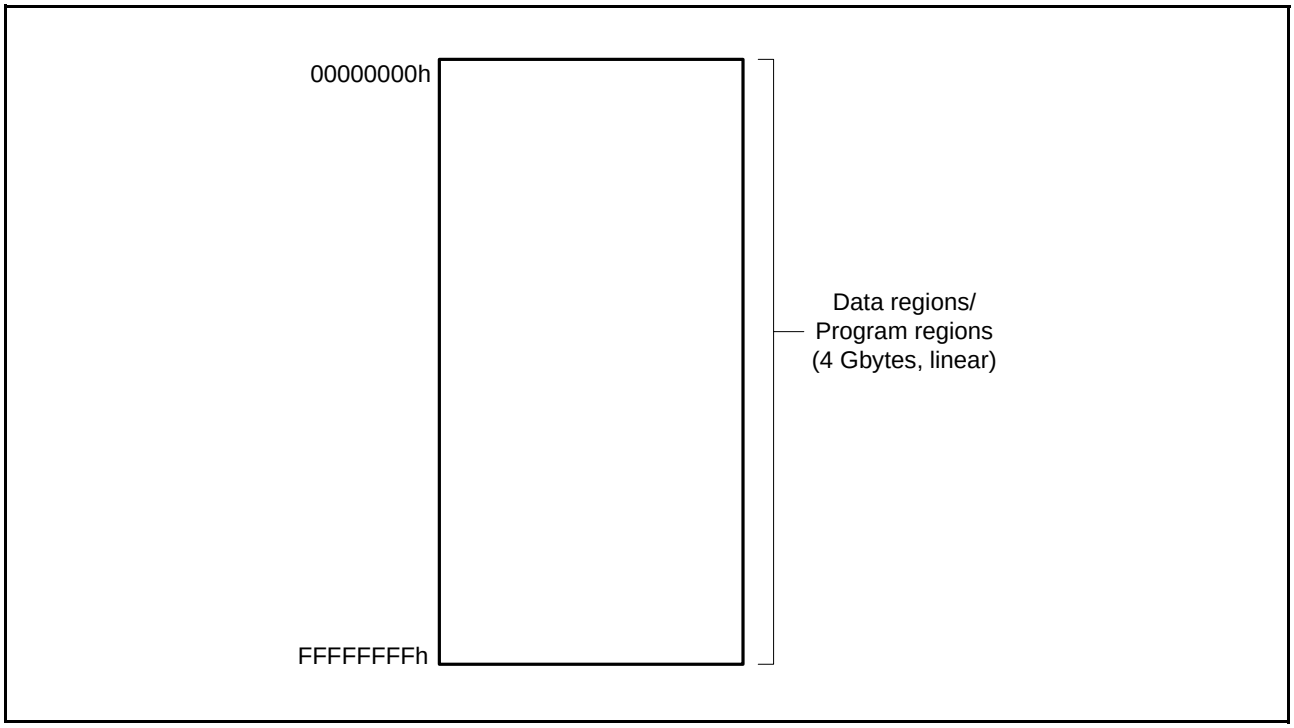


Figure 1.10 Address Space

2. Addressing Modes

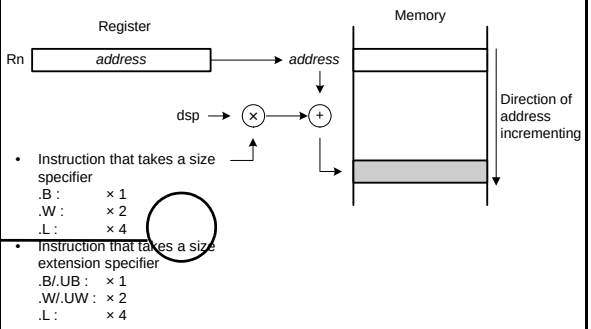
The following is a description of the notation and operations of each addressing mode.
There are ten types of addressing mode.

- 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

2.1 Guide to This Section

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

(1)

Register Relative		
(2) dsp:5[Rn] (Rn = R0 to R7) (3) dsp:8[Rn] (Rn = R0 to R15) (4) 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 MOVU instructions.	

(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

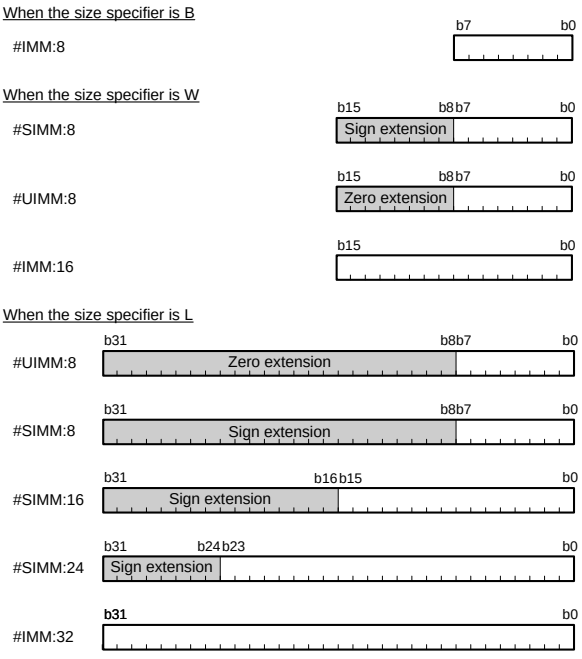
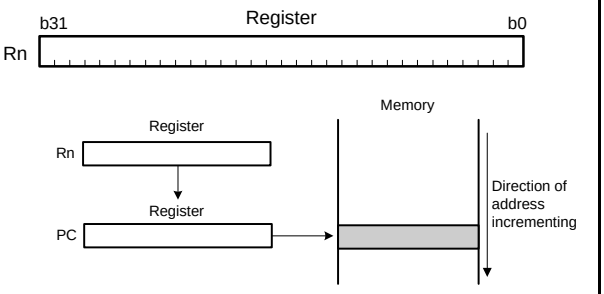
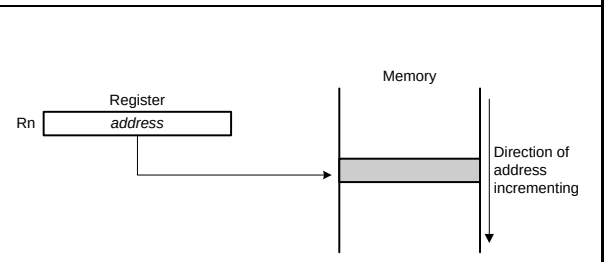
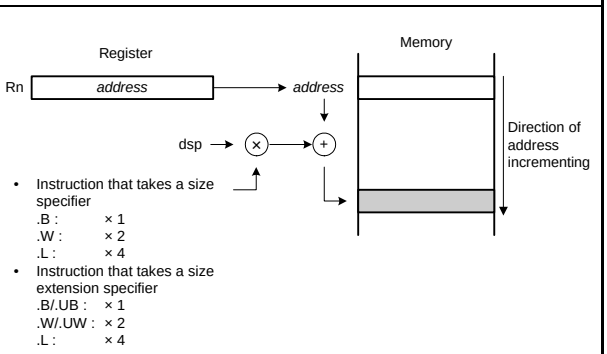
The operation and effective address range are described here.

(4) Operation diagram

The operation of the addressing mode is illustrated here.

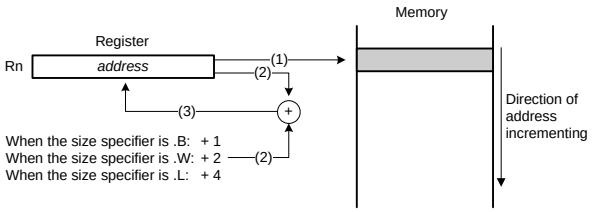
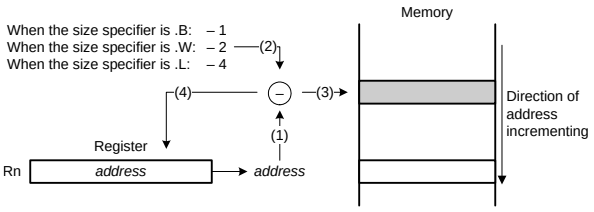
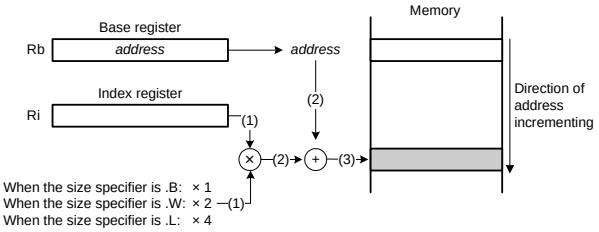
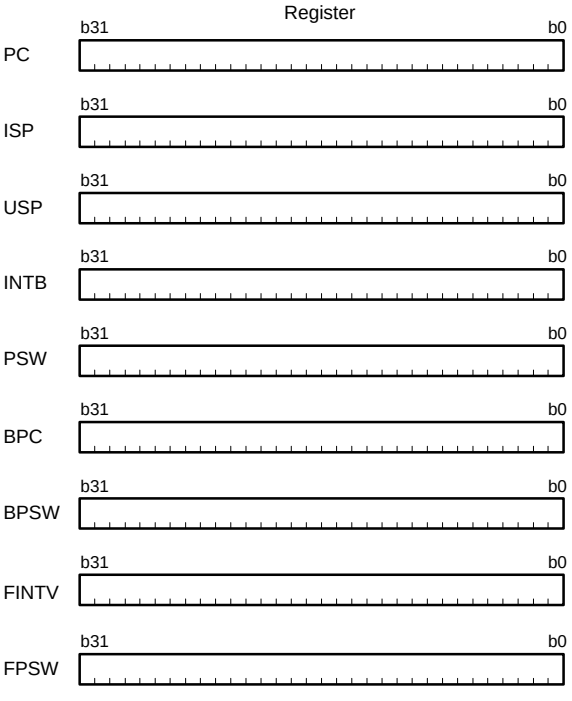
2.2 Addressing Modes

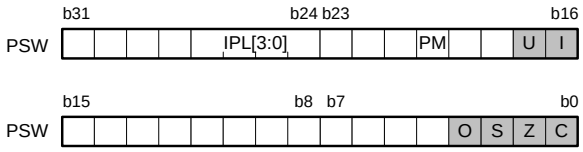
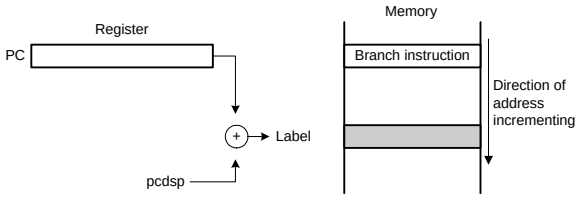
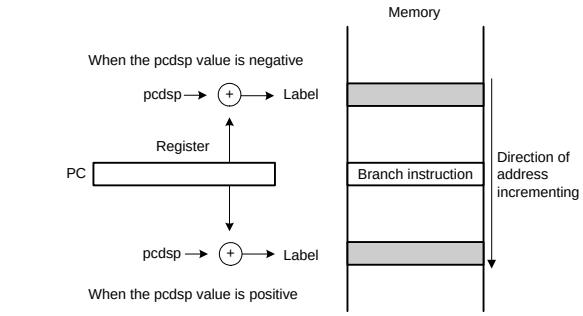
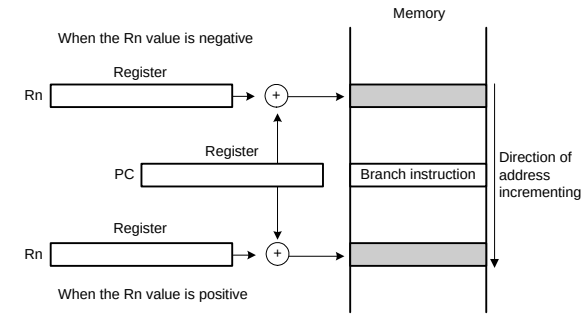
Immediate		
#IMM:1 #IMM:3 #IMM:4 #UIMM:4 #IMM:5	#IMM:1 The operand is the 1-bit immediate value indicated by #IMM. This addressing mode is used to specify sources for the RACW instruction. #IMM:3 The operand is the 3-bit immediate value indicated by #IMM. This addressing mode is used to specify the bit 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.	

<table>
<tr><td colspan="2">Immediate</td><td rowspan="2"> 
</td></tr>
<tr><td>#IMM:8
#SIMM:8
#UIMM:8
#IMM:16
#SIMM:16
#SIMM:24
#IMM:32</td><td>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.</td></tr>
<tr><td colspan="2">Register Direct</td><td rowspan="2"> 
</td></tr>
<tr><td>Rn
(Rn = R0 to R15)</td><td>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.</td></tr>
<tr><td colspan="2">Register Indirect</td><td rowspan="2"> 
</td></tr>
<tr><td>[Rn]
(Rn = R0 to R15)</td><td>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.</td></tr>
<tr><td colspan="2">Register Relative</td><td rowspan="2"> 
</td></tr>
<tr><td>dsp:5[Rn]
(Rn = R0 to R7)

dsp:8[Rn]
(Rn = R0 to R15)

dsp:16[Rn]
(Rn = R0 to R15)</td><td>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 MOVU instructions.</td></tr>
</table>

Post-increment Register Indirect		
[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		
[–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		
[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		
PC ISP USP INTB PSW BPC BPSW FINTV FPSW	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.	

<table>
<tr><td colspan="2">PSW Direct</td><td rowspan="2"> 
</td></tr>
<tr><td>C
Z
S
O
I
U</td><td>The operand is the specified flag or bit. This addressing mode is used with CLRPSW and SETPSW instructions.</td></tr>
<tr><td colspan="2">Program Counter Relative</td><td rowspan="2"> 
</td></tr>
<tr><td>pcdsp:3</td><td>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 to be used with the BCnd (only applicable in BEQ, BZ, BNE, and BNZ), and BRA instructions.</td></tr>
<tr><td>pcdsp:8
pcdsp:16
pcdsp:24</td><td>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. This addressing mode with the branch distance specifier “.B” is for use with any of the BCnd instructions and the BRA instruction, with the branch distance specifier “.W” is only for use with certain BCnd instructions (BEQ, BZ, BNE, and BNZ) and the BRA and BSR instructions, and with the branch distance specifier “.A” is only for use with the BRA and BSR instructions.</td><td> 
</td></tr>
<tr><td>Rn
(Rn = R0 to R15)</td><td>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.</td><td> 
</td></tr>
</table>

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 3.3, 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: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

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

Note: 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”.

Note: 3. For the ranges of immediate values for INT and RTSD instructions, see the relevant descriptions under section 3.3, Instructions in Detail.

3. Instruction Descriptions

3.1 Overview of Instruction Set

The number of instructions for the RXv1 Architecture is 90. A variable-length instruction format of 1 to 8 bytes is used.

The RXv1 instruction set is listed below.

List of Instructions (1 / 4)

Instruction Type	Mnemonic	Function	Instruction Described in Detail (on Page)	Instruction Code Described in Detail (on Page)	Notes
Basic instructions					
Arithmetic/ logic instructions	ABS	Absolute value	51	169	
	ADC	Add with carry	52	170	
	ADD	Add without carry	53	171	
	AND	Logical AND	55	173	
	CMP	Compare	68	187	
	DIV	Divide signed	69	188	
	DIVU	Divide unsigned	70	190	
	EMUL	Extended multiply signed	71	191	
	EMULU	Extended multiply unsigned	73	192	
	MAX	Maximum of two signed integers	95	203	
	MIN	Minimum of two signed integers	96	204	
	MUL	Multiply	102	211	
	NEG	Negate (two’s complement)	113	218	
	NOP	No operation	114	218	
	NOT	Logical NOT (one’s complement)	115	219	
	OR	Logical OR	116	220	
	RMPA	Repeat multiply-accumulate	127	226	
	ROLC	Rotate left with carry	129	226	
	RORC	Rotate right with carry	130	227	
	ROTL	Rotate left	131	227	
	ROTR	Rotate right	132	228	
	SAT	Saturate	141	231	
	SATR	Saturate for RMPA	142	231	
	SBB	Subtract with borrow	143	232	
	SHAR	Arithmetic shift right	147	235	
	SHLL	Logical shift left	148	236	
	SHLR	Logical shift right	149	237	
	SUB	Subtract without borrow	156	241	
	TST	Test logical	161	243	
	XOR	Logical Exclusive OR	164	245	

List of Instructions (2 / 4)

Instruction Type	Mnemonic		Function	Instruction Described in Detail (on Page)	Instruction Code Described in Detail (on Page)	Notes
Data transfer instructions	MOV		Move	97	205	
	MOVU		Move unsigned	100	210	
	POP		Pop register from stack	117	221	
	POPC		Pop a control register from stack	118	222	
	POPM		Pop multiple registers from stack	119	222	
	PUSH		Push register on stack	120	223	
	PUSHC		Push a control register on stack	121	224	
	PUSHM		Push multiple registers on stack	122	224	
	REVL		Reverse endian within longword	125	225	
	REVW		Reverse endian within word	126	225	
	SC*Cnd*	SCGEU	Store condition	144	233	
		SCC		144	233	
		SCEQ		144	233	
		SCZ		144	233	
		SCGTU		144	233	
		SCPZ		144	233	
		SCGE		144	233	
		SCGT		144	233	
		SCO		144	233	
		SCLTU		144	233	
		SCNC		144	233	
		SCNE		144	233	
		SCNZ		144	233	
		SCLEU		144	233	
		SCN		144	233	
		SCLE		144	233	
		SCLT		144	233	
		SCNO		144	233	
	STNZ		Store on not zero	154	239	
	STZ		Store on zero	155	240	
	XCHG		Exchange	163	244	

List of Instructions (3 / 4)

Instruction Type	Mnemonic		Function	Instruction Described in Detail (on Page)	Instruction Code Described in Detail (on Page)	Notes
Branch instructions	B*Cnd*	BGEU	Conditional relative branch	58	177	
		BC		58	177	
		BEQ		58	177	
		BZ		58	177	
		BGTU		58	177	
		BPZ		58	177	
		BGE		58	177	
		BGT		58	177	
		BO		58	177	
		BLTU		58	177	
		BNC		58	177	
		BNE		58	177	
		BNZ		58	177	
		BLEU		58	177	
		BN		58	177	
		BLE		58	177	
		BLT		58	177	
		BNO		58	177	
	BRA		Unconditional relative branch	62	181	
	BSR		Relative branch to subroutine	65	184	
	JMP		Unconditional jump	91	201	
	JSR		Jump to subroutine	92	201	
	RTS		Return from subroutine	138	230	
	RTSD		Return from subroutine after deallocating stack frame	139	230	
Bit manipulation instructions	BCLR		Clear a bit	57	175	
	BM*Cnd*	BMGEU	Conditional bit transfer	59	179	
		BMC		59	179	
		BMEQ		59	179	
		BMZ		59	179	
		BMGTU		59	179	
		BMPZ		59	179	
		BMGE		59	179	
		BMGT		59	179	
		BMO		59	179	
		BMLTU		59	179	
		BMNC		59	179	
		BMNE		59	179	
		BMNZ		59	179	
		BMLEU		59	179	
		BMN		59	179	
		BMLE		59	179	
		BMLT		59	179	
		BMNO		59	179	
	BNOT		Not a bit	61	180	
	BSET		Set a bit	64	182	
	BTST		Test a bit	66	185	

List of Instructions (4 / 4)

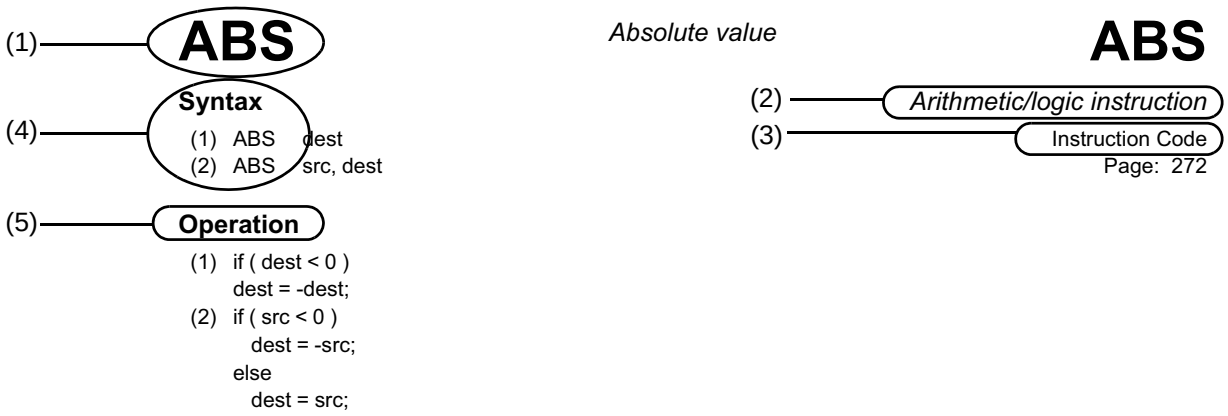
Instruction Type	Mnemonic	Function	Instruction Described in Detail (on Page)	Instruction Code Described in Detail (on Page)	Notes
String manipulation instructions	SCMPU	String compare until not equal	145	233	
	SMOVB	String move backward	150	238	
	SMOVF	String move forward	151	238	
	SMOVU	String move until zero detected	152	238	
	SSTR	String store	153	239	
	SUNTIL	String search until equal	157	242	
	SWHILE	String search while equal	159	242	
System manipulation instructions	BRK	Unconditional trap	63	182	
	CLRPSW	Clear a flag or bit in the PSW	67	186	
	INT	Software interrupt	88	200	
	MVFC	Move data from a control register	108	214	
	MVTC	Move data to a control register	111	216	
	MVTIPL (privileged instruction) *1	Move data to IPL	112	217	
	RTE (privileged instruction)	Return from exception	136	229	
	RTFI (privileged instruction)	Return from fast interrupt	137	230	
	SETPSW	Set a flag or bit in the PSW	146	234	
	WAIT (privileged instruction)	Wait	162	244	
Floating-point instructions (optional)					
Floating-point arithmetic instructions	FADD	Add floating-point	75	194	
	FCMP	Comparefloating-point	77	195	
	FDIV	Divide floating-point	79	196	
	FMUL	Multiply floating-point	81	197	
	FSUB	Subtractfloating-point	83	198	
	FTOI	Convert floating-point to signed integer	85	199	
	ITOF	Convert signed integer to floating-point	89	200	
	ROUND	Round floating-point to signed integer	133	229	
DSP instructions					
DSP instructions	MACHI	Multiply-Accumulate the upper words	93	202	
	MACLO	Multiply-Accumulate the lower words	94	202	
	MULHI	Multiply the upper words	104	213	
	MULLO	Multiply the lower words	105	213	
	MVFACHI	Move data from the upper longword of the accumulator	106	213	
	MVFACMI	Move data from the middle-order longword of the accumulator	107	214	
	MVTACHI	Move data to the upper longword of the accumulator	109	215	
	MVTACLO	Move data to the lower longword of the accumulator	110	215	
	RACW	Round the accumulator word	123	225	

Note: 1. Products of the RX610 Group do not support the MVTIPL instruction.

3.2 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.



(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

(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 number

(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 arithmetic 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	Operand src2	Operand 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	Operand 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) (d) (f) (e) (b) (c)

(a) Registers

Rs, Rs2, Rd, Rd2, Ri, and Rb mean that R0 to R15 are specifiable unless stated otherwise.

(b) Control registers

Rx indicates that the PC, ISP, USP, INTB, PSW, BPC, BPSW, FINTV, and FPSW 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

.memex indicates the size of an operand in memory and the form of extension. 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.

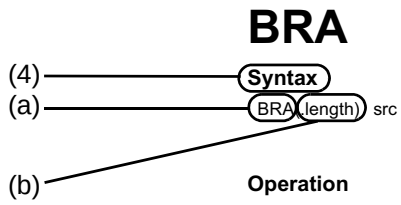
(f) Processing size

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* **BRA**

Branch instruction

Instruction Code
Page: 286

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 (Byte)
		src	Range of pcdsp/Rs	
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.

3.3 Instructions in Detail

The following pages give details of the RXv1 instructions.

ABS

Absolute value

ABS

Syntax

Arithmetic/logic instruction

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

Instruction Code
Page: 169

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

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

Description Example

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

ADC

Add with carry

ADC

Syntax

Arithmetic/logic instruction

ADC src, dest

Instruction Code
Page: 170

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

Syntax	Processing Size	Operand		Code Size (Byte)
		src	dest	
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

Add without carry

ADD

Syntax

Arithmetic/logic instruction

(1) ADD src, dest

(2) ADD src, src2, dest

Instruction Code
Page: 171

Operation

(1) dest = dest + src;

(2) dest = src2 + src;

Function

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

(2) This instruction adds src2 and src 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 src	Operand src2	Operand dest	Code Size (Byte)
(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

Syntax

Arithmetic/logic instruction

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

Instruction Code
Page: 173

Operation

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

Function

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

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

Clear a bit

BCLR

Syntax

Bit manipulation instruction

BCLR src, dest

Instruction Code
Page: 175

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

Syntax	Processing Size	Operand		Code Size (Byte)
		src	dest	
(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
```

B*Cnd*

Conditional relative branch

B*Cnd*

Syntax

Branch instruction

B*Cnd*(.length) src

Instruction Code
Page: 177

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

Syntax	Length	Operand		Code Size (Byte)
		src	Range of pcdsp	
(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* — BM*Cnd*

Syntax

Bit manipulation instruction

BM*Cnd* src, dest

Instruction Code
Page: 179

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		Code Size (Byte)
		src	dest	
(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

Not a bit

BNOT

Syntax

Bit manipulation instruction

BNOT src, dest

Instruction Code
Page: 180

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

Syntax	Processing Size	Operand		Code Size (Byte)
		src	dest	
(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

BRA

Syntax

BRA(.length) src

Branch instruction

Instruction Code
Page: 181

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 (Byte)
		src	Range of pcdsp/Rs	
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

BRK

Syntax

System manipulation instruction

BRK

Instruction Code

Page: 182

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 saved on the stack.

Instruction Format

Syntax	Code Size (Byte)
BRK	1

Description Example

```
BRK
```

BSET

Set a bit

BSET

Bit manipulation instruction

Instruction Code
Page: 182

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

Syntax	Processing Size	Operand		Code Size (Byte)
		src	dest	
(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 branch to subroutine

BSR

Syntax

Branch instruction

BSR(.length) src

Instruction Code
Page: 184

Operation

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

Note: * (PC + n) is the address of the instruction following the BSR instruction.
"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

Syntax	Length	Operand		Code Size (Byte)
		src	Range of pcdsp/Rs	
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

Test a bit

BTST

Syntax

Bit manipulation instruction

BTST src, src2

Instruction Code
Page: 185

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

Syntax	Processing Size	Operand src	Operand src2	Code Size (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* CLRPSW

Syntax

System manipulation instruction

CLRPSW dest

Instruction Code
Page: 186

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

Compare

CMP

Syntax

Arithmetic/logic instruction

CMP src, src2

Instruction Code
Page: 187

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

Syntax	Processing Size	Operand		Code Size (Byte)
		src	src2	
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”)

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

Note: 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

Divide signed

DIV

Syntax

Arithmetic/logic instruction

DIV src, dest

Instruction Code
Page: 188

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

Syntax	Processing Size	Operand src	Operand dest	Code Size (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

Divide unsigned

DIVU

Syntax

Arithmetic/logic instruction

DIVU src, dest

Instruction Code
Page: 190

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

Syntax	Processing Size	Operand src	Operand dest	Code Size (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
```

EMUL

Extended multiply signed

EMUL

Syntax

Arithmetic/logic instruction

EMUL src, dest

Instruction Code
Page: 191

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-purpose registers (Rn (n = 0 to 14)) is specifiable for dest.

Note: The accumulator (ACC) is used to perform the function. The value of ACC 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
```

EMULU

Extended multiply unsigned

EMULU

Syntax

Arithmetic/logic instruction

EMULU src, dest

Instruction Code
Page: 192

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-purpose registers (Rn (n = 0 to 14)) is specifiable for dest.

Note: The accumulator (ACC) is used to perform the function. The value of ACC 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 src	Operand dest	Code Size (Byte)
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

Add floating-point

FADD

Syntax

Floating-point arithmetic instruction

FADD src, dest

Instruction Code
Page: 194

Operation

dest = dest + src;

Function

- This instruction adds the single-precision floating-point numbers stored in dest and src and places the result in dest.
- Rounding of the result is in accordance 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 result of the operation is +0 if src and dest have the opposite signs and their sum is exactly 0, except when the rounding mode is 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	dest	
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

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.

Sources of Floating-Point Exceptions

Unimplemented processing
Invalid operation
Overflow
Underflow
Inexact

Description Example

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

Supplementary Description

- The following tables show the correspondences between the src and dest values and the results of operations when the value of the DN bit in the FPSW is 0 or 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 colspan="3" rowspan="3">Sum</td><td rowspan="4">+∞</td><td rowspan="3">–∞</td><td rowspan="5">Unimplemented processing</td><td rowspan="6">QNaN</td><td rowspan="7">Invalid operation</td></tr>
<tr><td>+0</td><td>+0</td><td>*</td></tr>
<tr><td>–0</td><td>*</td><td>–0</td></tr>
<tr><td>+∞</td><td colspan="3">+∞</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>

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

When DN = 1

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

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

FCMP *Comparefloating-point* FCMP

Syntax

Floating-point arithmetic instruction

FCMP src, src2

Instruction Code
Page: 195

Operation

src2 - src;

Function

- This instruction compares the single-precision floating-point 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.

Sources of Floating-Point Exceptions

Unimplemented processing
Invalid operation

Description Example

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

Supplementary Description

- The following tables show the correspondences between the src and src2 values and the results of operations when the value of the DN bit in the FPSW is 0 or 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			<	>		
	+0, +Denormalized		=					
	–0, –Denormalized							
	+∞	>			=			
	–∞	<				=		
	QNaN	Ordered classification impossible						
	SNaN	Invalid operation (Ordered classification impossible)						

FDIV *Divide floating-point* FDIV

Syntax

Floating-point arithmetic instruction

FDIV src, dest

Instruction Code
Page: 196

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 accordance 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

Syntax	Processing Size	Operand src	Operand dest	Code Size (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.

Sources of Floating-Point 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 the src and dest values and the results of operations when the value of the DN bit in the FPSW is 0 or 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>Division</td><td colspan="2">Division-by-zero</td><td colspan="2">0</td><td rowspan="5"></td><td rowspan="6"></td><td rowspan="7"></td></tr>
<tr><td>+0</td><td rowspan="2">0</td><td colspan="2" rowspan="2">Invalid operation</td><td>+0</td><td>–0</td></tr>
<tr><td>–0</td><td>–0</td><td>+0</td></tr>
<tr><td>+∞</td><td rowspan="2">∞</td><td>+∞</td><td>–∞</td><td colspan="2" rowspan="2">Invalid operation</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

<table>
<tr><td colspan="2" rowspan="2"></td><td colspan="7">src</td></tr>
<tr><td>Normalized</td><td>+0,
+Denormalized</td><td>–0,
–Denormalized</td><td>+∞</td><td>–∞</td><td>QNaN</td><td>SNaN</td></tr>
<tr><td rowspan="6">dest</td><td>Normalized</td><td>Division</td><td colspan="2">Division-by-zero</td><td colspan="2">0</td><td rowspan="5"></td><td rowspan="6"></td></tr>
<tr><td>+0,
+Denormalized</td><td rowspan="2">0</td><td colspan="2" rowspan="2">Invalid operation</td><td>+0</td><td>–0</td></tr>
<tr><td>–0,
–Denormalized</td><td>–0</td><td>+0</td></tr>
<tr><td>+∞</td><td rowspan="2">∞</td><td>+∞</td><td>–∞</td><td colspan="2" rowspan="2">Invalid operation</td></tr>
<tr><td>–∞</td><td>–∞</td><td>+∞</td></tr>
<tr><td>QNaN</td><td colspan="6">QNaN</td></tr>
<tr><td>SNaN</td><td colspan="7">Invalid operation</td></tr>
</table>

FMUL *Multiply floating-point* FMUL

Syntax

Floating-point arithmetic instruction

FMUL src, dest

Instruction Code
Page: 197

Operation

dest = dest * src;

Function

- This instruction multiplies 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 accordance 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 accumulator (ACC) is used to perform the function. The value of ACC 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 src	Operand dest	Code Size (Byte)
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

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.

Sources of Floating-Point Exceptions

Unimplemented processing
Invalid operation
Overflow
Underflow
Inexact

Description Example

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

Supplementary Description

- The following tables show the correspondences between the src and dest values and the results of operations when the value of the DN bit in the FPSW is 0 or 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 colspan="3" rowspan="3">Multiplication</td><td colspan="2">∞</td><td rowspan="3"></td><td rowspan="8"></td><td rowspan="8"></td></tr>
<tr><td>+0</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><td rowspan="2"></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="7">Invalid operation</td></tr>
</table>

When DN = 1

<table>
<tr><td colspan="2" rowspan="2"></td><td colspan="7">src</td></tr>
<tr><td>Normalized</td><td>+0,
+Denormalized</td><td>–0,
–Denormalized</td><td>+∞</td><td>–∞</td><td>QNaN</td><td>SNaN</td></tr>
<tr><td rowspan="6">dest</td><td>Normalized</td><td colspan="3" rowspan="3">Multiplication</td><td colspan="2">∞</td><td rowspan="3"></td><td rowspan="6"></td></tr>
<tr><td>+0,
+Denormalized</td><td>+0</td><td>–0</td><td colspan="2" rowspan="2">Invalid operation</td></tr>
<tr><td>–0,
–Denormalized</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><td rowspan="2"></td></tr>
<tr><td>–∞</td><td>–∞</td><td>+∞</td></tr>
<tr><td>QNaN</td><td colspan="6">QNaN</td></tr>
<tr><td>SNaN</td><td colspan="6">Invalid operation</td></tr>
</table>

FSUB *Subtractfloating-point* FSUB

Syntax

Floating-point arithmetic instruction

FSUB src, dest

Instruction Code
Page: 198

Operation

dest = dest - src;

Function

- This instruction subtracts the single-precision floating-point number stored in src from that stored in dest and places the result in dest.
- Rounding of the result is in accordance 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 result of the operation is +0 if src and dest have the same sign and the result of subtraction is exactly 0, except when the rounding mode is 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	Operand dest	Code Size (Byte)
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

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.

Sources of Floating-Point Exceptions

Unimplemented processing
Invalid operation
Overflow
Underflow
Inexact

Description Example

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

Supplementary Description

- The following tables show the correspondences between the src and dest values and the results of operations when the value of the DN bit in the FPSW is 0 or 1.

When DN = 0

		src							
		Normalized	+0	–0	+∞	–∞	Denormalized	QNaN	SNaN
dest	Normalized	Subtraction			–∞	+∞		QNaN	Invalid operation
	+0		*	+0					
	–0		–0	*					
	+∞	+∞			Invalid operation				
	–∞	–∞				Invalid operation			
	Denormalized	Unimplemented processing							
	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.

When DN = 1

		src						
		Normalized	+0, +Denormalized	–0, –Denormalized	+∞	–∞	QNaN	SNaN
dest	Normalized	Subtraction			–∞	+∞	QNaN	Invalid operation
	+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 *Convert floating-point to signed integer* FTOI

Syntax

Floating-point arithmetic instruction

FTOI src, dest

Instruction Code
Page: 199

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

Syntax	Processing Size	Operand src	Operand dest	Code Size (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.

Sources of Floating-Point Exceptions

Unimplemented processing
Invalid operation
Inexact

Description Example

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

Supplementary Description

- The following tables show the correspondences between the src value and the result of operations when the value of the DN bit in the FPSW is 0 or 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 = 1: No change	Invalid operation exception
	31 ≤ Exponent ≤ 127	Other cases: 7FFFFFFFh	
	–126 ≤ Exponent ≤ 30	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
	–126 ≤ Exponent ≤ 30	00000000h to 80000080h	None*1
	31 ≤ Exponent ≤ 127	When an invalid operation exception is generated with the EV = 1: No change	Invalid operation exception*2
	–∞	Other cases: 80000000h	
NaN	QNaN	When an invalid operation exception is generated with the EV = 1: No change	Invalid operation exception
		Other cases:	
	SNaN	Sign bit = 0: 7FFFFFFFh	
		Sign bit = 1: 80000000h	

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

Note: 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 = 1: No change	Invalid operation exception
	31 ≤ Exponent ≤ 127	Other cases: 7FFFFFFFh	
	–126 ≤ Exponent ≤ 30	00000000h to 7FFFFF80h	None*1
	+0, +Denormalized number	00000000h	None
src < 0	–0, –Denormalized number		
	–126 ≤ Exponent ≤ 30	00000000h to 80000080h	None*1
	31 ≤ Exponent ≤ 127	When an invalid operation exception is generated with the EV = 1: No change	Invalid operation exception*2
	–∞	Other cases: 80000000h	
NaN	QNaN	When an invalid operation exception is generated with the EV = 1: No change	Invalid operation exception
		Other cases:	
	SNaN	Sign bit = 0: 7FFFFFFFh	
		Sign bit = 1: 80000000h	

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

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

INT

Software interrupt

INT

Syntax

System manipulation instruction

INT src

Instruction Code
Page: 200

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 $0 \leq src \leq 255$.
- 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 saved on the stack.

Instruction Format

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

Description Example

```
INT    #0
```

ITOF

Convert signed integer to floating-point

ITOF

Syntax

ITOF src, dest

Floating-point arithmetic instruction

Instruction Code

Page: 200

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 accordance 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 src	Operand dest	Code Size (Byte)
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.

Sources of Floating-Point Exceptions

Inexact

Description Example

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

JMP

Unconditional jump

JMP

Branch instruction

Instruction Code

Page: 201

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

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

Description Example

JMP R1

JSR

Jump to subroutine

JSR

Syntax

Branch instruction

JSR src

Instruction Code
Page: 201

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

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

Description Example

JSR R1

MACHI

Multiply-Accumulate the upper words

MACHI

Syntax

DSP instruction

MACHI src, src2

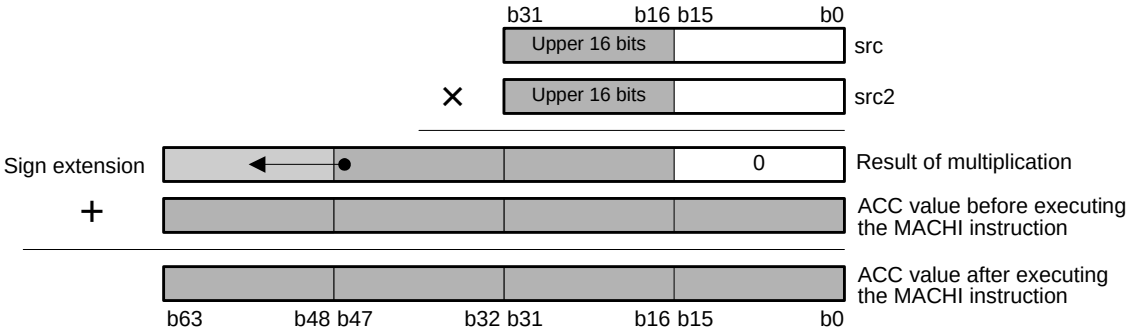
Instruction Code
Page: 202

Operation

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

Function

- This instruction multiplies the upper 16 bits of src by the upper 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 upper 16 bits of src and the upper 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**	**(Byte)**
MACHI src, src2	Rs	Rs2	3

Description Example

MACHI R1, R2

MACLO

Multiply-Accumulate the lower words

MACLO

Syntax

DSP instruction

MACLO src, src2

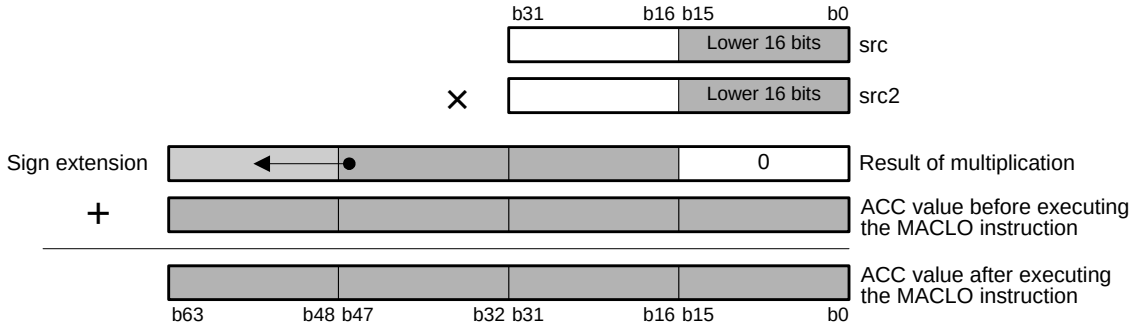
Instruction Code
Page: 202

Operation

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

Function

- This instruction multiplies the lower 16 bits of src by the lower 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 16 bits of src and the lower 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**	**(Byte)**
MACLO src, src2	Rs	Rs2	3

Description Example

MACLO R1, R2

MAX

Maximum of two signed integers

MAX

Syntax

Arithmetic/logic instruction

MAX src, dest

Instruction Code
Page: 203

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

Syntax	Processing Size	Operand src	Operand dest	Code Size (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

Minimum of two signed integers

MIN

Syntax

Arithmetic/logic instruction

MIN src, dest

Instruction Code
Page: 204

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

Syntax	Processing Size	Operand src	Operand dest	Code Size (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

Move

MOV

Syntax

Data transfer instruction

MOV.size src, dest

Instruction Code
Page: 205

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 least significant byte 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 lower word of the register (src) has been 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 least significant byte of the register is transferred. When the size specifier is .W, the lower word 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

Syntax	Size	Processing Size	Operand src	Operand dest	Code Size (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

Syntax	Size	Processing Size	Operand src	Operand dest	Code Size (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

Note: 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.

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

Note: 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.

Note: 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
```

MOVU

Move unsigned

MOVU

Syntax

Data transfer instruction

MOVU.size src, dest

Instruction Code
Page: 210

Operation

dest = src;

Function

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

src	dest	Function
Register	Register	Transfers the least significant byte or lower word of the 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 src	Operand dest	Code Size (Byte)
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

Note: 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.

Note: 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
```

MUL

Multiply

MUL

Syntax

Arithmetic/logic instruction

(1) MUL src, dest

(2) MUL src, src2, dest

Instruction Code
Page: 211

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 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 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 (ACC) is used to perform the function. The value of ACC after executing the instruction is undefined.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Processing Size	Operand src	Operand src2	Operand dest	Code Size (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 upper words

MULHI

Syntax

DSP instruction

MULHI src, src2

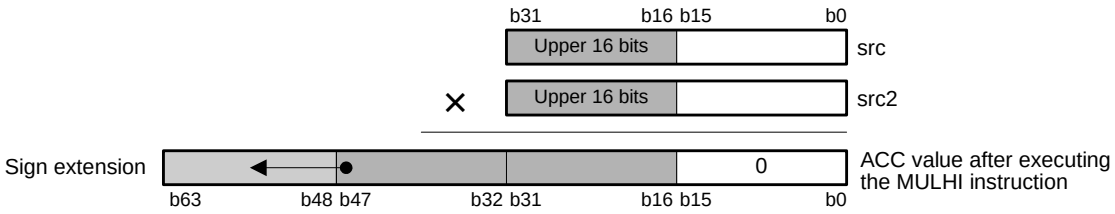
Instruction Code
Page: 213

Operation

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

Function

- This instruction multiplies the upper 16 bits of src by the upper 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 63 to 48 of ACC is sign-extended. Moreover, bits 15 to 0 of ACC are cleared to 0. The upper 16 bits of src and the upper 16 bits of src2 are treated as signed integers.



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

	Operand		
Syntax	**src**	**src2**	**Code Size (Byte)**
MULHI src, src2	Rs	Rs2	3

Description Example

MULHI R1, R2

MULLO

Multiply the lower words

MULLO

Syntax

DSP instruction

MULLO src, src2

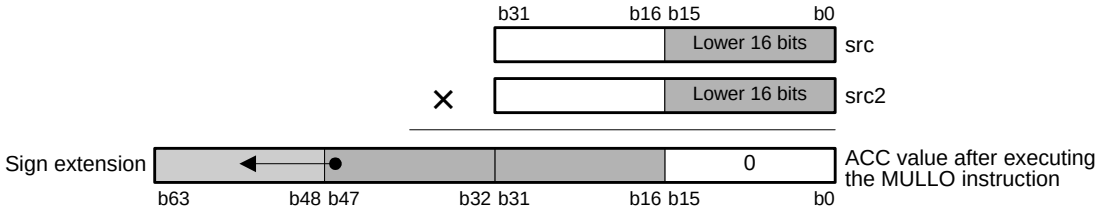
Instruction Code
Page: 213

Operation

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

Function

- This instruction multiplies the lower 16 bits of src by the lower 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 63 to 48 of ACC is sign-extended. Moreover, bits 15 to 0 of ACC are cleared to 0. The lower 16 bits of src and the lower 16 bits of src2 are treated as signed integers.



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

	Operand		
Syntax	**src**	**src2**	**Code Size (Byte)**
MULLO src, src2	Rs	Rs2	3

Description Example

MULLO R1, R2

MVFACHI

Move data from the upper longword of the accumulator

MVFACHI

Syntax

DSP instruction

MVFACHI dest

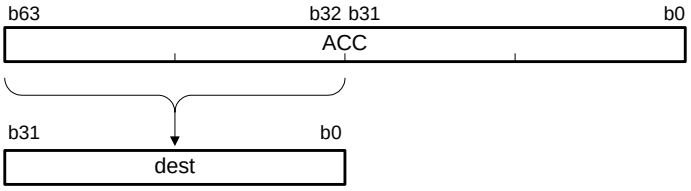
Instruction Code
Page: 213

Operation

dest = (signed long) (ACC >> 32);

Function

- This instruction moves the contents of upper 32 bits of the accumulator (ACC) to dest.



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

	Operand	
Syntax	**dest**	**Code Size (Byte)**
MVFACHI dest	Rd	3

Description Example

MVFACHI R1

MVFACMI

Move data from the middle-order longword of the accumulator

MVFACMI

Syntax

DSP instruction

MVFACMI dest

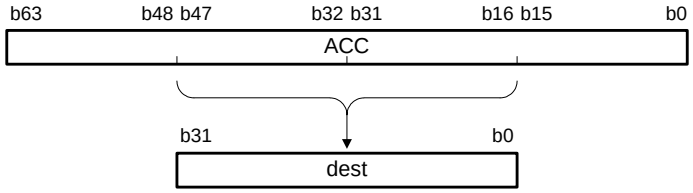
Instruction Code
Page: 214

Operation

dest = (signed long) (ACC >> 16);

Function

- 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

	Operand	
Syntax	**dest**	**Code Size (Byte)**
MVFACMI dest	Rd	3

Description Example

MVFACMI R1

MVFC

Move data from a control register

MVFC

Syntax

System manipulation instruction

MVFC src, dest

Instruction Code

Page: 214

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

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

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

Description Example

MVFC USP, R1

MVTACHI

Move data to the upper longword of the accumulator

MVTACHI

Syntax

DSP instruction

MVTACHI src

Instruction Code

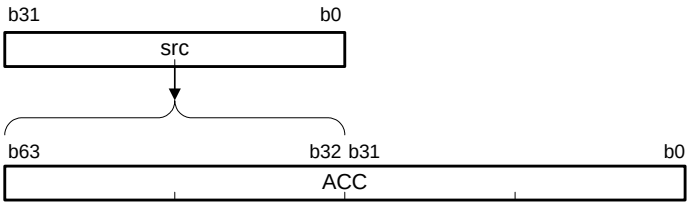
Page: 215

Operation

ACC = (ACC & 00000000FFFFFFFFh) | ((signed long long src << 32);

Function

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



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

	Operand	
Syntax	**src**	**Code Size (Byte)**
MVTACHI src	Rs	3

Description Example

MVTACHI R1

MVTACLO

Move data to the lower longword of the accumulator

MVTACLO

Syntax

DSP instruction

MVTACLO src

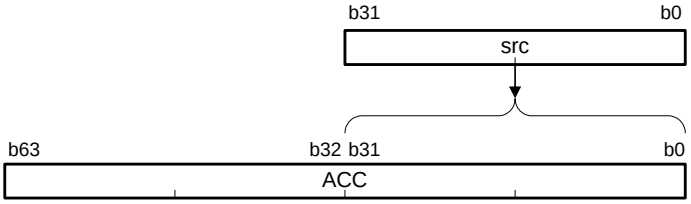
Instruction Code
Page: 215

Operation

ACC = (ACC & FFFFFFFF00000000h) | (unsigned long long) src;

Function

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



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

	Operand	
Syntax	**src**	**Code Size (Byte)**
MVTACLO src	Rs	3

Description Example

MVTACLO R1

MVTC

Move data to a control register

MVTC

Syntax

System manipulation instruction

MVTC src, dest

Instruction Code
Page: 216

Operation

dest = src;

Function

- This instruction transfers src to dest.
- In user mode, writing to the ISP, INTB, 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

			Operand		
Syntax		Processing Size	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

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

Description Example

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

MVTIPL

Move data to IPL

MVTIPL

Syntax

System manipulation instruction

MVTIPL src

Instruction Code
Page: 217

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

Description Example

MVTIPL #2

Note: Products of the RX610 Group do not support the MVTIPL instruction. Use the MVTC instruction for products of the RC610 Group.

NEG

Negate (two's complement)

NEG

Syntax

Arithmetic/logic instruction

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

Instruction Code
Page: 218

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

			Operand		
Syntax		Processing Size	src	dest	Code Size (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

NOP

Syntax

Arithmetic/logic instruction

NOP

Instruction Code
Page: 218

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 NOT (one's complement)

NOT

Syntax

Arithmetic/logic instruction

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

Instruction Code
Page: 219

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

Syntax	Processing Size	Operand		Code Size (Byte)
		src	dest	
(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

Syntax

Arithmetic/logic instruction

(1) OR src, dest

(2) OR src, src2, dest

Instruction Code
Page: 220

Operation

(1) dest = dest | src;

(2) dest = src2 | src;

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 src2 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 src2	Operand dest	Code Size (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

Pop register from stack

POP

Syntax

Data transfer instruction

POP dest

Instruction Code
Page: 221

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

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

Description Example

```
POP R1
```

POPC

Pop a control register from stack

POPC

Syntax

Data transfer instruction

POPC dest

Instruction Code

Page: 222

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, 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

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

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

Description Example

POPC PSW

POPM

Pop multiple registers from stack

POPM

Syntax

Data transfer instruction

POPM dest-dest2

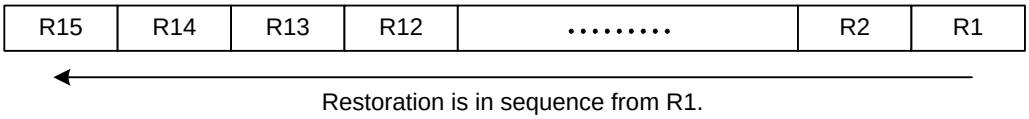
Instruction Code
Page: 222

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:



Restoration is in sequence from R1.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

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

PUSH

Push register on stack

PUSH

Syntax

Data transfer instruction

PUSH.size src

Instruction Code
Page: 223

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 upper 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 upper 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

			Operand	
Syntax	**Size**	**Processing Size**	**src**	**Code Size (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

Push a control register on stack

PUSHC

Syntax

Data transfer instruction

PUSHC src

Instruction Code
Page: 224

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

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

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

Description Example

```
PUSHC PSW
```

PUSHM

Push multiple registers on stack

PUSHM

Syntax

Data transfer instruction

PUSHM src-src2

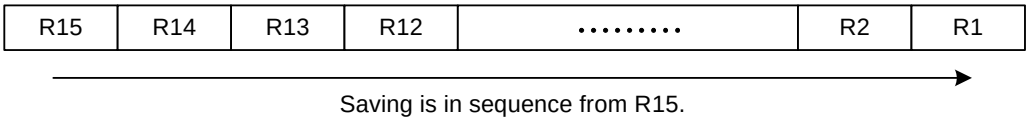
Instruction Code
Page: 224

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 on the stack in the following order:



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
```

RACW

Round the accumulator word

RACW

Syntax

DSP instruction

RACW src

Instruction Code

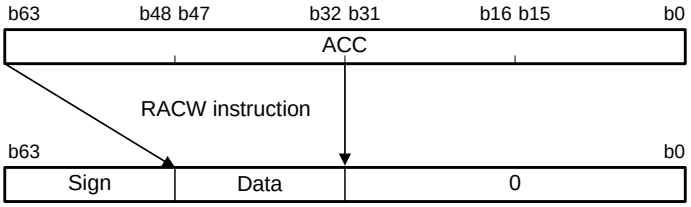
Page: 225

Operation

```
signed long long tmp;
tmp = (signed long long) ACC << src;
tmp = tmp + 0000000080000000h;
if (tmp > (signed long long) 00007FFF00000000h)
   ACC = 00007FFF00000000h;
else if (tmp < (signed long long) FFFF800000000000h)
   ACC = FFFF800000000000h;
else
   ACC = tmp & FFFFFFFF00000000h;
```

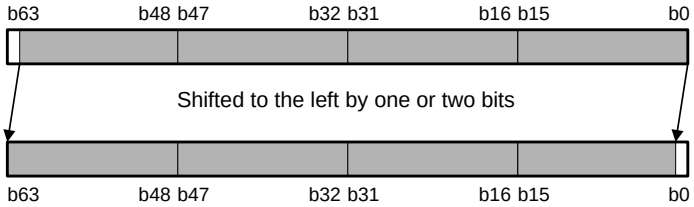
Function

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

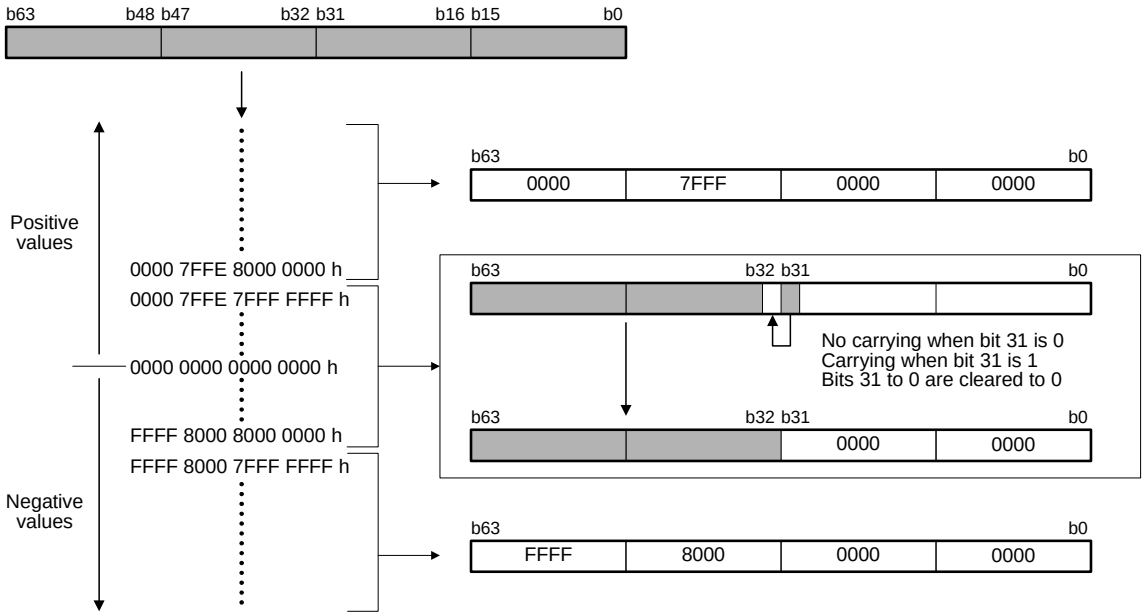


- The RACW instruction is executed according to the following procedures.

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



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

Description Example

```
RACW    #1
RACW    #2
```

REVL

Reverse endian within longword

REVL

Syntax

Data transfer instruction

REVL src, dest

Instruction Code

Page: 225

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

Description Example

REVL R1, R2

REVW

Reverse endian within word

REVW

Syntax

Data transfer instruction

REVW src, dest

Instruction Code
Page: 225

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

Description Example

REVW R1, R2

RMPA

Repeat multiply-accumulate

RMPA

Syntax

Arithmetic/logic instruction

RMPA.size

Instruction Code
Page: 226

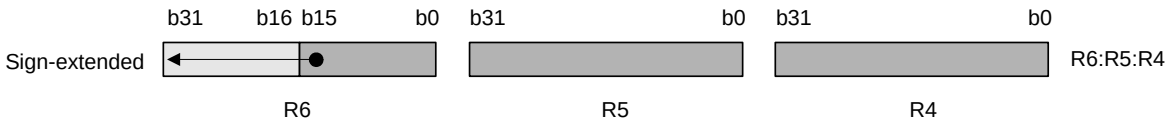
Operation

```
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 upper 16 bits of R6 are set to the value obtained by sign-extending the lower 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 user's manual: hardware of each product.

Note: The accumulator (ACC) is used to perform the function. The value of ACC 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

Rotate left with carry

ROLC

Syntax

Arithmetic/logic instruction

ROLC dest

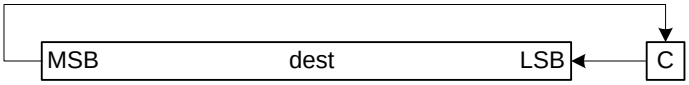
Instruction Code
Page: 226

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

Rotate right with carry

RORC

Syntax

Arithmetic/logic instruction

RORC dest

Instruction Code

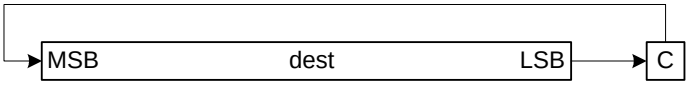
Page: 227

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

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

Description Example

RORC R1

ROTL

Rotate left

ROTL

Syntax

Arithmetic/logic instruction

ROTL src, dest

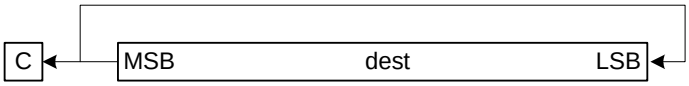
Instruction Code
Page: 227

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 $0 \leq src \leq 31$.
- 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 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

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

Description Example

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

ROTR

Rotate right

ROTR

Syntax

Arithmetic/logic instruction

ROTR src, dest

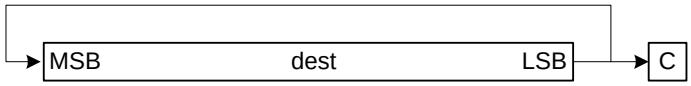
Instruction Code
Page: 228

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 $0 \leq src \leq 31$.
- 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

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

Description Example

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

ROUND

Round floating-point to signed integer

ROUND

Syntax

ROUND src, dest

Floating-point arithmetic instruction

Instruction Code

Page: 229

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.
- Rounding of the result is in accordance with 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 src	Operand dest	Code Size (Byte)
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.

Sources of Floating-Point Exceptions

Unimplemented processing
Invalid operation
Inexact

Description Example

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

Supplementary Description

- The following tables show the correspondences between the src value and the result of operations when the value of the DN bit in the FPSW is 0 or 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 = 1: No change	Invalid operation exception
	31 ≤ Exponent ≤ 127	Other cases: 7FFFFFFFh	
	–126 ≤ Exponent ≤ 30	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
	–126 ≤ Exponent ≤ 30	00000000h to 80000080h	None*1
	31 ≤ Exponent ≤ 127	When an invalid operation exception is generated with the EV = 1: No change	Invalid operation exception*2
	–∞	Other cases: 80000000h	
NaN	QNaN	When an invalid operation exception is generated with the EV = 1: No change	Invalid operation exception
		Other cases:	
	SNaN	Sign bit = 0: 7FFFFFFFh	
		Sign bit = 1: 80000000h	

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

Note: 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 = 1: No change	Invalid operation exception
	31 ≤ Exponent ≤ 127	Other cases: 7FFFFFFFh	
	–126 ≤ Exponent ≤ 30	00000000h to 7FFFFF80h	None*1
	+0, +Denormalized number	00000000h	None
src < 0	–0, –Denormalized number		
	–126 ≤ Exponent ≤ 30	00000000h to 80000080h	None*1
	31 ≤ Exponent ≤ 127	When an invalid operation exception is generated with the EV = 1: No change	Invalid operation exception*2
	–∞	Other cases: 80000000h	
NaN	QNaN	When an invalid operation exception is generated with the EV = 1: No change	Invalid operation exception
		Other cases:	
	SNaN	Sign bit = 0: 7FFFFFFFh	
		Sign bit = 1: 80000000h	

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

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

RTE

Return from exception

RTE

Syntax

System manipulation instruction

RTE

Instruction Code

Page: 229

Operation

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

Function

- This instruction returns execution from the exception handling routine by restoring the PC and PSW contents that were saved 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 fast interrupt

RTFI

Syntax

System manipulation instruction

RTFI

Instruction Code

Page: 230

Operation

PSW = BPSW;
PC = BPC;

Function

- This instruction returns execution from the fast-interrupt processing routine 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

Return from subroutine

RTS

Syntax

RTS

Branch instruction

Instruction Code

Page: 230

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

Return from subroutine after deallocating stack frame

RTSD

Syntax

Branch instruction

(1) RTSD src

(2) RTSD src, dest-dest2

Instruction Code
Page: 230

Operation

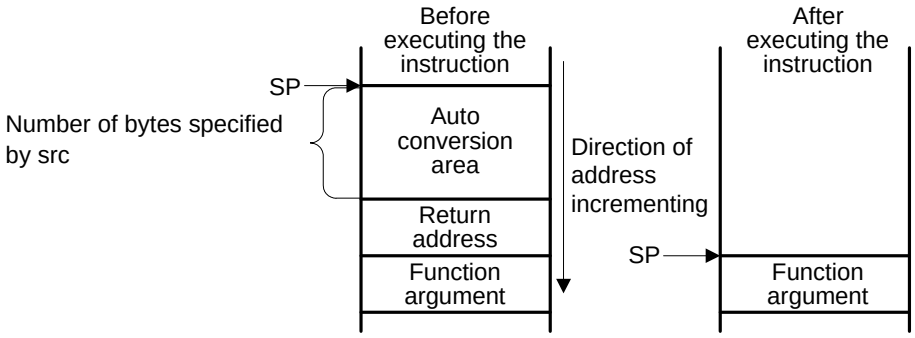
(1)
```
SP = SP + src;
PC = *SP;
SP = SP + 4;
```

(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

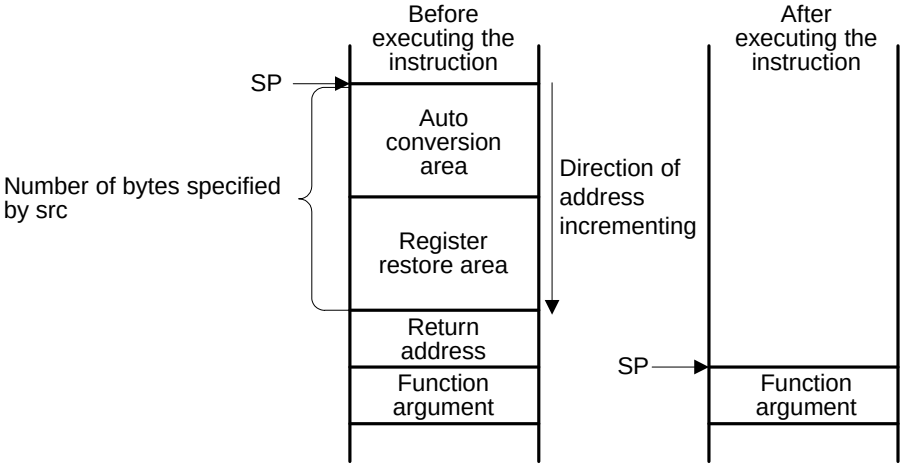
(1) This instruction returns the flow of execution from a subroutine after deallocating the stack frame for the subroutine.

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

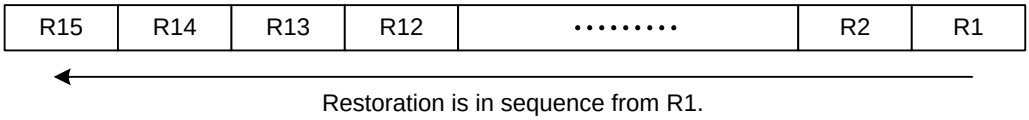


(2) This instruction returns the flow of execution from a subroutine after deallocating the stack frame for the subroutine and also restoring register values from the stack.

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



- 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:



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

Saturate

SAT

Syntax

Arithmetic/logic instruction

SAT dest

Instruction Code
Page: 231

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

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

Description Example

SAT R1

SATR

Saturate for RMPA

SATR

Syntax

Arithmetic/logic instruction

SATR

Instruction Code
Page: 231

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

Subtract with borrow

SBB

Syntax

Arithmetic/logic instruction

SBB src, dest

Instruction Code
Page: 232

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

Syntax	Processing Size	Operand src	Operand dest	Code Size (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*

Store condition

SC*Cnd*

Syntax

Data transfer instruction

SC*Cnd*.size dest

Instruction Code
Page: 233

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, SCC	C == 1	Equal to or greater than/ C flag is 1	≤	SCLTU, SCNC	C == 0	Less than/ C flag is 0	>
SCEQ, SCZ	Z == 1	Equal to/ Z flag is 1	=	SCNE, SCNZ	Z == 0	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

Syntax	Size	Processing Size	Operand dest	Code Size (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 compare until not equal* SCMPU

Syntax

String manipulation instruction

SCMPU

Instruction Code
Page: 233

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 user’s manual: hardware 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 *Set a flag or bit in the PSW* SETPSW

Syntax

System manipulation instruction

SETPSW dest

Instruction Code
Page: 234

Operation

dest = 1;

Function

- This instruction sets the O, S, Z, or C flag or the U or I bit specified by dest to 1.
- 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 right

SHAR

Syntax

Arithmetic/logic instruction

(1) SHAR src, dest

(2) SHAR src, src2, dest

Instruction Code
Page: 235

Operation

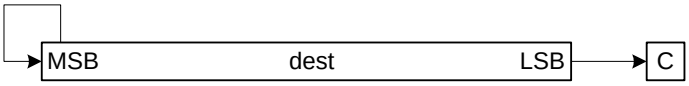
(1) dest = (signed long) dest >> (src & 31);

(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 $0 \leq src \leq 31$.
- 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 $0 \leq src \leq 31$.



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 Size	Operand			Code Size (Byte)
		src	src2	dest	
(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 shift left

SHLL

Syntax

Arithmetic/logic instruction

(1) SHLL src, dest

(2) SHLL src, src2, dest

Instruction Code
Page: 236

Operation

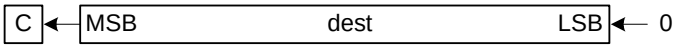
(1) dest = dest << (src & 31);

(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 src is 0, this flag is also cleared.

Instruction Format

Syntax	Processing Size	Operand			Code Size (Byte)
		src	src2	dest	
(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 right

SHLR

Syntax

Arithmetic/logic instruction

(1) SHLR src, dest

(2) SHLR src, src2, dest

Instruction Code
Page: 237

Operation

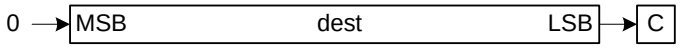
(1) dest = (unsigned long) dest >> (src & 31);

(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 $0 \leq src \leq 31$.
- 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 $0 \leq src \leq 31$.



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

Syntax	Processing Size	Operand			Code Size (Byte)
		src	src2	dest	
(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

String move backward

SMOVB

Syntax

String manipulation instruction

SMOVB

Instruction Code

Page: 238

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 user's manual: hardware 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

String move forward

SMOVF

Syntax

String manipulation instruction

SMOVF

Instruction Code

Page: 238

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 user's manual: hardware 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 *String move until zero detected* SMOVU

Syntax

String manipulation instruction

SMOVU

Instruction Code
Page: 238

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 user’s manual: hardware 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 *String store* SSTR

Syntax

String manipulation instruction

SSTR.size

Instruction Code

Page: 239

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

Store on not zero

STNZ

Syntax

Data transfer instruction

STNZ src, dest

Instruction Code

Page: 239

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

Description Example

STNZ #1, R2

STZ

Store on zero

STZ

Syntax

Data transfer instruction

STZ src, dest

Instruction Code
Page: 240

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

Syntax	Processing Size	Operand		Code Size (Byte)
		src	dest	
STZ src, dest	L	#SIMM:8	Rd	4
	L	#SIMM:16	Rd	5
	L	#SIMM:24	Rd	6
	L	#IMM:32	Rd	7

Description Example

STZ #1, R2

SUB

Subtract without borrow

SUB

Syntax

Arithmetic/logic instruction

(1) SUB src, dest

(2) SUB src, src2, dest

Instruction Code
Page: 241

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

Syntax		Processing Size	Operand			Code Size (Byte)
			src	src2	dest	
(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

String search until equal

SUNTIL

Syntax

String manipulation instruction

SUNTIL.size

Instruction Code

Page: 242

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 user's manual: hardware 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 *String search while equal* SWHILE

Syntax

String manipulation instruction

SWHILE.size

Instruction Code
Page: 242

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 user's manual: hardware 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

Test logical

TST

Syntax

Arithmetic/logic instruction

TST src, src2

Instruction Code
Page: 243

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

Syntax	Processing Size	Operand src	Operand src2	Code Size (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
```

WAIT

Wait

WAIT

Syntax

System manipulation instruction

WAIT

Instruction Code

Page: 244

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 user's manual: hardware 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

Exchange

XCHG

Syntax

Data transfer instruction

XCHG src, dest

Instruction Code
Page: 244

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 user's manual: hardware of each product.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Processing Size	Operand src	Operand dest	Code Size (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

XOR

Syntax

Arithmetic/logic instruction

XOR src, dest

Instruction Code
Page: 245

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.

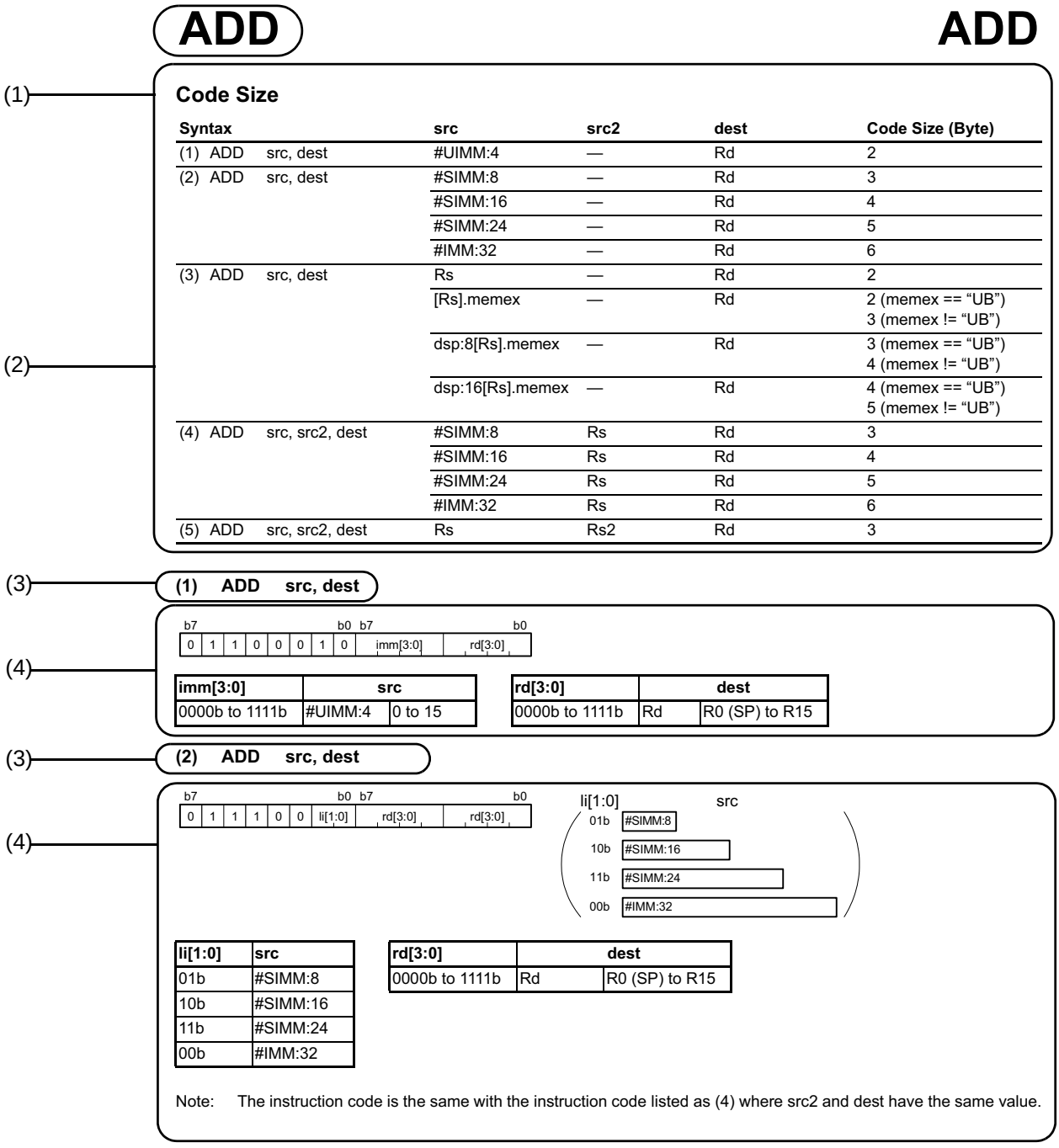
Description Example

```
XOR   #8, R1
XOR   R1, R2
XOR   [R1], R2
XOR   16[R1].L, R2
```

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.



Code Size

Syntax		src	src2	dest	Code Size (Byte)
(1) ADD	src, dest	#UIMM:4	—	Rd	2
(2) ADD	src, dest	#SIMM:8	—	Rd	3
		#SIMM:16	—	Rd	4
		#SIMM:24	—	Rd	5
		#IMM:32	—	Rd	6
(3) 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”)
(4) ADD	src, src2, dest	#SIMM:8	Rs	Rd	3
		#SIMM:16	Rs	Rd	4
		#SIMM:24	Rs	Rd	5
		#IMM:32	Rs	Rd	6
(5) ADD	src, src2, dest	Rs	Rs2	Rd	3

(1) ADD 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

(2) ADD 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

Note: The instruction code is the same with the instruction code listed as (4) where src2 and dest have the same value.

(1) Mnemonic

Indicates the mnemonic name of the instruction explained on the given page.

(2) List of Code Size

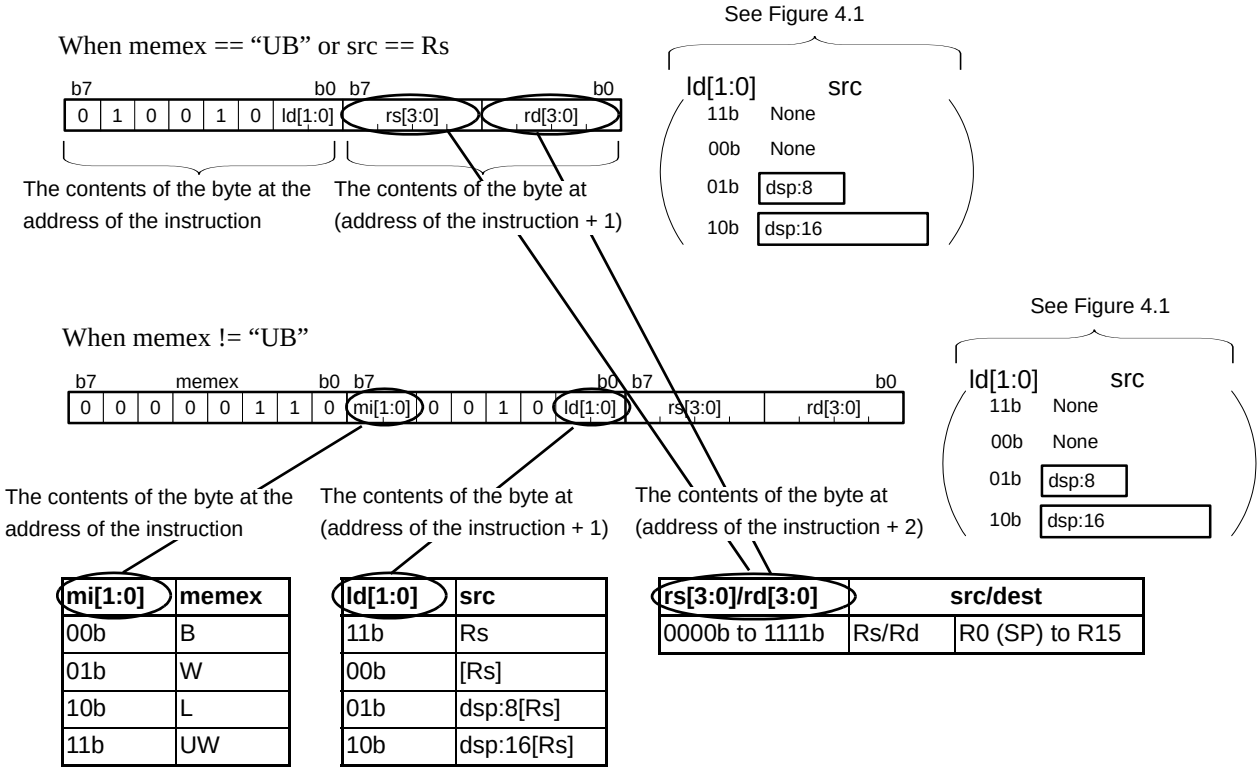
Indicates the number of bytes the instruction requires. An individual RXv1 CPU instruction takes up from one to eight bytes.

(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.



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.

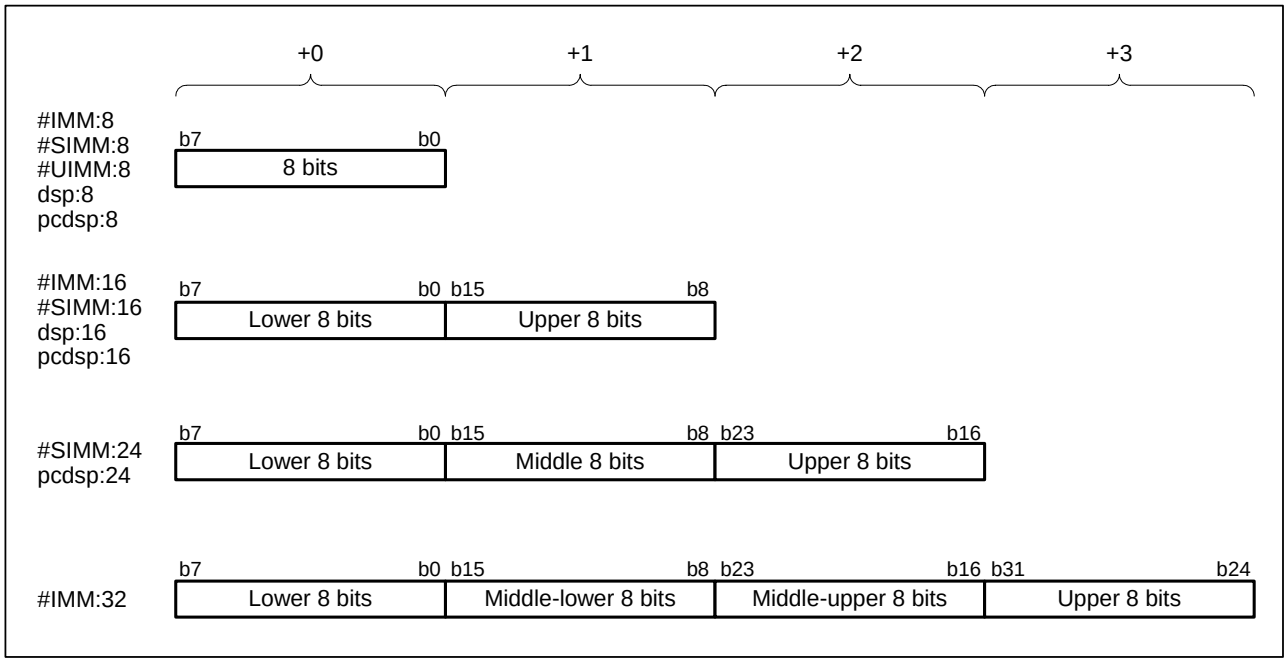


Figure 4.1 Immediate (IMM) and Displacement (dsp) Values

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

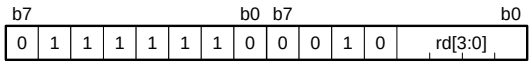
The following pages give details of the instruction codes for the RXv1 instructions.

ABS ABS

Code Size

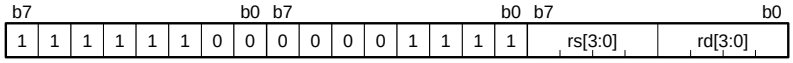
Syntax		src	dest	Code Size (Byte)
(1) ABS	dest	—	Rd	2
(2) ABS	src, dest	Rs	Rd	3

(1) ABS dest



rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) ABS src, dest



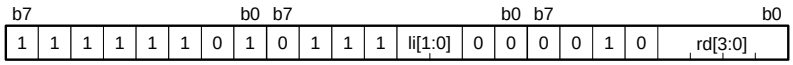
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

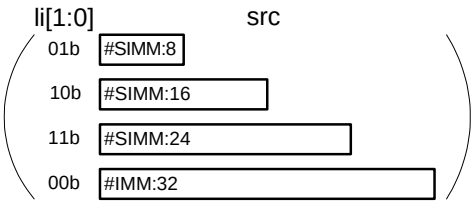
ADC ADC

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

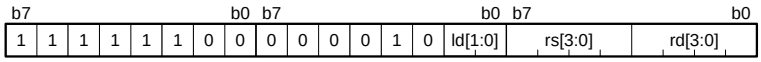




li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

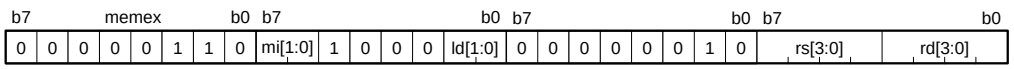
(2) ADC src, dest

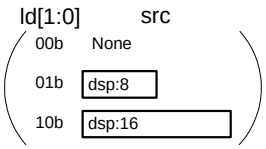


ld[1:0]	src
11b	Rs

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

(3) ADC src, dest





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

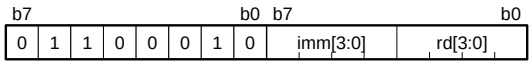
ADD

ADD

Code Size

Syntax		src	src2	dest	Code Size (Byte)
(1) ADD	src, dest	#UIMM:4	—	Rd	2
(2) ADD	src, dest	#SIMM:8	—	Rd	3
		#SIMM:16	—	Rd	4
		#SIMM:24	—	Rd	5
		#IMM:32	—	Rd	6
(3) 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”)
(4) ADD	src, src2, dest	#SIMM:8	Rs	Rd	3
		#SIMM:16	Rs	Rd	4
		#SIMM:24	Rs	Rd	5
		#IMM:32	Rs	Rd	6
(5) ADD	src, src2, dest	Rs	Rs2	Rd	3

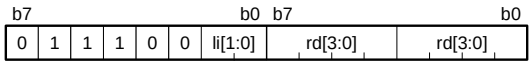
(1) ADD src, dest

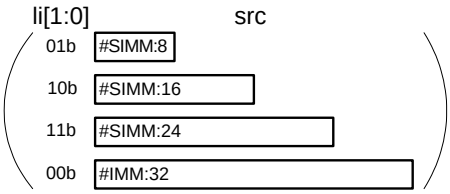


imm[3:0]	src	
0000b to 1111b	#UIMM:4	0 to 15

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) ADD src, dest





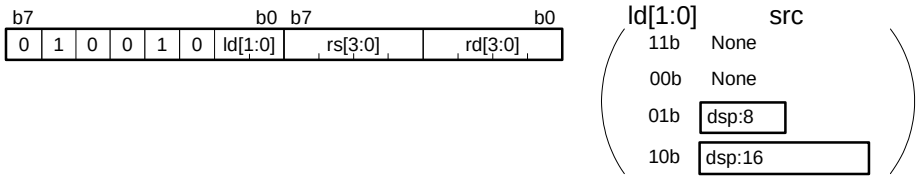
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

Note: The instruction code is the same with the instruction code listed as (4) where src2 and dest have the same value.

(3) ADD src, dest

When memex == “UB” or src == Rs



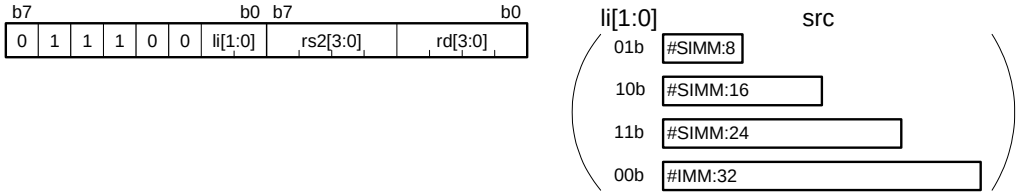
When memex != “UB”

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

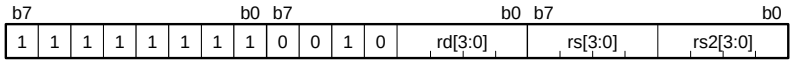
(4) ADD src, src2, dest



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

(5) ADD 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

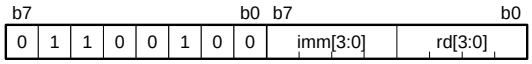
AND

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

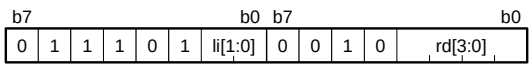
(1) AND src, dest

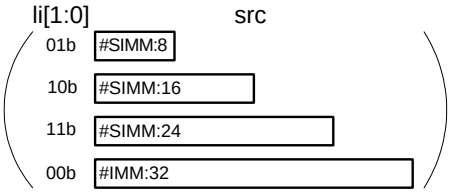


imm[3:0]	src	
0000b to 1111b	#UIMM:4	0 to 15

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) AND src, dest



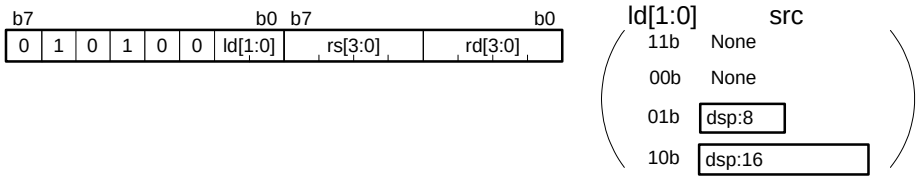


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

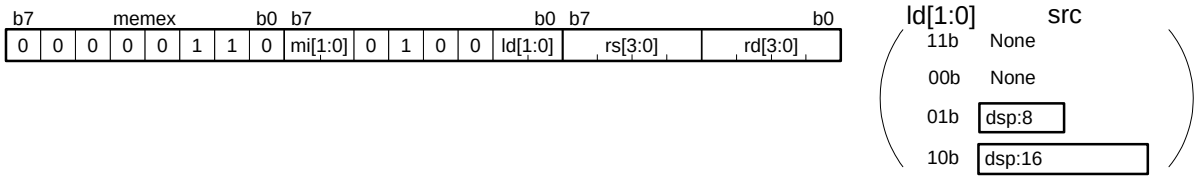
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(3) AND src, dest

When memex == “UB” or src == Rs



When memex != “UB”

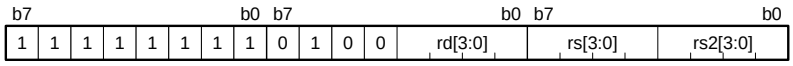


mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

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



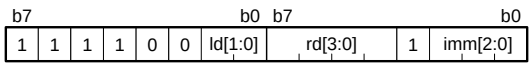
rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b to 1111b	Rs/Rs2/Rd	R0 (SP) to R15

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

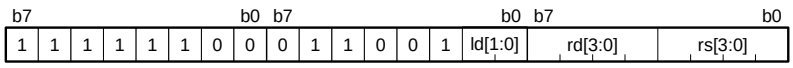


ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

imm[2:0]	src	
000b to 111b	#IMM:3	0 to 7

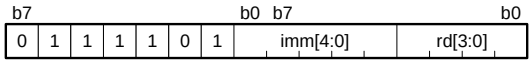
(2) BCLR src, dest



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

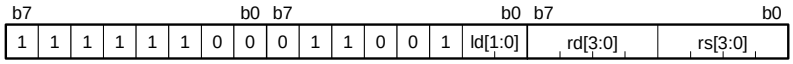
(3) BCLR src, dest



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



ld[1:0]	dest
11b	Rd

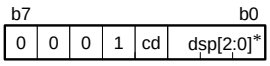
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

B*Cnd* B*Cnd*

Code Size

Syntax	src	Code Size (Byte)
(1) B*Cnd*.S src	pcdsp:3	1
(2) B*Cnd*.B src	pcdsp:8	2
(3) B*Cnd*.W src	pcdsp:16	3

(1) B*Cnd*.S src

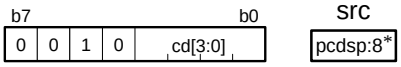


Note: * dsp[2:0] specifies pcdsp:3 = src.

cd	B*Cnd*
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

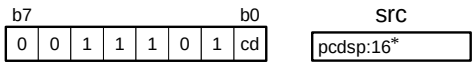
(2) B*Cnd*.B src



Note: * Address indicated by pcdsp:8 = src minus the address of the instruction

cd[3:0]	B*Cnd*	cd[3:0]	B*Cnd*
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



Note: * Address indicated by pcdsp:16 = src minus the address of the instruction

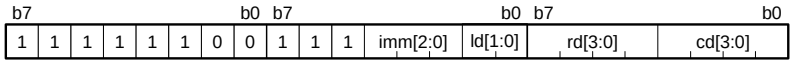
cd	B*Cnd*
0b	BEQ, BZ
1b	BNE, BNZ

BMC*nd* BMC*nd*

Code Size

Syntax		src	dest	Code Size (Byte)
(1) BMC*nd*	src, dest	#IMM:3	[Rd].B	3
		#IMM:3	dsp:8[Rd].B	4
		#IMM:3	dsp:16[Rd].B	5
(2) BMC*nd*	src, dest	#IMM:5	Rd	3

(1) BMC*nd* src, dest



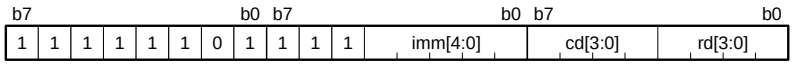
imm[2:0]	src	
000b to 111b	#IMM:3	0 to 7

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

cd[3:0]	BMC*nd*	cd[3:0]	BMC*nd*
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) BMC*nd* src, dest



imm[4:0]	src	
00000b to 11111b	#IMM:5	0 to 31

cd[3:0]	BMC*nd*	cd[3:0]	BMC*nd*
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

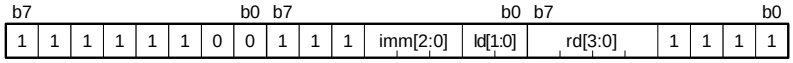
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

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

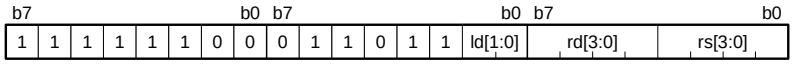


imm[2:0]	src	
000b to 111b	#IMM:3	0 to 7

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

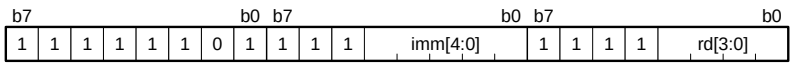
(2) BNOT src, dest



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

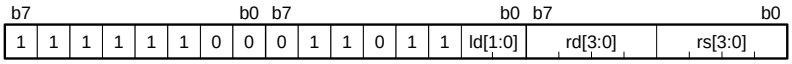
(3) BNOT src, dest



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



ld[1:0]	dest
11b	Rd

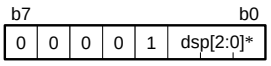
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

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

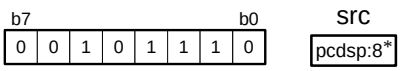
(1) BRA.S src



Note: * dsp[2:0] specifies pcdsp:3 = src.

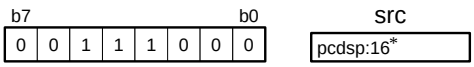
dsp[2:0]	Branch Distance
011b	3
100b	4
101b	5
110b	6
111b	7
000b	8
001b	9
010b	10

(2) BRA.B src



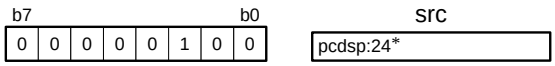
Note: * Address indicated by pcdsp:8 = src minus the address of the instruction

(3) BRA.W src



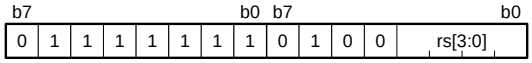
Note: * Address indicated by pcdsp:16 = src minus the address of the instruction

(4) BRA.A src



Note: * Address indicated by pcdsp:24 = src minus the address of the instruction

(5) BRA.L src



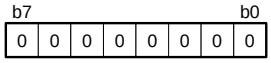
rs[3:0]	src	
0000b to 1111b	Rs	R0 (SP) to R15

BRK BRK

Code Size

Syntax	Code Size (Byte)
(1) BRK	1

(1) BRK

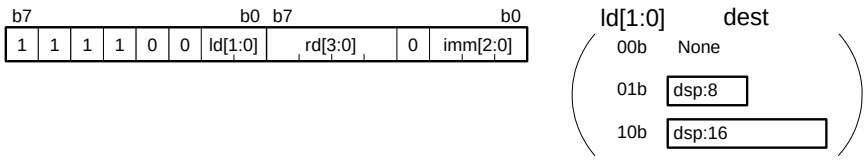


BSET BSET

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



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

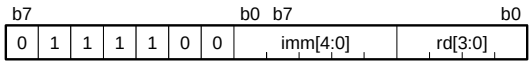
imm[2:0]	src	
000b to 111b	#IMM:3	0 to 7

(2) BSET src, dest

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

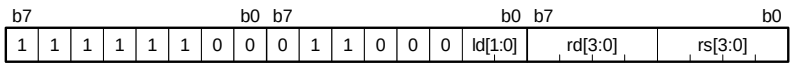
(3) BSET src, dest



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



ld[1:0]	dest
11b	Rd

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

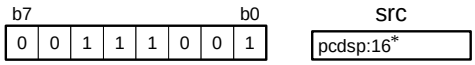
BSR

BSR

Code Size

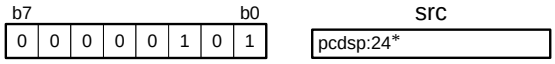
Syntax	src	Code Size (Byte)
(1) BSR.W src	pcdsp:16	3
(2) BSR.A src	pcdsp:24	4
(3) BSR.L src	Rs	2

(1) BSR.W src



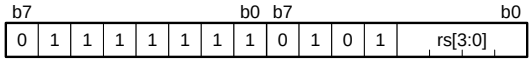
Note: * Address indicated by pcdsp:16 = src minus the address of the instruction

(2) BSR.A src



Note: * Address indicated by pcdsp:24 = src minus the address of the instruction

(3) BSR.L src



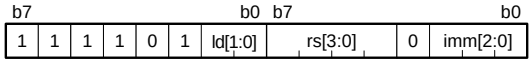
rs[3:0]	src	
0000b to 1111b	Rs	R0 (SP) to R15

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

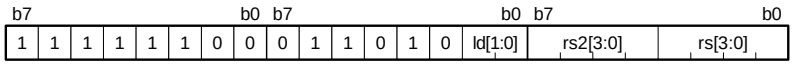


ld[1:0]	src2
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]	src2	
0000b to 1111b	Rs	R0 (SP) to R15

imm[2:0]	src	
000b to 111b	#IMM:3	0 to 7

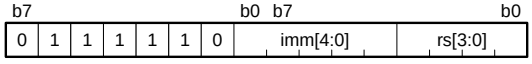
(2) BTST src, src2



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

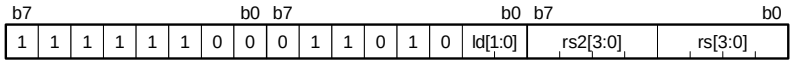
(3) BTST src, src2



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



ld[1:0]	src2
11b	Rs2

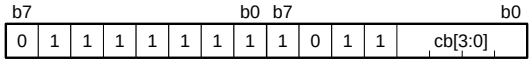
rs[3:0]/rs2[3:0]	src/src2	
0000b to 1111b	Rs/Rs2	R0 (SP) to R15

CLRPSW CLRPSW

Code Size

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

(1) CLRPSW dest



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

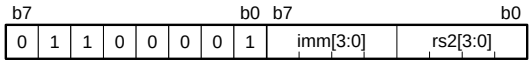
CMP

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”)

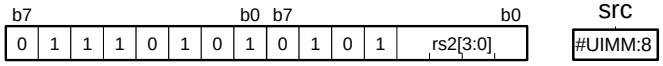
(1) CMP src, src2



imm[3:0]	src	
0000b to 1111b	#UIMM:4	0 to 15

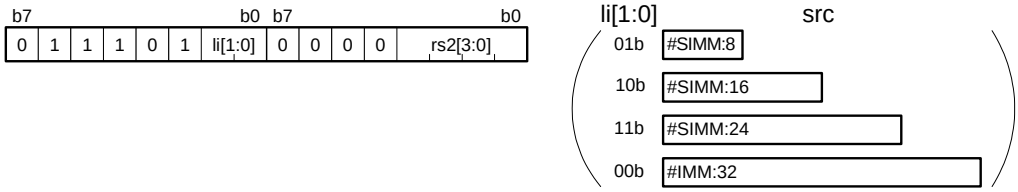
rs2[3:0]	src2	
0000b to 1111b	Rs	R0 (SP) to R15

(2) CMP src, src2



rs2[3:0]	src2	
0000b to 1111b	Rs	R0 (SP) to R15

(3) CMP src, src2

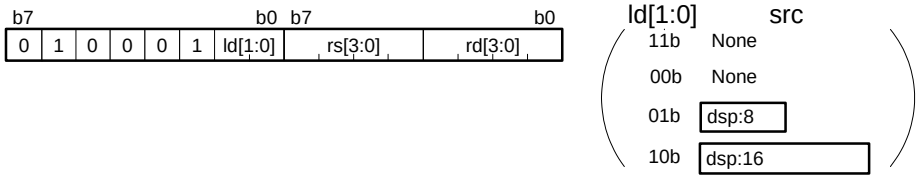


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

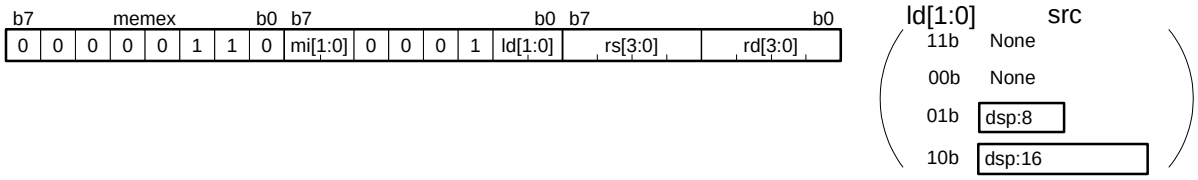
rs2[3:0]	src2	
0000b to 1111b	Rs	R0 (SP) to R15

(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]

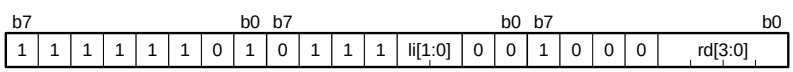
rs[3:0]/rs2[3:0]	src/src2	
0000b to 1111b	Rs/Rs2	R0 (SP) to R15

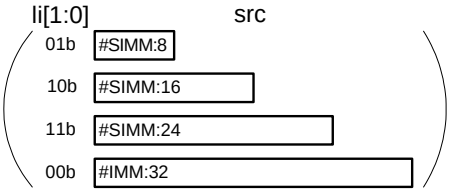
DIV DIV

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



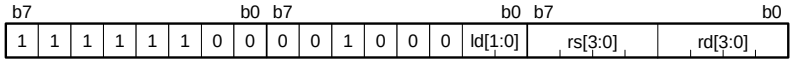


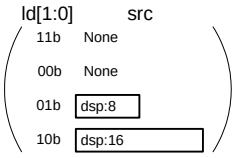
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

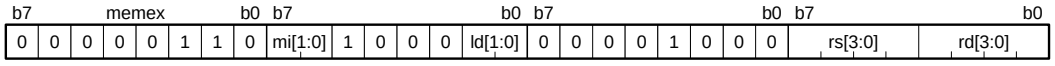
(2) DIV src, dest

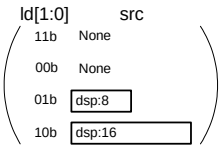
When memex == “UB” or src == Rs





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

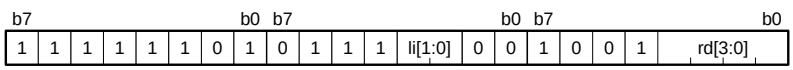
DIVU

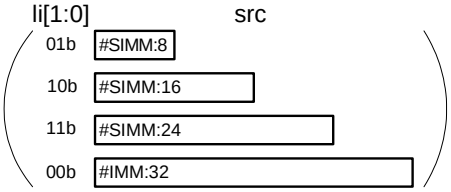
DIVU

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



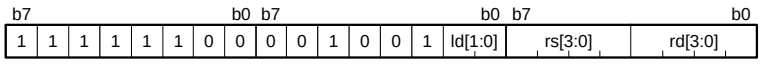


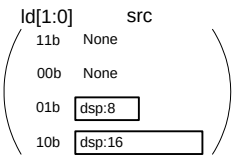
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

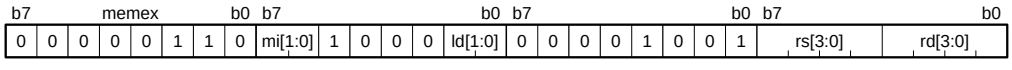
(2) DIVU src, dest

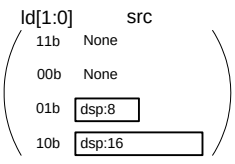
When memex == “UB” or src == Rs





When memex != “UB”





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

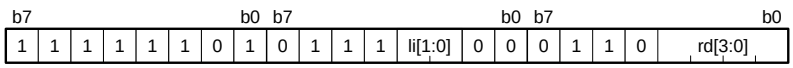
EMUL

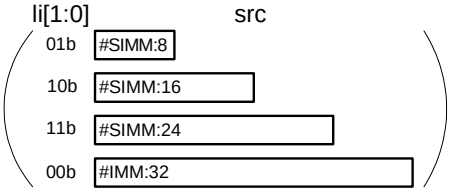
EMUL

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



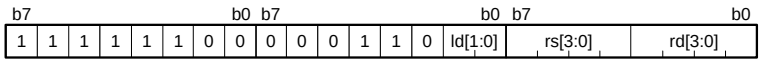


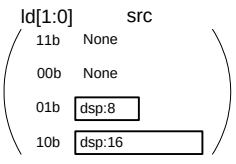
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

rd[3:0]	dest	
0000b to 1110b	Rd	R0 (SP) to R14

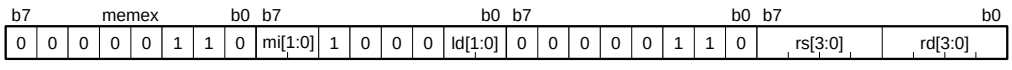
(2) EMUL src, dest

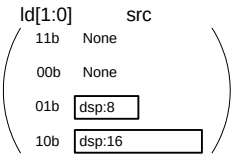
When memex == “UB” or src == Rs





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

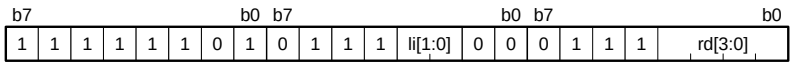
EMULU

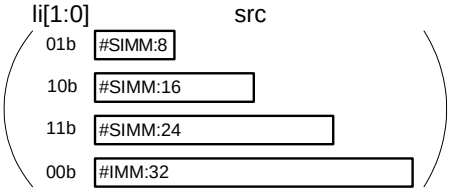
EMULU

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



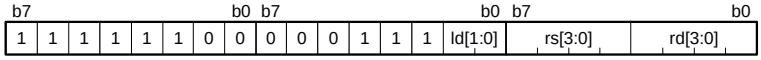


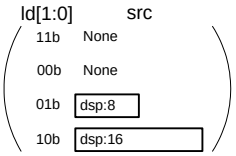
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

rd[3:0]	dest	
0000b to 1110b	Rd	R0 (SP) to R14

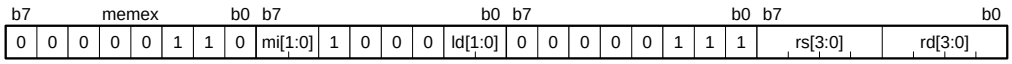
(2) EMULU src, dest

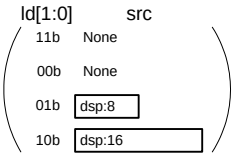
When memex == “UB” or src == Rs





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

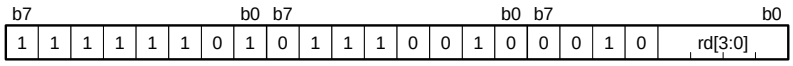
FADD

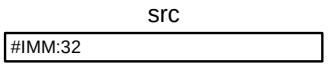
FADD

Code Size

Syntax	src	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

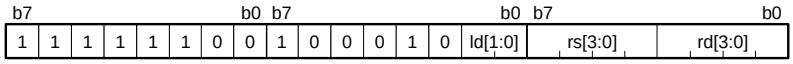
(1) FADD src, dest

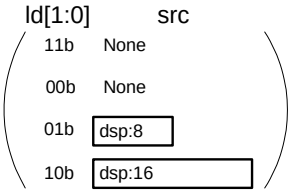




rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) FADD 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

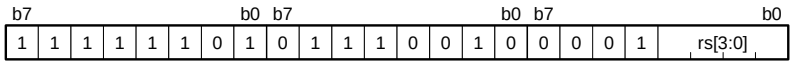
FCMP

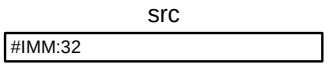
FCMP

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

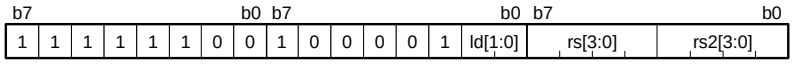
(1) FCMP src, src2

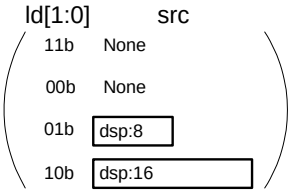




rs[3:0]	src2	
0000b to 1111b	Rs	R0 (SP) to R15

(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

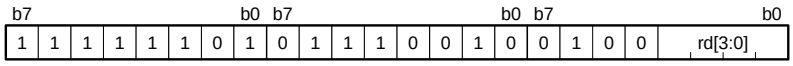
FDIV

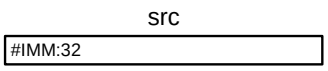
FDIV

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

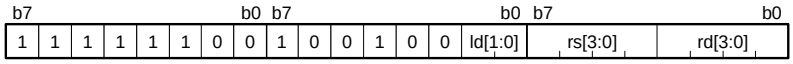
(1) FDIV src, dest

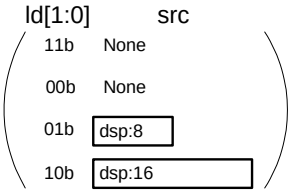




rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) FDIV 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

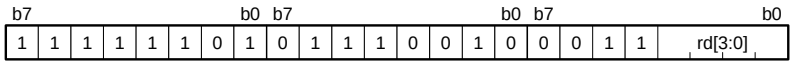
FMUL

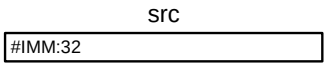
FMUL

Code Size

Syntax	src	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

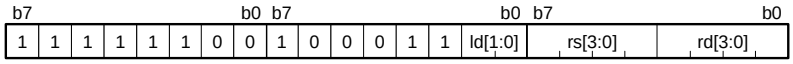
(1) FMUL src, dest

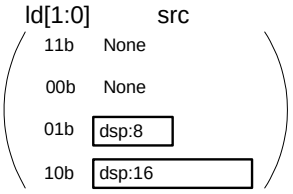




rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) FMUL 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

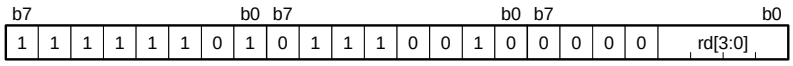
FSUB

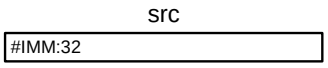
FSUB

Code Size

Syntax	src	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

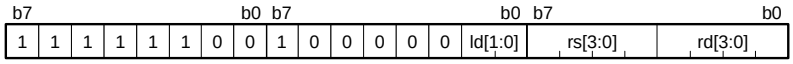
(1) FSUB src, dest

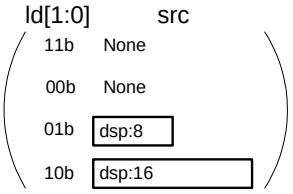




rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) FSUB 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

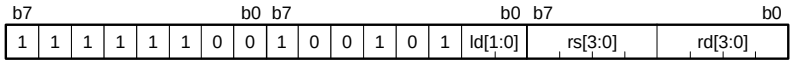
FTOI

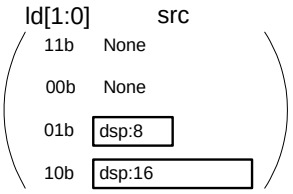
FTOI

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

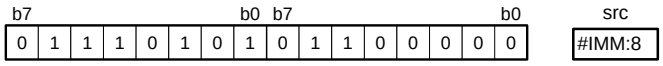
INT

INT

Code Size

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

(1) INT src



ITOF

ITOF

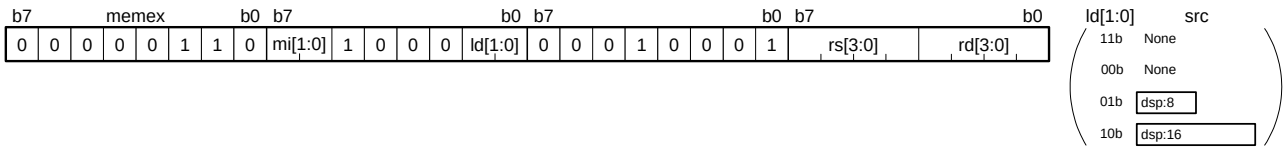
Code Size

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

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

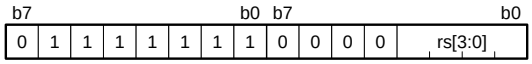
JMP

JMP

Code Size

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

(1) JMP src



rs[3:0]	src	
0000b to 1111b	Rs	R0 (SP) to R15

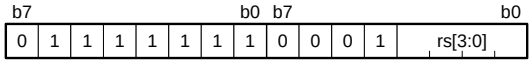
JSR

JSR

Code Size

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

(1) JSR src



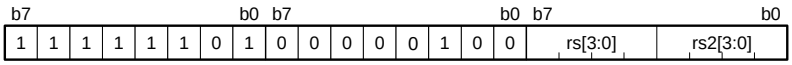
rs[3:0]	src	
0000b to 1111b	Rs	R0 (SP) to R15

MACHI MACHI

Code Size

Syntax	src	src2	Code Size (Byte)
(1) MACHI src, src2	Rs	Rs2	3

(1) MACHI src, src2



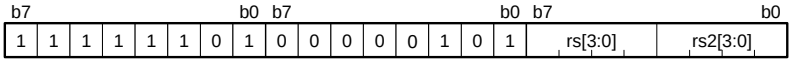
rs[3:0]/rs2[3:0]	src/src2	
0000b to 1111b	Rs/Rs2	R0 (SP) to R15

MACLO MACLO

Code Size

Syntax	src	src2	Code Size (Byte)
(1) MACLO src, src2	Rs	Rs2	3

(1) MACLO src, src2



rs[3:0]/rs2[3:0]	src/src2	
0000b to 1111b	Rs/Rs2	R0 (SP) to R15

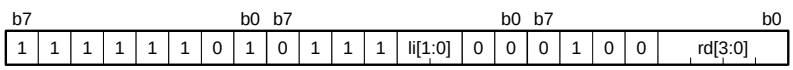
MAX

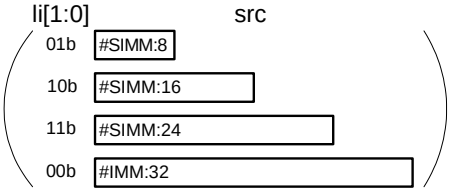
MAX

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



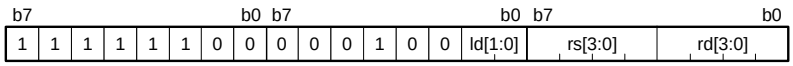


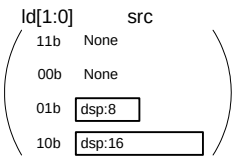
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

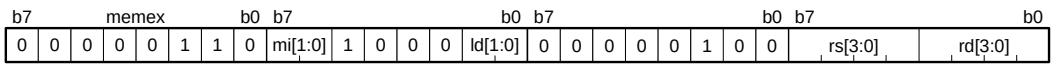
(2) MAX src, dest

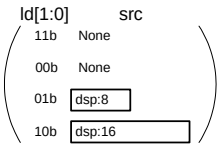
When memex == “UB” or src == Rs





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]

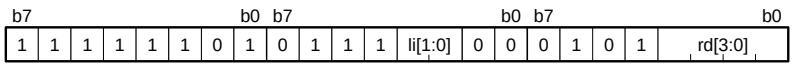
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

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

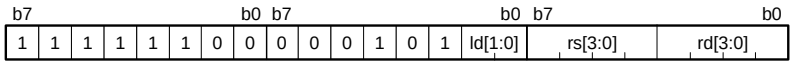


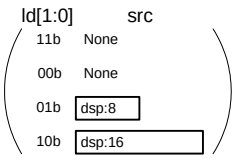
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

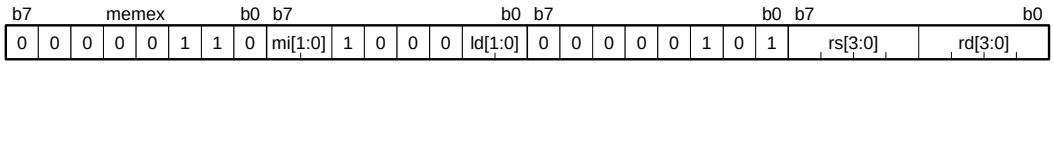
(2) MIN src, dest

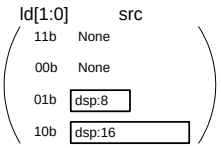
When memex == “UB” or src == Rs





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

MOV

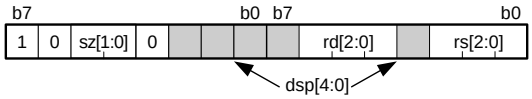
MOV

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

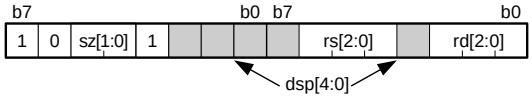


sz[1:0]	Size
00b	B
01b	W
10b	L

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

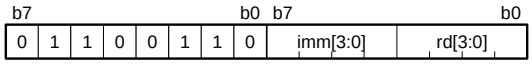


sz[1:0]	Size
00b	B
01b	W
10b	L

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

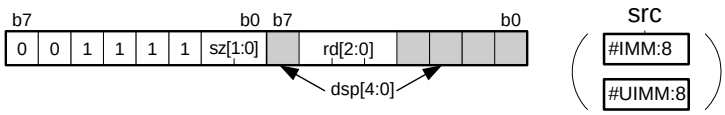
(3) MOV.size src, dest



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

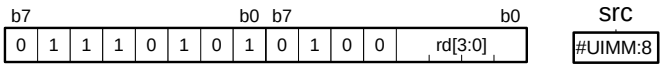


sz[1:0]	Size
00b	B
01b	W
10b	L

dsp[4:0]	dsp:5
00000b to 11111b	0 to 31

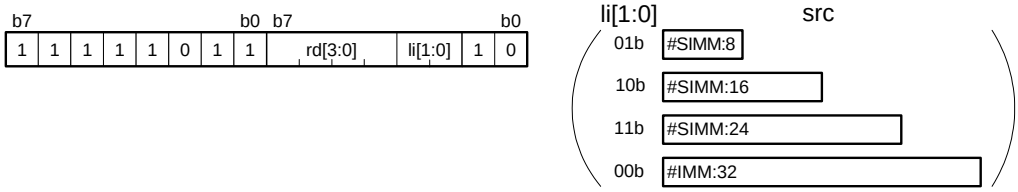
rd[2:0]	dest	
000b to 111b	Rd	R0 (SP) to R7

(5) MOV.size src, dest



rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

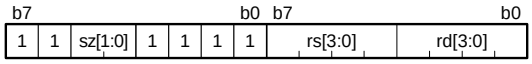
(6) MOV.size src, dest



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

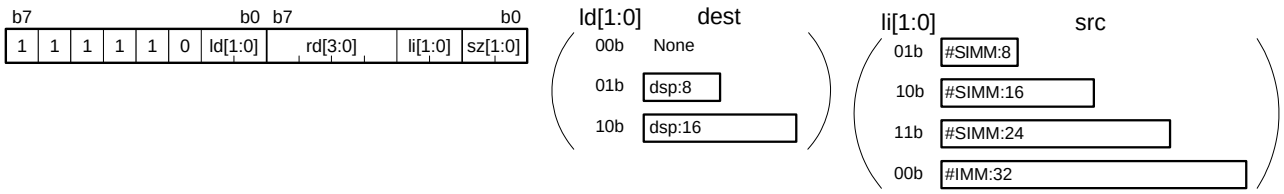
(7) MOV.size src, dest



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



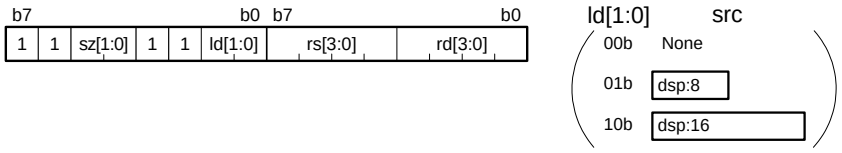
ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

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

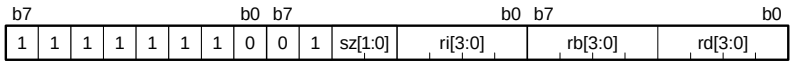


sz[1:0]	Size
00b	B
01b	W
10b	L

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

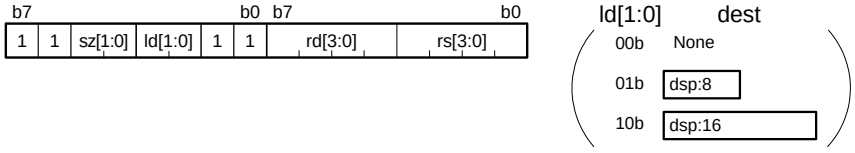
(10) MOV.size src, dest



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

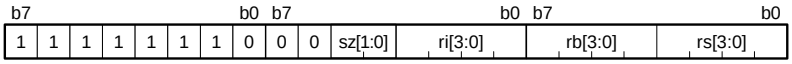


sz[1:0]	Size
00b	B
01b	W
10b	L

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

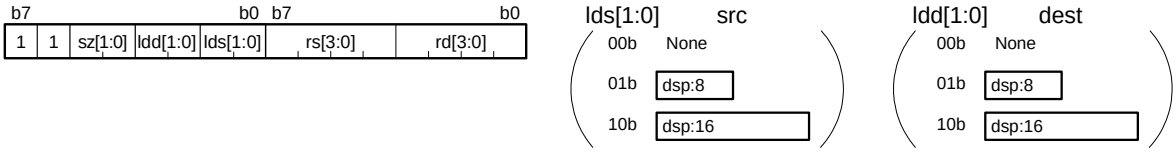
(12) MOV.size src, dest



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

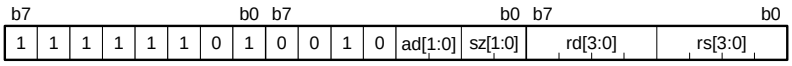


sz[1:0]	Size
00b	B
01b	W
10b	L

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

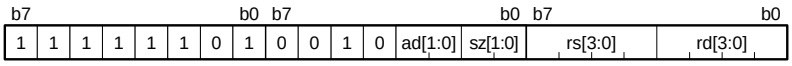


ad[1:0]	Addressing
00b	Rs, [Rd+]
01b	Rs, [-Rd]

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



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

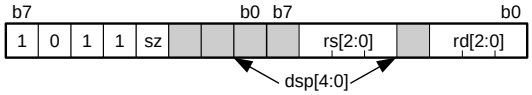
MOVU

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

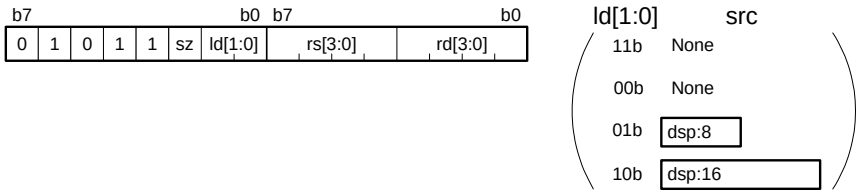


sz	Size
0b	B
1b	W

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

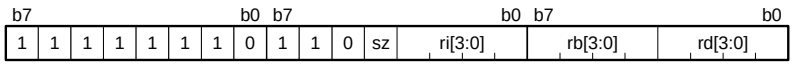


sz	Size
0b	B
1b	W

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

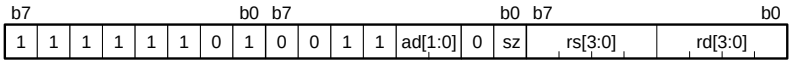
(3) MOVU.size src, dest



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



ad[1:0]	Addressing
10b	[Rs+], Rd
11b	[-Rs], Rd

sz	Size
0b	B
1b	W

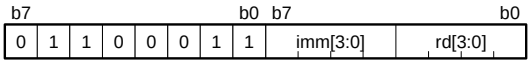
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

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

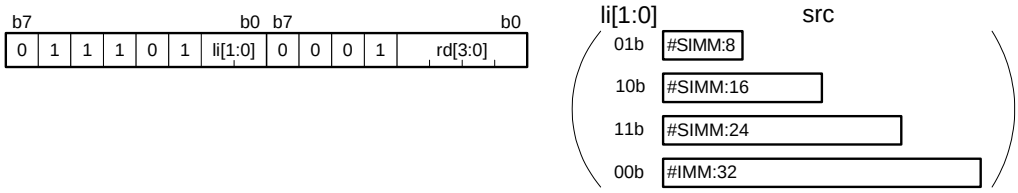
(1) MUL src, dest



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

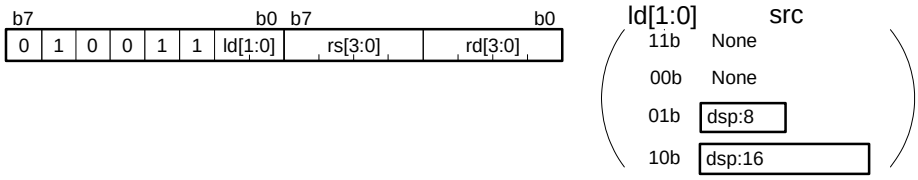


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

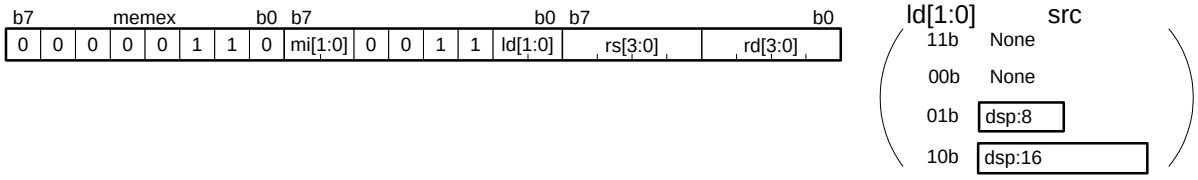
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(3) MUL src, dest

When memex == “UB” or src == Rs



When memex != “UB”

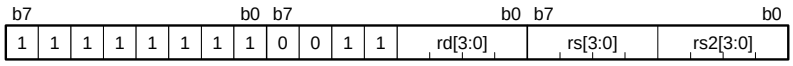


mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

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



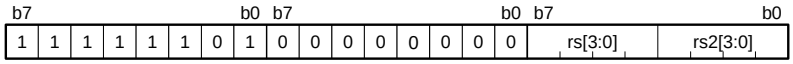
rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b to 1111b	Rs/Rs2/Rd	R0 (SP) to R15

MULHI MULHI

Code Size

Syntax	src	src2	Code Size (Byte)
(1) MULHI src, src2	Rs	Rs2	3

(1) MULHI src, src2



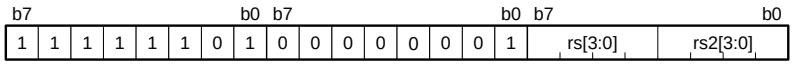
rs[3:0]/rs2[3:0]	src/src2	
0000b to 1111b	Rs/Rs2	R0 (SP) to R15

MULLO MULLO

Code Size

Syntax	src	src2	Code Size (Byte)
(1) MULLO src, src2	Rs	Rs2	3

(1) MULLO src, src2



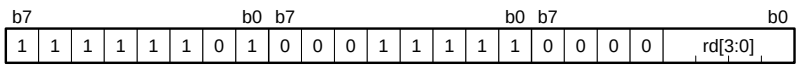
rs[3:0]/rs2[3:0]	src/src2	
0000b to 1111b	Rs/Rs2	R0 (SP) to R15

MVFACHI MVFACHI

Code Size

Syntax	dest	Code Size (Byte)
(1) MVFACHI dest	Rd	3

(1) MVFACHI dest



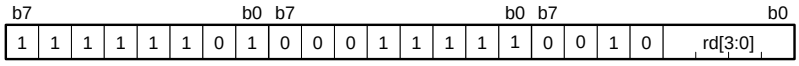
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

MVFACMI MVFACMI

Code Size

Syntax	dest	Code Size (Byte)
(1) MVFACMI dest	Rd	3

(1) MVFACMI dest



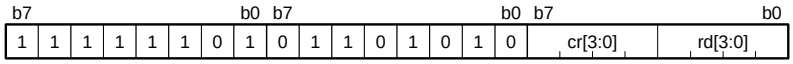
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

MVFC MVFC

Code Size

Syntax	src	dest	Code Size (Byte)
(1) MVFC src, dest	Rx	Rd	3

(1) MVFC src, dest



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 to 1111b		Reserved

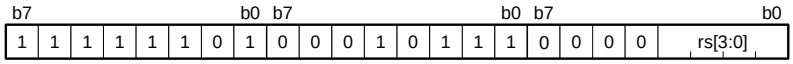
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

MVTACHI MVTACHI

Code Size

Syntax	src	Code Size (Byte)
(1) MVTACGU src	Rs	3

(1) MVTACHI src



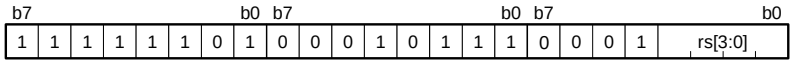
rs[3:0]	src	
0000b to 1111b	Rs	R0 (SP) to R15

MVTACLO MVTACLO

Code Size

Syntax	src	Code Size (Byte)
(1) MVTACLO src	Rs	3

(1) MVTACLO src



rs[3:0]	src	
0000b to 1111b	Rs	R0 (SP) to R15

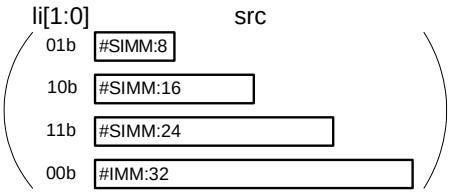
MVTC

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

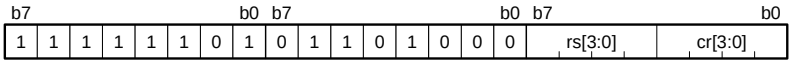
(1) MVTC src, dest



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 to 1111b		Reserved

(2) MVTC src, dest



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 to 1111b		Reserved

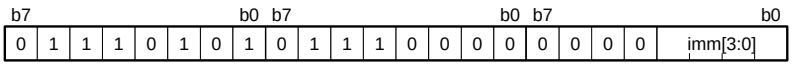
rs[3:0]	src	
0000b to 1111b	Rs	R0 (SP) to R15

MVTIPL MVTIPL

Code Size

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

(1) MVTIPL src



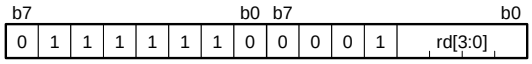
imm[3:0]	#IMM:4
0000b to 1111b	0 to 15

NEG NEG

Code Size

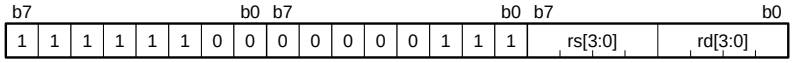
Syntax		src	dest	Code Size (Byte)
(1) NEG	dest	−	Rd	2
(2) NEG	src, dest	Rs	Rd	3

(1) NEG dest



rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) NEG src, dest



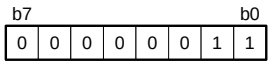
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

NOP NOP

Code Size

Syntax	Code Size (Byte)
(1) NOP	1

(1) NOP

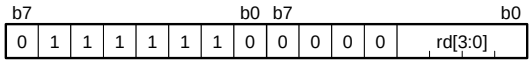


NOT NOT

Code Size

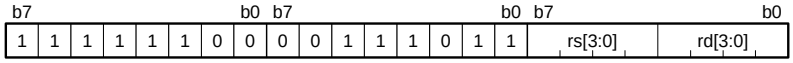
Syntax		src	dest	Code Size (Byte)
(1) NOT	dest	−	Rd	2
(2) NOT	src, dest	Rs	Rd	3

(1) NOT dest



rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) NOT src, dest



rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

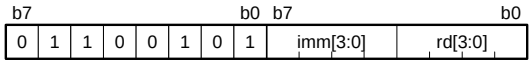
OR

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

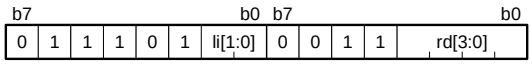
(1) OR src, dest

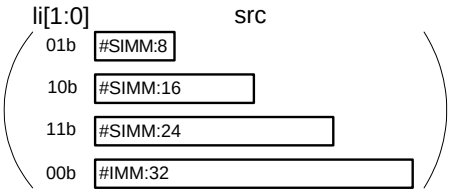


imm[3:0]	src	
0000b to 1111b	#UIMM:4	0 to 15

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) OR src, dest



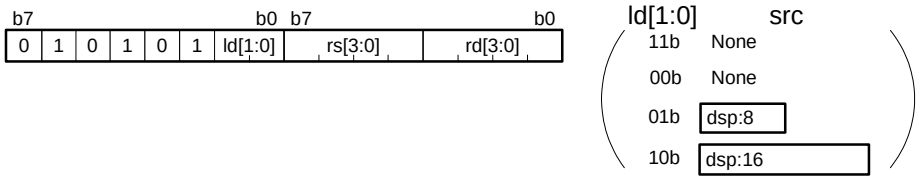


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

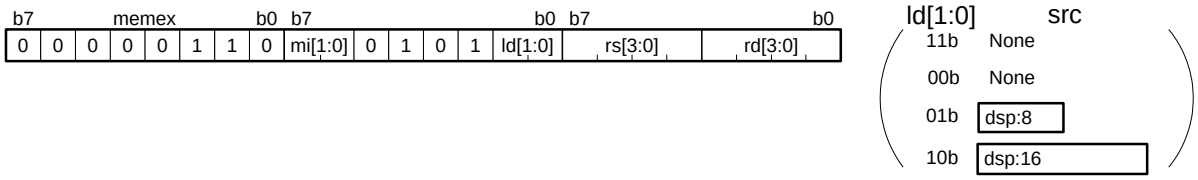
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(3) OR src, dest

When memex == “UB” or src == Rs



When memex != “UB”

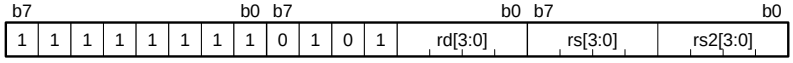


mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

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) OR 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

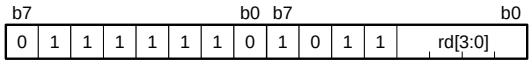
POP

POP

Code Size

Syntax	dest	Code Size (Byte)
(1) POP dest	Rd	2

(1) POP dest



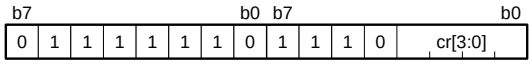
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

POPC POPC

Code Size

Syntax	dest	Code Size (Byte)
(1) POPC dest	Rx	2

(1) POPC dest



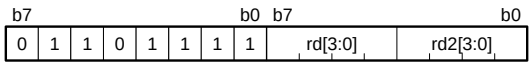
cr[3:0]	dest	
0000b	Rx	PSW
0001b		Reserved
0010b		USP
0011b		FPSW
0100b		Reserved
0101b		Reserved
0110b		Reserved
0111b		Reserved
1000b		BPSW
1001b		BPC
1010b		ISP
1011b		FINTV
1100b		INTB
1101b to 1111b		Reserved

POPM POPM

Code Size

Syntax	dest	dest2	Code Size (Byte)
(1) POPM dest-dest2	Rd	Rd2	2

(1) POPM dest-dest2



rd[3:0]	dest	
0001b to 1110b	Rd	R1 to R14

rd2[3:0]	dest2	
0010b to 1111b	Rd2	R2 to R15

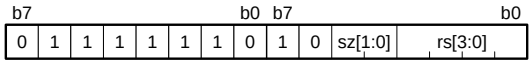
PUSH

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

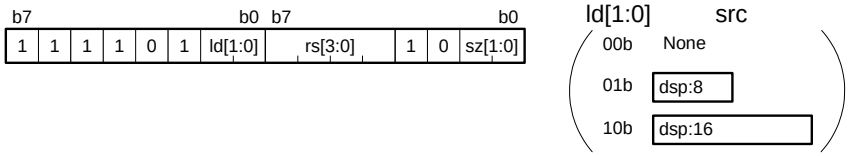
(1) PUSH.size src



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

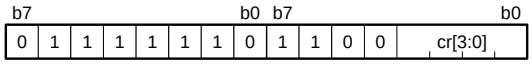
PUSHC

PUSHC

Code Size

Syntax	src	Code Size (Byte)
(1) PUSHC src	Rx	2

(1) PUSHC src



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 to 1111b		Reserved

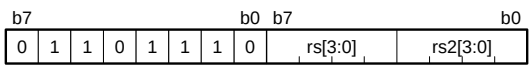
PUSHM

PUSHM

Code Size

Syntax	src	src2	Code Size (Byte)
(1) PUSHM src-src2	Rs	Rs2	2

(1) PUSHM src-src2



rs[3:0]	src	
0001b to 1110b	Rs	R1 to R14

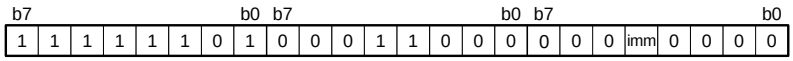
rs2[3:0]	src2	
0010b to 1111b	Rs2	R2 to R15

RACW RACW

Code Size

Syntax	src	Code Size (Byte)
(1) RACW src	#IMM:1 (IMM:1 = 1, 2)	3

(1) RACW src



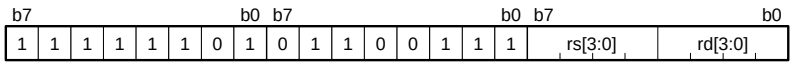
imm	src	
0b, 1b	#IMM:1	1, 2

REVL REVL

Code Size

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

(1) REVL src, dest



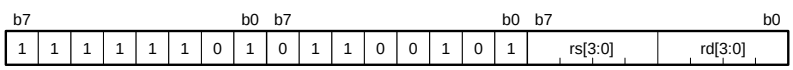
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

REVW REVW

Code Size

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

(1) REVW src, dest



rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

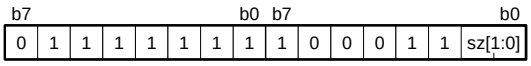
RMPA

RMPA

Code Size

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

(1) RMPA.size



sz[1:0]	Size
00b	B
01b	W
10b	L

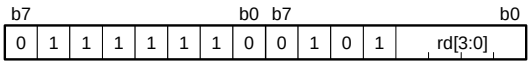
ROLC

ROLC

Code Size

Syntax	dest	Code Size (Byte)
(1) ROLC dest	Rd	2

(1) ROLC dest



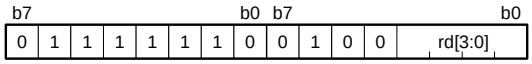
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

RORC RORC

Code Size

Syntax	dest	Code Size (Byte)
(1) RORC dest	Rd	2

(1) RORC dest



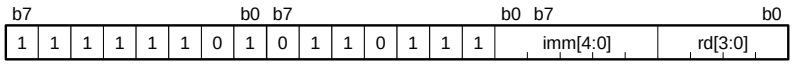
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

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

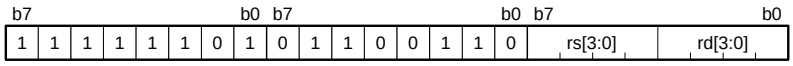
(1) ROTL src, dest



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



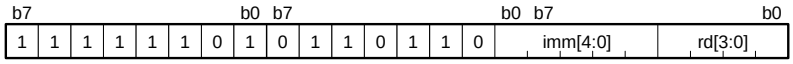
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

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

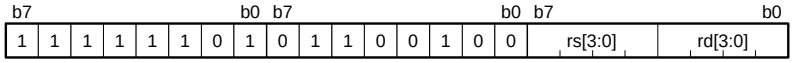
(1) ROTR src, dest



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



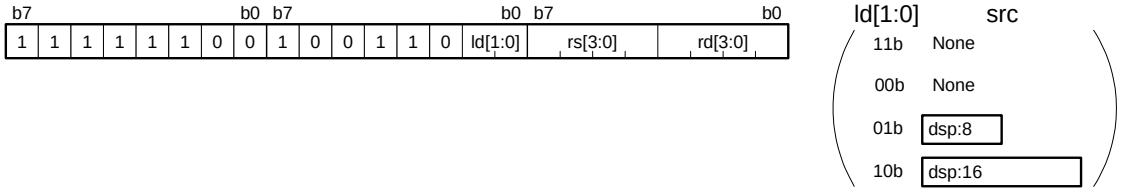
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

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



ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

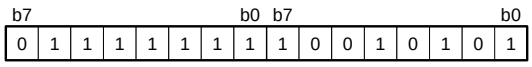
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

RTE RTE

Code Size

Syntax	Code Size (Byte)
(1) RTE	2

(1) RTE



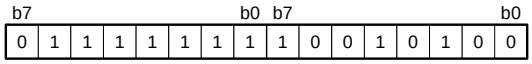
RTFI

RTFI

Code Size

Syntax	Code Size (Byte)
(1) RTFI	2

(1) RTFI



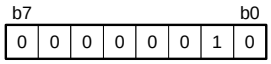
RTS

RTS

Code Size

Syntax	Code Size (Byte)
(1) RTS	1

(1) RTS



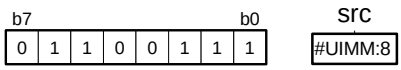
RTSD

RTSD

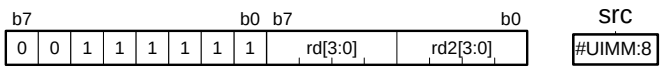
Code Size

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



(2) RTSD src, dest-dest2



rd[3:0]/rd2[3:0]	dest/dest2	
0001b to 1111b	Rd/Rd2	R1 to R15

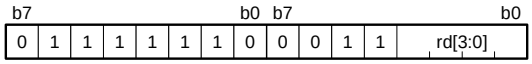
SAT

SAT

Code Size

Syntax	dest	Code Size (Byte)
(1) SAT dest	Rd	2

(1) SAT dest



rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

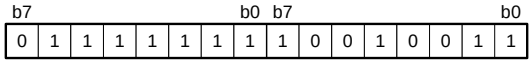
SATR

SATR

Code Size

Syntax	Code Size (Byte)
(1) SATR	2

(1) SATR

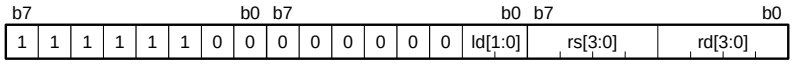


SBB SBB

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

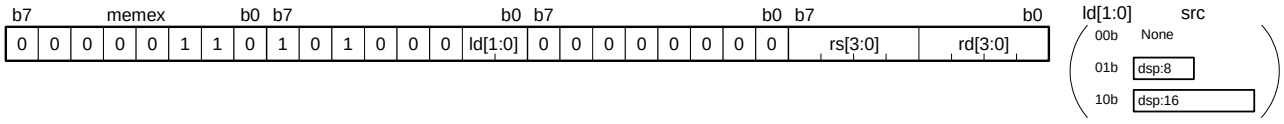
(1) SBB src, dest



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



ld[1:0]	src
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

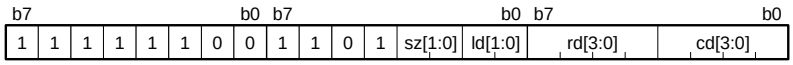
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

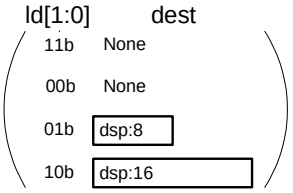
SC*Cnd* SC*Cnd*

Code Size

Syntax	Size	dest	Code Size (Byte)
(1) SC*Cnd*.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) SC*Cnd*.size dest





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

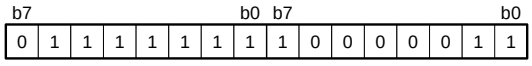
cd[3:0]	SC*Cnd*	cd[3:0]	SC*Cnd*
0000b	SCEQ, SCZ	1000b	SCGE
0001b	SCNE, SCNZ	1001b	SCLT
0010b	SCGEU, SCC	1010b	SCGT
0011b	SCLTU, SCNC	1011b	SCLE
0100b	SCGTU	1100b	SCO
0101b	SCLEU	1101b	SCNO
0110b	SCPZ	1110b	Reserved
0111b	SCN	1111b	Reserved

SCMPU SCMPU

Code Size

Syntax	Code Size (Byte)
(1) SCMPU	2

(1) SCMPU

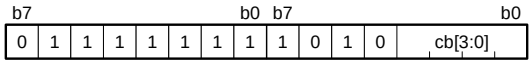


SETPSW SETPSW

Code Size

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

(1) SETPSW dest



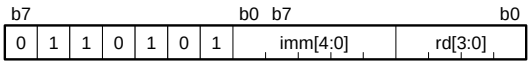
cb[3:0]	dest	
0000b	flag	C
0001b		Z
0010b		S
0011b		O
0100b		Reserved
0101b		Reserved
0110b		Reserved
0111b		Reserved
1000b		I
1001b		U
1010b		Reserved
1011b		Reserved
1100b		Reserved
1101b		Reserved
1110b		Reserved
1111b		Reserved

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

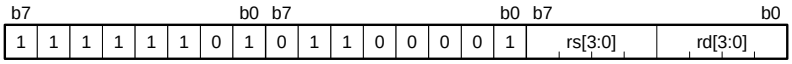
(1) SHAR src, dest



imm[4:0]	src	
00000b to 11111b	#IMM:5	0 to 31

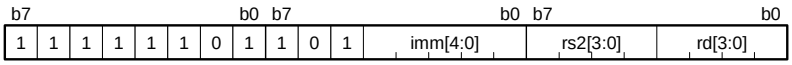
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) SHAR src, dest



rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

(3) SHAR src, src2, dest



imm[4:0]	src	
00000b to 11111b	#IMM:5	0 to 31

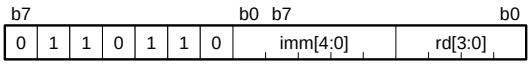
rs2[3:0]/rd[3:0]	src2/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

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

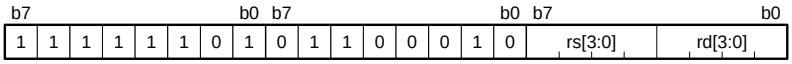
(1) SHLL src, dest



imm[4:0]	src	
00000b to 11111b	#IMM:5	0 to 31

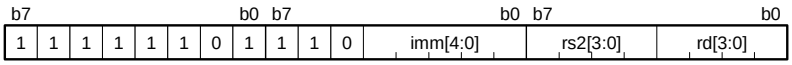
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) SHLL src, dest



rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

(3) SHLL src, src2, dest



imm[4:0]	src	
00000b to 11111b	#IMM:5	0 to 31

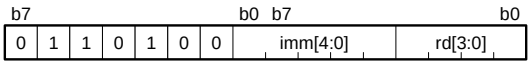
rs2[3:0]/rd[3:0]	src2/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

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

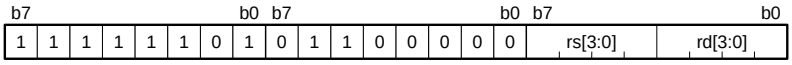
(1) SHLR src, dest



imm[4:0]	src	
00000b to 11111b	#IMM:5	0 to 31

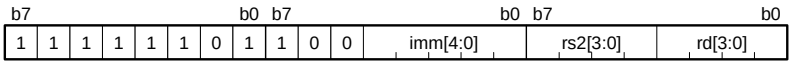
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) SHLR src, dest



rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

(3) SHLR src, src2, dest



imm[4:0]	src	
00000b to 11111b	#IMM:5	0 to 31

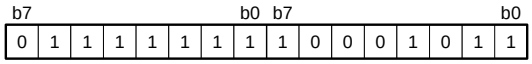
rs2[3:0]/rd[3:0]	src2/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

SMOVB SMOVB

Code Size

Syntax	Code Size (Byte)
(1) SMOVB	2

(1) SMOVB

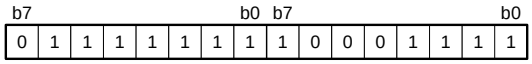


SMOVF SMOVF

Code Size

Syntax	Code Size (Byte)
(1) SMOVF	2

(1) SMOVF

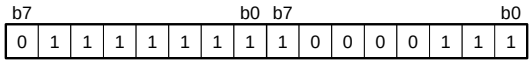


SMOVU SMOVU

Code Size

Syntax	Code Size (Byte)
(1) SMOVU	2

(1) SMOVU

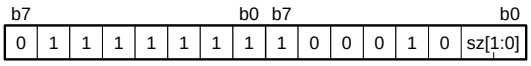


SSTR SSTR

Code Size

Syntax	Size	Processing Size	Code Size (Byte)
(1) SSTR.size	B	B	2
	W	W	2
	L	L	2

(1) SSTR.size



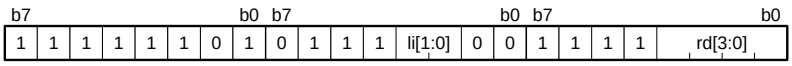
sz[1:0]	Size
00b	B
01b	W
10b	L

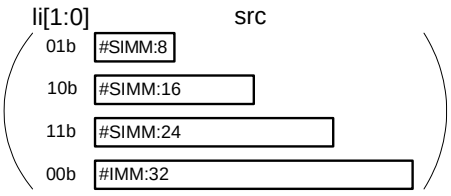
STNZ STNZ

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

(1) STNZ 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

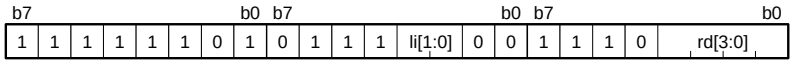
STZ

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

(1) STZ src, dest



li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

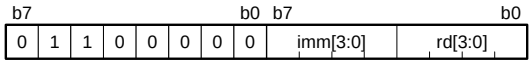
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

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

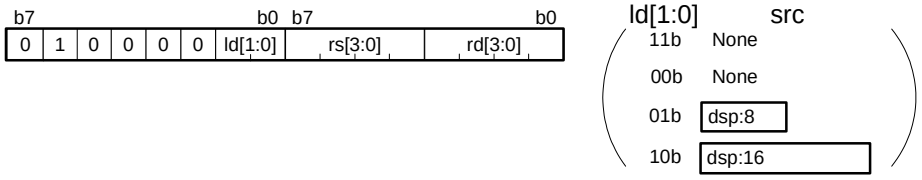


imm[3:0]	src	
0000b to 1111b	#UIMM:4	0 to 15

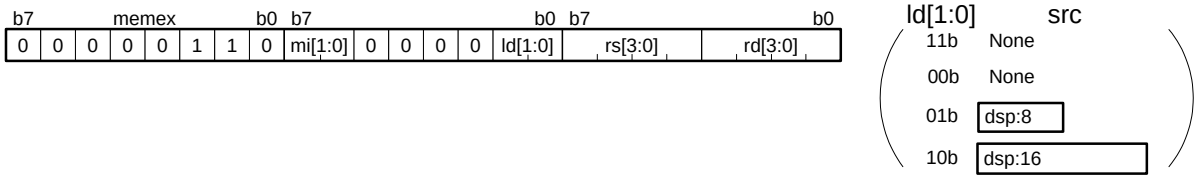
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) SUB src, dest

When memex == “UB” or src == Rs



When memex != “UB”

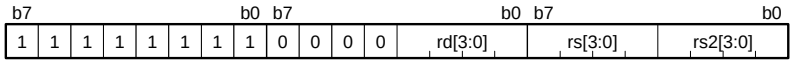


mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

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

(3) SUB src, src2, dest



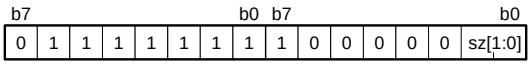
rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b to 1111b	Rs/Rs2/Rd	R0 (SP) to R15

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



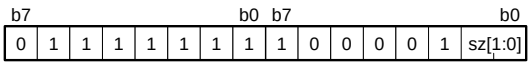
sz[1:0]	Size
00b	B
01b	W
10b	L

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



sz[1:0]	Size
00b	B
01b	W
10b	L

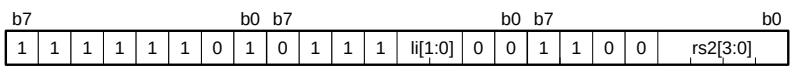
TST

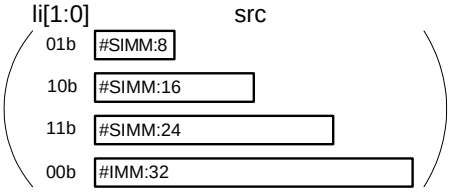
TST

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



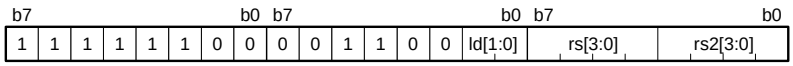


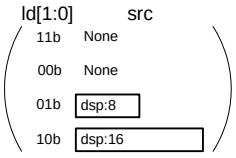
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

rs2[3:0]	src2	
0000b to 1111b	Rs	R0 (SP) to R15

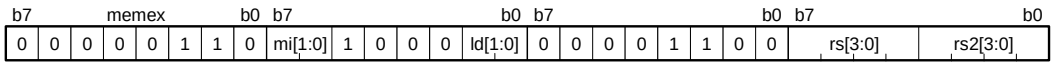
(2) TST src, src2

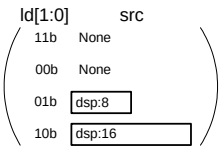
When memex == “UB” or src == Rs





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

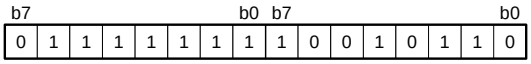
WAIT

WAIT

Code Size

Syntax	Code Size (Byte)
(1) WAIT	2

(1) WAIT



XCHG

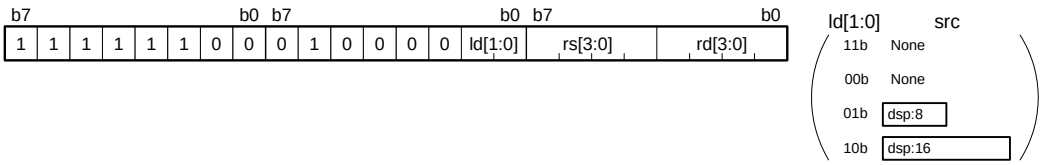
XCHG

Code Size

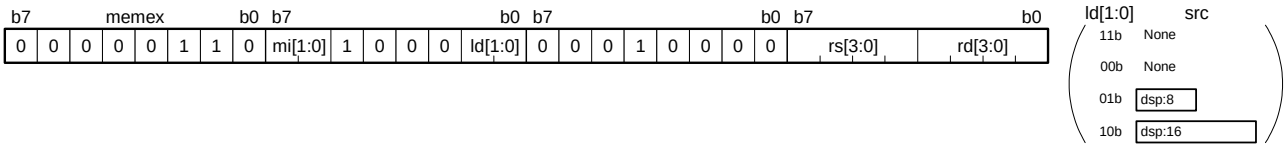
Syntax	src	dest	Code Size (Byte)
(1) XCHG src, dest	Rs	Rd	3
	[Rs].memex	Rd	3 (memex == “UB”) 4 (memex != “UB”)
	dsp:8[Rs].memex	Rd	4 (memex == “UB”) 5 (memex != “UB”)
	dsp:16[Rs].memex	Rd	5 (memex == “UB”) 6 (memex != “UB”)

(1) XCHG src, dest

When memex == “UB” or src == Rs



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]

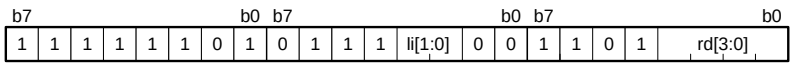
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

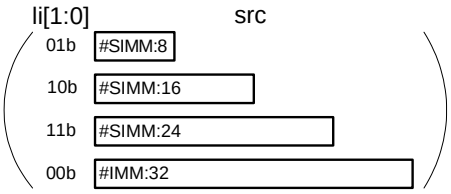
XOR XOR

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



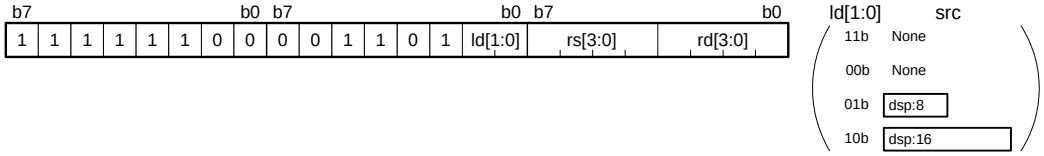


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

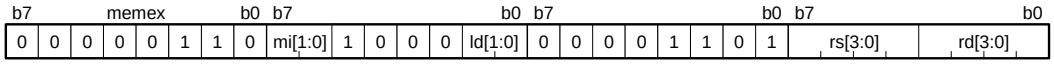
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

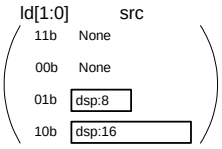
(2) XOR src, dest

When memex == “UB” or src == Rs



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

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.

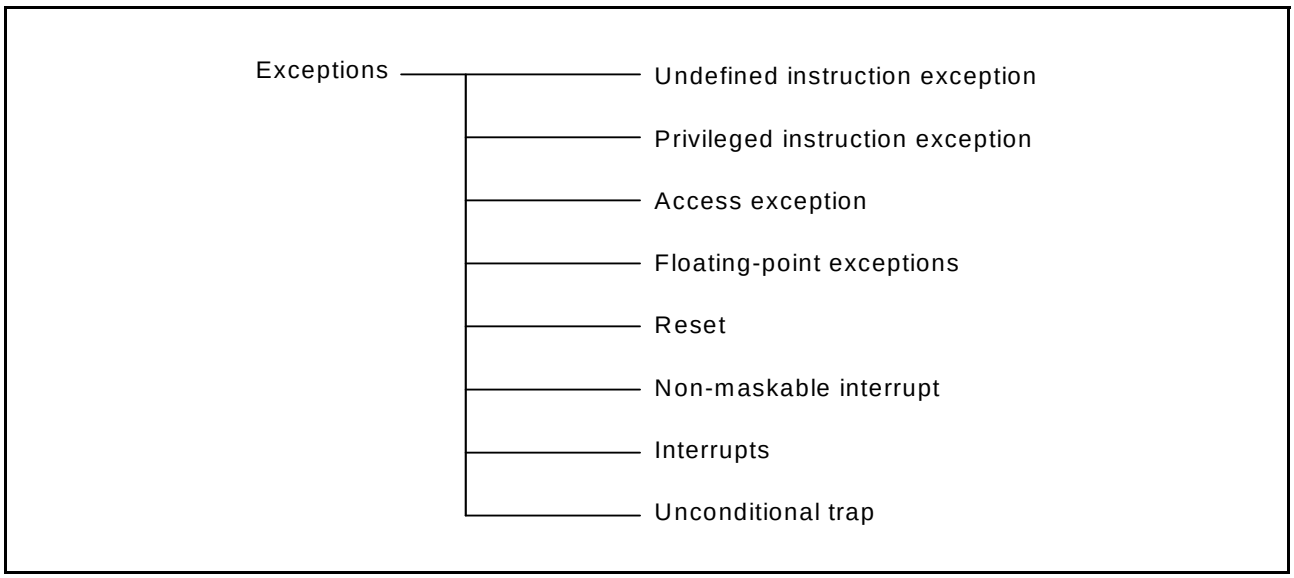


Figure 5.1 Types of Exception

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 are generated when any of the five exceptions specified in the IEEE754 standard, namely overflow, underflow, inexact, division-by-zero, or invalid operation, or an attempts to use processing that is not implemented, is detected upon execution of a floating-point arithmetic instruction. Exception handling by the CPU only proceeds when any among the EX, EU, EZ, EO, or EV bits in the FPSW, which corresponding to the five types of exception, is set to 1.

Note: Floating-point exceptions do not occur on the products which do not support the floating-point arithmetic instructions.

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 15* (the highest). The exception processing of interrupts is masked when the I bit in PSW is 0.

Note: * The priority level of the fast interrupt is 7 in products of the RX610 group.

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.

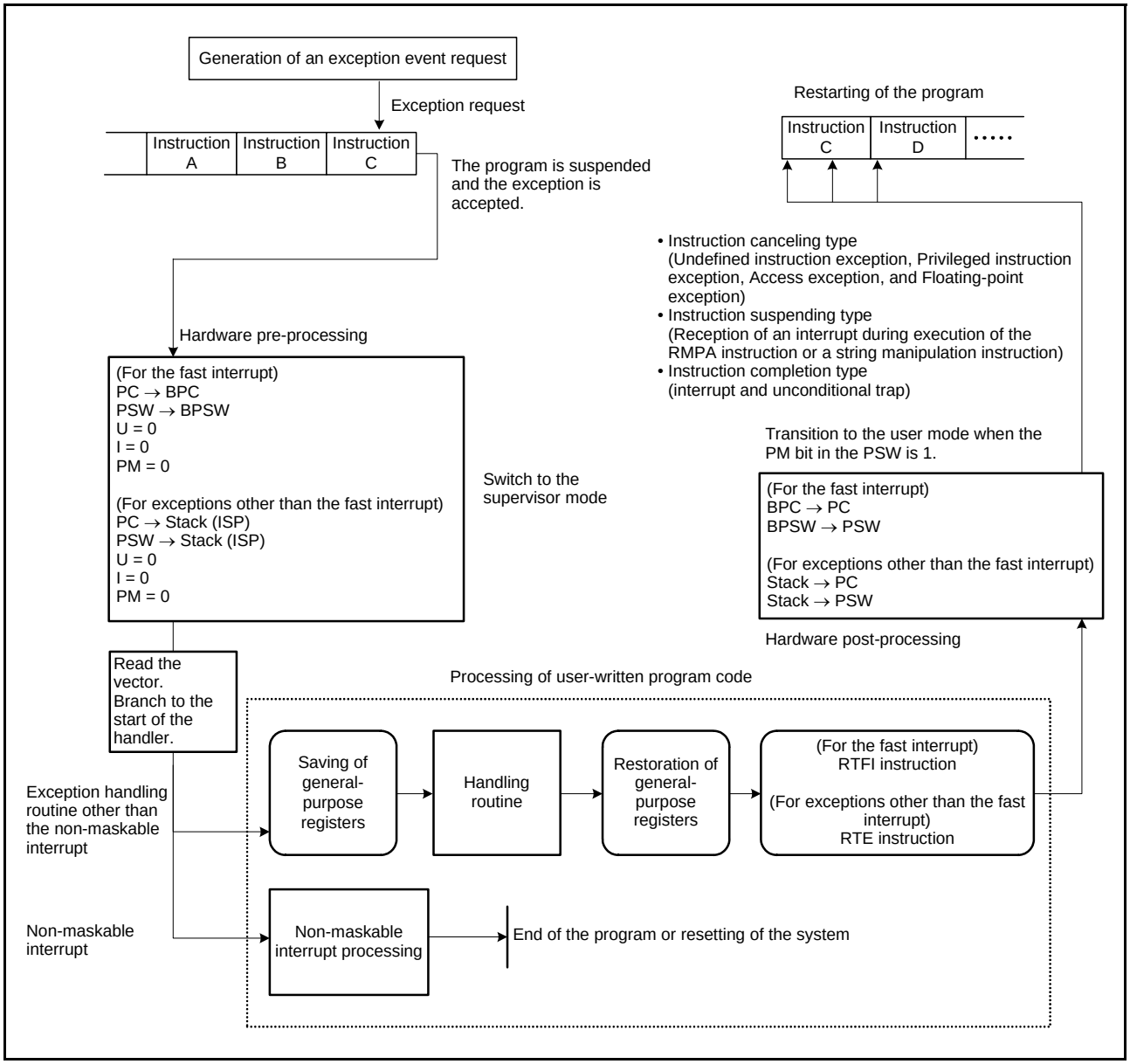


Figure 5.2 Outline of the Exception Handling Procedure

When an exception is accepted, hardware processing by the 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 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 saved on the stack. General purpose registers and control registers other than the PC and PSW that are to be used within the exception handling routine must be saved by user program code at the start of the exception handling routine.

At the end of exception handling routine, after the restoration of registers saved by the user, the RTE instruction is

executed to return 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, end the program or reset the system without returning to the original program.

Hardware post-processing by the 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 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 generated the exception
Privileged instruction exception		Instruction canceling type	During instruction execution	PC value of the instruction that generated the exception
Access exception		Instruction canceling type	During instruction execution	PC value of the instruction that generated the exception
Floating-point exceptions		Instruction canceling type	During instruction execution	PC value of the instruction that generated 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		Fixed vector table	Stack
Privileged instruction exception		Fixed vector table	Stack
Access exception		Fixed vector table	Stack
Floating-point exceptions		Fixed vector table	Stack
Reset		Fixed vector table	Nowhere
Non-maskable interrupt		Fixed vector table	Stack
Interrupts	Fast interrupt	FINTV	BPC and BPSW
	Other than the above	Relocatable vector table	Stack
Unconditional trap		Relocatable 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

Note: The FPSW is not saved by the hardware preprocessing. If floating-point arithmetic instructions are to be used within an exception handling routine, save the FPSW on the stack 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

(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 → PSW

(b) Restoring the PC

(For the fast interrupt)

BPC → PC

(For other exceptions)

Stack → PC

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 address FFFFFFDCh.
(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 address FFFFFFD0h.
(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 address FFFFFFD4h.
(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 address FFFFFFE4h.
(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 address FFFFFFF8h.
(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 on the stack (ISP). 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 relocatable 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 relocatable vector table.
For the BRK instruction, the value at the vector from the start address is fetched from the relocatable 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 the backup PC (BPC) or the backup PSW (BPSW) by the hardware preprocessing.

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		Return is 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

Numerics

A

B

C

D

E

F

G

I

N

O

P

Q

R

S

U

V

Z

REVISION HISTORY	RX Family RXv1 Instruction Set Architecture User's Manual: Software

		Description	
Rev.	**Date**	**Page**	**Summary**
0.10	Nov. 12, 2007	—	First edition issued
0.20	Mar. 18, 2008	3 to 5	Notation in This Manual changed
		8 to 13	List of Instructions for RX Family changed
		14	Section 1 CPU Functions changed
		14	1.1 Features changed
		15	1.2 Register Set of the CPU changed
		15	Figure 1.1 Register Set of the CPU changed
		16	1.2.2 Control Registers changed
		17	1.2.2.1 Interrupt Stack Pointer (ISP)/User Stack Pointer (USP) changed
		18	1.2.2.4 Processor Status Word (PSW): b31 to b4 changed, Notes 1 and 2 changed
		19	IPL[2:0] bits (Processor interrupt priority level) changed
		20	1.2.2.6 Backup PSW Register (BPSW) added
		20	1.2.2.7 Vector Register (VCT) → 1.2.2.7 Fast Interrupt Vector Register (FINTV) changed
		21	1.2.2.8 Floating-Point Status Word (FPSW): b25 to b15, b9, b7 to b0 changed
		22	1.2.2.9 Coprocessor Enable Register (CPEN) added
		24	Table 1.5 Conditions Leading to an Invalid Exception and the Operation Results changed
		25	1.4.1 Supervisor Mode changed
		25	1.4.2 User Mode added
		25	1.4.3 Privileged Instruction changed
		25	1.4.4 Switching Between Processor Modes changed
		29	1.7 Vector Table changed
		29	1.7.1 Fixed Vector Table changed
		29	Figure 1.8 Fixed Vector Table changed
		30	1.7.2 Relocatable Vector Table changed
		31	2.1 Types of Addressing Mode, (3) Special Instruction Addressing Modes added
		32	2.2 Guide to This Section, (2) Symbolic notation changed
		33	Immediate: #IMM:S8, #IMMEX:U8 added
		33	Register Indirect: Operation diagram added
		33	Register Relative: Description, Operation diagram changed
		34	Short Immediate: #IMM:2 added, Description for #IMM:3 changed
		34	Short Register Relative: Description changed, Operation diagram added
		35	Post-increment Register Indirect: Operation diagram added
		35	Pre-decrement Register Indirect: Description changed, Operation diagram added
		35	Indexed Register Indirect: Operation diagram added
		36	Control Register Direct: VCT → FINTV changed, CPEN added, Description changed, Operation diagram changed
		36	Program Counter Relative: Rn added
		36	Program Counter Relative: label (dsp:3) → pcdsp:3 changed, Description changed, Operation diagram changed
		37	Program Counter Relative: label (dsp:8) (dsp:16) (dsp:24) → pcdsp:8 pcdsp:16 pcdsp:24 changed, Description changed, Operation diagram changed

Rev.	Date	Description	
		Page	**Summary**
		37	Register Direct: added
		38	Section 3 Instruction Descriptions added
		159	Section 5 EXCEPTIONS added
0.30	Jul. 31, 2008	3 to 5	Notation in This Manual Symbols: IMM, IMMEX → IMM, SIMM, UIMM changed Bit length specifiers: :1 added Bit length extension specifier: :S8, :U8 deleted Operations: tmp2, tmp3 added
		8 to 13	List of Instructions for RX Family FREIT instruction → RTFI instruction, REIT instruction → RTE instruction changed EDIV instruction, EDIVU instruction, MULU instruction, PUSHA instruction, and STOP instruction deleted For floating-point operation instructions and coprocessor instructions, the description as an optional function added DSP instructions added
		14	Section 1 CPU Functions changed
		14	1.1 Features changed
		15	1.2 Register Set of the CPU changed
		15	Figure 1.1 Register Set of the CPU changed
		17	1.2.2.2 Interrupt Table Register (INTB) Interrupt vector table → Relocatable vector table changed
		18	1.2.2.4 Processor Status Word (PSW), Note 3 changed
		19	U bit (Stack pointer select bit) changed
		22	1.2.2.8 Floating-Point Status Word (FPSW), Note 3 added
		23	1.2.3 Accumulator (ACC) added
		24	1.3.2 Underflow added
		24	Table 1.3 Conditions Leading to an Inexact Exception and the Operation Results, Notes added
		25	1.3.4 Division-by-Zero, Note for denormalized number, QNaN, and SNaN added
		25	Table 1.5 Conditions Leading to an Invalid Exception and the Operation Results changed
		26	Table 1.6 Rules for Generating QNaNs added
		26	1.3.6 Unimplemented Processing changed, Note deleted
		27	1.4.3 Privileged Instruction changed
		27	1.4.4 Switching Between Processor Modes, (2) Switching from supervisor mode to user mode changed
		33 to 39	Section 2 Addressing Modes changed
		42	(5) Operation, (c) Special notation added
		43	(8) Instruction Format, (d) Immediate value changed
		47 to 171	Code Size in Instruction Format added
		48	ADC instruction: Instruction Format changed
		50	ADD instruction: Instruction Format changed
		51	AND instruction: Instruction Format changed
		54	B*Cnd* instruction: Instruction Format changed
		58	BRA instruction: Instruction Format changed
		64	CMP instruction: Instruction Format, Description Example changed

Rev.	Date	Description	
		Page	Summary
0.30	Jul. 31, 2008	65	DIV instruction: Instruction Format changed
		67	DIVU instruction: Instruction Format changed
		69 to 70	EMUL instruction: Note in Function added, Instruction Format changed
		71 to 72	EMULU instruction: Note in Function added, Instruction Format changed
		73	FADD instruction: Flag Change, Note in Instruction Format changed
		75 to 77	FCMP instruction: Syntax, Operation, Function, Flag Change, Instruction Format, Supplementary Description changed
		78	FDIV instruction: Flag Change, Note in Instruction Format changed
		80 to 82	FMUL instruction: Note in Function added, Flag Change, Note in Instruction Format, Supplementary Description changed
		83 to 84	FSUB instruction: Flag Change, Note in Instruction Format changed
		86 to 88	FTOI instruction: Function, Flag Change, Instruction Format, Supplementary Description changed
		89	INT instruction: Instruction Format, Syntax: INT → INT src changed
		90 to 91	ITOF instruction: Function, Flag Change, Instruction Format changed
		94	MACHI instruction added
		95	MACLO instruction added
		96	MAX instruction: Instruction Format changed
		97	MIN instruction: Instruction Format changed
		98 to 100	MOV instruction: Function, Instruction Format, Description Example changed
		101	MOVU instruction: Note in Instruction Format changed
		103 to 104	MUL instruction: Syntax, Operation, Function, Flag Change, Instruction Format, Description Example changed
		105	MULHI instruction added
		106	MULLO instruction added
		107	MVFACHI instruction added
		108	MVFACMI instruction added
		111	MVTACHI instruction added
		112	MVTACLO instruction added
		113	MVTC instruction: Instruction Format changed
		114	MVTCP instruction: Instruction Format changed
		117	NOP instruction: Operation, Function changed
		120	OR instruction: Instruction Format changed
		125	PUSH instruction: Function added, Note in Instruction Format changed
		128 to 129	RACW instruction added
		132	RMPA instruction: Function added, Note added
		138 to 140	ROUND instruction: Function, Flag Change, Instruction Format changed, Supplementary Description added
		141	RTE instruction: REIT instruction → RTE instruction changed
		142	RTFI instruction: FREIT instruction → RTFI instruction changed
		144 to 145	RTSD instruction: Operation, Function, Instruction Format changed
		148	SBB instruction: Note in Instruction Format changed
		149	SC*Cnd* instruction: Note in Instruction Format changed
		151	SCMPU instruction: Operation, Function, Flag Change changed
		156	SMOVB instruction: Operation, Function changed
		157	SMOVF instruction: Operation, Function changed
		158	SMOVU instruction: Operation, Function changed

Rev.	Date	Description	
		Page	Summary
0.30	Jul. 31, 2008	159	SSTR instruction: Operation, Function changed
		160	STNZ instruction: Instruction Format changed
		161	STZ instruction: Instruction Format changed
		162	SUB instruction: Instruction Format changed
		163 to 164	SUNTIL instruction: Operation, Function, Flag Change, Instruction Format changed
		165 to 166	SWHILE instruction: Note 3 in Operation deleted, Operation, Function, Flag Change, Instruction Format changed
		167	TST instruction: Instruction Format changed
		169 to 170	XCHG instruction: Syntax, Function, Instruction Format, Description Example changed
		171	XOR instruction: Instruction Format changed
		172 to 260	Section 4 Instruction Code added
		262	5.2.1 Undefined Instruction Exception added
		262	5.2.5 Reset changed
		262	5.2.6 Non-Maskable Interrupt changed
		264	Figure 5.2 Outline of the Exception Handling Procedure changed
		265	5.3 Exception Handling Procedure: FREIT instruction → RTFI instruction, REIT instruction → RTE instruction changed
		268	5.5 Hardware Processing for Accepting and Returning from Exceptions (2) FREIT instruction → RTFI instruction, REIT instruction → RTE instruction changed (a) Changed
		269 to 270	5.6 Exception Sequences: Processor mode select bit, RM → PM error amended
		271	Table 5.3 Return from Exception Processing Routines: FREIT instruction → RTFI instruction, REIT instruction → RTE instruction changed
		271	Table 5.4 Order of Priority for Exceptions changed
0.50	Feb. 3, 2009	3	Notation in This Manual Rx added, Fx → flag changed
		9, 13	List of Instructions for RX Family Coprocessor instructions (MVFCP, MVTCP, and OPECP instructions) deleted
		14	Section 1 CPU Functions, 1.1 Features, changed
		15	Figure 1.1 Register Set of the CPU, CPEN register deleted
		16	1.2.2 Control Registers, CPEN register deleted
		17	1.2.2.2 Interrupt Table Register (INTB) changed
		18	1.2.2.4 Processor Status Word (PSW): I bit changed, PM bit added
		20	1.2.2.7 Fast Interrupt Vector Register (FINTV) changed
		22	1.2.2.8 Floating-Point Status Word (FPSW): Notes changed and added
		22	[Explanation of Floating-Point Rounding Modes] added
		26	1.4.4 Switching Between Processor Modes, (2) Switching from supervisor mode to user mode, changed
		30	Figure 1.8 Fixed Vector Table changed
		31	1.7.2 Relocatable Vector Table, Description changed
		32	1.8 Address Space added
			Section 2 Addressing Modes
		35 to 36	Immediate: #IMM:2 deleted, Operation diagram for #UIMM:8 added
		37	Control Register Direct: PC added, CPEN deleted
		39	2.2.1 Ranges for Immediate Values added

		Description	
Rev.	**Date**	**Page**	**Summary**
0.50	Feb. 3, 2009		Section 3 Instruction Descriptions, 3.1 Guide to This Section:
		41	(4) Syntax, (c) Operand, changed
		42	(5) Operation, (b) Pseudo-functions, changed
		43	(8) Instruction Format, (b) Control registers, changed, (c) Flag and bit, changed
		—	Coprocessor instructions (MVFCP, MVTCP, and OPECP instructions) deleted
			Bit pattern of the instruction → Instruction code changed
		53	BCLR instruction: Function added
		54	B*Cnd* instruction, Description Example: Note added
		55	BM*Cnd* instruction: Function added
		57	BNOT instruction: Function added
		58	BRA instruction, Description Example: Note added
		59	BRK instruction: Function changed
		60	BSET instruction: Function added
		61	BSR instruction: Note in Operation added
		61	BSR instruction, Description Example: Note added
		62	BTST instruction: Function added
		70	EMUL instruction: Instruction Format added
		72	EMULU instruction: Instruction Format added
		73	FADD instruction: Note in Flag Change changed
		75	FCMP instruction: Function changed, Note in Flag Change changed
		78	FDIV instruction: Note in Flag Change changed
		80	FMUL instruction: Note in Flag Change changed
		83	FSUB instruction: Note in Flag Change changed
		86	FTOI instruction: Note in Flag Change changed
		89	INT instruction: Function changed
		90	ITOF instruction: Note in Flag Change changed
		99 to 100	MOV instruction: Instruction Format changed, Note 1 changed
		101	MOVU instruction: Note 1 in Instruction Format changed
		109	MVFC instruction: Function added, Note in Instruction Format changed
		112	MVTC instruction: Note in Instruction Format changed
		113	MVTIPL instruction: Function added
		120	POPC instruction: Instruction Format changed
		123	PUSHC instruction: Function added, Instruction Format changed
		129	RMPA instruction: Note in Operation changed
		135	ROUND instruction: Note in Flag Change changed
		142	RTSD instruction, Instruction Format: Description added, Note changed
		148	SCMPU instruction: Note in Operation changed
		153	SMOVB instruction: Note in Operation changed
		154	SMOVF instruction: Note in Operation changed
		155	SMOVU instruction: Note in Operation changed
		156	SSTR instruction: Note in Operation changed
		160	SUNTIL instruction: Note in Operation changed
		162	SWHILE instruction: Note in Operation changed
		165	WAIT instruction, Function: Description added, Note added

Rev.	Date	Description	
		Page	**Summary**
0.50	Feb. 3, 2009		Section 4 Instruction Code
		170	4.1 Guide to This Section, (2) List of Code Size: Description added
		—	Coprocessor instructions (MVFCP, MVTCP, and OPECP instructions) deleted
		180 to 181	B*Cnd*: Instruction codes (1) and (3) changed
		213 to 214	MOV: Code Size (list) changed
		217	MOV: Instruction code (14) changed, Instruction code (15) added
		222	MVFACMI: Instruction code (1) changed
		223	MVFC: Instruction code (1) changed
		225 to 226	MVTC: Instruction codes (1) and (2) changed
		231	POPC: Instruction code (1) changed
		233	PUSHC: Instruction code (1) changed
			Section 5 Exceptions
		257	5.1 Types of Exception: Section title changed
		257	Figure 5.1 Types of Exception changed
		258	5.1.4 Floating-Point Exceptions changed
		258	5.1.7 Interrupts changed
		258	5.1.8 Unconditional Trap added (5.2.8 INT Instruction Exceptions and 5.2.9 BRK Instruction Exception deleted)
		259	Figure 5.2 Outline of the Exception Handling Procedure changed
		260	5.2 Exception Handling Procedure changed
		261	Table 5.1 Timing of Acceptance and Saved PC Value changed
		262	Table 5.2 Vector Table and Site for Preserving the PC and PSW Registers changed
		263	5.4 Hardware Processing for Accepting and Returning from Exceptions, (1) Hardware pre-processing for accepting an exception, (a) Preserving the PSW register: Note added
		265	5.5.8 Unconditional Trap added (5.6.8 INT Instruction Exceptions and 5.6.9 BRK Instruction Exception deleted)
		266	Table 5.3 Return from Exception Processing Routines changed
		266	Table 5.4 Order of Priority for Exceptions changed
		267	Index added
0.51	Mar. 24, 2009	—	DSP instructions, floating-point operation instructions, floating-point operation unit are described without the phase “(as an optional function)”.
		30	1.7.1 Fixed Vector Table, Figure 1.8 Fixed Vector Table Reserved area is added to addresses in the range from FFFFFF80h to FFFFFFCCh.
0.60	May. 26, 2009	9	List of Instructions Classified in Alphabetical Order MVTIPL (privileged instruction) deleted
		13	List of Instructions Classified by Type MVTIPL (privileged instruction) deleted
		18	1.2.2.4 Processor Status Word (PSW) Description on the MVTIPL deleted from Note 1
		26	1.4.3 Privileged Instruction Description on the MVTIPL deleted
		35	2.2 Addressing Modes Immediate, #IMM:3: Description on the MVTIPL deleted
		—	3.2 Instructions in Detail Description on the MVTIPL deleted
		—	4.2 Instruction Code Described in Detail Description on the MVTIPL including the code size deleted

Rev.	Date	Description	
		Page	Summary
1.00	June 11, 2010	5	Notation in This Manual, Operations: << and >> added, tmp32 and tmp64 deleted
		8 to 16	List of Instructions for RX Family
			B*Cnd*, BM*Cnd*, and SC*Cnd* instructions: *Cnd* described as mnemonic
			MVTIPL instruction (privileged instruction) added, table note added
		All	Exception sequence → Hardware pre-processing, Exception handler → Exception handling routine, changed
			Section 1 CPU Functions
			1.1 Features
		17	Register set of the CPU, and the accumulator, changed
			1.2 Register Set of the CPU
		18	Figure 1.1 Register Set of the CPU, changed
			1.2.2.3 Program Counter (PC)
		20	Bit arrangement diagram, Value after reset, changed
			1.2.2.4 Processor Status Word (PSW)
		21	Bit arrangement diagram: Note for b27, added
		21	Bits IPL[2:0] → Bits IPL[3:0] changed
		22	Note 1 changed, Note 4 added
		22	Description on bits IPL[3:0] changed
			1.2.2.8 Floating-Point Status Word (FPSW)
		25	FS: Floating-point flag summary bit → Floating-point error summary flag, changed
		25 to 26	Description on bits added
		26	1.2.3 Accumulator (ACC), changed
		29	1.3.6 Unimplemented Processing, changed
			1.4.2 User Mode
		30	Bits IPL[2:0] → Bits IPL[3:0] changed
			1.4.3 Privileged Instruction
		30	MVTIPL instruction added
			Section 2 Addressing Modes
			2.2 Addressing Modes
		39	Immediate, #IMM:3: changed, Immediate, #IMM:4: added
		41	PSW Direct, Operation diagram: Bits IPL[2:0] → Bits IPL[3:0] changed
		43	Table 2.1 Ranges for Immediate Values: IMM:4 added
			Section 3 Instruction Descriptions
		46	3.1 Guide to This Section, (a) Data type: signed long long, unsigned long long, and float, added
		57	BCLR instruction: Operation (1) and (2), changed
		58	B*Cnd* instruction, Function: The column for *Cnd* described as mnemonic
		59	BM*Cnd* instruction: Operation (1) and (2), changed
			Function: The column for *Cnd* described as mnemonic
		61	BNOT instruction: Operation (1) and (2), changed
		80	FCMP instruction:
			Supplementary Description, =: src2 = src → src2 == src changed
		98	MACHI instruction: Operation and Function, changed
		99	MACLO instruction: Operation and Function, changed
		109	MULHI instruction: Operation changed
		110	MULLO instruction: Operation changed
		114	MVTACHI instruction: Operation changed
		115	MVTACLO instruction: Operation changed
		116	MVTC instruction: Function changed
		117	MVTIPL instruction, added

Rev.	Date	Description	
		Page	**Summary**
1.00	June 11, 2010	124	POPC instruction: Function changed
		129	RACW instruction: Operation changed
		135	ROLC instruction: Operation added, Function changed
		136	RORC instruction: Operation added, Function changed
		137	ROTL instruction: Operation added, Function changed
		138	ROTR instruction: Operation added, Function changed
		145	RTSD instruction: Operation (2), changed
		147	SAT instruction: Operation changed
		148	SATR instruction: Operation changed
		150	SC*Cnd* instruction, Function: The column for *Cnd* described as mnemonic
		154	SHAR instruction: Operation added, Function changed
		155	SHLL instruction: Operation added, Function changed
		156	SHLR instruction: Operation added, Function changed
		164	SUNTIL instruction: Operation changed
		166	SWHILE instruction: Operation changed
			Section 4 Instruction Code
			4.1 Guide to This Section
		174	(4) Instruction Code: Instruction code for memex (when memex == UB or src == Rs, when memex != UB) and src/dest description changed
			4.2 Instruction Code Described in Detail
		177 to 255	Description of memex specifier: SB → B, SW → W, changed
		185 to 186	B*Cnd* instruction: The column for *Cnd* described as mnemonic
		187	BM*Cnd* instruction: The column for *Cnd* described as mnemonic
		227	MVTIPL instruction, added
		243	SC*Cnd* instruction: The column for *Cnd* described as mnemonic
			Section 5 Exceptions
		257	5.1.3 Access Exception, changed
		257	5.1.7 Interrupts, changed
		258	5.2 Exception Handling Procedure, changed
		261	5.3.2 Vector and Site for Preserving the PC and PSW, changed
		261	Table 5.2 Vector and Site for Preserving the PC and PSW, changed
			5.4 Hardware Processing for Accepting and Returning from Exceptions: Description added
		262	(b) Updating of the PM, U, and I bits in the PSW, changed
		264	5.5.6 Non-Maskable Interrupt, (4) changed
		264	5.5.7 Interrupts, (4) changed
1.10	Aug. 11, 2011	All	RX200 specifications in the RX200 Series are reflected
			Section 2 Addressing Modes
		39	2.2 Addressing Modes
			Immediate, Symbol: #IMM:4, added
			Section 3 Instruction Descriptions
		58	B*Cnd* instruction, Function: The expression described in the condition column, changed (parentheses added)
		59	BM*Cnd* instruction, Function: The expression described in the condition column, changed (parentheses added)
		66	BTST instruction, Instruction Format: The column for src2, changed
		80	FCMP instruction, Instruction Format: The column for src2, changed
		150	SC*Cnd* instruction, Function: The expression described in the condition column, changed (parentheses added)

		Description	
Rev.	**Date**	**Page**	**Summary**
1.10	Aug. 11, 2011		Section 4 Instruction Code
		194	BTST instruction, Code Size: Description of (1) and (3) in the column for src2, changed
		205	FCMP instruction, Code Size: Description of (1) in the column for src2, changed
1.20	Apr. 15, 2013	All	RX100 specifications in the RX100 Series are reflected
1.30	Dec 26, 2019	Front cover	Document title changed to “RX Family RXv1 Instruction Set Architecture User’s Manual: Software”
		All	Support details by product series deleted Expression “for details, refer to the user’s manual: hardware for each product” used as required
		3	How to Use This Manual: The entire section added
			Notation in This Manual
		4	Symbols: “src2” added to Notation; meanings for “Rs2” and “Rd2” changed; “ACC” added; “tmp, tmp0, tmp1, tmp2, tmp3” moved from Operations
		5	Operations: “!” deleted
		9	List of RXv1 Instruction Set for RX Family: The title changed; description deleted
		12	Quick Page Reference in Alphabetical Order: Description for Note 1 modified
			List of Instructions: The title changed; moved to section 3
		13	1. CPU Programming Model: Section title changed; descriptions modified
			1.1 Features: Descriptions and note modified
		14	Figure 1.1 Register Set of the CPU: Layout changed
		16	1.2.2.1 Interrupt Stack Pointer (ISP)/User Stack Pointer (USP): The second paragraph deleted
		17, 18	1.2.2.4 Processor Status Word (PSW): Note deleted; descriptions for flags modified
		21	1.2.2.8 Floating-Point Status Word (FPSW): Descriptions modified
		23	1.2.3 Accumulator: Description in the figure modified
		24	1.3 Floating-Point Exceptions: Descriptions modified
		25	1.3.4 Division-by-Zero: Description in note included to the text
		26	1.3.5 Invalid Operation: Term “mantissa” modified to “fraction”
		28	1.5.2 Floating-Point Number: Term “Mantissa” modified to “Fraction”
			2. Addressing Modes
		37	2.2 Addressing Modes, Register Relative: Mnemonic “MOVE” corrected to “MOVU”
			3. Instruction Descriptions
		41	3.1 Overview of Instruction Set: newly added
			List of Instructions: The title changed; moved from Notation in This Manual
		All	3.3 Instructions in Detail: Caption “Possible Exceptions” changed to “Sources of Floating-Point Exceptions”
		76	FADD instruction, Supplementary Description: Note for the table of operation result when DN = 0 added
		81	FMUL instruction, Function: Note modified
		84	FSUB instruction, Supplementary Description: Note for the table of operation result when DN = 0 added
		110	MVTACLO instruction: Operation modified
		124	RACW instruction, Instruction Format: Note deleted
		143	SBB instruction, Operation: Operator “!” corrected to “~”
		146	SETPSW instruction, Function: Description corrected
		148	SHLL instruction, Flag Change: Term “scr” corrected to “src”

		Description	
Rev.	**Date**	**Page**	**Summary**
1.30	Dec 26, 2019		4. Instruction Code
		171	ADD instruction, Code Size: Item number (2) added
		225	RACW instruction, Code Size: Range of immediate value for src added
			5. Exceptions
		247	Figure 5.1 Types of Exception: Note deleted
		256	Table 5.3 Return from Exception Handling Routines, Non-maskable interrupt: Description “Return is impossible” corrected to “Return is disabled”

RX Family RXv1 Instruction Set Architecture
User's Manual: Software

Publication Date: Rev.0.10 Nov 12, 2007
Rev.1.30 Dec 26, 2019

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 Corporation
TOYOSU FORESIA, 3-2-24 Toyosu, Koto-ku, Tokyo 135-0061, Japan

Renesas Electronics America Inc.
1001 Murphy Ranch Road, Milpitas, CA 95035, U.S.A.
Tel: +1-408-432-8888, Fax: +1-408-434-5351

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 101-T01, Floor 1, Building 7, Yard No. 7, 8th Street, Shangdi, Haidian District, Beijing 100085, China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai 200333, China
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, 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 No 3A-1 Level 3A Tower 8 UOA Business Park, No 1 Jalan Pengaturcara U1/51A, Seksyen U1, 40150 Shah Alam, Selangor, Malaysia
Tel: +60-3-5022-1288, Fax: +60-3-5022-1290

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL 2nd Stage, Indiranagar, Bangalore 560 038, India
Tel: +91-80-67208700

Renesas Electronics Korea Co., Ltd.
17F, KAMCO Yangjae Tower, 262, Gangnam-daero, Gangnam-gu, Seoul, 06265 Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5338

Colophon 6.0

RX Family RXv1 Instruction Set Architecture

Renesas Electronics Corporation

R01US0032EJ0130