User’s Manual

32

RX Family

User’s Manual: Software

RENESAS 32-Bit MCU
RX Family

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

Renesas Electronics
www.renesas.com

Rev.1.20 Apr 2013

General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU products from Renesas. For detailed usage notes on the products covered by this manual, refer to the relevant sections of the manual. If the descriptions under General Precautions in the Handling of MPU/MCU Products and in the body of the manual differ from each other, the description in the body of the manual takes precedence.

1. Handling of Unused Pins

   Handle unused pins in accord with the directions given under Handling of Unused Pins in the manual.

   — The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment when power is supplied.

   — The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.

   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.

   In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.

   — When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

5. Differences between Products

   Before changing from one product to another, i.e. to one with a different part number, confirm that the change will not lead to problems.

   — The characteristics of MPU/MCU in the same group but having different part numbers may differ because of the differences in internal memory capacity and layout pattern. When changing to products of different part numbers, implement a system-evaluation test for each of the products.

Notation in This Manual

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

Classification	Notation	Meaning
Symbols	IMM	Immediate value
	SIMM	Immediate value for sign extension according to the processing size
	UIMM	Immediate value for zero extension according to the processing size
	src	Source of an instruction operand
	dest	Destination of an instruction operand
	dsp	Displacement of relative addressing
	pcdsp	Displacement of relative addressing of the program counter
	[ ]	Represents indirect addressing
	Rn	General-purpose register. R0 to R15 are specifiable unless stated otherwise.
	Rs	General-purpose register as a source. R0 to R15 are specifiable unless stated otherwise.
	Rs2	Used in the description for the ADD, AND, CMP, MUL, OR, PUSHM, SUB, and TST instructions. In these instructions, since two general-purpose registers can be specified for an 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	Used in the description for the POPM and RTSD instructions. In these instructions, since two general-purpose registers can be specified for an 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.
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 accord with C language syntax. The following is the notation in this manual.)	
	=	Assignment operator. The value on the right is assigned to the variable on the left.
	–	Indicates negation as a unary operator or a "difference" as a binary operator.
	+	Indicates "sum" as a binary operator.
	*	Indicates a pointer or a "product" as a binary operator.
	/	Indicates "quotient" as a binary operator.
	%	Indicates "remainder" as a binary operator.
	~	Indicates bit-wise "NOT" as a unary operator.
	&	Indicates bit-wise "AND" as a binary operator.
	|	Indicates bit-wise "OR" as a binary operator.
	^	Indicates bit-wise "Exclusive OR" as a binary operator.
	;	Indicates the end of a statement.
	{ }	Indicates the start and end of a complex sentence. Multiple statements can be put in { }.
	if (expression) statement 1 else statement 2	Indicates an if-statement. The expression is evaluated; statement 1 is executed if the result is true and statement 2 is executed if the result is false.
	for (statement 1; expression; statement 2) statement 3	Indicates a for-statement. After executing statement 1 and then evaluating the expression, statement 3 is executed if the result is true. After statement 3 is executed the first time, the expression is evaluated after executing statement 2.
	do statement while (expression);	Indicates a do-statement. As long as the expression is true, the statement is executed. Regardless of whether the expression is true or false, the statement is executed at least once.
	while (expression) statement	Indicates a while-statement. As long as the expression is true, the statement is executed.

Classification	Notation	Meaning
Operations	==, !=	Comparison operators. "==" means "is equal to" and "!=" means "is not equal to".
	>, <	Comparison operators. ">" means "greater than" and "<" means "less than".
	>=, <=	Comparison operators. The condition includes "==" as well as ">" or "<".
	&&	Logical operator. Indicates the "AND" of the conditions to the left and right of the operator.
	||	Logical operator. Indicates the "OR" of the conditions to the left and right of the operator.
	<<, >>	Shift operators, respectively indicating leftward and rightward shifts.
	tmp, tmp0, tmp1, tmp2, tmp3	Temporary register
	!	Logical NOT, that is, inversion of the boolean value of a variable or expression.
Floating point number	NaN	Not a number
Floating-point standard	SNaN	Signaling NaN
	QNaN	Quiet NaN

Contents

List of Instructions for RX Family

The RX Family has a total of 90 instructions.

While the RX600 Series supports all of the instructions, the RX100 Series and RX200 Series support the 82 instructions other than the eight for floating-point operations (FADD, FCMP, FDIV, FMUL, FSUB, FTOI, ITOF, and ROUND).

Mnemonic		Function	Instruction Described in Detail (on Page)	Instruction Code Described in Detail (on Page)
ABS		Absolute value	51	177
ADC		Addition with carry	52	178
ADD		Addition without carry	53	179
AND		Logical AND	55	181
BCLR		Clearing a bit	57	183
B*Cnd*	BGEU	Relative conditional branch	58	185
	BC		58	185
	BEQ		58	185
	BZ		58	185
	BGTU		58	185
	BPZ		58	185
	BGE		58	185
	BGT		58	185
	BO		58	185
	BLTU		58	185
	BNC		58	185
	BNE		58	185
	BNZ		58	185
	BLEU		58	185
	BN		58	185
	BLE		58	185
	BLT		58	185
	BNO		58	185
BM*Cnd*	BMGEU	Conditional bit transfer	59	187
	BMC		59	187
	BMEQ		59	187
	BMZ		59	187
	BMGTU		59	187
	BMPZ		59	187
	BMGE		59	187
	BMGT		59	187
	BMO		59	187
	BMLTU		59	187
	BMNC		59	187
	BMNE		59	187
	BMNZ		59	187
	BMLEU		59	187
	BMN		59	187
	BMLE		59	187
	BMLT		59	187
	BMNO		59	187

Mnemonic	Function	Instruction Described in Detail (on Page)	Instruction Code Described in Detail (on Page)
BNOT	Inverting a bit	61	188
BRA	Unconditional relative branch	62	190
BRK	Unconditional trap	63	191
BSET	Setting a bit	64	191
BSR	Relative subroutine branch	65	193
BTST	Testing a bit	66	194
CLRPSW	Clear a flag or bit in the PSW	67	196
CMP	Comparison	68	197
DIV	Signed division	69	199
DIVU	Unsigned division	71	201
EMUL	Signed multiplication	73	202
EMULU	Unsigned multiplication	75	203
FADD*1	Floating-point addition	77	204
FCMP*1	Floating-point comparison	79	205
FDIV*1	Floating-point division	82	206
FMUL*1	Floating-point multiplication	84	207
FSUB*1	Floating-point subtraction	87	208
FTOI*1	Floating point to integer conversion	90	209
INT	Software interrupt	93	209
ITOF*1	Integer to floating-point conversion	94	210
JMP	Unconditional jump	96	211
JSR	Jump to a subroutine	97	211
MACHI	Multiply-Accumulate the high-order word	98	212
MACLO	Multiply-Accumulate the low-order word	99	212
MAX	Selecting the highest value	100	213
MIN	Selecting the lowest value	101	214
MOV	Transferring data	102	215
MOVU	Transfer unsigned data	105	220
MUL	Multiplication	107	221
MULHI	Multiply the high-order word	109	223
MULLO	Multiply the low-order word	110	223
MVFACHI	Move the high-order longword from accumulator	111	224
MVFACMI	Move the middle-order longword from accumulator	112	224
MVFC	Transfer from a control register	113	225
MVTACHI	Move the high-order longword to accumulator	114	225
MVTACLO	Move the low-order longword to accumulator	115	226
MVTC	Transfer to a control register	116	226
MVTIPL*2 (privileged instruction)	Interrupt priority level setting	117	227
NEG	Two's complementation	118	228
NOP	No operation	119	228
NOT	Logical complementation	120	229
OR	Logical OR	121	230
POP	Restoring data from stack to register	123	231
POPC	Restoring a control register	124	232

Mnemonic		Function	Instruction Described in Detail (on Page)	Instruction Code Described in Detail (on Page)
POPM		Restoring multiple registers from the stack	125	232
PUSH		Saving data on the stack	126	233
PUSHC		Saving a control register	127	234
PUSHM		Saving multiple registers	128	234
RACW		Round the accumulator word	129	235
REVL		Endian conversion	131	235
REVW		Endian conversion	132	236
RMPA		Multiply-and-accumulate operation	133	236
ROLC		Rotation with carry to left	135	237
RORC		Rotation with carry to right	136	237
ROTL		Rotation to left	137	238
ROTR		Rotation to right	138	238
ROUND[*1]		Conversion from floating-point to integer	139	239
RTE (privileged instruction)		Return from the exception	142	239
RTFI (privileged instruction)		Return from the fast interrupt	143	240
RTS		Returning from a subroutine	144	240
RTSD		Releasing stack frame and returning from subroutine	145	240
SAT		Saturation of signed 32-bit data	147	241
SATR		Saturation of signed 64-bit data for RMPA	148	241
SBB		Subtraction with borrow	149	242
SC*Cnd*	SCGEU	Condition setting	150	243
	SCC		150	243
	SCEQ		150	243
	SCZ		150	243
	SCGTU		150	243
	SCPZ		150	243
	SCGE		150	243
	SCGT		150	243
	SCO		150	243
	SCLTU		150	243
	SCNC		150	243
	SCNE		150	243
	SCNZ		150	243
	SCLEU		150	243
	SCN		150	243
	SCLE		150	243
	SCLT		150	243
	SCNO		150	243
SCMPU		String comparison	152	243
SETPSW		Setting a flag or bit in the PSW	153	244
SHAR		Arithmetic shift to the right	154	245
SHLL		Logical and arithmetic shift to the left	155	246
SHLR		Logical shift to the right	156	247
SMOVB		Transferring a string backward	157	248

Mnemonic	Function	Instruction Described in Detail (on Page)	Instruction Code Described in Detail (on Page)
SMOVF	Transferring a string forward	158	248
SMOVU	Transferring a string	159	248
SSTR	Storing a string	160	249
STNZ	Transfer with condition	161	249
STZ	Transfer with condition	162	250
SUB	Subtraction without borrow	163	251
SUNTIL	Searching for a string	164	252
SWHILE	Searching for a string	166	252
TST	Logical test	168	253
WAIT (privileged instruction)	Waiting	169	254
XCHG	Exchanging values	170	254
XOR	Logical exclusive or	172	255

Notes: 1. Products of the RX100 Series and RX200 Series do not support the instructions for floating-point operations.

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

List of Instructions Classified by Type

Instruction Type	Mnemonic	Function	Instruction Described in Detail (on Page)	Instruction Code Described in Detail (on Page)
Arithmetic/logic instructions	ABS	Absolute value	51	177
	ADC	Addition with carry	52	178
	ADD	Addition without carry	53	179
	AND	Logical AND	55	181
	CMP	Comparison	68	197
	DIV	Signed division	69	199
	DIVU	Unsigned division	71	201
	EMUL	Signed multiplication	73	202
	EMULU	Unsigned multiplication	75	203
	MAX	Selecting the highest value	100	213
	MIN	Selecting the lowest value	101	214
	MUL	Multiplication	107	221
	NEG	Two’s complementation	118	228
	NOP	No operation	119	228
	NOT	Logical complementation	120	229
	OR	Logical OR	121	230
	RMPA	Multiply-and-accumulate operation	133	236
	ROLC	Rotation with carry to left	135	237
	RORC	Rotation with carry to right	136	237
	ROTL	Rotation to left	137	238
	ROTR	Rotation to right	138	238
	SAT	Saturation of signed 32-bit data	147	241
	SATR	Saturation of signed 64-bit data for RMPA	148	241
	SBB	Subtraction with borrow	149	242
	SHAR	Arithmetic shift to the right	154	245
	SHLL	Logical and arithmetic shift to the left	155	246
	SHLR	Logical shift to the right	156	247
	SUB	Subtraction without borrow	163	251
	TST	Logical test	168	253
	XOR	Logical exclusive or	172	255
Floating-point operation instructions*1	FADD	Floating-point addition	77	204
	FCMP	Floating-point comparison	79	205
	FDIV	Floating-point division	82	206
	FMUL	Floating-point multiplication	84	207
	FSUB	Floating-point subtraction	87	208
	FTOI	Floating point to integer conversion	90	209
	ITOF	Integer to floating-point conversion	94	210
	ROUND	Conversion from floating-point to integer	139	239

Instruction Type	Mnemonic		Function	Instruction Described in Detail (on Page)	Instruction Code Described in Detail (on Page)
Data transfer instructions	MOV		Transferring data	102	215
	MOVU		Transfer unsigned data	105	220
	POP		Restoring data from stack to register	123	231
	POPC		Restoring a control register	124	232
	POPM		Restoring multiple registers from the stack	125	232
	PUSH		Saving data on the stack	126	233
	PUSHC		Saving a control register	127	234
	PUSHM		Saving multiple registers	128	234
	REVL		Endian conversion	131	235
	REVW		Endian conversion	132	236
	SC*Cnd*	SCGEU	Condition setting	150	243
		SCC		150	243
		SCEQ		150	243
		SCZ		150	243
		SCGTU		150	243
		SCPZ		150	243
		SCGE		150	243
		SCGT		150	243
		SCO		150	243
		SCLTU		150	243
		SCNC		150	243
		SCNE		150	243
		SCNZ		150	243
		SCLEU		150	243
		SCN		150	243
		SCLE		150	243
		SCLT		150	243
		SCNO		150	243
	STNZ		Transfer with condition	161	249
	STZ		Transfer with condition	162	250
	XCHG		Exchanging values	170	254

Instruction Type	Mnemonic		Function	Instruction Described in Detail (on Page)	Instruction Code Described in Detail (on Page)
Branch instructions	B*Cnd*	BGEU	Relative conditional branch	58	185
		BC		58	185
		BEQ		58	185
		BZ		58	185
		BGTU		58	185
		BPZ		58	185
		BGE		58	185
		BGT		58	185
		BO		58	185
		BLTU		58	185
		BNC		58	185
		BNE		58	185
		BNZ		58	185
		BLEU		58	185
		BN		58	185
		BLE		58	185
		BLT		58	185
		BNO		58	185
	BRA		Unconditional relative branch	62	190
	BSR		Relative subroutine branch	65	193
	JMP		Unconditional jump	96	211
	JSR		Jump to a subroutine	97	211
	RTS		Returning from a subroutine	144	240
	RTSD		Releasing stack frame and returning from subroutine	145	240

Instruction Type	Mnemonic		Function	Instruction Described in Detail (on Page)	Instruction Code Described in Detail (on Page)
Bit manipulation instructions	BCLR		Clearing a bit	57	183
	BM*Cnd*	BMGEU	Conditional bit transfer	59	187
		BMC		59	187
		BMEQ		59	187
		BMZ		59	187
		BMGTU		59	187
		BMPZ		59	187
		BMGE		59	187
		BMGT		59	187
		BMO		59	187
		BMLTU		59	187
		BMNC		59	187
		BMNE		59	187
		BMNZ		59	187
		BMLEU		59	187
		BMN		59	187
		BMLE		59	187
		BMLT		59	187
		BMNO		59	187
	BNOT		Inverting a bit	61	188
	BSET		Setting a bit	64	191
	BTST		Testing a bit	66	194
String manipulation instructions	SCMPU		String comparison	152	243
	SMOVB		Transferring a string backward	157	248
	SMOVF		Transferring a string forward	158	248
	SMOVU		Transferring a string	159	248
	SSTR		Storing a string	160	249
	SUNTIL		Searching for a string	164	252
	SWHILE		Searching for a string	166	252
System manipulation instructions	BRK		Unconditional trap	63	191
	CLRPSW		Clear a flag or bit in the PSW	67	196
	INT		Software interrupt	93	209
	MVFC		Transfer from a control register	113	225
	MVTC		Transfer to a control register	116	226
	MVTIPL[*2] (privileged instruction)		Interrupt priority level setting	117	227
	RTE (privileged instruction)		Return from the exception	142	239
	RTFI (privileged instruction)		Return from the fast interrupt	143	240
	SETPSW		Setting a flag or bit in the PSW	153	244
	WAIT (privileged instruction)		Waiting	169	254

Instruction Type	Mnemonic	Function	Instruction Described in Detail (on Page)	Instruction Code Described in Detail (on Page)
DSP instructions	MACHI	Multiply-Accumulate the high-order word	98	212
	MACLO	Multiply-Accumulate the low-order word	99	212
	MULHI	Multiply the high-order word	109	223
	MULLO	Multiply the low-order word	110	223
	MVFACHI	Move the high-order longword from accumulator	111	224
	MVFACMI	Move the middle-order longword from accumulator	112	224
	MVTACHI	Move the high-order longword to accumulator	114	225
	MVTACLO	Move the low-order longword to accumulator	115	226
	RACW	Round the accumulator word	129	235

Notes: 1. Products of the RX100 Series and RX200 Series do not support the instructions for floating-point operations.
2. Products of the RX610 Group do not support the MVTIPL instruction.

Section 1 CPU Functions

The RX CPU has short formats for frequently used instructions, facilitating the development of efficient programs that take up less memory. Moreover, some instructions are executable in one clock cycle, and this realizes high-speed arithmetic processing.

The RX CPU has a total of 90 instructions, consisting of 73 basic instructions, eight floating-point operation instructions, and nine DSP instructions.

While the RX600 Series supports all of the instructions, the RX100 Series and RX200 Series support the 82 instructions other than the eight for floating-point operations.

The RX CPU has 10 addressing modes, with register-register operations, register-memory operations, and bitwise operations included. Data transfer between memory locations is also possible. An internal multiplier is included for high-speed multiplication.

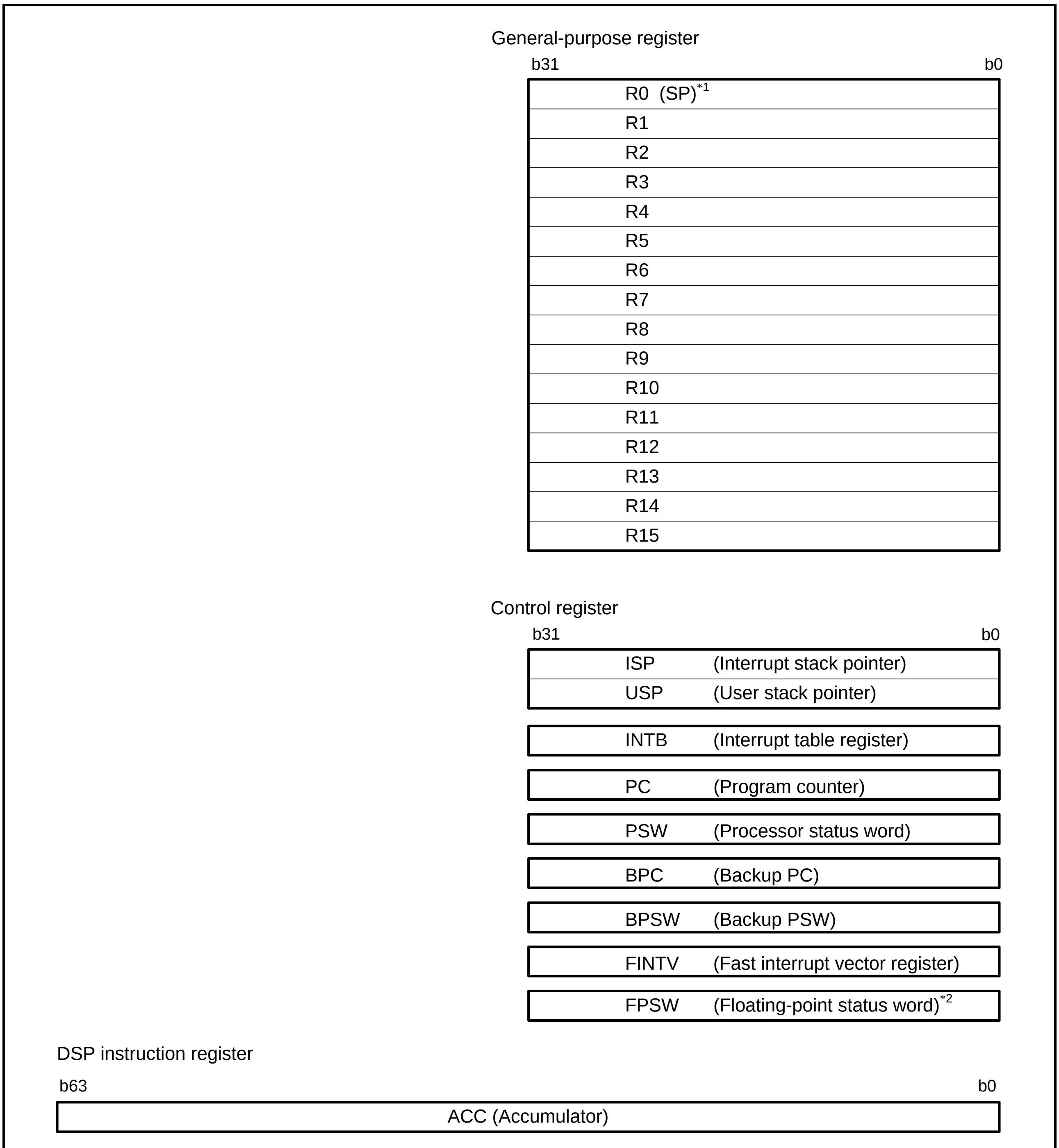
1.1 Features

- High instruction execution rate: One instruction in 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
- Basic instructions: 73
  Relative branch instructions to suit branch distances
  Variable-length instruction format (lengths from one to eight bytes)
  Short formats are provided for frequently used instructions.
- Floating-point operation instructions: 8*
- DSP instructions: 9
  Supports 16-bit × 16-bit multiplication and multiply-and-accumulate operations.
  Rounds the data in the accumulator.
- Addressing modes: 10
- Processor modes
  Supports a supervisor mode and a user mode.
- Floating-point operation unit*
  Supports single precision (32-bit) floating-point.
  Supports data types and exceptions conforming to the IEEE754 standard.
- Memory protection unit (as an optional function)
- Data arrangement
  Selectable as little endian or big endian

Note: * Products of the RX100 Series and RX200 Series do not support this feature.

1.2 Register Set of the CPU

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



Notes: 1. The stack pointer (SP) can be the interrupt stack pointer (ISP) or user stack pointer (USP), according to the value of the U bit in the PSW.
2. The FPSW is not specifiable as an operand in products of the RX100 Series and RX200 Series.

Figure 1.1 Register Set of the CPU

1.2.1 General-Purpose Registers (R0 to R15)

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

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

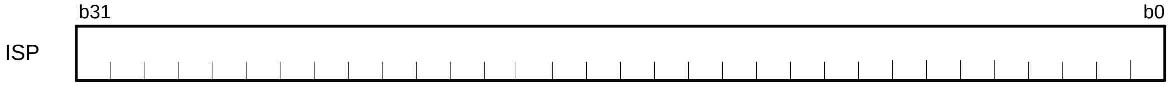
1.2.2 Control Registers

This CPU has the following nine control registers.

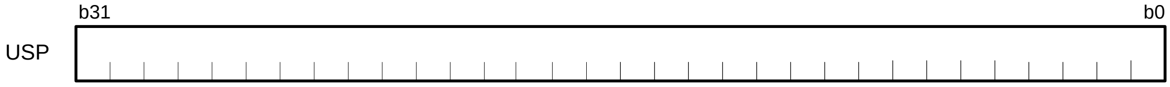
- Interrupt stack pointer (ISP)
- User stack pointer (USP)
- Interrupt table register (INTB)
- Program counter (PC)
- Processor status word (PSW)
- Backup PC (BPC)
- Backup PSW (BPSW)
- Fast interrupt vector register (FINTV)
- Floating-point status word (FPSW)*

Note: * The FPSW is not specifiable as an operand in products of the RX100 Series and RX200 Series.

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



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

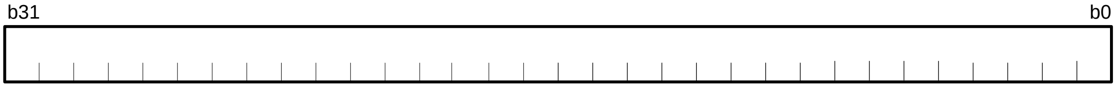


Value after reset: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

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

Set the ISP or USP to a multiple of four, as this reduces the numbers of cycles required to execute interrupt sequences and instructions entailing stack manipulation.

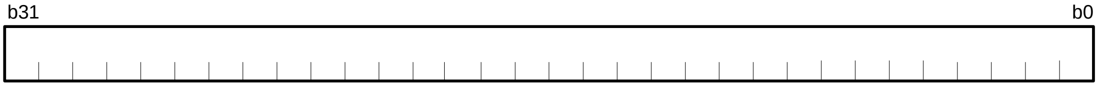
1.2.2.2 Interrupt Table Register (INTB)



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

Note : * Since the interrupt priority levels are from 0 to 7 for the RX610 Group, bit 27 is reserved. Writing to bit 27 is ineffective.

Reserved

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	When writing, write 0 to these bits. The value read is always 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	When writing, write 0 to these bits. The value read is always 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	When writing, write 0 to these bits. The value read is always 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

Bit	Symbol	Bit Name	Description	R/W
b31 to b28	—	Reserved	When writing, write 0 to these bits. The value read is always 0.	R/W

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

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

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

4. Since the interrupt priority levels are from 0 to 7 for the RX610 Group, bit 27 is reserved. Writing to bit 27 is ineffective.

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

C flag (Carry flag)

This flag indicates whether a carry, borrow, or shift-out has occurred as the result of an operation.

Z flag (Zero flag)

This flag indicates that the result of an operation was 0.

S flag (Sign flag)

This flag indicates that the result of an operation was negative.

O flag (Overflow flag)

This flag indicates that an overflow occurred during an operation.

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)

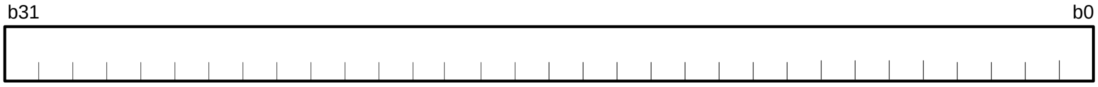
The IPL[3:0] bits specify the processor interrupt priority level as one of sixteen levels from zero to fifteen, where priority level zero is the lowest and priority level fifteen the highest. When the priority level of a requested interrupt is higher than the processor interrupt priority level, the interrupt 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	When writing, write 0 to this bit. The value read is always 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	When writing, write 0 to these bits. The value read is always 0.	R/W
b26	FV[3]	Invalid operation flag	0: No invalid operation has been encountered. 1: Invalid operation has been encountered.[8]	R/W

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

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

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

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

Note: The FPSW is not specifiable as an operand in products of the RX100 Series and RX200 Series.

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

These bits specify the floating-point rounding-mode.

Explanation of Floating-Point Rounding Modes

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

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

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

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

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

- The bit that has been set to 1 is cleared to 0 when the FPU instruction is executed.

- When 0 is written to the bit by the MVTC and POPC instructions, the bit is set to 0; the bit retains the previous value when 1 is written by the instruction.

DN bit (0 flush bit of denormalized number)

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

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

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

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

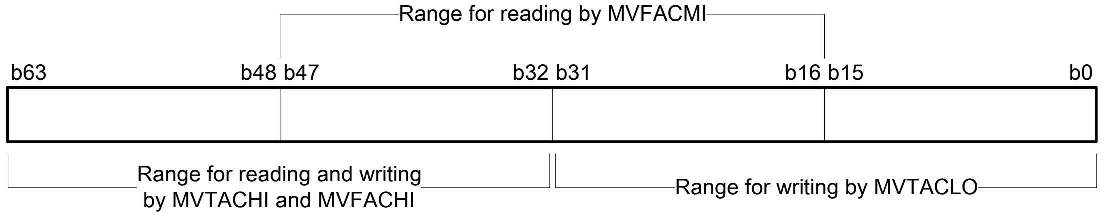
While the exception handling enable bit (Ej) is 0 (exception handling is masked), if any of five floating-point exceptions specified in the IEEE754 standard is generated, the corresponding bit is set to 1.

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

FS flag (Floating-point error summary flag)

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

1.2.3 Accumulator (ACC)


Range for reading by MVFACMI
b63
b48 b47
b32 b31
b16 b15
b0
Range for reading and writing by MVTACHI and MVFACHI
Range for writing by MVTACLO

Value after reset: Undefined

The accumulator (ACC) is a 64-bit register used for DSP instructions. The accumulator is also used for the multiply and multiply-and-accumulate instructions; EMUL, EMULU, FMUL, MUL, and RMPA, in which case the prior value in the accumulator 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 higher-order 32 bits (bits 63 to 32) and the lower-order 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 higher-order 32 bits (bits 63 to 32) and the middle 32 bits (bits 47 to 16), respectively.

1.3 Floating-Point Exceptions

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

Note: Since products of the RX100 Series and RX200 Series do not support instructions for floating-point operations, the floating-point exception does not occur.

1.3.1 Overflow

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

Table 1.1 Operation Results When an Overflow Exception Has Occurred

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

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

1.3.2 Underflow

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

Table 1.2 Operation Results When an Underflow Exception Has Occurred

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

1.3.3 Inexact

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

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

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

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

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

1.3.4 Division-by-Zero

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

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

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

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

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

1.3.5 Invalid Operation

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

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

	Operation Result (Value in the Destination Register)	
Occurrence Condition	**EV = 0**	**EV = 1**
Operation on SNaN operands	QNaN	No change
+∞+(–∞), +∞–(+∞), –∞–(–∞)		
0 × ∞		
0 ÷ 0, ∞ ÷ ∞		
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 mantissa part is 0.
Using an SNaN as a source operand in an operation generates an invalid operation. Using an SNaN as the initial value of a variable facilitates the detection of bugs in programs. Note that the hardware will not generate an SNaN.

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

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

Table 1.6 Rules for Generating QNaNs

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

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

1.3.6 Unimplemented Processing

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

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

1.4 Processor Mode

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

1.4.1 Supervisor Mode

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

1.4.2 User Mode

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

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

1.4.3 Privileged Instruction

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

1.4.4 Switching Between Processor Modes

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

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

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

1.5 Data Types

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

1.5.1 Integer

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

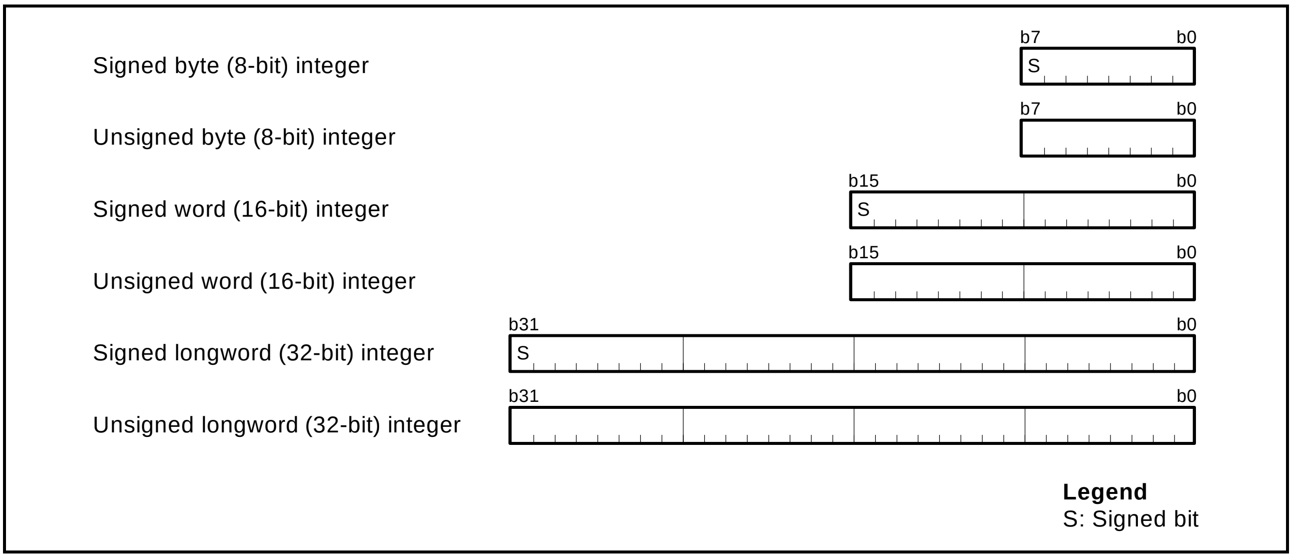

Signed byte (8-bit) integer
b7
b0
S
Unsigned byte (8-bit) integer
b7
b0
Signed word (16-bit) integer
b15
b0
S
Unsigned word (16-bit) integer
b15
b0
Signed longword (32-bit) integer
b31
b0
S
Unsigned longword (32-bit) integer
b31
b0
Legend
S: Signed bit

Figure 1.2 Integer

1.5.2 Floating-Point

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

Note: Since products of the RX100 Series and RX200 Series do not support instructions for floating-point operations, the floating-point exception does not occur.

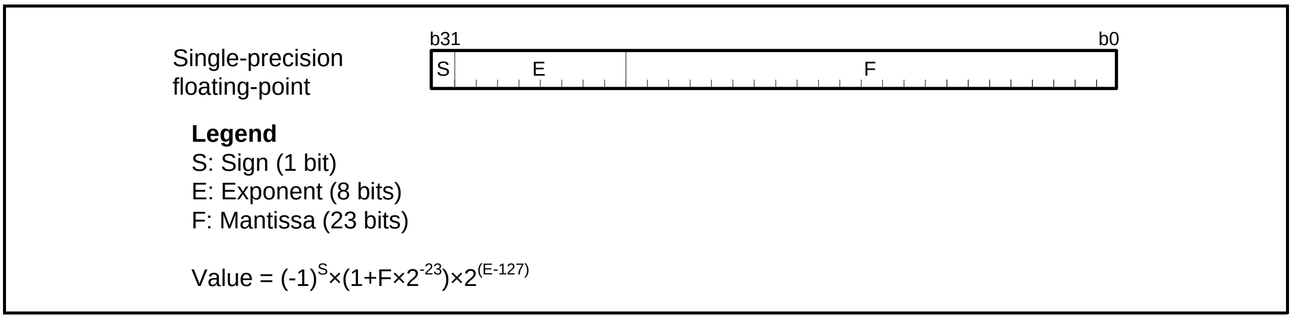


Figure 1.3 Floating-Point

The floating-point format supports the values listed below.

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

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

1.5.3 Bitwise Operations

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

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

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

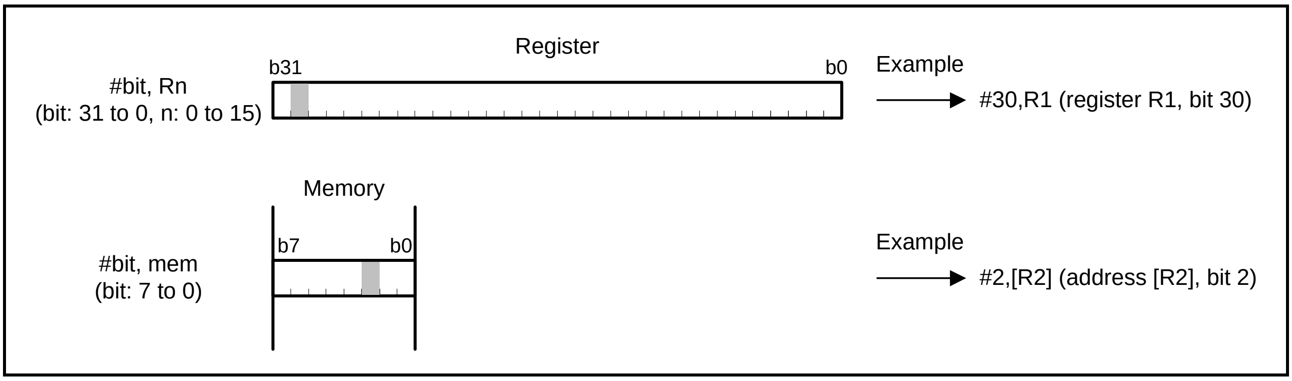

Register
#bit, Rn
(bit: 31 to 0, n: 0 to 15)
b31
b0
Example
#30,R1 (register R1, bit 30)
Memory
#bit, mem
(bit: 7 to 0)
b7
b0
Example
#2,[R2] (address [R2], bit 2)

Figure 1.4 Bit

1.5.4 Strings

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

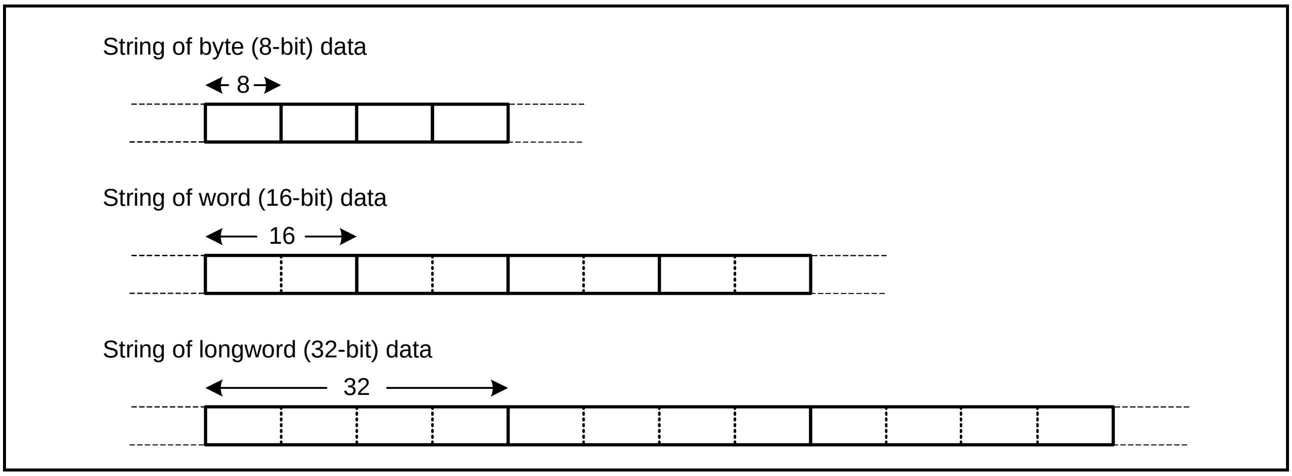

String of byte (8-bit) data
8
String of word (16-bit) data
16
String of longword (32-bit) data
32

Figure 1.5 String

1.6 Data Arrangement

1.6.1 Data Arrangement in Registers

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

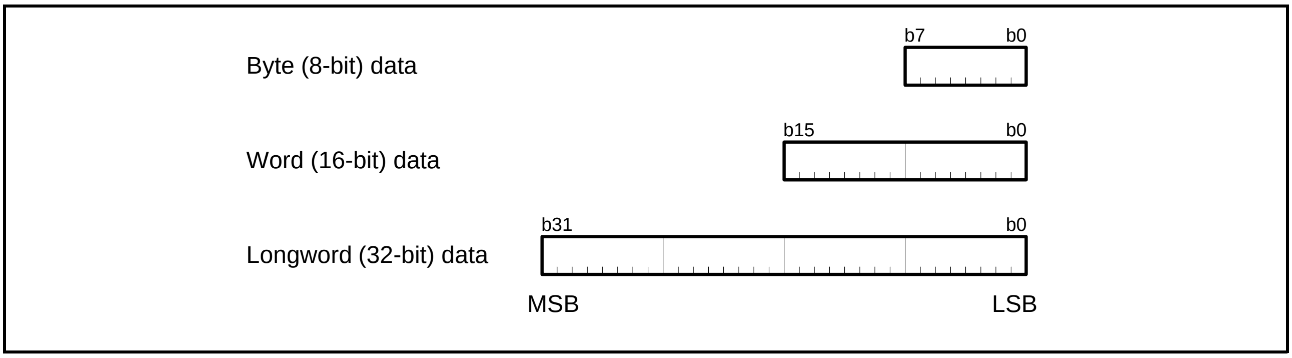
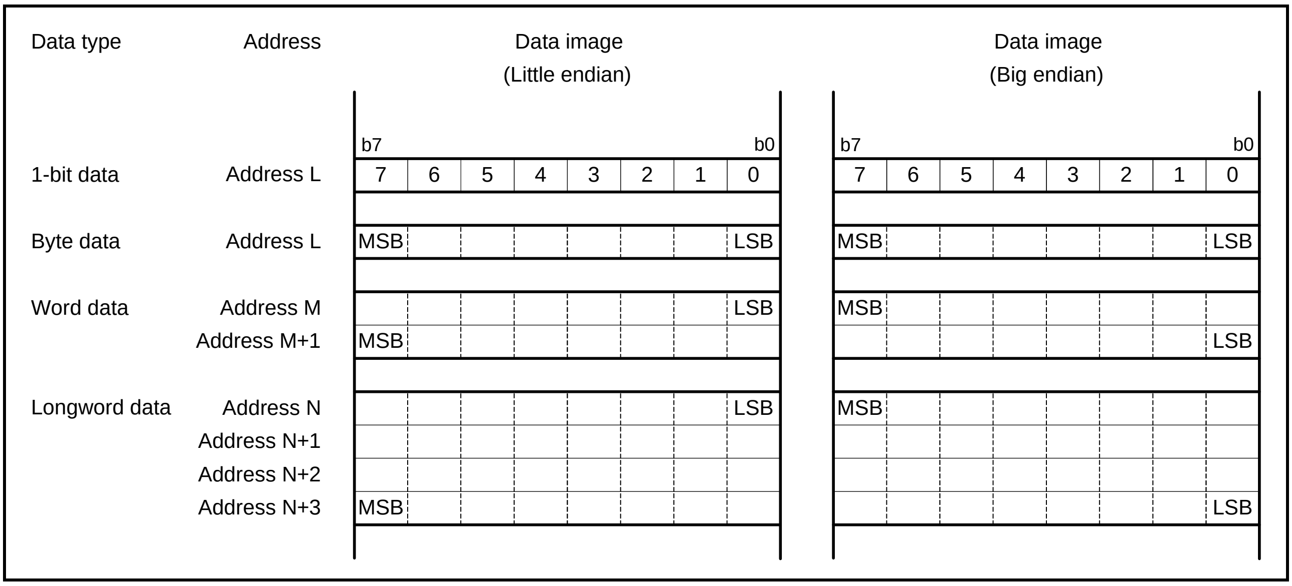


Figure 1.6 Data Arrangement in Registers

1.6.2 Data Arrangement in Memory

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



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

Figure 1.7 Data Arrangement in Memory

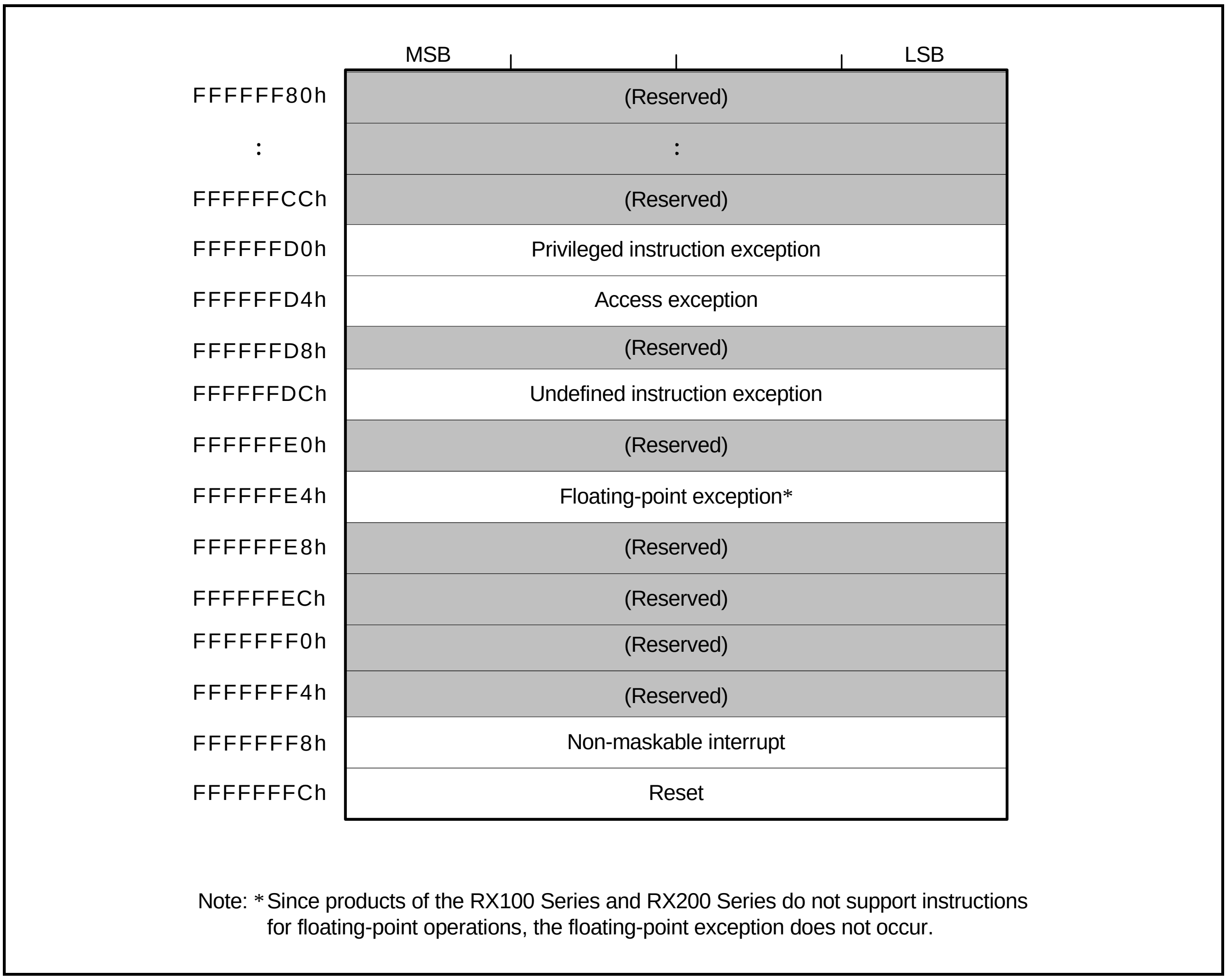
1.7 Vector Table

There are two types of vector table: 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.

Note: * Since products of the RX100 Series and RX200 Series do not support instructions for floating-point operations, the floating-point exception does not occur.



FFFFFF80h	(Reserved)
:	:
FFFFFFCCh	(Reserved)
FFFFFFD0h	Privileged instruction exception
FFFFFFD4h	Access exception
FFFFFFD8h	(Reserved)
FFFFFFDCh	Undefined instruction exception
FFFFFFE0h	(Reserved)
FFFFFFE4h	Floating-point exception*
FFFFFFE8h	(Reserved)
FFFFFFECh	(Reserved)
FFFFFFF0h	(Reserved)
FFFFFFF4h	(Reserved)
FFFFFFF8h	Non-maskable interrupt
FFFFFFFCh	Reset

Note: * Since products of the RX100 Series and RX200 Series do not support instructions for floating-point operations, the floating-point exception does not occur.

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.

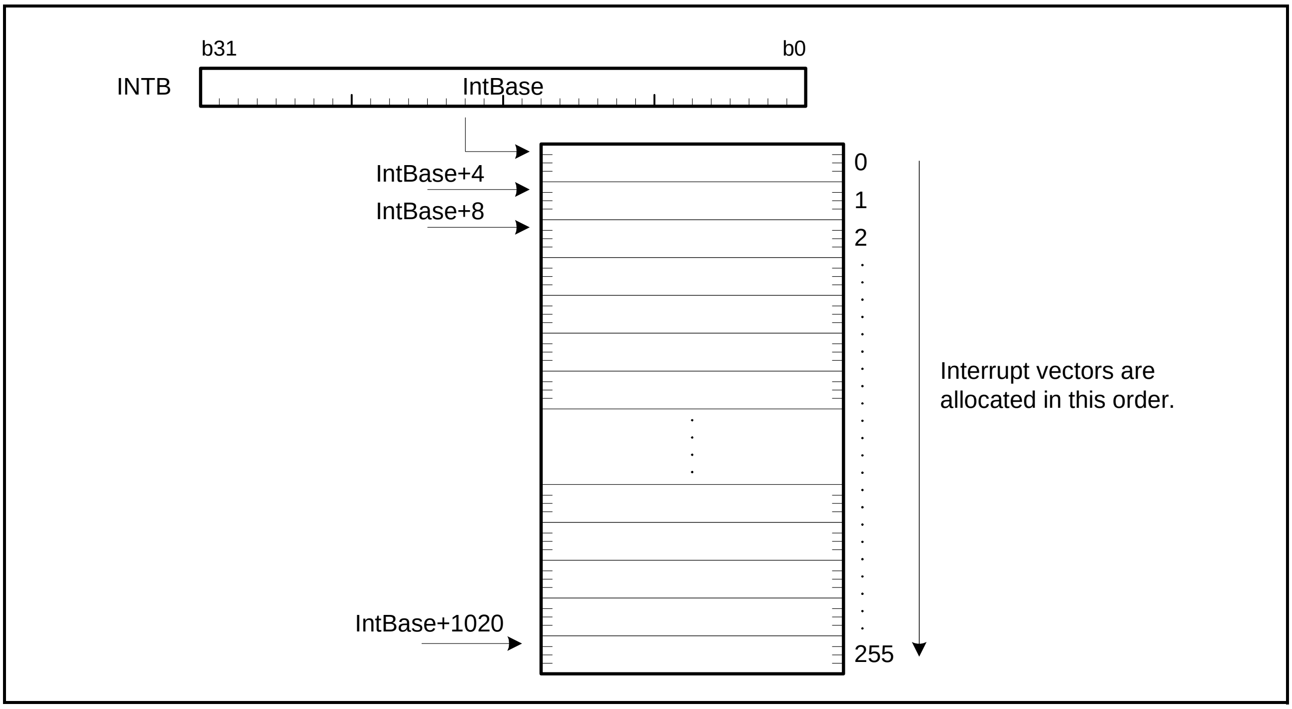


Figure 1.9 Relocatable Vector Table

1.8 Address Space

The address space of the RX 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 RX-CPU is depicted in figure 1.10. For all regions, the designation may differ with the product and operating mode. For details, see the hardware manuals for the respective products.

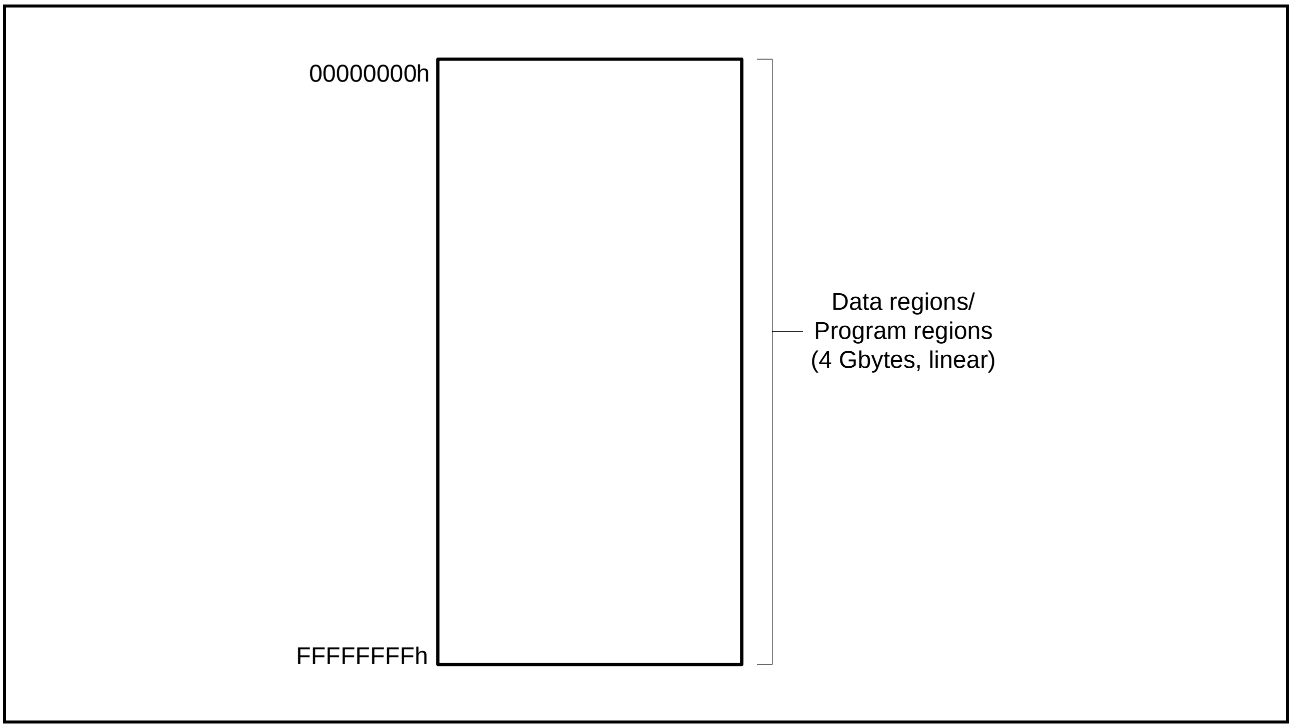


Figure 1.10 Address Space

Section 2 Addressing Modes

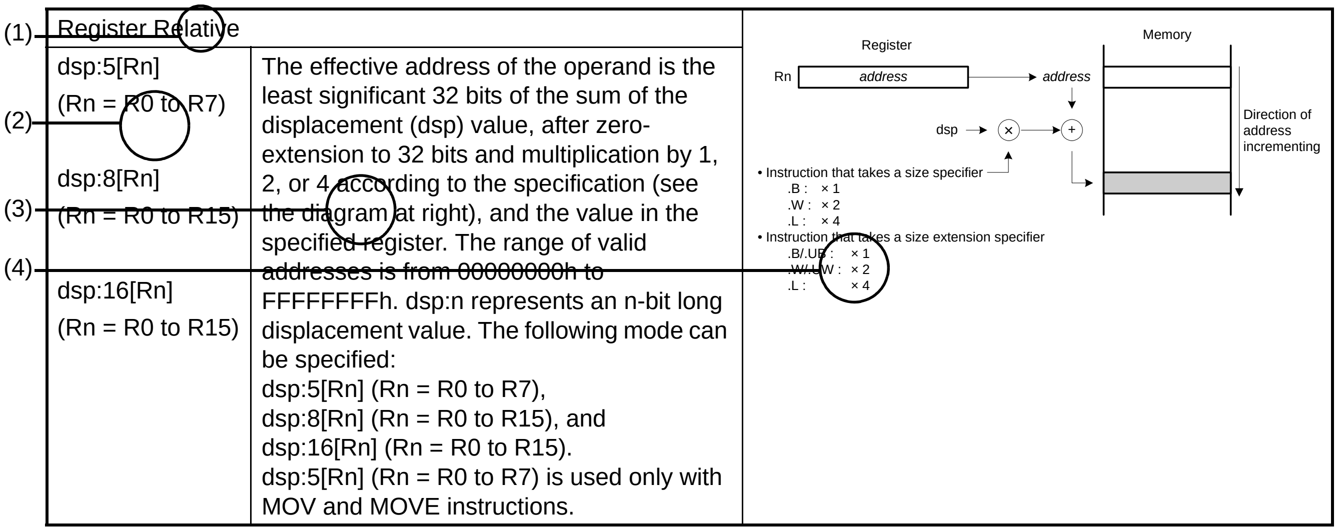
The following is a description of the notation and operations of each addressing mode.

There are 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.



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

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

(1) Name

The name of the addressing mode is given here.

(2) Symbolic notation

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

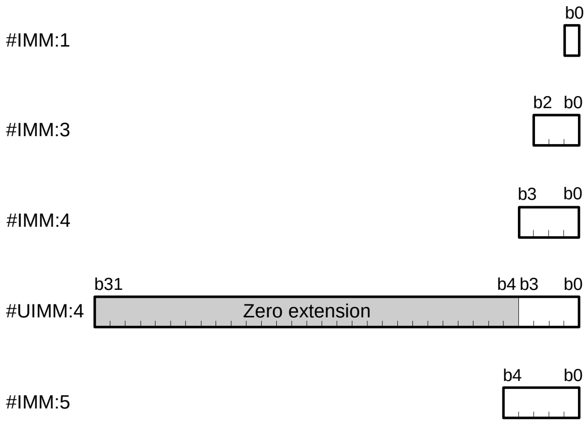
(3) Description

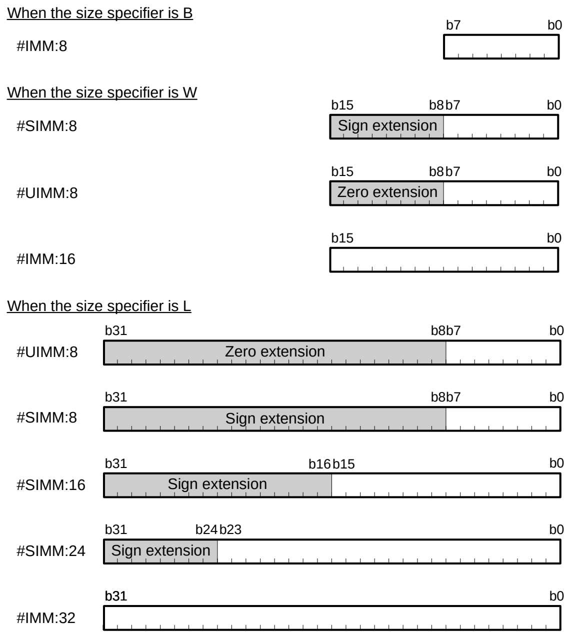
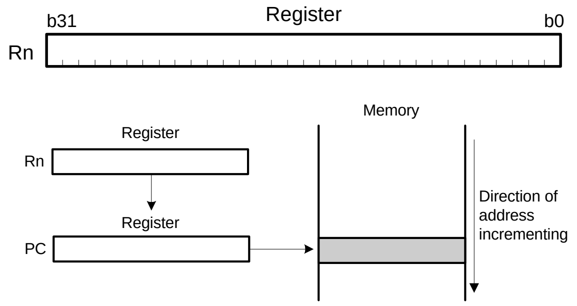
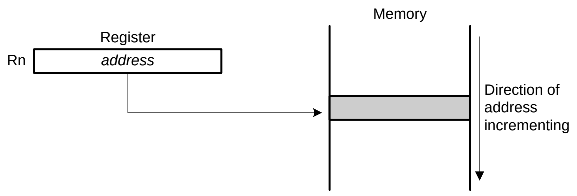
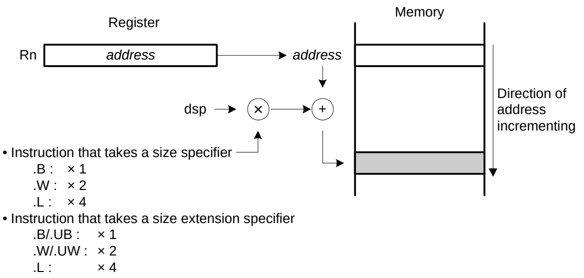
The operation and effective address range are described here.

(4) Operation diagram

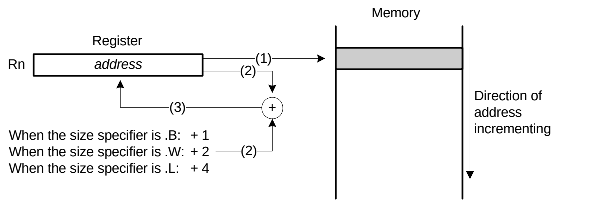
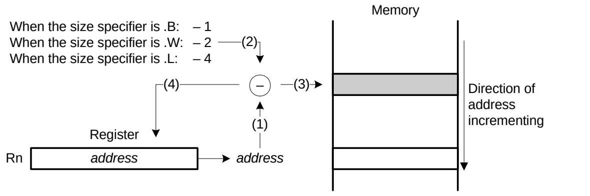
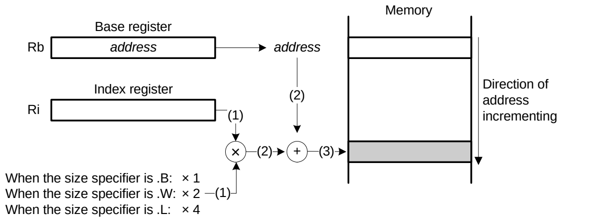
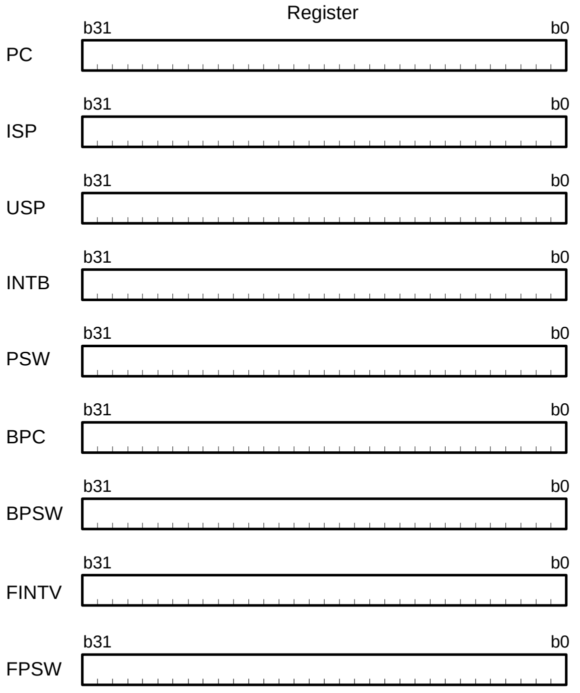
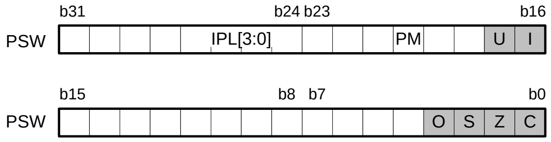
The operation of the addressing mode is illustrated here.

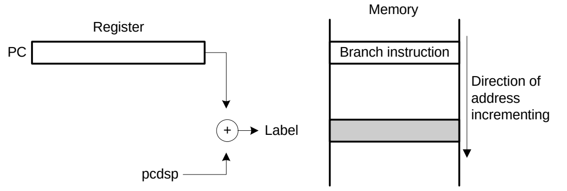
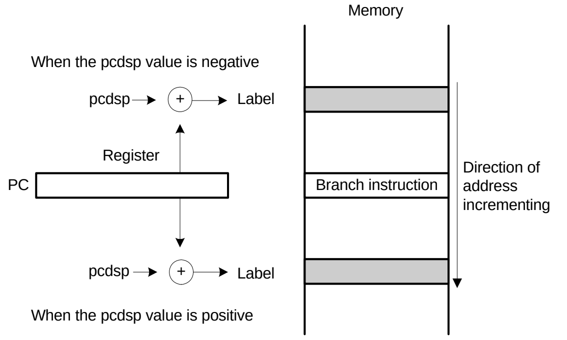
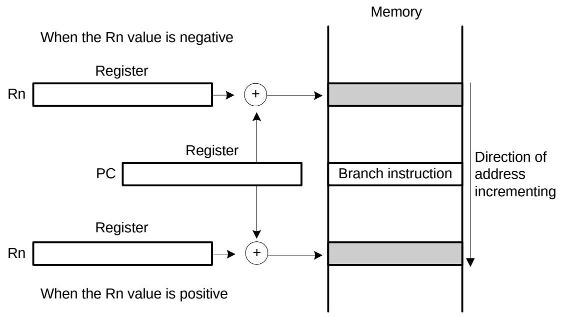
2.2 Addressing Modes

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

Immediate		
#IMM:8 #SIMM:8 #UIMM:8 #IMM:16 #SIMM:16 #SIMM:24 #IMM:32	The operand is the value specified by the immediate value. In addition, the operand will be the result of zero-extending or sign-extending the immediate value when it is specified by #UIMM or #SIMM. #IMM:n, #UIMM:n, and #SIMM:n represent n-bit long immediate values. For the range of IMM, refer to section 2.2.1, Ranges for Immediate Values.	
Register Direct		
Rn (Rn = R0 to R15)	The operand is the specified register. In addition, the Rn value is transferred to the program counter (PC) when this addressing mode is used with JMP and JSR instructions. The range of valid addresses is from 00000000h to FFFFFFFFh. Rn (Rn = R0 to R15) can be specified.	
Register Indirect		
[Rn] (Rn = R0 to R15)	The value in the specified register is the effective address of the operand. The range of valid addresses is from 00000000h to FFFFFFFFh. [Rn] (Rn = R0 to R15) can be specified.	
Register Relative		
dsp:5[Rn] (Rn = R0 to R7) dsp:8[Rn] (Rn = R0 to R15) dsp:16[Rn] (Rn = R0 to R15)	The effective address of the operand is the least significant 32 bits of the sum of the displacement (dsp) value, after zero-extension to 32 bits and multiplication by 1, 2, or 4 according to the specification (see the diagram at right), and the value in the specified register. The range of valid addresses is from 00000000h to FFFFFFFFh. dsp:n represents an n-bit long displacement value. The following mode can be specified: dsp:5[Rn] (Rn = R0 to R7), dsp:8[Rn] (Rn = R0 to R15), and dsp:16[Rn] (Rn = R0 to R15). dsp:5[Rn] (Rn = R0 to R7) is used only with MOV and MOVE instructions.	

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

Post-increment Register Indirect		 Memory Register Rn address (1) (2) (3) + When the size specifier is .B: + 1 When the size specifier is .W: + 2 When the size specifier is .L: + 4 (2) Direction of address incrementing
[Rn+] (Rn = R0 to R15)	The value in the specified register is the effective address of the operand. The range of valid addresses is from 00000000h to FFFFFFFFh. After the operation, 1, 2, or 4 is added to the value in the specified register according to the size specifier: .B, .W, or .L. This addressing mode is used with MOV and MOVU instructions.	
Pre-decrement Register Indirect		 When the size specifier is .B: – 1 When the size specifier is .W: – 2 When the size specifier is .L: – 4 (2) Memory (4) – (3) (1) Register Rn address address Direction of address incrementing
[–Rn] (Rn = R0 to R15)	According to the size specifier: .B, .W, or .L, 1, 2, or 4 is subtracted from the value in the specified register. The value after the operation is the effective address of the operand. The range of valid addresses is from 00000000h to FFFFFFFFh. This addressing mode is used with MOV and MOVU instructions.	
Indexed Register Indirect		 Memory Base register Rb address address Index register Ri (2) (1) × (2) + (3) When the size specifier is .B: × 1 When the size specifier is .W: × 2 When the size specifier is .L: × 4 (1) Direction of address incrementing
[Ri,Rb] (Ri = R0 to R15, Rb = R0 to R15)	The effective address of the operand is the least significant 32 bits of the sum of the value in the index register (Ri), multiplied by 1, 2, or 4 according to the size specifier: .B, .W, or .L, and the value in the base register (Rb). The range of valid addresses is from 00000000h to FFFFFFFFh. This addressing mode is used with MOV and MOVU instructions.	
Control Register Direct		 Register b31 b0 PC ISP USP INTB PSW BPC BPSW FINTV FPSW
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.	
PSW Direct		 b31 b24 b23 b16 PSW IPL[3:0] PM U I b15 b8 b7 b0 PSW O S Z C
C Z S O I U	The operand is the specified flag or bit. This addressing mode is used with CLRPSW and SETPSW instructions.	

Program Counter Relative		
pcdsp:3	When the branch distance specifier is .S, the effective address is the least significant 32 bits of the unsigned sum of the value in the program counter (PC) and the displacement (pcdsp) value. The range of the branch is from 3 to 10. The range of valid addresses is from 00000000h to FFFFFFFFh. This addressing mode is used with B*Cnd* (where *Cnd*==EQ/Z or NE/NZ) and BRA instructions.	 Memory Register PC Branch instruction Direction of address incrementing + Label pcdsp
pcdsp:8 pcdsp:16 pcdsp:24	When the branch distance specifier is .B, .W, or .A, the effective address is the signed sum of the value in the program counter (PC) and the displacement (pcdsp) value. The range of pcdsp depends on the branch distance specifier. For .B: –128 ≤ pcdsp:8 ≤ 127 For .W: –32768 ≤ pcdsp:16 ≤ 32767 For .A: –8388608 ≤ pcdsp:24 ≤ 8388607 The range of valid addresses is from 00000000h to FFFFFFFFh. When the branch distance specifier is .B, this addressing mode is used with B*Cnd* and BRA instructions. When the branch distance specifier is .W, this addressing mode is used with B*Cnd* (where *Cnd*==EQ/Z or NE/NZ), BRA, and BSR instructions. When the branch distance specifier is .A, this addressing mode is used with BRA and BSR instructions.	 Memory When the pcdsp value is negative pcdsp + Label Register PC Branch instruction Direction of address incrementing pcdsp + Label When the pcdsp value is positive
Rn (Rn = R0 to R15)	The effective address is the signed sum of the value in the program counter (PC) and the Rn value. The range of the Rn value is from –2147483648 to 2147483647. The range of valid addresses is from 00000000h to FFFFFFFFh. This addressing mode is used with BRA(.L) and BSR(.L) instructions.	 Memory When the Rn value is negative Register Rn + Register PC Branch instruction Direction of address incrementing Register Rn + When the Rn value is positive

2.2.1 Ranges for Immediate Values

Ranges for immediate values are listed in table 2.1.

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

Table 2.1 Ranges for Immediate Values

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

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

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

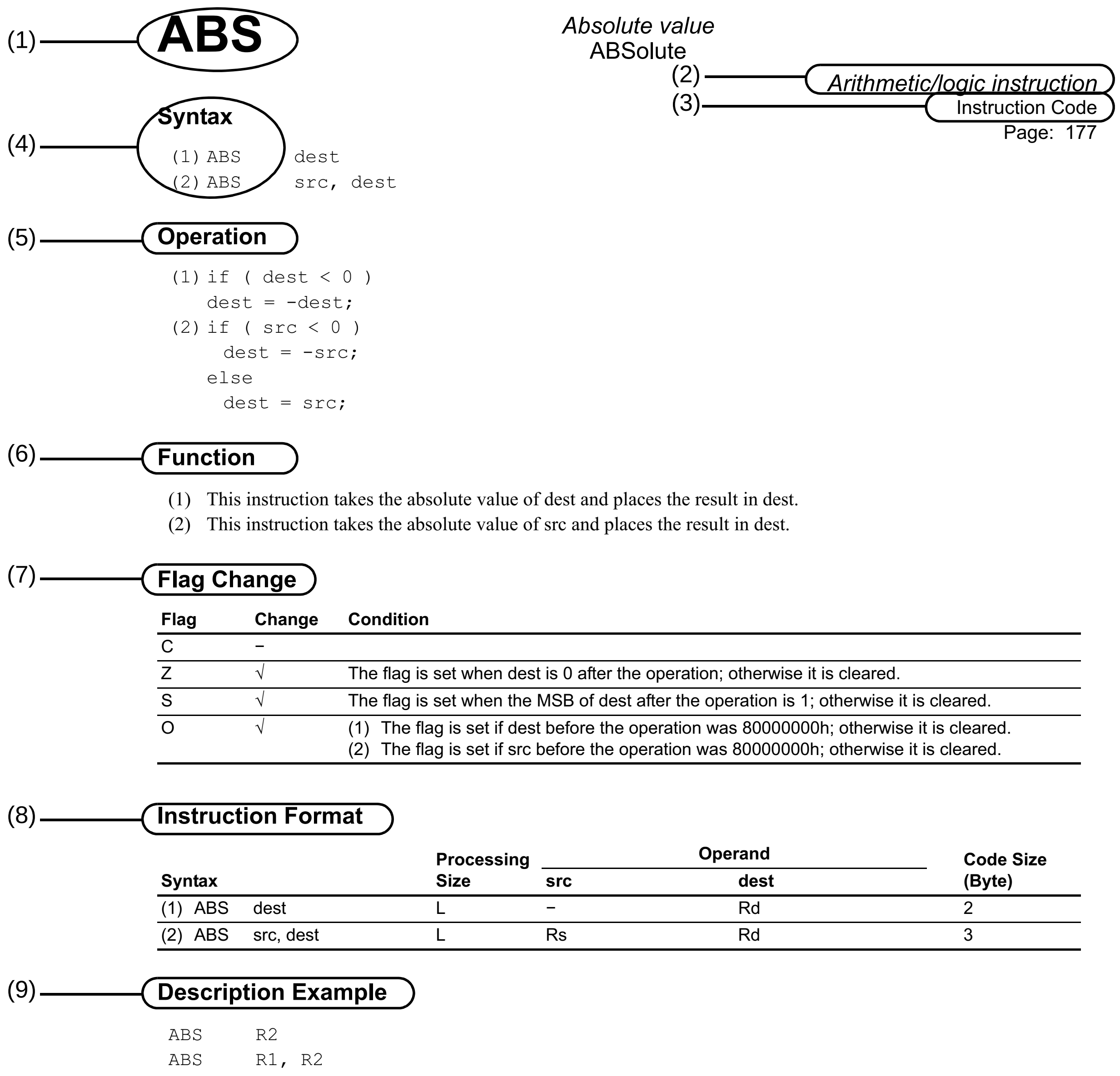
3. For the ranges of immediate values for INT and RTSD instructions, see the relevant descriptions under section 3.2, Instructions in Detail.

Section 3 Instruction Descriptions

3.1 Guide to This Section

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

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



ABS

Absolute value
ABSolute

Arithmetic/logic instruction
Instruction Code
Page: 177

Syntax

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

Operation

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

Function

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

Flag Change

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

Instruction Format

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
```

(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

(b) Pseudo-functions

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

(c) Special notation

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

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

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

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

(6) Function

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

(7) Flag Change

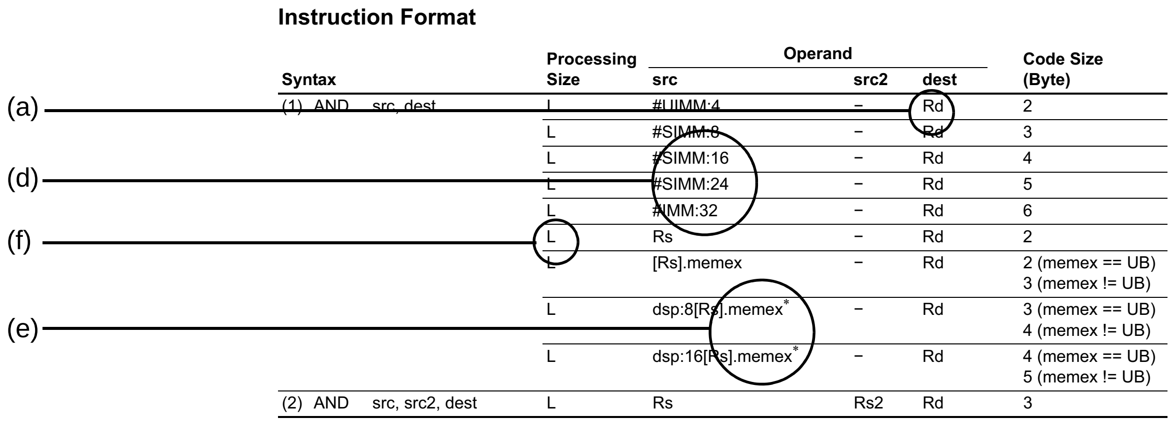
Indicates changes in the states of flags (O, S, Z, and C) in the PSW. For floating-point instructions, changes in the states of flags (FX, FU, FZ, FO, FV, CE, CX, CU, CZ, CO, and CV) in the FPSW are also indicated.

The symbols in the table mean the following:

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

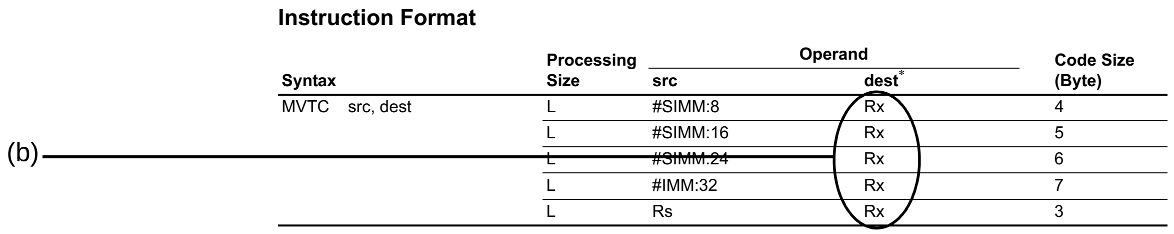
(8) Instruction Format

Indicates the instruction format.



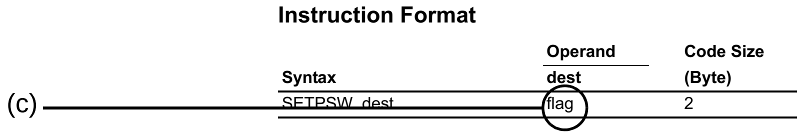
Instruction Format

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



Instruction Format

		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



Instruction Format

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

(a) Registers

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

(b) Control registers

Rx indicates that 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.

(c) Flag and bit

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

(d) Immediate value

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

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

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

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

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

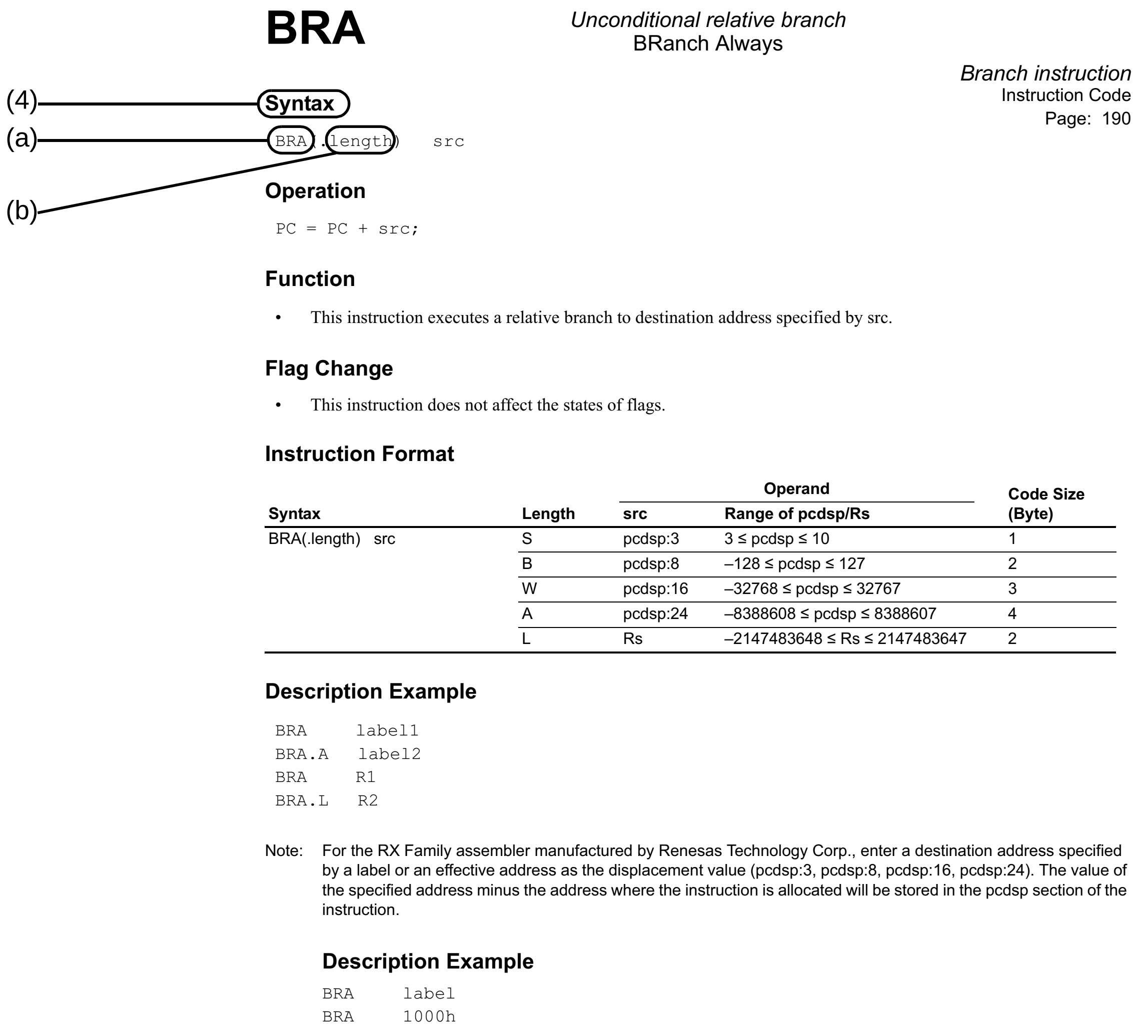
(f) Processing size

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

(9) Description Example

Shows a description example for the instruction.

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



BRA

Unconditional relative branch
BRanch Always

Branch instruction
Instruction Code
Page: 190

Syntax

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

Operation

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

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

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

Note: For the RX Family assembler manufactured by Renesas Technology 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.

3.2 Instructions in Detail

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

ABS

Absolute value
ABSolute

Arithmetic/logic instruction
Instruction Code
Page: 177

Syntax

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

Operation

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

Function

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

Flag Change

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

Instruction Format

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

Description Example

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

ADC

Addition with carry
ADd with Carry

Arithmetic/logic instruction
Instruction Code
Page: 178

Syntax

```
ADC     src, dest
```

Operation

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

Function

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

Flag Change

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

Instruction Format

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

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

Description Example

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

ADD

Addition without carry
ADD

Arithmetic/logic instruction
Instruction Code
Page: 179

Syntax

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

Operation

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

Function

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

Flag Change

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

Instruction Format

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

Note: * For the RX Family assembler manufactured by Renesas Technology 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

Arithmetic/logic instruction
Instruction Code
Page: 181

Syntax

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

Operation

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

Function

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

Flag Change

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

Instruction Format

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

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

Description Example

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

BCLR

Clearing a bit
Bit CLeaR

Bit manipulation instruction
Instruction Code
Page: 183

Syntax

```
BCLR    src, dest
```

Operation

(1) When dest is a memory location:

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

(2) When dest is a register:

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

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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*

Relative conditional branch
Branch Conditionally

B*Cnd*

Branch instruction
Instruction Code
Page: 185

Syntax

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

Operation

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

Function

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

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

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

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

Description Example

```
BC      label
BC      1000h
```

BM*Cnd*

Conditional bit transfer
Bit Move Conditional

Bit manipulation instruction
Instruction Code
Page: 187

Syntax

```
BMCnd     src, dest
```

Operation

(1) When dest is a memory location:

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

(2) When dest is a register:

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

Function

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

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

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Processing Size	Operand		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

Inverting a bit
Bit NOT

Bit manipulation instruction
Instruction Code
Page: 188

Syntax

```
BNOT    src, dest
```

Operation

(1) When dest is a memory location:

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

(2) When dest is a register:

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

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

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

BRA

Unconditional relative branch
BRanch Always

Branch instruction
Instruction Code
Page: 190

Syntax

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

Operation

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

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

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

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

Description Example

```
BRA     label
BRA     1000h
```

BRK

Unconditional trap
BReaK

System manipulation instruction
Instruction Code
Page: 191

Syntax

```
BRK
```

Operation

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

Function

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

Flag Change

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

Instruction Format

Syntax	Code Size (Byte)
BRK	1

Description Example

```
BRK
```

BSET

Setting a bit
Bit SET

Bit manipulation instruction
Instruction Code
Page: 191

Syntax

```
BSET    src, dest
```

Operation

(1) When dest is a memory location:

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

(2) When dest is a register:

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

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

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

BSR

Relative subroutine branch
Branch to SubRoutine

Branch instruction
Instruction Code
Page: 193

Syntax

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

Operation

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

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

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

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

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

Description Example

```
BSR     label
BSR     1000h
```

BTST

Testing a bit
Bit TeST

Bit manipulation instruction
Instruction Code
Page: 194

Syntax

```
BTST    src, src2
```

Operation

(1) When src2 is a memory location:

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

(2) When src2 is a register:

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

Function

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

Flag Change

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

Instruction Format

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

Description Example

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

CLRPSW

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

System manipulation instruction
Instruction Code
Page: 196

Syntax

```
CLRPSW  dest
```

Operation

```
dest = 0;
```

Function

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

Flag Change

Flag	Change	Condition
C	*	
Z	*	
S	*	
O	*	

Note: * The specified flag becomes 0.

Instruction Format

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

Description Example

```
CLRPSW  C
CLRPSW  Z
```

CMP

Comparison
CoMPare

Arithmetic/logic instruction
Instruction Code
Page: 197

Syntax

```
CMP     src, src2
```

Operation

```
src2 - src;
```

Function

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

Flag Change

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

Instruction Format

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

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

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

Description Example

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

DIV

Signed division
DIVide

Arithmetic/logic instruction
Instruction Code
Page: 199

Syntax

```
DIV     src, dest
```

Operation

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

Function

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

Flag Change

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

Instruction Format

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

Description Example

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

DIVU

Unsigned division
DIVide Unsigned

Arithmetic/logic instruction
Instruction Code
Page: 201

Syntax

```
DIVU    src, dest
```

Operation

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

Function

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

Flag Change

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

Instruction Format

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

Note: * For the RX Family assembler manufactured by Renesas Technology 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

Signed multiplication
Extended MULtiply, signed

Arithmetic/logic instruction
Instruction Code
Page: 202

Syntax

```
EMUL    src, dest
```

Operation

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

Function

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

Note: The accumulator (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 Technology 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

Unsigned multiplication
Extended MULtiply, Unsigned

Arithmetic/logic instruction
Instruction Code
Page: 203

Syntax

```
EMULU   src, dest
```

Operation

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

Function

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

Note: The accumulator (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	
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 Technology 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
```

Products of the RX100 Series and RX200 Series do not support the FADD instruction.

FADD

Floating-point addition
Floating-point ADD

Floating-point operation instruction
Instruction Code
Page: 204

Syntax

```
FADD    src, dest
```

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

Flag Change

Flag	Change	Condition
C	–	
Z	√	The flag is set if the result of the operation is +0 or –0; otherwise it is cleared.
S	√	The flag is set if the sign bit (bit 31) of the result of the operation is 1; otherwise it is cleared.
O	–	
CV	√	The flag is set if an invalid operation exception is generated; otherwise it is cleared.
CO	√	The flag is set if an overflow exception is generated; otherwise it is cleared.
CZ	√	The value of the flag is always 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

	Processing	Operand		Code Size
Syntax	Size	src	dest	(Byte)
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 Technology Corp., enter a scaled value (the actual value multiplied by 4) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 1020 (255 × 4) can be specified; with dsp:16, values from 0 to 262140 (65535 × 4) can be specified. The value divided by 4 will be stored in the instruction code.

Possible Exceptions

Unimplemented processing
Invalid operation
Overflow
Underflow
Inexact

Description Example

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

Supplementary Description

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

When DN = 0

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

When DN = 1

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

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

Products of the RX100 Series and RX200 Series do not support the FCMP instruction.

FCMP

Floating-point comparison
Floating-point CoMPare

Floating-point operation instruction
Instruction Code
Page: 205

Syntax

```
FCMP    src, src2
```

Operation

```
src2 - src;
```

Function

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

Flag Change

Flag	Change	Condition
C	–	
Z	√	The flag is set if src2 == src; otherwise it is cleared.
S	√	The flag is set if src2 < src; otherwise it is cleared.
O	√	The flag is set if an ordered classification based on the comparison result is impossible; otherwise it is cleared.
CV	√	The flag is set if an invalid operation exception is generated; otherwise it is cleared.
CO	√	The value of the flag is always 0.
CZ	√	The value of the flag is always 0.
CU	√	The value of the flag is always 0.
CX	√	The value of the flag is always 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		Code Size (Byte)
		src	src2	
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 Technology Corp., enter a scaled value (the actual value multiplied by 4) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 1020 (255 × 4) can be specified; with dsp:16, values from 0 to 262140 (65535 × 4) can be specified. The value divided by 4 will be stored in the instruction code.

Possible Exceptions

Unimplemented processing
Invalid operation

Description Example

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

Supplementary Description

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

When DN = 0

<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">src2</td><td>Normalized</td><td colspan="3">Comparison</td><td rowspan="3"><</td><td rowspan="4">></td><td rowspan="5"></td><td rowspan="6"></td><td rowspan="7"></td></tr>
<tr><td>+0</td><td rowspan="2"></td><td colspan="2" rowspan="2">=</td></tr>
<tr><td>–0</td></tr>
<tr><td>+∞</td><td colspan="3">></td><td>=</td></tr>
<tr><td>–∞</td><td colspan="4"><</td><td>=</td></tr>
<tr><td>Denormalized</td><td colspan="6">Unimplemented processing</td></tr>
<tr><td>QNaN</td><td colspan="7">Ordered classification impossible</td></tr>
<tr><td>SNaN</td><td colspan="8">Invalid operation (Ordered classification impossible)</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="7">src2</td><td>Normalized</td><td colspan="3">Comparison</td><td rowspan="3">&lt;</td><td rowspan="4">&gt;</td><td rowspan="6">Ordered classification impossible</td><td rowspan="7">Invalid operation (Ordered classification impossible)</td></tr>
<tr><td>+0,
+Denormalized</td><td rowspan="2"></td><td colspan="2" rowspan="2">=</td></tr>
<tr><td>–0,
–Denormalized</td></tr>
<tr><td>+∞</td><td colspan="3">&gt;</td><td>=</td></tr>
<tr><td>–∞</td><td colspan="4">&lt;</td><td>=</td></tr>
<tr><td>QNaN</td><td colspan="5"></td></tr>
<tr><td>SNaN</td><td colspan="6"></td></tr>
</table>

Products of the RX100 Series and RX200 Series do not support the FDIV instruction.

FDIV

Floating-point division
Floating-point DIVide

Floating-point operation instruction
Instruction Code
Page: 206

Syntax

```
FDIV    src, dest
```

Operation

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

Function

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

Flag Change

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

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

Instruction Format

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

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

Possible Exceptions

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

Description Example

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

Supplementary Description

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

When DN = 0

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

When DN = 1

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

Products of the RX100 Series and RX200 Series do not support the FMUL instruction.

FMUL

Floating-point multiplication
Floating-point MULtiply

Floating-point operation instruction
Instruction Code
Page: 207

Syntax

```
FMUL    src, dest
```

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

Note: The 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	√	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 always 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 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	–	
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, 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 Technology Corp., enter a scaled value (the actual value multiplied by 4) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 1020 (255 × 4) can be specified; with dsp:16, values from 0 to 262140 (65535 × 4) can be specified. The value divided by 4 will be stored in the instruction code.

Possible Exceptions

Unimplemented processing
Invalid operation
Overflow
Underflow
Inexact

Description Example

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

Supplementary Description

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

When DN = 0

		src							
		Normalized	+0	–0	+∞	–∞	Denormalized	QNaN	SNaN
dest	Normalized	Multiplication			∞				
	+0		+0	–0	Invalid operation				
	–0		–0	+0					
	+∞	∞	Invalid operation		+∞	–∞			
	–∞				–∞	+∞			
	Denormalized						Unimplemented processing		
	QNaN							QNaN	
	SNaN								Invalid operation

When DN = 1

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

Products of the RX100 Series and RX200 Series do not support the FSUB instruction.

FSUB

Floating-point subtraction
Floating-point SUBtract

Floating-point operation instruction
Instruction Code
Page: 208

Syntax

```
FSUB    src, dest
```

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

Flag Change

Flag	Change	Condition
C	–	
Z	√	The flag is set if the result of the operation is +0 or –0; otherwise it is cleared.
S	√	The flag is set if the sign bit (bit 31) of the result of the operation is 1; otherwise it is cleared.
O	–	
CV	√	The flag is set if an invalid operation exception is generated; otherwise it is cleared.
CO	√	The flag is set if an overflow exception is generated; otherwise it is cleared.
CZ	√	The value of the flag is always 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 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	–	
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, 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 Technology Corp., enter a scaled value (the actual value multiplied by 4) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 1020 (255 × 4) can be specified; with dsp:16, values from 0 to 262140 (65535 × 4) can be specified. The value divided by 4 will be stored in the instruction code.

Possible Exceptions

Unimplemented processing
Invalid operation
Overflow
Underflow
Inexact

Description Example

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

Supplementary Description

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

When DN = 0

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

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.

Products of the RX100 Series and RX200 Series do not support the FTOI instruction.

FTOI

Floating point to integer conversion
Float TO Integer

Floating-point operation instruction
Instruction Code
Page: 209

Syntax

```
FTOI    src, dest
```

Operation

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

Function

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

Flag Change

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

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

Instruction Format

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

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

Possible Exceptions

Unimplemented processing
Invalid operation
Inexact

Description Example

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

Supplementary Description

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

When DN = 0

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

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

When DN = 1

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

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

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

INT

Software interrupt
INTerrupt

System manipulation instruction
Instruction Code
Page: 209

Syntax

```
INT      src
```

Operation

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

Function

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

Flag Change

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

Instruction Format

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

Description Example

```
INT      #0
```

Products of the RX100 Series and RX200 Series do not support the ITOF instruction.

ITOF

Integer to floating-point conversion
Integer TO Floating-point

Floating-point operation instruction
Instruction Code
Page: 210

Syntax

```
ITOF    src, dest
```

Operation

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

Function

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

Flag Change

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

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

Instruction Format

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

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

Possible Exceptions

Inexact

Description Example

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

JMP

Unconditional jump
JuMP

Branch instruction
Instruction Code
Page: 211

Syntax

```
JMP     src
```

Operation

```
PC = src;
```

Function

- This instruction branches to the instruction specified by src.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

```
JMP     R1
```

JSR

Jump to a subroutine
Jump SubRoutine

Branch instruction
Instruction Code
Page: 211

Syntax

```
JSR     src
```

Operation

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

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

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

```
JSR     R1
```

MACHI

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

DSP instruction
Instruction Code
Page: 212

Syntax

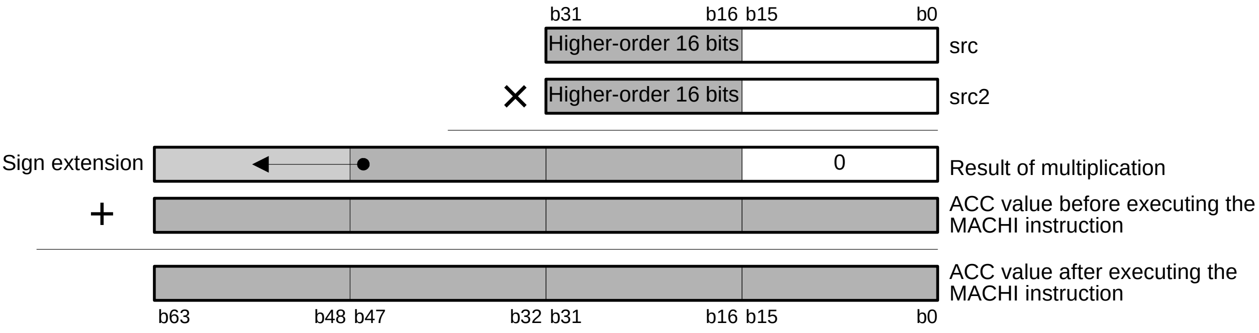
```
MACHI   src, src2
```

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



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

```
MACHI   R1, R2
```

MACLO

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

DSP instruction
Instruction Code
Page: 212

Syntax

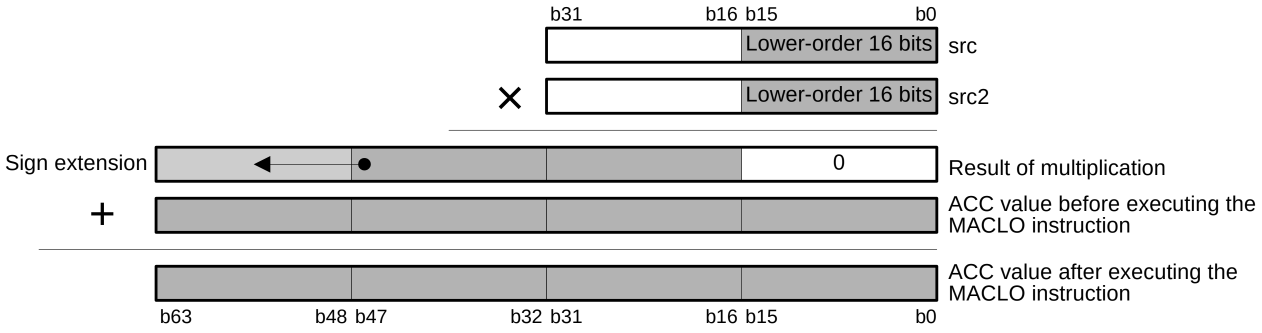
```
MACLO   src, src2
```

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



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

```
MACLO   R1, R2
```

MAX

Selecting the highest value
MAXimum value select

Arithmetic/logic instruction
Instruction Code
Page: 213

Syntax

```
MAX      src, dest
```

Operation

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

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Processing Size	Operand		Code Size (Byte)
		src	dest	
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 Technology Corp., enter a scaled value (the actual value multiplied by 2 when the size extension specifier is .W or .UW, or by 4 when the specifier is .L) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 510 (255 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 1020 (255 × 4) when the specifier is .L. With dsp:16, values from 0 to 131070 (65535 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 262140 (65535 × 4) when the specifier is .L. The value divided by 2 or 4 will be stored in the instruction code.

Description Example

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

MIN

Selecting the lowest value
MINimum value select

Arithmetic/logic instruction
Instruction Code
Page: 214

Syntax

```
MIN     src, dest
```

Operation

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

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Processing Size	Operand		Code Size (Byte)
		src	dest	
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 Technology Corp., enter a scaled value (the actual value multiplied by 2 when the size extension specifier is .W or .UW, or by 4 when the specifier is .L) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 510 (255 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 1020 (255 × 4) when the specifier is .L. With dsp:16, values from 0 to 131070 (65535 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 262140 (65535 × 4) when the specifier is .L. The value divided by 2 or 4 will be stored in the instruction code.

Description Example

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

MOV

Transferring data
MOVe

Data transfer instruction
Instruction Code
Page: 215

Syntax

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

Operation

```
dest = src;
```

Function

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

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

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

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

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

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

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

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

Description Example

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

MOVU

Transfer unsigned data
MOVe Unsigned data

Data transfer instruction
Instruction Code
Page: 220

Syntax

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

Operation

```
dest = src;
```

Function

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

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

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

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

Description Example

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

MUL

Multiplication
MULtiply

Arithmetic/logic instruction
Instruction Code
Page: 221

Syntax

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

Operation

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

Function

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

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

Note: The accumulator (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			Code Size (Byte)
		src	src2	dest	
(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 Technology Corp., enter a scaled value (the actual value multiplied by 2 when the size extension specifier is .W or .UW, or by 4 when the specifier is .L) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 510 (255 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 1020 (255 × 4) when the specifier is .L. With dsp:16, values from 0 to 131070 (65535 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 262140 (65535 × 4) when the specifier .L. The value divided by 2 or 4 will be stored in the instruction code.

Description Example

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

MULHI

Multiply the high-order word
MULtiply HIgh-order word

DSP instruction
Instruction Code
Page: 223

Syntax

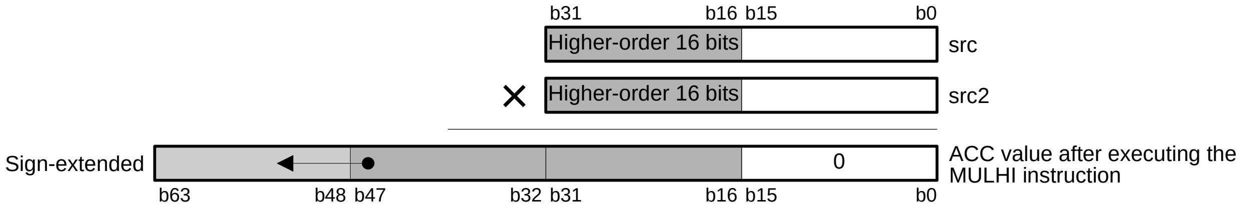
```
MULHI   src, src2
```

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 higher-order 16 bits of src by the higher-order 16 bits of src2, and stores the result in the accumulator (ACC). When the result is stored, the least significant bit of the result corresponds to bit 16 of ACC, and the section corresponding to bits 63 to 48 of ACC is sign-extended. Moreover, bits 15 to 0 of ACC are cleared to 0. The higher-order 16 bits of src and the higher-order 16 bits of src2 are treated as signed integers.



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

```
MULHI   R1, R2
```

MULLO

Multiply the low-order word
MULtiply LOw-order word

DSP instruction
Instruction Code
Page: 223

Syntax

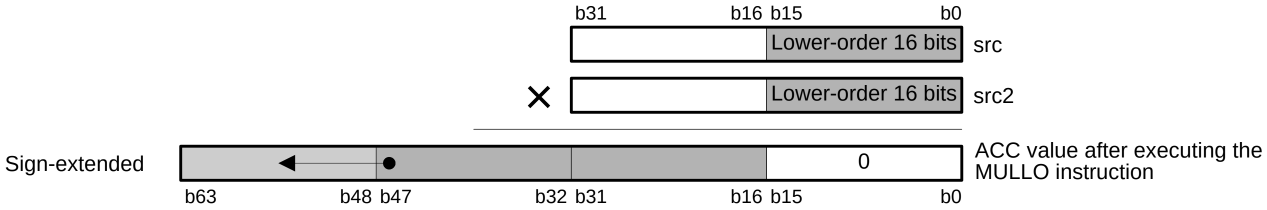
```
MULLO   src, src2
```

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-order 16 bits of src by the lower-order 16 bits of src2, and stores the result in the accumulator (ACC). When the result is stored, the least significant bit of the result corresponds to bit 16 of ACC, and the section corresponding to bits 63 to 48 of ACC is sign-extended. Moreover, bits 15 to 0 of ACC are cleared to 0. The lower-order 16 bits of src and the lower-order 16 bits of src2 are treated as signed integers.



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

```
MULLO   R1, R2
```

MVFACHI

Move the high-order longword from accumulator

MoVe From ACcumulator HIgh-order longword

DSP instruction

Instruction Code

Page: 224

Syntax

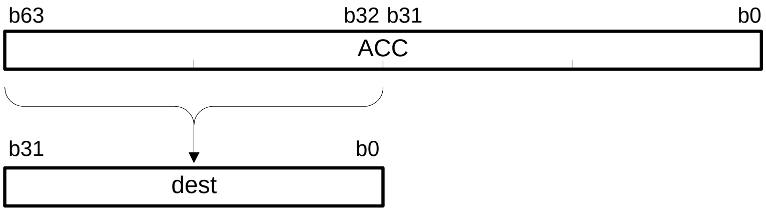
```
MVFACHI  dest
```

Operation

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

Function

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



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

```
MVFACHI  R1
```

MVFACMI

Move the middle-order longword from accumulator
MoVe From ACcumulator MIddle-order longword

DSP instruction
Instruction Code
Page: 224

Syntax

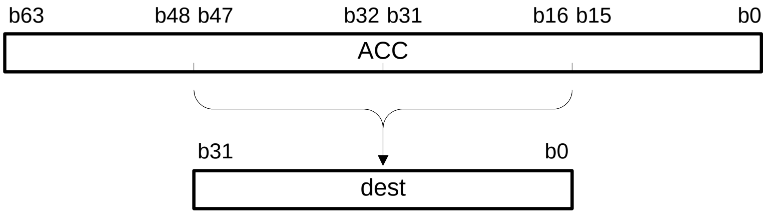
```
MVFACMI  dest
```

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	Code Size
Syntax	dest	(Byte)
MVFACMI dest	Rd	3

Description Example

```
MVFACMI  R1
```

MVFC

Transfer from a control register
MoVe From Control register

System manipulation instruction
Instruction Code
Page: 225

Syntax

```
MVFC    src, dest
```

Operation

```
dest = src;
```

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Processing Size	Operand		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
The FPSW is not selectable in products of the RX100 Series and RX200 Series.

Description Example

```
MVFC    USP, R1
```

MVTACHI

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

DSP instruction
Instruction Code
Page: 225

Syntax

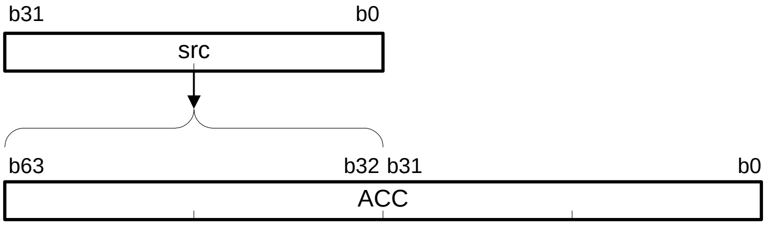
```
MVTACHI  src
```

Operation

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

Function

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



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

```
MVTACHI  R1
```

MVTACLO

Move the low-order longword to accumulator

MoVe To ACcumulator LOw-order longword

DSP instruction

Instruction Code

Page: 226

Syntax

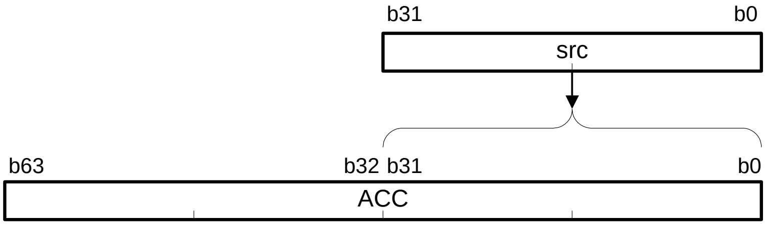
```
MVTACLO  src
```

Operation

```
ACC = (ACC & FFFFFFFF00000000h) | src;
```

Function

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



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

```
MVTACLO  R1
```

MVTC

Transfer to a control register
MoVe To Control register

System manipulation instruction
Instruction Code
Page: 226

Syntax

```
MVTC    src, dest
```

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

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

Note: * Selectable dest: Registers ISP, USP, INTB, PSW, BPC, BPSW, FINTV, and FPSW
Note that the PC cannot be specified as dest.
The FPSW is not selectable in products of the RX100 Series and RX200 Series.

Description Example

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

MVTIPL

Interrupt priority level setting
MoVe To Interrupt Priority Level

System manipulation instruction
Instruction Code
Page: 227

Syntax

```
MVTIPL  src
```

Operation

```
IPL = src;
```

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

```
MVTIPL  #2
```

Note: The MVTIPL instruction is not available in products of the RX610 Group. Use the MVTC instruction to write interrupt priority levels to the processor interrupt-priority level (IPL[2:0]) bits in the processor status word (PSW).

NEG

Two’s complementation
NEGate

Arithmetic/logic instruction
Instruction Code
Page: 228

Syntax

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

Operation

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

Function

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

Flag Change

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

Instruction Format

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

Description Example

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

NOP

No operation
No OPeration

Arithmetic/logic instruction
Instruction Code
Page: 228

Syntax

```
NOP
```

Operation

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

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Code Size (Byte)
NOP	1

Description Example

```
NOP
```

NOT

Logical complementation
NOT

Arithmetic/logic instruction
Instruction Code
Page: 229

Syntax

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

Operation

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

Function

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

Flag Change

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

Instruction Format

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

Description Example

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

OR

Logical OR
OR

Arithmetic/logic instruction
Instruction Code
Page: 230

Syntax

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

Operation

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

Function

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

Flag Change

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

Instruction Format

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

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

Description Example

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

POP

Restoring data from stack to register
POP data from the stack

Data transfer instruction
Instruction Code
Page: 231

Syntax

```
POP     dest
```

Operation

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

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

```
POP     R1
```

POPC

Restoring a control register
POP Control register

Data transfer instruction
Instruction Code
Page: 232

Syntax

```
POPC    dest
```

Operation

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

Function

- This instruction restores data from the stack and transfers it to the control register specified as dest.
- The stack pointer in use is specified by the U bit in the PSW.
- In user mode, writing to the ISP, INTB, 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

Syntax	Processing Size	Operand 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.
The FPSW is not selectable in products of the RX100 Series and RX200 Series.

Description Example

```
POPC    PSW
```

POPM

Restoring multiple registers from the stack
POP Multiple registers

Data transfer instruction
Instruction Code
Page: 232

Syntax

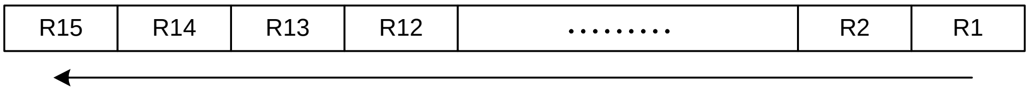
```
POPM    dest-dest2
```

Operation

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

Function

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



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

Saving data on the stack
PUSH data onto the stack

Data transfer instruction
Instruction Code
Page: 233

Syntax

```
PUSH.size  src
```

Operation

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

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

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

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

Description Example

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

PUSHC

Saving a control register
PUSH Control register

Data transfer instruction
Instruction Code
Page: 234

Syntax

```
PUSHC   src
```

Operation

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

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Note: * Selectable src: Registers PC, ISP, USP, INTB, PSW, BPC, BPSW, FINTV, and FPSW
The FPSW is not selectable in products of the RX100 Series and RX200 Series.

Description Example

```
PUSHC   PSW
```

PUSHM

Saving multiple registers
PUSH Multiple registers

Data transfer instruction
Instruction Code
Page: 234

Syntax

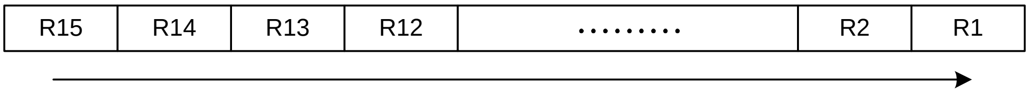
```
PUSHM   src-src2
```

Operation

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

Function

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



Saving is in sequence from R15.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

	Processing	Operand		Code Size
Syntax	Size	src	src2	(Byte)
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
Round ACcumulator Word

DSP instruction
Instruction Code
Page: 235

Syntax

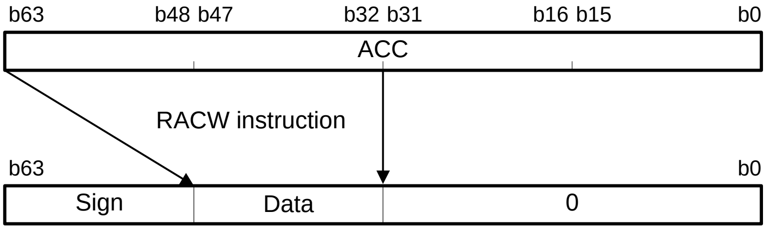
```
RACW    src
```

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;
```

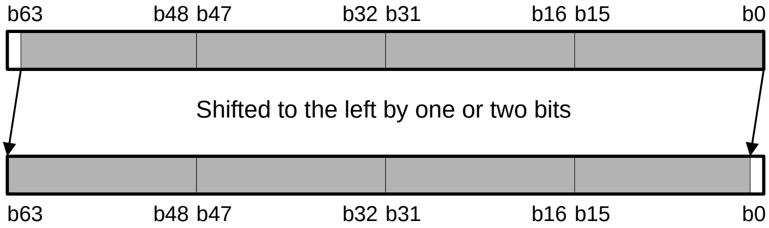
Function

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

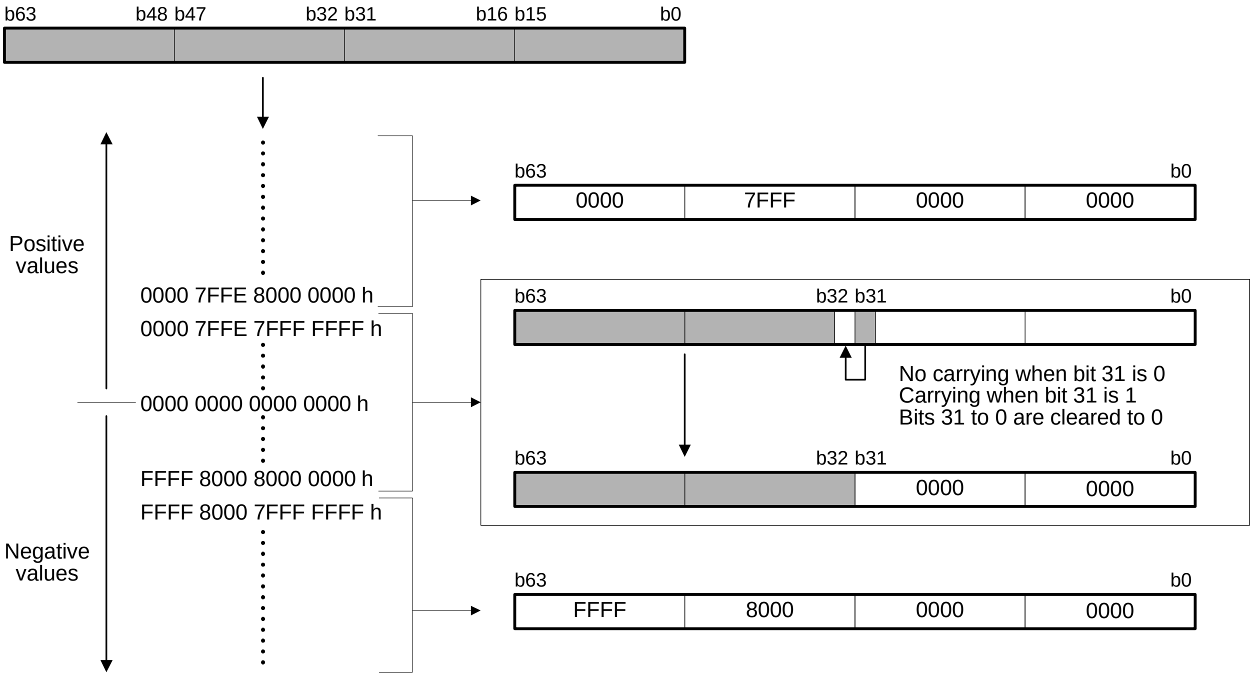


- The RACW instruction is executed according to the following procedures.

  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

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

Note: * For the RX Family assembler manufactured by Renesas Technology Corp., enter 1 or 2 as the immediate (IMM:1). As the instruction code, the value minus 1 will be stored.

Description Example

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

REVL

Endian conversion
REVerse Longword data

Data transfer instruction
Instruction Code
Page: 235

Syntax

```
REVL    src, dest
```

Operation

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

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

```
REVL    R1, R2
```

REVW

Endian conversion
REVerse Word data

Data transfer instruction
Instruction Code
Page: 236

Syntax

```
REVW    src, dest
```

Operation

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

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

```
REVW    R1, R2
```

RMPA

Multiply-and-accumulate operation
Repeated MultiPly and Accumulate

Arithmetic/logic instruction
Instruction Code
Page: 236

Syntax

```
RMPA.size
```

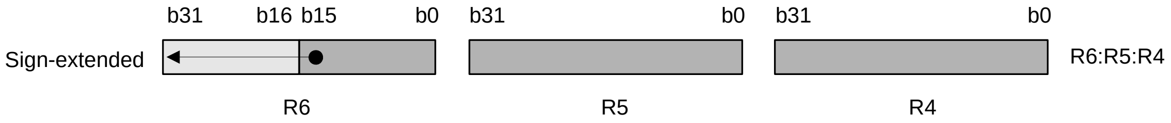
Operation

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

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

Function

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



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

Note: The accumulator (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

Rotation with carry to left
ROtate Left with Carry

Arithmetic/logic instruction
Instruction Code
Page: 237

Syntax

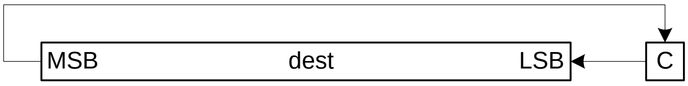
```
ROLC    dest
```

Operation

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

Function

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



Flag Change

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

Instruction Format

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

Description Example

```
ROLC    R1
```

RORC

Rotation with carry to right
ROtate Right with Carry

Arithmetic/logic instruction
Instruction Code
Page: 237

Syntax

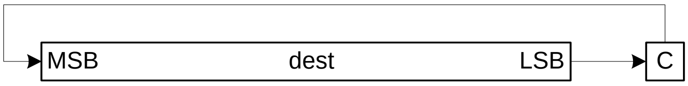
```
RORC    dest
```

Operation

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

Function

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



Flag Change

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

Instruction Format

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

Description Example

```
RORC    R1
```

ROTL

Rotation to left
ROTate Left

Arithmetic/logic instruction
Instruction Code
Page: 238

Syntax

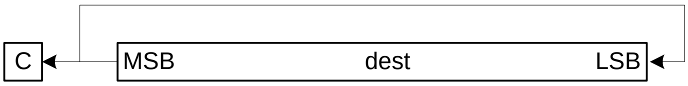
```
ROTL    src, dest
```

Operation

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

Function

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



Flag Change

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

Instruction Format

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

Description Example

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

ROTR

Rotation to right
ROTate Right

Arithmetic/logic instruction
Instruction Code
Page: 238

Syntax

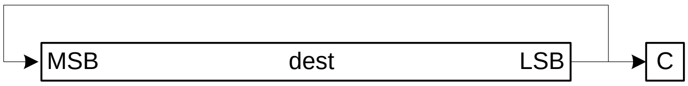
```
ROTR    src, dest
```

Operation

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

Function

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



Flag Change

Flag	Change	Condition
C	√	After the operation, this flag will have the same 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 src	Operand dest	Code Size (Byte)
ROTR src, dest	L	#IMM:5	Rd	3
	L	Rs	Rd	3

Description Example

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

Products of the RX100 Series and RX200 Series do not support the ROUND instruction.

ROUND

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

Floating-point operation instruction
Instruction Code
Page: 239

Syntax

```
ROUND   src, dest
```

Operation

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

Function

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

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

Flag Change

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

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

Instruction Format

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

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

Possible Exceptions

Unimplemented processing
Invalid operation
Inexact

Description Example

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

Supplementary Description

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

When DN = 0

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

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

When DN = 1

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

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

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

RTE

Return from the exception
ReTurn from Exception

System manipulation instruction
Instruction Code
Page: 239

Syntax

```
RTE
```

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 preserved when the exception was accepted.
- This instruction is a privileged instruction. Attempting to execute this instruction in user mode generates a privileged instruction exception.
- If returning is accompanied by a transition to user mode, the U bit in the PSW becomes 1.

Flag Change

Flag	Change	Condition
C	*	
Z	*	
S	*	
O	*	

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

Instruction Format

Syntax	Code Size (Byte)
RTE	2

Description Example

```
RTE
```

RTFI

Return from the fast interrupt
ReTurn from Fast Interrupt

System manipulation instruction
Instruction Code
Page: 240

Syntax

```
RTFI
```

Operation

```
PSW = BPSW;
PC = BPC;
```

Function

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

Flag Change

Flag	Change	Condition
C	*	
Z	*	
S	*	
O	*	

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

Instruction Format

Syntax	Code Size (Byte)
RTFI	2

Description Example

```
RTFI
```

RTS

Returning from a subroutine
ReTurn from Subroutine

Branch instruction
Instruction Code
Page: 240

Syntax

```
RTS
```

Operation

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

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Code Size (Byte)
RTS	1

Description Example

```
RTS
```

RTSD

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

Branch instruction
Instruction Code
Page: 240

Syntax

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

Operation

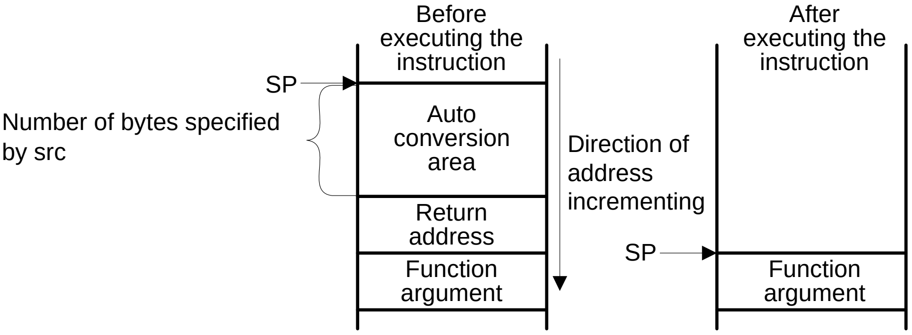
```
(1) SP = SP + src;
    PC = *SP;
    SP = SP + 4;

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

Function

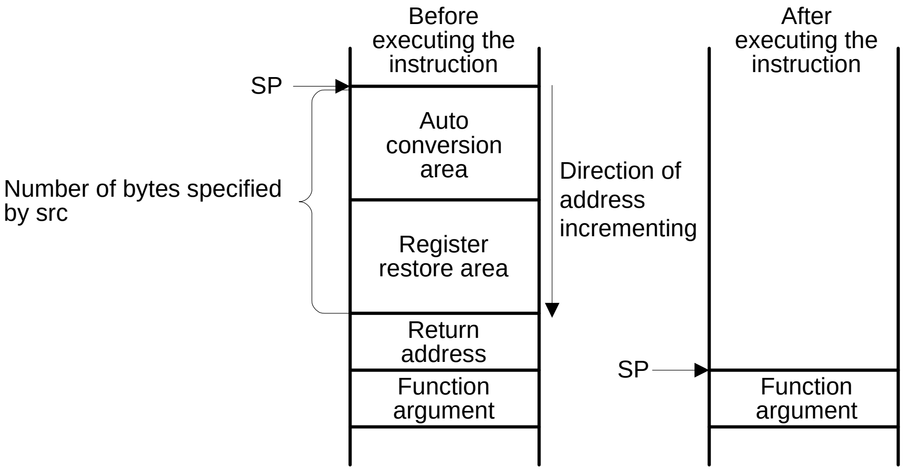
(1) This instruction returns the flow of execution from a subroutine after deallocating the stack frame for the subroutine.

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


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

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

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


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

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

R15	R14	R13	R12	··········	R2	R1

←

Restoration is in sequence from R1.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax		Operand			Code Size
		src	dest	dest2	(Byte)
(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 Technology Corp., enter a scaled value (the actual value multiplied by 4) as the immediate value. With UIMM:8, values from 0 to 1020 (255 × 4) can be specified. The value divided by 4 will be stored in the instruction code.

Description Example

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

SAT

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

Arithmetic/logic instruction
Instruction Code
Page: 241

Syntax

```
SAT     dest
```

Operation

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

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

```
SAT     R1
```

SATR

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

Arithmetic/logic instruction
Instruction Code
Page: 241

Syntax

```
SATR
```

Operation

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

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Code Size (Byte)
SATR	2

Description Example

```
SATR
```

SBB

Subtraction with borrow
SuBtract with Borrow

Arithmetic/logic instruction
Instruction Code
Page: 242

Syntax

```
SBB     src, dest
```

Operation

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

Function

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

Flag Change

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

Instruction Format

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

SCCnd

Condition setting
Store Condition Conditionally

SCCnd

Data transfer instruction
Instruction Code
Page: 243

Syntax

```
SCCnd.size   dest
```

Operation

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

Function

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

SC*Cnd*		**Condition**	**Expression**	**SC*Cnd***		**Condition**	**Expression**
SCGEU, 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

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

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

Description Example

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

SCMPU

String comparison
String CoMPare Until not equal

String manipulation instruction
Instruction Code
Page: 243

Syntax

```
SCMPU
```

Operation

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

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

Function

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

Flag Change

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

Instruction Format

Syntax	Processing Size	Code Size (Byte)
SCMPU	B	2

Description Example

```
SCMPU
```

SETPSW

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

System manipulation instruction
Instruction Code
Page: 244

Syntax

```
SETPSW  dest
```

Operation

```
dest = 1;
```

Function

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

Flag Change

Flag	Change	Condition
C	*	
Z	*	
S	*	
O	*	

Note: * The specified flag is set to 1.

Instruction Format

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

Description Example

```
SETPSW  C
SETPSW  Z
```

SHAR

Arithmetic shift to the right
SHift Arithmetic Right

Arithmetic/logic instruction
Instruction Code
Page: 245

Syntax

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

Operation

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

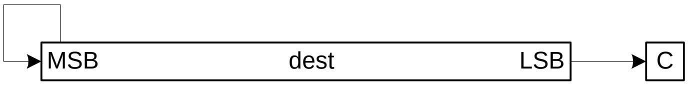
Function

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

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

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

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



Flag Change

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

Instruction Format

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

Description Example

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

SHLL

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

Arithmetic/logic instruction
Instruction Code
Page: 246

Syntax

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

Operation

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

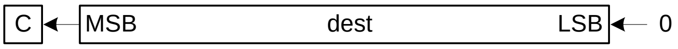
Function

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

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

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

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



Flag Change

Flag	Change	Condition
C	√	The flag is set if the shifted-out bit is 1; otherwise it is cleared. However, when src is 0, this flag is also cleared.
Z	√	The flag is set if dest is 0 after the operation; otherwise it is cleared.
S	√	The flag is set if the MSB of dest after the operation is 1; otherwise it is cleared.
O	√	This bit is cleared to 0 when the MSB of the result of the operation is equal to all bit values that have been shifted out (i.e. the shift operation has not changed the sign); otherwise it is set to 1. However, when scr is 0, this flag is also cleared.

Instruction Format

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

Description Example

```
SHLL    #3, R2
SHLL    R1, R2
SHLL    #3, R1, R2
```

SHLR

Logical shift to the right
SHift Logical Right

Arithmetic/logic instruction
Instruction Code
Page: 247

Syntax

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

Operation

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

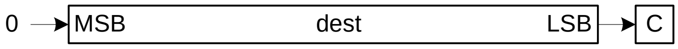
Function

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

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

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

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



Flag Change

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

Instruction Format

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

Description Example

```
SHLR    #3, R2
SHLR    R1, R2
SHLR    #3, R1, R2
```

SMOVB

Transferring a string backward
Strings MOVe Backward

String manipulation instruction
Instruction Code
Page: 248

Syntax

```
SMOVB
```

Operation

```
unsigned char *R1, *R2;
unsigned long R3;
while ( R3 != 0 ) {
  *R1-- = *R2--;
  R3 = R3 - 1;
}
```

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

Function

- This instruction transfers a string consisting of the number of bytes specified by R3 from the source address specified by R2 to the destination address specified by R1, with transfer proceeding in the direction of decreasing addresses.
- In execution of the instruction, data may be prefetched from the source address specified by R2, with R3 as the upper limit. For details of the data size to be prefetched, refer to the hardware manual of each product.
- The destination address specified by R1 should not be included in the range of data to be prefetched, which starts from the source address specified by R2.
- On completion of instruction execution, R1 and R2 indicate the next addresses in sequence from those for the last transfer.
- An interrupt request during execution of this instruction will be accepted, so processing of the instruction will be suspended. That is, execution of the instruction will continue on return from the interrupt processing routine. However, be sure to save the contents of the R1, R2, R3, and PSW when an interrupt is generated and restore them when execution is returned from the interrupt routine.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Processing Size	Code Size (Byte)
SMOVB	B	2

Description Example

```
SMOVB
```

SMOVF

Transferring a string forward
Strings MOVe Forward

String manipulation instruction
Instruction Code
Page: 248

Syntax

```
SMOVF
```

Operation

```
unsigned char *R1, *R2;
unsigned long R3;
while ( R3 != 0 ) {
  *R1++ = *R2++;
  R3 = R3 - 1;
}
```

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

Function

- This instruction transfers a string consisting of the number of bytes specified by R3 from the source address specified by R2 to the destination address specified by R1, with transfer proceeding in the direction of increasing addresses.
- In execution of the instruction, data may be prefetched from the source address specified by R2, with R3 as the upper limit. For details of the data size to be prefetched, refer to the hardware manual of each product.
- The destination address specified by R1 should not be included in the range of data to be prefetched, which starts from the source address specified by R2.
- On completion of instruction execution, R1 and R2 indicate the next addresses in sequence from those for the last transfer.
- An interrupt request during execution of this instruction will be accepted, so processing of the instruction will be suspended. That is, execution of the instruction will continue on return from the interrupt processing routine. However, be sure to save the contents of the R1, R2, R3, and PSW when an interrupt is generated and restore them when execution is returned from the interrupt routine.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Processing Size	Code Size (Byte)
SMOVF	B	2

Description Example

```
SMOVF
```

SMOVU

Transferring a string
Strings MOVe while Unequal to zero

String manipulation instruction
Instruction Code
Page: 248

Syntax

```
SMOVU
```

Operation

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

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

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Processing Size	Code Size (Byte)
SMOVU	B	2

Description Example

```
SMOVU
```

SSTR

Storing a string
String SToRe

String manipulation instruction
Instruction Code
Page: 249

Syntax

```
SSTR.size
```

Operation

```
unsigned { char | short | long } *R1, R2;
unsigned long R3;
while ( R3 != 0 ) {
  *R1++ = R2;
  R3 = R3 - 1;
}
```

Notes:
1. If this instruction is executed with R3 set to 0, it is ignored and has no effect on registers and flags.
2. R1++: Incrementation is by the value corresponding to the size specifier (.size), i.e. by 1 for .B, 2 for .W, and 4 for .L.
3. R2: How much of the value in R2 is stored depends on the size specifier (.size): the byte from the LSB end of R2 is stored for .B, the word from the LSB end of R2 is stored for .W, and the longword in R2 is stored for .L.

Function

- This instruction stores the contents of R2 successively proceeding in the direction of increasing addresses specified by R1 up to the number specified by R3.
- On completion of instruction execution, R1 indicates the next address in sequence from that for the last transfer.
- An interrupt request during execution of this instruction will be accepted, so processing of the instruction will be suspended. That is, execution of the instruction will continue on return from the interrupt processing routine. However, be sure to save the contents of the R1, R2, R3, and PSW when an interrupt is generated and restore them when execution is returned from the interrupt routine.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

```
SSTR.W
```

STNZ

Transfer with condition
STore on Not Zero

Data transfer instruction
Instruction Code
Page: 249

Syntax

```
STNZ    src, dest
```

Operation

```
if ( Z == 0 )
   dest = src;
```

Function

- This instruction moves src to dest when the Z flag is 0. dest does not change when the Z flag is 1.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Transfer with condition
STore on Zero

Data transfer instruction
Instruction Code
Page: 250

Syntax

```
STZ     src, dest
```

Operation

```
if ( Z == 1 )
   dest = src;
```

Function

- This instruction moves src to dest when the Z flag is 1. dest does not change when the Z flag is 0.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Subtraction without borrow
SUBtract

Arithmetic/logic instruction
Instruction Code
Page: 251

Syntax

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

Operation

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

Function

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

Flag Change

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

Instruction Format

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

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

Description Example

```
SUB     #15, R2
SUB     R1, R2
SUB     [R1], R2
SUB     1[R1].B, R2
SUB     R1, R2, R3
```

SUNTIL

Searching for a string
Search UNTIL equal string

String manipulation instruction
Instruction Code
Page: 252

Syntax

```
SUNTIL.size
```

Operation

```
unsigned { char | short | long } *R1;
unsigned long R2, R3, tmp;
while ( R3 != 0 ) {
  tmp = ( unsigned long ) *R1++;
  R3--;
  if ( tmp == R2 ) {
      break;
  }
}
```

Notes: 1. If this instruction is executed with R3 set to 0, it is ignored and has no effect on registers and flags.
2. R1++: Incrementation is by the value corresponding to the size specifier (.size), i.e. by 1 for .B, 2 for .W, and 4 for .L.

Function

- This instruction searches a string for comparison from the first address specified by R1 for a match with the value specified in R2, with the search proceeding in the direction of increasing addresses and the number specified by R3 as the upper limit on the number of comparisons. When the size specifier (.size) is .B or .W, the byte or word data on the memory is compared with the value in R2 after being zero-extended to form a longword of data.
- In execution of the instruction, data may be prefetched from the destination address for comparison specified by R1, with R3 as the upper limit. For details of the data size to be prefetched, refer to the hardware manual of each product.
- Flags change according to the results of the operation "*R1 – R2".
- The value in R1 upon completion of instruction execution indicates the next address where the data matched. Unless there was a match within the limit, the value in R1 is the next address in sequence from that for the last comparison.
- The value in R3 on completion of instruction execution is the initial value minus the number of comparisons.
- An interrupt request during execution of this instruction will be accepted, so processing of the instruction will be suspended. That is, execution of the instruction will continue on return from the interrupt processing routine. However, be sure to save the contents of the R1, R2, R3, and PSW when an interrupt is generated and restore them when execution is returned from the interrupt routine.

Flag Change

Flag	Change	Condition
C	√	The flag is set if a comparison operation as unsigned integers results in any value equal to or greater than 0; otherwise it is cleared.
Z	√	The flag is set if matched data is found; otherwise it is cleared.
S	–	
O	–	

Instruction Format

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

Description Example

```
SUNTIL.W
```

SWHILE

Searching for a string
Search WHILE unequal string

String manipulation instruction
Instruction Code
Page: 252

Syntax

```
SWHILE.size
```

Operation

```
unsigned { char | short | long } *R1;
unsigned long R2, R3, tmp;
while ( R3 != 0 ) {
   tmp = ( unsigned long ) *R1++;
   R3--;
   if ( tmp != R2 ) {
      break;
   }
}
```

Notes: 1. If this instruction is executed with R3 set to 0, it is ignored and has no effect on registers and flags.
2. R1++: Incrementation is by the value corresponding to the size specifier (.size), i.e. by 1 for .B, 2 for .W, and 4 for .L.

Function

- This instruction searches a string for comparison from the first address specified by R1 for an unmatch with the value specified in R2, with the search proceeding in the direction of increasing addresses and the number specified by R3 as the upper limit on the number of comparisons. When the size specifier (.size) is. B or .W, the byte or word data on the memory is compared with the value in R2 after being zero-extended to form a longword of data.
- In execution of the instruction, data may be prefetched from the destination address for comparison specified by R1, with R3 as the upper limit. For details of the data size to be prefetched, refer to the hardware manual of each product.
- Flags change according to the results of the operation "*R1 – R2".
- The value in R1 upon completion of instruction execution indicates the next addresses where the data did not match. If all the data contents match, the value in R1 is the next address in sequence from that for the last comparison.
- The value in R3 on completion of instruction execution is the initial value minus the number of comparisons.
- An interrupt request during execution of this instruction will be accepted, so processing of the instruction will be suspended. That is, execution of the instruction will continue on return from the interrupt processing routine. However, be sure to save the contents of the R1, R2, R3, and PSW when an interrupt is generated and restore them when execution is returned from the interrupt routine.

Flag Change

Flag	Change	Condition
C	√	The flag is set if a comparison operation as unsigned integers results in any value equal to or greater than 0; otherwise it is cleared.
Z	√	The flag is set if all the data contents match; otherwise it is cleared.
S	–	
O	–	

Instruction Format

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

Description Example

```
SWHILE.W
```

TST

Logical test
TeST logical

Arithmetic/logic instruction
Instruction Code
Page: 253

Syntax

```
TST     src, src2
```

Operation

```
src2 & src;
```

Function

- This instruction changes the flag states in the PSW according to the result of logical AND of src2 and src.

Flag Change

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

Instruction Format

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 Technology 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

Waiting
WAIT

System manipulation instruction
Instruction Code
Page: 254

Syntax

```
WAIT
```

Operation

Function

- This instruction stops program execution. Program execution is then restarted by acceptance of a non-maskable interrupt, interrupt, or generation of a reset.
- This instruction is a privileged instruction. Attempting to execute this instruction in user mode generates a privileged instruction exception.
- The I bit in the PSW becomes 1.
- The address of the PC saved at the generation of an interrupt is the one next to the WAIT instruction.

Note: For the power-down state when the execution of the program is stopped, refer to the hardware manual of each product.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Code Size (Byte)
WAIT	2

Description Example

```
WAIT
```

XCHG

Exchanging values
eXCHanGe

Data transfer instruction
Instruction Code
Page: 254

Syntax

```
XCHG    src, dest
```

Operation

```
tmp = src;
src = dest;
dest = tmp;
```

Function

- This instruction exchanges the contents of src and dest as listed in the following table.

src	dest	Function
Register	Register	Exchanges the data in the source register (src) and the destination register (dest).
Memory location	Register	Exchanges the data at the memory location and the register. When the size extension specifier (.size) is .B or .UB, the byte of data in the LSB of the register is exchanged with the data at the memory location. When the size extension specifier (.size) is .W or .UW, the word of data in the LSB of the register is exchanged with the data at the memory location. When the size extension specifier is other than .L, the data at the memory location is transferred to the register after being extended with the specified type of extension to form a longword of data.

- This instruction may be used for the exclusive control. For details, refer to the hardware manual of each product.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

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

Description Example

```
XCHG    R1, R2
XCHG    [R1].W, R2
```

XOR

Logical exclusive or
eXclusive OR logical

Arithmetic/logic instruction
Instruction Code
Page: 255

Syntax

```
XOR     src, dest
```

Operation

```
dest = dest ^ src;
```

Function

- This instruction exclusive-ORs dest and src and places the result in dest.

Flag Change

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

Instruction Format

	Processing	Operand		Code Size
Syntax	Size	src	dest	(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 Technology Corp., enter a scaled value (the actual value multiplied by 2 when the size extension specifier is .W or .UW, or by 4 when the specifier is .L) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 510 (255 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 1020 (255 × 4) when the specifier is .L. With dsp:16, values from 0 to 131070 (65535 × 2) can be specified when the size extension specifier is .W or .UW, or values from 0 to 262140 (65535 × 4) when the specifier is .L. The value divided by 2 or 4 will be stored in the instruction code.

Description Example

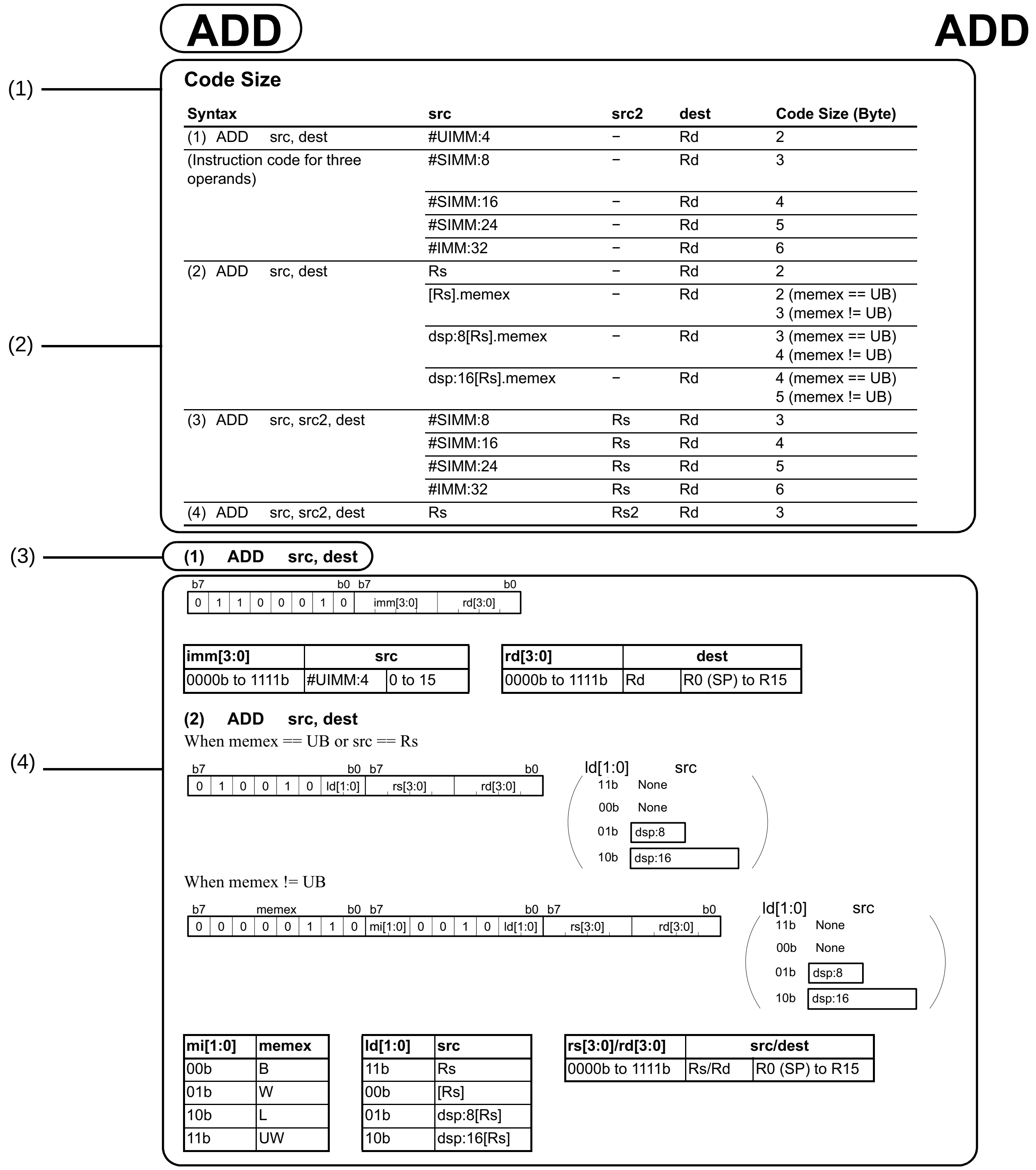
```
XOR     #8, R1
XOR     R1, R2
XOR     [R1], R2
XOR     16[R1].L, R2
```

Section 4 Instruction Code

4.1 Guide to This Section

This section describes instruction codes by showing the respective opcodes.

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



ADD ADD

Code Size

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

(1) ADD src, dest

b7							b0	b7	b0
0	1	1	0	0	0	1	0	imm[3:0]	rd[3:0]

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

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

(2) ADD src, dest

When memex == UB or src == Rs

b7						b0	b7	b0
0	1	0	0	1	0	ld[1:0]	rs[3:0]	rd[3:0]

ld[1:0]	src
11b	None
00b	None
01b	dsp:8
10b	dsp:16

When memex != UB

b7			memex				b0	b7					b0	b7	b0
0	0	0	0	0	1	1	0	mi[1:0]	0	0	1	0	ld[1:0]	rs[3:0]	rd[3:0]

ld[1:0]	src
11b	None
00b	None
01b	dsp:8
10b	dsp:16

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

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

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

(1) Mnemonic

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

(2) List of Code Size

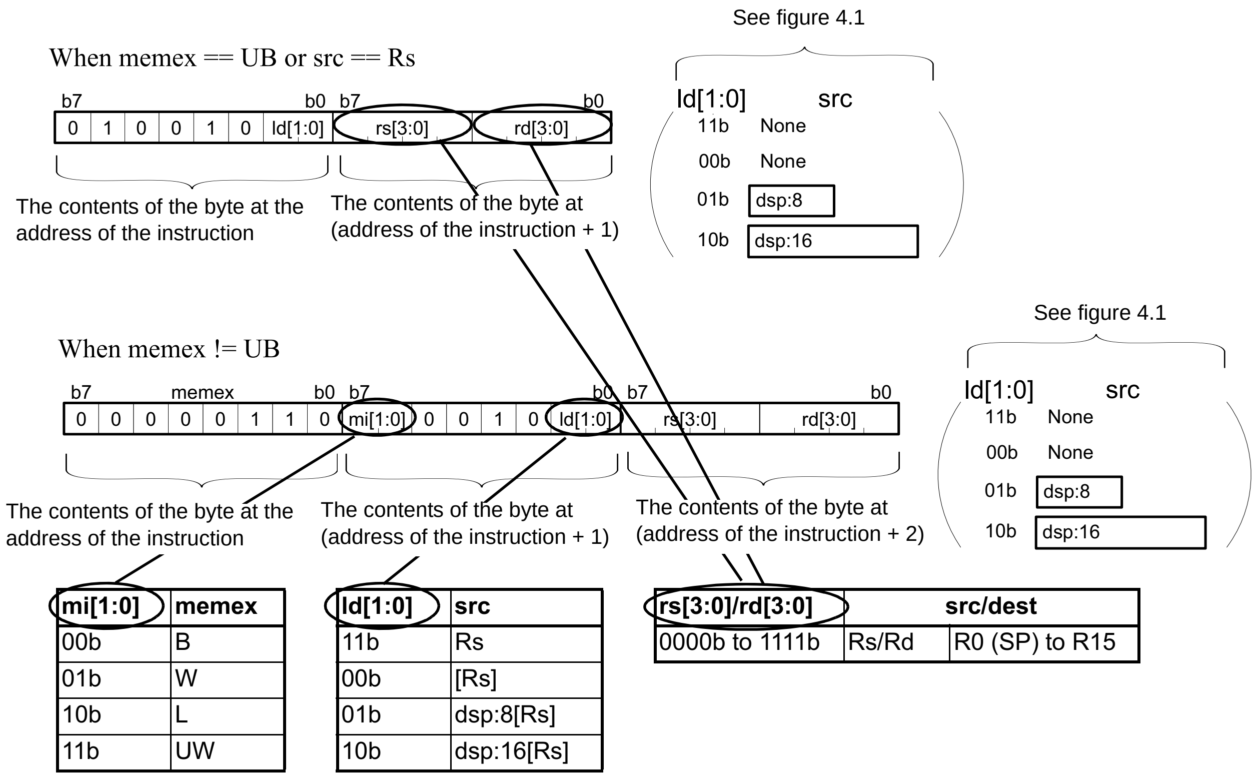
Indicates the number of bytes the instruction requires. An individual RX CPU instruction takes up from one to eight bytes.

(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.


See figure 4.1
When memex == UB or src == Rs
b7
b0
b7
b0
0
1
0
0
1
0
ld[1:0]
rs[3:0]
rd[3:0]
The contents of the byte at the address of the instruction
The contents of the byte at (address of the instruction + 1)
ld[1:0]
src
11b
None
00b
None
01b
dsp:8
10b
dsp:16
See figure 4.1
When memex != UB
b7
memex
b0
b7
b0
b7
b0
0
0
0
0
0
1
1
0
mi[1:0]
0
0
1
0
ld[1:0]
rs[3:0]
rd[3:0]
The contents of the byte at the address of the instruction
The contents of the byte at (address of the instruction + 1)
The contents of the byte at (address of the instruction + 2)
ld[1:0]
src
11b
None
00b
None
01b
dsp:8
10b
dsp:16

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

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

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

The contents of the operand, that is the byte at (address of the instruction +2) or (following address of the instruction +3) in the previous page, are arranged as shown in figure 4.1.

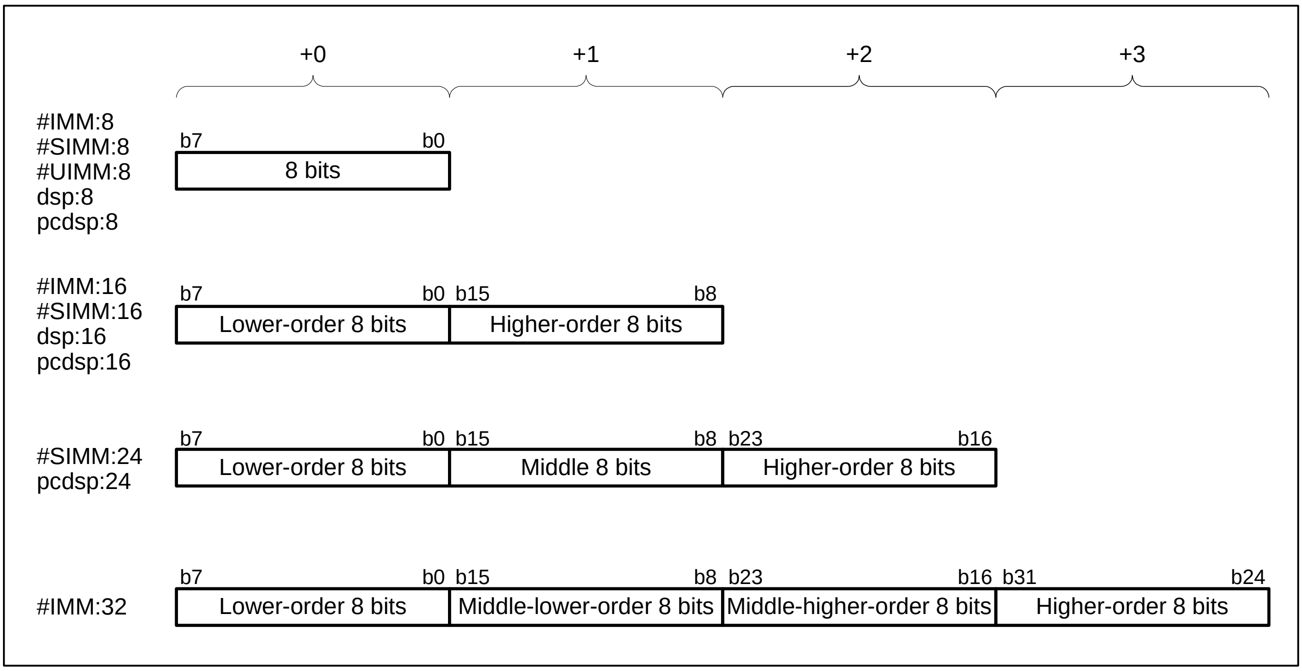


Figure 4.1 Immediate (IMM) and Displacement (dsp) Values

The abbreviations such as for rs, rd, ld, and mi represent the following.

rs:	Source register
rs2:	Second source register
rd:	Destination register
rd2:	Second destination register
ri:	Index register
rb:	Base register
li:	Length of immediate
ld:	Length of displacement
lds:	Length of source displacement
ldd:	Length of destination displacement
mi:	Memory extension size infix
imm:	Immediate
dsp:	Displacement
cd:	Condition code
cr:	Control register
cb:	Control bit
sz:	Size specifier
ad:	Addressing

4.2 Instruction Code Described in Detail

The following pages give details of the instruction codes for the RX CPU.

ABS ABS

Code Size

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

(1) ABS dest

b7							b0	b7				b0
0	1	1	1	1	1	1	0	0	0	1	0	rd[3:0]

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

(2) ABS src, dest

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	0	0	0	0	0	1	1	1	1	rs[3:0]	rd[3:0]

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

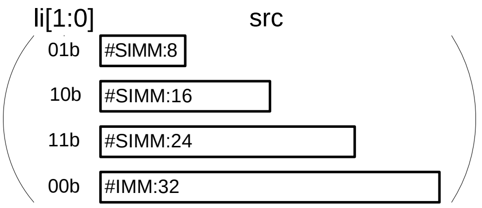
ADC ADC

Code Size

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

(1) ADC src, dest

b7							b0	b7						b0	b7					b0
1	1	1	1	1	1	0	1	0	1	1	1	li[1:0]	0	0	0	0	1	0	rd[3:0]	



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

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

(2) ADC src, dest

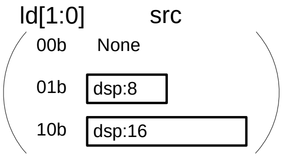
b7							b0	b7						b0	b7	b0
1	1	1	1	1	1	0	0	0	0	0	0	1	0	ld[1:0]	rs[3:0]	rd[3:0]

ld[1:0]	src
11b	Rs

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

(3) ADC src, dest

b7			memex				b0	b7				b0	b7							b0	b7	b0
0	0	0	0	0	1	1	0	mi[1:0]	1	0	0	0	ld[1:0]	0	0	0	0	0	0	1	0	rs[3:0]	rd[3:0]



mi[1:0]	memex
10b	L

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

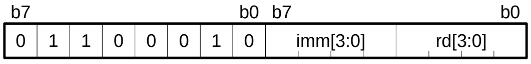
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

ADD ADD

Code Size

Syntax	src	src2	dest	Code Size (Byte)
(1) ADD src, dest	#UIMM:4	−	Rd	2
(Instruction code for three operands)	#SIMM:8	−	Rd	3
	#SIMM:16	−	Rd	4
	#SIMM:24	−	Rd	5
	#IMM:32	−	Rd	6
(2) ADD src, dest	Rs	−	Rd	2
	[Rs].memex	−	Rd	2 (memex == UB) 3 (memex != UB)
	dsp:8[Rs].memex	−	Rd	3 (memex == UB) 4 (memex != UB)
	dsp:16[Rs].memex	−	Rd	4 (memex == UB) 5 (memex != UB)
(3) ADD src, src2, dest	#SIMM:8	Rs	Rd	3
	#SIMM:16	Rs	Rd	4
	#SIMM:24	Rs	Rd	5
	#IMM:32	Rs	Rd	6
(4) ADD src, src2, dest	Rs	Rs2	Rd	3

(1) ADD src, dest

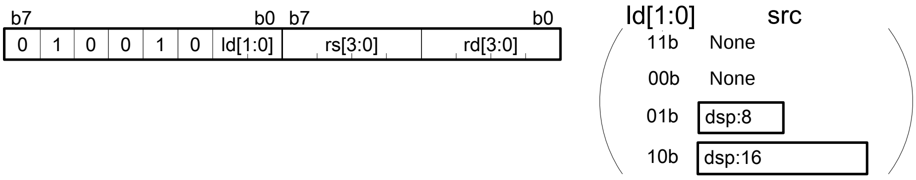


imm[3:0]	src	
0000b to 1111b	#UIMM:4	0 to 15

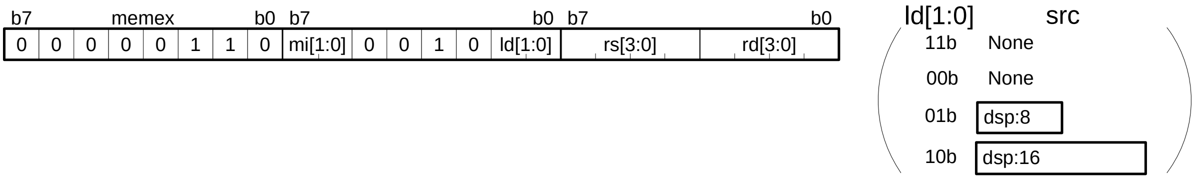
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) ADD src, dest

When memex == UB or src == Rs



When memex != UB



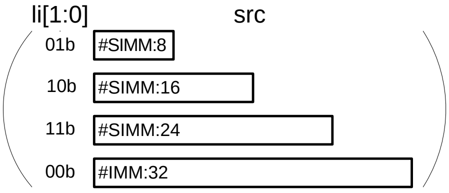
mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

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) ADD src, src2, dest

b7						b0	b7	b0
0	1	1	1	0	0	li[1:0]	rs2[3:0]	rd[3:0]



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

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

(4) ADD src, src2, dest

b7							b0	b7				b0	b7	b0
1	1	1	1	1	1	1	1	0	0	1	0	rd[3:0]	rs[3:0]	rs2[3:0]

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

AND AND

Code Size

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

(1) AND src, dest

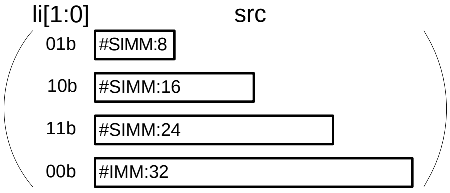
b7							b0	b7	b0
0	1	1	0	0	1	0	0	imm[3:0]	rd[3:0]

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

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

(2) AND src, dest

b7							b0	b7				b0
0	1	1	1	0	1	li[1:0]		0	0	1	0	rd[3:0]

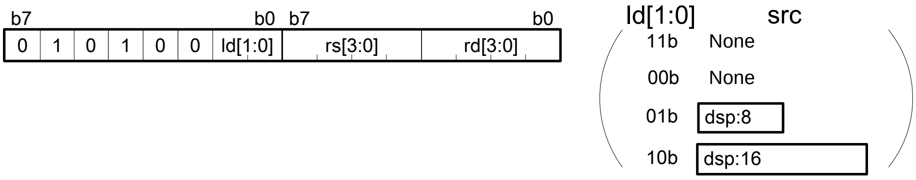


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

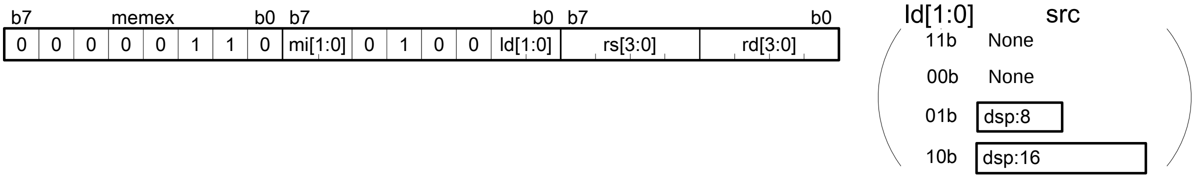
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(3) AND src, dest

When memex == UB or src == Rs



When memex != UB

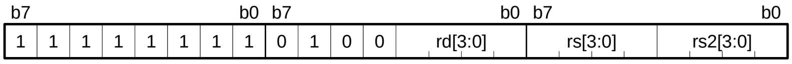


mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

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

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

(4) AND src, src2, dest



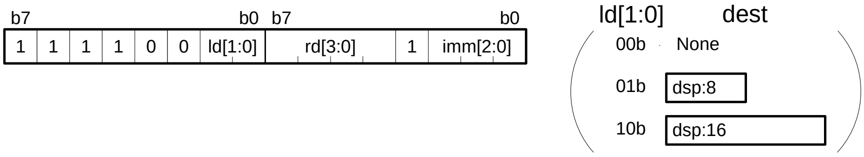
rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b to 1111b	Rs/Rs2/Rd	R0 (SP) to R15

BCLR BCLR

Code Size

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

(1) BCLR src, dest

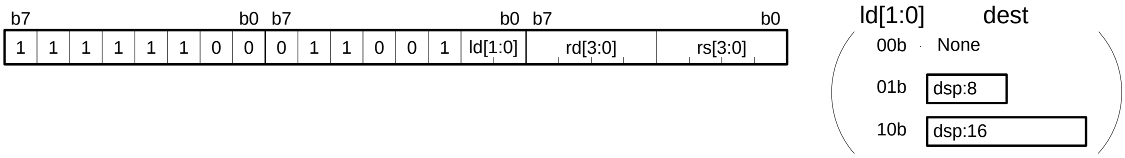


ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

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

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

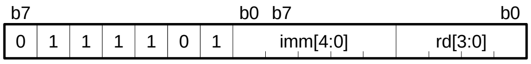
(2) BCLR src, dest



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

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

(3) BCLR src, dest



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

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

(4) BCLR src, dest

b7							b0	b7						b0	b7	b0
1	1	1	1	1	1	0	0	0	1	1	0	0	1	ld[1:0]	rd[3:0]	rs[3:0]

ld[1:0]	dest
11b	Rd

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

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

b7					b0
0	0	0	1	cd	dsp[2:0]*

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

b7				b0
0	0	1	0	cd[3:0]

src
pcdsp:8*

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

b7							b0
0	0	1	1	1	0	1	cd

src
pcdsp:16*

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

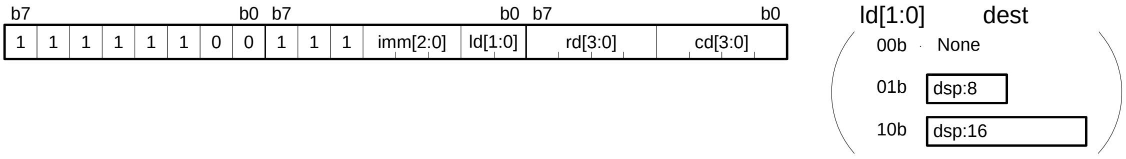
cd	**B*Cnd***
0b	BEQ, BZ
1b	BNE, BNZ

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



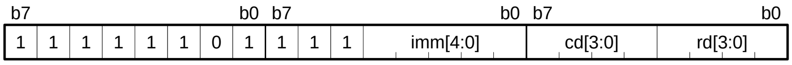
imm[2:0]	src	
000b to 111b	#IMM:3	0 to 7

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

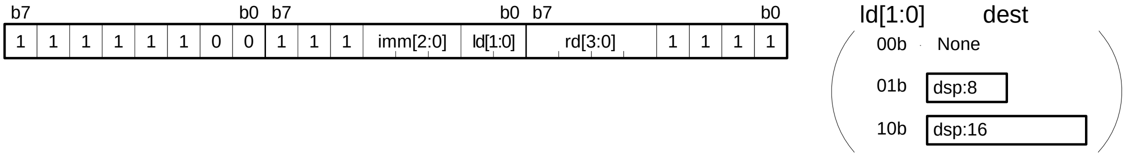
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

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



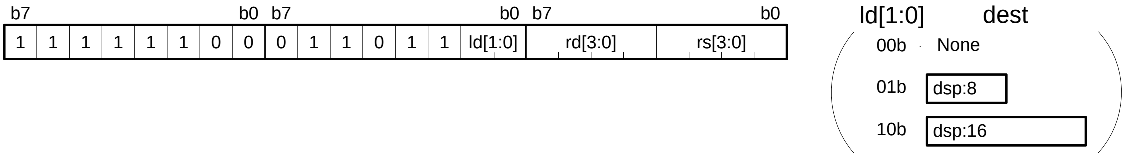
b7							b0	b7							b0	b7							b0
1	1	1	1	1	1	0	0	1	1	1	imm[2:0]			ld[1:0]		rd[3:0]				1	1	1	1

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



b7							b0	b7							b0	b7							b0
1	1	1	1	1	1	0	0	0	1	1	0	1	1	ld[1:0]		rd[3:0]				rs[3:0]			

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

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

(3) BNOT src, dest

b7							b0	b7							b0	b7							b0
1	1	1	1	1	1	0	1	1	1	1	imm[4:0]					1	1	1	1	rd[3:0]			

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

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

(4) BNOT src, dest

b7							b0	b7							b0	b7							b0
1	1	1	1	1	1	0	0	0	1	1	0	1	1	ld[1:0]		rd[3:0]				rs[3:0]			

ld[1:0]	dest
11b	Rd

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

BRA BRA

Code Size

Syntax	src	Code Size (Byte)
(1) BRA.S src	pcdsp:3	1
(2) BRA.B src	pcdsp:8	2
(3) BRA.W src	pcdsp:16	3
(4) BRA.A src	pcdsp:24	4
(5) BRA.L src	Rs	2

(1) BRA.S src

b7							b0
0	0	0	0	1	dsp[2:0]*		

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

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

(2) BRA.B src

b7							b0	src
0	0	1	0	1	1	1	0	pcdsp:8*

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

(3) BRA.W src

b7							b0	src
0	0	1	1	1	0	0	0	pcdsp:16*

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

(4) BRA.A src

b7							b0	src
0	0	0	0	0	1	0	0	pcdsp:24*

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

(5) BRA.L src

b7							b0	b7							b0
0	1	1	1	1	1	1	1	0	1	0	0	rs[3:0]			

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

BRK BRK

Code Size

Syntax	Code Size (Byte)
(1) BRK	1

(1) BRK

b7							b0
0	0	0	0	0	0	0	0

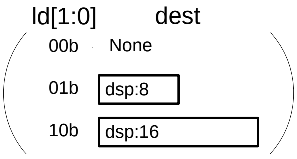
BSET BSET

Code Size

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

(1) BSET src, dest

b7							b0	b7							b0
1	1	1	1	0	0	ld[1:0]		rd[3:0]				0	imm[2:0]		



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

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

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

(2) BSET src, dest

b7							b0	b7							b0	b7							b0
1	1	1	1	1	1	0	0	0	1	1	0	0	0	ld[1:0]		rd[3:0]				rs[3:0]			

ld[1:0]	dest
00b	None
01b	dsp:8
10b	dsp:16

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

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

(3) BSET src, dest

b7							b0	b7							b0
0	1	1	1	1	0	0	imm[4:0]					rd[3:0]			

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

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

(4) BSET src, dest

b7							b0	b7							b0	b7							b0
1	1	1	1	1	1	0	0	0	1	1	0	0	0	ld[1:0]		rd[3:0]				rs[3:0]			

ld[1:0]	dest
11b	Rd

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

BSR BSR

Code Size

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

(1) BSR.W src

b7							b0	src
0	0	1	1	1	0	0	1	pcdsp:16*

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

(2) BSR.A src

b7							b0	src
0	0	0	0	0	1	0	1	pcdsp:24*

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

(3) BSR.L src

b7							b0	b7							b0
0	1	1	1	1	1	1	1	0	1	0	1	rs[3:0]			

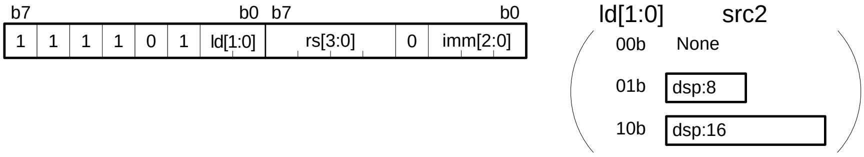
rs[3:0]	src	
0000b to 1111b	Rs	R0 (SP) to R15

BTST BTST

Code Size

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

(1) BTST src, src2

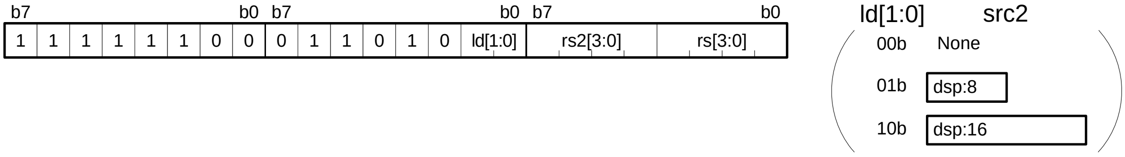


ld[1:0]	src2
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

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

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

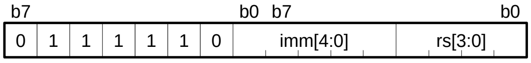
(2) BTST src, src2



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

b7							b0	b7						b0	b7	b0
1	1	1	1	1	1	0	0	0	1	1	0	1	0	ld[1:0]	rs2[3:0]	rs[3:0]

ld[1:0]	src2
11b	Rs2

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

CLRPSW CLRPSW

Code Size

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

(1) CLRPSW dest

b7							b0	b7				b0
0	1	1	1	1	1	1	1	1	0	1	1	cb[3:0]

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

CMP CMP

Code Size

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

(1) CMP src, src2

b7							b0	b7		b0
0	1	1	0	0	0	0	1	imm[3:0]		rs2[3:0]

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

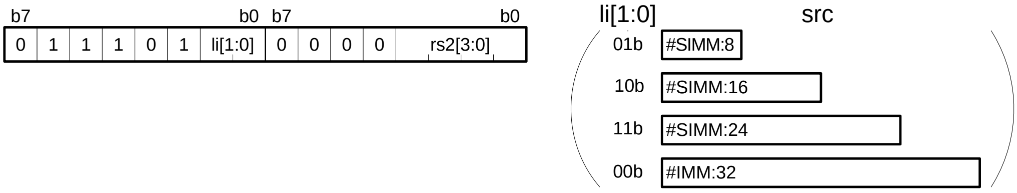
rs2[3:0]	src2	
0000b to 1111b	Rs	R0 (SP) to R15

(2) CMP src, src2

b7							b0	b7				b0	src
0	1	1	1	0	1	0	1	0	1	0	1	rs2[3:0]	#UIMM:8

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

(3) CMP src, src2

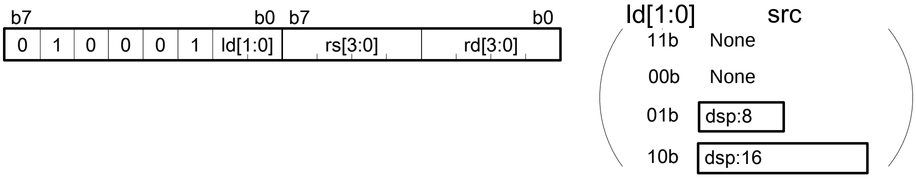


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

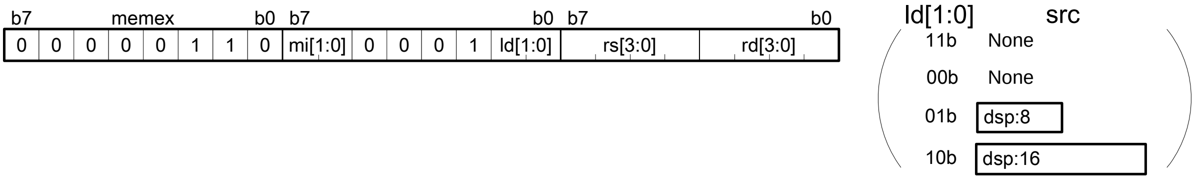
rs2[3:0]	src2	
0000b to 1111b	Rs	R0 (SP) to R15

(4) CMP src, src2

When memex == UB or src == Rs



When memex != UB



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

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

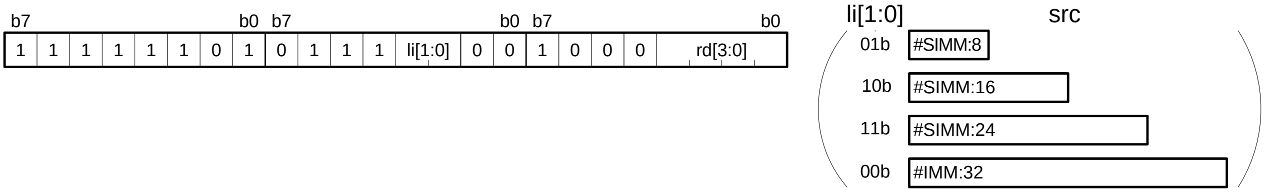
rs[3:0]/rs2[3:0]	src/src2	
0000b to 1111b	Rs/Rs2	R0 (SP) to R15

DIV DIV

Code Size

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

(1) DIV src, dest

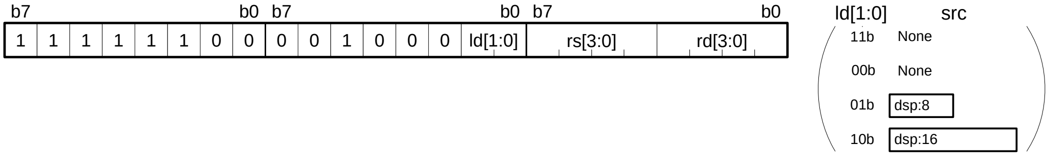


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

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

(2) DIV src, dest

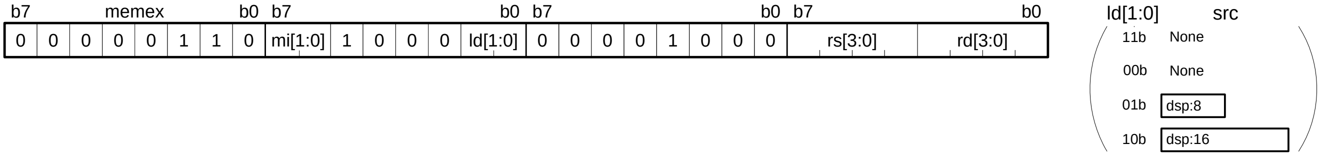
When memex == UB or src == Rs



b7							b0	b7						b0	b7	b0
1	1	1	1	1	1	0	0	0	0	1	0	0	0	ld[1:0]	rs[3:0]	rd[3:0]

ld[1:0]	src
11b	None
00b	None
01b	dsp:8
10b	dsp:16

When memex != UB



b7			memex				b0	b7					b0	b7							b0	b7	b0
0	0	0	0	0	1	1	0	mi[1:0]	1	0	0	0	ld[1:0]	0	0	0	0	1	0	0	0	rs[3:0]	rd[3:0]

ld[1:0]	src
11b	None
00b	None
01b	dsp:8
10b	dsp:16

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

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

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

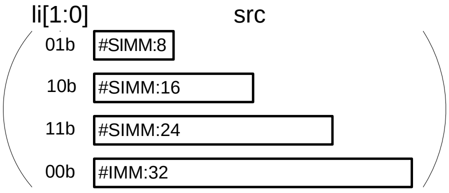
DIVU DIVU

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

b7							b0	b7						b0	b7				b0
1	1	1	1	1	1	0	1	0	1	1	1	li[1:0]	0	0	1	0	0	1	rd[3:0]



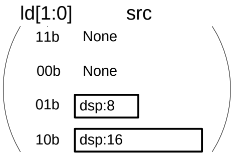
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

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

(2) DIVU src, dest

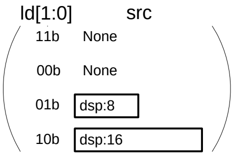
When memex == UB or src == Rs

b7							b0	b7						b0	b7	b0
1	1	1	1	1	1	0	0	0	0	1	0	0	1	ld[1:0]	rs[3:0]	rd[3:0]



When memex != UB

b7	memex						b0	b7					b0	b7							b0	b7	b0
0	0	0	0	0	1	1	0	mi[1:0]	1	0	0	0	ld[1:0]	0	0	0	0	1	0	0	1	rs[3:0]	rd[3:0]



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

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

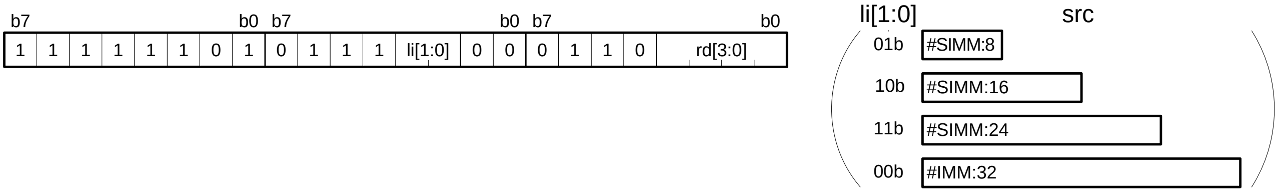
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

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

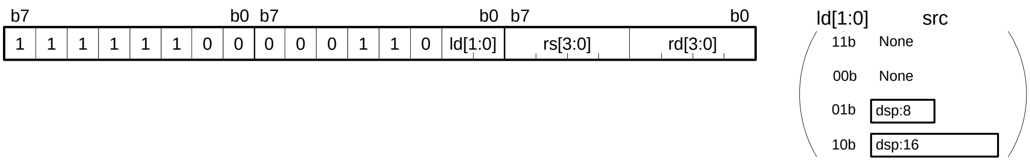


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

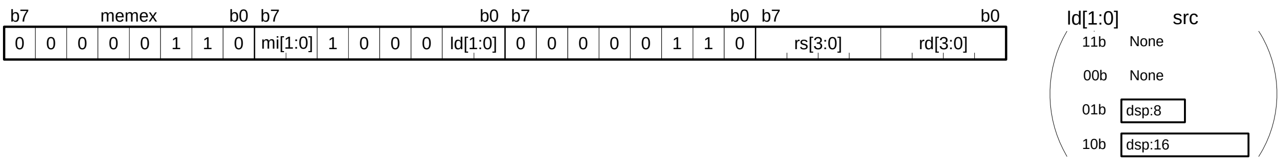
rd[3:0]	dest	
0000b to 1110b	Rd	R0 (SP) to R14

(2) EMUL src, dest

When memex == UB or src == Rs



When memex != UB


b7 memex b0 b7 b0 b7 b0 b7 b0
0 0 0 0 0 1 1 0 | mi[1:0] 1 0 0 0 ld[1:0] | 0 0 0 0 0 1 1 0 | rs[3:0] rd[3:0]
ld[1:0] src
11b None
00b None
01b dsp:8
10b dsp:16

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

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

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

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

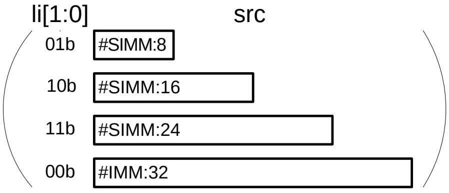
EMULU EMULU

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

b7							b0	b7						b0	b7				b0
1	1	1	1	1	1	0	1	0	1	1	1	li[1:0]	0	0	0	1	1	1	rd[3:0]



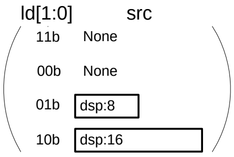
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

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

(2) EMULU src, dest

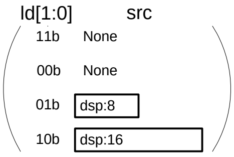
When memex == UB or src == Rs

b7							b0	b7						b0	b7	b0
1	1	1	1	1	1	0	0	0	0	0	1	1	1	ld[1:0]	rs[3:0]	rd[3:0]



When memex != UB

b7				memex			b0	b7					b0	b7							b0	b7	b0
0	0	0	0	0	1	1	0	mi[1:0]	1	0	0	0	ld[1:0]	0	0	0	0	0	1	1	1	rs[3:0]	rd[3:0]



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

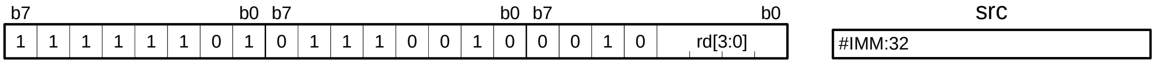
FADD FADD

Products of the RX100 Series and RX200 Series do not support the FADD instruction.

Code Size

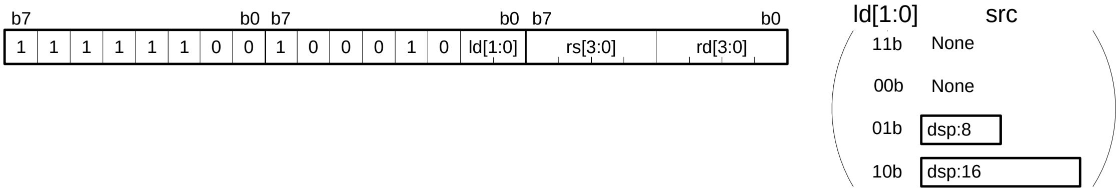
Syntax	src	dest	Code Size (Byte)
(1) FADD src, dest	#IMM:32	Rd	7
(2) FADD src, dest	Rs	Rd	3
	[Rs].L	Rd	3
	dsp:8[Rs].L	Rd	4
	dsp:16[Rs].L	Rd	5

(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

Products of the RX100 Series and RX200 Series do not support the FCMP instruction.

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

b7							b0	b7							b0	b7							b0
1	1	1	1	1	1	0	1	0	1	1	1	0	0	1	0	0	0	0	1	rs[3:0]			

src
#IMM:32

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

(2) FCMP src, src2

b7							b0	b7							b0	b7							b0
1	1	1	1	1	1	0	0	1	0	0	0	0	1	ld[1:0]		rs[3:0]				rs2[3:0]			

ld[1:0]	src
11b	None
00b	None
01b	dsp:8
10b	dsp:16

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

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

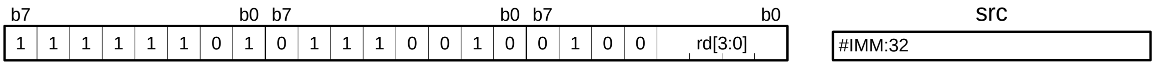
FDIV FDIV

Products of the RX100 Series and RX200 Series do not support the FDIV instruction.

Code Size

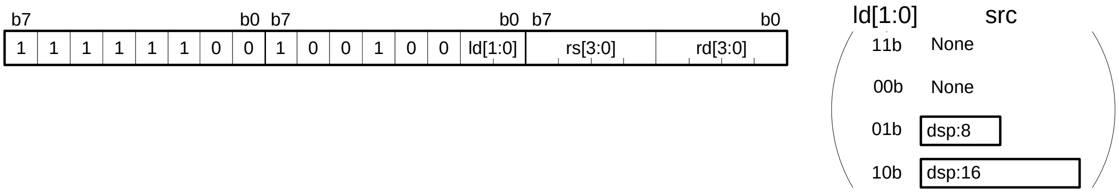
Syntax	src	dest	Code Size (Byte)
(1) FDIV src, dest	#IMM:32	Rd	7
(2) FDIV src, dest	Rs	Rd	3
	[Rs].L	Rd	3
	dsp:8[Rs].L	Rd	4
	dsp:16[Rs].L	Rd	5

(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

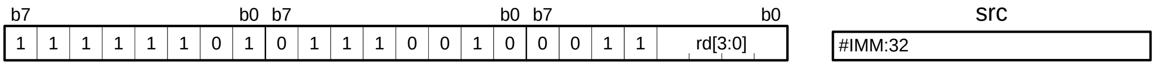
FMUL FMUL

Products of the RX100 Series and RX200 Series do not support the FMUL instruction.

Code Size

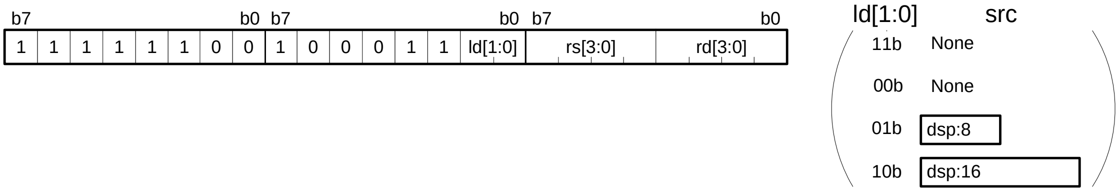
Syntax	src	dest	Code Size (Byte)
(1) FMUL src, dest	#IMM:32	Rd	7
(2) FMUL src, dest	Rs	Rd	3
	[Rs].L	Rd	3
	dsp:8[Rs].L	Rd	4
	dsp:16[Rs].L	Rd	5

(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

Products of the RX100 Series and RX200 Series do not support the FSUB instruction.

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

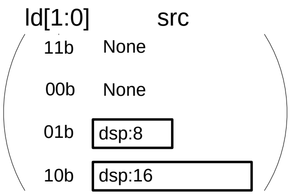
b7							b0	b7							b0	b7				b0
1	1	1	1	1	1	0	1	0	1	1	1	0	0	1	0	0	0	0	0	rd[3:0]

src
#IMM:32

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

(2) FSUB src, dest

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	0	1	0	0	0	0	0	ld[1:0]		rs[3:0]	rd[3:0]



ld[1:0]	src
11b	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

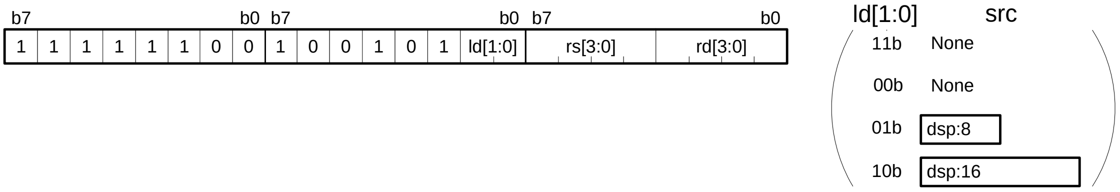
FTOI FTOI

Products of the RX100 Series and RX200 Series do not support the FTOI instruction.

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]

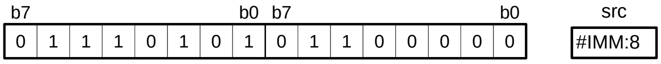
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

INT INT

Code Size

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

(1) INT src



ITOF ITOF

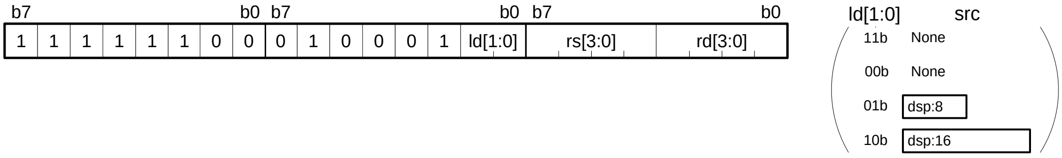
Products of the RX100 Series and RX200 Series do not support the ITOF instruction.

Code Size

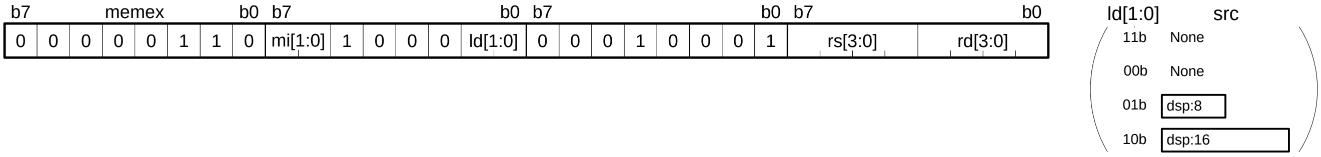
Syntax	src	dest	Code Size (Byte)
(1) ITOF src, dest	Rs	Rd	3
	[Rs].memex	Rd	3 (memex == UB) 4 (memex != UB)
	dsp:8[Rs].memex	Rd	4 (memex == UB) 5 (memex != UB)
	dsp:16[Rs].memex	Rd	5 (memex == UB) 6 (memex != UB)

(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

b7							b0	b7							b0
0	1	1	1	1	1	1	1	0	0	0	0	rs[3:0]			

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

JSR JSR

Code Size

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

(1) JSR src

b7							b0	b7							b0
0	1	1	1	1	1	1	1	0	0	0	1	rs[3:0]			

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

MACHI MACHI

Code Size

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

(1) MACHI src, src2

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	1	0	0	0	0	0	1	0	0	rs[3:0]	rs2[3:0]

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

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	1	0	0	0	0	0	1	0	1	rs[3:0]	rs2[3:0]

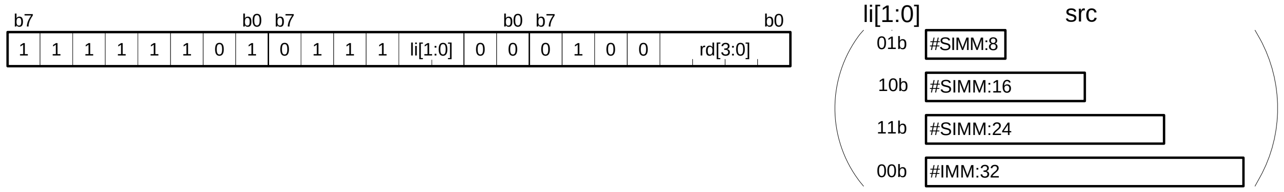
rs[3:0]/rs2[3:0]	src/src2	
0000b to 1111b	Rs/Rs2	R0 (SP) to R15

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

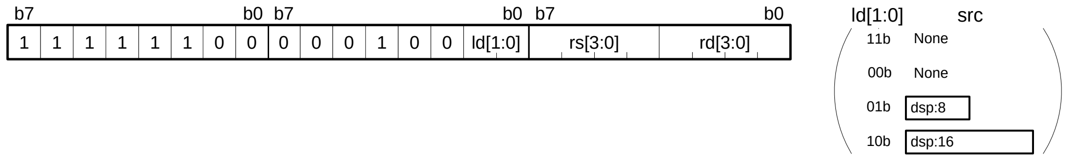


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

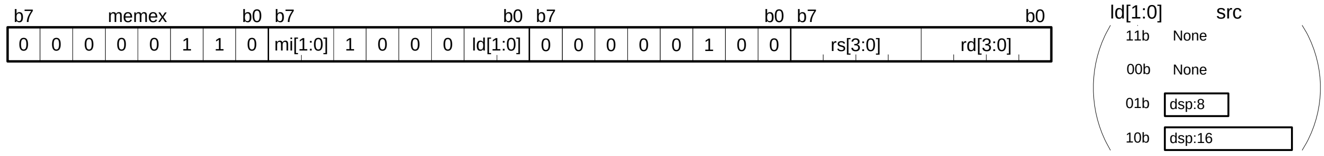
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) MAX src, dest

When memex == UB or src == Rs



When memex != UB


b7
memex
b0
b7
b0
b7
b0
b7
b0
0 0 0 0 0 1 1 0 mi[1:0] 1 0 0 0 ld[1:0] 0 0 0 0 0 1 0 0 rs[3:0] rd[3:0]
ld[1:0]
src
11b None
00b None
01b dsp:8
10b dsp:16

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

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

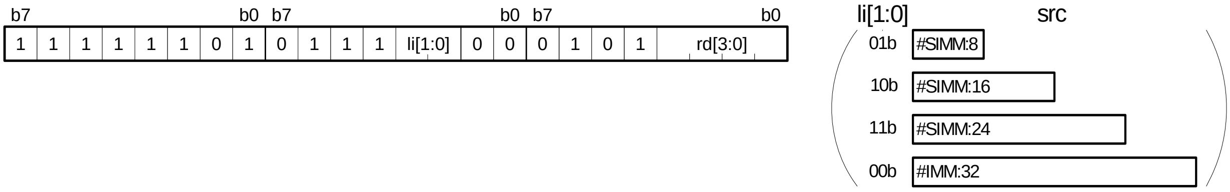
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

MIN MIN

Code Size

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

(1) MIN src, dest



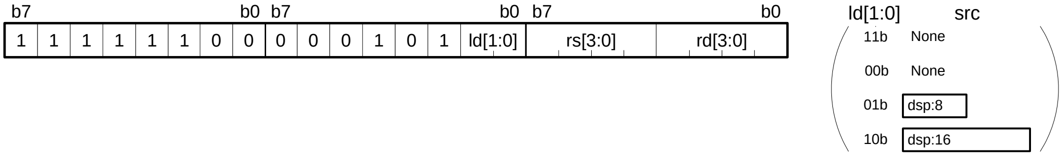
b7							b0	b7						b0	b7				b0
1	1	1	1	1	1	0	1	0	1	1	1	li[1:0]	0	0	0	1	0	1	rd[3:0]

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

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

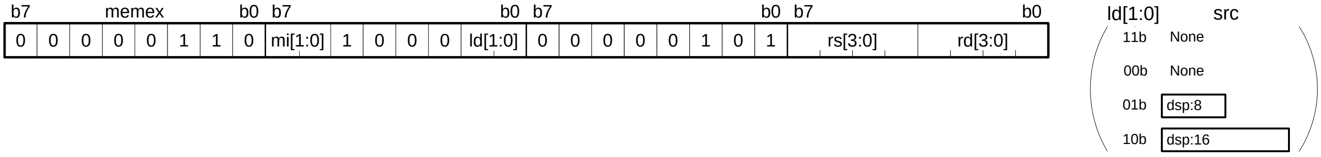
(2) MIN src, dest

When memex == UB or src == Rs



b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	0	0	0	0	1	0	1	ld[1:0]		rs[3:0]	rd[3:0]

When memex != UB



b7			memex				b0	b7					b0	b7							b0	b7	b0
0	0	0	0	0	1	1	0	mi[1:0]	1	0	0	0	ld[1:0]	0	0	0	0	0	1	0	1	rs[3:0]	rd[3:0]

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

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

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

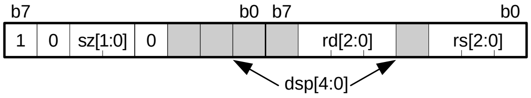
MOV MOV

Code Size

Syntax	Size	Processing Size	src	dest	Code Size (Byte)
(1) MOV.size src, dest	B/W/L	size	Rs (Rs = R0 to R7)	dsp:5[Rd] (Rd = R0 to R7)	2
(2) MOV.size src, dest	B/W/L	L	dsp:5[Rs] (Rs = R0 to R7)	Rd (Rd = R0 to R7)	2
(3) MOV.size src, dest	L	L	#UIMM:4	Rd	2
(4) MOV.size src, dest	B	B	#IMM:8	dsp:5[Rd] (Rd = R0 to R7)	3
	W/L	size	#UIMM:8	dsp:5[Rd] (Rd = R0 to R7)	3
(5) MOV.size src, dest	L	L	#UIMM:8	Rd	3
(6) MOV.size src, dest	L	L	#SIMM:8	Rd	3
	L	L	#SIMM:16	Rd	4
	L	L	#SIMM:24	Rd	5
	L	L	#IMM:32	Rd	6
(7) MOV.size src, dest	B/W	L	Rs	Rd	2
	L	L	Rs	Rd	2
(8) MOV.size src, dest	B	B	#IMM:8	[Rd]	3
	B	B	#IMM:8	dsp:8[Rd]	4
	B	B	#IMM:8	dsp:16[Rd]	5
	W	W	#SIMM:8	[Rd]	3
	W	W	#SIMM:8	dsp:8[Rd]	4
	W	W	#SIMM:8	dsp:16[Rd]	5
	W	W	#IMM:16	[Rd]	4
	W	W	#IMM:16	dsp:8[Rd]	5
	W	W	#IMM:16	dsp:16[Rd]	6
	L	L	#SIMM:8	[Rd]	3
	L	L	#SIMM:8	dsp:8[Rd]	4
	L	L	#SIMM:8	dsp:16 [Rd]	5
	L	L	#SIMM:16	[Rd]	4
	L	L	#SIMM:16	dsp:8[Rd]	5
	L	L	#SIMM:16	dsp:16 [Rd]	6
	L	L	#SIMM:24	[Rd]	5
	L	L	#SIMM:24	dsp:8[Rd]	6
	L	L	#SIMM:24	dsp:16 [Rd]	7
	L	L	#IMM:32	[Rd]	6
	L	L	#IMM:32	dsp:8[Rd]	7
	L	L	#IMM:32	dsp:16 [Rd]	8
(9) MOV.size src, dest	B/W/L	L	[Rs]	Rd	2
	B/W/L	L	dsp:8[Rs]	Rd	3
	B/W/L	L	dsp:16[Rs]	Rd	4
(10) MOV.size src, dest	B/W/L	L	[Ri, Rb]	Rd	3
(11) MOV.size src, dest	B/W/L	size	Rs	[Rd]	2
	B/W/L	size	Rs	dsp:8[Rd]	3
	B/W/L	size	Rs	dsp:16[Rd]	4

Syntax	Size	Processing Size	src	dest	Code Size (Byte)
(12) MOV.size src, dest	B/W/L	size	Rs	[Ri, Rb]	3
(13) MOV.size src, dest	B/W/L	size	[Rs]	[Rd]	2
	B/W/L	size	[Rs]	dsp:8[Rd]	3
	B/W/L	size	[Rs]	dsp:16[Rd]	4
	B/W/L	size	dsp:8[Rs]	[Rd]	3
	B/W/L	size	dsp:8[Rs]	dsp:8[Rd]	4
	B/W/L	size	dsp:8[Rs]	dsp:16[Rd]	5
	B/W/L	size	dsp:16[Rs]	[Rd]	4
	B/W/L	size	dsp:16[Rs]	dsp:8[Rd]	5
	B/W/L	size	dsp:16[Rs]	dsp:16[Rd]	6
(14) MOV.size src, dest	B/W/L	size	Rs	[Rd+]	3
	B/W/L	size	Rs	[–Rd]	3
(15) MOV.size src, dest	B/W/L	L	[Rs+]	Rd	3
	B/W/L	L	[–Rs]	Rd	3

(1) MOV.size src, dest

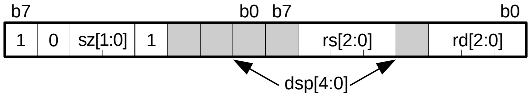


sz[1:0]	Size
00b	B
01b	W
10b	L

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

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

(2) MOV.size src, dest

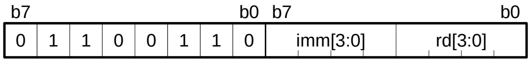


sz[1:0]	Size
00b	B
01b	W
10b	L

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

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

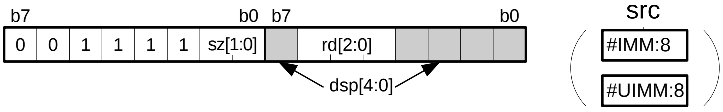
(3) MOV.size src, dest



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

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

(4) MOV.size src, dest

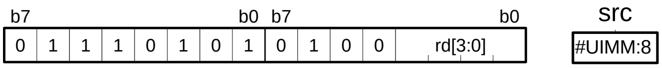


sz[1:0]	Size
00b	B
01b	W
10b	L

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

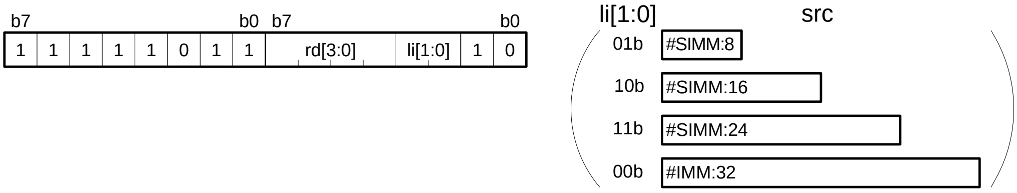
rd[2:0]	dest	
000b to 111b	Rd	R0 (SP) to R7

(5) MOV.size src, dest



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

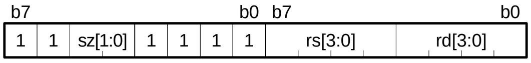
(6) MOV.size src, dest



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

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

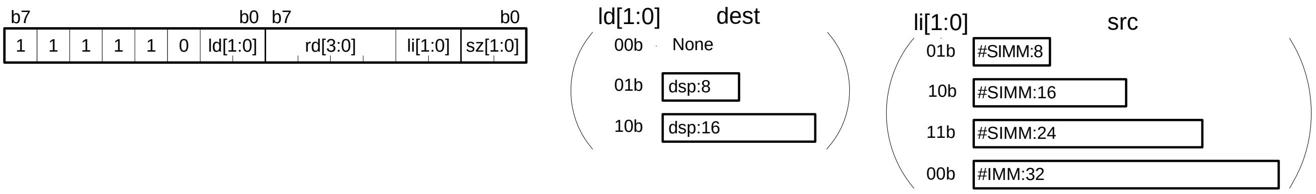
(7) MOV.size src, dest



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

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

(8) MOV.size src, dest



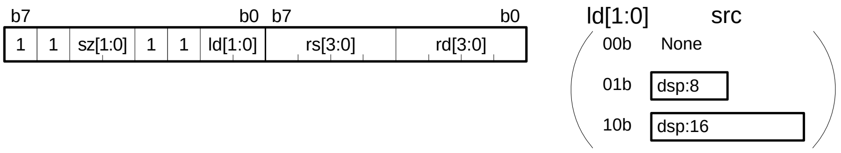
ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

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

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

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

(9) MOV.size src, dest

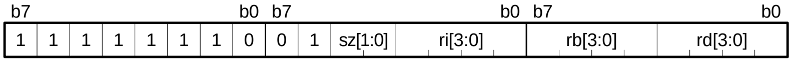


sz[1:0]	Size
00b	B
01b	W
10b	L

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

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

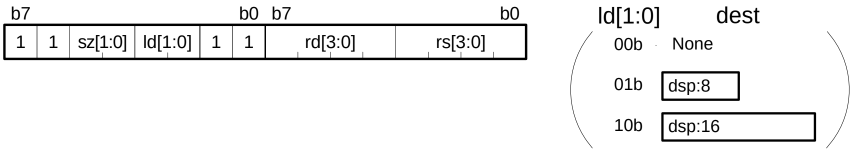
(10) MOV.size src, dest



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

ri[3:0]/rb[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Ri/Rb/Rd	R0 (SP) to R15

(11) MOV.size src, dest



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

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

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

(12) MOV.size src, dest

b7							b0	b7			b0	b7	b0
1	1	1	1	1	1	1	0	0	0	sz[1:0]	ri[3:0]	rb[3:0]	rs[3:0]

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

rs[3:0]/ri[3:0]/rb[3:0]	src/dest	
0000b to 1111b	Rs/Ri/Rb	R0 (SP) to R15

(13) MOV.size src, dest

b7				b0	b7	b0
1	1	sz[1:0]	ldd[1:0]	lds[1:0]	rs[3:0]	rd[3:0]

lds[1:0]	src
00b	None
01b	dsp:8
10b	dsp:16

ldd[1:0]	dest
00b	None
01b	dsp:8
10b	dsp:16

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

lds[1:0]/ldd[1:0]	src/dest
00b	[Rs]/[Rd]
01b	dsp:8[Rs]/dsp:8[Rd]
10b	dsp:16[Rs]/dsp:16[Rd]

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

(14) MOV.size src, dest

b7							b0	b7					b0	b7	b0
1	1	1	1	1	1	0	1	0	0	1	0	ad[1:0]	sz[1:0]	rd[3:0]	rs[3:0]

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

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

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

(15) MOV.size src, dest

b7							b0	b7					b0	b7	b0
1	1	1	1	1	1	0	1	0	0	1	0	ad[1:0]	sz[1:0]	rs[3:0]	rd[3:0]

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

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

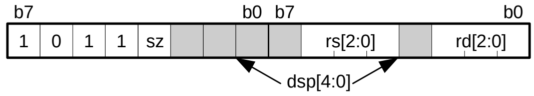
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

MOVU MOVU

Code Size

Syntax	Size	Processing Size	src	dest	Code Size (Byte)
(1) MOVU.size src, dest	B/W	L	dsp:5[Rs] (Rs = R0 to R7)	Rd (Rd = R0 to R7)	2
(2) MOVU.size src, dest	B/W	L	Rs	Rd	2
	B/W	L	[Rs]	Rd	2
	B/W	L	dsp:8[Rs]	Rd	3
	B/W	L	dsp:16[Rs]	Rd	4
(3) MOVU.size src, dest	B/W	L	[Ri, Rb]	Rd	3
(4) MOVU.size src, dest	B/W	L	[Rs+]	Rd	3
	B/W	L	[–Rs]	Rd	3

(1) MOVU.size src, dest

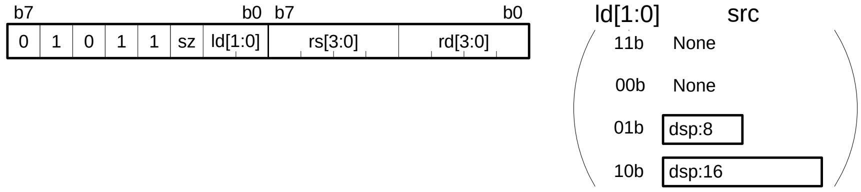


sz	Size
0b	B
1b	W

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

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

(2) MOVU.size src, dest

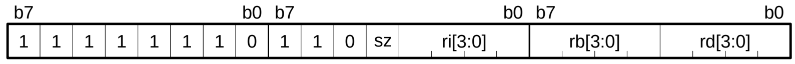


sz	Size
0b	B
1b	W

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

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

(3) MOVU.size src, dest



sz	Size
0b	B
1b	W

ri[3:0]/rb[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Ri/Rb/Rd	R0 (SP) to R15

(4) MOVU.size src, dest

b7							b0	b7						b0	b7	b0
1	1	1	1	1	1	0	1	0	0	1	1	ad[1:0]	0	sz	rs[3:0]	rd[3:0]

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

sz	Size
0b	B
1b	W

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

MUL MUL

Code Size

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

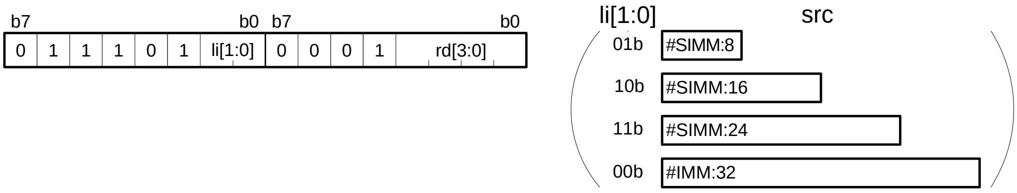
(1) MUL src, dest

b7							b0	b7	b0
0	1	1	0	0	0	1	1	imm[3:0]	rd[3:0]

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

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

(2) MUL src, dest

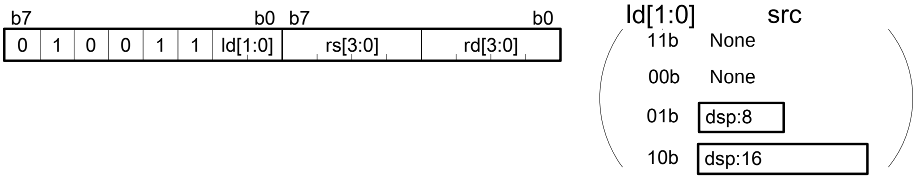


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

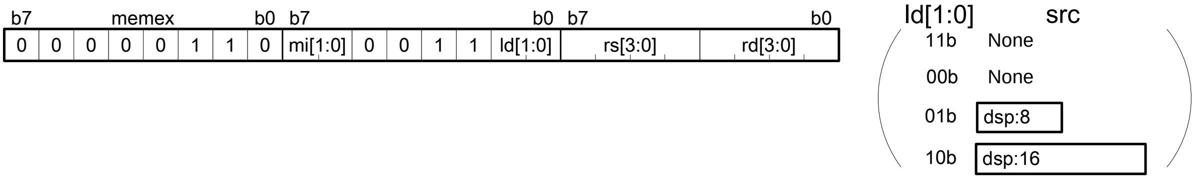
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(3) MUL src, dest

When memex == UB or src == Rs



When memex != UB

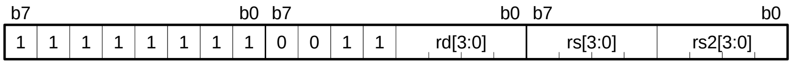


mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

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

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

(4) MUL src, src2, dest



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

MULHI MULHI

Code Size

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

(1) MULHI src, src2

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	rs[3:0]	rs2[3:0]

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

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	1	rs[3:0]	rs2[3:0]

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

b7							b0	b7							b0	b7				b0
1	1	1	1	1	1	0	1	0	0	0	1	1	1	1	1	0	0	0	0	rd[3:0]

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

b7							b0	b7							b0	b7				b0
1	1	1	1	1	1	0	1	0	0	0	1	1	1	1	1	0	0	1	0	rd[3:0]

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

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	1	0	1	1	0	1	0	1	0	cr[3:0]	rd[3:0]

cr[3:0]	src	
0000b	Rx	PSW
0001b		PC
0010b		USP
0011b		FPSW
0100b		Reserved
0101b		Reserved
0110b		Reserved
0111b		Reserved
1000b		BPSW
1001b		BPC
1010b		ISP
1011b		FINTV
1100b		INTB
1101b 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) MVTACHI src	Rs	3

(1) MVTACHI src

b7							b0	b7							b0	b7				b0
1	1	1	1	1	1	0	1	0	0	0	1	0	1	1	1	0	0	0	0	rs[3:0]

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

b7							b0	b7							b0	b7				b0
1	1	1	1	1	1	0	1	0	0	0	1	0	1	1	1	0	0	0	1	rs[3:0]

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

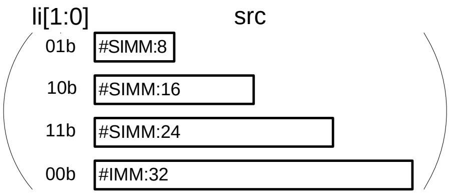
MVTC MVTC

Code Size

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

(1) MVTC src, dest

b7							b0	b7						b0	b7				b0
1	1	1	1	1	1	0	1	0	1	1	1	li[1:0]	1	1	0	0	0	0	cr[3:0]



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

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

(2) MVTC src, dest

b7							b0	b7							b0	b7							b0
1	1	1	1	1	1	0	1	0	1	1	0	1	0	0	0	rs[3:0]				cr[3:0]			

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

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

MVTIPL MVTIPL

Code Size

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

(1) MVTIPL src

b7							b0	b7							b0	b7							b0
0	1	1	1	0	1	0	1	0	1	1	1	0	0	0	0	0	0	0	0	imm[3:0]			

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

Note: The MVTIPL instruction is not available in products of the RX610 Group. Use the MVTC instruction to write interrupt priority levels to the processor interrupt-priority level (IPL[2:0]) bits in the processor status word (PSW).

NEG NEG

Code Size

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

(1) NEG dest

b7							b0	b7							b0
0	1	1	1	1	1	1	0	0	0	0	1	rd[3:0]			

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

(2) NEG src, dest

b7							b0	b7							b0	b7							b0
1	1	1	1	1	1	0	0	0	0	0	0	0	1	1	1	rs[3:0]				rd[3:0]			

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

NOP NOP

Code Size

Syntax	Code Size (Byte)
(1) NOP	1

(1) NOP

b7							b0
0	0	0	0	0	0	1	1

NOT NOT

Code Size

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

(1) NOT dest

b7							b0	b7				b0
0	1	1	1	1	1	1	0	0	0	0	0	rd[3:0]

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

(2) NOT src, dest

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	0	0	0	1	1	1	0	1	1	rs[3:0]	rd[3:0]

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

OR OR

Code Size

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

(1) OR src, dest

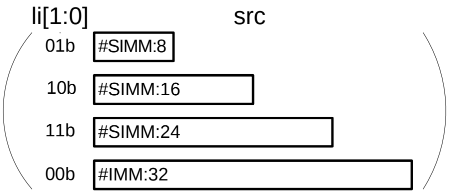
b7							b0	b7	b0
0	1	1	0	0	1	0	1	imm[3:0]	rd[3:0]

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

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

(2) OR src, dest

b7							b0	b7				b0
0	1	1	1	0	1	li[1:0]		0	0	1	1	rd[3:0]

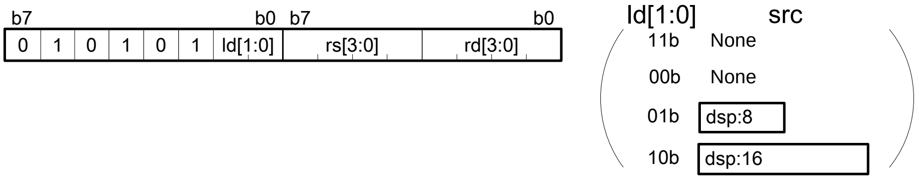


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

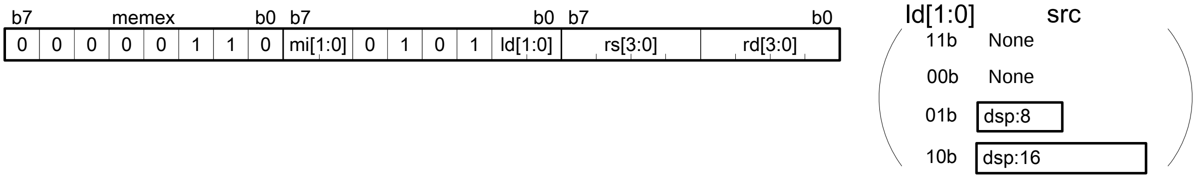
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(3) OR src, dest

When memex == UB or src == Rs



When memex != UB

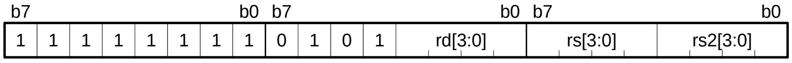


mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

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



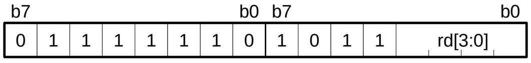
rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b to 1111b	Rs/Rs2/Rd	R0 (SP) to R15

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

b7							b0	b7				b0
0	1	1	1	1	1	1	0	1	1	1	0	cr[3:0]

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

POPM POPM

Code Size

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

(1) POPM dest-dest2

b7							b0	b7	b0
0	1	1	0	1	1	1	1	rd[3:0]	rd2[3:0]

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

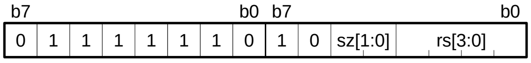
rd2[3:0]	dest2	
0010b to 1111b	Rd2	R2 to R15

PUSH PUSH

Code Size

Syntax	src	Code Size (Byte)
(1) PUSH.size src	Rs	2
(2) PUSH.size src	[Rs]	2
	dsp:8[Rs]	3
	dsp:16[Rs]	4

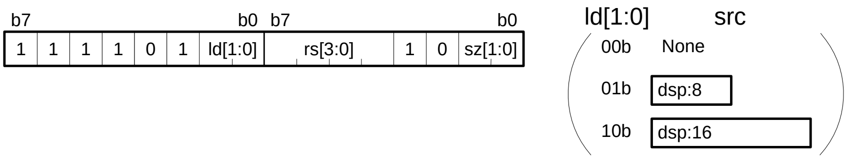
(1) PUSH.size src



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

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

(2) PUSH.size src



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

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

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

PUSHC PUSHC

Code Size

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

(1) PUSHC src

b7							b0	b7				b0
0	1	1	1	1	1	1	0	1	1	0	0	cr[3:0]

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

PUSHM PUSHM

Code Size

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

(1) PUSHM src-src2

b7							b0	b7	b0
0	1	1	0	1	1	1	0	rs[3:0]	rs2[3:0]

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

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

RACW RACW

Code Size

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

(1) RACW src

b7							b0	b7							b0	b7							b0
1	1	1	1	1	1	0	1	0	0	0	1	1	0	0	0	0	0	0	imm	0	0	0	0

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

REVL REVL

Code Size

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

(1) REVL src, dest

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	1	0	1	1	0	0	1	1	1	rs[3:0]	rd[3:0]

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

REVW REVW

Code Size

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

(1) REVW src, dest

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	1	0	1	1	0	0	1	0	1	rs[3:0]	rd[3:0]

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

RMPA RMPA

Code Size

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

(1) RMPA.size

b7							b0	b7						b0
0	1	1	1	1	1	1	1	1	0	0	0	1	1	sz[1:0]

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

ROLC ROLC

Code Size

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

(1) ROLC dest

b7							b0	b7							b0
0	1	1	1	1	1	1	0	0	1	0	1	rd[3:0]			

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

RORC RORC

Code Size

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

(1) RORC dest

b7							b0	b7							b0
0	1	1	1	1	1	1	0	0	1	0	0	rd[3:0]			

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

ROTL ROTL

Code Size

Syntax	src	dest	Code Size (Byte)
(1) ROTL src, dest	#IMM:5	Rd	3
(2) ROTL src, dest	Rs	Rd	3

(1) ROTL src, dest

b7							b0	b7							b0	b7							b0
1	1	1	1	1	1	0	1	0	1	1	0	1	1	1	imm[4:0]					rd[3:0]			

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

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

(2) ROTL src, dest

b7							b0	b7							b0	b7							b0
1	1	1	1	1	1	0	1	0	1	1	0	0	1	1	0	rs[3:0]				rd[3:0]			

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

ROTR ROTR

Code Size

Syntax	src	dest	Code Size (Byte)
(1) ROTR src, dest	#IMM:5	Rd	3
(2) ROTR src, dest	Rs	Rd	3

(1) ROTR src, dest

b7							b0	b7							b0	b7							b0
1	1	1	1	1	1	0	1	0	1	1	0	1	1	0	imm[4:0]					rd[3:0]			

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

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

(2) ROTR src, dest

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	1	0	1	1	0	0	1	0	0	rs[3:0]	rd[3:0]

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

ROUND ROUND

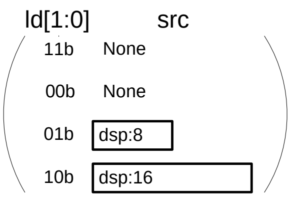
Products of the RX100 Series and RX200 Series do not support the ROUND instruction.

Code Size

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

(1) ROUND src, dest

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	0	1	0	0	1	1	0	ld[1:0]		rs[3:0]	rd[3:0]



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

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

RTE RTE

Code Size

Syntax	Code Size (Byte)
(1) RTE	2

(1) RTE

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

RTFI RTFI

Code Size

Syntax	Code Size (Byte)
(1) RTFI	2

(1) RTFI

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

RTS RTS

Code Size

Syntax	Code Size (Byte)
(1) RTS	1

(1) RTS

b7							b0
0	0	0	0	0	0	1	0

RTSD RTSD

Code Size

Syntax	src	dest	dest2	Code Size (Byte)
(1) RTSD src	#UIMM:8	–	–	2
(2) RTSD src, dest-dest2	#UIMM:8	Rd	Rd2	3

(1) RTSD src

b7							b0	src
0	1	1	0	0	1	1	1	#UIMM:8

(2) RTSD src, dest-dest2

b7							b0	b7	b0	src
0	0	1	1	1	1	1	1	rd[3:0]	rd2[3:0]	#UIMM:8

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

SAT SAT

Code Size

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

(1) SAT dest

b7							b0	b7							b0
0	1	1	1	1	1	1	0	0	0	1	1	rd[3:0]			

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

SATR SATR

Code Size

Syntax	Code Size (Byte)
(1) SATR	2

(1) SATR

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

SBB SBB

Code Size

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

(1) SBB src, dest

b7							b0	b7							b0	b7							b0
1	1	1	1	1	1	0	0	0	0	0	0	0	0	ld[1:0]		rs[3:0]				rd[3:0]			

ld[1:0]	src
11b	Rs

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

(2) SBB src, dest

b7		memex					b0	b7							b0	b7							b0	b7							b0
0	0	0	0	0	1	1	0	1	0	1	0	0	0	ld[1:0]		0	0	0	0	0	0	0	0	rs[3:0]				rd[3:0]			

ld[1:0]	src
00b	None
01b	dsp:8
10b	dsp:16

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

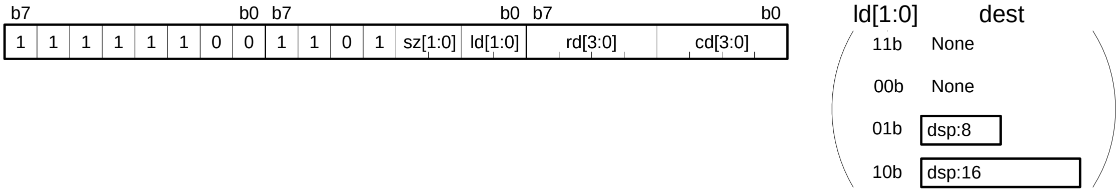
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

SCCnd *SCCnd*

Code Size

Syntax	Size	dest	Code Size (Byte)
(1) 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



b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	0	1	1	0	1	sz[1:0]		ld[1:0]		rd[3:0]	cd[3:0]

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

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

SETPSW SETPSW

Code Size

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

(1) SETPSW dest

b7							b0	b7							b0
0	1	1	1	1	1	1	1	1	0	1	0	cb[3:0]			

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

SHAR SHAR

Code Size

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

(1) SHAR src, dest

b7							b0 b7	b0
0	1	1	0	1	0	1	imm[4:0]	rd[3:0]

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

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

(2) SHAR src, dest

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	1	0	1	1	0	0	0	0	1	rs[3:0]	rd[3:0]

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

(3) SHAR src, src2, dest

b7							b0	b7			b0	b7	b0
1	1	1	1	1	1	0	1	1	0	1	imm[4:0]	rs2[3:0]	rd[3:0]

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

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

SHLL SHLL

Code Size

Syntax	src	src2	dest	Code Size (Byte)
(1) SHLL src, dest	#IMM:5	−	Rd	2
(2) SHLL src, dest	Rs	−	Rd	3
(3) SHLL src, src2, dest	#IMM:5	Rs	Rd	3

(1) SHLL src, dest

b7							b0 b7	b0
0	1	1	0	1	1	0	imm[4:0]	rd[3:0]

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

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

(2) SHLL src, dest

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	1	0	1	1	0	0	0	1	0	rs[3:0]	rd[3:0]

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

(3) SHLL src, src2, dest

b7							b0	b7			b0	b7	b0
1	1	1	1	1	1	0	1	1	1	0	imm[4:0]	rs2[3:0]	rd[3:0]

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

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

SHLR SHLR

Code Size

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

(1) SHLR src, dest

b7							b0 b7	b0
0	1	1	0	1	0	0	imm[4:0]	rd[3:0]

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

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

(2) SHLR src, dest

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	1	0	1	1	0	0	0	0	0	rs[3:0]	rd[3:0]

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

(3) SHLR src, src2, dest

b7							b0	b7			b0	b7	b0
1	1	1	1	1	1	0	1	1	0	0	imm[4:0]	rs2[3:0]	rd[3:0]

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

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

SMOVB SMOVB

Code Size

Syntax	Code Size (Byte)
(1) SMOVB	2

(1) SMOVB

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

SMOVF SMOVF

Code Size

Syntax	Code Size (Byte)
(1) SMOVF	2

(1) SMOVF

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

SMOVU SMOVU

Code Size

Syntax	Code Size (Byte)
(1) SMOVU	2

(1) SMOVU

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

SSTR SSTR

Code Size

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

(1) SSTR.size

b7							b0	b7						b0
0	1	1	1	1	1	1	1	1	0	0	0	1	0	sz[1:0]

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

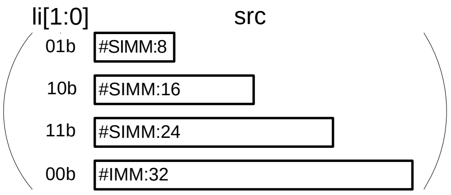
STNZ STNZ

Code Size

Syntax	src	dest	Code Size (Byte)
(1) STNZ src, dest	#SIMM:8	Rd	4
	#SIMM:16	Rd	5
	#SIMM:24	Rd	6
	#IMM:32	Rd	7

(1) STNZ src, dest

b7							b0	b7						b0	b7				b0
1	1	1	1	1	1	0	1	0	1	1	1	li[1:0]	0	0	1	1	1	1	rd[3:0]



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

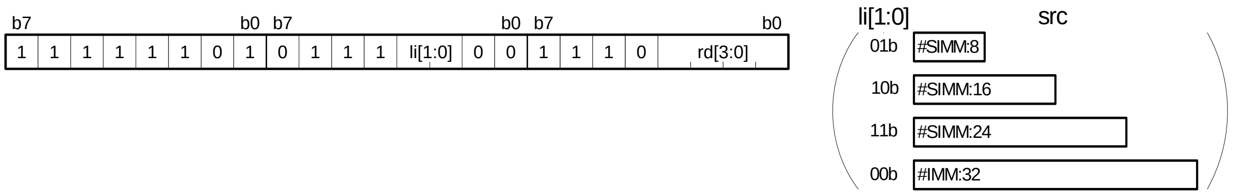
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

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

b7							b0	b7		b0
0	1	1	0	0	0	0	0	imm[3:0]	rd[3:0]	

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

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

(2) SUB src, dest

When memex == UB or src == Rs

b7						b0	b7		b0
0	1	0	0	0	0	ld[1:0]	rs[3:0]	rd[3:0]	

ld[1:0]	src
11b	None
00b	None
01b	dsp:8
10b	dsp:16

When memex != UB

b7		memex					b0	b7					b0	b7		b0
0	0	0	0	0	1	1	0	mi[1:0]	0	0	0	0	ld[1:0]	rs[3:0]	rd[3:0]	

ld[1:0]	src
11b	None
00b	None
01b	dsp:8
10b	dsp:16

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

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

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

(3) SUB src, src2, dest

b7							b0	b7					b0	b7	b0
1	1	1	1	1	1	1	1	0	0	0	0	rd[3:0]		rs[3:0]	rs2[3:0]

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

SUNTIL SUNTIL

Code Size

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

(1) SUNTIL.size

b7							b0	b7						b0
0	1	1	1	1	1	1	1	1	0	0	0	0	0	sz[1:0]

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

SWHILE SWHILE

Code Size

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

(1) SWHILE.size

b7							b0	b7						b0
0	1	1	1	1	1	1	1	1	0	0	0	0	1	sz[1:0]

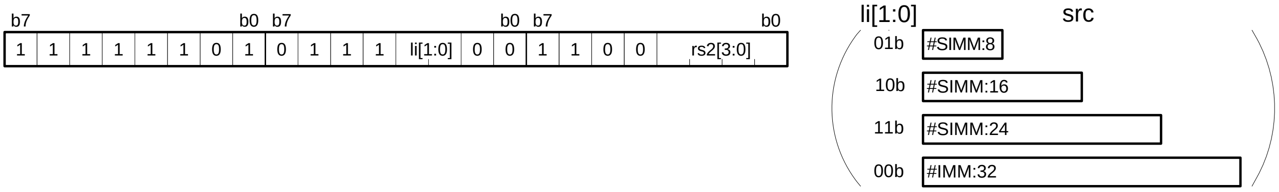
sz[1:0]	Size
00b	B
01b	W
10b	L

TST TST

Code Size

Syntax	src	src2	Code Size (Byte)
(1) TST src, src2	#SIMM:8	Rs	4
	#SIMM:16	Rs	5
	#SIMM:24	Rs	6
	#IMM:32	Rs	7
(2) TST src, src2	Rs	Rs2	3
	[Rs].memex	Rs2	3 (memex == UB) 4 (memex != UB)
	dsp:8[Rs].memex	Rs2	4 (memex == UB) 5 (memex != UB)
	dsp:16[Rs].memex	Rs2	5 (memex == UB) 6 (memex != UB)

(1) TST src, src2



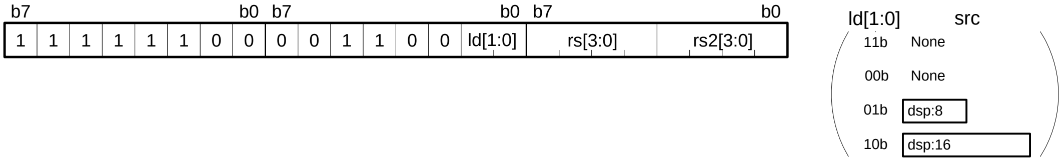
b7							b0	b7						b0	b7				b0
1	1	1	1	1	1	0	1	0	1	1	1	li[1:0]	0	0	1	1	0	0	rs2[3:0]

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

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

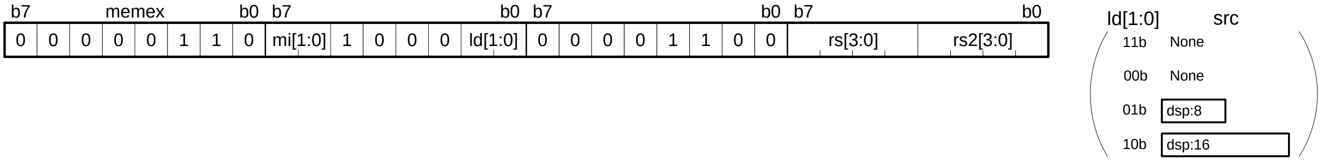
(2) TST src, src2

When memex == UB or src == Rs



b7							b0	b7						b0	b7	b0
1	1	1	1	1	1	0	0	0	0	1	1	0	0	ld[1:0]	rs[3:0]	rs2[3:0]

When memex != UB



b7			memex				b0	b7					b0	b7							b0	b7	b0
0	0	0	0	0	1	1	0	mi[1:0]	1	0	0	0	ld[1:0]	0	0	0	0	1	1	0	0	rs[3:0]	rs2[3:0]

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

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

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

WAIT WAIT

Code Size

Syntax	Code Size (Byte)
(1) WAIT	2

(1) WAIT

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

XCHG XCHG

Code Size

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

(1) XCHG src, dest

When memex == UB or src == Rs

b7							b0	b7							b0	b7	b0
1	1	1	1	1	1	0	0	0	1	0	0	0	0	ld[1:0]		rs[3:0]	rd[3:0]

ld[1:0]	src
11b	None
00b	None
01b	dsp:8
10b	dsp:16

When memex != UB

b7			memex				b0	b7						b0	b7							b0	b7	b0
0	0	0	0	0	1	1	0	mi[1:0]	1	0	0	0	ld[1:0]		0	0	0	1	0	0	0	0	rs[3:0]	rd[3:0]

ld[1:0]	src
11b	None
00b	None
01b	dsp:8
10b	dsp:16

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

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

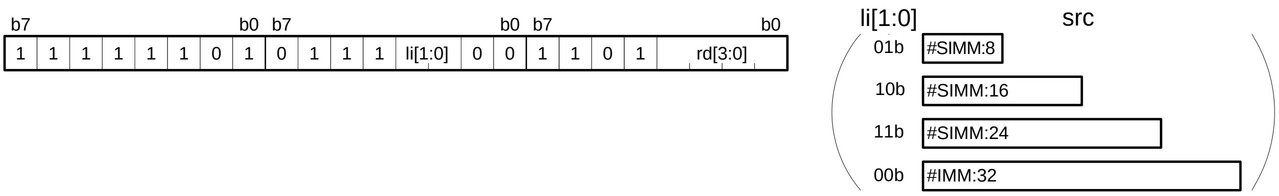
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

XOR XOR

Code Size

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

(1) XOR src, dest

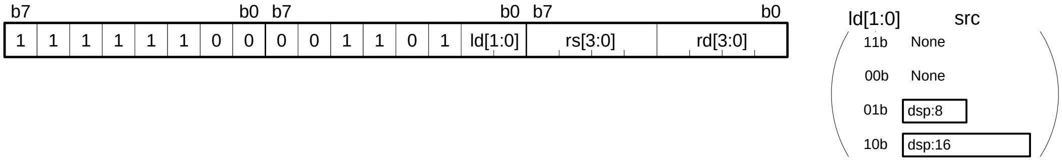


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

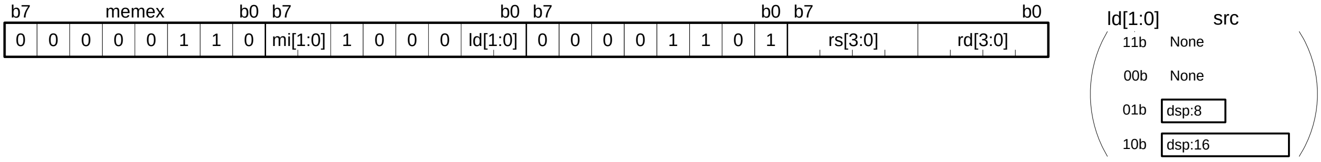
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) XOR src, dest

When memex == UB or src == Rs



When memex != UB


b7 memex b0 b7 b0 b7 b0 b7 b0
0 0 0 0 0 1 1 0 mi[1:0] 1 0 0 0 ld[1:0] 0 0 0 0 1 1 0 1 rs[3:0] rd[3:0]
ld[1:0] src
11b None
00b None
01b dsp:8
10b dsp:16

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

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

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

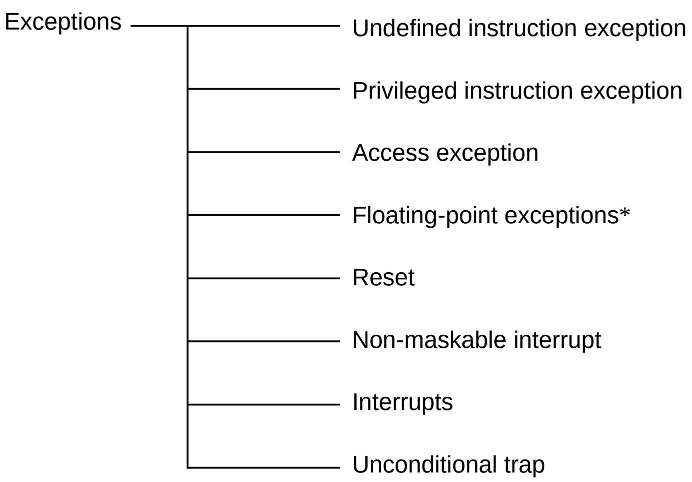
Section 5 EXCEPTIONS

5.1 Types of Exception

During the execution of a program by the CPU, the occurrence of certain events may necessitate suspending execution of the main flow of the program and starting the execution of another flow. Such events are called exceptions.

RX CPUs of the RX600 Series support eight types of exceptions and those of the RX100 Series and RX200 Series support seven types of exceptions (i.e. all except the floating-point exception). The RX CPU supports the eight types of exception listed in figure 5.1.

The occurrence of an exception causes the processor mode to switch to supervisor mode.


Exceptions
Undefined instruction exception
Privileged instruction exception
Access exception
Floating-point exceptions*
Reset
Non-maskable interrupt
Interrupts
Unconditional trap

Note: * Since products of the RX100 Series and RX200 Series do not support instructions for floating-point operations, the floating-point exception does not occur.

Figure 5.1 Types of Exception

5.1.1 Undefined Instruction Exception

An undefined instruction exception occurs when execution of an undefined instruction (an instruction not implemented) is detected.

5.1.2 Privileged Instruction Exception

A privileged instruction exception occurs when execution of a privileged instruction is detected while operation is in user mode. Privileged instructions can only be executed in supervisor mode.

5.1.3 Access Exception

When it detects an error in memory access, the CPU generates an access exception. Detection of memory protection errors for memory protection units generates exceptions of two types: instruction-access exceptions and operand-access exceptions.

5.1.4 Floating-Point Exceptions

Floating-point exceptions include the five specified in the IEEE754 standard, namely overflow, underflow, inexact, division-by-zero, and invalid operation, and a further floating-point exception that is generated on the detection of unimplemented processing. The exception processing of floating-point exceptions is masked when the EX, EU, EZ, EO, or EV bit in FPSW is 0.

Note: Since products of the RX100 Series and RX200 Series do not support instructions for floating-point operations, the floating-point exception does not occur.

5.1.5 Reset

A reset through input of the reset signal to the CPU causes the exception handling. This has the highest priority of any exception and is always accepted.

5.1.6 Non-Maskable Interrupt

The non-maskable interrupt is generated by input of the non-maskable interrupt signal to the CPU and is only used when the occurrence of a fatal fault has been detected in the system. Never end the exception handling routine for the non-maskable interrupt with an attempt to return to the program that was being executed at the time of interrupt generation.

5.1.7 Interrupts

Interrupts are generated by the input of interrupt signals to the CPU. The interrupt with the highest priority can be selected for handling as a fast interrupt. In the case of the fast interrupt, hardware pre-processing and hardware post-processing are handled fast. The priority level of the fast interrupt is fifteen (the highest)*. The exception processing of interrupts is masked when the I bit in PSW is 0.

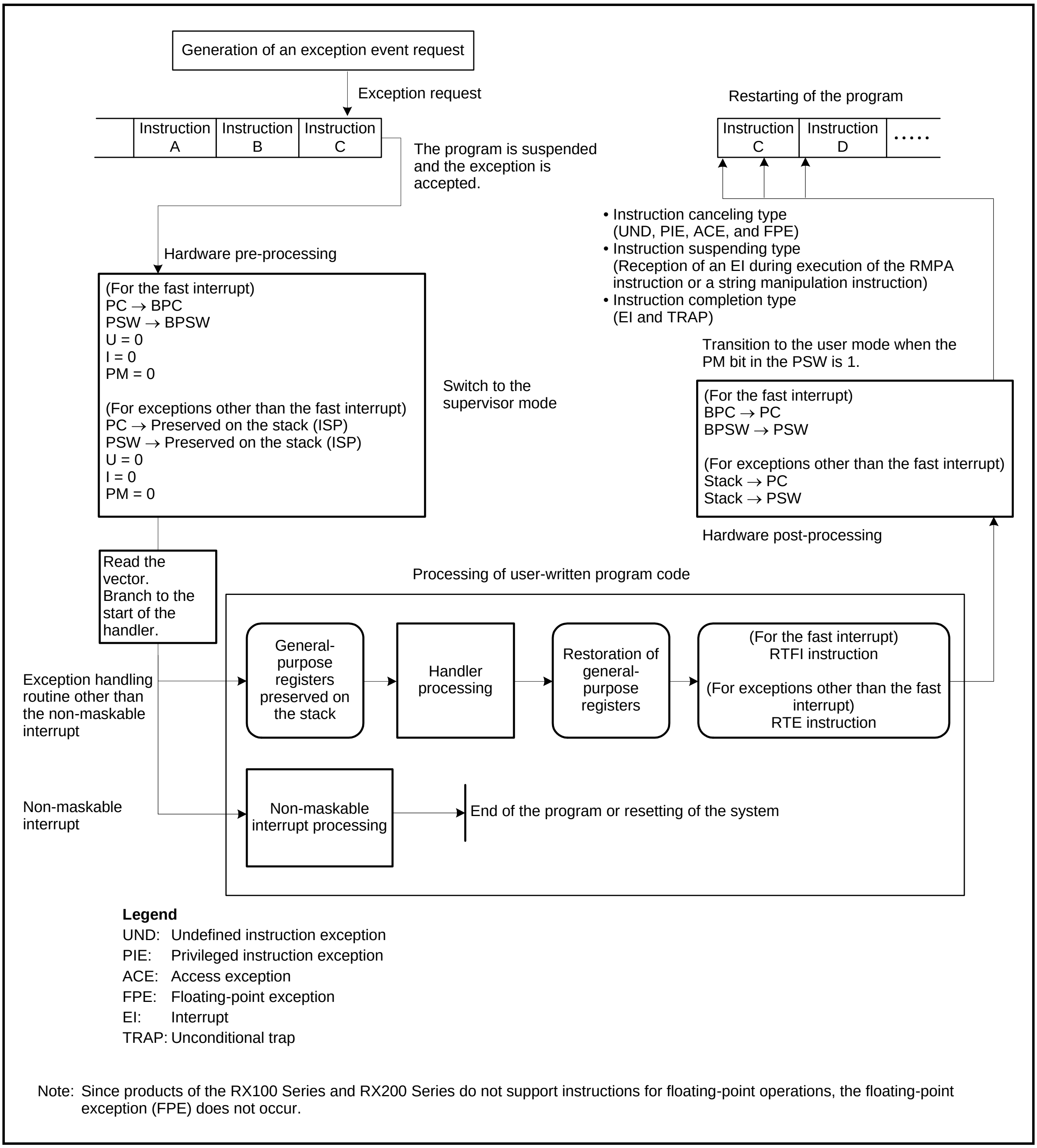
Note: * The priority level of the fast interrupt is seven (the highest) in products of the RX610 Group.

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.


Generation of an exception event request
Exception request
Instruction A
Instruction B
Instruction C
The program is suspended and the exception is accepted.
Restarting of the program
Instruction C
Instruction D
.....
• Instruction canceling type (UND, PIE, ACE, and FPE)
• Instruction suspending type (Reception of an EI during execution of the RMPA instruction or a string manipulation instruction)
• Instruction completion type (EI and TRAP)
Hardware pre-processing
(For the fast interrupt)
PC → BPC
PSW → BPSW
U = 0
I = 0
PM = 0
(For exceptions other than the fast interrupt)
PC → Preserved on the stack (ISP)
PSW → Preserved on the stack (ISP)
U = 0
I = 0
PM = 0
Switch to the supervisor mode
Transition to the user mode when the PM bit in the PSW is 1.
(For the fast interrupt)
BPC → PC
BPSW → PSW
(For exceptions other than the fast interrupt)
Stack → PC
Stack → PSW
Hardware post-processing
Read the vector. Branch to the start of the handler.
Processing of user-written program code
Exception handling routine other than the non-maskable interrupt
General-purpose registers preserved on the stack
Handler processing
Restoration of general-purpose registers
(For the fast interrupt)
RTFI instruction
(For exceptions other than the fast interrupt)
RTE instruction
Non-maskable interrupt
Non-maskable interrupt processing
End of the program or resetting of the system

Legend

UND:	Undefined instruction exception
PIE:	Privileged instruction exception
ACE:	Access exception
FPE:	Floating-point exception
EI:	Interrupt
TRAP:	Unconditional trap

Note: Since products of the RX100 Series and RX200 Series do not support instructions for floating-point operations, the floating-point exception (FPE) does not occur.

Figure 5.2 Outline of the Exception Handling Procedure

When an exception is accepted, hardware processing by the RX CPU is followed by vector 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 RX CPU handles saving of the contents of the program counter (PC) and processor status word (PSW). In the case of the fast interrupt, the contents are saved in the backup PC (BPC) and the backup PSW (BPSW), respectively. In the case of other exceptions, the contents are preserved in the stack area. General purpose registers and control registers other than the PC and PSW that are to be used within the exception handling routine must be preserved on the stack by user program code at the start of the exception handling routine.

On completion of processing by most exception handling routine, registers preserved under program control are restored and the RTE instruction is executed to restore execution from the exception handling routine to the original program. For return from the fast interrupt, the RTFI instruction is used instead. In the case of the non-maskable interrupt, however, end the program or reset the system without returning to the original program.

Hardware post-processing by the RX CPU handles restoration of the pre-exception contents of the PC and PSW. In the case of the fast interrupt, the contents of the BPC and BPSW are restored to the PC and PSW, respectively. In the case of other exceptions, the contents are restored from the stack area to the PC and PSW.

5.3 Acceptance of Exceptions

When an exception occurs, the CPU suspends the execution of the program and processing branches to the start of the exception handling routine.

5.3.1 Timing of Acceptance and Saved PC Value

Table 5.1 lists the timing of acceptance and program counter (PC) value to be saved for each type of exception event.

Table 5.1 Timing of Acceptance and Saved PC Value

Exception		Type of Handling	Timing of Acceptance	Value Saved in the BPC/ on the Stack
Undefined instruction exception		Instruction canceling type	During instruction execution	PC value of the instruction that is generated by the exception
Privileged instruction exception		Instruction canceling type	During instruction execution	PC value of the instruction that is generated by the exception
Access exception		Instruction canceling type	During instruction execution	PC value of the instruction that is generated by the exception
Floating-point exceptions*		Instruction canceling type	During instruction execution	PC value of the instruction that is generated by the exception
Reset		Program abandonment type	Any machine cycle	None
Non-maskable interrupt	During execution of the RMPA, SCMPU, SMOVB, SMOVF, SMOVU, SSTR, SUNTIL, and SWHILE instructions	Instruction suspending type	During instruction execution	PC value of the instruction being executed
	Other than the above	Instruction completion type	At the next break between instructions	PC value of the next instruction
Interrupts	During execution of the RMPA, SCMPU, SMOVB, SMOVF, SMOVU, SSTR, SUNTIL, and SWHILE instructions	Instruction suspending type	During instruction execution	PC value of the instruction being executed
	Other than the above	Instruction completion type	At the next break between instructions	PC value of the next instruction
Unconditional trap		Instruction completion type	At the next break between instructions	PC value of the next instruction

Note: * Since products of the RX100 Series and RX200 Series do not support instructions for floating-point operations, the floating-point exception does not occur.

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 (INTB)	Stack
Unconditional trap		Relocatable vector table (INTB)	Stack

Note: * Since products of the RX100 Series and RX200 Series do not support instructions for floating-point operations, the floating-point exception does not occur.

5.4 Hardware Processing for Accepting and Returning from Exceptions

This section describes the hardware processing for accepting and returning from an exception other than a reset.

(1) Hardware pre-processing for accepting an exception

(a) Preserving the PSW

(For the fast interrupt)

PSW → BPSW

(For other exceptions)

PSW → Stack area

Note: The FPSW is not preserved by hardware pre-processing. Therefore, if this is used within the exception handling routine for floating-point instructions, the user should ensure that it is preserved in the stack area from within the exception handling routine.

(b) Updating of the PM, U, and I bits in the PSW

I: Cleared to 0

U: Cleared to 0

PM: Cleared to 0

(c) Preserving the PC

(For the fast interrupt)

PC → BPC

(For other exceptions)

PC → Stack area

(d) Set the branch-destination address of the exception handling routine in the PC

Processing is shifted to the exception handling routine by acquiring the vector corresponding to the exception and branching accordingly.

(2) Hardware post-processing for executing RTE and RTFI instructions

(a) Restoring the PSW

(For the fast interrupt)

BPSW → PSW

(For other exceptions)

Stack area → PSW

(b) Restoring the PC

(For the fast interrupt)

BPC → PC

(For other exceptions)

Stack area → PC

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 address of the processing routine is fetched from the vector 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 address of the processing routine is fetched from the vector 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 address of the processing routine is fetched from the vector 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 address of the processing routine is fetched from the vector address, FFFFFFE4h.

(5) The PC is set to the fetched address and processing branches to the start of the exception handling routine.

Note: Since products of the RX100 Series and RX200 Series do not support instructions for floating-point operations, the floating-point exception does not occur.

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 address of the processing routine is fetched from the vector 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. 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 address of the processing routine 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 control registers (BPC and BPSW) immediately before the exception handling sequence.

Table 5.3 Return from Exception Handling Routines

Exception		Instruction for Return
Undefined instruction exception		RTE
Privileged instruction exception		RTE
Access exception		RTE
Floating-point exceptions*		RTE
Reset		Return is impossible
Non-maskable interrupt		Return is impossible
Interrupts	Fast interrupt	RTFI
	Other than the above	RTE
Unconditional trap		RTE

Note: * Since products of the RX100 Series and RX200 Series do not support instructions for floating-point operations, the floating-point exception does not occur.

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*

Note: * Since products of the RX100 Series and RX200 Series do not support instructions for floating-point operations, the floating-point exception does not occur.

Index

REVISION HISTORY RX Family Software Manual

Rev.	Date	Description 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
		37	Register Direct: added
		38	Section 3 Instruction Descriptions added
		159	Section 5 EXCEPTIONS added

		Description	
Rev.	Date	Page	Summary
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
		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

Rev.	Date	Description	
		Page	Summary
0.30	Jul. 31, 2008	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
		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

Rev.	Date	Description Page	Summary
0.30	Jul. 31, 2008	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
			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

Rev.	Date	Description Page	 Summary
0.50	Feb. 3, 2009	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
			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

Rev.	Date	Description Page	 Summary
0.50	Feb. 3, 2009	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
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

		Description	
Rev.	Date	Page	Summary
1.00	June 11, 2010		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
		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

		Description	
Rev.	**Date**	**Page**	**Summary**
1.00	June 11, 2010	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
		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)
			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

RX Family User’s Manual: Software

Publication Date: Rev.0.10 Nov. 12, 2007
Rev.1.20 Apr. 15, 2013

Published by: Renesas Electronics Corporation

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700, Fax: +44-1628-651-804

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 1.3

RX Family

Renesas Electronics Corporation