User's Manual

32

RX Family RXv3e Instruction Set Architecture

User's Manual: Software

RENESAS 32-Bit MCU
RX Family

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

Renesas Electronics
www.renesas.com

Rev.1.01 Aug 2026

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation.
All trademarks and registered trademarks are the property of their respective owners.

© 2026 Renesas Electronics Corporation. All rights reserved.

How to Use This Manual

This manual is designed to provide users with an understanding of RXv3e instruction set architecture (RXv3e). The manual contains detailed descriptions of CPU features and instruction sets. The manual is intended for users who are designing application systems using this CPU. Target users are expected to understand the fundamentals of microcomputers.

Notation in This Manual

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

Classification	Notation	Meaning
Symbols	IMM	Immediate value
	SIMM	Immediate value for sign extension according to the processing size
	UIMM	Immediate value for zero extension according to the processing size
	src, src2	Source of an instruction operand
	dest	Destination of an instruction operand
	dsp	Displacement of relative addressing
	pcdsp	Displacement of relative addressing of the program counter
	[ ]	Represents indirect addressing
	Rn	General-purpose register. R0 to R15 are specifiable unless stated otherwise.
	Rs	General-purpose register as a source. R0 to R15 are specifiable unless stated otherwise.
	Rs2	In the instructions where two general-purpose registers can be specified for operand, the first general-purpose register specified as a source is described as Rs and the second general-purpose register specified as a source is described as Rs2.
	Rd	General-purpose register as a destination. R0 to R15 are specifiable unless stated otherwise.
	Rd2	In the instructions where two general-purpose registers can be specified for operand, the first general-purpose register specified as a destination is described as Rd and the second general-purpose register specified as a destination is described as Rd2.
	Rb	General-purpose register specified as a base register. R0 to R15 are specifiable unless stated otherwise.
	Ri	General-purpose register as an index register. R0 to R15 are specifiable unless stated otherwise.
	Rx	Represents a control register. The PC, ISP, USP, INTB, EXTB, PSW, BPC, BPSW, FINTV, FPSW, IVC, WID, TRCTB, WINTB, and WPRI are selectable. The control registers that are specifiable depend on the instruction. For details, refer to the descriptions of the individual instructions.
	flag	Represents a bit (U or I) or flag (O, S, Z, or C) in the PSW.
	Adest	Accumulator as a destination. A0 and A1 are specifiable.
	Asrc	Accumulator as a source. A0 and A1 are specifiable.
	tmp, tmp0, tmp1, tmp2, tmp3 etc.	Temporary registers
	slsb, dlsb, width	Indicates bit-field information for the bit-field transfer instructions.
	DRs, DRs2	Represent double-precision floating-point data registers (as sources). DR0 to DR15 are specifiable.
	DRLs	Represents the lower 32 bits in double-precision floating-point data registers (as sources). DRL0 to DRL15 are specifiable.
	DRHs	Represents the upper 32 bits in double-precision floating-point data registers (as sources). DRH0 to DRH15 are specifiable.
	DRd, DRd2	Represent double-precision floating-point data registers (as destinations). DR0 to DR15 are specifiable.
	DRLd	Represents the lower 32 bits in double-precision floating-point data registers (as destinations). DRL0 to DRL15 are specifiable.

Classification	Notation	Meaning
Symbols	DRHd	Represents the upper 32 bits in double-precision floating-point data registers (as destinations). DRH0 to DRH15 are specifiable.
	DCRs, DCRs2	Represent double-precision floating-point control registers (as sources). DPSW, DCMR, DECNT, and DEPC are specifiable.
	DCRd, DCRd2	Represent double-precision floating-point control registers (as destinations). DPSW, DCMR, DECNT, and DEPC are specifiable.
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.
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.
	.D	Double-longword (64 bits) is specified.
Branch distance specifiers	BRA.A etc.	Indicates the length of the valid bits to represent the distance to the branch relative destination.
	.S	3-bit PC forward relative is specified. The range of valid values is 3 to 10.
	.B	8-bit PC relative is specified. The range of valid values is –128 to 127.
	.W	16-bit PC relative is specified. The range of valid values is –32768 to 32767.
	.A	24-bit PC relative is specified. The range of valid values is –8388608 to 8388607.
	.L	32-bit PC relative is specified. The range of valid values is –2147483648 to 2147483647.
Size extension specifiers added to memory operands	dsp:16[Rs].UB etc.	Indicates the size of a memory operand and the type of extension. If the specifier is omitted, the memory operand is handled as longword.
	.B	Byte (8 bits) is specified. The extension is sign extension.
	.UB	Byte (8 bits) is specified. The extension is zero extension.
	.W	Word (16 bits) is specified. The extension is sign extension.
	.UW	Word (16 bits) is specified. The extension is zero extension.
	.L	Longword (32 bits) is specified.

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

Contents

How to Use This Manual 3

List of RXv3e Instruction Set for RX Family 10

Quick Page Reference in Alphabetical Order 10

1. CPU Programming Model 16

1.1 Features 16
1.2 Protect Zone Technology 17
1.3 Register Set of the CPU 18
1.3.1 General-Purpose Registers (R0 to R15) 19
1.3.2 Control Registers 19
1.3.2.1 Restrictions on Access to Control Registers 20
1.3.2.2 Interrupt Stack Pointer (ISP)/User Stack Pointer (USP) 22
1.3.2.3 Interrupt Table Register (INTB) 22
1.3.2.4 Program Counter (PC) 22
1.3.2.5 Processor Status Word (PSW) 23
1.3.2.6 Backup PC (BPC) 26
1.3.2.7 Backup PSW (BPSW) 26
1.3.2.8 Fast Interrupt Vector Register (FINTV) 26
1.3.2.9 Single-Precision Floating-Point Status Word (FPSW) 27
1.3.2.10 Exception Vector Table Register (EXTB) 29
1.3.2.11 Interrupt Vector Control Register (IVC) 30
1.3.2.12 World ID Register (WID) 32
1.3.2.13 Root Call Vector Table Register (TRCTB) 33
1.3.2.14 Vector Table Register for Interrupts That Cause World Transitions (WINTB) 33
1.3.2.15 World-Shared Processor Interrupt Priority Level Register (WPRI) 34
1.3.3 Accumulator 36
1.4 Single-Precision Floating-Point Exceptions 37
1.4.1 Overflow 37
1.4.2 Underflow 37
1.4.3 Inexact 38
1.4.4 Division-by-Zero 38
1.4.5 Invalid Operation 39
1.4.6 Unimplemented Processing 39
1.5 Processor Mode 40
1.5.1 Supervisor Mode 40
1.5.2 User Mode 40
1.5.3 Privileged Instruction 40
1.5.4 Switching Between Processor Modes 41
1.6 Data Types 42
1.6.1 Integer 42
1.6.2 Single-Precision Floating-Point Number 42
1.6.3 Bit 43
1.6.4 String 43
1.7 Data Arrangement 44
1.7.1 Data Arrangement in Registers 44
1.7.2 Data Arrangement in Memory 44
1.8 Vector Table 45
1.8.1 Exception Vector Table 45
1.8.2 Interrupt Vector Table 46
1.8.3 Root Call Vector Table 47
1.8.4 Vector Table for Interrupts that Cause World Transitions 47
1.9 Address Space 48
1.10 Register Bank Save Function 49
1.11 Double-Precision Floating-Point Coprocessor 51
1.11.1 Features 51
1.11.2 Double-Precision Floating-Point Register Set 51

1.11.2.1 Double-Precision Floating-Point Data Registers (DR0 to DR15) 52
1.11.2.2 Double-Precision Floating-Point Control Registers 52
1.11.3 Double-Precision Floating-Point Exceptions 57
1.11.3.1 Overflow 57
1.11.3.2 Underflow 57
1.11.3.3 Inexact 58
1.11.3.4 Division-by-Zero 58
1.11.3.5 Invalid Operation 59
1.11.3.6 Unimplemented Processing 59
1.11.4 Data Types (for the Double-Precision Floating-Point Coprocessor) 60
1.11.4.1 Double-Precision Floating-Point Numbers 60
1.11.5 Data Arrangement (for the Double-Precision Floating-Point Coprocessor) 61
1.11.5.1 Arrangement of Data in the Double-Precision Floating-Point Registers 61
1.11.5.2 Arrangement of Data for Double-Precision Floating-Point Numbers in Memory 61

2. Addressing Modes 62
2.1 Guide to This Section 63
2.2 Addressing Modes 64
2.2.1 Ranges for Immediate Values 69

3. Instruction Descriptions 70
3.1 Overview of Instruction Set 70
3.2 List of RXv3e Extended Instructions 76
3.2.1 RXv3e Newly Added Instructions 76
3.2.2 Specification Extended Instructions 77
3.3 Guide to This Section 78
3.4 Instructions in Detail 83
3.4.1 Standard provided instructions 83
3.4.2 Instructions for Register Bank Save Function 229
3.4.3 Double-Precision Floating-Point Instructions 232
3.4.4 Protect Zone Instructions 271

4. Instruction Code 276
4.1 Guide to This Section 276
4.2 Instruction Code Described in Detail 279
4.2.1 Standard provided instructions 279
4.2.2 Instructions for Register Bank Save Function 369
4.2.3 Double-Precision Floating-Point Instructions 371
4.2.4 Protect Zone Instructions 385

5. Exceptions 390
5.1 Types of Exception 390
5.1.1 Undefined Instruction Exception 391
5.1.2 Privileged Instruction Exception 391
5.1.3 Access Exception 391
5.1.4 Secure Exception 391
5.1.5 Address Exceptions 391
5.1.6 Single-Precision Floating-Point Exceptions 391
5.1.7 Reset 391
5.1.8 Non-Maskable Interrupt 392
5.1.9 Interrupts 392
5.1.10 Unconditional Trap 393
5.2 Exception Handling Procedure 394
5.2.1 Exception Handling Procedure without World Transitions 395
5.2.2 Exception Handling Procedure with World Transitions 397
5.3 Acceptance of Exceptions 399
5.3.1 Timing of Acceptance and Saved PC Value 399

5.3.2 Vector and Location for Saving the PC and PSW 400
5.4 Hardware Processing for Accepting and Returning from Exceptions 401
5.4.1 Hardware Processing for Accepting and Returning from Exceptions Not Involving World Transitions 401
5.4.2 Hardware Processing for Accepting and Returning from Exceptions Involving World Transitions 402
5.5 Hardware Pre-processing 403
5.5.1 Undefined Instruction Exception 403
5.5.2 Privileged Instruction Exception 403
5.5.3 Access Exception 403
5.5.4 Secure Exception 403
5.5.5 Address Exceptions 403
5.5.6 Single-Precision Floating-Point Exceptions 404
5.5.7 Reset 404
5.5.8 Non-Maskable Interrupts (with Normal Acceptance) 404
5.5.9 Non-Maskable Interrupts (with World-Transition Acceptance) 404
5.5.10 Interrupts (with Normal Acceptance) 404
5.5.11 Interrupts (with World-Transition Acceptance) 405
5.5.12 Unconditional Trap (with Normal Acceptance) 405
5.5.13 Unconditional Trap (with World-Transition Acceptance) 405
5.6 Return from Exception Handling Routines 406
5.7 Order of Priority for Exceptions 406
5.8 Exception Generated in Coprocessor 407
5.8.1 Double-Precision Floating-Point Exceptions 407

Index 408

REVISION HISTORY 412

List of RXv3e Instruction Set for RX Family

Quick Page Reference in Alphabetical Order (1 / 6)

Mnemonic		Function	Instruction Described in Detail (on Page)	Instruction Code Described in Detail (on Page)
ABS		Absolute value	84	280
ADC		Add with carry	85	281
ADD		Add without carry	86	282
AND		Logical AND	88	284
BCLR		Clear a bit	90	286
B*Cnd*	BGEU	Conditional relative branch	91	288
	BC		91	288
	BEQ		91	288
	BZ		91	288
	BGTU		91	288
	BPZ		91	288
	BGE		91	288
	BGT		91	288
	BO		91	288
	BLTU		91	288
	BNC		91	288
	BNE		91	288
	BNZ		91	288
	BLEU		91	288
	BN		91	288
	BLE		91	288
	BLT		91	288
	BNO		91	288
BFMOV		Bit field transfer	92	290
BFMOVZ		Bit field transfer with setting the other bits at the destination to zero	93	290

Quick Page Reference in Alphabetical Order (2 / 6)

Mnemonic		Function	Instruction Described in Detail (on Page)	Instruction Code Described in Detail (on Page)
BM*Cnd*	BMGEU	Conditional bit transfer	94	291
	BMC		94	291
	BMEQ		94	291
	BMZ		94	291
	BMGTU		94	291
	BMPZ		94	291
	BMGE		94	291
	BMGT		94	291
	BMO		94	291
	BMLTU		94	291
	BMNC		94	291
	BMNE		94	291
	BMNZ		94	291
	BMLEU		94	291
	BMN		94	291
	BMLE		94	291
	BMLT		94	291
	BMNO		94	291
BNOT		Not a bit	96	292
BRA		Unconditional relative branch	97	293
BRK		Unconditional trap	98	294
BSET		Set a bit	99	294
BSR		Relative branch to subroutine	100	296
BTST		Test a bit	101	297
CLRPSW		Clear a flag or bit in the PSW	102	298
CMP		Compare	103	299
DABS		Absolute value of double-precision floating-point	233	372
DADD		Add double-precision floating-point	234	372
DCMPcm	DCMPUN	Compare double-precision floating-point	236	373
	DCMPEQ		236	373
	DCMPLT		236	373
	DCMPLE		236	373
DDIV		Divide double-precision floating-point	239	373
DIV		Divide signed	104	300
DIVU		Divide unsigned	105	302
DMOV		Move double-precision floating-point	241	374
DMUL		Multiply double-precision floating-point	243	377
DNEG		Negate double-precision floating-point	245	377
DPOPM		Pop multiple double-precision floating-point registers from stack	246	378
DPUSHM		Push multiple double-precision floating-point registers on stack	248	379
DROUND		Round double-precision floating-point to signed integer	250	380
DSQRT		Square root of double-precision floating-point	253	380
DSUB		Subtract double-precision floating-point	255	380

Quick Page Reference in Alphabetical Order (3 / 6)

Mnemonic	Function	Instruction Described in Detail (on Page)	Instruction Code Described in Detail (on Page)
DTOF	Convert double-precision floating-point to single-precision floating-point	257	381
DTOI	Convert double-precision floating-point to signed integer	260	381
DTOU	Convert double-precision floating-point to unsigned integer	262	381
EMACA	Extended multiply-accumulate to the accumulator	106	303
EMSBA	Extended multiply-subtract from the accumulator	107	303
EMUL	Extended multiply signed	108	304
EMULA	Extended multiply to the accumulator	110	305
EMULU	Extended multiply unsigned	111	305
FADD	Add single-precision floating-point	113	307
FCMP	Compare single-precision floating-point	116	308
FDIV	Divide single-precision floating-point	118	309
FMUL	Multiply single-precision floating-point	120	310
FSQRT	Square root of single-precision floating-point	123	311
FSUB	Subtract single-precision floating-point	125	312
FTOD	Convert single-precision floating-point to double-precision floating-point	264	382
FTOI	Convert single-precision floating-point to signed integer	128	313
FTOU	Convert single-precision floating-point to unsigned integer	131	313
INT	Software interrupt	134	314
ITOD	Convert signed integer to double-precision floating-point	266	382
ITOF	Convert signed integer to single-precision floating-point	135	314
JMP	Unconditional jump	137	315
JSR	Jump to subroutine	138	315
MACHI	Multiply-Accumulate the upper words	139	316
MACLH	Multiply-Accumulate the lower and upper words	140	316
MACLO	Multiply-Accumulate the lower words	141	316
MAX	Maximum of two signed integers	142	317
MIN	Minimum of two signed integers	143	318
MOV	Move	144	319
MOVCO	Store data with LI flag clear	147	324
MOVLI	Load data with LI flag set	148	324
MOVU	Move unsigned	149	325
MSBHI	Multiply-Subtract the upper words	151	326
MSBLH	Multiply-Subtract the lower and upper words	152	326
MSBLO	Multiply-Subtract the lower words	153	327
MUL	Multiply	154	327
MULHI	Multiply the upper words	156	329
MULLH	Multiply the lower and upper words	157	329
MULLO	Multiply the lower words	158	330
MVFACGU	Move data from the guard bits of the accumulator	159	330
MVFACHI	Move data from the upper longword of the accumulator	160	331
MVFACLO	Move data from the lower longword of the accumulator	161	331
MVFACMI	Move data from the middle-order longword of the accumulator	162	332

Quick Page Reference in Alphabetical Order (4 / 6)

Mnemonic	Function	Instruction Described in Detail (on Page)	Instruction Code Described in Detail (on Page)
MVFC	Move data from a control register	163	333
MVFDC	Move data from double-precision floating-point control register	267	383
MVFDR	Move data from double-precision floating-point comparison result register	268	383
MVTACGU	Move data to the guard bits of the accumulator	164	334
MVTACHI	Move data to the upper longword of the accumulator	165	334
MVTACLO	Move data to the lower longword of the accumulator	166	334
MVTC	Move data to a control register	167	335
MVTDC	Move data to double-precision floating-point control register	269	384
MVTIPL (privileged instruction)	Move data to IPL	168	336
NEG	Negate (two's complement)	169	337
NOP	No operation	170	337
NOT	Logical NOT (one's complement)	171	338
OR	Logical OR	172	339
POP	Pop register from stack	173	340
POPC	Pop a control register from stack	174	341
POPM	Pop multiple registers from stack	175	341
PUSH	Push register on stack	176	342
PUSHC	Push a control register on stack	177	343
PUSHM	Push multiple registers on stack	178	343
RACL	Round the accumulator longword	179	344
RACW	Round the accumulator word	181	344
RDACL	Round down the accumulator longword	183	345
RDACW	Round down the accumulator word	185	345
REVL	Reverse endian within longword	187	346
REVW	Reverse endian within word	188	346
RMPA	Repeat multiply-accumulate	189	347
ROLC	Rotate left with carry	191	347
RORC	Rotate right with carry	192	348
ROTL	Rotate left	193	348
ROTR	Rotate right	194	349
ROUND	Round single-precision floating-point to signed integer	195	350
RSTR (privileged instruction)	Restore registers collectively	230	370
RTE (privileged instruction)	Return from exception	198	350
RTFI (privileged instruction)	Return from fast interrupt	199	351
RTS	Return from subroutine	200	351
RTSD	Return from subroutine after deallocating stack frame	201	351
SAT	Saturate	203	352
SATR	Saturate for RMPA	204	352
SAVE (privileged instruction)	Save registers collectively	231	370

Quick Page Reference in Alphabetical Order (5 / 6)

Mnemonic		Function	Instruction Described in Detail (on Page)	Instruction Code Described in Detail (on Page)
SBB		Subtract with borrow	205	353
SC*Cnd*	SCGEU	Store condition	206	354
	SCC		206	354
	SCEQ		206	354
	SCZ		206	354
	SCGTU		206	354
	SCPZ		206	354
	SCGE		206	354
	SCGT		206	354
	SCO		206	354
	SCLTU		206	354
	SCNC		206	354
	SCNE		206	354
	SCNZ		206	354
	SCLEU		206	354
	SCN		206	354
	SCLE		206	354
	SCLT		206	354
	SCNO		206	354
SCMPU		String compare until not equal	207	354
SETPSW		Set a flag or bit in the PSW	208	355
SHAR		Arithmetic shift right	209	356
SHLL		Logical shift left	210	357
SHLR		Logical shift right	211	358
SMOVB		String move backward	212	359
SMOVF		String move forward	213	359
SMOVU		String move until zero detected	214	359
SSTR		String store	215	360
STNZ		Store on not zero	216	360
STZ		Store on zero	217	361
SUB		Subtract without borrow	218	362
SUNTIL		String search until equal	219	363
SWHILE		String search while equal	221	363
TRCALL (privileged instruction)		Call for transition to the root world	272	386
TRMVFC (privileged instruction)		Transfer from a control register in the root world	273	386
TRMVTC (privileged instruction)		Transfer to a control register in the root world	274	387
TRRTE (privileged instruction)		Return from exception in the root world	275	389
TST		Test logical	223	364
UTOD		Convert unsigned integer to double-precision floating-point	270	384
UTOF		Convert unsigned integer to single-precision floating-point	224	365

Quick Page Reference in Alphabetical Order (6 / 6)

Mnemonic	Function	Instruction Described in Detail (on Page)	Instruction Code Described in Detail (on Page)
WAIT (privileged instruction)	Wait	226	366
XCHG	Exchange	227	366
XOR	Logical Exclusive OR	228	367

1. CPU Programming Model

The RXv3e instruction set architecture (RXv3e) has upward compatibility with the RXv3 instruction set architecture (RXv3).

- Protect Zone
  The RXv3e CPU is equipped with Protect Zone technology. This is used to define and manage the security states of the CPU core, and supports the partitioning of software execution environments in terms of these security states.
- Adoption of variable-length instruction format
  The RXv3e CPU has short formats for frequently used instructions, facilitating the development of efficient programs that take up less memory.
- Powerful instruction set
  The RXv3e supports 117 selected instructions. DSP instructions and floating-point arithmetic instructions realize high-speed arithmetic processing.
- Versatile addressing modes
  The RXv3e CPU has 11 versatile addressing modes, with register-register operations, register-memory operations, and bitwise operations included. Data transfer between memory locations is also possible.

1.1 Features

- Minimum instruction execution rate: One clock cycle
- Address space: 4-Gbyte linear addresses
- Protect Zone: Three “worlds” (non-secure, secure, and root)
- Register set of the CPU
  General purpose: Sixteen 32-bit registers
  Control: Fifteen 32-bit registers
  Accumulator: Two 72-bit registers
- Variable-length instruction format (lengths from one to eight bytes)
- 117 instructions/11 addressing modes
  Standard provided instructions: 111
    Basic instructions: 77
    Single-precision floating-point arithmetic instructions: 11
    DSP instructions: 23
  Instructions for register bank save function: 2 (as an optional function)
  Protect Zone instructions: 4
- Processor modes
  Supervisor mode and user mode
- Vector tables
  Exception vector table, interrupt vector table, root call vector table, and vector table for interrupts that cause world transitions
- Memory protection unit (as an optional function)
- Data arrangement
  Selectable as little endian or big endian
- Double-precision floating-point coprocessor (as an optional function)
  Double-precision floating-point instructions: 21

1.2 Protect Zone Technology

The CPU core, for which Protect Zone technology has been adopted, has three security states that are called “worlds”. Each world is distinguished by a world ID. World ID 0 is called the “root world”, world ID 1 is called the “secure world”, and world ID 2 is called the “non-secure world”.

Hereafter, in cases where reference is obviously to a “world”, the terms may simply be used as “root”, “secure”, and “non-secure” with the term “world” itself omitted.

The programs to be executed in and the resources accessible from the secure world and the non-secure world can be partitioned from each other. Direct transitions between the secure and non-secure worlds are not possible. Instead, such transitions must always be made via the root world. Some resources of the CPU core, such as control registers, are partitioned by the hardware of the CPU core, but other resources require partitioning by programs in the root world.

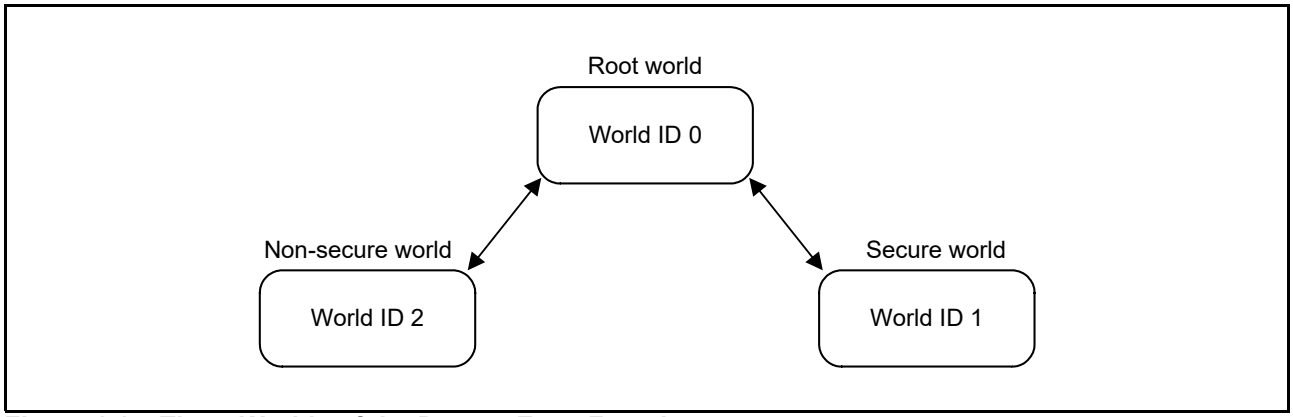


Figure 1.1 Three Worlds of the Protect Zone Function

The world following release from the reset state is the root world.

This instruction set architecture mainly defines the specifications related to the CPU core. For the specifications of interrupts or address spaces in terms of the Protect Zone function, refer to the user’s manual: hardware for each product.

The root, secure, and non-secure worlds each have a different purpose.

The root world serves as the starting point for transitions to the non-secure and secure worlds. Passing control between the other two worlds always has to proceed via the root world, and software processing related to world transitions only proceeds in this world. For this reason, certain instructions are stipulated as only being executable in the root world. Resources for the root, secure, and non-secure worlds are all accessible from the root world.

The non-secure world is assumed to be the world for the execution of non-secure applications. Only resources for the non-secure world are accessible from the non-secure world.

The secure world is assumed to be the world for the execution of secure applications. Only resources for the secure and non-secure worlds are accessible from the secure world.

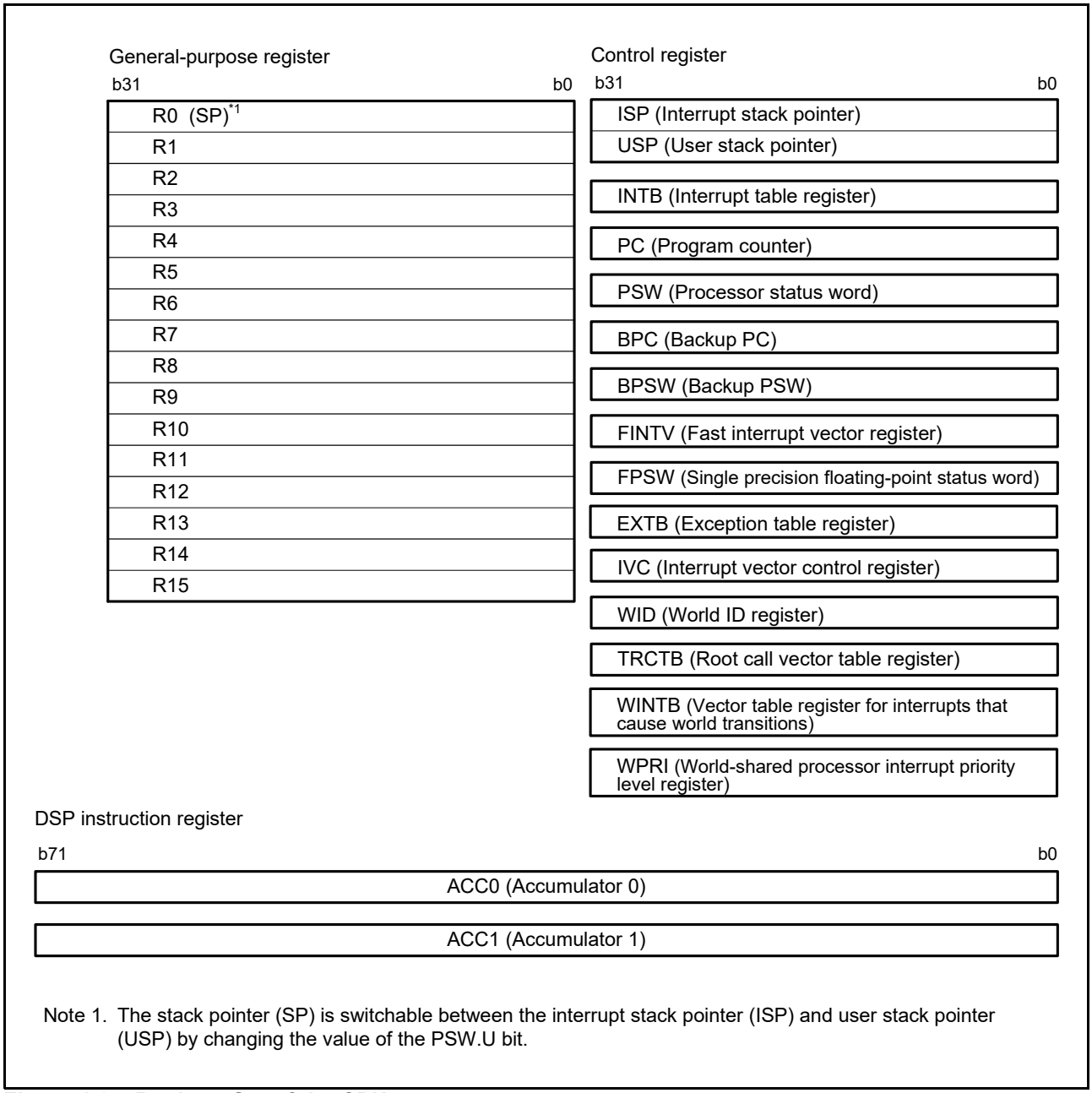
Programs such as those for the RXv3 core can also be run without taking the concept of the Protect Zone worlds into account. Such software projects are called Flat-Projects. In terms of the Protect Zone worlds, a Flat-Project is only run in the root world so that awareness of the other worlds is not required.

Each interrupt request that is input to the CPU has information about which world is its destination (destination world). Interrupt priority levels 1 to 31 can be used for interrupts having root or secure as the destination world. Interrupt priority levels 1 to 15 can be used for interrupts having non-secure as the destination world.

Fast interrupts are only usable in Flat-Projects.

1.3 Register Set of the CPU

The RXv3e CPU has sixteen general-purpose registers, fifteen control registers, and two accumulator used for DSP instructions.



Note 1. The stack pointer (SP) is switchable between the interrupt stack pointer (ISP) and user stack pointer (USP) by changing the value of the PSW.U bit.

Figure 1.2 Register Set of the CPU

1.3.1 General-Purpose Registers (R0 to R15)

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

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

1.3.2 Control Registers

This CPU has the following fifteen control registers.

- Interrupt stack pointer (ISP)*1
- User stack pointer (USP)
- Interrupt table register (INTB)*1
- Program counter (PC)
- Processor status word (PSW)
- Backup PC (BPC)*1
- Backup PSW (BPSW)*1
- Fast interrupt vector register (FINTV)
- Single-precision floating-point status word (FPSW)
- Exception table register (EXTB)*1
- Interrupt vector control register (IVC)*1
- World ID register (WID)
- Root call vector table register (TRCTB)
- Vector table register for interrupts that cause world transitions (WINTB)
- World-shared processor interrupt priority level register (WPRI)

Note: 1. This control register is actually implemented as three registers to support the Protect Zone function. Where this is required, the three registers are distinguished by postfixes appended to the register names, specifically _TR for the root world register, _S for the secure world register, and _NS for the non-secure world register. When a _TR, _S, or _NS suffix is not appended to a register name, the register for the current world is indicated unless otherwise specified.

1.3.2.1 Restrictions on Access to Control Registers

Table 1.1 lists the restrictions on access to each of the control registers.

The access restrictions listed here are for privileges and worlds in which at least one field in the control register is accessible. For each control register, different access restrictions may be set for each field. For details, refer to the descriptions of the individual control registers.

For control registers that are actually implemented as physically different registers for the root, secure, and non-secure worlds, registers for other worlds are not accessible by the MVTC, MVFC, PUSHC, and POPC instructions. However, registers for other worlds are accessible by the TRMVTC and TRMVFC instructions, which are only for use in the root world.

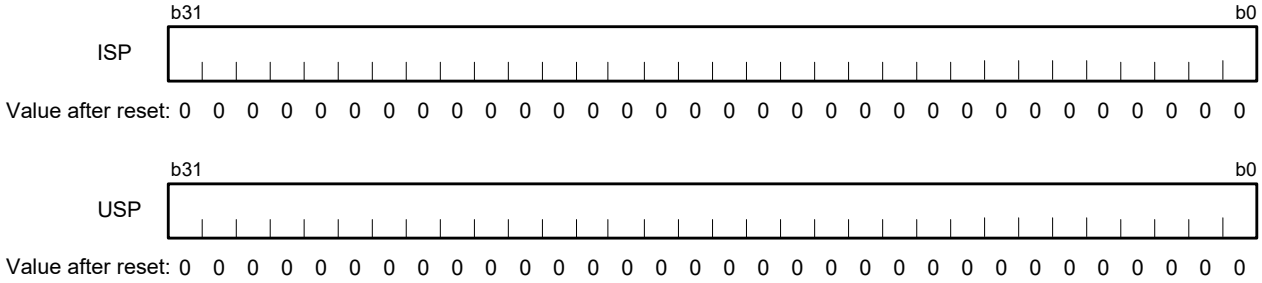
Access to the control registers is only possible when it is allowed under the restrictions on access in terms of the combination of the instruction group for the instruction, source world for access, and processor mode as listed in the table below. If the conditions for write access are not met, writing does not proceed. If the conditions for read access are not met, 0 is read. However, if the current world is other than root or the processor mode is user mode, an exception will occur in the case of an attempt to execute the TRMVTC or TRMVFC instruction.

Table 1.1 Restrictions on Access to Control Registers

Control Register	CR	Instruction, Source World for Access, and Processor Mode						
		Group 1 Instruction	Group 2 Instruction					
		Root	Root		Secure		Non-Secure	
		Supervisor	Supervisor	User	Supervisor	User	Supervisor	User
PSW	0	—	○	○	○	○	○	○
PC	1	—	R	R	R	R	R	R
USP	2	—	○	○	○	○	○	○
FPSW	3	—	○	○	○	○	○	○
IVC_TR	5	○	○	R	—	—	—	—
IVC_S		○	—	—	R	R	—	—
IVC_NS		○	—	—	—	—	R	R
BPSW_TR	8	○	○	R	—	—	—	—
BPSW_S		○	—	—	○	R	—	—
BPSW_NS		○	—	—	—	—	○	R
BPC_TR	9	○	○	R	—	—	—	—
BPC_S		○	—	—	○	R	—	—
BPC_NS		○	—	—	—	—	○	R
ISP_TR	10	○	○	R	—	—	—	—
ISP_S		○	—	—	○	R	—	—
ISP_NS		○	—	—	—	—	○	R
FINTV	11	—	○	R	—	—	—	—
INTB_TR	12	○	○	R	—	—	—	—
INTB_S		○	—	—	○	R	—	—
INTB_NS		○	—	—	—	—	○	R
EXTB_TR	13	○	○	R	—	—	—	—
EXTB_S		○	—	—	○	R	—	—
EXTB_NS		○	—	—	—	—	○	R
WPRI	14	—	○	R	○	R	—	—
TRCTB	14	○	—	—	—	—	—	—
WID	15	—	○	R	R	R	R	R
WINTB	15	○	—	—	—	—	—	—

The CR column is the value of the cr field when the register is accessed by an MVTC, MVFC, TRMVTC, TRMVFC, PUSHC, or POPC instruction.
Group 1 instruction: TRMVTC and TRMVFC instructions
Group 2 instruction: MVTC, MVFC, PUSHC, and POPC instructions
○: Both reading and writing are possible.
R: Reading is possible but writing is not.
—: Not accessible

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

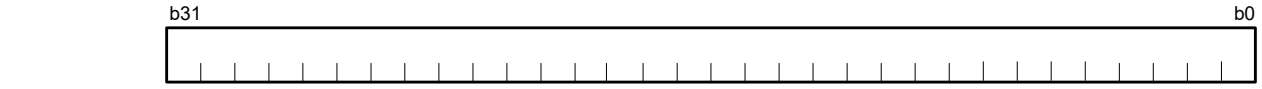


The stack pointer (SP) can be either of two types, the interrupt stack pointer (ISP) or the user stack pointer (USP). Whether the stack pointer operates as the ISP or USP depends on the value of the stack pointer select bit (U) in the processor status word (PSW).

The ISP is actually implemented as the following three registers to support the Protect Zone function.

- For the root world: ISP_TR (only accessible in the root world)
- For the secure world: ISP_S (only accessible in the secure world or root world)
- For the non-secure world: ISP_NS (only accessible in the non-secure world or root world)

1.3.2.3 Interrupt Table Register (INTB)

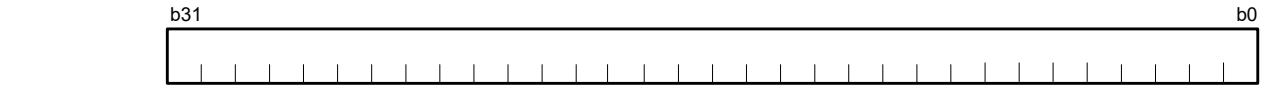


Value after reset: Undefined

The interrupt table register (INTB) specifies the address where the interrupt vector table starts. This register is actually implemented as the following three registers to support the Protect Zone function.

- For the root world: INTB_TR (only accessible in the root world)
- For the secure world: INTB_S (only accessible in the secure world or root world)
- For the non-secure world: INTB_NS (only accessible in the non-secure world or root world)

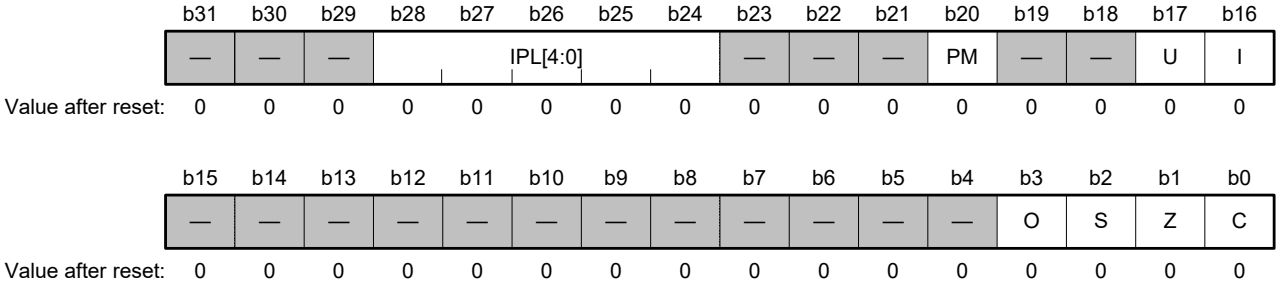
1.3.2.4 Program Counter (PC)



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

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

1.3.2.5 Processor Status Word (PSW)



Bit	Symbol	Bit Name	Description	R/W
b0	C	Carry flag	0: No carry has occurred. 1: A carry has occurred.	R/W
b1	Z	Zero flag	0: Result is non-zero. 1: Result is 0.	R/W
b2	S	Sign flag	0: Result is a positive value or 0. 1: Result is a negative value.	R/W
b3	O	Overflow flag	0: No overflow has occurred. 1: An overflow has occurred.	R/W
b15 to b4	—	Reserved	These bits are read as 0. The write value should be 0.	R/W
b16	I*1	Interrupt enable bit	0: See Table 1.2 1: Interrupt enabled.	R/W
b17	U*1	Stack pointer select bit	0: Interrupt stack pointer (ISP) is selected. 1: User stack pointer (USP) is selected.	R/W
b19, b18	—	Reserved	These bits are read as 0. The write value should be 0.	R/W
b20	PM*1, *2, *3	Processor mode select bit	0: Supervisor mode is selected. 1: User mode is selected.	R/W
b23 to b21	—	Reserved	These bits are read as 0. The write value should be 0.	R/W

Bit	Symbol	Bit Name	Description	R/W
b28 to b24	IPL[4:0] *1, *4	Processor interrupt priority level	b28 b24 0 0 0 0 0: Priority level 0 (lowest) 0 0 0 0 1: Priority level 1 0 0 0 1 0: Priority level 2 0 0 0 1 1: Priority level 3 0 0 1 0 0: Priority level 4 0 0 1 0 1: Priority level 5 0 0 1 1 0: Priority level 6 0 0 1 1 1: Priority level 7 0 1 0 0 0: Priority level 8 0 1 0 0 1: Priority level 9 0 1 0 1 0: Priority level 10 0 1 0 1 1: Priority level 11 0 1 1 0 0: Priority level 12 0 1 1 0 1: Priority level 13 0 1 1 1 0: Priority level 14 0 1 1 1 1: Priority level 15 (non-secure, highest) 1 0 0 0 0: Priority level 16 1 0 0 0 1: Priority level 17 1 0 0 1 0: Priority level 18 1 0 0 1 1: Priority level 19 1 0 1 0 0: Priority level 20 1 0 1 0 1: Priority level 21 1 0 1 1 0: Priority level 22 1 0 1 1 1: Priority level 23 1 1 0 0 0: Priority level 24 1 1 0 0 1: Priority level 25 1 1 0 1 0: Priority level 26 1 1 0 1 1: Priority level 27 1 1 1 0 0: Priority level 28 1 1 1 0 1: Priority level 29 1 1 1 1 0: Priority level 30 1 1 1 1 1: Priority level 31 (highest)	R/W
b31 to b29	—	Reserved	These bits are read as 0. The write value should be 0.	R/W

Note: 1. In user mode, writing to the IPL[4:0], PM, U, and I bits by an MVTC or POPC instruction is ignored. Writing to the IPL[4:0] bits by an MVTIPL instruction generates a privileged instruction exception.

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

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

Note: 4. Writing to the IPL[4:0] bits in response to an MVTC, POPC, or MVTIPL instruction in the non-secure world always causes clearing of the IPL[4] bit (to 0).
Executing an RTE or RTFI instruction in the non-secure world also always causes clearing of the IPL[4] bit (to 0).

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

C flag (Carry flag)

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

Z flag (Zero flag)

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

S flag (Sign flag)

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

O flag (Overflow flag)

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

I bit (Interrupt enable bit)

This bit enables interrupt requests. When an exception is accepted, the value of this bit becomes 0. Even if the value of this bit is 0, an interrupt request will only be accepted in the non-secure world when the conditions of both the setting of the NSIC bit in the interrupt vector control register (IVC) and the priority level of the requested interrupt are met as listed in Table 1.2.

Table 1.2 Conditions of Interrupt Acceptance

Current World	I bit	IVC.NSIC	IPR	Enabling or Disabling of Interrupts
Root or secure	0	—	—	Disabled
	1	—	—	Enabled
Non-secure	0	0	—	Disabled
		1	Less than 16	Disabled
			16 or greater	Enabled
	1	—	—	Enabled

IPR: The priority level of a requested interrupt

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.

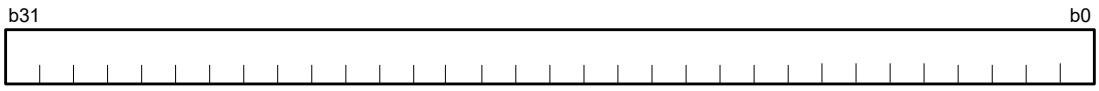
IPL[4:0] bits (Processor interrupt priority level)

The IPL[4:0] bits specify the processor interrupt priority level in the root world or secure world as one of 32 levels from 0 to 31, wherein priority level 0 is the lowest and priority level 31 the highest. In the non-secure world, the level is specified as

one of 16 levels from 0 to 15, wherein level 0 is the lowest and level 15 the highest. Priority level 16 or a higher level cannot be specified in the non-secure world.

When the priority level of a requested interrupt is higher than the processor interrupt priority level specified in these bits, the interrupt is enabled. Setting the IPL[4:0] bits to level 31 (1Fh) in the root world or secure world disables all interrupt requests. In the non-secure world, setting the IPL[4:0] bits to level 15 (0Fh) disables all interrupt requests for the non-secure world. The IPL[4:0] bits are set to level 31 (1Fh) when a non-maskable interrupt for the root world or secure world is generated. The IPL[4:0] bits are set to level 15 (0Fh) when a non-maskable interrupt for the non-secure world is generated while execution is in progress in the non-secure world. When interrupts in general are generated, the bits are set to the priority levels of accepted interrupts.

1.3.2.6 Backup PC (BPC)

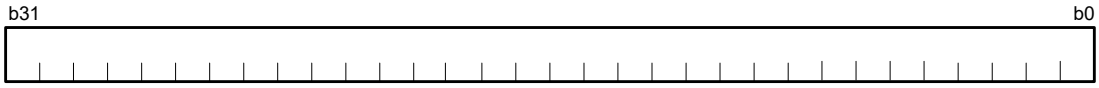


Value after reset: Undefined

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. This register is actually implemented as the following three registers to support the Protect Zone function.

- For the root world: BPC_TR (only accessible in the root world)
- For the secure world: BPC_S (only accessible in the secure world or root world)
- For the non-secure world: BPC_NS (only accessible in the non-secure world or root world)

1.3.2.7 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. This register is actually implemented as the following three registers to support the Protect Zone function.

- For the root world: BPSW_TR (only accessible in the root world)
- For the secure world: BPSW_S (only accessible in the secure world or root world)
- For the non-secure world: BPSW_NS (only accessible in the non-secure world or root world)

In the case of an attempt to write a value to bit 28 in BPSW_NS, 0 is written to the bit.

1.3.2.8 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. This register is only accessible in the root world.

1.3.2.9 Single-Precision 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]	Single-precision floating-point rounding-mode setting bits	b1 b0 0 0: Round to the nearest value 0 1: Round towards 0 1 0: Round towards +∞ 1 1: Round towards –∞	R/W
b2	CV	Invalid operation cause flag	0: No invalid operation has been encountered. 1: Invalid operation has been encountered.	R/(W)*1
b3	CO	Overflow cause flag	0: No overflow has occurred. 1: Overflow has occurred.	R/(W)*1
b4	CZ	Division-by-zero cause flag	0: No division-by-zero has occurred. 1: Division-by-zero has occurred.	R/(W)*1
b5	CU	Underflow cause flag	0: No underflow has occurred. 1: Underflow has occurred.	R/(W)*1
b6	CX	Inexact cause flag	0: No inexact exception has been generated. 1: Inexact exception has been generated.	R/(W)*1
b7	CE	Unimplemented processing cause flag	0: No unimplemented processing has been encountered. 1: Unimplemented processing has been encountered.	R/(W)*1
b8	DN	0 flush bit of denormalized number	0: A denormalized number is handled as a denormalized number. 1: A denormalized number is handled as 0.*2	R/W
b9	—	Reserved	This bit is read as 0. The write value should be 0.	R/W
b10	EV	Invalid operation exception enable bit	0: Invalid operation exception is masked. 1: Invalid operation exception is enabled.	R/W
b11	EO	Overflow exception enable bit	0: Overflow exception is masked. 1: Overflow exception is enabled.	R/W
b12	EZ	Division-by-zero exception enable bit	0: Division-by-zero exception is masked. 1: Division-by-zero exception is enabled.	R/W
b13	EU	Underflow exception enable bit	0: Underflow exception is masked. 1: Underflow exception is enabled.	R/W
b14	EX	Inexact exception enable bit	0: Inexact exception is masked. 1: Inexact exception is enabled.	R/W
b25 to b15	—	Reserved	These bits are read as 0. The write value should be 0.	R/W
b26	FV*3	Invalid operation flag	0: No invalid operation has been encountered. 1: Invalid operation has been encountered.*8	R/W
b27	FO*4	Overflow flag	0: No overflow has occurred. 1: Overflow has occurred.*8	R/W
b28	FZ*5	Division-by-zero flag	0: No division-by-zero has occurred. 1: Division-by-zero has occurred.*8	R/W

Bit	Symbol	Bit Name	Description	R/W
b29	FU[*6]	Underflow flag	0: No underflow has occurred. 1: Underflow has occurred.[*8]	R/W
b30	FX[*7]	Inexact flag	0: No inexact exception has been generated. 1: Inexact exception has been generated.[*8]	R/W
b31	FS	Single-precision floating-point error summary flag	This bit reflects the logical OR of the FU, FZ, FO, and FV flags.	R

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

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

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

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

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

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

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

Note: 8. Once the bit has been set to 1, this value is retained until it is cleared to 0 by software.

The single-precision floating-point status word (FPSW) indicates the results of single-precision floating-point arithmetic operations.

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

RM[1:0] bits (Single-precision floating-point rounding-mode setting bits)

These bits specify the single-precision floating-point rounding-mode.

Explanation of Single-Precision Floating-Point Rounding Modes

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

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

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

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

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

- If an exception or processing that is not implemented is not encountered in the execution of a single-precision floating-point arithmetic instruction, the corresponding flags become 0.
- When 0 is written to the bit by the MVTC and POPC instructions, the bit is set to 0; the bit retains the previous value when 1 is written by the instruction.

DN bit (0 flush bit of denormalized number)

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

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

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

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

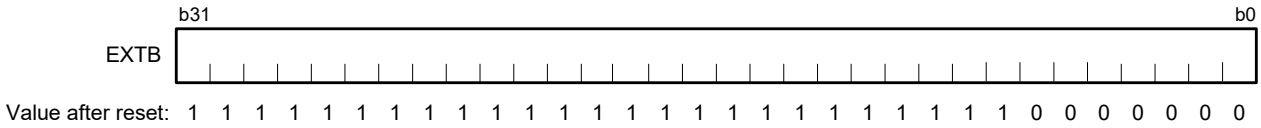
While the exception handling enable bit (Ej) is 0 (exception handling is masked), if any of five single-precision floating-point exceptions specified in the IEEE754 standard is generated, the corresponding bit is set to 1.

- 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 (Single-precision floating-point error summary flag)

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

1.3.2.10 Exception Vector Table Register (EXTB)



The exception table register (EXTB) specifies the address where the exception vector table starts. This register is actually implemented as the following three registers to support the Protect Zone function.

- For the root world: EXTB_TR (only accessible in the root world)
- For the secure world: EXTB_S (only accessible in the secure world or root world)
- For the non-secure world: EXTB_NS (only accessible in the non-secure world or root world)

The initial value depends on the world.

- EXTB_TR, EXTB_S: FFFF FF80h
- EXTB_NS: 7FFF FF80h

1.3.2.11 Interrupt Vector Control Register (IVC)

	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	EXCSE	—	—	—	NSIC	—	—	—	—	—	—	UIR[1:0]	
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
	—	—	—	—	—	—	—	—	BWINM[7:0]							
Value after reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Symbol	Bit Name	Description	R/W
b7 to b0	BWINM[7:0]	Backup World Transition Interrupt Source Number	These bits hold the source number of an interrupt accepted through the world-transition acceptance method.	R/W
b15 to b8	—	Reserved	These bits are read as 0. The write value should be 0.	R/W
b17, b16	UIR[1:0]	Software Interrupt Restriction	These bits control the range of interrupt numbers that can be specified as the operand in an INT instruction to be executed in user mode. A number outside this range being specified as the operand will cause a privileged instruction exception. In supervisor mode, a number from 0 to 255 can be specified as the operand regardless of the setting in these bits. • For INT #imm:8 b17 b16 0 0: A number from 0 to 255 can be specified. 0 1: A number from 0 to 15 can be specified. 1 x: Setting prohibited • For INT Rs b17 b16 0 0: A number from 0 to 255 can be specified. 0 1: Software interrupts cannot be executed. 1 x: Setting prohibited	R/W
b23 to b18	—	Reserved	These bits are read as 0. The write value should be 0.	R/W
b24	NSIC	Non-secure Interrupt Enable Control	This bit controls the function of the PSW.I bit in the non-secure world. 0: Interrupts are disabled while the PSW.I bit is 0. 1: Interrupts up to priority level 15 are disabled while the PSW.I bit is 0.	R/W*1
b27 to b25	—	Reserved	These bits are read as 0. The write value should be 0.	R/W
b28	EXCSE	Secure Exception Detection during Hardware Pre-processing	This bit is set if a secure exception occurs during hardware pre-processing. 0: A secure exception has not occurred during hardware pre-processing. 1: A secure exception has occurred during hardware preprocessing.	R/W*1
b31 to b29	—	Reserved	These bits are read as 0. The write value should be 0.	R/W

Note: 1. In the IVC_NS and IVC_S registers, this bit is always read as 0.

This register is actually implemented as the following three registers to support the Protect Zone function.

- For the root world: IVC_TR (only accessible in the root world)
- For the secure world: IVC_S (only accessible in the secure world or root world)
- For the non-secure world: IVC_NS (only accessible in the non-secure world or root world)

These actual registers are only modifiable during execution in supervisor mode while in the root world.

BWINM[7:0] bits (Backup World Transition Interrupt Source Number)

When an interrupt accepted through the world-transition acceptance method is generated, the vector number in the interrupt vector table is stored in the IVC_TR.BWINM[7:0] bits and in the IVC.BWINM[7:0] bits in the destination world of the interrupt. The same value is written to the BWINM[7:0] bits of both the IVC_TR register and IVC register in the destination world for the transition (that is, the IVC_S.BWINM[7:0] bits in the secure world or the IVC_NS.BWINM[7:0] bits in the non-secure world).

UIR[1:0] bits (Software Interrupt Restriction)

These bits are used to restrict the interrupt numbers that can be specified as the operand in an INT instruction to be executed in user mode. This setting is ignored in supervisor mode.

NSIC bit (Non-secure Interrupt Enable Control)

This bit is used to control the function of the PSW.I bit for interrupts with a priority level 16 or higher in the non-secure world. When the NSIC bit is set to 1, interrupts with a priority level of 16 or higher are accepted regardless of the setting of the PSW.I bit. Interrupts with a priority level up to 15 are only accepted while the setting of the PSW.I bit is 1.

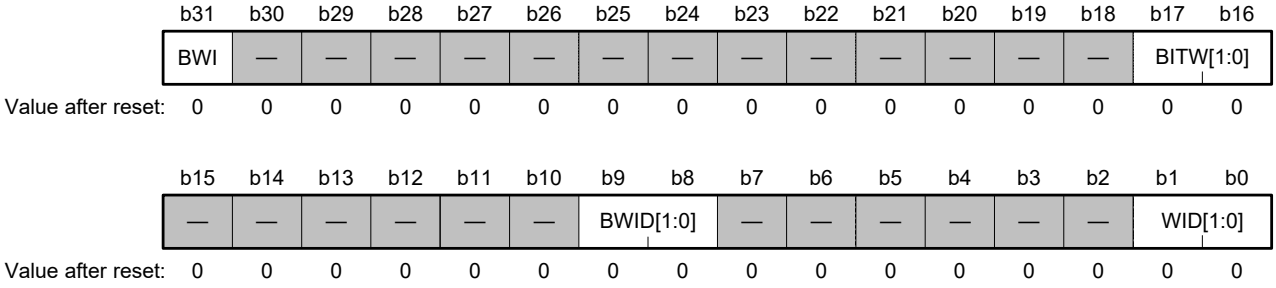
When the NSIC bit is set to 0, interrupts, whatever their priority level, are only accepted while the PSW.I bit is 1.

EXCSE bit (Secure Exception Detection during Hardware Pre-processing)

This bit indicates that a secure exception has been detected during hardware pre-processing.

If a secure exception occurs during hardware pre-processing, the pre-processing is aborted and the secure exception is accepted. In this case, execution cannot be returned from the processing of the secure exception by using the TRRTE instruction. This bit can be used to identify such situations.

1.3.2.12 World ID Register (WID)



Bit	Symbol	Bit Name	Description	R/W
b1, b0	WID[1:0]	World ID	These bits hold the ID of the current world.	R
b7 to b2	—	Reserved	These bits are read as 0. The write value should be 0.	R/W
b9, b8	BWID[1:0]	Backup World ID	These bits hold the world ID that was saved as the backup in a transition to the root world.	R/W*1
b15 to b10	—	Reserved	These bits are read as 0. The write value should be 0.	R/W
b17, b16	BITW[1:0]	Backup Destination World ID	These bits hold the destination world ID for a backup interrupt (with world-transition acceptance) or non-maskable interrupt (with world-transition acceptance).	R
b30 to b18	—	Reserved	These bits are read as 0. The write value should be 0.	R/W
b31	BWI	Backup Interrupt Enable for Source World of Transition	This bit holds the value of the interrupt enable bit (PSW.I) specified in the source world of a transition for an unconditional trap (with world-transition acceptance).	R

Note: 1. The bit is read as 0 during execution in the secure world or non-secure world.

The following world IDs are defined.

- 00: Root world
- 01: Secure world
- 10: Non-secure world

This register is only modifiable during execution in supervisor mode while in the root world. It is read-only during execution in the secure world or non-secure world.

1.3.2.13 Root Call Vector Table Register (TRCTB)



	b31																												b0
Value after reset:	1	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

Use the root call vector table register (TRCTB) to specify the first address of the root call vector table to be used for unconditional traps accepted through the world-transition acceptance method. The initial value of this register is 8000 0000h. This register is only accessible during execution in supervisor mode while in the root world.

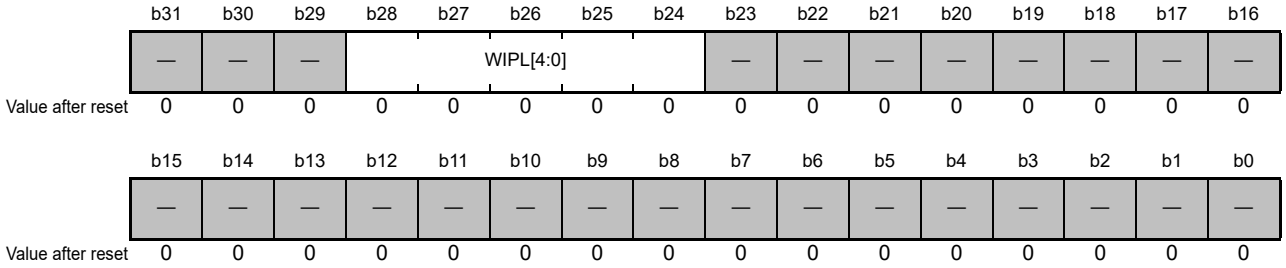
1.3.2.14 Vector Table Register for Interrupts That Cause World Transitions (WINTB)



	b31																												b0
Value after reset:	1	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

Use the vector table register for interrupts that cause world transitions (WINTB) to specify the first address of the vector table for interrupts that cause world transitions to be used for interrupts accepted through the world-transition acceptance method. The initial value of this register is 8000 0000h. This register is only accessible during execution in supervisor mode while in the root world.

1.3.2.15 World-Shared Processor Interrupt Priority Level Register (WPRI)



Bit	Symbol	Bit Name	Description	R/W
b23 to b0	—	Reserved	These bits are read as 0. The write value should be 0.	R
b28 to b24	WIPL[4:0]	World-Shared Processor Interrupt Priority Level	b28 b24 0 0 0 0 0: Priority level 0 (lowest) 0 0 0 0 1: Priority level 1 0 0 0 1 0: Priority level 2 0 0 0 1 1: Priority level 3 0 0 1 0 0: Priority level 4 0 0 1 0 1: Priority level 5 0 0 1 1 0: Priority level 6 0 0 1 1 1: Priority level 7 0 1 0 0 0: Priority level 8 0 1 0 0 1: Priority level 9 0 1 0 1 0: Priority level 10 0 1 0 1 1: Priority level 11 0 1 1 0 0: Priority level 12 0 1 1 0 1: Priority level 13 0 1 1 1 0: Priority level 14 0 1 1 1 1: Priority level 15 (non-secure, highest) 1 0 0 0 0: Priority level 16 1 0 0 0 1: Priority level 17 1 0 0 1 0: Priority level 18 1 0 0 1 1: Priority level 19 1 0 1 0 0: Priority level 20 1 0 1 0 1: Priority level 21 1 0 1 1 0: Priority level 22 1 0 1 1 1: Priority level 23 1 1 0 0 0: Priority level 24 1 1 0 0 1: Priority level 25 1 1 0 1 0: Priority level 26 1 1 0 1 1: Priority level 27 1 1 1 0 0: Priority level 28 1 1 1 0 1: Priority level 29 1 1 1 1 0: Priority level 30 1 1 1 1 1: Priority level 31 (secure/root, highest)	R/W
b31 to b29	—	Reserved	These bits are read as 0. The write value should be 0.	R

This register is only readable and writable in the secure or root world.

This register is only writable in supervisor mode.

WIPL[4:0] bits (World-Shared Processor Interrupt Priority Level)

The WIPL[4:0] bits specify the world-shared processor interrupt priority level as one of 32 levels from 0 to 31, wherein priority level 0 is the lowest and priority level 31 the highest.

When the priority level of a requested interrupt is higher than the current processor interrupt priority level specified in PSW.IPL[4:0] and higher than the world-shared processor interrupt priority level specified in WPRI.WIPL[4:0], the interrupt is enabled.

1.3.3 Accumulator

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

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

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

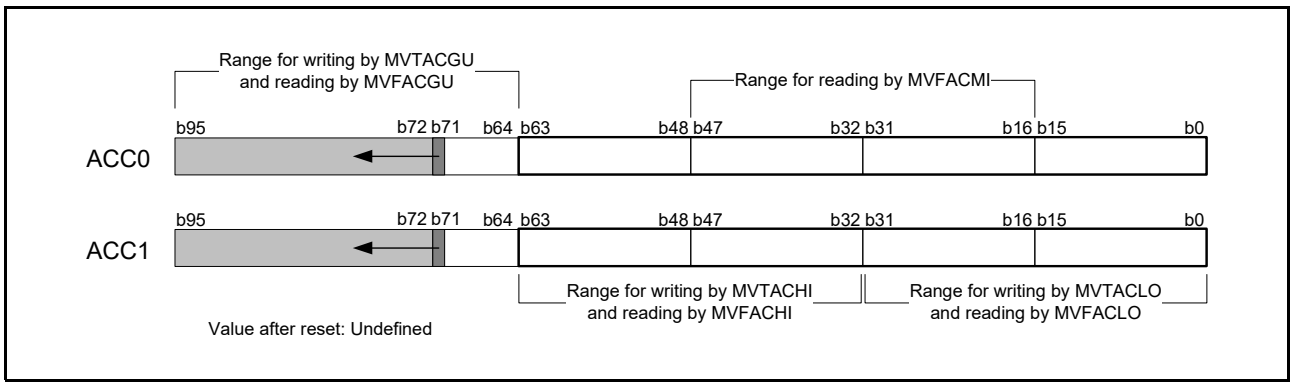


Figure 1.3 Accumulator

Note: When bits 95 to 72 of the accumulator are read, the value is the sign-extended value of bit 71. Writing to this area is ignored.

1.4 Single-Precision Floating-Point Exceptions

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

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

1.4.1 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 single-precision floating-point format. Table 1.3 lists the results of operations when an overflow exception occurs.

Table 1.3 Operation Results When an Overflow Exception Has Occurred

Single-Precision Floating-Point Rounding Mode	Sign of Result	Operation Result (Value in the Destination Register)	
		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.

1.4.2 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 single-precision floating-point format. (However, this does not apply when the result is 0.) Table 1.4 lists the results of operations when an underflow exception occurs.

Table 1.4 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.	

1.4.3 Inexact

An inexact exception occurs when the result of a hypothetical calculation with infinite precision differs from the actual result of the operation. Table 1.5 lists the conditions leading to an inexact exception and the results of operations.

Table 1.5 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 1.3, Operation Results When an Overflow Exception Has Occurred	No change
Rounding has been produced.	Value after rounding	

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

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

1.4.4 Division-by-Zero

Dividing a non-zero finite number by zero produces a division-by-zero exception. Table 1.6 lists the results of operations that have led to a division-by-zero exception. However, if the dividend is one of those listed in Table 1.7, the operation is not treated as division by zero.

Table 1.6 Operation Results When a Division-by Zero Exception Has Occurred

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

Table 1.7 Dividends and Operations that are not Treated as Division by Zero

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

1.4.5 Invalid Operation

Executing an invalid operation produces an invalid exception. Table 1.8 lists the conditions leading to an invalid exception and the results of operations.

Table 1.8 Conditions Leading to an Invalid Exception and the Operation Results

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

Legend

NaN (Not a Number): Not a Number

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

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

Table 1.9 lists the rules for generating QNaNs as the results of operations.

Table 1.9 Rules for Generating QNaNs

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

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

1.4.6 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.

1.5 Processor Mode

The RXv3e 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.

1.5.1 Supervisor Mode

In supervisor mode, all CPU resources that are allowed in the current world are accessible and all instructions that are allowed in the current world can be executed. However, writing to the processor mode select bit (PM) in the processor status word (PSW) by executing an MVTC or POPC instruction will be ignored. For details on how to write to the PM bit, refer to section 1.3.2.5, Processor Status Word (PSW).

1.5.2 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[4:0], 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)
- Interrupt vector control register (IVC)
- World ID register (WID)
- Root call vector table register (TRCTB)
- Vector table register for interrupts that cause world transitions (WINTB)
- World-shared processor interrupt priority level register (WPRI)

1.5.3 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, WAIT, TRCALL, TRMVFC, TRMVTC, TRRTE, SAVE, and RSTR instructions.

If the INT instruction is executed in user mode with a number outside the specified allowable range, a privileged instruction exception occurs. For details, see the description of the IVC.UIR bit.

1.5.4 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, TRMVTC, or POPC instruction is prohibited. Switch the processor mode by following the procedures described below.

(1) Switching from user mode to supervisor mode

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

(2) Switching from supervisor mode to user mode

Executing an RTE or TRRTE instruction when the value of the PM bit in the PSW that has been saved on the stack is “1” or an RTFI instruction when the value of the PM bit in the PSW that has been saved in the backup PSW (BPSW) is “1” causes a transition to user mode. In the transition to user mode, the value of the stack pointer designation bit (the U bit in the PSW) becomes “1”.

1.6 Data Types

The RXv3e CPU can handle four types of data: integer, single-precision floating-point number, bit, and string.

1.6.1 Integer

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

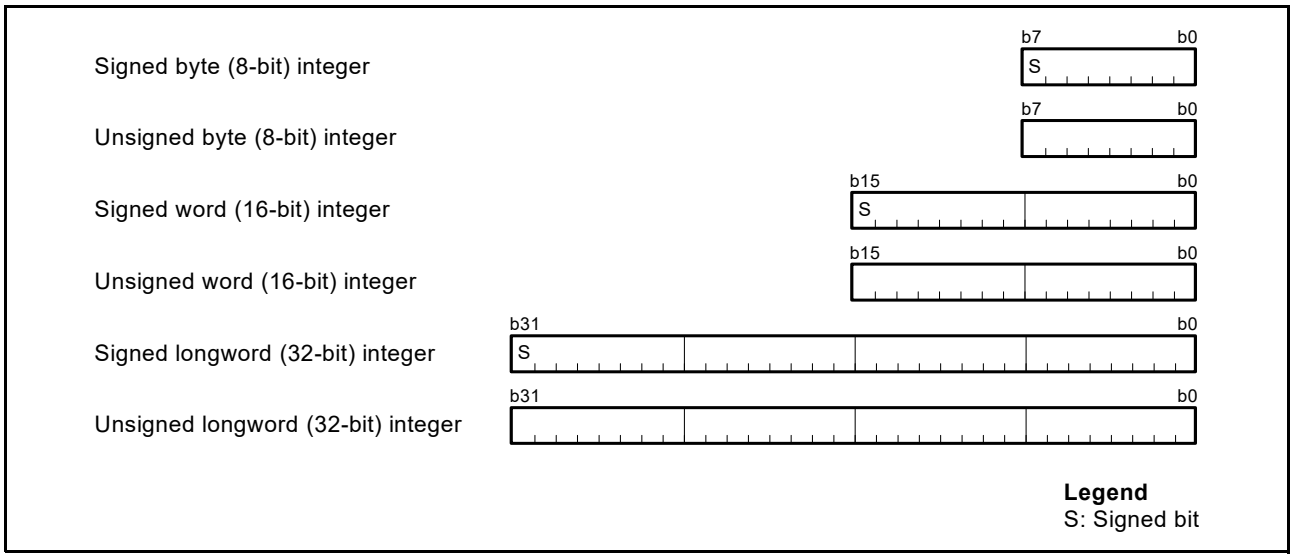


Figure 1.4 Integer

1.6.2 Single-Precision Floating-Point Number

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

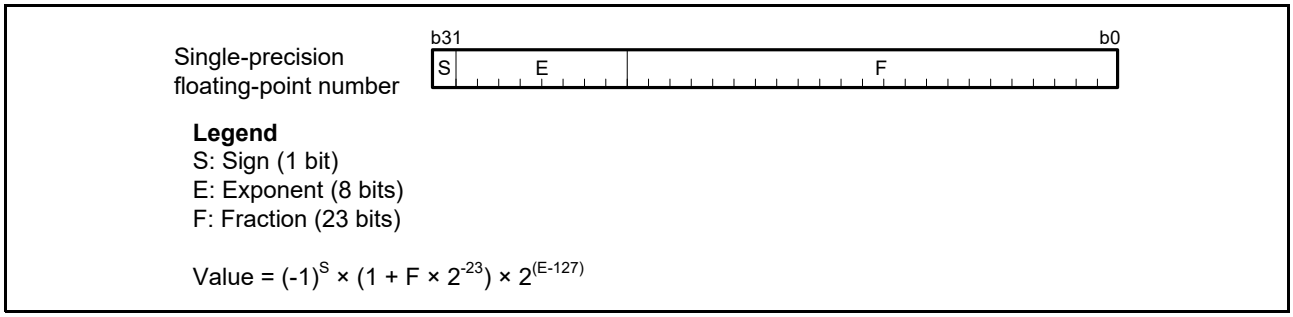


Figure 1.5 Single-Precision Floating-Point Number

The single-precision floating-point number can represent the values listed below.

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

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

1.6.3 Bit

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

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

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



Figure 1.6 Bit

1.6.4 String

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

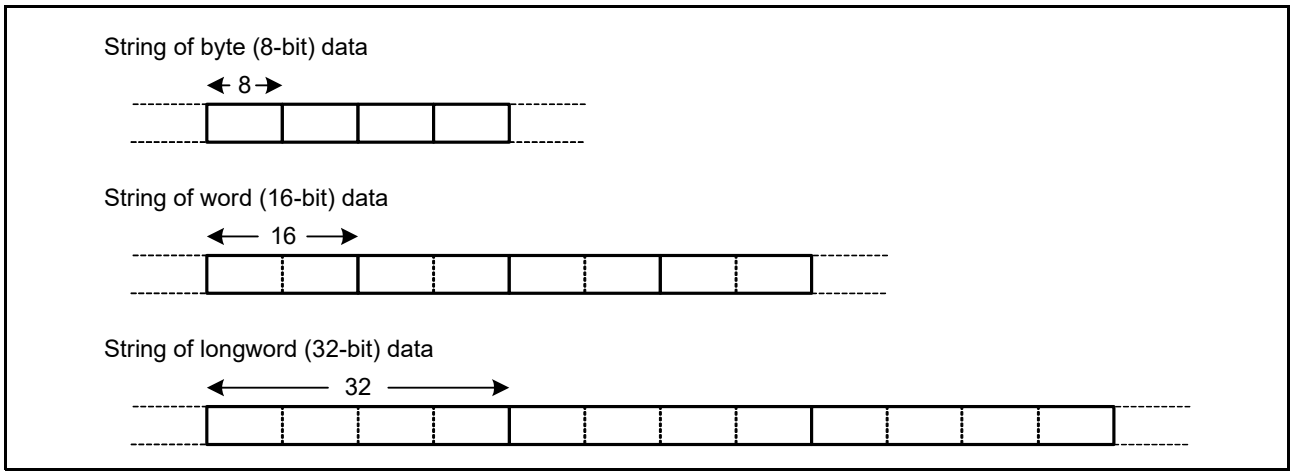


Figure 1.7 String

1.7 Data Arrangement

1.7.1 Data Arrangement in Registers

Figure 1.8 shows the relation between the sizes of registers and bit numbers.

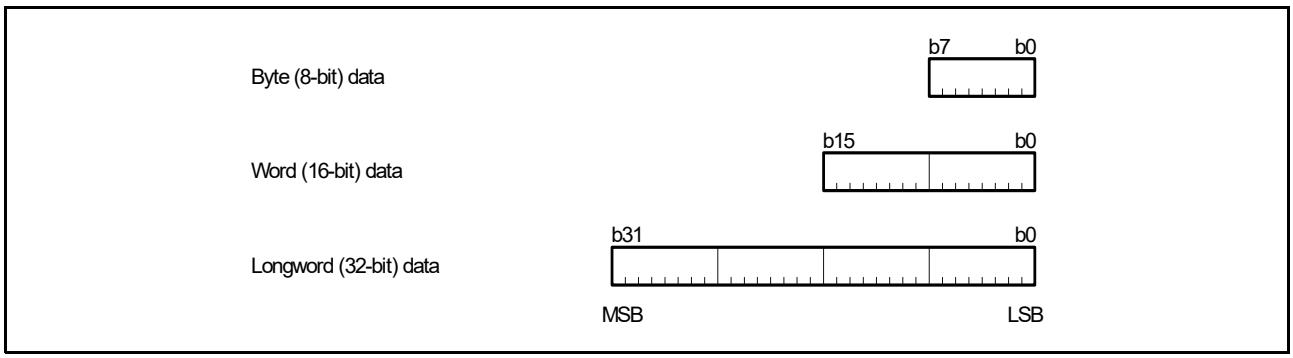


Figure 1.8 Data Arrangement in Registers

1.7.2 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 1.9 shows the arrangement of data in memory.

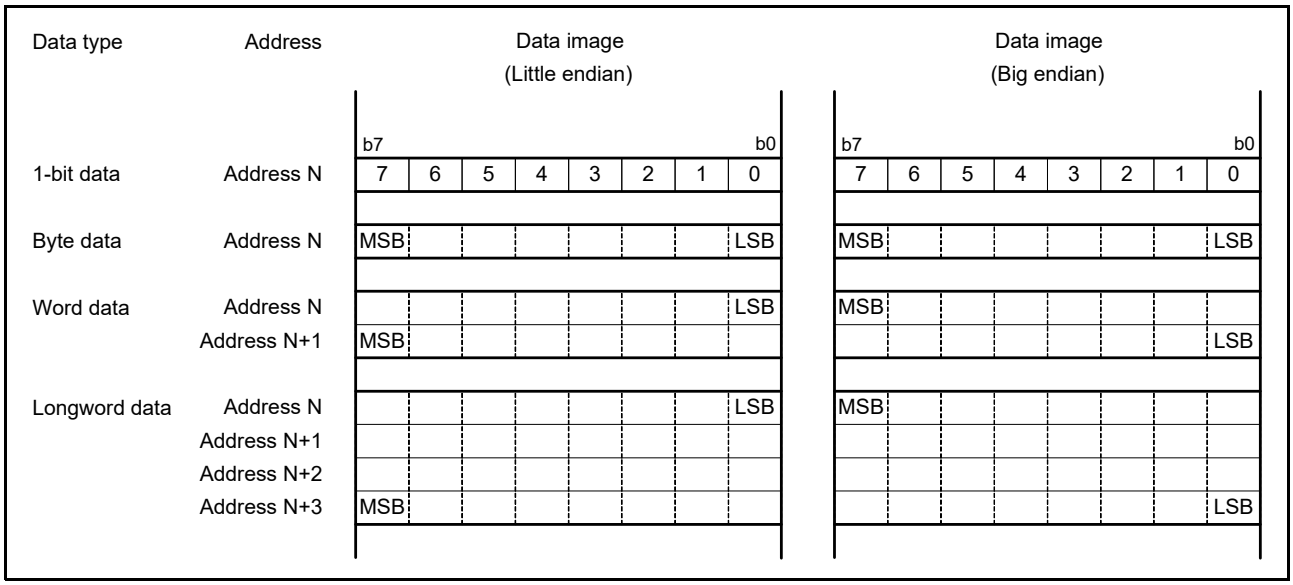


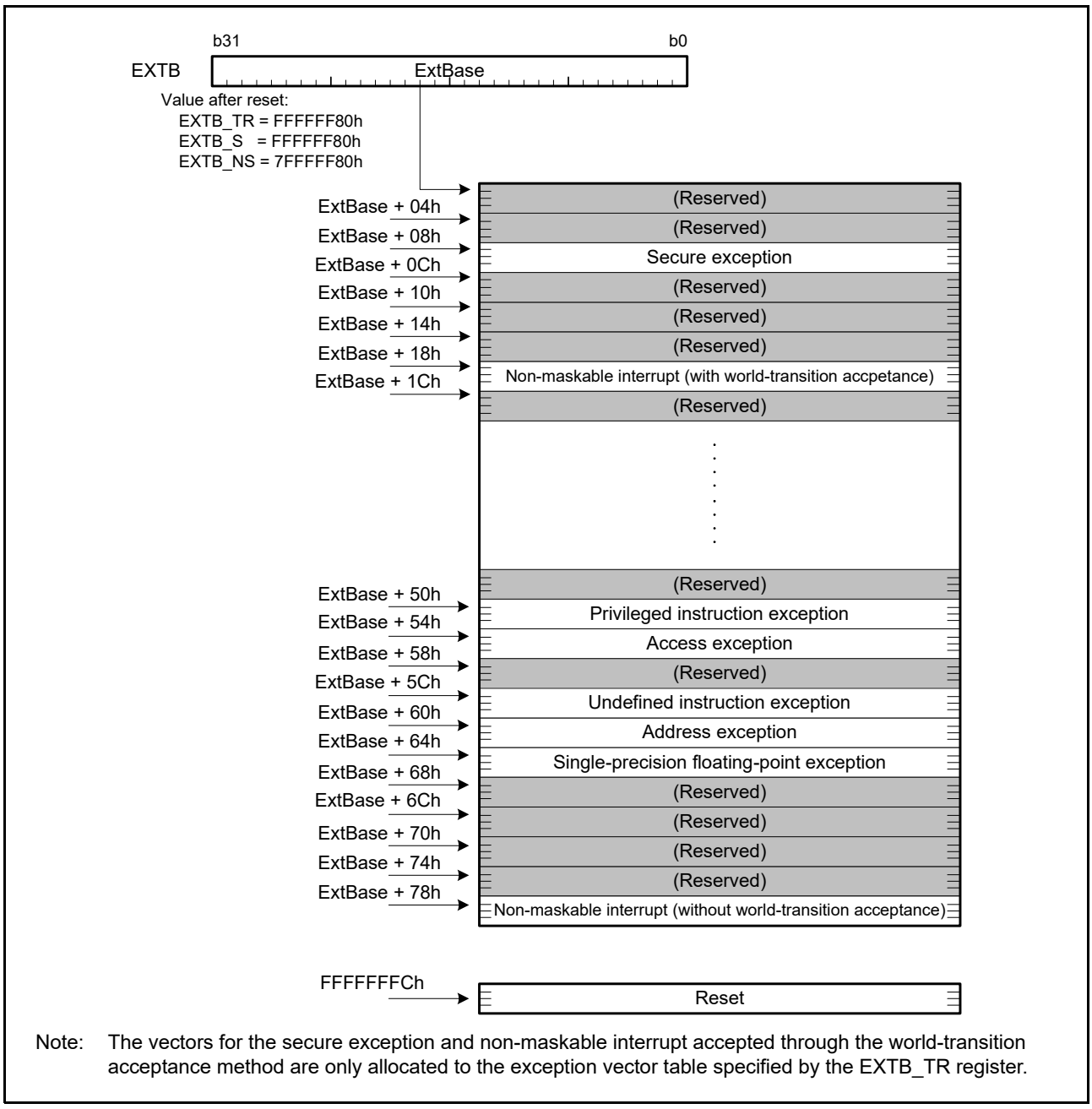
Figure 1.9 Data Arrangement in Memory

1.8 Vector Table

There are four types of vector tables: exception vector table, interrupt vector table, root call vector table, and vector table for interrupts that cause world transitions. Each vector in a vector table consists of four bytes and specifies the address where the corresponding exception handling routine starts.

1.8.1 Exception Vector Table

In the exception vector table, the individual vectors for the secure exceptions, non-maskable interrupt (with world-transition acceptance), privileged instruction exception, access exception, address exception, undefined instruction exception, single-precision floating-point exception, and non-maskable interrupt (with normal acceptance) are allocated to the 124-byte area where the value indicated by the exception table register (EXTB) is used as the starting address (ExtBase). Note, however, that the reset vector is always allocated to FFFFFFFCh. Figure 1.10 shows the exception vector table.



Note: The vectors for the secure exception and non-maskable interrupt accepted through the world-transition acceptance method are only allocated to the exception vector table specified by the EXTB_TR register.

Figure 1.10 Exception Vector Table

1.8.2 Interrupt Vector Table

The address where the interrupt vector table is placed can be adjusted. The table is a 1,024-byte region that contains all vectors for unconditional traps and interrupts and starts at the address (IntBase) specified in the interrupt table register (INTB). Figure 1.11 shows the interrupt vector table.

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

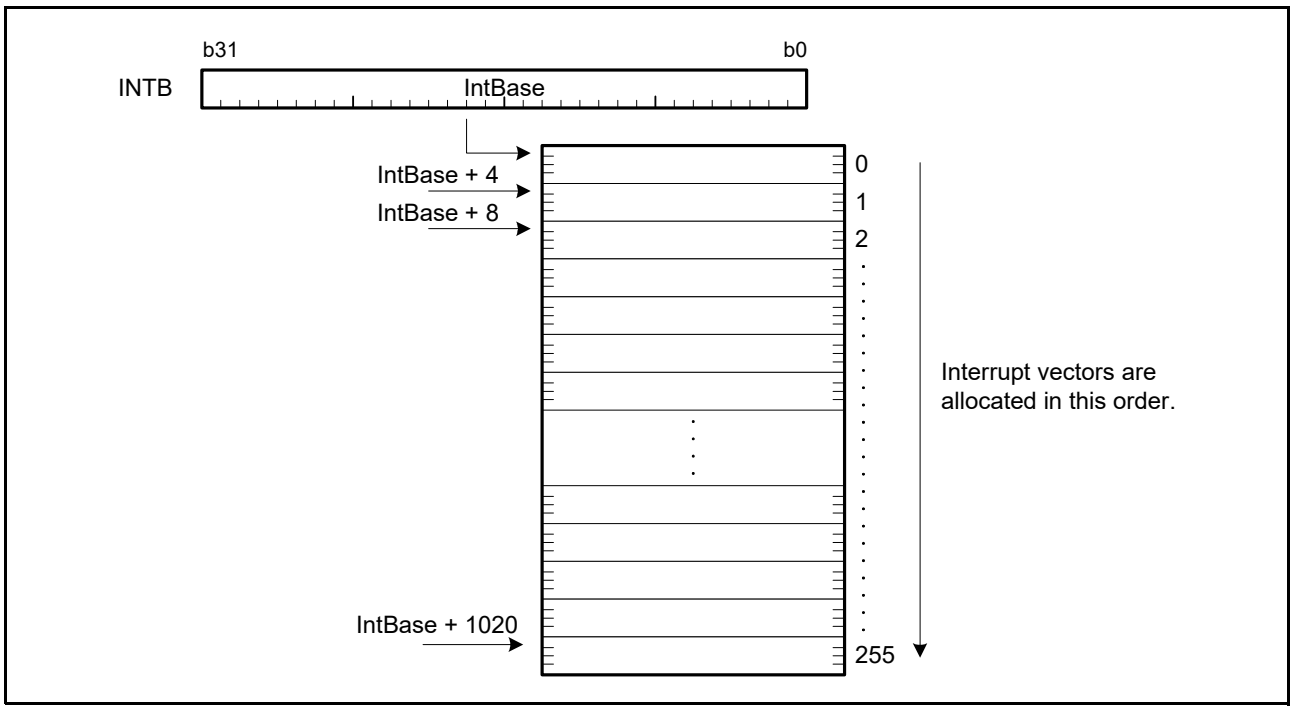


Figure 1.11 Interrupt Vector Table

1.8.3 Root Call Vector Table

The root call vector table is a 1024-byte region starting at the address (TRCallBase) specified in the root call vector table register (TRCTB). The individual vectors for the unconditional traps accepted through the world-transition acceptance method are allocated to this vector table. Figure 1.12 shows the root call vector table.

Each vector in the root call vector table has a vector number from 0 to 255. Vector numbers 0 to 255 are assigned for the TRCALL instruction and the vector corresponding to the number specified in the operand of the instruction is read from the vector table.

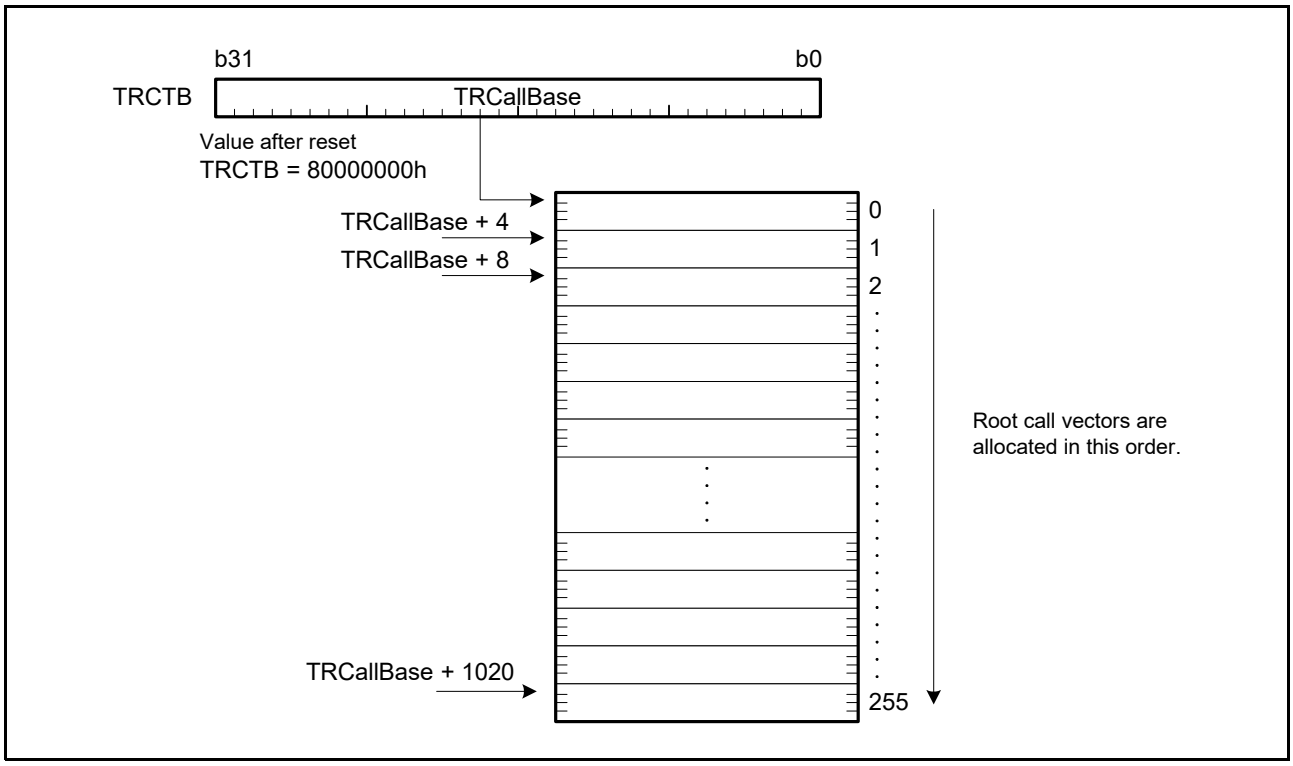


Figure 1.12 Root Call Vector Table

1.8.4 Vector Table for Interrupts that Cause World Transitions

The vector table for interrupts that cause world transitions is a 12-byte region starting at the address (WIntBase) specified in the vector table register for interrupts that cause world transitions (WINTB). The individual vectors for the interrupts accepted through the world-transition acceptance method are allocated to this vector table. Figure 1.13 shows the vector table for interrupts that cause world transitions.

Each vector in this vector table has a vector number from 0 to 2. The vector number corresponds to the ID of the destination world for the interrupt accepted through the world-transition acceptance method.

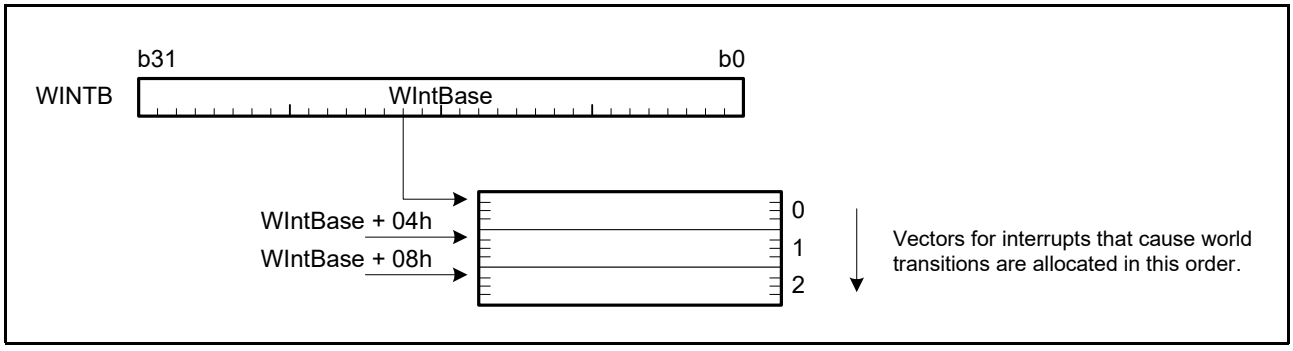


Figure 1.13 Vector Table for Interrupts that Cause World Transitions

1.9 Address Space

The address space of the RXv3e 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 RXv3e CPU is depicted in Figure 1.14. For all regions, the designation may differ with the product and operating mode. For details, refer to the user's manual: hardware for each product.

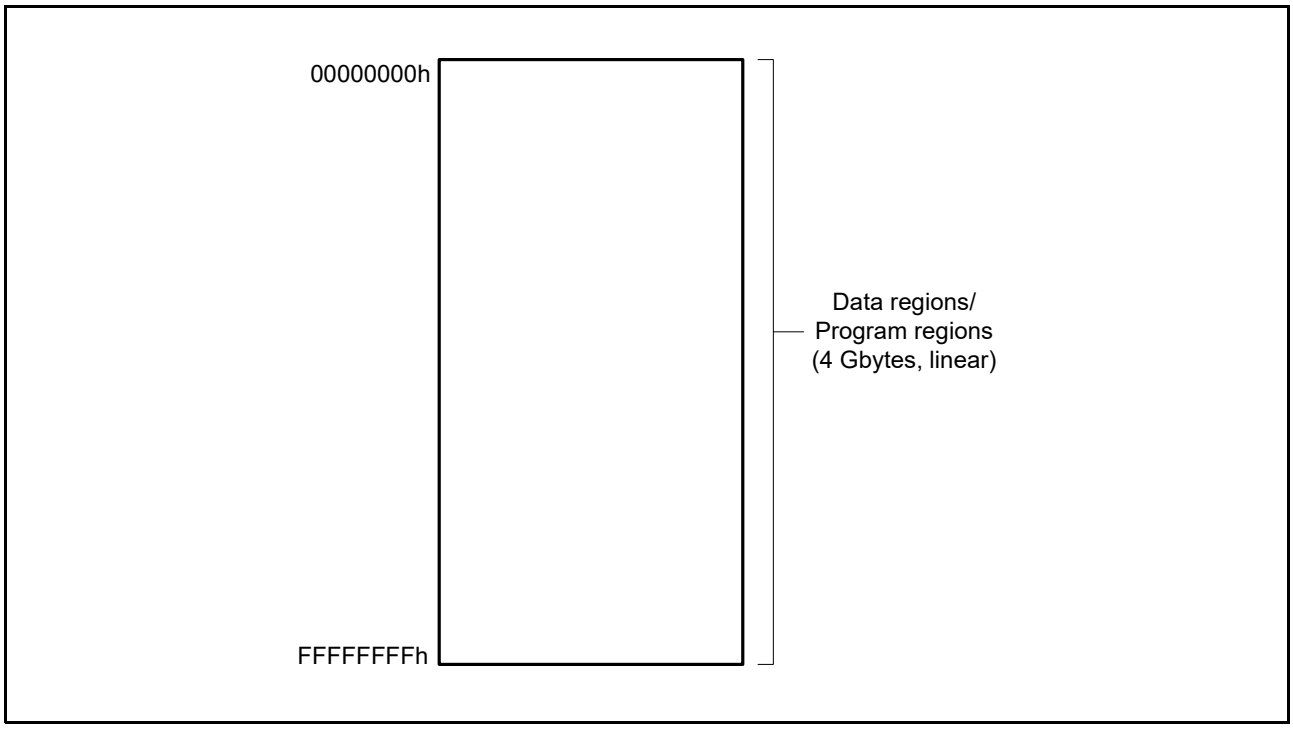


Figure 1.14 Address Space

1.10 Register Bank Save Function

The RXv3e CPU has dedicated register-saving banks and functionality for using them for the fast saving and restoring of the values of CPU registers (see Figure 1.15). The register-saving banks enable the fast collective saving at the start of the exception handling routine and fast collective restoring of register values at the end of the exception handling routine.

The register-saving banks are only accessible by the SAVE and RSTR instructions, and are independent of the 4-Gbyte address space. Each of the multiple banks is used to save and restore the values of the following CPU registers: all general purpose registers except R0, the USP, FPSW, and accumulators (ACC0, ACC1). Values in the register-saving banks are undefined after a reset.

A unique number (bank number) is allocated to each register-saving bank. The range of bank numbers is from 0 to 255.

The register bank save function is optional. However, whether or not a CPU has register-saving banks, and the actual range of usable bank numbers (capacity of the memory installed for this purpose) if a CPU does have register-saving banks will depend on the product. For details, refer to the user's manual: hardware for each product.

For handling the occurrence of an exception, refer to section 5.2, Exception Handling Procedure.

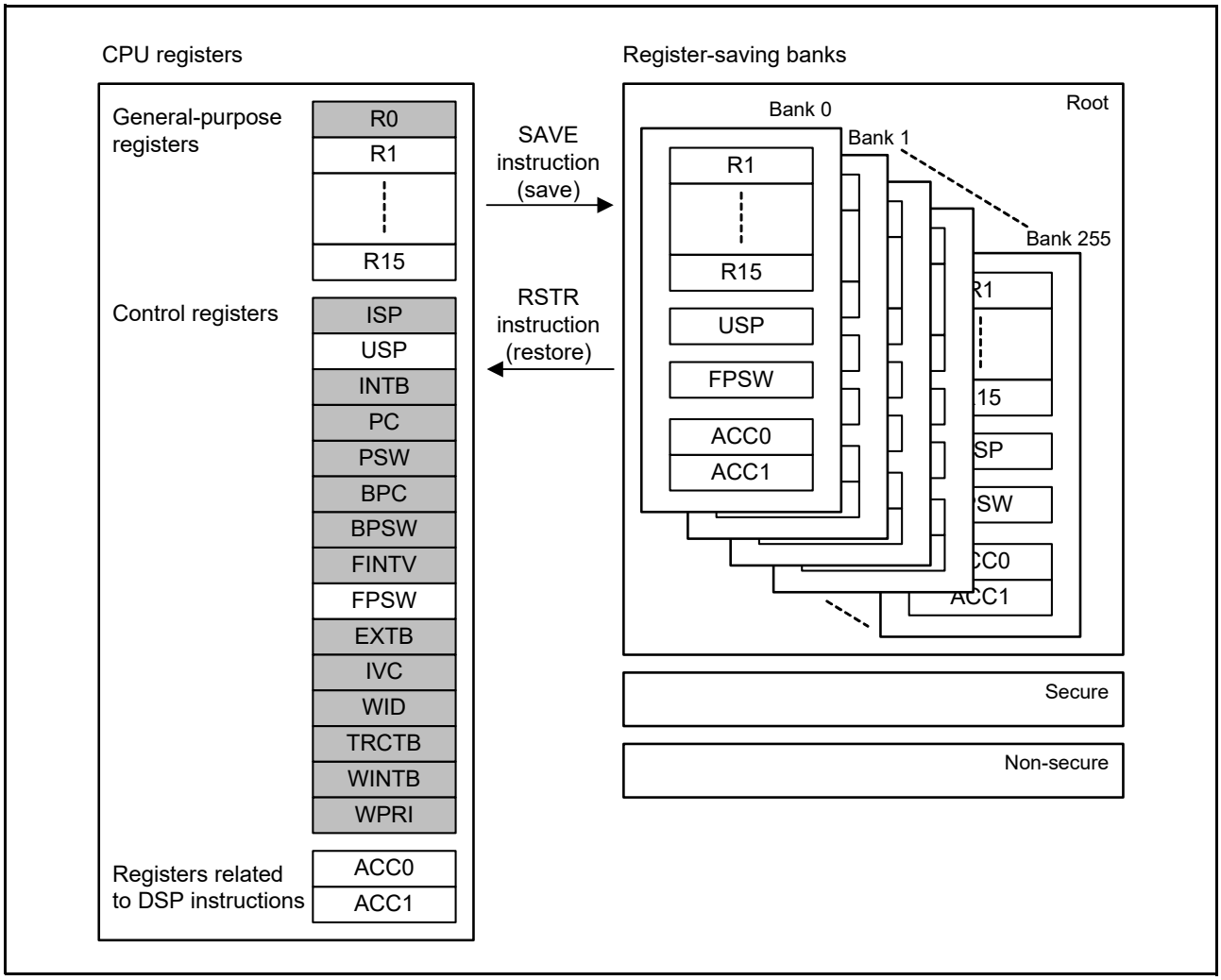


Figure 1.15 Register-Saving Banks

Each world has its own register-saving bank. There is a TR register-saving bank for root, an S register-saving bank for secure, and an NS register-saving bank for non-secure. The register-saving bank corresponding to the world in which the SAVE or RSTR instruction was executed is used.

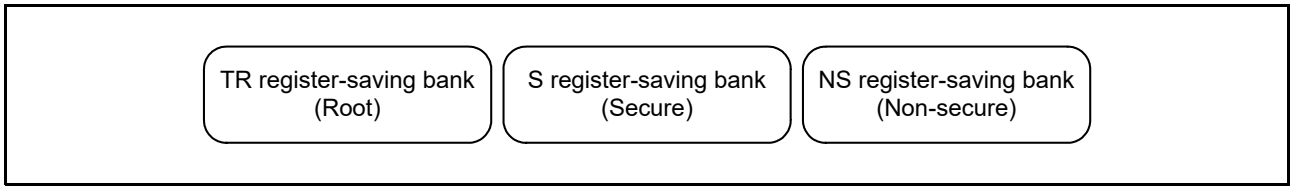


Figure 1.16 Configuration of Register-Saving Banks to Support the Protect Zone Function

1.11 Double-Precision Floating-Point Coprocessor

The double-precision floating-point coprocessor operates as a coprocessor of the CPU, and executes double-precision floating-point instructions. Using the double-precision floating-point coprocessor considerably accelerates the processing of double-precision floating point arithmetic.

The double-precision floating-point coprocessor is optional. Whether or not a CPU has a double-precision floating-point coprocessor depends on the product. For details, refer to the user's manual: hardware for each product.

1.11.1 Features

- Double-precision floating-point register set
  Double-precision floating-point data registers: Sixteen 64-bit registers
  Double-precision floating-point control registers: Four 32-bit registers
- Double-precision floating-point instructions: 21
- Notifying the interrupt controller of double-precision floating-point exceptions

1.11.2 Double-Precision Floating-Point Register Set

The double-precision floating-point coprocessor consists of 16 double-precision floating-point data registers and 4 double-precision floating-point control registers.



Figure 1.17 Double-Precision Floating-Point Register Set

1.11.2.1 Double-Precision Floating-Point Data Registers (DR0 to DR15)

16 double-precision floating-point data registers with 64-bit width are provided (DR0 to DR15). To specify 32-bit values, use the upper 32 bits (DRH0 to DRH15) or lower 32 bits (DRL0 to DRL15) as separate units.

1.11.2.2 Double-Precision Floating-Point Control Registers

Four double-precision floating-point control registers are provided.

- Double-precision floating-point status word (DPSW)
- Double-precision floating-point comparison result register (DCMR)
- Double-precision floating-point exception handling control register (DECNT)
- Double-precision floating-point exception program counter value (DEPC)

Note: The double-precision floating-point control registers are also represented by “DCR” and a number for the control register (DCRn) in descriptions of instructions that apply to the control registers in general.

DCR0: DPSW
DCR1: DCMR
DCR2: DECNT
DCR3: DEPC

(1) Double-Precision Floating-Point Status Word (DPSW)

	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	DFS	DFX	DFU	DFZ	DFO	DFV	—	—	—	—	—	—	—	—	—	—
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
	—	DEX	DEU	DEZ	DEO	DEV	—	DDN	DCE	DCX	DCU	DCZ	DCO	DCV	DRM[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	DRM[1:0]	Double-precision 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	DCV	Invalid operation cause flag	0: No invalid operation has been encountered. 1: Invalid operation has been encountered.	R/(W)*1
b3	DCO	Overflow cause flag	0: No overflow has occurred. 1: Overflow has occurred.	R/(W)*1
b4	DCZ	Division-by-zero cause flag	0: No division-by-zero has occurred. 1: Division-by-zero has occurred.	R/(W)*1
b5	DCU	Underflow cause flag	0: No underflow has occurred. 1: Underflow has occurred.	R/(W)*1
b6	DCX	Inexact cause flag	0: No inexact exception has been generated. 1: Inexact exception has been generated.	R/(W)*1
b7	DCE	Unimplemented processing cause flag	0: No unimplemented processing has been encountered. 1: Unimplemented processing has been encountered.	R/(W)*1
b8	DDN	0 flush bit of denormalized number	0: A denormalized number is handled as a denormalized number. 1: A denormalized number is handled as 0.*2	R/W
b9	—	Reserved	This bit is read as 0. The write value should be 0.	R/W
b10	DEV	Invalid operation exception enable bit	0: Invalid operation exception is masked. 1: Invalid operation exception is enabled.	R/W
b11	DEO	Overflow exception enable bit	0: Overflow exception is masked. 1: Overflow exception is enabled.	R/W
b12	DEZ	Division-by-zero exception enable bit	0: Division-by-zero exception is masked. 1: Division-by-zero exception is enabled.	R/W
b13	DEU	Underflow exception enable bit	0: Underflow exception is masked. 1: Underflow exception is enabled.	R/W
b14	DEX	Inexact exception enable bit	0: Inexact exception is masked. 1: Inexact exception is enabled.	R/W
b25 to b15	—	Reserved	These bits are read as 0. The write value should be 0.	R/W
b26	DFV*3	Invalid operation flag	0: No invalid operation has been encountered. 1: Invalid operation has been encountered.*8	R/W
b27	DFO*4	Overflow flag	0: No overflow has occurred. 1: Overflow has occurred.*8	R/W
b28	DFZ*5	Division-by-zero flag	0: No division-by-zero has occurred. 1: Division-by-zero has occurred.*8	R/W

Bit	Symbol	Bit Name	Description	R/W
b29	DFU*6	Underflow flag	0: No underflow has occurred. 1: Underflow has occurred.*8	R/W
b30	DFX*7	Inexact flag	0: No inexact exception has been generated. 1: Inexact exception has been generated.*8	R/W
b31	DFS	Double-precision floating-point error summary flag	This bit reflects the logical OR of the DFU, DFZ, DFO, and DFV flags.	R

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

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

Note: 3. When the DEV bit is set to 0, the DFV flag is enabled.

Note: 4. When the DEO bit is set to 0, the DFO flag is enabled.

Note: 5. When the DEZ bit is set to 0, the DFZ flag is enabled.

Note: 6. When the DEU bit is set to 0, the DFU flag is enabled.

Note: 7. When the DEX bit is set to 0, the DFX flag is enabled.

Note: 8. Once the bit has been set to 1, this value is retained until it is cleared to 0 by software.

The double-precision floating-point status word (DPSW) indicates the results of double-precision floating-point arithmetic operations.

When the corresponding exception handling enable bits (DEj) are set to enable processing of the exceptions (DEj=1), the DCj flags can be used by the exception handling routine (the interrupt handling routine with the double-precision floating-point exception as its source) to identify the source of that exception. If handling of an exception is masked (DEj=0), the DFj flag can be used to check for the generation of the exception at the end of a sequence of processing.

The DFj flags operate in an accumulative fashion (j = X, U, Z, O, or V).

The single-precision floating-point status word (FPSW) is neither referred to nor updated in double-precision floating-point arithmetic operations.

DRM[1:0] bits (Double-precision floating-point rounding-mode setting bits)

These bits specify the double-precision floating-point rounding-mode.

Explanation of Double-Precision Floating-Point Rounding Modes

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

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

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

DCV flag (Invalid operation cause flag), DCO flag (Overflow cause flag), DCZ flag (Division-by-zero cause flag), DCU flag (Underflow cause flag), DCX flag (Inexact cause flag), and DCE flag (Unimplemented processing cause flag)

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

- If an exception or processing that is not implemented is not encountered in the execution of a double-precision floating-point arithmetic instruction other than DABS or DNEG, the corresponding flags become 0.

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

DDN 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.

DEV bit (Invalid operation exception enable bit), DEO bit (Overflow exception enable bit), DEZ bit (Division-by-zero exception enable bit), DEU bit (Underflow exception enable bit), and DEX bit (Inexact exception enable bit)

When any of the five floating-point exceptions specified in the IEEE754 standard is generated by executing a double-precision floating-point arithmetic instruction, these bits control whether the CPU will start the exception handling procedure (interrupt information will be held by setting the DECNT.EHS bit and an interrupt request will be sent to the interrupt controller). When the bit corresponding to the exception is 0, the DECNT.EHS bit is not set to 1. When the bit corresponding to the exception is 1, the DECNT.EHS bit is set to 1. However, if the DECNT.EHM bit is 0, the DECNT.EHS bit will not be set to 1 regardless of the value of the bit corresponding to the exception.

DFV flag (Invalid operation flag), DFO flag (Overflow flag), DFZ flag (Division-by-zero flag), DFU flag (Underflow flag), and DFX flag (Inexact flag)

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

- When DEj 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)

DFS flag (Double-precision floating-point error summary flag)

This bit reflects the logical OR of the DFU, DFZ, DFO, and DFV flags.

The value of this register is not updated when the EHM and EHS bits in the DECNT register are 1.

(2) Double-Precision Floating-Point Comparison Result Register (DCMR)

	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
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
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	RES
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Symbol	Bit Name	Description	R/W
b0	RES	Double-precision floating-point compare instruction result flag	0: Condition for comparison was not satisfied. 1: Condition for comparison was satisfied.	R/W
b31 to b1	—	Reserved	These bits are read as 0. The write value should be 0.	R/W

(3) Double-Precision Floating-Point Exception Handling Control Register (DECNT)

	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	EHS
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
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	EHM
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Bit	Symbol	Bit Name	Description	R/W
b0	EHM	Double-precision floating-point exception information saving mode bit	0: Mode in which information on the generation of exceptions is not saved. Information on the generation of double-precision floating-point exceptions is not saved when they occur. No interrupt request is sent to the interrupt controller. 1: Mode in which information on the generation of exceptions is saved. When a double-precision floating-point exception is generated and an interrupt request is sent to the interrupt controller, the EHS bit is changed to 1 to save information at the time the exception was generated, and the interrupt request is sent to the interrupt controller.	R/W
b15 to b1	—	Reserved	These bits are read as 0. The write value should be 0.	R/W
b16	EHS	Double-precision floating-point exception information saving status bit	0: Exception information is not being saved. 1: Exception information is being saved. Updating of the following registers stops when the EHM and EHS bits are 1, and interrupts due to double-precision floating-point exceptions do not occur. • Double-precision floating-point status word • Double-precision floating-point exception program counter In addition, an interrupt request is sent to the interrupt controller. Writing 0 to this bit releases the information recorded at the time the exception was generated from the retained state.	R/W
b31 to b17	—	Reserved	These bits are read as 0. The write value should be 0.	R/W

(4) Double-Precision Floating-Point Exception Program Counter (DEPC)

	b31	b0
Value after reset	Undefined	

When a double-precision floating-point exception is generated and the corresponding DEX, DEU, DEZ, DEO, or DEV bit of the DPSW is 1, the value of the program counter for the instruction that has caused the most recent double-precision floating-point exception is saved in this register. This register is read-only. The value of this register is not updated when the value of the EHS bit in the DECNT register is 1 while that of the EHM bit in the same register is also 1.

1.11.3 Double-Precision Floating-Point Exceptions

Double-precision floating-point exceptions are generated when any of the five exceptions specified in the IEEE 754 standard, namely overflow, underflow, inexact, division-by-zero, or invalid operation, or an attempt to use processing that is not implemented, is detected upon execution of a double-precision floating-point arithmetic instruction. When a double-precision floating-point exception is generated, an interrupt request is sent to the interrupt controller according to the settings (the DEX, DEU, DEZ, DEO, and DEV bits of the DPSW corresponding to the five exceptions, and the EHM bit of DECNT) without the CPU performing exception handling.

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

1.11.3.1 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 double-precision floating-point format. Executing the DTOF instruction causes an overflow when the absolute value of the result of conversion is greater than the range of values that can be represented in the single-precision floating-point format. Table 1.10 lists the results of operations when an overflow exception occurs.

Table 1.10 Operation Results When an Overflow Exception Has Occurred

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

Note: An inexact exception occurs when the result of a double-precision floating-point arithmetic operation causes an overflow while DEO=0.

1.11.3.2 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 double-precision floating-point format. (However, this does not apply when the result is 0.) Executing the DTOF instruction causes an underflow when the absolute value of the result of conversion is smaller than the range of values that can be represented in the single-precision floating-point format (however, this does not apply when the result is 0). Table 1.11 lists the results of operations when an underflow exception occurs.

Table 1.11 Operation Results When an Underflow Exception Has Occurred

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

1.11.3.3 Inexact

An inexact exception occurs when the result of a hypothetical calculation with infinite precision differs from the actual result of the operation. Table 1.12 lists the conditions leading to an inexact exception and the results of operations.

Table 1.12 Conditions Leading to an Inexact Exception and the Operation Results

	Operation Result (Value in the Destination Register)	
Occurrence Condition	**DEX = 0**	**DEX = 1**
An overflow exception has occurred while overflow exceptions are masked.	Refer to Table 1.10, Operation Results When an Overflow Exception Has Occurred	No change
Rounding has been produced.	Value after rounding	

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

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

1.11.3.4 Division-by-Zero

Dividing a non-zero finite number by zero produces a division-by-zero exception. Table 1.13 lists the results of operations that have led to a division-by-zero exception. However, if the dividend is one of those listed in Table 1.14, the operation is not treated as division by zero.

Table 1.13 Operation Results When a Division-by Zero Exception Has Occurred

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

Table 1.14 Dividends and Operations that are not Treated as Division by Zero

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

1.11.3.5 Invalid Operation

Executing an invalid operation produces an invalid exception. Table 1.15 lists the conditions leading to an invalid exception and the results of operations.

Table 1.15 Conditions Leading to an Invalid Exception and the Operation Results

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

Table 1.16 lists the rules for generating QNaNs as the results of operations.

Table 1.16 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	src2 converted into a QNaN
Two QNaNs	src2
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	7FFFFFFFFFFFFFFFh

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

1.11.3.6 Unimplemented Processing

While DDN = 0, giving a denormalized number as an operand or calculation leading to an underflow constitutes processing of a double-precision floating-point arithmetic operation that is not implemented. An unimplemented processing exception will not occur with DDN = 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.

1.11.4 Data Types (for the Double-Precision Floating-Point Coprocessor)

The double-precision floating-point coprocessor can handle double-precision floating-point numbers.

In addition, it can handle single-precision floating-point numbers with the DTOF and FTOD instructions, and 32-bit integers with the DROUND, DTOI, DTOU, ITOD, and UTOD instructions.

For single-precision floating-point numbers, refer to section 1.6.2, Single-Precision Floating-Point Number, and for 32-bit integers, refer to section 1.6.1, Integer.

1.11.4.1 Double-Precision Floating-Point Numbers

The double-precision floating-point number is compliant with that specified in the IEEE754 standard. Operands of this type can be used in fifteen double-precision floating-point arithmetic instructions: DABS, DADD, DCMPcm, DDIV, DMUL, DNEG, DROUND, DSUB, DSQRT, DTOF, DTOI, DTOU, FTOD, ITOD, and UTOD.

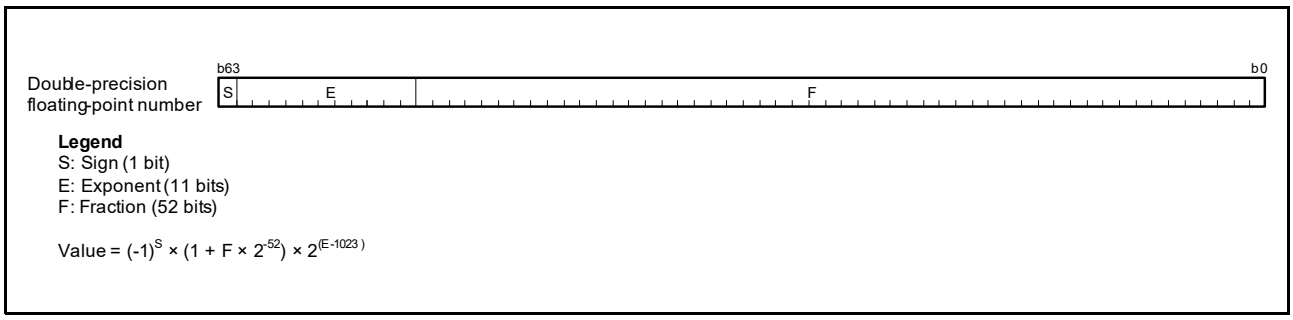


Figure 1.18 Double-Precision Floating-Point Number

The double-precision floating-point number can represent the values listed below.

- 0 < E < 2047 (normal numbers)
- E = 0 and F = 0 (signed zero)
- E = 0 and F > 0 (denormalized numbers)*
- E = 2047 and F = 0 (infinity (∞))
- E = 2047 and F > 0 (Not-a-Number (NaN))

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

1.11.5 Data Arrangement (for the Double-Precision Floating-Point Coprocessor)

The double-precision floating-point coprocessor can handle 32-bit and 64-bit values.
For 32-bit values, refer to section 1.7, Data Arrangement.

1.11.5.1 Arrangement of Data in the Double-Precision Floating-Point Registers

Figure 1.19 shows the relation between the sizes of registers and bit numbers.

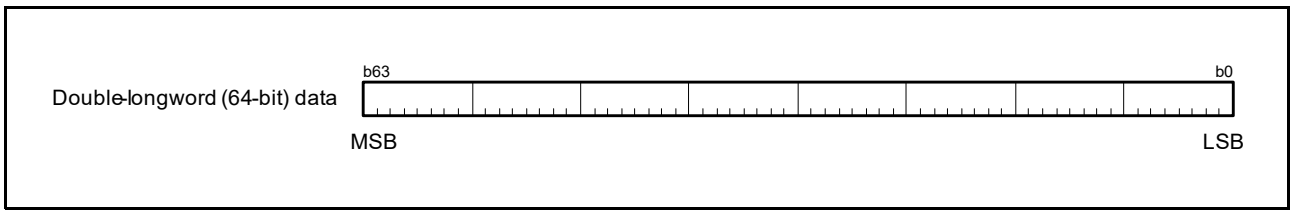


Figure 1.19 Data Arrangement in Registers

1.11.5.2 Arrangement of Data for Double-Precision Floating-Point Numbers in Memory

Double-longword (64 bits) is the only available data size for a double-precision floating-point number in memory. The data arrangement is selectable as little endian or big endian. Figure 1.20 shows the arrangement of data in memory.

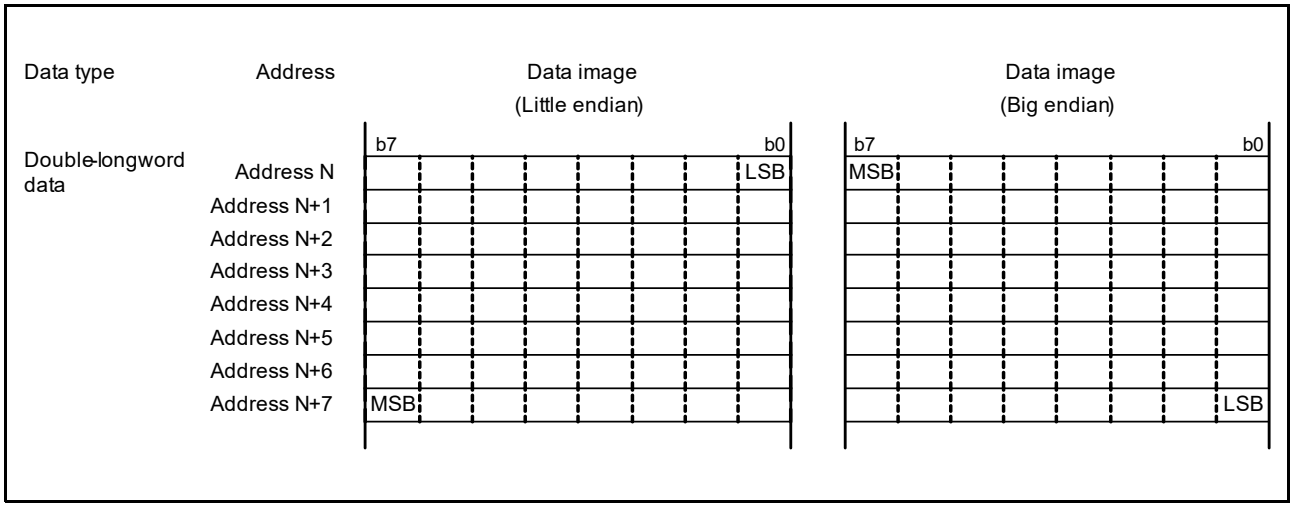


Figure 1.20 Data Arrangement in Memory

2. Addressing Modes

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

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

Products equipped with a double-precision floating-point coprocessor have a further two types of addressing mode.

- Double-precision floating-point data register direct
- Double-precision floating-point control register direct

2.1 Guide to This Section

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

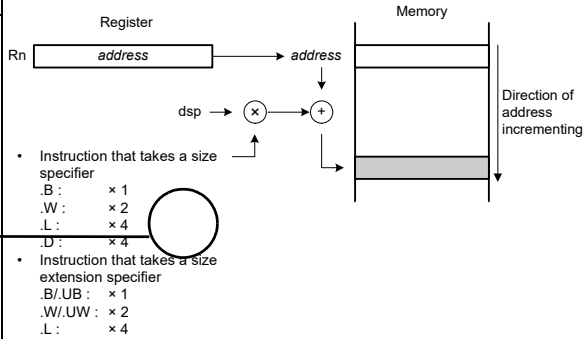
(1)

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 MOVU instructions.

(2)

(3)

(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.

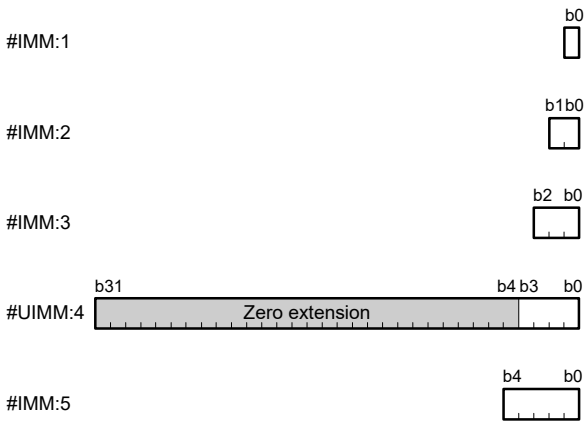
(3) Description

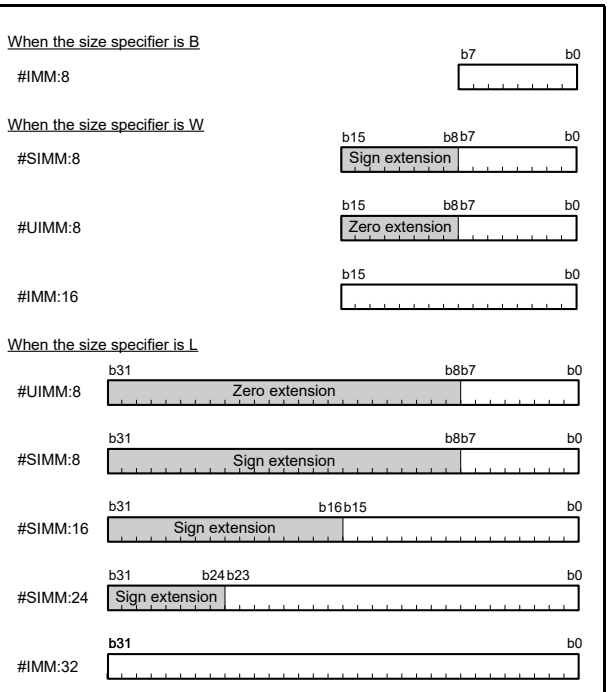
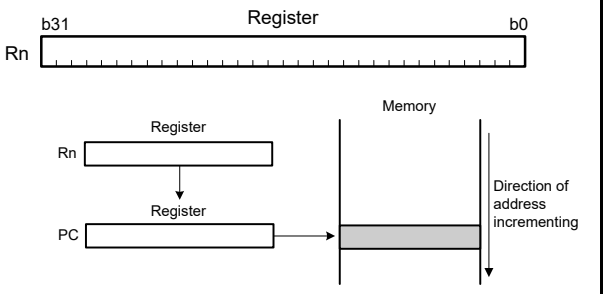
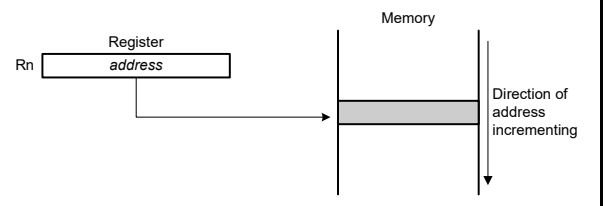
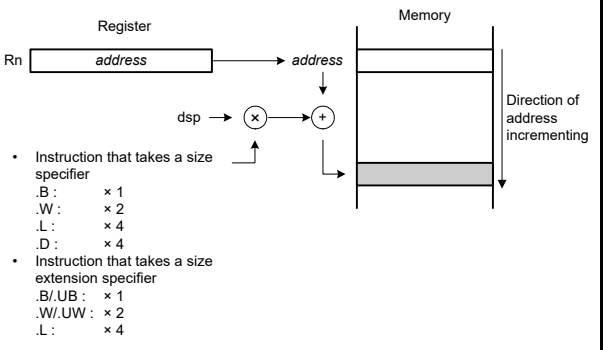
The operation and effective address range are described here.

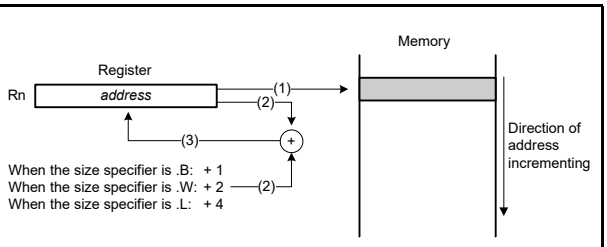
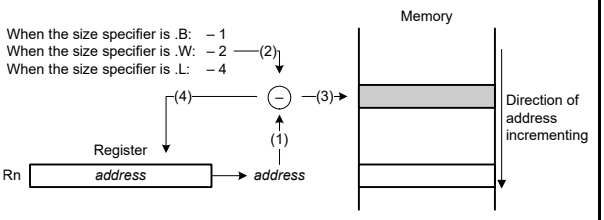
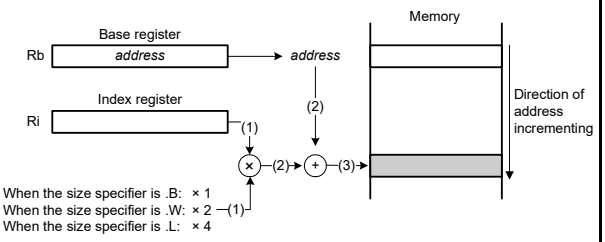
(4) Operation diagram

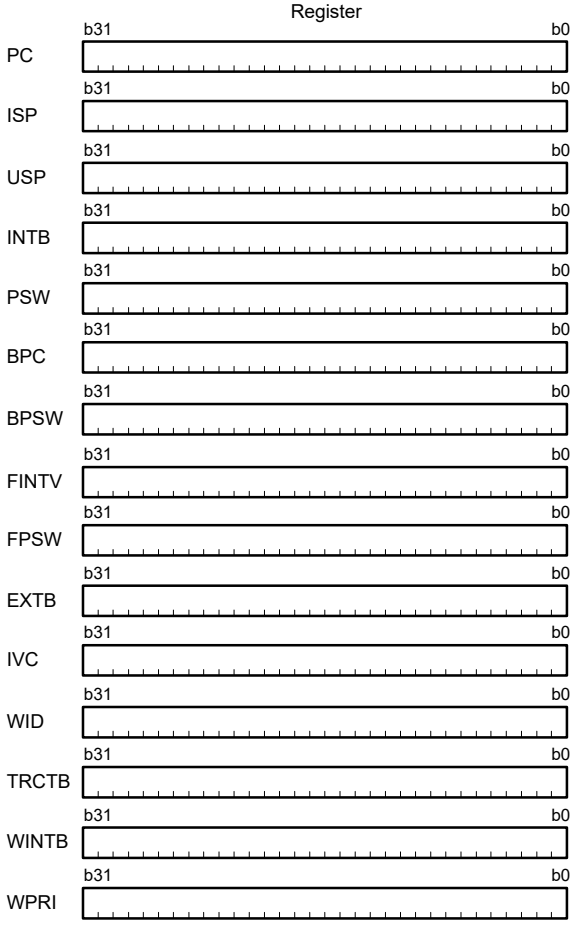
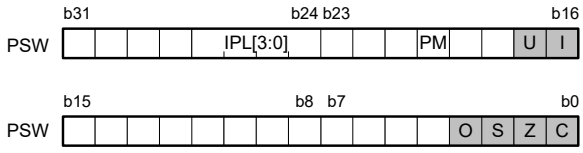
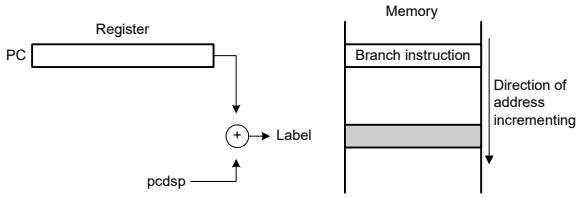
The operation of the addressing mode is illustrated here.

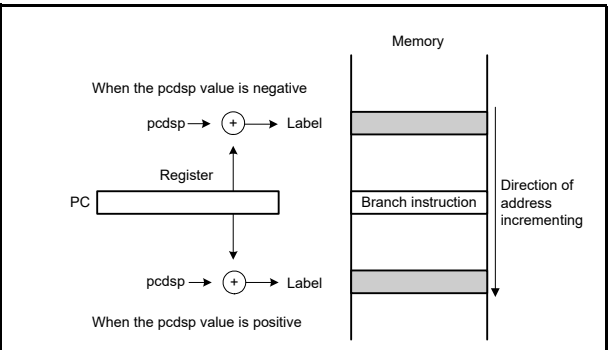
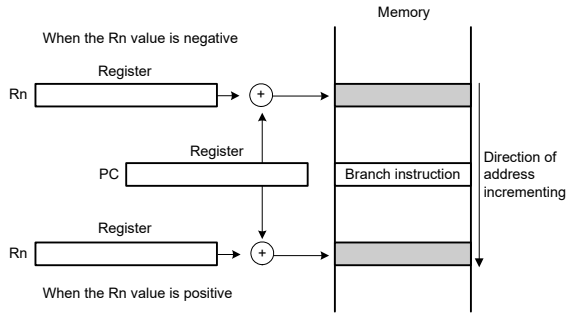
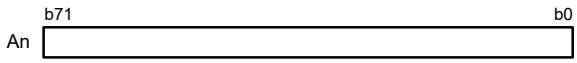
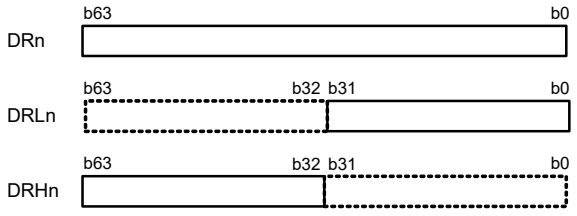
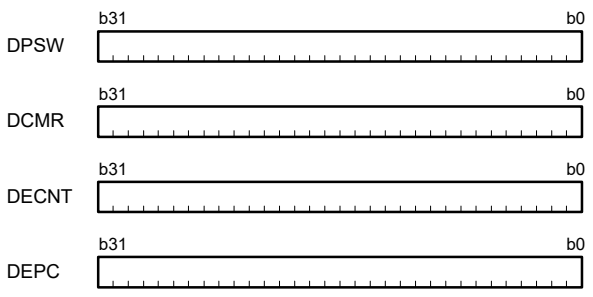
2.2 Addressing Modes

Immediate		
#IMM:1 #IMM:2 #IMM:3 #UIMM:4 #IMM:5	#IMM:1 The operand is the 1-bit immediate value indicated by #IMM. This addressing mode is used to specify sources for RACL, RACW, RDACL, and RDACW instructions. #IMM:2 The operand is the 2-bit immediate value indicated by #IMM. This addressing mode is used to specify sources for MVFACGU, MVFACHI, MVFACLO, and MVFACMI instructions. #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. #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 bit number and bit width for the data transfer instructions: BFMOV and BFMOVZ; - 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. - to specify the interrupt priority level for the MVTIPL instruction.	

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 MOVU instructions.	

Post-increment Register Indirect		
[Rn+] (Rn = R0 to R15)	The value in the specified register is the effective address of the operand. The range of valid addresses is from 00000000h to FFFFFFFFh. After the operation, 1, 2, or 4 is added to the value in the specified register according to the size specifier: .B, .W, or .L. This addressing mode is used with MOV and MOVU instructions.	
Pre-decrement Register Indirect		
[–Rn] (Rn = R0 to R15)	According to the size specifier: .B, .W, or .L, 1, 2, or 4 is subtracted from the value in the specified register. The value after the operation is the effective address of the operand. The range of valid addresses is from 00000000h to FFFFFFFFh. This addressing mode is used with MOV and MOVU instructions.	
Indexed Register Indirect		
[Ri, Rb] (Ri = R0 to R15, Rb = R0 to R15)	The effective address of the operand is the least significant 32 bits of the sum of the value in the index register (Ri), multiplied by 1, 2, or 4 according to the size specifier: .B, .W, or .L, and the value in the base register (Rb). The range of valid addresses is from 00000000h to FFFFFFFFh. This addressing mode is used with MOV and MOVU instructions.	

Control Register Direct		
PC ISP USP INTB PSW BPC BPSW FINTV FPSW EXTB IVC WID TRCTB WINTB WPRI	The operand is the specified control register. This addressing mode is used with MVFC, MVTC, POPC, PUSHC, TRMVFC, and TRMVTC instructions. The control registers that are specifiable depend on the instruction. For details, refer to the descriptions of the individual instructions.	
PSW Direct		
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 to be used with the B*Cnd* (only applicable in BEQ, BZ, BNE, and BNZ), and BRA instructions.	

<table>
<tr><td>pcdsp:8
pcdsp:16
pcdsp:24</td><td>When the branch distance specifier is .B, .W, or .A, the effective address is the signed sum of the value in the program counter (PC) and the displacement (pcdsp) value. The range of pcdsp depends on the branch distance specifier.
For .B: –128 ≤ pcdsp:8 ≤ 127
For .W: –32768 ≤ pcdsp:16 ≤ 32767
For .A: –8388608 ≤ pcdsp:24 ≤ 8388607
The range of valid addresses is from 00000000h to FFFFFFFFh. This addressing mode with the branch distance specifier “.B” is for use with any of the BCnd instructions and the BRA instruction, with the branch distance specifier “.W” is only for use with certain BCnd instructions (BEQ, BZ, BNE, and BNZ) and the BRA and BSR instructions, and with the branch distance specifier “.A” is only for use with the BRA and BSR instructions.</td><td> 
</td></tr>
<tr><td>Rn
(Rn = R0 to R15)</td><td>The effective address is the signed sum of the value in the program counter (PC) and the Rn value. The range of the Rn value is from –2147483648 to 2147483647. The range of valid addresses is from 00000000h to FFFFFFFFh. This addressing mode is used with BRA(.L) and BSR(.L) instructions.</td><td> 
</td></tr>
<tr><td colspan="2">Accumulator Direct</td><td rowspan="2"> 
</td></tr>
<tr><td>A0, A1
(A0 = ACC0, A1 = ACC1)</td><td>The specified accumulators (ACC0 and ACC1) are operands.</td></tr>
<tr><td colspan="2">Double-precision floating-point data register direct</td><td rowspan="2"> 
</td></tr>
<tr><td>DRn
(DRn = DR0 to DR15)
DRLn
(DRLn = DRL0 to DRL15)
DRHn
(DRHn = DRH0 to DRH15)</td><td>The operand is the specified double-precision floating-point data register.
DRLn indicates the lower 32 bits and DRHn indicates the upper 32 bits.</td></tr>
<tr><td colspan="2">Double-precision floating-point control register direct</td><td rowspan="2"> 
</td></tr>
<tr><td>DPSW
DCMR
DECNT
DEPC</td><td>The operand is the specified double-precision floating-point control register.</td></tr>
</table>

2.2.1 Ranges for Immediate Values

Ranges for immediate values are listed in Table 2.1.

Unless specifically stated otherwise in descriptions of the various instructions under section 3.4, Instructions in Detail, ranges for immediate values are as listed below.

Table 2.1 Ranges for Immediate Values

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

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

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

Note: 3. For the ranges of immediate values for INT and RTSD instructions, see the relevant descriptions under section 3.4, Instructions in Detail.

3. Instruction Descriptions

3.1 Overview of Instruction Set

The number of instructions for the RXv3e Architecture is 117. A variable-length instruction format of 1 to 8 bytes is used.

The RXv3e instruction set architecture provides upward compatibility from the RXv1, RXv2, and RXv3.

- Compared to the RXv1, the RXv2 realized higher speed arithmetic processing, comparable to that of a DSP, mainly through the addition of instructions for DSP and single-precision floating-point operations.
- Compared to the RXv2, the response of the RXv3 to interrupts has been made considerably faster, mainly through the addition of a register bank save function.
- Compared to the RXv3, the RXv3e core includes an implementation of the Protect Zone function for enhanced security.
- Products equipped with a double-precision floating-point coprocessor have a further 21 instructions for double-precision floating-point processing.

The RXv3e instruction set is listed below.

List of Instructions (1 / 6)

Instruction Type	Mnemonic	Function	Instruction Described in Detail (on Page)	Instruction Code Described in Detail (on Page)	Notes
Standard provided instructions					
Arithmetic/ logic instructions	ABS	Absolute value	84	280	
	ADC	Add with carry	85	281	
	ADD	Add without carry	86	282	
	AND	Logical AND	88	284	
	CMP	Compare	103	299	
	DIV	Divide signed	104	300	
	DIVU	Divide unsigned	105	302	
	EMUL	Extended multiply signed	108	304	
	EMULU	Extended multiply unsigned	111	305	
	MAX	Maximum of two signed integers	142	317	
	MIN	Minimum of two signed integers	143	318	
	MUL	Multiply	154	327	
	NEG	Negate (two's complement)	169	337	
	NOP	No operation	170	337	
	NOT	Logical NOT (one's complement)	171	338	
	OR	Logical OR	172	339	
	RMPA	Repeat multiply-accumulate	189	347	
	ROLC	Rotate left with carry	191	347	
	RORC	Rotate right with carry	192	348	
	ROTL	Rotate left	193	348	
	ROTR	Rotate right	194	349	
	SAT	Saturate	203	352	
	SATR	Saturate for RMPA	204	352	
	SBB	Subtract with borrow	205	353	
	SHAR	Arithmetic shift right	209	356	
	SHLL	Logical shift left	210	357	
	SHLR	Logical shift right	211	358	
	SUB	Subtract without borrow	218	362	
	TST	Test logical	223	364	
	XOR	Logical Exclusive OR	228	367	Extended for the RXv3

List of Instructions (2 / 6)

Instruction Type	Mnemonic		Function	Instruction Described in Detail (on Page)	Instruction Code Described in Detail (on Page)	Notes
Data transfer instructions	BFMOV		Bit field transfer	92	290	Supported by the RXv3 and later
	BFMOVZ		Bit field transfer with setting the other bits at the destination to zero	93	290	Supported by the RXv3 and later
	MOV		Move	144	319	
	MOVCO		Store data with LI flag clear	147	324	Supported by the RXv2 and later
	MOVLI		Load data with LI flag set	148	324	Supported by the RXv2 and later
	MOVU		Move unsigned	149	325	
	POP		Pop register from stack	173	340	
	POPC		Pop a control register from stack	174	341	Extended for the RXv3e
	POPM		Pop multiple registers from stack	175	341	
	PUSH		Push register on stack	176	342	
	PUSHC		Push a control register on stack	177	343	Extended for the RXv3e
	PUSHM		Push multiple registers on stack	178	343	
	REVL		Reverse endian within longword	187	346	
	REVW		Reverse endian within word	188	346	
	SC*Cnd*	SCGEU	Store condition	206	354	
		SCC		206	354	
		SCEQ		206	354	
		SCZ		206	354	
		SCGTU		206	354	
		SCPZ		206	354	
		SCGE		206	354	
		SCGT		206	354	
		SCO		206	354	
		SCLTU		206	354	
		SCNC		206	354	
		SCNE		206	354	
		SCNZ		206	354	
		SCLEU		206	354	
		SCN		206	354	
		SCLE		206	354	
		SCLT		206	354	
		SCNO		206	354	
	STNZ		Store on not zero	216	360	Extended for the RXv2
	STZ		Store on zero	217	361	Extended for the RXv2
	XCHG		Exchange	227	366	

List of Instructions (3 / 6)

Instruction Type	Mnemonic		Function	Instruction Described in Detail (on Page)	Instruction Code Described in Detail (on Page)	Notes
Branch instructions	B*Cnd*	BGEU	Conditional relative branch	91	288	
		BC		91	288	
		BEQ		91	288	
		BZ		91	288	
		BGTU		91	288	
		BPZ		91	288	
		BGE		91	288	
		BGT		91	288	
		BO		91	288	
		BLTU		91	288	
		BNC		91	288	
		BNE		91	288	
		BNZ		91	288	
		BLEU		91	288	
		BN		91	288	
		BLE		91	288	
		BLT		91	288	
		BNO		91	288	
	BRA		Unconditional relative branch	97	293	
	BSR		Relative branch to subroutine	100	296	
	JMP		Unconditional jump	137	315	
	JSR		Jump to subroutine	138	315	
	RTS		Return from subroutine	200	351	
	RTSD		Return from subroutine after deallocating stack frame	201	351	
Bit manipulation instructions	BCLR		Clear a bit	90	286	
	BM*Cnd*	BMGEU	Conditional bit transfer	94	291	
		BMC		94	291	
		BMEQ		94	291	
		BMZ		94	291	
		BMGTU		94	291	
		BMPZ		94	291	
		BMGE		94	291	
		BMGT		94	291	
		BMO		94	291	
		BMLTU		94	291	
		BMNC		94	291	
		BMNE		94	291	
		BMNZ		94	291	
		BMLEU		94	291	
		BMN		94	291	
		BMLE		94	291	
		BMLT		94	291	
		BMNO		94	291	
	BNOT		Not a bit	96	292	
	BSET		Set a bit	99	294	
	BTST		Test a bit	101	297	

List of Instructions (4 / 6)

Instruction Type	Mnemonic	Function	Instruction Described in Detail (on Page)	Instruction Code Described in Detail (on Page)	Notes
String manipulation instructions	SCMPU	String compare until not equal	207	354	
	SMOVB	String move backward	212	359	
	SMOVF	String move forward	213	359	
	SMOVU	String move until zero detected	214	359	
	SSTR	String store	215	360	
	SUNTIL	String search until equal	219	363	
	SWHILE	String search while equal	221	363	
System manipulation instructions	BRK	Unconditional trap	98	294	
	CLRPSW	Clear a flag or bit in the PSW	102	298	
	INT	Software interrupt	134	314	Extended for the RXv3e
	MVFC	Move data from a control register	163	333	Extended for the RXv3e
	MVTC	Move data to a control register	167	335	Extended for the RXv3e
	MVTIPL (privileged instruction)	Move data to IPL	168	336	Extended for the RXv3e
	RTE (privileged instruction)	Return from exception	198	350	Extended for the RXv3e
	RTFI (privileged instruction)	Return from fast interrupt	199	351	Extended for the RXv3e
	SETPSW	Set a flag or bit in the PSW	208	355	
	WAIT (privileged instruction)	Wait	226	366	
Single-precision floating-point arithmetic instructions	FADD	Add single-precision floating-point	113	307	Extended for the RXv2
	FCMP	Compare single-precision floating-point	116	308	
	FDIV	Divide single-precision floating-point	118	309	
	FMUL	Multiply single-precision floating-point	120	310	Extended for the RXv2
	FSUB	Subtract single-precision floating-point	125	312	Extended for the RXv2
	FSQRT	Square root of single-precision floating-point	123	311	Supported by the RXv2 and later
	FTOI	Convert single-precision floating-point to signed integer	128	313	
	FTOU	Convert single-precision floating-point to unsigned integer	131	313	Supported by the RXv2 and later
	ITOF	Convert signed integer to single-precision floating-point	135	314	
	ROUND	Round single-precision floating-point to signed integer	195	350	
	UTOF	Convert unsigned integer to single-precision floating-point	224	365	Supported by the RXv2 and later

List of Instructions (5 / 6)

Instruction Type	Mnemonic	Function	Instruction Described in Detail (on Page)	Instruction Code Described in Detail (on Page)	Notes
DSP instructions	EMACA	Extended multiply-accumulate to the accumulator	106	303	Supported by the RXv2 and later
	EMSBA	Extended multiply-subtract from the accumulator	107	303	Supported by the RXv2 and later
	EMULA	Extended multiply to the accumulator	110	305	Supported by the RXv2 and later
	MACHI	Multiply-Accumulate the upper words	139	316	Extended for the RXv2
	MACLH	Multiply-Accumulate the lower and upper words	140	316	Supported by the RXv2 and later
	MACLO	Multiply-Accumulate the lower words	141	316	Extended for the RXv2
	MSBHI	Multiply-Subtract the upper words	151	326	Supported by the RXv2 and later
	MSBLH	Multiply-Subtract the lower and upper words	152	326	Supported by the RXv2 and later
	MSBLO	Multiply-Subtract the lower words	153	327	Supported by the RXv2 and later
	MULHI	Multiply the upper words	156	329	Extended for the RXv2
	MULLH	Multiply the lower and upper words	157	329	Supported by the RXv2 and later
	MULLO	Multiply the lower words	158	330	Extended for the RXv2
	MVFACGU	Move data from the guard bits of the accumulator	159	330	Supported by the RXv2 and later
	MVFACHI	Move data from the upper longword of the accumulator	160	331	Extended for the RXv2
	MVFACLO	Move data from the lower longword of the accumulator	161	331	Supported by the RXv2 and later
	MVFACMI	Move data from the middle-order longword of the accumulator	162	332	Extended for the RXv2
	MVTACGU	Move data to the guard bits of the accumulator	164	334	Supported by the RXv2 and later
	MVTACHI	Move data to the upper longword of the accumulator	165	334	Extended for the RXv2
	MVTACLO	Move data to the lower longword of the accumulator	166	334	Extended for the RXv2
	RACL	Round the accumulator longword	179	344	Supported by the RXv2 and later
	RACW	Round the accumulator word	181	344	Extended for the RXv2
	RDACL	Round down the accumulator longword	183	345	Supported by the RXv2 and later
	RDACW	Round down the accumulator word	185	345	Supported by the RXv2 and later
Instructions for register bank save function (optional)					
Instructions for register bank save function	RSTR (privileged instruction)	Restore registers collectively	230	370	Supported by the RXv3 and later
	SAVE (privileged instruction)	Save registers collectively	231	370	

List of Instructions (6 / 6)

Instruction Type	Mnemonic	Function	Instruction Described in Detail (on Page)	Instruction Code Described in Detail (on Page)	Notes
Double-precision floating-point instructions (optional)					
Double-precision floating-point data transfer instructions	DMOV	Move double-precision floating-point	241	374	Supported by products with the double-precision floating-point coprocessor
	DPOPM	Pop multiple double-precision floating-point registers from stack	246	378	
	DPUSHM	Push multiple double-precision floating-point registers on stack	248	379	
	MVFDC	Move data from double-precision floating-point control register	267	383	
	MVFDR	Move data from double-precision floating-point comparison result register	268	383	
	MVTDC	Move data to double-precision floating-point control register	269	384	
Double-precision floating-point arithmetic instructions	DABS	Absolute value of double-precision floating-point	233	372	Supported by products with the double-precision floating-point coprocessor
	DADD	Add double-precision floating-point	234	372	
	DCMPcm	Compare double-precision floating-point	236	373	
	DDIV	Divide double-precision floating-point	239	373	
	DMUL	Multiply double-precision floating-point	243	377	
	DNEG	Negate double-precision floating-point	245	377	
	DROUND	Round double-precision floating-point to signed integer	250	380	
	DSQRT	Square root of double-precision floating-point	253	380	
	DSUB	Subtract double-precision floating-point	255	380	
	DTOF	Convert double-precision floating-point to single-precision floating-point	257	381	
	DTOI	Convert double-precision floating-point to signed integer	260	381	
	DTOU	Convert double-precision floating-point to unsigned integer	262	381	
	FTOD	Convert single-precision floating-point to double-precision floating-point	264	382	
	ITOD	Convert signed integer to double-precision floating-point	266	382	
	UTOD	Convert unsigned integer to double-precision floating-point	270	384	
Protect Zone instructions					
Protect Zone instructions	TRCALL (privileged instruction)	Call for transition to the root world	272	386	Supported by the RXv3e and later
	TRMVFC (privileged instruction)	Transfer from a control register in the root world	273	386	
	TRMVTC (privileged instruction)	Transfer to a control register in the root world	274	387	
	TRRTE (privileged instruction)	Return from exception in the root world	275	389	

3.2 List of RXv3e Extended Instructions

For the RXv3e architecture, 4 instructions are added (newly added instructions) and the specifications of 8 instructions are extended (specification extended instructions) from the RXv3 architecture.

3.2.1 RXv3e Newly Added Instructions

Table 3.1 lists the RXv3e instructions that are newly added compared to the RXv3 instruction set.

Table 3.1 List of Newly Added Instructions

Item	Mnemonic	Function
Protect Zone instructions	TRCALL	Call for transition to the root world
	TRMVFC	Transfer from a control register in the root world
	TRMVTC	Transfer to a control register in the root world
	TRRTE	Return from exception in the root world

3.2.2 Specification Extended Instructions

Table 3.2 lists the RXv3e instructions with specifications extended from the RXv3 instruction set.

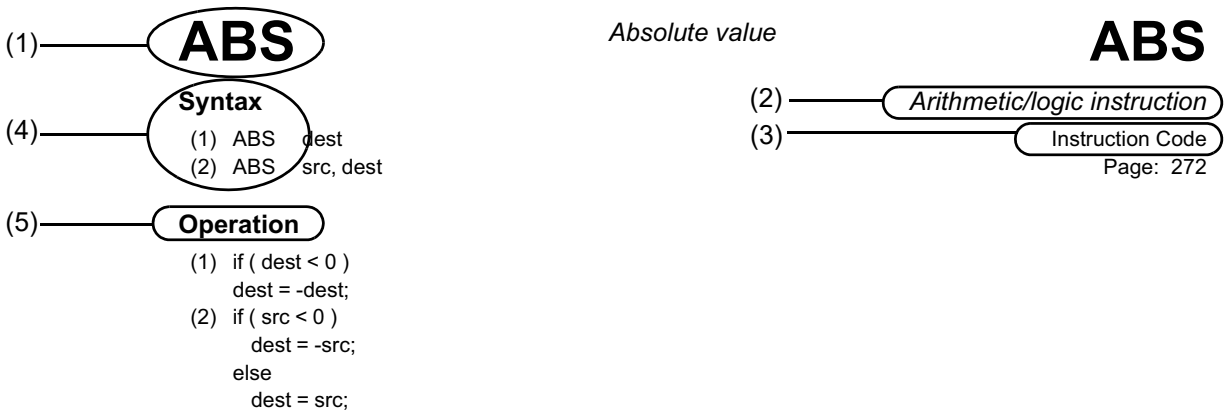
Table 3.2 List of Specification Extended Instructions

Item	Mnemonic	Overview of Specification Extension
Data transfer instructions	POPC	Adds the IVC, WID, and WPRI registers to the targets for manipulation.
	PUSHC	
System manipulation instructions	INT	Adds the register specification to the source operand.
	MVFC	Adds the IVC, WID, and WPRI registers to the targets for manipulation.
	MVTC	
	MVTIPL	Extends the interrupt priority levels that can be specified.
	RTE	Adds a restriction on writing to the PSW.IPL[4] bit in the non-secure world.
	RTFI	

3.3 Guide to This Section

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

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



(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)
.D	Double-longword (64 bits)

(c) Operand src, dest

Describes the operand.

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

(5) Operation

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

(a) Data type

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

(b) Pseudo-functions

register(n):	Returns register Rn, where n is the register number (n: 0 to 15).
register_num(Rn):	Returns register number n for Rn or, DRn, DCRn.
sqrt(src):	Returns the square root of src.
isNaN(src):	Returns 1 when src is NaN.
DR(n):	Represents the double-precision floating-point data register DRn that has the register number n. (n: 0 to 15)
DCR(n):	Represents the double-precision floating-point control register DCRn that has the register number n. (n: 0 to 3)
bank(n):	Represents the register-saving bank that has the bank number n.

(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.
{ R1, R2,…, ACC1}:	Represents the set of registers enumerated within “{}”.

(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 single-precision floating-point arithmetic instructions, changes in the states of flags (FX, FU, FZ, FO, FV, CE, CX, CU, CZ, CO, and CV) in the FPSW are also indicated.
For double-precision floating-point arithmetic instructions, changes in the states of flags (DFX, DFU, DFZ, DFO, DFV, DCE, DCX, DCU, DCZ, DCO, and DCV) in the DPSW are also indicated.
The symbols in the table mean the following:

—:	The flag does not change.
✓:	The flag changes depending on condition.

(8) Instruction Format

Indicates the instruction format.

Instruction Format

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

Instruction Format

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

Instruction Format

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

(a) Registers

Rs, Rs2, Rd, Rd2, Ri, and Rb mean that R0 to R15 are specifiable unless stated otherwise.
A0 and A1 are specifiable as the accumulators for DSP instructions.
DRs, DRs2, DRd, and DRd2 for the double-precision floating-point instructions are specifiable from among DR0 to DR15. DRLs and DRLd are specifiable from among DRL0 to DRL15. DRHs and DRHd are specifiable from among DRH0 to DRH15.

(b) Control registers

Rx indicates that the PC, ISP, USP, INTB, PSW, BPC, BPSW, FINTV, FPSW, EXTB, IVC, WID, WPRI, TRCTB, and WINTB are selectable. The control registers that are specifiable depend on the instruction. For details, refer to the descriptions of the individual instructions.
DCRs, DCRs2, DCRd, and DCRd2 for the double-precision floating-point instructions are specifiable from among DPSW, DCMR, DECNT, and DEPC.

(c) Flag and bit

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

(d) Immediate value

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

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

.memex indicates the size of an operand in memory and the form of extension. Each instruction with a size-extension specifier is expanded accordingly and then executed at the corresponding processing size.

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

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

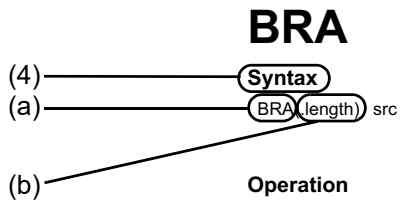
(f) Processing size

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

(9) Description Example

Shows a description example for the instruction.

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



BRA *Unconditional relative branch* **BRA**

Branch instruction

Instruction Code
Page: 286

Operation

PC = PC + src;

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

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

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

Description Example

```
BRA    label
BRA    1000h
```

(4) Syntax

Indicates the syntax of the instruction using symbols.

(a) Mnemonic

Describes the mnemonic.

(b) Branch distance specifier .length

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

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

3.4 Instructions in Detail

The RXv3e instructions are described in detail in this section.

3.4.1 Standard provided instructions

The following pages give details of the standard provided instructions.

ABS

Absolute value

ABS

Arithmetic/logic instruction

Instruction Code
Page: 280

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
```

ADC

Add with carry

ADC

Syntax

Arithmetic/logic instruction

ADC src, dest

Instruction Code
Page: 281

Operation

dest = dest + src + C;

Function

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

Flag Change

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

Instruction Format

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

Note: * For the RX Family assembler manufactured by Renesas Electronics Corp., enter a scaled value (the actual value multiplied by 4) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 1020 (255 × 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

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

ADD

Add without carry

ADD

Arithmetic/logic instruction

Instruction Code

Page: 282

Syntax

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

Operation

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

Function

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

Flag Change

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

Instruction Format

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

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

Description Example

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

AND

Logical AND

AND

Arithmetic/logic instruction

Instruction Code

Page: 284

Syntax

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

Operation

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

Function

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

Flag Change

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

Instruction Format

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

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

Description Example

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

BCLR

Clear a bit

BCLR

Syntax

Bit manipulation instruction

BCLR src, dest

Instruction Code
Page: 286

Operation

(1) When dest is a memory location:
```
unsigned char dest;
dest &= ~( 1 << ( src & 7 ));
```

(2) When dest is a register:
```
register unsigned long dest;
dest &= ~( 1 << ( src & 31 ));
```

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

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

B*Cnd*

Conditional relative branch

B*Cnd*

Syntax

Branch instruction

B*Cnd*(.length) src

Instruction Code
Page: 288

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

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

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

Description Example

```
BC    label
BC    1000h
```

BFMOV

Bit field transfer

BFMOV

Syntax

Data transfer instruction

BFMOV slsb, dlsb, width, src, dest

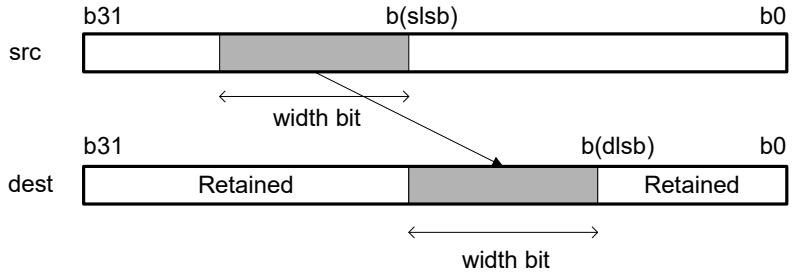
Instruction Code
Page: 290

Operation

```
unsigned long tmp1, tmp2;
tmp1 = (0FFFFFFFFh >> (32-width)) << dlsb;
tmp2 = (src >> slsb) << dlsb;
dest = (tmp2 & tmp1) | (dest & ~tmp1);
```

Function

- The number of bits specified by width from the bit position slsb at the location src are transferred to the number of bits specified by width from the bit position dlsb at the location dest. The values of the rest of the bits at dest are retained.
- The range of slsb is 0 ≤ slsb ≤ 31, the range of dlsb is 0 ≤ dlsb ≤ 31, and the range of width is 1 ≤ width ≤ 31.
- If (slsb + width) > 32 or (dlsb + width) > 32, then dest becomes undefined.



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax		Processing Size	Operand					Code Size (Byte)
			slsb	dlsb	width	src	dest	
BFMOV	slsb, dlsb, width, src, dest	L	#IMM:5	#IMM:5	#IMM:5	Rs	Rd	5

Description Example

BFMOV #5, #10, #3, R1, R2

BFMOVZ

Bit field transfer with setting the other bits at the destination to zero

BFMOVZ

Syntax

Data transfer instruction

BFMOVZ slsb, dlsb, width, src, dest

Instruction Code

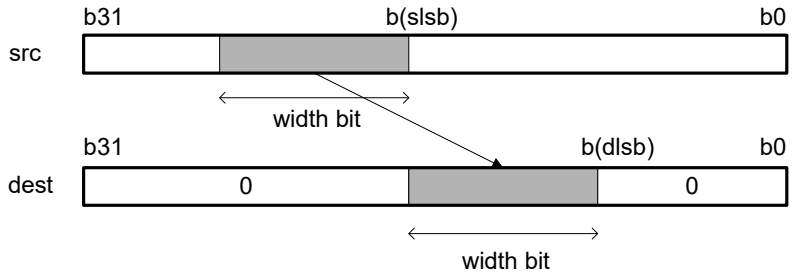
Page: 290

Operation

```
unsigned long tmp1, tmp2;
tmp1 = (0FFFFFFFFh >> (32-width)) << dlsb;
tmp2 = (src >> slsb) << dlsb;
dest = (tmp2 & tmp1);
```

Function

- The number of bits specified by width from the bit position slsb at the location src are transferred to the number of bits specified by width from the bit position dlsb at the location dest. The rest of the bits at dest become 0.
- The range of slsb is 0 ≤ slsb ≤ 31, the range of dlsb is 0 ≤ dlsb ≤ 31, and the range of width is 1 ≤ width ≤ 31.
- If (slsb + width) > 32 or (dlsb + width) > 32, then dest becomes undefined.



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax		Processing Size	Operand					Code Size (Byte)
			slsb	dlsb	width	src	dest	
BFMOVZ	slsb, dlsb, width, src, dest	L	#IMM:5	#IMM:5	#IMM:5	Rs	Rd	5

Description Example

BFMOVZ #5, #10, #3, R1, R2

BM*Cnd*

Conditional bit transfer

BM*Cnd*

Syntax

Bit manipulation instruction

BM*Cnd* src, dest

Instruction Code
Page: 291

Operation

(1) When dest is a memory location:

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

(2) When dest is a register:

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

Function

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

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

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

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

BNOT

Not a bit

BNOT

Syntax

Bit manipulation instruction

BNOT src, dest

Instruction Code
Page: 292

Operation

(1) When dest is a memory location:
unsigned char dest;
dest ^= (1 << (src & 7));

(2) When dest is a register:
register unsigned long dest;
dest ^= (1 << (src & 31));

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

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

BRA

Unconditional relative branch

BRA

Syntax

BRA(.length) src

Branch instruction

Instruction Code
Page: 293

Operation

PC = PC + src;

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

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

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

Description Example

```
BRA    label
BRA    1000h
```

BRK

Unconditional trap

BRK

Syntax

System manipulation instruction

BRK

Instruction Code
Page: 294

Operation

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

Function

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

Flag Change

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

Instruction Format

Syntax	Code Size (Byte)
BRK	1

Description Example

```
BRK
```

BSET

Set a bit

BSET

Syntax

Bit manipulation instruction

BSET src, dest

Instruction Code
Page: 294

Operation

(1) When dest is a memory location:
```
unsigned char dest;
dest |= ( 1 << ( src & 7 ));
```

(2) When dest is a register:
```
register unsigned long dest;
dest |= ( 1 << ( src & 31 ));
```

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

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

BSR

Relative branch to subroutine

BSR

Syntax

Branch instruction

BSR(.length) src

Instruction Code
Page: 296

Operation

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

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

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

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

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

Description Example

```
BSR    label
BSR    1000h
```

BTST

Test a bit

BTST

Syntax

Bit manipulation instruction

BTST src, src2

Instruction Code
Page: 297

Operation

(1) When src2 is a memory location:
unsigned char src2;
Z = ˜((src2 >> (src & 7)) & 1);
C = ((src2 >> (src & 7)) & 1);

(2) When src2 is a register:
register unsigned long src2;
Z = ˜((src2 >> (src & 31)) & 1);
C = ((src2 >> (src & 31)) & 1);

Function

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

Flag Change

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

Instruction Format

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

Description Example

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

CLRPSW *Clear a flag or bit in the PSW* CLRPSW

Syntax

System manipulation instruction

CLRPSW dest

Instruction Code
Page: 298

Operation

dest = 0;

Function

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

Flag Change

Flag	Change	Condition
C	*	
Z	*	
S	*	
O	*	

Note: * The specified flag becomes 0.

Instruction Format

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

Description Example

```
CLRPSW C
CLRPSW Z
```

CMP

Compare

CMP

Arithmetic/logic instruction

Syntax

CMP src, src2

Instruction Code
Page: 299

Operation

src2 - src;

Function

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

Flag Change

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

Instruction Format

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

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

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

Description Example

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

DIV

Divide signed

DIV

Syntax

Arithmetic/logic instruction

DIV src, dest

Instruction Code
Page: 300

Operation

dest = dest / src;

Function

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

Flag Change

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

Instruction Format

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

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

Description Example

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

DIVU

Divide unsigned

DIVU

Syntax

Arithmetic/logic instruction

DIVU src, dest

Instruction Code

Page: 302

Operation

dest = dest / src;

Function

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

Flag Change

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

Instruction Format

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

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

Description Example

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

EMACA

Extended multiply-accumulate to the accumulator

EMACA

DSP instruction

Instruction Code

Page: 303

Syntax

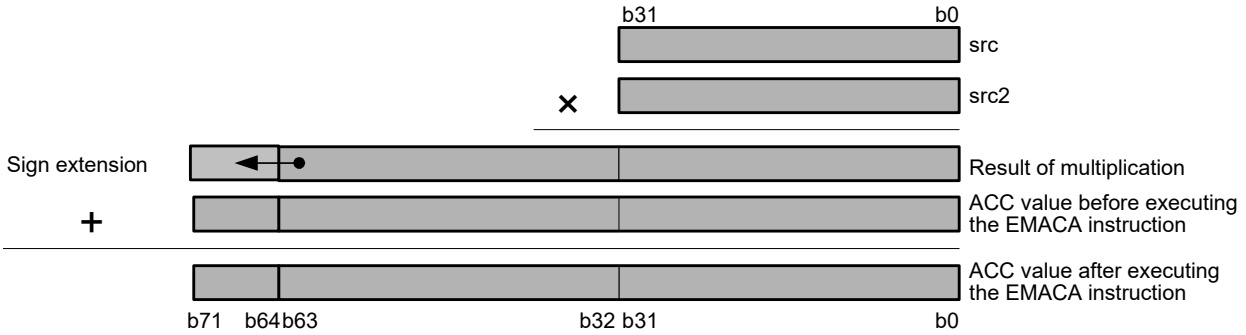
EMACA src, src2, Adest

Operation

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

Function

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



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

EMACA R1, R2, A1

EMSBA

Extended multiply-subtract from the accumulator

EMSBA

Syntax

DSP instruction

EMSBA src, src2, Adest

Instruction Code

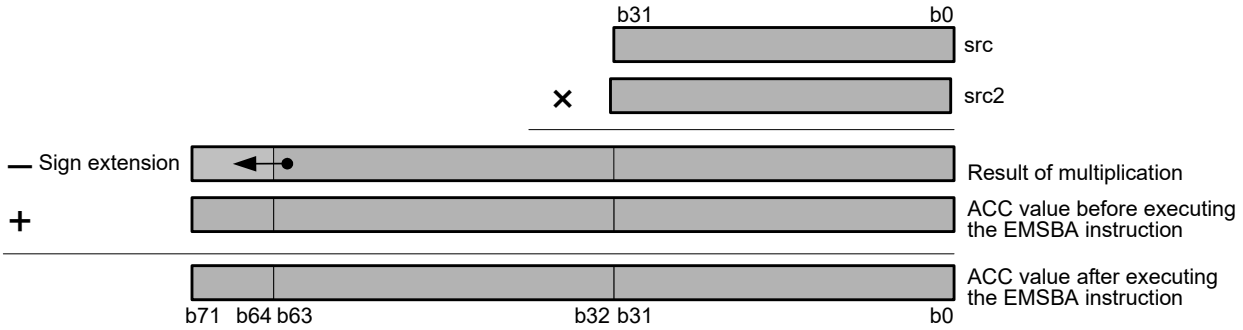
Page: 303

Operation

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

Function

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



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

EMSBA R1, R2, A1

EMUL

Extended multiply signed

EMUL

Syntax

Arithmetic/logic instruction

EMUL src, dest

Instruction Code

Page: 304

Operation

dest2:dest = dest * src;

Function

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

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

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

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

Description Example

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

EMULA

Extended multiply to the accumulator

EMULA

Syntax

DSP instruction

EMULA src, src2, Adest

Instruction Code

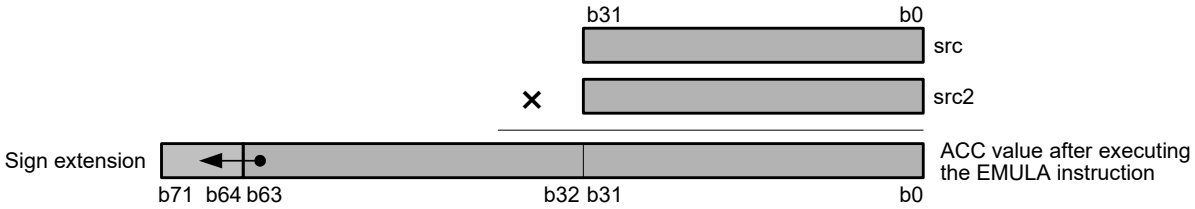
Page: 305

Operation

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

Function

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



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

EMULA R1, R2, A1

EMULU

Extended multiply unsigned

EMULU

Syntax

Arithmetic/logic instruction

EMULU src, dest

Instruction Code
Page: 305

Operation

dest2:dest = dest * src;

Function

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

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

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

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

Description Example

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

FADD

Add single-precision floating-point

FADD

Syntax

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

Single-precision floating-point arithmetic instruction

Instruction Code
Page: 307

Operation

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

Function

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

- Rounding of the result is in accordance with the setting of the RM[1:0] bits in the FPSW.
- Handling of denormalized numbers depends on the setting of the DN bit in the FPSW.
- The result of the operation is +0 if "src and dest" or "src and src2" have the opposite signs and their sum is exactly 0, except when the rounding mode is towards –∞. The operation result is –0 when the rounding mode is towards –∞.

Flag Change

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

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

Instruction Format

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

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

Sources of Single-Precision Floating-Point Exceptions

Unimplemented processing
Invalid operation
Overflow
Underflow
Inexact

Description Example

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

Supplementary Description

- The following tables show the correspondences between the src, src2, and dest values and the results of operations when the value of the DN bit in the FPSW is 0 or 1.

When DN = 0

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

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

When DN = 1

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

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

FCMP

Compare single-precision floating-point

FCMP

Syntax

FCMP src, src2

Single-precision floating-point arithmetic instruction

Instruction Code

Page: 308

Operation

src2 - src;

Function

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

Flag Change

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

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

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

Instruction Format

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

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

Sources of Single-Precision Floating-Point Exceptions

Unimplemented processing
Invalid operation

Description Example

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

Supplementary Description

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

When DN = 0

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

When DN = 1

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

FDIV

Divide single-precision floating-point

FDIV

Syntax

FDIV src, dest

Single-precision floating-point arithmetic instruction

Instruction Code
Page: 309

Operation

dest = dest / src;

Function

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

Flag Change

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

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

Instruction Format

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

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

Sources of Single-Precision Floating-Point Exceptions

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

Description Example

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

Supplementary Description

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

When DN = 0

<table>
<tr><td colspan="2" rowspan="2"></td><td colspan="8">src</td></tr>
<tr><td>Normalized</td><td>+0</td><td>–0</td><td>+∞</td><td>–∞</td><td>Denormalized</td><td>QNaN</td><td>SNaN</td></tr>
<tr><td rowspan="8">dest</td><td>Normalized</td><td>Division</td><td colspan="2">Division-by-zero</td><td colspan="2">0</td><td rowspan="5"></td><td rowspan="6"></td><td rowspan="7"></td></tr>
<tr><td>+0</td><td rowspan="2">0</td><td colspan="2" rowspan="2">Invalid operation</td><td>+0</td><td>–0</td></tr>
<tr><td>–0</td><td>–0</td><td>+0</td></tr>
<tr><td>+∞</td><td rowspan="2">∞</td><td>+∞</td><td>–∞</td><td colspan="2" rowspan="2">Invalid operation</td></tr>
<tr><td>–∞</td><td>–∞</td><td>+∞</td></tr>
<tr><td>Denormalized</td><td colspan="6">Unimplemented processing</td></tr>
<tr><td>QNaN</td><td colspan="7">QNaN</td></tr>
<tr><td>SNaN</td><td colspan="8">Invalid operation</td></tr>
</table>

When DN = 1

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

FMUL

Multiply single-precision floating-point

FMUL

Syntax

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

Single-precision floating-point arithmetic instruction

Instruction Code
Page: 310

Operation

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

Function

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

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

Note: The accumulator (ACC0) is used to perform the function. The value of ACC0 after executing the instruction is undefined regardless of generation of single-precision floating-point exceptions.

Flag Change

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

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

Instruction Format

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

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

Sources of Single-Precision Floating-Point Exceptions

Unimplemented processing
Invalid operation
Overflow
Underflow
Inexact

Description Example

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

Supplementary Description

- The following tables show the correspondences between the src, src2, and dest values and the results of operations when the value of the DN bit in the FPSW is 0 or 1.

When DN = 0

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

When DN = 1

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

FSQRT *Square root of single-precision floating-point* FSQRT

Syntax

FSQRT src, dest

Single-precision floating-point arithmetic instruction

Instruction Code
Page: 311

Operation

dest = sqrt(src);

Function

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

Flag Change

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

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

Instruction Format

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

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

Sources of Single-Precision Floating-Point Exceptions

Unimplemented processing
Invalid operation
Inexact

Description Example

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

Supplementary Description

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

When DN = 0

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

When DN = 1

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

Rules for Generating QNaN When Invalid Operation is Generated

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

Note: Corresponds to Table 1.9, Rules for Generating QNaNs.

FSUB

Subtract single-precision floating-point

FSUB

Syntax

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

Single-precision floating-point arithmetic instruction

Instruction Code
Page: 312

Operation

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

Function

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

- Rounding of the result is in accordance with the setting of the RM[1:0] bits in the FPSW.
- Handling of denormalized numbers depends on the setting of the DN bit in the FPSW.
- The result of the operation is +0 if “src and dest” or “src and src2” have the same sign and the result of subtraction is exactly 0, except when the rounding mode is towards –∞. The operation result is –0 when the rounding mode is towards –∞.

Flag Change

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

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

Instruction Format

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

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

Sources of Single-Precision Floating-Point Exceptions

Unimplemented processing
Invalid operation
Overflow
Underflow
Inexact

Description Example

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

Supplementary Description

- The following tables show the correspondences between the src, src2, and dest values and the results of operations when the value of the DN bit in the FPSW is 0 or 1.

When DN = 0

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

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

When DN = 1

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

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

FTOI

Convert single-precision floating-point to signed integer

FTOI

Syntax

FTOI src, dest

Single-precision floating-point arithmetic instruction

Instruction Code

Page: 313

Operation

dest = (signed long) src;

Function

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

Flag Change

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

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

Instruction Format

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

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

Sources of Single-Precision Floating-Point Exceptions

Unimplemented processing
Invalid operation
Inexact

Description Example

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

Supplementary Description

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

When DN = 0

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

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

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

When DN = 1

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

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

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

FTOU

Convert single-precision floating-point to unsigned integer

FTOU

Syntax

FTOU src, dest

Single-precision floating-point arithmetic instruction

Instruction Code

Page: 313

Operation

dest = (unsigned long) src;

Function

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

Flag Change

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

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

Instruction Format

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

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

Sources of Single-Precision Floating-Point Exceptions

Unimplemented processing
Invalid operation
Inexact

Description Example

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

Supplementary Description

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

When DN = 0

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

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

When DN = 1

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

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

INT

Software interrupt

INT

Syntax

System manipulation instruction

INT src

Instruction Code
Page: 314

Operation

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

Function

- This instruction generates the unconditional trap which corresponds to the number specified as src.
- The INT instruction number (src) is in the range $0 \leq src \leq 255$.
- This instruction causes a transition to supervisor mode, and clears the PM bit in the PSW to 0.
- This instruction clears the U and I bits in the PSW to 0.
- When src is in register, only eight bits in the LSB are valid.
- Whether this instruction becomes privileged depends on the value of the IVC.UIR[1:0] bits, the addressing mode, and the value of src. For details, see the description of the interrupt vector control register (IVC).

Flag Change

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

Instruction Format

Syntax	Operand src	Code Size (Byte)
INT src	#IMM:8	3
	Rsrc	4

Description Example

```
INT    #0
INT    R1
```

ITOF

Convert signed integer to single-precision floating-point

ITOF

Syntax

ITOF src, dest

Single-precision floating-point arithmetic instruction

Instruction Code

Page: 314

Operation

dest = (float) src;

Function

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

Flag Change

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

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

Instruction Format

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

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

Sources of Single-Precision Floating-Point Exceptions

Inexact

Description Example

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

JMP

Unconditional jump

JMP

Branch instruction

Instruction Code
Page: 315

Syntax

JMP src

Operation

PC = src;

Function

- This instruction branches to the instruction specified by src.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

JMP R1

JSR

Jump to subroutine

JSR

Syntax

Branch instruction

JSR src

Instruction Code
Page: 315

Operation

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

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

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

JSR R1

MACHI

Multiply-Accumulate the upper words

MACHI

Syntax

DSP instruction

MACHI src, src2, Adest

Instruction Code

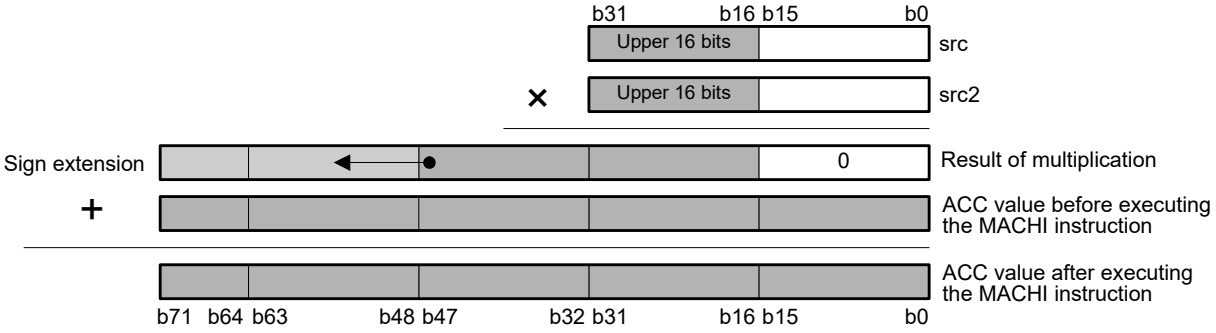
Page: 316

Operation

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

Function

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



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

MACHI R1, R2, A1

MACLH

Multiply-Accumulate the lower and upper words

MACLH

DSP instruction

Instruction Code
Page: 316

Syntax

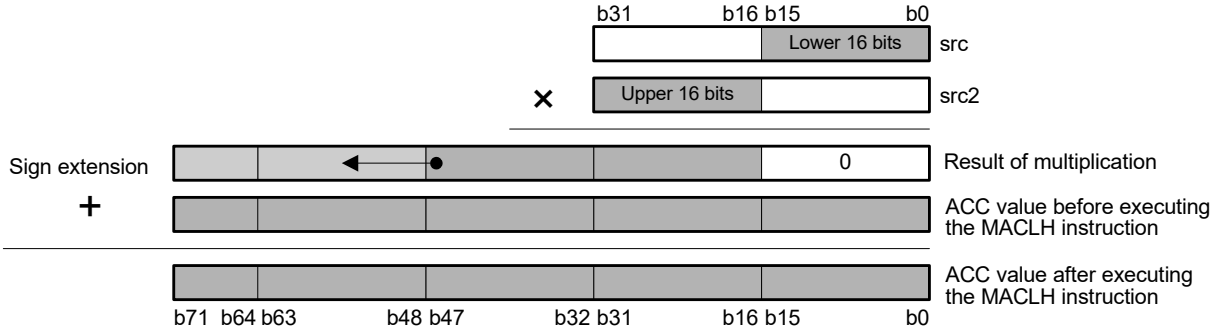
MACLH src, src2, Adest

Operation

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

Function

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



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

MACLH R1, R2, A1

MACLO

Multiply-Accumulate the lower words

MACLO

Syntax

DSP instruction

MACLO src, src2, Adest

Instruction Code

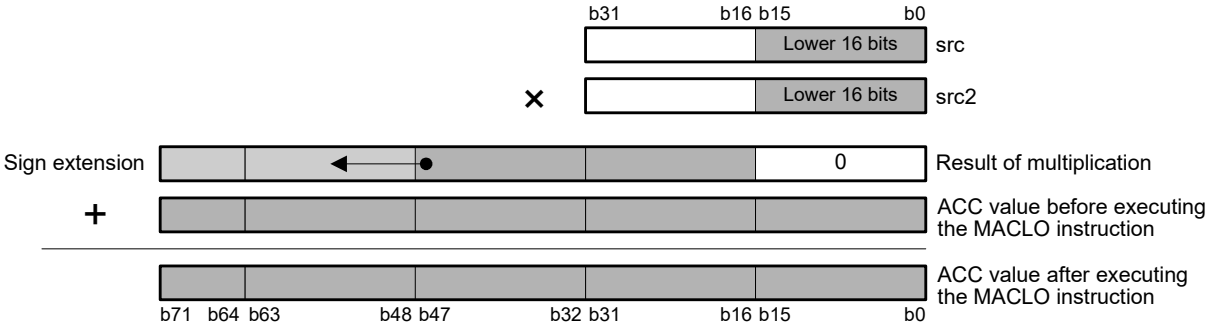
Page: 316

Operation

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

Function

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



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

MACLO R1, R2, A1

MAX

Maximum of two signed integers

MAX

Syntax

Arithmetic/logic instruction

MAX src, dest

Instruction Code
Page: 317

Operation

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

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

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

Description Example

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

MIN

Minimum of two signed integers

MIN

Syntax

Arithmetic/logic instruction

MIN src, dest

Instruction Code
Page: 318

Operation

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

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

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

Description Example

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

MOV

Move

MOV

Syntax

Data transfer instruction

MOV.size src, dest

Instruction Code
Page: 319

Operation

dest = src;

Function

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

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

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

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

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

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

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

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

Description Example

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

MOVCO

Store data with LI flag clear

MOVCO

Syntax

Data transfer instruction

MOVCO src, dest

Instruction Code
Page: 324

Operation

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

Function

- When the LI flag is 1, data in src (register) is stored in dest (memory) and the LI flag and src are cleared to 0.
- This instruction will not store the data when the LI flag is 0, but sets src to 1.

Notes: 1. The LI flag is in the inside of CPU. The bit can be accessed only by MOVCO, MOVLI, RTE or RTFI instruction. Customer can not access the LI flag directly.
2. Before executing the MOVCO instruction, execute the MOVLI instruction for the same address.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

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

MOVLI

Load data with LI flag set

MOVLI

Syntax

Data transfer instruction

MOVLI src, dest

Instruction Code

Page: 324

Operation

LI = 1;
dest = src;

Function

- This instruction transfers the longword value from src (in memory) to dest (a register), and sets the LI flag to 1.

Notes: 1. The LI flag is cleared when the conditions below are satisfied.
When an MOVCO instruction is executed
When an RTE or RTFI instruction is executed.
2. The LI flag is in the inside of CPU. The bit can be accessed only by MOVCO, MOVLI, RTE or RTFI instruction. Customer can not access the LI flag directly.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

MOVLI [R1], R2

MOVU

Move unsigned

MOVU

Syntax

Data transfer instruction

MOVU.size src, dest

Instruction Code
Page: 325

Operation

dest = src;

Function

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

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

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

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

Description Example

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

MSBHI

Multiply-Subtract the upper words

MSBHI

Syntax

DSP instruction

MSBHI src, src2, Adest

Instruction Code

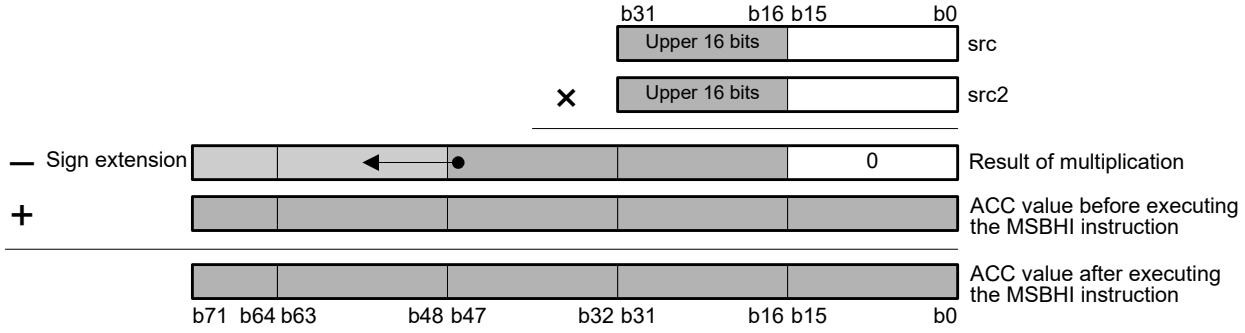
Page: 326

Operation

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

Function

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



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

MSBHI R1, R2, A1

MSBLH

Multiply-Subtract the lower and upper words

MSBLH

DSP instruction

Instruction Code
Page: 326

Syntax

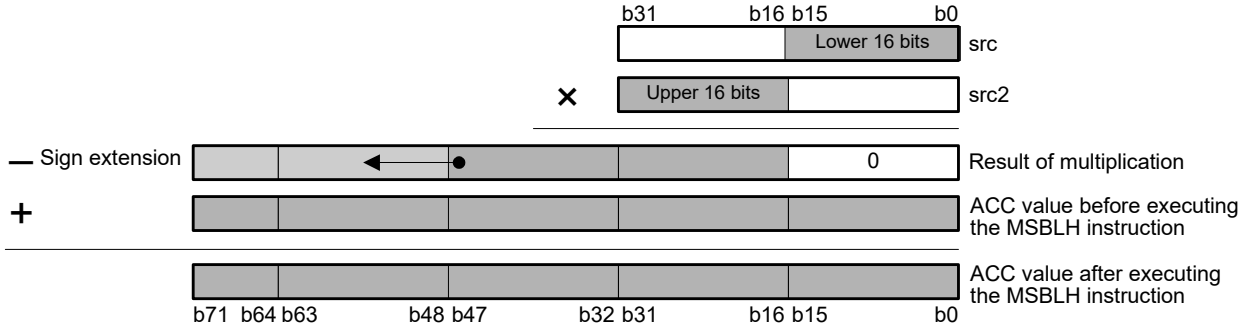
MSBLH src, src2, Adest

Operation

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

Function

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



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

MSBLH R1, R2, A1

MSBLO

Multiply-Subtract the lower words

MSBLO

Syntax

DSP instruction

MSBLO src, src2, Adest

Instruction Code

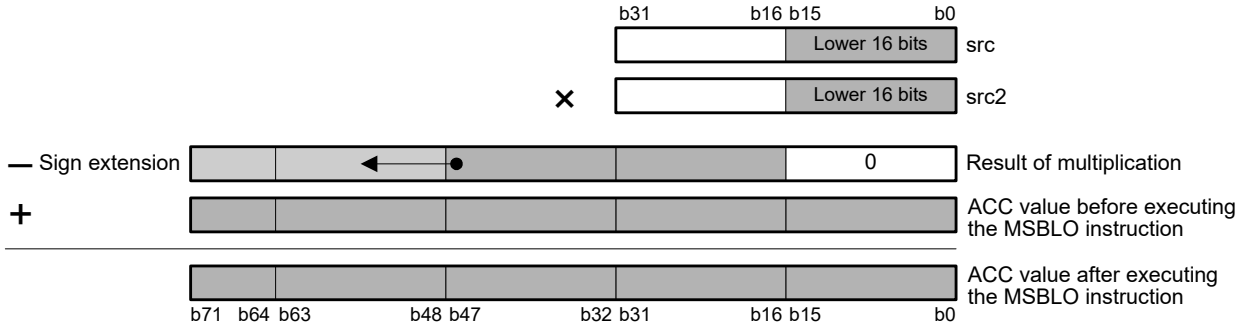
Page: 327

Operation

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

Function

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



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

MSBLO R1, R2, A1

MUL

Multiply

MUL

Syntax

Arithmetic/logic instruction

(1) MUL src, dest

(2) MUL src, src2, dest

Instruction Code
Page: 327

Operation

(1) dest = src * dest;

(2) dest = src * src2;

Function

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

- The calculation is performed in 32 bits and the lower 32 bits of the result are placed.
- The operation result will be the same whether a singed or unsigned multiply is executed.

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

- The calculation is performed in 32 bits and the lower 32 bits of the result are placed.
- The operation result will be the same whether a singed or unsigned multiply is executed.

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

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

Description Example

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

MULHI

Multiply the upper words

MULHI

Syntax

DSP instruction

MULHI src, src2, Adest

Instruction Code

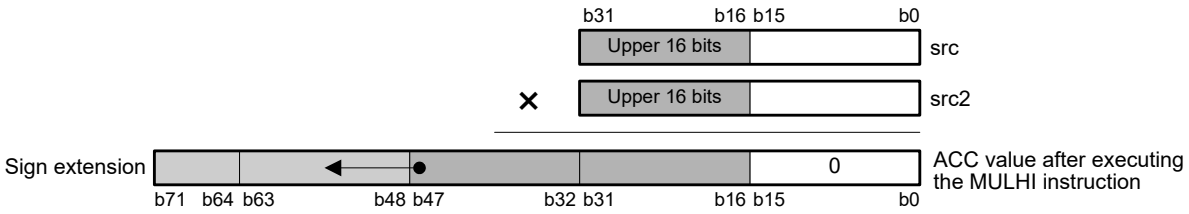
Page: 329

Operation

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

Function

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



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

MULHI R1, R2, A1

MULLH

Multiply the lower and upper words

MULLH

Syntax

DSP instruction

MULLH src, src2, Adest

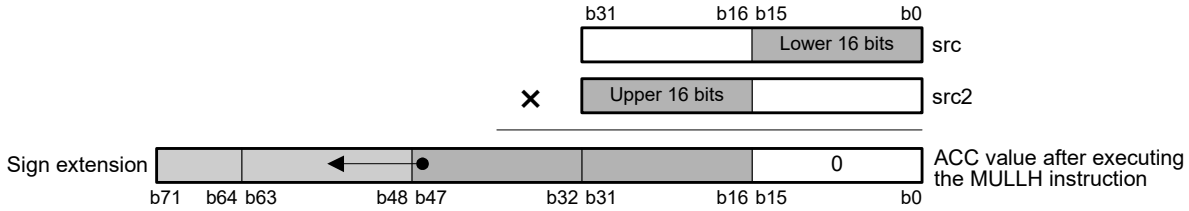
Instruction Code
Page: 329

Operation

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

Function

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



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

MULLH R1, R2, A1

MULLO

Multiply the lower words

MULLO

Syntax

DSP instruction

MULLO src, src2, Adest

Instruction Code

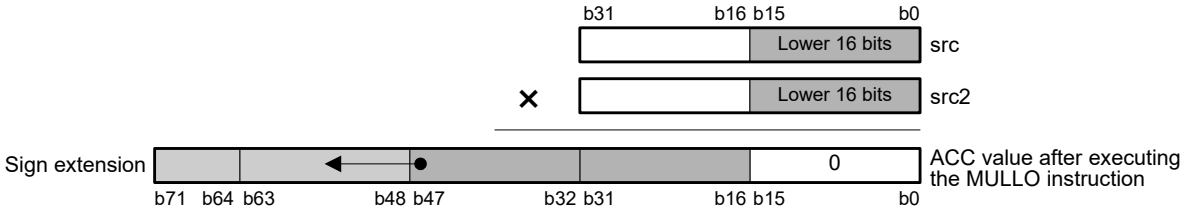
Page: 330

Operation

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

Function

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



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

MULLO R1, R2, A1

MVFACGU

Move data from the guard bits of the accumulator

MVFACGU

Syntax

DSP instruction

MVFACGU src, Asrc, dest

Instruction Code

Page: 330

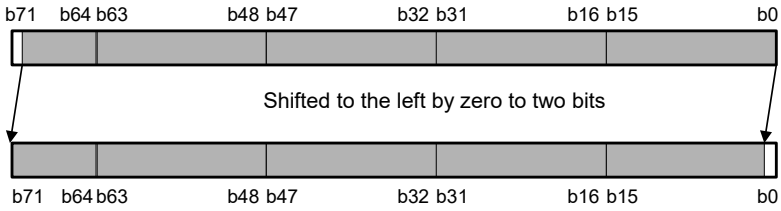
Operation

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

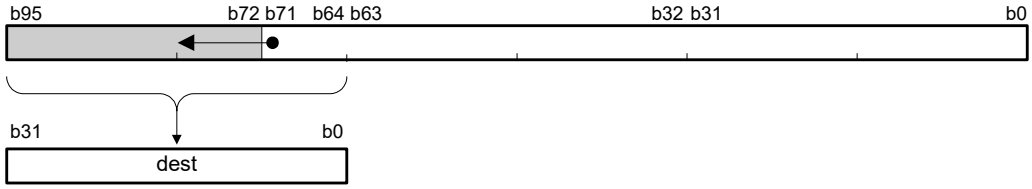
Function

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

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



Processing 2:
Contents of the most significant 32 bits of the value after shifting are moved to dest.



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

MVFACGU #1, A1, R1

MVFACHI

Move data from the upper longword of the accumulator

MVFACHI

Syntax

DSP instruction

MVFACHI src, Asrc, dest

Instruction Code
Page: 331

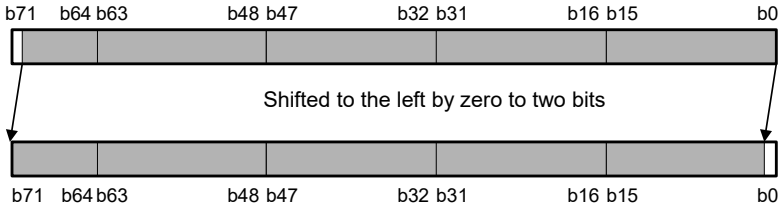
Operation

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

Function

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

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



Processing 2:
Contents of bits 63 to 32 of the value after shifting are moved to dest.



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

MVFACHI #1, A1, R1

MVFACLO

Move data from the lower longword of the accumulator

MVFACLO

Syntax

DSP instruction

MVFACLO src, Asrc, dest

Instruction Code
Page: 331

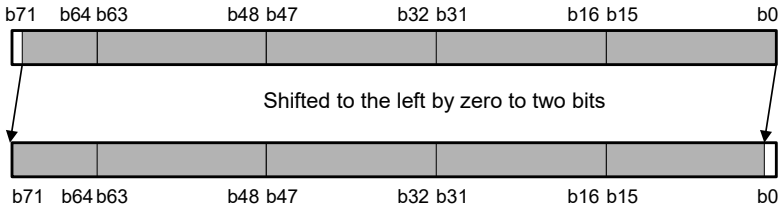
Operation

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

Function

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

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



Processing 2:
Contents of bits 31 to 0 of the value after shifting are moved to dest.



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

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

MVFACMI

Move data from the middle-order longword of the accumulator

MVFACMI

Syntax

DSP instruction

MVFACMI src, Asrc, dest

Instruction Code

Page: 332

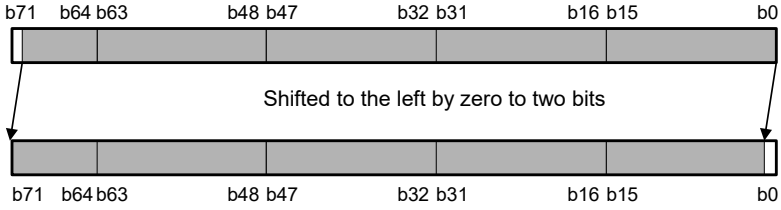
Operation

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

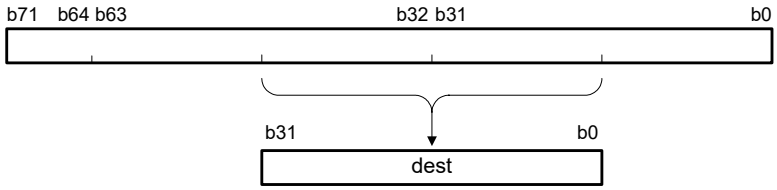
Function

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

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



Processing 2:
Contents of bits 47 to 16 of the value after shifting are moved to dest.



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

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

MVFC

Move data from a control register

MVFC

Syntax

System manipulation instruction

MVFC src, dest

Instruction Code

Page: 333

Operation

dest = src;

Function

- This instruction transfers src to dest.
- When the PC is specified as src, this instruction transfers its own address to dest.
- The value read from the FINTV register is 0 in the secure or non-secure world.
- The value read from the WPRI register is 0 in the non-secure world.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

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

Description Example

MVFC USP, R1

MVTACGU

Move data to the guard bits of the accumulator

MVTACGU

Syntax

DSP instruction

MVTACGU src, Adest

Instruction Code

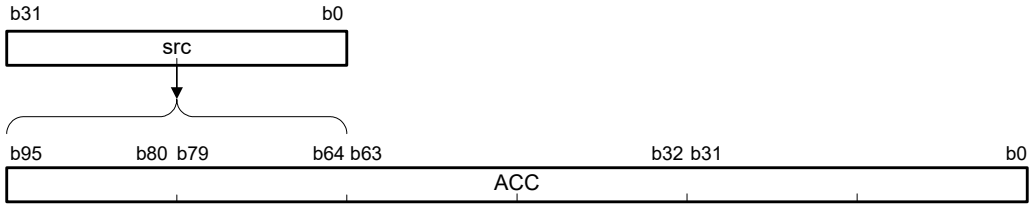
Page: 334

Operation

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

Function

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



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

MVTACGU R1, A1

MVTACHI

Move data to the upper longword of the accumulator

MVTACHI

Syntax

DSP instruction

MVTACHI src, Adest

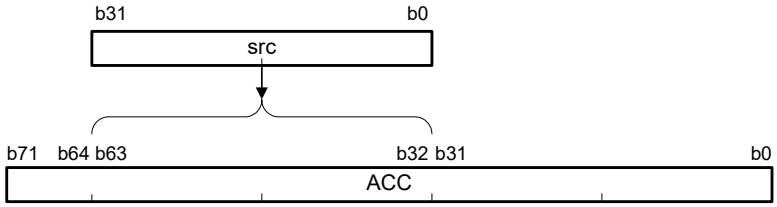
Instruction Code
Page: 334

Operation

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

Function

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



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

MVTACHI R1, A1

MVTACLO

Move data to the lower longword of the accumulator

MVTACLO

Syntax

DSP instruction

MVTACLO src, Adest

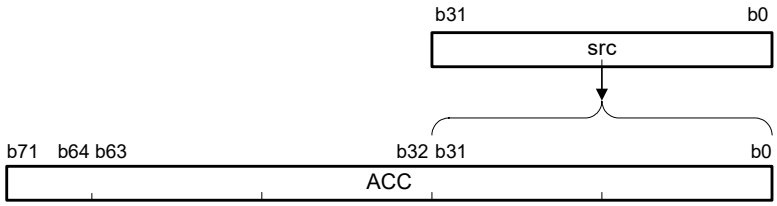
Instruction Code
Page: 334

Operation

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

Function

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



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

MVTACLO R1, A1

MVTC *Move data to a control register* MVTC

Syntax

System manipulation instruction

MVTC src, dest

Instruction Code

Page: 335

Operation

dest = src;

Function

- This instruction transfers src to dest.
- In user mode, writing to the ISP, INTB, EXTB, BPC, BPSW, FINTV, IVC, WID, and WPRI, and the IPL[4:0], PM, U, and I bits in the PSW is ignored. In supervisor mode, writing to the PM bit in the PSW is ignored.
- When the PSW or BPSW is written to in the non-secure world, 0 is always written to the IPL[4] bit.
- Writing to the FINTV, IVC, or WID register is ignored in the secure or non-secure world.
- Writing to the WPRI register is ignored in the non-secure world.

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

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

Description Example

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

MVTIPL

Move data to IPL

MVTIPL

Syntax

System manipulation instruction

MVTIPL src

Instruction Code
Page: 336

Operation

```
if ( WID.WID == 2 )
   IPL = src & 0Fh;
else
   IPL = src;
```

Function

- This instruction transfers src to the IPL[4:0] 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 ≤ 31.
- In the non-secure world, 0 is written to the IPL[4] bit.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

	Operand	
Syntax	**src**	**Code Size (Byte)**
MVTIPL src	#IMM:5	3

Description Example

```
MVTIPL #2
```

NEG

Negate (two's complement)

NEG

Syntax

Arithmetic/logic instruction

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

Instruction Code
Page: 337

Operation

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

Function

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

Flag Change

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

Instruction Format

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

Description Example

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

NOP

No operation

NOP

Syntax

Arithmetic/logic instruction

NOP

Instruction Code
Page: 337

Operation

/* No operation */

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Code Size (Byte)
NOP	1

Description Example

NOP

NOT

Logical NOT (one's complement)

NOT

Syntax

Arithmetic/logic instruction

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

Instruction Code
Page: 338

Operation

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

Function

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

Flag Change

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

Instruction Format

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

Description Example

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

OR

Logical OR

OR

Syntax

Arithmetic/logic instruction

(1) OR src, dest

(2) OR src, src2, dest

Instruction Code
Page: 339

Operation

(1) dest = dest | src;

(2) dest = src2 | src;

Function

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

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

Flag Change

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

Instruction Format

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

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

Description Example

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

POP

Pop register from stack

POP

Syntax

Data transfer instruction

POP dest

Instruction Code
Page: 340

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

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

Description Example

```
POP R1
```

POPC

Pop a control register from stack

POPC

Syntax

POPC dest

Data transfer instruction

Instruction Code

Page: 341

Operation

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

Function

- This instruction restores data from the stack and transfers it to the control register specified as dest.
- The stack pointer in use is specified by the U bit in the PSW.
- In user mode, writing to the ISP, INTB, EXTB, BPC, BPSW, FINTV, IVC, WID, and WPRI, and the IPL[4:0], PM, U, and I bits in the PSW is ignored. In supervisor mode, writing to the PM bit in the PSW is ignored.
- When writing to the PSW or BPSW in the non-secure world, 0 is always written to the IPL[4] bit.
- Writing to the FINTV, IVC, or WID register is ignored in the secure or non-secure world.
- Writing to the WPRI register is ignored in the non-secure world.

Flag Change

Flag	Change	Condition
C	*	
Z	*	
S	*	
O	*	

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

Instruction Format

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

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

Description Example

POPC PSW

POPM

Pop multiple registers from stack

POPM

Syntax

Data transfer instruction

POPM dest-dest2

Instruction Code

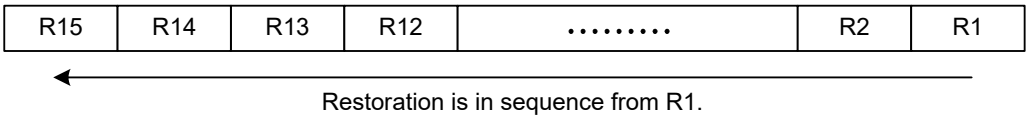
Page: 341

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:



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

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

PUSH

Push register on stack

PUSH

Syntax

Data transfer instruction

PUSH.size src

Instruction Code
Page: 342

Operation

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

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

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

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

Description Example

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

PUSHC

Push a control register on stack

PUSHC

Syntax

Data transfer instruction

PUSHC src

Instruction Code
Page: 343

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.
- The value read from the FINTV register is 0 in the secure or non-secure world.
- The value read from the WPRI register is 0 in the non-secure world.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

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

Description Example

```
PUSHC PSW
```

PUSHM

Push multiple registers on stack

PUSHM

Syntax

Data transfer instruction

PUSHM src-src2

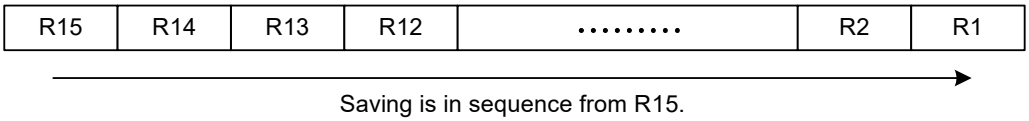
Instruction Code
Page: 343

Operation

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

Function

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



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

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

RACL

Round the accumulator longword

RACL

Syntax

DSP instruction

RACL src, Adest

Instruction Code
Page: 344

Operation

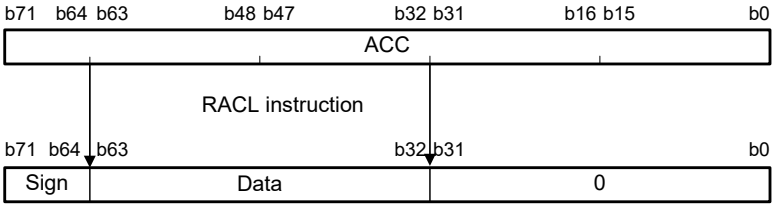
```
signed 72bit tmp;
signed 73bit tmp73;

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

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

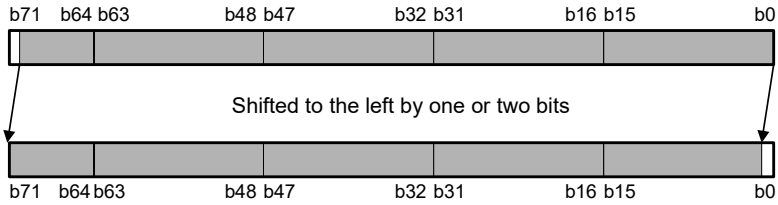
Function

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

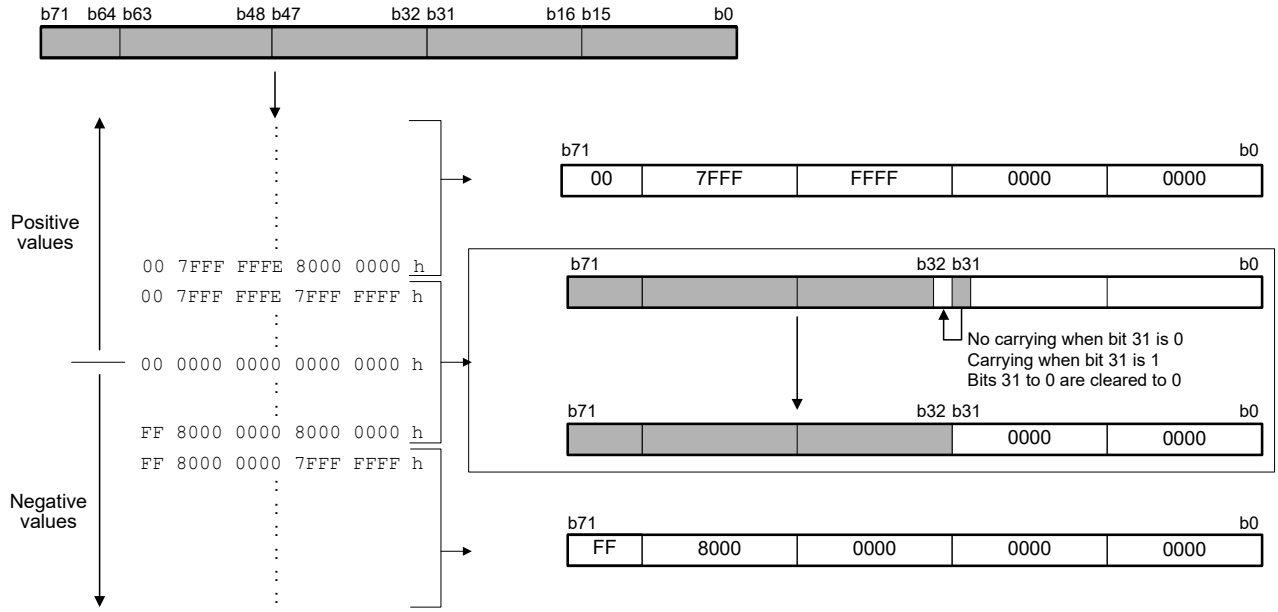


- The RACL instruction is executed according to the following procedures.

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



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



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

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

RACW

Round the accumulator word

RACW

DSP instruction

Instruction Code

Page: 344

Syntax

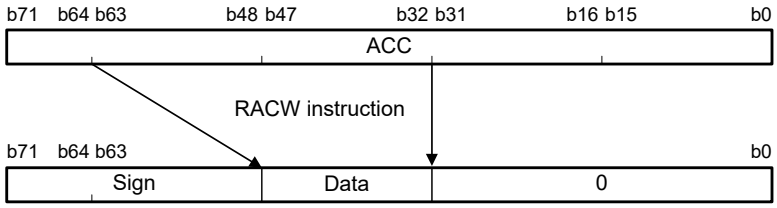
RACW src, Adest

Operation

```
signed 72bit tmp;
signed 73bit tmp73;
tmp = (signed 72bit) Adest << src;
tmp73 = (signed 73bit) tmp + 0000000000080000000h;
if (tmp73 > (signed 73bit) 00000007FFF00000000h)
   Adest = 0000007FFF00000000h;
else if (tmp73 < (signed 73bit) 1FFFFFF800000000000h)
   Adest = FFFFFF800000000000h;
else
   Adest = tmp73 & FFFFFFFFFF00000000h;
```

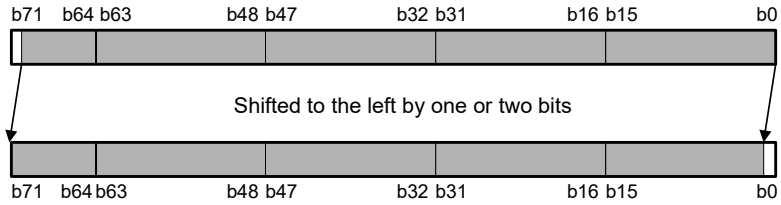
Function

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

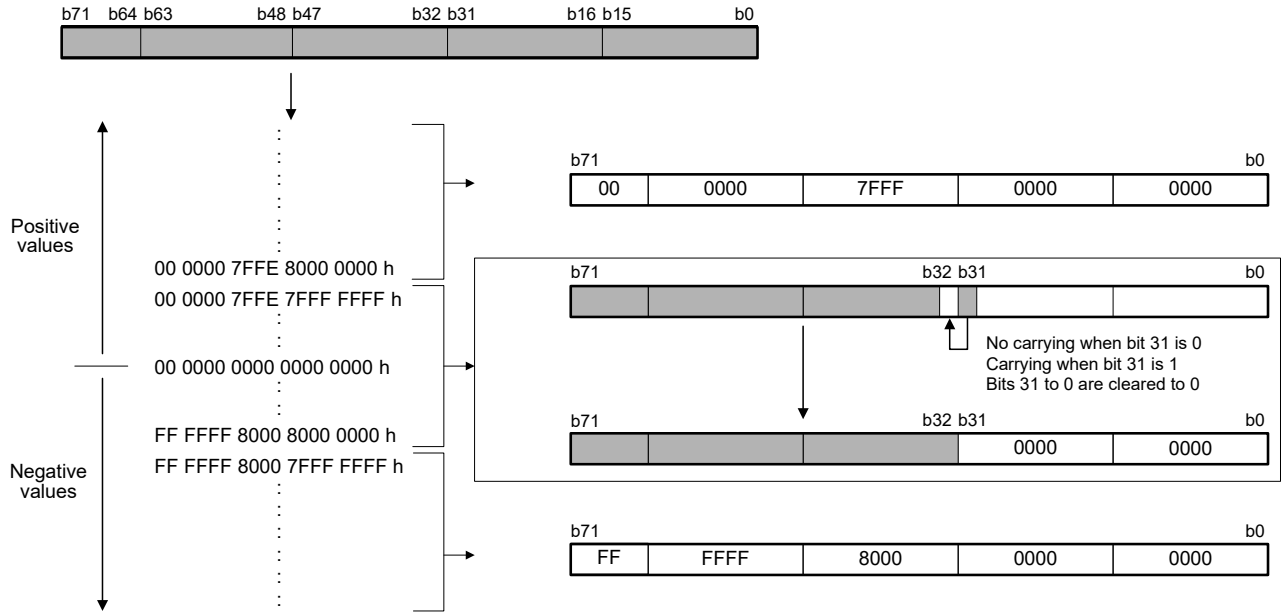


- The RACW instruction is executed according to the following procedures.

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



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



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

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

RDACL

Round down the accumulator longword

RDACL

Syntax

DSP instruction

RDACL src, Adest

Instruction Code
Page: 345

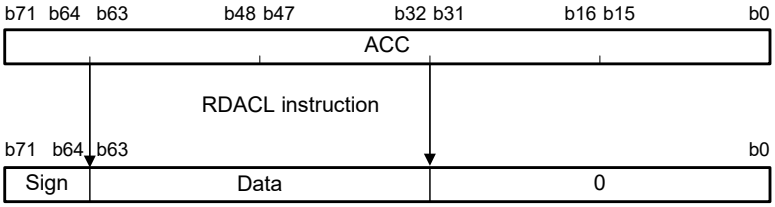
Operation

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

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

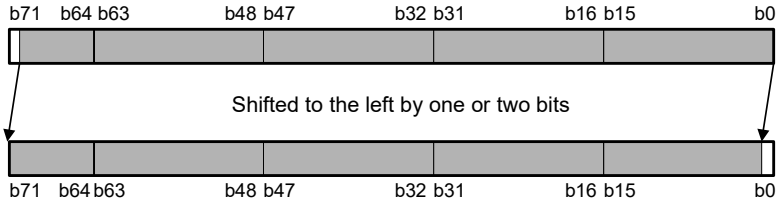
Function

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

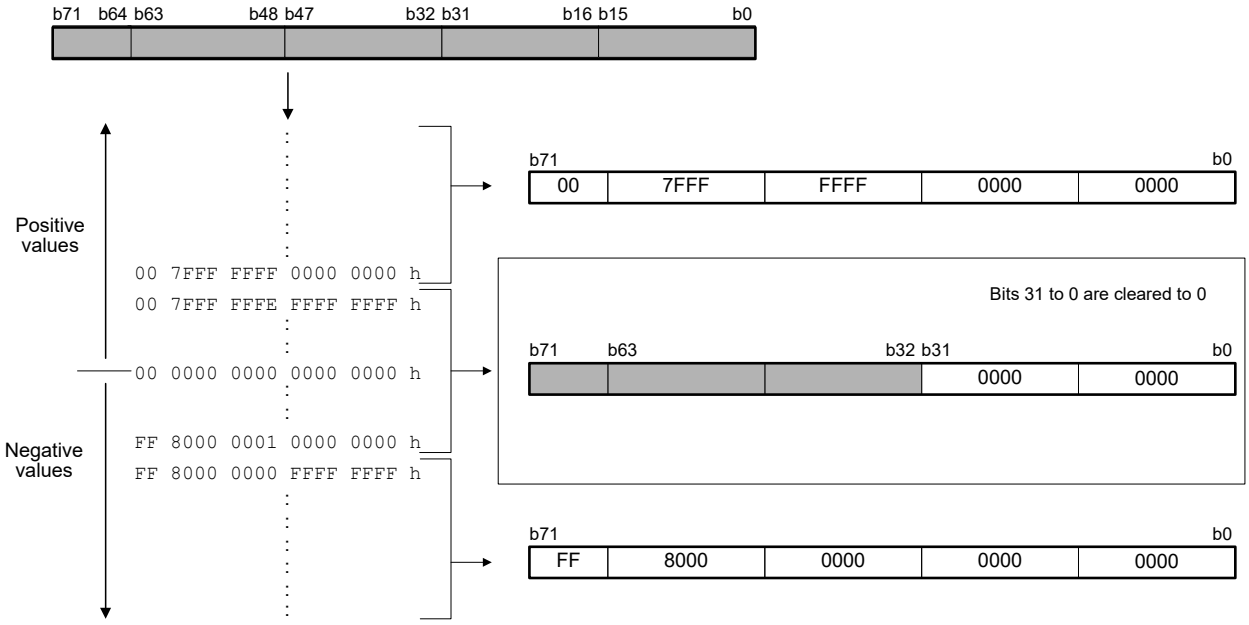


- The RDACL instruction is executed according to the following procedures.

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



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



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

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

RDACW

Round down the accumulator word

RDACW

Syntax

DSP instruction

RDACW src, Adest

Instruction Code
Page: 345

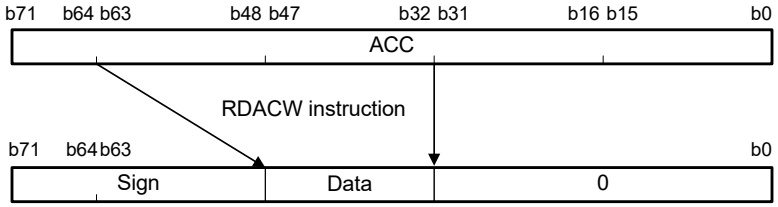
Operation

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

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

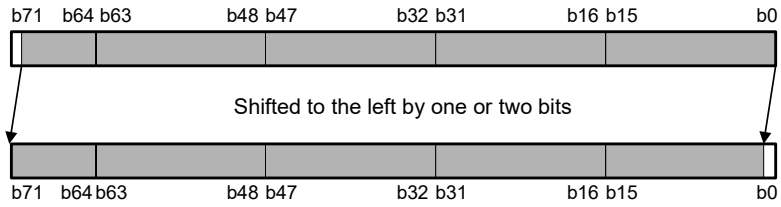
Function

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

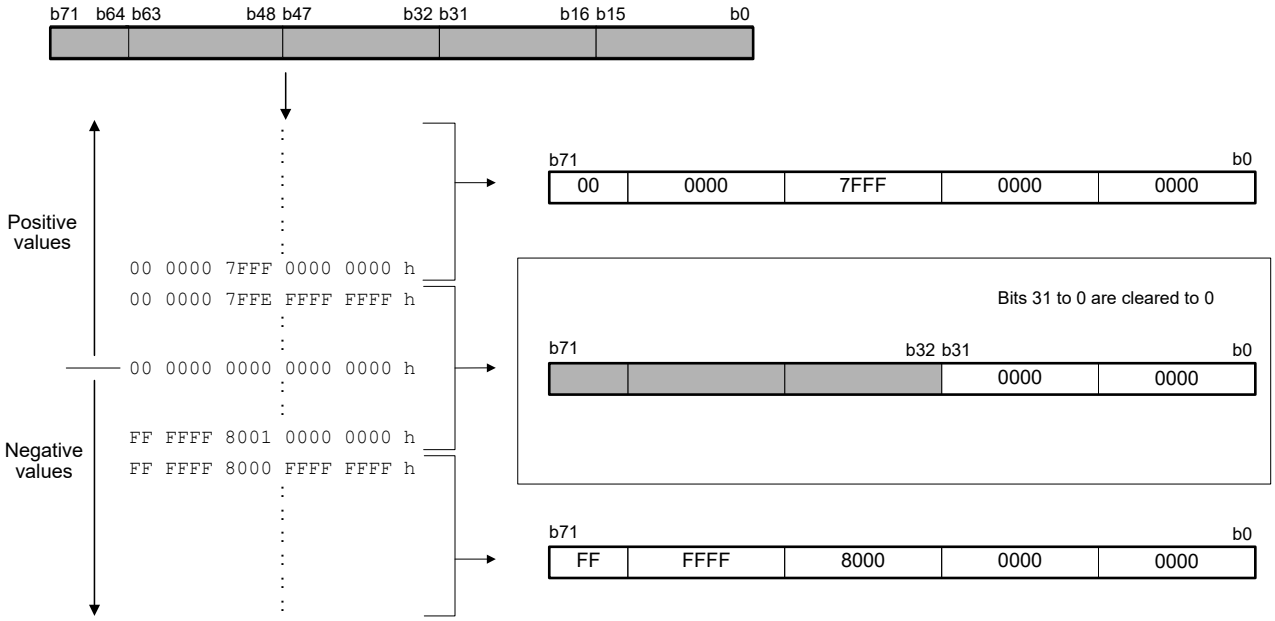


- The RDACW instruction is executed according to the following procedures.

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



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



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

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

REVL

Reverse endian within longword

REVL

Syntax

Data transfer instruction

REVL src, dest

Instruction Code
Page: 346

Operation

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

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

REVL R1, R2

REVW

Reverse endian within word

REVW

Syntax

Data transfer instruction

REVW src, dest

Instruction Code
Page: 346

Operation

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

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

REVW R1, R2

RMPA

Repeat multiply-accumulate

RMPA

Syntax

Arithmetic/logic instruction

RMPA.size

Instruction Code
Page: 347

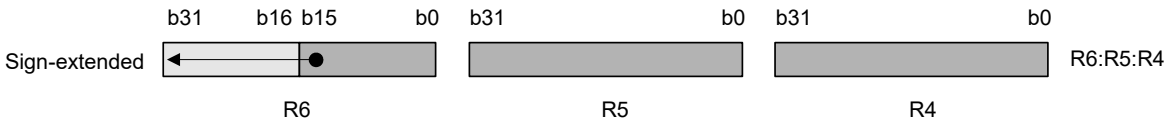
Operation

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

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

Function

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



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

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

Flag Change

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

Instruction Format

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

Description Example

RMPA.W

ROLC

Rotate left with carry

ROLC

Syntax

Arithmetic/logic instruction

ROLC dest

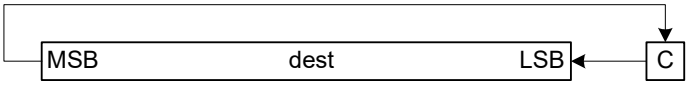
Instruction Code
Page: 347

Operation

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

Function

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



Flag Change

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

Instruction Format

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

Description Example

ROLC R1

RORC

Rotate right with carry

RORC

Arithmetic/logic instruction

Instruction Code
Page: 348

Syntax

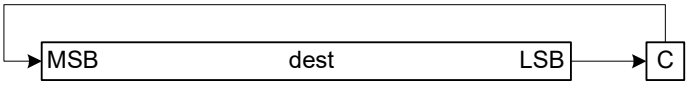
RORC dest

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

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

Description Example

RORC R1

ROTL

Rotate left

ROTL

Syntax

Arithmetic/logic instruction

ROTL src, dest

Instruction Code

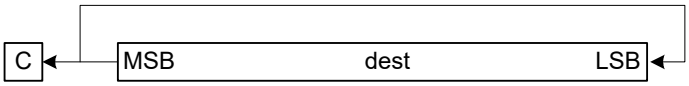
Page: 348

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

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

Description Example

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

ROTR

Rotate right

ROTR

Syntax

Arithmetic/logic instruction

ROTR src, dest

Instruction Code

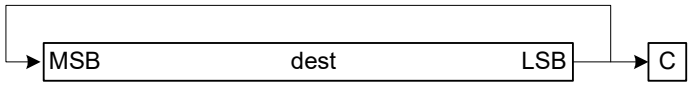
Page: 349

Operation

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

Function

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



Flag Change

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

Instruction Format

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

Description Example

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

ROUND

Round single-precision floating-point to signed integer

ROUND

Syntax

ROUND src, dest

Single-precision floating-point arithmetic instruction

Instruction Code

Page: 350

Operation

dest = (signed long) src;

Function

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

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

Flag Change

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

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

Instruction Format

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

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

Sources of Single-Precision Floating-Point Exceptions

Unimplemented processing
Invalid operation
Inexact

Description Example

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

Supplementary Description

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

When DN = 0

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

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

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

When DN = 1

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

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

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

RTE

Return from exception

RTE

Syntax

System manipulation instruction

RTE

Instruction Code
Page: 350

Operation

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

Function

- This instruction returns execution from the exception handling routine by restoring the PC and PSW contents that were saved when the exception was accepted.
- This instruction is a privileged instruction. Attempting to execute this instruction in user mode generates a privileged instruction exception.
- If returning is accompanied by a transition to user mode, the U bit in the PSW becomes 1.
- When this instruction is executed in the non-secure world, 0 is always written to the PSW.IPL[4] bit.

Flag Change

Flag	Change	Condition
C	*	
Z	*	
S	*	
O	*	

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

Instruction Format

Syntax	Code Size (Byte)
RTE	2

Description Example

```
RTE
```

RTFI

Return from fast interrupt

RTFI

Syntax

System manipulation instruction

RTFI

Instruction Code
Page: 351

Operation

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

Function

- This instruction returns execution from the fast-interrupt processing routine by restoring the PC and PSW contents that were saved in the BPC and BPSW when the fast interrupt request was accepted.
- This instruction is a privileged instruction. Attempting to execute this instruction in user mode generates a privileged instruction exception.
- If returning is accompanied by a transition to user mode, the U bit in the PSW becomes 1.
- The data in the BPC and BPSW are undefined when instruction execution is completed.
- When this instruction is executed in the non-secure world, 0 is always written to the PSW.IPL[4] bit.

Flag Change

Flag	Change	Condition
C	*	
Z	*	
S	*	
O	*	

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

Instruction Format

Syntax	Code Size (Byte)
RTFI	2

Description Example

```
RTFI
```

RTS

Return from subroutine

RTS

Syntax

RTS

Branch instruction

Instruction Code

Page: 351

Operation

PC = *SP;
SP = SP + 4;

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Code Size (Byte)
RTS	1

Description Example

RTS

RTSD

Return from subroutine after deallocating stack frame

RTSD

Syntax

Branch instruction

(1) RTSD src

(2) RTSD src, dest-dest2

Instruction Code
Page: 351

Operation

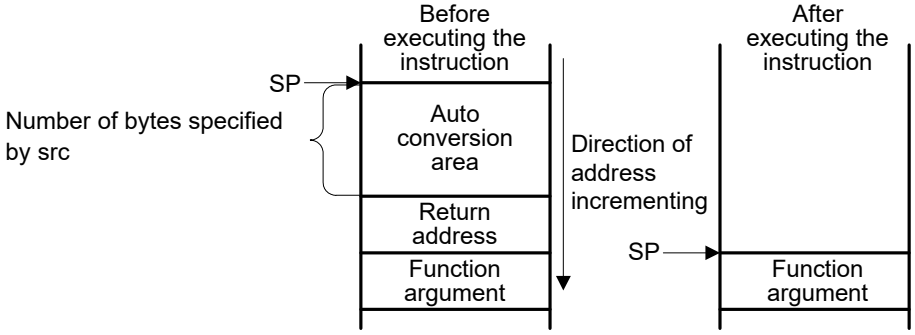
(1)
```
SP = SP + src;
PC = *SP;
SP = SP + 4;
```

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

Function

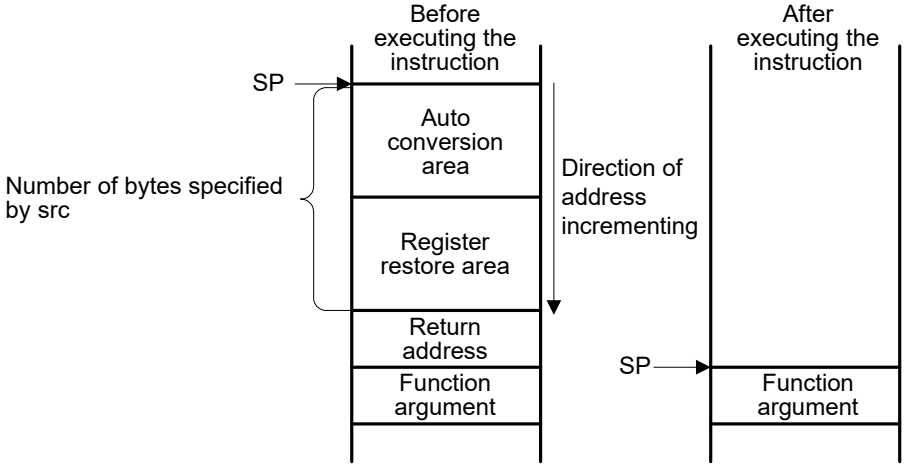
(1) This instruction returns the flow of execution from a subroutine after deallocating the stack frame for the subroutine.

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

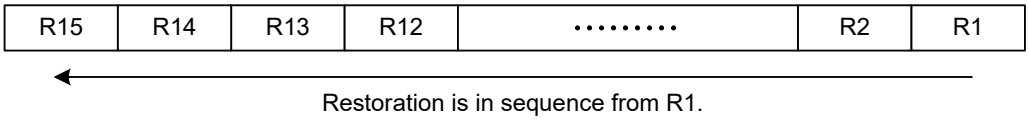


(2) This instruction returns the flow of execution from a subroutine after deallocating the stack frame for the subroutine and also restoring register values from the stack.

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



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



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

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

Description Example

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

SAT

Saturate

SAT

Syntax

Arithmetic/logic instruction

SAT dest

Instruction Code
Page: 352

Operation

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

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

```
SAT    R1
```

SATR

Saturate for RMPA

SATR

Syntax

Arithmetic/logic instruction

SATR

Instruction Code
Page: 352

Operation

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

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Code Size (Byte)
SATR	2

Description Example

```
SATR
```

SBB

Subtract with borrow

SBB

Syntax

Arithmetic/logic instruction

SBB src, dest

Instruction Code
Page: 353

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

Description Example

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

SC*Cnd*

Store condition

SC*Cnd*

Syntax

Data transfer instruction

SC*Cnd*.size dest

Instruction Code
Page: 354

Operation

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

Function

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

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

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

Description Example

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

SCMPU *String compare until not equal* SCMPU

Syntax

String manipulation instruction

SCMPU

Instruction Code
Page: 354

Operation

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

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

Function

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

Flag Change

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

Instruction Format

Syntax	Processing Size	Code Size (Byte)
SCMPU	B	2

Description Example

SCMPU

SETPSW *Set a flag or bit in the PSW* SETPSW

Syntax

System manipulation instruction

SETPSW dest

Instruction Code
Page: 355

Operation

dest = 1;

Function

- This instruction sets the O, S, Z, or C flag or the U or I bit specified by dest to 1.
- In user mode, writing to the U or I bit in the PSW will be ignored. In supervisor mode, all flags and bits can be written to.

Flag Change

Flag	Change	Condition
C	*	
Z	*	
S	*	
O	*	

Note: * The specified flag is set to 1.

Instruction Format

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

Description Example

```
SETPSW C
SETPSW Z
```

SHAR *Arithmetic shift right* SHAR

Syntax

Arithmetic/logic instruction

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

Instruction Code
Page: 356

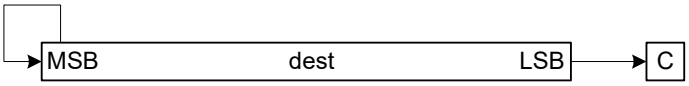
Operation

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

Function

(1) This instruction arithmetically shifts dest to the right by the number of bit positions specified by src and saves the value in dest.
- Bits overflowing from the LSB are transferred to the C flag.
- src is an unsigned in the range of $0 \leq src \leq 31$.
- When src is in register, only five bits in the LSB are valid.

(2) After this instruction transfers src2 to dest, it arithmetically shifts dest to the right by the number of bit positions specified by src and saves the value in dest.
- Bits overflowing from the LSB are transferred to the C flag.
- src is an unsigned integer in the range of $0 \leq src \leq 31$.



Flag Change

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

Instruction Format

Syntax	Processing Size	Operand src	Operand src2	Operand dest	Code Size (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 shift left

SHLL

Syntax

Arithmetic/logic instruction

(1) SHLL src, dest

(2) SHLL src, src2, dest

Instruction Code
Page: 357

Operation

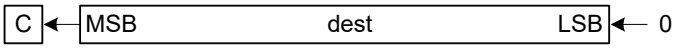
(1) dest = dest << (src & 31);

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

Function

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

(2) After this instruction transfers src2 to dest, it arithmetically shifts dest to the left by the number of bit positions specified by src and saves the value in dest.
- Bits overflowing from the MSB are transferred to the C flag.
- src is an unsigned integer in the range of 0 ≤ src ≤ 31.



Flag Change

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

Instruction Format

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

Description Example

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

SHLR *Logical shift right* SHLR

Syntax

Arithmetic/logic instruction

(1) SHLR src, dest

(2) SHLR src, src2, dest

Instruction Code
Page: 358

Operation

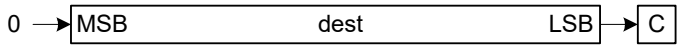
(1) dest = (unsigned long) dest >> (src & 31);

(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

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

Description Example

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

SMOVB

String move backward

SMOVB

String manipulation instruction

Syntax

SMOVB

Instruction Code
Page: 359

Operation

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

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

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Processing Size	Code Size (Byte)
SMOVB	B	2

Description Example

SMOVB

SMOVF

String move forward

SMOVF

Syntax

String manipulation instruction

SMOVF

Instruction Code

Page: 359

Operation

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

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

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Processing Size	Code Size (Byte)
SMOVF	B	2

Description Example

SMOVF

SMOVU *String move until zero detected* SMOVU

Syntax

String manipulation instruction

SMOVU

Instruction Code

Page: 359

Operation

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

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

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Processing Size	Code Size (Byte)
SMOVU	B	2

Description Example

SMOVU

SSTR

String store

SSTR

Syntax

String manipulation instruction

SSTR.size

Instruction Code
Page: 360

Operation

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

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

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

SSTR.W

STNZ

Store on not zero

STNZ

Syntax

Data transfer instruction

STNZ src, dest

Instruction Code

Page: 360

Operation

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

Function

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Description Example

```
STNZ #1, R2
STNZ R1, R2
```

STZ

Store on zero

STZ

Syntax

Data transfer instruction

STZ src, dest

Instruction Code
Page: 361

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
	L	Rs	Rd	3

Description Example

```
STZ    #1, R2
STZ    R1, R2
```

SUB

Subtract without borrow

SUB

Syntax

Arithmetic/logic instruction

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

Instruction Code
Page: 362

Operation

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

Function

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

Flag Change

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

Instruction Format

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

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

Description Example

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

SUNTIL

String search until equal

SUNTIL

Syntax

String manipulation instruction

SUNTIL.size

Instruction Code
Page: 363

Operation

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

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

Function

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

Flag Change

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

Instruction Format

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

Description Example

SUNTIL.W

SWHILE *String search while equal* SWHILE

Syntax

String manipulation instruction

SWHILE.size

Instruction Code
Page: 363

Operation

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

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

Function

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

Flag Change

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

Instruction Format

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

Description Example

SWHILE.W

TST

Test logical

TST

Syntax

Arithmetic/logic instruction

TST src, src2

Instruction Code
Page: 364

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		Code Size (Byte)
		src	src2	
TST src, src2	L	#SIMM:8	Rs	4
	L	#SIMM:16	Rs	5
	L	#SIMM:24	Rs	6
	L	#IMM:32	Rs	7
	L	Rs	Rs2	3
	L	[Rs].memex	Rs2	3 (memex == “UB”) 4 (memex != “UB”)
	L	dsp:8[Rs].memex*	Rs2	4 (memex == “UB”) 5 (memex != “UB”)
	L	dsp:16[Rs].memex*	Rs2	5 (memex == “UB”) 6 (memex != “UB”)

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

Description Example

```
TST   #7, R2
TST   R1, R2
TST   [R1], R2
TST   1[R1].UB, R2
```

UTOF

Convert unsigned integer to single-precision floating-point

UTOF

Syntax

UTOF src, dest

Single-precision floating-point arithmetic instruction

Instruction Code
Page: 365

Operation

dest = (float) (unsigned long) src;

Function

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

Flag Change

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

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

Instruction Format

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

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

Sources of Single-Precision Floating-Point Exceptions

Inexact

Description Example

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

WAIT

Wait

WAIT

Syntax

System manipulation instruction

WAIT

Instruction Code

Page: 366

Operation

Function

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

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Code Size (Byte)
WAIT	2

Description Example

WAIT

XCHG

Exchange

XCHG

Syntax

Data transfer instruction

XCHG src, dest

Instruction Code
Page: 366

Operation

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

Function

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

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

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

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

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

Description Example

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

XOR

Logical Exclusive OR

XOR

Syntax

Arithmetic/logic instruction

(1) XOR src, dest

(2) XOR src, src2, dest

Instruction Code
Page: 367

Operation

(1) dest = dest ^ src;

(2) dest = src2 ^ src;

Function

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

(2) This instruction exclusive ORs src2 and src and places the result in dest.

Flag Change

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

Instruction Format

Syntax		Processing Size	Operand src	Operand src2	Operand dest	Code Size (Byte)
(1) 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”)
(2) XOR	src, src2, dest	L	Rs	Rs2	Rd	3

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

Description Example

```
XOR   #8, R1
XOR   R1, R2
XOR   [R1], R2
XOR   16[R1].L, R2
XOR   R1, R2, R3
```

3.4.2 Instructions for Register Bank Save Function

The following pages give details of the instructions for register bank save function.

RSTR

Restore registers collectively

RSTR

Syntax

Instructions for register bank save function

RSTR src

Instruction Code
Page: 370

Operation

{ R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R15, USP, FPSW, ACC0, ACC1 } = bank(src);

Function

- This instruction collectively restores the values in a register-saving bank to CPU registers (R1 to R15, USP, FPSW, ACC0, and ACC1).
- The bank number of the source for restoration is specified by src.
- This instruction is privileged. Attempting to execute it in user mode leads to a privileged instruction exception.
- For the availability of register-saving banks and the range of bank numbers (which is determined by the capacity of the memory installed for use as register-saving banks), refer to the user's manual: hardware for each product.
- When a bank number that does not exist is specified, the operation is not defined.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

		Operand	
Syntax	**Processing Size**	**src**	**Code Size (Byte)**
RSTR src	—	#UIMM:8	4
	—	Rs	4

Description Example

```
RSTR #5
RSTR R1
```

SAVE

Save registers collectively

SAVE

Syntax

Instructions for register bank save function

SAVE src

Instruction Code
Page: 370

Operation

bank(src) = { R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R15, USP, FPSW, ACC0, ACC1};

Function

- This instruction collectively saves the values of CPU registers (R1 to R15, USP, FPSW, ACC0, and ACC1) in a register-saving bank.
- The bank number of the destination for saving is specified by src.
- This instruction is privileged. Attempting to execute it in user mode leads to a privileged instruction exception.
- For the availability of register-saving banks and the range of bank numbers (which is determined by the capacity of the memory installed for use as register-saving banks), refer to the user's manual: hardware for each product.
- When a bank number that does not exist is specified, the operation is not defined.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

		Operand	
Syntax	**Processing Size**	**src**	**Code Size (Byte)**
SAVE src	—	#UIMM:8	4
	—	Rs	4

Description Example

```
SAVE  #5
SAVE  R1
```

3.4.3 Double-Precision Floating-Point Instructions

The following pages give details of the double-precision floating-point instructions.

DABS *Absolute value of double-precision floating-point* DABS

Syntax

DABS src, dest

Double-precision floating-point arithmetic instruction

Instruction Code

Page: 372

Operation

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

Function

- This instruction calculates the absolute value of the double-precision floating-point number stored in src and places the result in dest.
- Denormalized numbers are handled in the same way regardless of the setting of the DDN bit in the DPSW.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

			Operand		
Syntax		**Processing Size**	**src**	**dest**	**Code Size (Byte)**
DABS	src, dest	D	DRs	DRd	4

Sources of Double-Precision Floating-Point Exceptions

None

Description Example

DABS DR0, DR1

Supplementary Description

- The following table shows the correspondences between src value and the result of operations

	src								
	+Normalized	–Normalized	+0	–0	+∞	–∞	Denormalized	QNaN	SNaN
Result	+Normalized	+Normalized	+0	+0	+∞	+∞	Positive denormalized numbers	Positive QNaN	Positive SNaN

DADD

Add double-precision floating-point

DADD

Syntax

DADD src, src2, dest

Double-precision floating-point arithmetic instruction

Instruction Code
Page: 372

Operation

dest = src2 + src;

Function

- This instruction adds the double-precision floating-point numbers stored in src2 and src and places the result in dest.
- Rounding of the result is in accordance with the setting of the DRM[1:0] bits in the DPSW.
- Handling of denormalized numbers depends on the setting of the DDN bit in the DPSW.
- The result of the operation is +0 if src and src2 have the opposite signs and their sum is exactly 0, except when the rounding mode is towards –∞. The operation result is –0 when the rounding mode is towards –∞.

Flag Change

Flag	Change	Condition
C	—	
Z	—	
S	—	
O	—	
DCV	✓	The flag is set if an invalid operation exception is generated; otherwise it is cleared.
DCO	✓	The flag is set if an overflow exception is generated; otherwise it is cleared.
DCZ	✓	The value of the flag is 0.
DCU	✓	The flag is set if an underflow exception is generated; otherwise it is cleared.
DCX	✓	The flag is set if an inexact exception is generated; otherwise it is cleared.
DCE	✓	The flag is set if an unimplemented processing is generated; otherwise it is cleared.
DFV	✓	The flag is set if an invalid operation exception is generated, and otherwise left unchanged.
DFO	✓	The flag is set if an overflow exception is generated, and otherwise left unchanged.
DFZ	—	
DFU	✓	The flag is set if an underflow exception is generated, and otherwise left unchanged.
DFX	✓	The flag is set if an inexact exception is generated, and otherwise left unchanged.

Note: The DFX, DFU, DFZ, DFO, and DFV flags do not change if any of the exception enable bits (DEX, DEU, DEZ, DEO, and DEV) in the DPSW is 1.

Instruction Format

Syntax		Processing Size	Operand			Code Size (Byte)
			src	src2	dest	
DADD	src, src2, dest	D	DRs	DRs2	DRd	4

Sources of Double-Precision Floating-Point Exceptions

Unimplemented processing
Invalid operation
Overflow
Underflow
Inexact

Description Example

```
DADD    DR0, DR1, DR2
```

Supplementary Description

- The following tables show the correspondences between the src and src2 values and the results of operations when the value of the DDN bit in the DPSW is 0 or 1.

When DDN = 0

		src							
		Normalized	+0	–0	+∞	–∞	Denormalized	QNaN	SNaN
src2	Normalized	Sum							
	+0		+0	*		–∞			
	–0		*	–0					
	+∞				+∞	Invalid operation			
	–∞		–∞		Invalid operation	–∞			
	Denormalized						Unimplemented processing		
	QNaN							QNaN	
	SNaN								Invalid operation

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

When DDN = 1

		src						
		Normalized	+0, +Denormalized	–0, –Denormalized	+∞	–∞	QNaN	SNaN
src2	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.

DCMPcm

Compare double-precision floating-point

DCMPcm

Double-precision floating-point arithmetic instruction

Instruction Code
Page: 373

Syntax

DCMPcm src, src2

Operation

```
If ( cm == UN )
  RES = isNaN(src) || isNaN(src2);
else if (cm == EQ)
  RES = ( src2 == src );
else if (cm == LT)
  RES = ( src2 < src );
else if (cm == LE)
  RES = ( src2 <= src );
```

Function

- This instruction compares the double-precision floating-point numbers stored in src2 and src based on the condition specified by cm and indicates the result in the RES bit of the DCMR register.
- If the numbers satisfy the condition specified by cm, the RES bit becomes 1, and if not, the bit becomes 0.
- Handling of denormalized numbers depends on the setting of the DDN bit in the DPSW.

Comparison Condition

cm	Definition	Description
UN	Unordered	This condition is for detecting cases where classification of order based on the comparison is impossible.
EQ	src2 == src	This condition is for detecting that src2 is equal to src.
LT	src2 < src	This condition is for detecting that src2 is less than src.
LE	src2 ≤ src	This condition is for detecting that src2 is less than or equal to src.

Flag Change

Flag	Change	Condition
C	—	
Z	—	
S	—	
O	—	
DCV	✓	The flag is set if an invalid operation exception is generated; otherwise it is cleared.
DCO	✓	The value of the flag is 0.
DCZ	✓	The value of the flag is 0.
DCU	✓	The value of the flag is 0.
DCX	✓	The value of the flag is 0.
DCE	✓	The flag is set if an unimplemented processing exception is generated; otherwise it is cleared.
DFV	✓	The flag is set if an invalid operation exception is generated; otherwise it does not change.
DFO	—	
DFZ	—	
DFU	—	
DFX	—	

Note: The DFV flag does not change if the exception enable bit (DEV) in the DPSW is 1.

Instruction Format

			Operand		
Syntax		**Processing Size**	**src**	**src2**	**Code Size (Byte)**
DCMPcm	src, src2	D	DRs	DRs2	4

Sources of Double-Precision Floating-Point Exceptions

Unimplemented processing
Invalid operation

Description Example

```
DCMPEQ  DR0, DR1
```

Supplementary Description

- The following tables show the correspondences between the src and src2 values and the results of operations when the value of the DDN bit in the DPSW is 0 or 1.
  (>: src2 > src, <: src2 < src, =: src2 == src)

When DDN = 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" rowspan="3">Comparison</td><td rowspan="3"><</td><td rowspan="3">></td><td rowspan="6"></td><td rowspan="6"></td><td rowspan="6"></td></tr>
<tr><td>+0</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="5"></td><td>Unimplemented processing</td></tr>
<tr><td>QNaN</td><td colspan="7"></td><td>Ordered classification impossible</td></tr>
<tr><td>SNaN</td><td colspan="8">Invalid operation (Ordered classification impossible)</td></tr>
</table>

When DDN = 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"><</td><td rowspan="3">></td><td rowspan="5"></td><td rowspan="5"></td></tr>
<tr><td>+0,
+Denormalized</td><td colspan="2" rowspan="2">=</td></tr>
<tr><td>–0,
–Denormalized</td></tr>
<tr><td>+∞</td><td colspan="3">></td><td>=</td></tr>
<tr><td>–∞</td><td colspan="4"><</td><td>=</td></tr>
<tr><td>QNaN</td><td colspan="6"></td><td>Ordered classification impossible</td></tr>
<tr><td>SNaN</td><td colspan="7">Invalid operation (Ordered classification impossible)</td></tr>
</table>

DDIV *Divide double-precision floating-point* DDIV

Syntax

DDIV src, src2, dest

Double-precision floating-point arithmetic instruction

Instruction Code
Page: 373

Operation

dest = src2 / src;

Function

- This instruction divides the double-precision floating-point number stored in src2 by that stored in src and places the result in dest.
- Rounding of the result is in accordance with the setting of the DRM[1:0] bits in the DPSW.
- Handling of denormalized numbers depends on the setting of the DDN bit in the DPSW.

Flag Change

Flag	Change	Condition
C	—	
Z	—	
S	—	
O	—	
DCV	✓	The flag is set if an invalid operation exception is generated; otherwise it is cleared.
DCO	✓	The flag is set if an overflow exception is generated; otherwise it is cleared.
DCZ	✓	The flag is set if a division-by-zero exception is generated; otherwise it is cleared.
DCU	✓	The flag is set if an underflow exception is generated; otherwise it is cleared.
DCX	✓	The flag is set if an inexact exception is generated; otherwise it is cleared.
DCE	✓	The flag is set if an unimplemented processing exception is generated; otherwise it is cleared.
DFV	✓	The flag is set if an invalid operation exception is generated; otherwise it does not change.
DFO	✓	The flag is set if an overflow exception is generated; otherwise it does not change.
DFZ	✓	The flag is set if a division-by-zero exception is generated; otherwise it does not change.
DFU	✓	The flag is set if an underflow exception is generated; otherwise it does not change.
DFX	✓	The flag is set if an inexact exception is generated; otherwise it does not change.

Note: The DFX, DFU, DFZ, DFO, and DFV flags do not change if any of the exception enable bits (DEX, DEU, DEZ, DEO, and DEV) in the DPSW is 1.

Instruction Format

Syntax		Processing Size	Operand			Code Size (Byte)
			src	src2	dest	
DDIV	src, src2, dest	D	DRs	DRs2	DRd	4

Sources of Double-Precision Floating-Point Exceptions

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

Description Example

DDIV DR0, DR1, DR2

Supplementary Description

- The following tables show the correspondences between src and src2 values and the results of operations when the value of the DDN bit in the DPSW is 0 or 1.

When DDN = 0

<table>
<tr><td colspan="2" rowspan="2"></td><td colspan="9">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>Division</td><td colspan="2">Division-by-zero</td><td colspan="2">0</td><td rowspan="6">Unimplemented processing</td><td rowspan="5">QNaN</td><td rowspan="4">Invalid operation</td></tr>
<tr><td>+0</td><td rowspan="2">0</td><td colspan="2" rowspan="2">Invalid operation</td><td>+0</td><td>–0</td></tr>
<tr><td>–0</td><td>–0</td><td>+0</td></tr>
<tr><td>+∞</td><td rowspan="2">∞</td><td>+∞</td><td>–∞</td><td colspan="2" rowspan="2">Invalid operation</td></tr>
<tr><td>–∞</td><td>–∞</td><td>+∞</td></tr>
<tr><td>Denormalized</td><td colspan="6"></td></tr>
<tr><td>QNaN</td><td colspan="7"></td><td>QNaN</td></tr>
<tr><td>SNaN</td><td colspan="8"></td><td>Invalid operation</td></tr>
</table>

When DDN = 1

<table>
<tr><td colspan="2" rowspan="2"></td><td colspan="8">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>Division</td><td colspan="2">Division-by-zero</td><td colspan="2">0</td><td rowspan="5">QNaN</td><td rowspan="4">Invalid operation</td></tr>
<tr><td>+0, +Denormalized</td><td rowspan="2">0</td><td colspan="2" rowspan="2">Invalid operation</td><td>+0</td><td>–0</td></tr>
<tr><td>–0, –Denormalized</td><td>–0</td><td>+0</td></tr>
<tr><td>+∞</td><td rowspan="2">∞</td><td>+∞</td><td>–∞</td><td colspan="2" rowspan="2">Invalid operation</td></tr>
<tr><td>–∞</td><td>–∞</td><td>+∞</td></tr>
<tr><td>QNaN</td><td colspan="6"></td><td>QNaN</td></tr>
<tr><td>SNaN</td><td colspan="7"></td><td>Invalid operation</td></tr>
</table>

DMOV *Move double-precision floating-point* DMOV

Syntax

DMOV.size src, dest

Double-precision floating-point data transfer instruction

Instruction Code

Page: 374

Operation

dest = src;

Function

- This instruction transfers src to dest in the ways described in the following table.

src	dest	Function
General-purpose register	Double-precision floating-point data register	The data are transferred from the source register (src) to a double-precision floating-point data register (dest). The size specifier .L or .D can be selected. The size specifier .D can only be selected when the upper 32 bits in the double-precision floating-point data register (dest) are specified. When the upper 32 bits in the double-precision floating-point data register (dest) are specified, 0 is transferred to the lower 32 bits if the size specifier is .D, or the lower 32 bits are retained for the size specifier .L. When the lower 32 bits in the double-precision floating-point data register (dest) are specified, the upper 32 bits are retained.
Double-precision floating-point data register	General-purpose register	The data are transferred from a double-precision floating-point data register (src) to the destination register (dest). Only the size specifier .L can be selected. Either the upper 32 bits or the lower 32 bits of the double-precision floating-point data register (src) are specified.
Double-precision floating-point data register	Double-precision floating-point data register	The data are transferred from one double-precision floating-point data register (src) to another double-precision floating-point data register (dest). Only the size specifier .D can be selected.
Double-precision floating-point data register	Memory location	The data are transferred from a double-precision floating-point data register to locations in memory. Only the size specifier .D can be selected.
Memory location	Double-precision floating-point data register	The data are transferred from memory locations to a double-precision floating-point data register. Only the size specifier .D can be selected.
Immediate value	Double-precision floating-point data register	The immediate value specified as an operand of the instruction is transferred to a double-precision floating-point data register (dest). The size specifier .L or .D can be selected. The size specifier .D can only be selected when the upper 32 bits in the double-precision floating-point data register (dest) are specified. When the upper 32 bits in the double-precision floating-point data register (dest) are specified, 0 is transferred to the lower 32 bits if the size specifier is .D, or the lower 32 bits are retained if the size specifier is .L. When the lower 32 bits in the double-precision floating-point data register (dest) are specified, the upper 32 bits are retained.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Size	Processing Size	Operand src	Operand dest	Code Size (Byte)
DMOV.size src, dest	Data transfer between registers				
	D	D	Rs	DRHd	4
	L	L	Rs	DRHd	4
	L	L	Rs	DRLd	4
	L	L	DRHs	Rd	4
	L	L	DRLs	Rd	4
	D	D	DRs	DRd	4
	Store				
	D	D	DRs	[Rd]	4
	D	D	DRs	dsp:8[Rd]*1	5
	D	D	DRs	dsp:16[Rd]*1	6
	Load				
	D	D	[Rs]	DRd	4
	D	D	dsp:8[Rs]*1	DRd	5
	D	D	dsp:16[Rs]*1	DRd	6
	Set immediate value to register				
	D	D	#IMM:32	DRHd	7
	L	L	#IMM:32	DRHd	7
	L	L	#IMM:32	DRLd	7

Note: 1. For the RX Family assembler manufactured by Renesas Electronics Corp., enter a scaled value (the actual value multiplied by 4 when the size extension specifier is .D) as the displacement value (dsp:8, dsp:16). With dsp:8, values from 0 to 1020 (255 × 4) can be specified when the size specifier is .D; with dsp:16, values from 0 to 262140 (65535 × 4) can be specified when the size specifier is .D. The value divided by 4 will be stored in the instruction code.

Sources of Double-Precision Floating-Point Exceptions

None

Description Example

```
DMOV.D  R1, DRH2
DMOV.L  R1, DRH2
DMOV.L  R1, DRL2
DMOV.L  DRH2, R1
DMOV.L  DRL2, R1
DMOV.D  DR0, DR1
DMOV.D  DR1, [R1]
DMOV.D  [R1], DR1
DMOV.D #3FF00000h, DRH2
DMOV.L #4C000000h, DRH2
DMOV.L #00000000h, DRL2
```

DMUL *Multiply double-precision floating-point* DMUL

Syntax

DMUL src, src2, dest

Double-precision floating-point arithmetic instruction

Instruction Code
Page: 377

Operation

dest = src2 * src;

Function

- This instruction multiplies the double-precision floating-point number stored in src2 and src and places the result in dest.
- Rounding of the result is in accordance with the setting of the DRM[1:0] bits in the DPSW.
- Handling of denormalized numbers depends on the setting of the DDN bit in the DPSW.

Flag Change

Flag	Change	Condition
C	—	
Z	—	
S	—	
O	—	
DCV	✓	The flag is set if an invalid operation exception is generated; otherwise it is cleared.
DCO	✓	The flag is set if an overflow exception is generated; otherwise it is cleared.
DCZ	✓	The value of the flag is 0.
DCU	✓	The flag is set if an underflow exception is generated; otherwise it is cleared.
DCX	✓	The flag is set if an inexact exception is generated; otherwise it is cleared.
DCE	✓	The flag is set if an unimplemented processing is generated; otherwise it is cleared.
DFV	✓	The flag is set if an invalid operation exception is generated, and otherwise left unchanged.
DFO	✓	The flag is set if an overflow exception is generated, and otherwise left unchanged.
DFZ	—	
DFU	✓	The flag is set if an underflow exception is generated, and otherwise left unchanged.
DFX	✓	The flag is set if an inexact exception is generated, and otherwise left unchanged.

Note: The DFX, DFU, DFZ, DFO, and DFV flags do not change if any of the exception enable bits (DEX, DEU, DEZ, DEO, and DEV) in the DPSW is 1.

Instruction Format

Syntax		Processing Size	Operand			Code Size (Byte)
			src	src2	dest	
DMUL	src, src2, dest	D	DRs	DRs2	DRd	4

Sources of Double-Precision Floating-Point Exceptions

Unimplemented processing
Invalid operation
Overflow
Underflow
Inexact

Description Example

DMUL DR0, DR1, DR2

Supplementary Description

- The following tables show the correspondences between the src and src2 values and the results of operations when the value of the DDN bit in the DPSW is 0 or 1.

When DDN = 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">Multiplication</td><td colspan="2">∞</td><td rowspan="6">Unimplemented processing</td><td rowspan="5">QNaN</td><td rowspan="4"></td></tr>
<tr><td>+0</td><td rowspan="2"></td><td>+0</td><td>–0</td><td colspan="2" rowspan="2">Invalid operation</td></tr>
<tr><td>–0</td><td>–0</td><td>+0</td></tr>
<tr><td>+∞</td><td rowspan="2">∞</td><td colspan="2" rowspan="2">Invalid operation</td><td>+∞</td><td>–∞</td></tr>
<tr><td>–∞</td><td>–∞</td><td>+∞</td></tr>
<tr><td>Denormalized</td><td colspan="6"></td></tr>
<tr><td>QNaN</td><td colspan="7"></td><td></td></tr>
<tr><td>SNaN</td><td colspan="8"></td><td>Invalid operation</td></tr>
</table>

When DDN = 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">Multiplication</td><td colspan="2">∞</td><td rowspan="5"></td><td rowspan="4"></td></tr>
<tr><td>+0,
+Denormalized</td><td>+0</td><td>–0</td><td colspan="2" rowspan="2">Invalid operation</td></tr>
<tr><td>–0,
–Denormalized</td><td>–0</td><td>+0</td></tr>
<tr><td>+∞</td><td rowspan="2">∞</td><td colspan="2" rowspan="2">Invalid operation</td><td>+∞</td><td>–∞</td></tr>
<tr><td>–∞</td><td>–∞</td><td>+∞</td></tr>
<tr><td>QNaN</td><td colspan="6"></td><td></td></tr>
<tr><td>SNaN</td><td colspan="7"></td><td>Invalid operation</td></tr>
</table>

DNEG *Negate double-precision floating-point* DNEG

Syntax

DNEG src, dest

Double-precision floating-point arithmetic instruction

Instruction Code
Page: 377

Operation

dest = -src;

Function

- The instruction negates the double-precision floating-point number stored in src and places the result in dest.
- Denormalized numbers are handled in the same way regardless of the setting of the DDN bit in the DPSW.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

			Operand		
Syntax		**Processing Size**	**src**	**dest**	**Code Size (Byte)**
DNEG	src, dest	D	DRs	DRd	4

Sources of Double-Precision Floating-Point Exceptions

None

Description Example

DNEG DR0, DR1

Supplementary Description

- The following table shows the correspondences between src value and the result of operations

	src								
	+Normalized	–Normalized	+0	–0	+∞	–∞	Denormalized	QNaN	SNaN
Result	–Normalized	+Normalized	–0	+0	–∞	+∞	Negated denormalized numbers	Negated QNaN	Negated SNaN

DPOPM

Pop multiple double-precision floating-point registers from stack

DPOPM

Syntax

DPOPM.size dest-dest2

Double-precision floating-point data transfer instruction

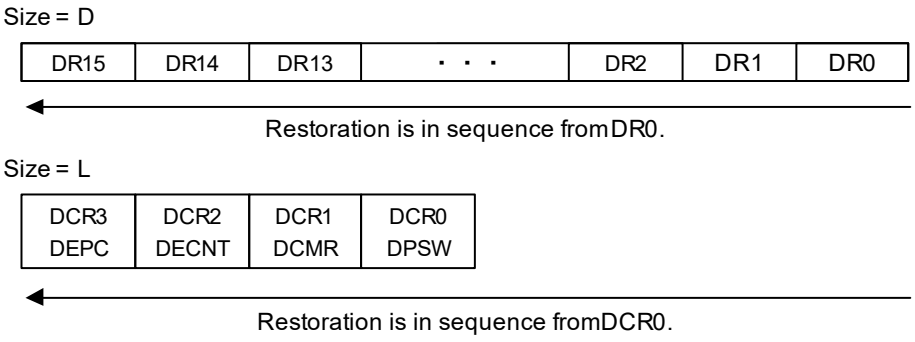
Instruction Code
Page: 378

Operation

```
signed char i;
for ( i = register_num(dest); i <= register_num(dest2); i++ ) {
  tmp = *SP;
  SP = SP + (( size == D ) ? 8 : 4);
  If ( size == D ) {
     DR(i) = tmp;
  } else {
     DCR(i) = tmp;
  }
}
```

Function

- This instruction collectively restores values of double-precision floating-point registers in the range specified by dest and dest2 from the stack.
- The range is specified by dest and dest2. Note that the register number for dest must be no higher than the register number for dest2.
- The stack pointer in use is specified by the U bit in the PSW.
- The values to be transferred to are those of double-precision floating-point data registers when size = D.
- The values to be transferred to are those of double-precision floating-point control registers when size = L.
- Registers are restored from the stack in the following order:



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Processing Size	Operand		Code Size (Byte)
		dest	dest2	
DPOPM.size dest-dest2	D	DRd	DRd2	3
	L	DCRd	DCRd2	3

Sources of Double-Precision Floating-Point Exceptions

None

Description Example

```
DPOPM.D DR0-DR15
DPOPM.L DPSW-DCMR
DPOPM.D DR1-DR1
```

DPUSHM

Push multiple double-precision floating-point registers on stack

DPUSHM

Syntax

DPUSHM.size src-src2

Double-precision floating-point data transfer instruction

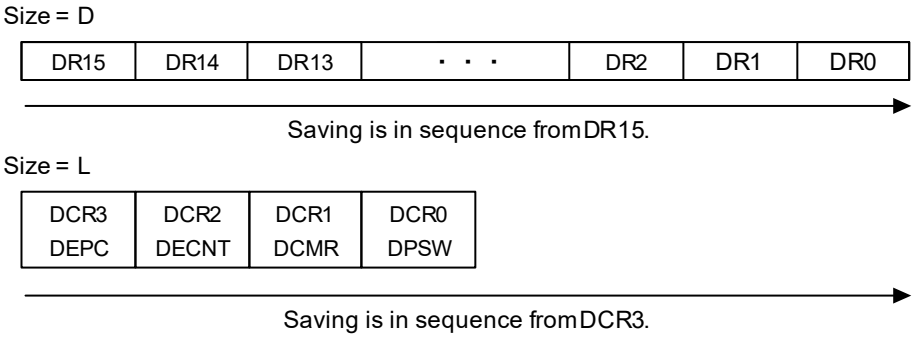
Instruction Code
Page: 379

Operation

```
signed char i;
for ( i = register_num(src2); i >= register_num(src); i-- ) {
  tmp = ( size == D ) ? DR(i) : DCR(i);
  SP = SP - (( size == D ) ? 8 : 4);
  *SP = tmp;
}
```

Function

- This instruction collectively saves the values of the double-precision floating-point registers in the range specified by src and src2 on the stack.
- The range is specified by src and src2. Note that the register number for src must be no higher than the register number for src2.
- The stack pointer in use is specified by the U bit in the PSW.
- The values to be transferred to are those of double-precision floating-point data registers when size = D.
- The values to be transferred to are those of double-precision floating-point control registers when size = L.
- Registers are saved on the stack in the following order:



Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Processing Size	Operand		Code Size (Byte)
		src	src2	
DPUSHM.size src-src2	D	DRs	DRs2	3
	L	DCRs	DCRs2	3

Sources of Double-Precision Floating-Point Exceptions

None

Description Example

```
DPUSHM.D DR0-DR15
DPUSHM.L DPSW-DCMR
DPUSHM.D DR1-DR1
```

DROUND

Round double-precision floating-point to signed integer

DROUND

Syntax

DROUND src, dest

Double-precision floating-point arithmetic instruction

Instruction Code

Page: 380

Operation

dest = (signed long) src;

Function

- This instruction converts the double-precision floating-point number stored in src into a signed longword (32-bit) integer and places the result in the lower 32 bits of dest.
- The result is rounded according to the setting of the DRM[1:0] bits in the DPSW.
- The upper 32 bits (bits 63 to 32) of dest are undefined.

Bits DRM[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	—	
S	—	
O	—	
DCV	✓	The flag is set if an invalid operation exception is generated; otherwise it is cleared.
DCO	✓	The value of the flag is 0.
DCZ	✓	The value of the flag is 0.
DCU	✓	The value of the flag is 0.
DCX	✓	The flag is set if an inexact exception is generated; otherwise it is cleared.
DCE	✓	The flag is set if an unimplemented processing exception is generated; otherwise it is cleared.
DFV	✓	The flag is set if an invalid operation exception is generated; otherwise it does not change.
DFO	—	
DFZ	—	
DFU	—	
DFX	✓	The flag is set if an inexact exception is generated; otherwise it does not change.

Note: The DFX and DFV flags do not change if any of the exception enable bits (DEX and DEV) in the DPSW is 1.

Instruction Format

			Operand		
Syntax		Processing Size	src	dest	Code Size (Byte)
DROUND	src, dest	D	DRs	DRd	4

Sources of Double-Precision Floating-Point Exceptions

Unimplemented processing
Invalid operation
Inexact

Description Example

```
DROUND  DR0, DR1
```

Supplementary Description

- The following tables show the correspondences between the src value and the results of operations when the value of the DDN bit in the DPSW is 0 or 1.

When DDN = 0

src Value (exponent is shown without bias)		dest[*1]	Exception
src ≥ 0	+∞	7FFFFFFFh	Invalid operation exception
	31 ≤ Exponent ≤ 1023		
	–1022 ≤ Exponent ≤ 30	00000000h to 7FFFFFFFh	None[*2, *3]
	+Denormalized number	No change	Unimplemented processing exception
	+0	00000000h	None
src < 0	–0		
	–Denormalized number	No change	Unimplemented processing exception
	–1022 ≤ Exponent ≤ 30	00000000h to 80000000h	None[*2]
	31 ≤ Exponent ≤ 1023	80000000h[*4]	Invalid operation exception[*2, *5]
	–∞		
NaN	QNaN	Sign bit = 0: 7FFFFFFFh	Invalid operation exception
	SNaN	Sign bit = 1: 80000000h	

Note: 1. dest is left unchanged if any of the double-precision floating-point exceptions is generated while the corresponding DEj (j= X, U, Z, O, or V) is 1.
Note: 2. If an invalid operation exception is not generated, rounding of the value will cause an inexact exception.
Note: 3. If the value after rounding exceeds 7FFFFFFFh, an invalid operation exception is generated.
Note: 4. dest becomes 80000000h regardless of the value after rounding.
Note: 5. No invalid operation exception occurs when the value after rounding is 80000000h.

When DDN = 1

src Value (exponent is shown without bias)		dest[*1]	Exception
src ≥ 0	+∞	7FFFFFFFh	Invalid operation exception
	31 ≤ Exponent ≤ 1023		
	–1022 ≤ Exponent ≤ 30	00000000h to 7FFFFFFFh	None[*2, *3]
	+Denormalized number	00000000h	None
	+0		
src < 0	–0,		
	–Denormalized number		
	–1022 ≤ Exponent ≤ 30	00000000h to 80000000h	None[*2]
	31 ≤ Exponent ≤ 1023	80000000h[*4]	Invalid operation exception[*2, *5]
	–∞		
NaN	QNaN	Sign bit = 0: 7FFFFFFFh	Invalid operation exception
	SNaN	Sign bit = 1: 80000000h	

Note: 1. dest is left unchanged if any of the double-precision floating-point exceptions is generated while the corresponding DEj (j= X, U, Z, O, or V) is 1.
Note: 2. If an invalid operation exception is not generated, rounding of the value will cause an inexact exception.
Note: 3. If the value after rounding exceeds 7FFFFFFFh, an invalid operation exception is generated.
Note: 4. dest becomes 80000000h regardless of the value after rounding.
Note: 5. No invalid operation exception occurs when the value after rounding is 80000000h.

DSQRT *Square root of double-precision floating-point* DSQRT

Syntax

DSQRT src, dest

Double-precision floating-point arithmetic instruction

Instruction Code
Page: 380

Operation

dest = sqrt(src);

Function

- This instruction calculates the square root of the double-precision floating-point number stored in src and places the result in dest.
- Rounding of the result is in accordance with the setting of the DRM[1:0] bits in the DPSW.
- Handling of denormalized numbers depends on the setting of the DDN bit in the DPSW.

Flag Change

Flag	Change	Condition
C	—	
Z	—	
S	—	
O	—	
DCV	✓	The flag is set if an invalid operation exception is generated; otherwise it is cleared.
DCO	✓	The value of the flag is 0.
DCZ	✓	The value of the flag is 0.
DCU	✓	The value of the flag is 0.
DCX	✓	The flag is set if an inexact exception is generated; otherwise it is cleared.
DCE	✓	The flag is set if an unimplemented processing is generated; otherwise it is cleared.
DFV	✓	The flag is set if an invalid operation exception is generated, and otherwise left unchanged.
DFO	—	
DFZ	—	
DFU	—	
DFX	✓	The flag is set if an inexact exception is generated, and otherwise left unchanged.

Note: The DFX and DFV flags do not change if any of the exception enable bits (DEX and DEV) in the DPSW is 1.

Instruction Format

			Operand		
Syntax		Processing Size	src	dest	Code Size (Byte)
DSQRT	src, dest	D	DRs	DRd	4

Sources of Double-Precision Floating-Point Exceptions

Unimplemented processing
Invalid operation
Inexact

Description Example

DSQRT DR0, DR1

Supplementary Description

- The following tables show the correspondences between the src value and the result of operations when the value of the DDN bit in the DPSW is 0 or 1.

When DDN = 0

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

When DDN = 1

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

Rules for Generating QNaN When Invalid Operation is Generated

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

DSUB *Subtract double-precision floating-point* DSUB

Syntax

DSUB src, src2, dest

Double-precision floating-point arithmetic instruction

Instruction Code
Page: 380

Operation

dest = src2 - src;

Function

- This instruction subtracts the double-precision floating-point number stored in src from the one in src2 and places the result in dest.
- Rounding of the result is in accordance with the setting of the DRM[1:0] bits in the DPSW.
- Handling of denormalized numbers depends on the setting of the DDN bit in the DPSW.
- The result of the operation is +0 if src and src2 have the same sign and the result of subtraction is exactly 0, except when the rounding mode is towards –∞. The operation result is –0 when the rounding mode is towards –∞.

Flag Change

Flag	Change	Condition
C	—	
Z	—	
S	—	
O	—	
DCV	✓	The flag is set if an invalid operation exception is generated; otherwise it is cleared.
DCO	✓	The flag is set if an overflow exception is generated; otherwise it is cleared.
DCZ	✓	The value of the flag is 0.
DCU	✓	The flag is set if an underflow exception is generated; otherwise it is cleared.
DCX	✓	The flag is set if an inexact exception is generated; otherwise it is cleared.
DCE	✓	The flag is set if an unimplemented processing is generated; otherwise it is cleared.
DFV	✓	The flag is set if an invalid operation exception is generated, and otherwise left unchanged.
DFO	✓	The flag is set if an overflow exception is generated, and otherwise left unchanged.
DFZ	—	
DFU	✓	The flag is set if an underflow exception is generated, and otherwise left unchanged.
DFX	✓	The flag is set if an inexact exception is generated, and otherwise left unchanged.

Note: The DFX, DFU, DFZ, DFO, and DFV flags do not change if any of the exception enable bits (DEX, DEU, DEZ, DEO, and DEV) in the DPSW is 1.

Instruction Format

Syntax		Processing Size	Operand			Code Size (Byte)
			src	src2	dest	
DSUB	src, src2, dest	D	DRs	DRs2	DRd	4

Sources of Double-Precision Floating-Point Exceptions

Unimplemented processing
Invalid operation
Overflow
Underflow
Inexact

Description Example

```
DSUB    DR0, DR1, DR2
```

Supplementary Description

- The following tables show the correspondences between the src and src2 values and the results of operations when the value of the DDN bit in the DPSW is 0 or 1.

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

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

When DDN = 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">Subtraction</td><td rowspan="3">–∞</td><td rowspan="4">+∞</td><td rowspan="6"></td><td rowspan="7"></td></tr>
<tr><td>+0,
+Denormalized</td><td>*</td><td>+0</td></tr>
<tr><td>–0,
–Denormalized</td><td>–0</td><td>*</td></tr>
<tr><td>+∞</td><td colspan="2">+∞</td><td>Invalid operation</td></tr>
<tr><td>–∞</td><td colspan="3">–∞</td><td>Invalid operation</td></tr>
<tr><td>QNaN</td><td colspan="5"></td><td>QNaN</td></tr>
<tr><td>SNaN</td><td colspan="6"></td><td>Invalid operation</td></tr>
</table>

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

DTOF

Convert double-precision floating-point to single-precision floating-point

DTOF

Syntax

DTOF src, dest

Double-precision floating-point arithmetic instruction

Instruction Code

Page: 381

Operation

dest = (float) src;

Function

- This instruction converts the double-precision floating-point number stored in src into a single-precision floating-point number and places the result in the lower 32 bits of dest.
- Rounding of the result is in accordance with the setting of the DRM[1:0] bits in the DPSW.
- The upper 32 bits (bits 63 to 32) of dest are undefined.

Flag Change

Flag	Change	Condition
C	—	
Z	—	
S	—	
O	—	
DCV	✓	The flag is set if an invalid operation exception is generated; otherwise it is cleared.
DCO	✓	The flag is set if an overflow exception is generated; otherwise it is cleared.
DCZ	✓	The value of the flag is 0.
DCU	✓	The flag is set if an underflow exception is generated; otherwise it is cleared.
DCX	✓	The flag is set if an inexact exception is generated; otherwise it is cleared.
DCE	✓	The flag is set if an unimplemented processing is generated; otherwise it is cleared.
DFV	✓	The flag is set if an invalid operation exception is generated, and otherwise left unchanged.
DFO	—	
DFZ	—	
DFU	—	
DFX	✓	The flag is set if an inexact exception is generated, and otherwise left unchanged.

Note: The DFX and DFV flags do not change if any of the exception enable bits (DEX and DEV) in the DPSW is 1.

Instruction Format

Syntax	Processing Size	Operand		Code Size (Byte)
		src	dest	
DTOF src, dest	D	DRs	DRd	4

Sources of Double-Precision Floating-Point Exceptions

Unimplemented processing
Invalid operation
Overflow
Underflow
Inexact

Description Example

DTOF DR0, DR1

Supplementary Description

- The following tables show the correspondences between the src value and the results of operations when the value of the DDN bit in the DPSW is 0 or 1.

When DDN = 0

src Value (exponent is shown without bias)		dest[*1]	Exception
src ≥ 0	+∞	+∞	None
	128 ≤ Exponent ≤ 1023		Overflow exception
	–126 ≤ Exponent ≤ 127	Result of operation	None[*2, *3]
	–1022 ≤ Exponent ≤ –127	No change	Unimplemented processing exception Underflow exception
	+Denormalized number	No change	Unimplemented processing exception
	+0	+0	None
src < 0	–0	–0	
	–Denormalized number	No change	Unimplemented processing exception
	–1022 ≤ Exponent ≤ –127	No change	Unimplemented processing exception Underflow exception
	–126 ≤ Exponent ≤ 127	Result of operation	None[*2, *3]
	128 ≤ Exponent ≤ 1023	–∞	Overflow exception
	–∞		None
NaN	QNaN	QNaN[*4]	None
	SNaN	SNaN[*5]	Invalid operation exception

Note: 1. dest is left unchanged if any of the double-precision floating-point exceptions is generated while the corresponding DEj (j= X, U, Z, O, or V) is 1.

Note: 2. An inexact exception occurs when the result is rounded.

Note: 3. An overflow exception is generated if the exponent after rounding is 128.

Note: 4. dest becomes a QNaN which has the same sign as the input QNaN and contains bits [51:29] of the input value as the fraction.

Note: 5. dest becomes a QNaN which has the same sign as the input SNaN and contains bits [51:29] of the input value after it has been converted into a QNaN as the fraction.

When DDN = 1

src Value (exponent is shown without bias)		dest[*1]	Exception
src ≥ 0	+∞	+∞	None
	128 ≤ Exponent ≤ 1023		Overflow exception
	–126 ≤ Exponent ≤ 127	Result of operation	None[*2, *3]
	–1022 ≤ Exponent ≤ –127	+0	Underflow exception
	+Denormalized number	+0	None
	+0	+0	None
src < 0	–0	–0	
	–Denormalized number	–0	None
	–1022 ≤ Exponent ≤ –127	–0	Underflow exception
	–126 ≤ Exponent ≤ 127	Result of operation	None[*2, *3]
	128 ≤ Exponent ≤ 1023	–∞	Overflow exception
	–∞		None
NaN	QNaN	QNaN[*4]	None
	SNaN	SNaN[*5]	Invalid operation exception

Note: 1. dest is left unchanged if any of the double-precision floating-point exceptions is generated while the corresponding DEj (j= X, U, Z, O, or V) is 1.

Note: 2. An inexact exception occurs when the result is rounded.

Note: 3. An overflow exception is generated if the exponent after rounding is 128.

Note: 4. dest becomes a QNaN which has the same sign as the input QNaN and contains bits [51:29] of the input value as the fraction.

Note: 5. dest becomes a QNaN which has the same sign as the input SNaN and contains bits [51:29] of the input value after it has been converted into a QNaN as the fraction.

DTOI

Convert double-precision floating-point to signed integer

DTOI

Syntax

DTOI src, dest

Double-precision floating-point arithmetic instruction

Instruction Code

Page: 381

Operation

dest = (long) src;

Function

- This instruction converts the double-precision floating-point number stored in src into a signed longword (32-bit) integer and places the result in the lower 32 bits of dest.
- The result is always rounded towards 0, regardless of the setting of the DRM[1:0] bits in the DPSW.
- The upper 32 bits (bits 63 to 32) of dest are undefined.

Flag Change

Flag	Change	Condition
C	—	
Z	—	
S	—	
O	—	
DCV	✓	The flag is set if an invalid operation exception is generated; otherwise it is cleared.
DCO	✓	The value of the flag is 0.
DCZ	✓	The value of the flag is 0.
DCU	✓	The value of the flag is 0.
DCX	✓	The flag is set if an inexact exception is generated; otherwise it is cleared.
DCE	✓	The flag is set if an unimplemented processing exception is generated; otherwise it is cleared.
DFV	✓	The flag is set if an invalid operation exception is generated; otherwise it does not change.
DFO	—	
DFZ	—	
DFU	—	
DFX	✓	The flag is set if an inexact exception is generated; otherwise it does not change.

Note: The DFX and DFV flags do not change if any of the exception enable bits (DEX and DEV) in the DPSW is 1.

Instruction Format

			Operand		
Syntax		Processing Size	src	dest	Code Size (Byte)
DTOI	src, dest	D	DRs	DRd	4

Sources of Double-Precision Floating-Point Exceptions

Unimplemented processing
Invalid operation
Inexact

Description Example

DTOI DR0, DR1

Supplementary Description

- The following tables show the correspondences between the src value and the results of operations when the value of the DDN bit in the DPSW is 0 or 1.

When DDN = 0

src Value (exponent is shown without bias)		dest[*1]	Exception
src ≥ 0	+∞	7FFFFFFFh	Invalid operation exception
	31 ≤ Exponent ≤ 1023		
	–1022 ≤ Exponent ≤ 30	00000000h to 7FFFFFFFh	None[*2, *3]
	+Denormalized number	No change	Unimplemented processing exception
	+0	00000000h	None
src < 0	–0		
	–Denormalized number	No change	Unimplemented processing exception
	–1022 ≤ Exponent ≤ 30	00000000h to 80000000h	None[*2]
	31 ≤ Exponent ≤ 1023	80000000h[*4]	Invalid operation exception[*2, *5]
	–∞		
NaN	QNaN	Sign bit = 0: 7FFFFFFFh	Invalid operation exception
	SNaN	Sign bit = 1: 80000000h	

Note: 1. dest is left unchanged if any of the double-precision floating-point exceptions is generated while the corresponding DEj (j= X, U, Z, O, or V) is 1.

Note: 2. If an invalid operation exception is not generated, rounding of the value will cause an inexact exception.

Note: 3. If the value after rounding exceeds 7FFFFFFFh, an invalid operation exception is generated.

Note: 4. dest becomes 80000000h regardless of the value after rounding.

Note: 5. No invalid operation exception occurs when the value after rounding is 80000000h.

When DDN = 1

src Value (exponent is shown without bias)		dest[*1]	Exception
src ≥ 0	+∞	7FFFFFFFh	Invalid operation exception
	31 ≤ Exponent ≤ 1023		
	–1022 ≤ Exponent ≤ 30	00000000h to 7FFFFFFFh	None[*2, *3]
	+Denormalized number	00000000h	None
	+0		
src < 0	–0		
	–Denormalized number		
	–1022 ≤ Exponent ≤ 30	00000000h to 80000000h	None[*2]
	31 ≤ Exponent ≤ 1023	80000000h[*4]	Invalid operation exception[*2, *5]
	–∞		
NaN	QNaN	Sign bit = 0: 7FFFFFFFh	Invalid operation exception
	SNaN	Sign bit = 1: 80000000h	

Note: 1. dest is left unchanged if any of the double-precision floating-point exceptions is generated while the corresponding DEj (j= X, U, Z, O, or V) is 1.

Note: 2. If an invalid operation exception is not generated, rounding of the value will cause an inexact exception.

Note: 3. If the value after rounding exceeds 7FFFFFFFh, an invalid operation exception is generated.

Note: 4. dest becomes 80000000h regardless of the value after rounding.

Note: 5. No invalid operation exception occurs when the value after rounding is 80000000h.

DTOU

Convert double-precision floating-point to unsigned integer

DTOU

Syntax

DTOU src, dest

Double-precision floating-point arithmetic instruction

Instruction Code

Page: 381

Operation

dest = (unsigned long) src;

Function

- This instruction converts the double-precision floating-point number stored in src into an unsigned longword (32-bit) integer and places the result in the lower 32 bits of dest.
- The result is always rounded towards 0, regardless of the setting of the DRM[1:0] bits in the DPSW.
- The upper 32 bits (bits 63 to 32) of dest are undefined.

Flag Change

Flag	Change	Condition
C	—	
Z	—	
S	—	
O	—	
DCV	✓	The flag is set if an invalid operation exception is generated; otherwise it is cleared.
DCO	✓	The value of the flag is 0.
DCZ	✓	The value of the flag is 0.
DCU	✓	The value of the flag is 0.
DCX	✓	The flag is set if an inexact exception is generated; otherwise it is cleared.
DCE	✓	The flag is set if an unimplemented processing is generated; otherwise it is cleared.
DFV	✓	The flag is set if an invalid operation exception is generated, and otherwise left unchanged.
DFO	—	
DFZ	—	
DFU	—	
DFX	✓	The flag is set if an inexact exception is generated, and otherwise left unchanged.

Note: The DFX and DFV flags do not change if any of the exception enable bits (DEX and DEV) in the DPSW is 1.

Instruction Format

Syntax	Processing Size	Operand		Code Size (Byte)
		src	dest	
DTOU src, dest	D	DRs	DRd	4

Sources of Double-Precision Floating-Point Exceptions

Unimplemented processing
Invalid operation
Inexact

Description Example

```
DTOU  DR0, DR1
```

Supplementary Description

- The following tables show the correspondences between the src value and the results of operations when the value of the DDN bit in the DPSW is 0 or 1.

When DDN = 0

src Value (exponent is shown without bias)		dest*1	Exception
src ≥ 0	+∞	FFFFFFFFh	Invalid operation exception
	32 ≤ Exponent ≤ 1023		
	–1022 ≤ Exponent ≤ 31	00000000h to FFFFFFFFh	None*2, *3
	+Denormalized number	No change	Unimplemented processing exception
	+0	00000000h	None
src < 0	–0		
	–Denormalized number	No change	Unimplemented processing exception
	–Normalized number, –∞	00000000h	Invalid operation exception
NaN	QNaN	Most significant bit = 0: FFFFFFFFh	Invalid operation exception
	SNaN	Most significant bit = 1: 00000000h	

Note: 1. dest is left unchanged if any of the double-precision floating-point exceptions is generated while the corresponding DEj (j= X, U, Z, O, or V) is 1.

Note: 2. An inexact exception occurs when the result is rounded.

Note: 3. If the value after rounding exceeds FFFFFFFFh, an invalid operation exception is generated.

When DDN = 1

src Value (exponent is shown without bias)		dest*1	Exception
src ≥ 0	+∞	FFFFFFFFh	Invalid operation exception
	32 ≤ Exponent ≤ 1023		
	–1022 ≤ Exponent ≤ 31	00000000h to FFFFFFFFh	None*2, *3
	+0, +Denormalized number	00000000h	None
src < 0	–0, –Denormalized number		
	–Normalized number, –∞	00000000h	Invalid operation exception
NaN	QNaN	Sign bit = 0: FFFFFFFFh	Invalid operation exception
	SNaN	Sign bit = 1: 00000000h	

Note: 1. dest is left unchanged if any of the double-precision floating-point exceptions is generated while the corresponding DEj (j= X, U, Z, O, or V) is 1.

Note: 2. An inexact exception occurs when the result is rounded.

Note: 3. If the value after rounding exceeds FFFFFFFFh, an invalid operation exception is generated.

FTOD

Convert single-precision floating-point to double-precision floating-point

FTOD

Syntax

FTOD src, dest

Double-precision floating-point arithmetic instruction

Instruction Code
Page: 382

Operation

dest = (double) src;

Function

- This instruction converts the single-precision floating-point number stored in src into a double-precision floating-point number and places the result in dest.

Flag Change

Flag	Change	Condition
C	—	
Z	—	
S	—	
O	—	
DCV	✓	The flag is set if an invalid operation exception is generated; otherwise it is cleared.
DCO	✓	The value of the flag is 0.
DCZ	✓	The value of the flag is 0.
DCU	✓	The value of the flag is 0.
DCX	✓	The value of the flag is 0.
DCE	✓	The flag is set if an unimplemented processing is generated; otherwise it is cleared.
DFV	✓	The flag is set if an invalid operation exception is generated, and otherwise left unchanged.
DFO	—	
DFZ	—	
DFU	—	
DFX	—	

Note: The DFV flag does not change if the exception enable bit (DEV) in the DPSW is 1.

Instruction Format

		Operand		
Syntax	Processing Size	src	dest	Code Size (Byte)
FTOD src, dest	D	Rs	DRd	4

Sources of Double-Precision Floating-Point Exceptions

Unimplemented processing
Invalid operation

Description Example

FTOD R1, DR1

Supplementary Description

- The following tables show the correspondences between the src value and the results of operations when the value of the DDN bit in the DPSW is 0 or 1.

When DDN = 0

src Value (exponent is shown without bias)		dest[*1]	Exception
src ≥ 0	+∞	+∞	None
	–126 ≤ Exponent ≤ 127	Result of operation	None
	+Denormalized number	No change	Unimplemented processing exception
	+0	+0	None
src < 0	–0	–0	
	–Denormalized number	No change	Unimplemented processing exception
	–126 ≤ Exponent ≤ 127	Result of operation	None
	–∞	–∞	None
NaN	QNaN	QNaN[*2]	None
	SNaN	SNaN[*3]	Invalid operation exception

Note: 1. dest is left unchanged if any of the double-precision floating-point exceptions is generated while the corresponding DEj (j= X, U, Z, O, or V) is 1.

Note: 2. dest becomes a QNaN with the same sign as the input QNaN. Bits [22:0] of the input value become bits [51:29] of the fraction, and bits [28:0] of the fraction become 0.

Note: 3. dest becomes a QNaN with the same sign as the input SNaN, and bits [22:0] of the input value are converted into a QNaN and stored in bits [51:29] of the fraction. Bits [28:0] of the fraction become 0.

When DDN = 1

src Value (exponent is shown without bias)		dest[*1]	Exception
src ≥ 0	+∞	+∞	None
	–126 ≤ Exponent ≤ 127	Result of operation	None
	+Denormalized number	+0	None
	+0	+0	None
src < 0	–0	–0	
	–Denormalized number	–0	None
	–126 ≤ Exponent ≤ 127	Result of operation	None
	–∞	–∞	None
NaN	QNaN	QNaN[*2]	None
	SNaN	SNaN[*3]	Invalid operation exception

Note: 1. dest is left unchanged if any of the double-precision floating-point exceptions is generated while the corresponding DEj (j= X, U, Z, O, or V) is 1.

Note: 2. dest becomes a QNaN with the same sign as the input QNaN. Bits [22:0] of the input value become bits [51:29] of the fraction, and bits [28:0] of the fraction become 0.

Note: 3. dest becomes a QNaN with the same sign as the input SNaN, and bits [22:0] of the input value are converted into a QNaN and stored in bits [51:29] of the fraction. Bits [28:0] of the fraction become 0.

ITOD

Convert signed integer to double-precision floating-point

ITOD

Syntax

ITOD src, dest

Double-precision floating-point arithmetic instruction

Instruction Code

Page: 382

Operation

dest = (double) src;

Function

- This instruction converts the signed longword (32-bit) integer stored in src into a double-precision floating-point number and places the result in dest.

Flag Change

Flag	Change	Condition
C	—	
Z	—	
S	—	
O	—	
DCV	✓	The value of the flag is 0.
DCO	✓	The value of the flag is 0.
DCZ	✓	The value of the flag is 0.
DCU	✓	The value of the flag is 0.
DCX	✓	The value of the flag is 0.
DCE	✓	The value of the flag is 0.
DFV	—	
DFO	—	
DFZ	—	
DFU	—	
DFX	—	

Instruction Format

		Operand		
Syntax	Processing Size	src	dest	Code Size (Byte)
ITOD src, dest	D	Rs	DRd	4

Sources of Double-Precision Floating-Point Exceptions

None

Description Example

ITOD R1, DR0

MVFDC

Move data from double-precision floating-point control register

MVFDC

Double-precision floating-point data transfer instruction

Instruction Code
Page: 383

Syntax

MVFDC src, dest

Operation

dest = src;

Function

- This instruction transfers src to dest.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Processing Size	Operand		Code Size (Byte)
		src*	dest	
MVFDC src, dest	L	DCRs	Rd	4

Note: * Selectable src: Registers DPSW, DCMR, DECNT, and DEPC

Sources of Double-Precision Floating-Point Exceptions

None

Description Example

MVFDC DPSW, R2

MVFDR

Move data from double-precision floating-point comparison result register

MVFDR

Syntax

MVFDR

Double-precision floating-point data transfer instruction

Instruction Code

Page: 383

Operation

Z = DCMR.RES;

Function

- This instruction transfers the value of the RES bit in the DCMR to the Z flag of the PSW.

Flag Change

Flag	Change	Condition
C	—	
Z	✓	The value of DCMR.RES is written here.
S	—	
O	—	

Note: The values of bits in the DPSW do not change.

Instruction Format

Syntax	Code Size (Byte)
MVFDR	3

Sources of Double-Precision Floating-Point Exceptions

None

Description Example

MVFDR

MVTDC

Move data to double-precision floating-point control register

MVTDC

Double-precision floating-point data transfer instruction

Instruction Code
Page: 384

Syntax

MVTDC src, dest

Operation

dest = src;

Function

- This instruction transfers src to dest.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

Syntax	Processing Size	Operand		Code Size (Byte)
		src	dest*	
MVTDC src, dest	L	Rs	DCRd	4

Note: * Selectable dest: Registers DPSW, DCMR, DECNT, and DEPC

Sources of Double-Precision Floating-Point Exceptions

None

Description Example

MVTDC R1,DPSW

UTOD

Convert unsigned integer to double-precision floating-point

UTOD

Syntax

UTOD src, dest

Double-precision floating-point arithmetic instruction

Instruction Code

Page: 384

Operation

dest = (double) src;

Function

- This instruction converts the unsigned longword (32-bit) integer stored in src into a double-precision floating-point number and places the result in dest.

Flag Change

Flag	Change	Condition
C	—	
Z	—	
S	—	
O	—	
DCV	✓	The value of the flag is 0.
DCO	✓	The value of the flag is 0.
DCZ	✓	The value of the flag is 0.
DCU	✓	The value of the flag is 0.
DCX	✓	The value of the flag is 0.
DCE	✓	The value of the flag is 0.
DFV	—	
DFO	—	
DFZ	—	
DFU	—	
DFX	—	

Instruction Format

			Operand		
Syntax		Processing Size	src	dest	Code Size (Byte)
UTOD	src, dest	D	Rs	DRd	4

Sources of Double-Precision Floating-Point Exceptions

None

Description Example

UTOD R1, DR0

3.4.4 Protect Zone Instructions

The following pages give details of the Protect Zone instructions.

TRCALL

Call for transition to the root world

TRCALL

Syntax

Protect Zone instruction

TRCALL src

Instruction Code
Page: 386

Operation

```
WID.BWI = PSW.I;
WID.BWID = WID.WID;
WID.WID = 0;
tmp0 = PSW;
I = 0;
U=0;
tmp1 = PC + 4;
PC = *(TRCTB+src*4);
ISP_TR = ISP_TR - 4;
*ISP_TR = tmp0;
ISP_TR = ISP_TR - 4;
*ISP_TR = tmp1;
```

Function

- This instruction generates an unconditional trap (with world-transition acceptance) with a number matching that specified in src.
- The value of src is in the range 0 ≤ src ≤ 255.
- The CPU is placed in supervisor mode in the root world and the WID bits in the WID register become 00b.
- The world ID before execution of this instruction is saved in the BWID bits of the WID register.
- This instruction is a privileged instruction. Attempting to execute this instruction in user mode generates a privileged instruction exception.

Flag Change

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

Instruction Format

	Operand	
Syntax	**src**	**Code Size (Byte)**
TRCALL src	#IMM:8	4

Description Example

```
TRCALL #0
```

TRMVFC

Transfer from a control register in the root world

TRMVFC

Protect Zone instruction

Syntax

TRMVFC src, dest

Instruction Code
Page: 336

Operation

dest = src;

Function

- This instruction transfers src to dest.
- When a register specified as src is separately implemented for root, secure, and non-secure, data are transferred from the register for the world indicated by the BWID bits in the WID register.
  WID.BWID==2: **_NS
  WID.BWID==1: **_S
  WID.BWID==0: **_TR
- This instruction is only for the root world. If it is executed in any other world, an undefined instruction exception occurs.
- This instruction is a privileged instruction. Attempting to execute this instruction in user mode while in the root world generates a privileged instruction exception.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

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

Note: * Selectable src: Registers ISP, INTB, BPC, BPSW, EXTB, IVC, TRCTB, and WINTB

Description Example

TRMVFC ISP, R1
TRMVFC TRCTB, R1

TRMVTC

Transfer to a control register in the root world

TRMVTC

Protect Zone instruction

Syntax

TRMVTC src, dest

Instruction Code
Page: 336

Operation

dest = src;

Function

- This instruction transfers src to dest.
- When a register specified as dest is separately implemented for root, secure, and non-secure, data are transferred to the register for the world indicated by the BWID bits in the WID register.
  WID.BWID==2: **_NS
  WID.BWID==1: **_S
  WID.BWID==0: **_TR
- Regardless of whether the value written includes 1 or 0 for bit 28 in BPSW_NS, 0 is always written to the bit.
- This instruction is exclusively for use in the root world. If it is executed in any other world, an undefined instruction exception occurs.
- This instruction is a privileged instruction. Attempting to execute this instruction in user mode while in the root world generates a privileged instruction exception.

Flag Change

- This instruction does not affect the states of flags.

Instruction Format

		Operand		
Syntax	**Processing Size**	**src**	**dest***	**Code Size (Byte)**
TRMVTC 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, INTB, BPC, BPSW, EXTB, IVC, TRCTB, and WINTB

Description Example

```
TRMVTC #0FFFF0000h, TRCTB
TRMVTC R1, ISP
TRMVTC R1,WINTB
```

TRRTE *Return from exception in the root world* TRRTE

Syntax

Protect Zone instruction

TRRTE

Instruction Code
Page: 336

Operation

```
PC = *SP_TR;
SP_TR = SP_TR + 4;
tmp1 = *SP_TR;
SP_TR = SP_TR + 4;
if ( WID.BWID == 2 ) {
 tmp1 = 0effffffh & tmp1;
}
PSW = tmp1;
WID.WID = WID.BWID;
WID.BWID = 0;
WID.BITW = 0;
WID.BWI = 0;
LI = 0;
```

Note: When PSW.U==0, SP_TR is ISP_TR, and when PSW.U==1, SP_TR is USP

Function

- This instruction restores the values of the PC and PSW that were saved on acceptance of the exception, and returns the world ID from the root world to the secure or non-secure world.
- This instruction is only for the root world. If it is executed in any other world, an undefined instruction exception occurs.
- This instruction is a privileged instruction. Attempting to execute this instruction in user mode while in the root world 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

- C, Z, S, and O are values on the stack in the root world.

Instruction Format

Syntax	Code Size (Byte)
TRRTE	2

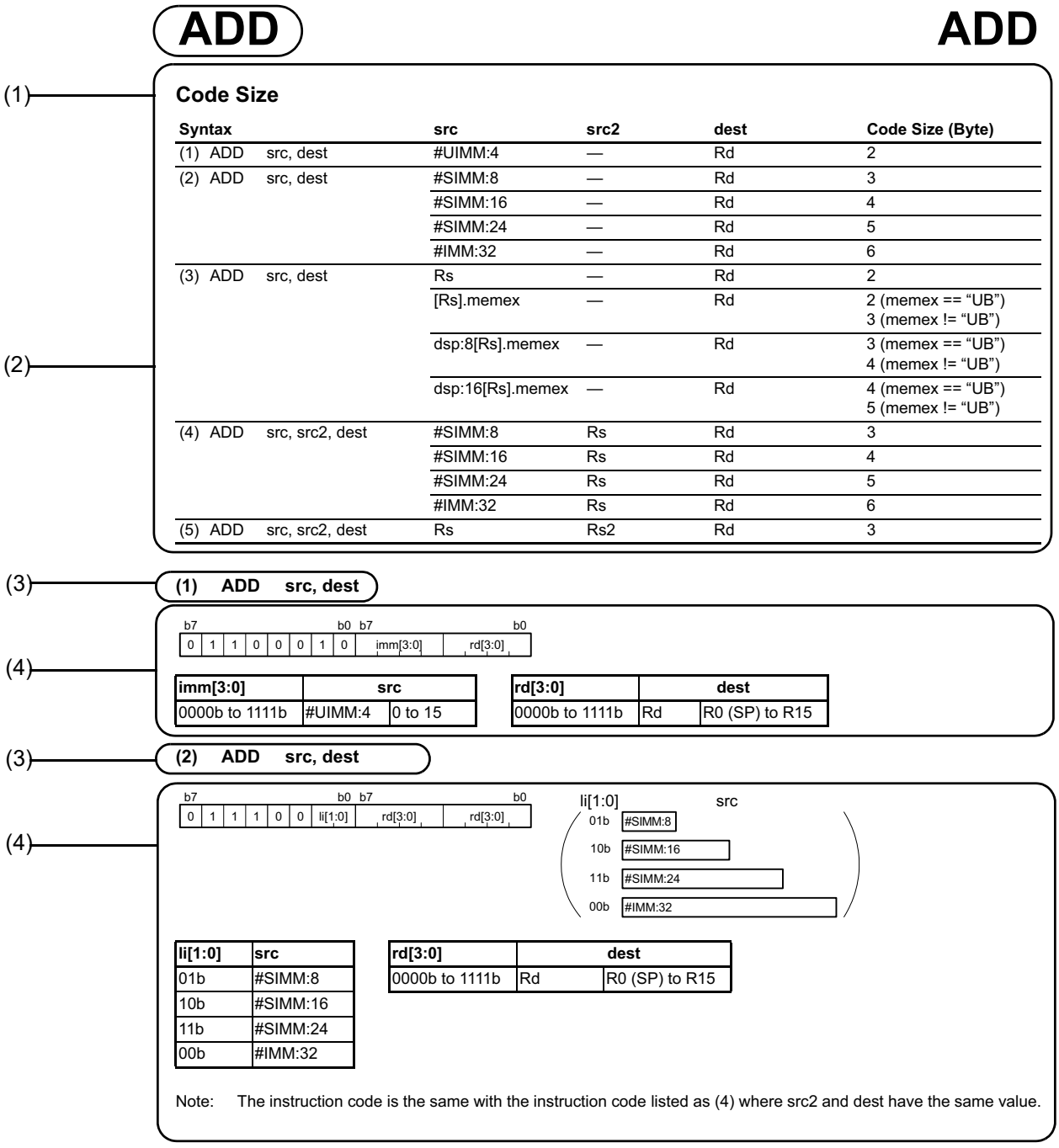
Description Example

```
TRRTE
```

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.



(1) Mnemonic

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

(2) List of Code Size

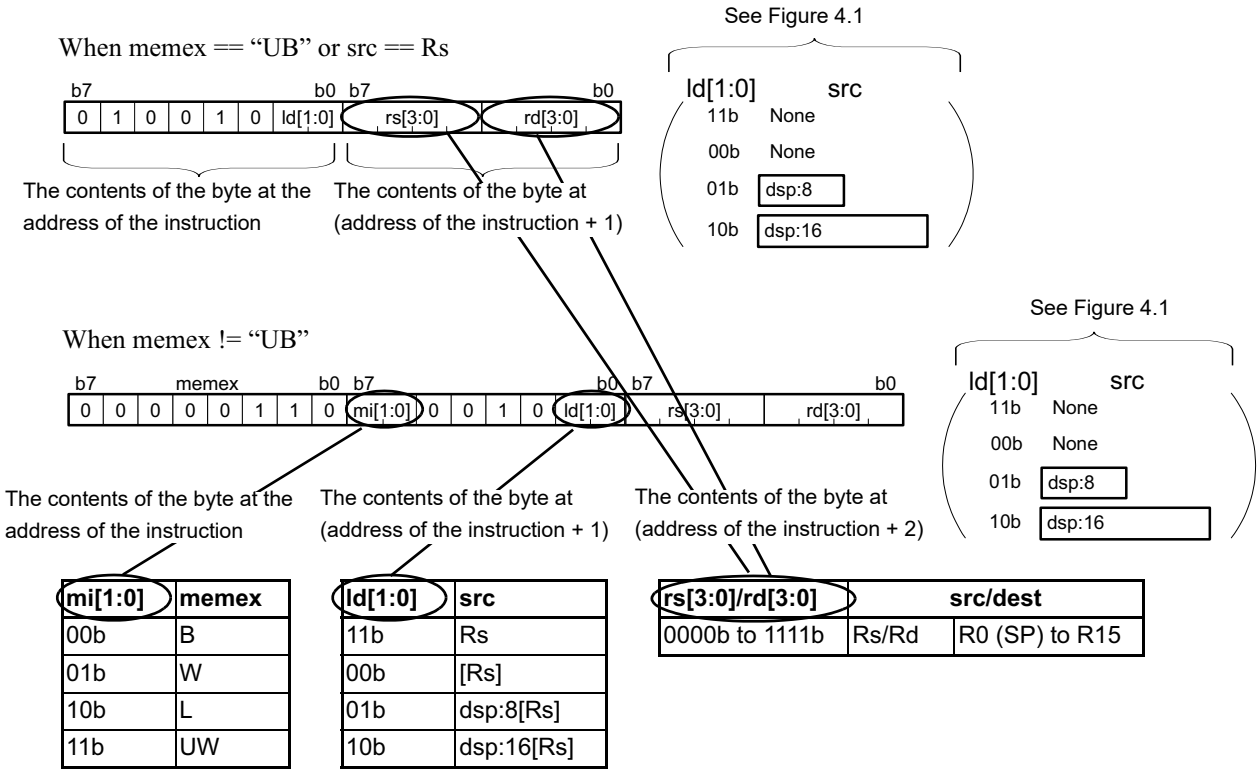
Indicates the number of bytes the instruction requires. An individual RXv3e CPU instruction takes up from one to eight bytes.

(3) Syntax

Indicates the syntax of the instruction using symbols.

(4) Instruction Code

Indicates the instruction code. The code in parentheses may be selected or omitted depending on src/dest to be selected.



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

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

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

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

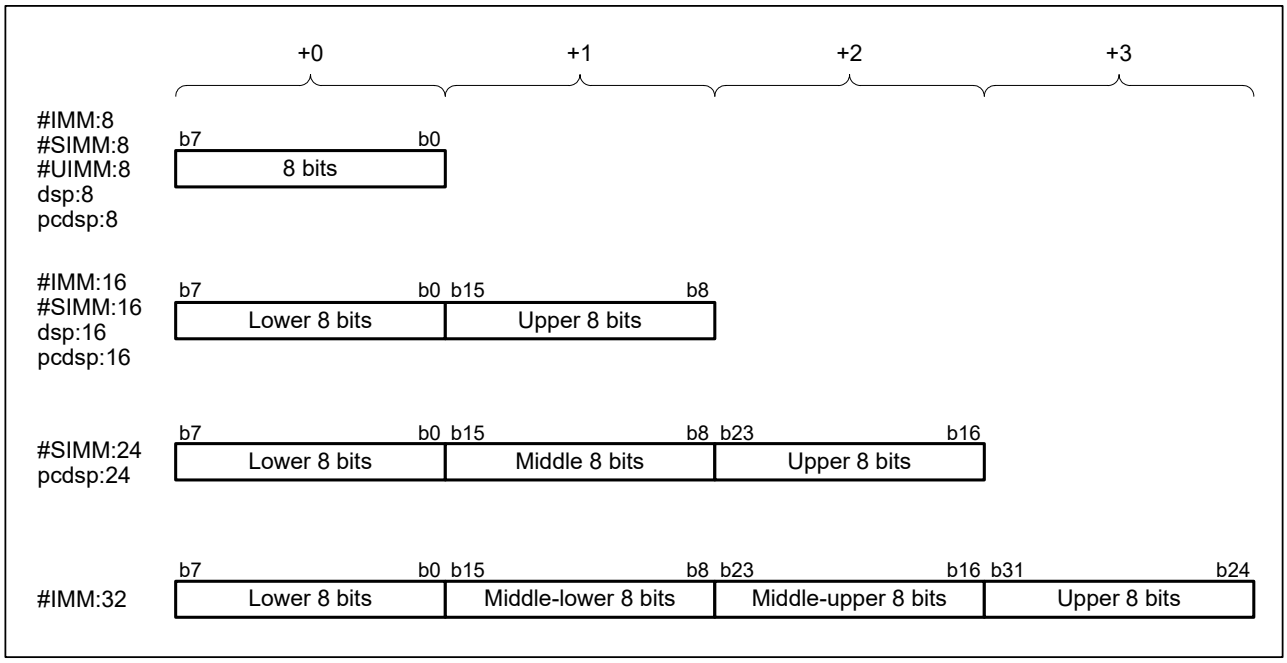


Figure 4.1 Immediate (IMM) and Displacement (dsp) Values

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

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

4.2 Instruction Code Described in Detail

The instruction codes for the RXv3e instructions are described in detail in this section.

4.2.1 Standard provided instructions

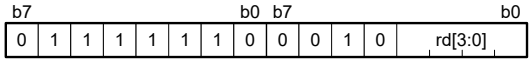
The following pages give details of the instruction codes for the standard provided instructions.

ABS ABS

Code Size

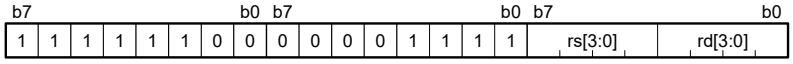
Syntax		src	dest	Code Size (Byte)
(1) ABS	dest	—	Rd	2
(2) ABS	src, dest	Rs	Rd	3

(1) ABS dest



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

(2) ABS src, dest



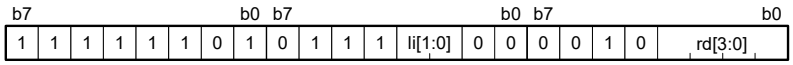
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

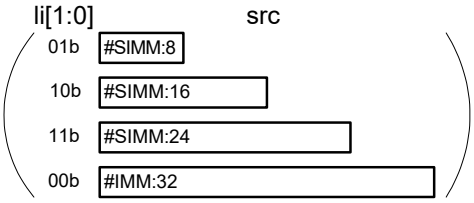
ADC ADC

Code Size

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

(1) ADC src, dest

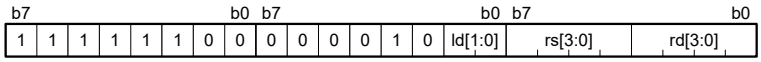




li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

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

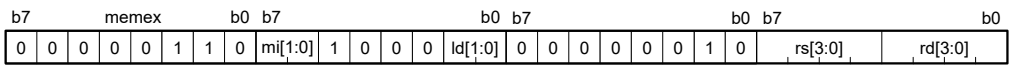
(2) ADC src, dest



ld[1:0]	src
11b	Rs

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

(3) ADC src, dest





mi[1:0]	memex
10b	L

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

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

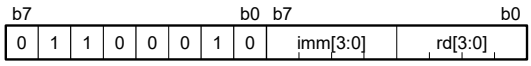
ADD

ADD

Code Size

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

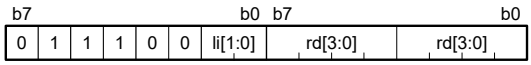
(1) ADD src, dest

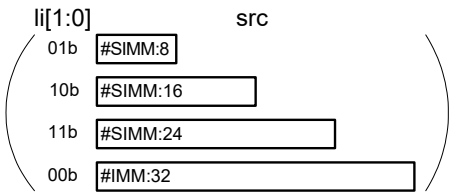


imm[3:0]	src	
0000b to 1111b	#UIMM:4	0 to 15

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

(2) ADD src, dest





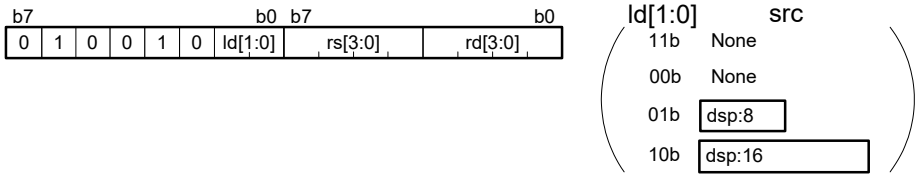
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

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

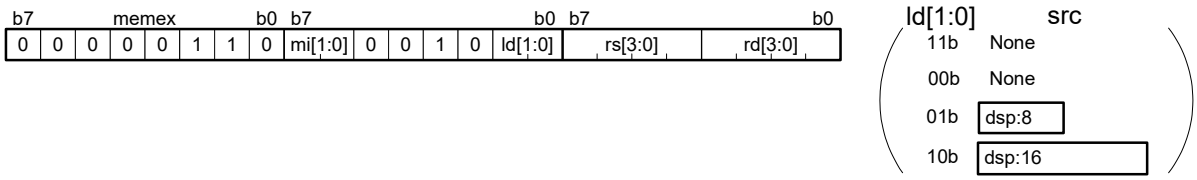
Note: The instruction code is the same with the instruction code listed as (4) where src2 and dest have the same value.

(3) ADD src, dest

When memex == “UB” or src == Rs



When memex != “UB”

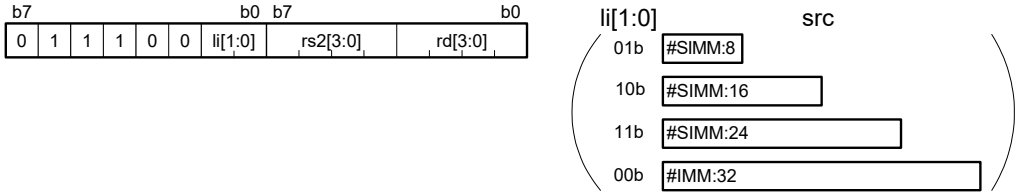


mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

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

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

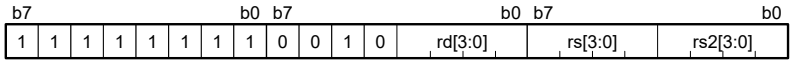
(4) ADD src, src2, dest



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

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

(5) ADD src, src2, dest



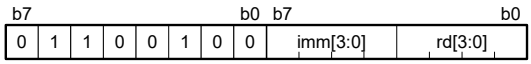
rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b to 1111b	Rs/Rs2/Rd	R0 (SP) to R15

AND AND

Code Size

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

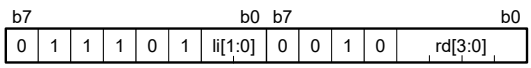
(1) AND src, dest

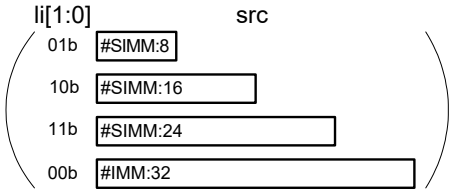


imm[3:0]	src	
0000b to 1111b	#UIMM:4	0 to 15

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

(2) AND src, dest



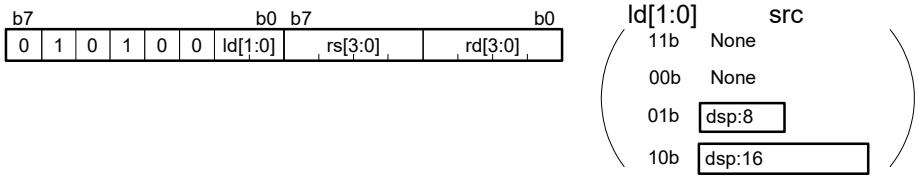


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

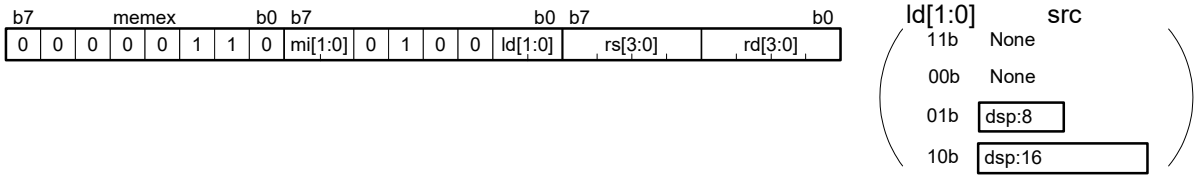
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(3) AND src, dest

When memex == “UB” or src == Rs



When memex != “UB”

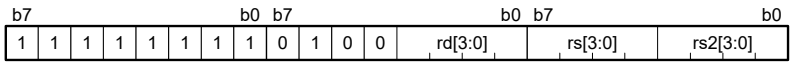


mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

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

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

(4) AND src, src2, dest



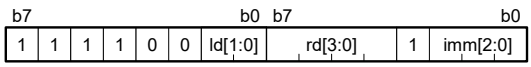
rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b to 1111b	Rs/Rs2/Rd	R0 (SP) to R15

BCLR BCLR

Code Size

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

(1) BCLR src, dest



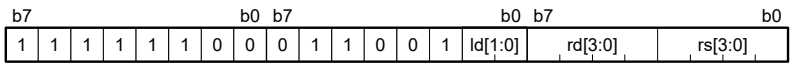


ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

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

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

(2) BCLR src, dest

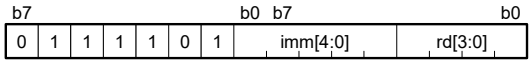




ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

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

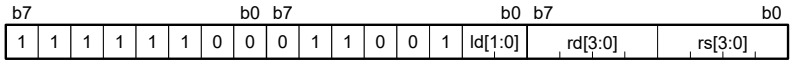
(3) BCLR src, dest



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

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

(4) BCLR src, dest



ld[1:0]	dest
11b	Rd

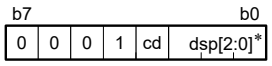
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

B*Cnd* B*Cnd*

Code Size

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

(1) B*Cnd*.S src

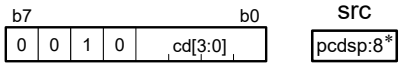


Note: * dsp[2:0] specifies pcdsp:3 = src.

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

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

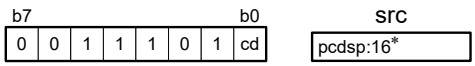
(2) B*Cnd*.B src



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

cd[3:0]	B*Cnd*	cd[3:0]	B*Cnd*
0000b	BEQ, BZ	1000b	BGE
0001b	BNE, BNZ	1001b	BLT
0010b	BGEU, BC	1010b	BGT
0011b	BLTU, BNC	1011b	BLE
0100b	BGTU	1100b	BO
0101b	BLEU	1101b	BNO
0110b	BPZ	1110b	BRA.B
0111b	BN	1111b	Reserved

(3) B*Cnd*.W src



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

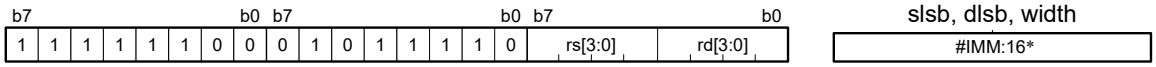
cd	B*Cnd*
0b	BEQ, BZ
1b	BNE, BNZ

BFMOV BFMOV

Code Size

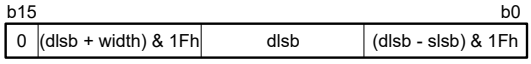
		Operand					
Syntax		**slsb**	**dlsb**	**width**	**src**	**dest**	**Code Size (Byte)**
(1) BFMOV	slsb, dlsb, width, src, dest	#IMM:5	#IMM:5	#IMM:5	Rs	Rd	5

(1) BFMOV slsb, dlsb, width, src, dest



Note: * The #IMM:16 value is generated based on the following rule according to the operands slsb, dlsb, and width.
#IMM:16 = (((dlsb + width) & 1Fh) << 10) | (dlsb << 5) | ((dlsb - slsb) & 1Fh)

Image



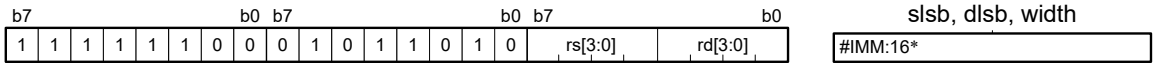
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

BFMOVZ BFMOVZ

Code Size

		Operand					
Syntax		**slsb**	**dlsb**	**width**	**src**	**dest**	**Code Size (Byte)**
(1) BFMOVZ	slsb, dlsb, width, src, dest	#IMM:5	#IMM:5	#IMM:5	Rs	Rd	5

(1) BFMOVZ slsb, dlsb, width, src, dest



Note: * The #IMM:16 value is generated based on the following rule according to the operands slsb, dlsb, and width.
#IMM:16 = (((dlsb + width) & 1Fh) << 10) | (dlsb << 5) | ((dlsb - slsb) & 1Fh)

Image



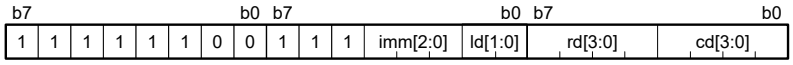
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

BM*Cnd* BM*Cnd*

Code Size

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

(1) BM*Cnd* src, dest





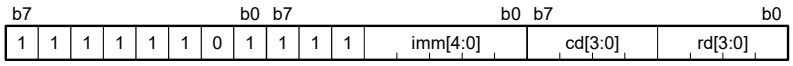
imm[2:0]	src	
000b to 111b	#IMM:3	0 to 7

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

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

cd[3:0]	BM*Cnd*	cd[3:0]	BM*Cnd*
0000b	BMEQ, BMZ	1000b	BMGE
0001b	BMNE, BMNZ	1001b	BMLT
0010b	BMGEU, BMC	1010b	BMGT
0011b	BMLTU, BMNC	1011b	BMLE
0100b	BMGTU	1100b	BMO
0101b	BMLEU	1101b	BMNO
0110b	BMPZ	1110b	Reserved
0111b	BMN	1111b	Reserved

(2) BM*Cnd* src, dest



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

cd[3:0]	BM*Cnd*	cd[3:0]	BM*Cnd*
0000b	BMEQ, BMZ	1000b	BMGE
0001b	BMNE, BMNZ	1001b	BMLT
0010b	BMGEU, BMC	1010b	BMGT
0011b	BMLTU, BMNC	1011b	BMLE
0100b	BMGTU	1100b	BMO
0101b	BMLEU	1101b	BMNO
0110b	BMPZ	1110b	Reserved
0111b	BMN	1111b	Reserved

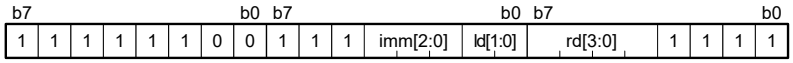
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

BNOT BNOT

Code Size

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

(1) BNOT src, dest



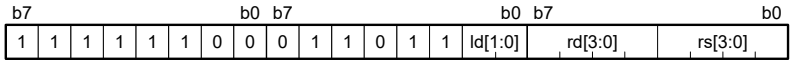


imm[2:0]	src	
000b to 111b	#IMM:3	0 to 7

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

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

(2) BNOT src, dest

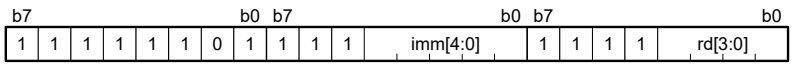




ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

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

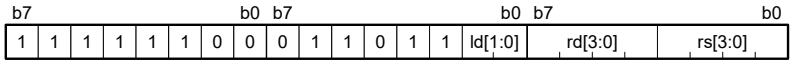
(3) BNOT src, dest



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

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

(4) BNOT src, dest



ld[1:0]	dest
11b	Rd

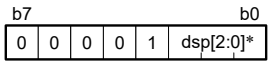
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

BRA BRA

Code Size

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

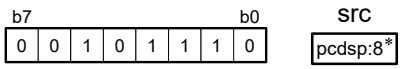
(1) BRA.S src



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

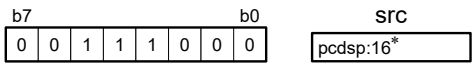
dsp[2:0]	Branch Distance
011b	3
100b	4
101b	5
110b	6
111b	7
000b	8
001b	9
010b	10

(2) BRA.B src



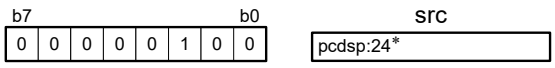
Note: * Address indicated by pcdsp:8 = src minus the address of the instruction

(3) BRA.W src



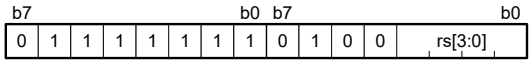
Note: * Address indicated by pcdsp:16 = src minus the address of the instruction

(4) BRA.A src



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

(5) BRA.L src



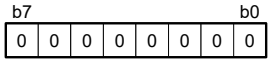
rs[3:0]	src	
0000b to 1111b	Rs	R0 (SP) to R15

BRK BRK

Code Size

Syntax	Code Size (Byte)
(1) BRK	1

(1) BRK

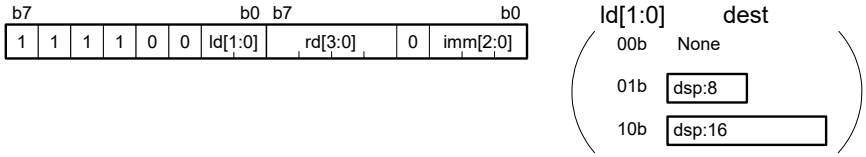


BSET BSET

Code Size

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

(1) BSET src, dest

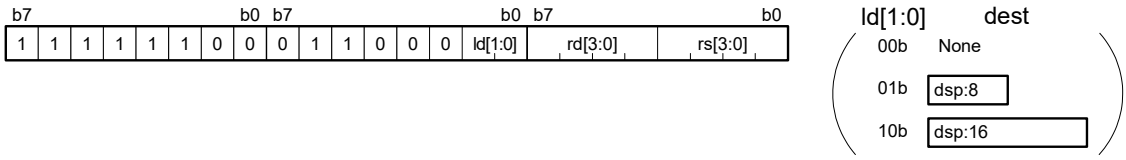


ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

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

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

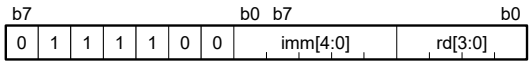
(2) BSET src, dest



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

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

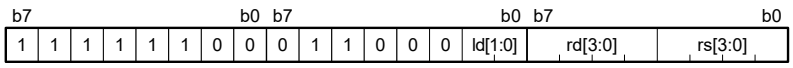
(3) BSET src, dest



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

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

(4) BSET src, dest



ld[1:0]	dest
11b	Rd

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

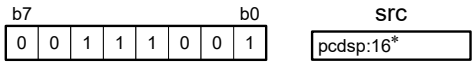
BSR

BSR

Code Size

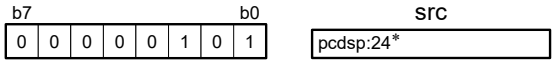
Syntax	src	Code Size (Byte)
(1) BSR.W src	pcdsp:16	3
(2) BSR.A src	pcdsp:24	4
(3) BSR.L src	Rs	2

(1) BSR.W src



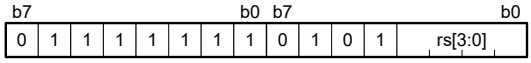
Note: * Address indicated by pcdsp:16 = src minus the address of the instruction

(2) BSR.A src



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

(3) BSR.L src



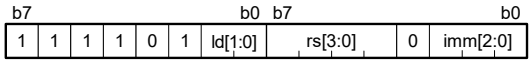
rs[3:0]	src	
0000b to 1111b	Rs	R0 (SP) to R15

BTST BTST

Code Size

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

(1) BTST src, src2



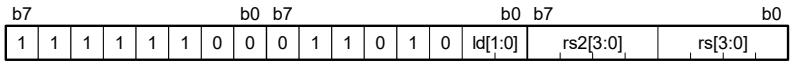


ld[1:0]	src2
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

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

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

(2) BTST src, src2

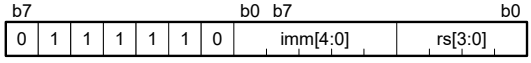




ld[1:0]	src2
00b	[Rs2]
01b	dsp:8[Rs2]
10b	dsp:16[Rs2]

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

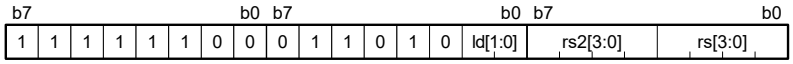
(3) BTST src, src2



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

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

(4) BTST src, src2



ld[1:0]	src2
11b	Rs2

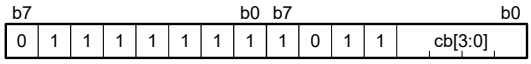
rs[3:0]/rs2[3:0]	src/src2	
0000b to 1111b	Rs/Rs2	R0 (SP) to R15

CLRPSW CLRPSW

Code Size

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

(1) CLRPSW dest



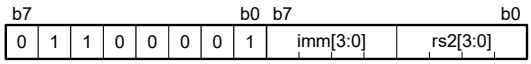
cb[3:0]	dest	
0000b	flag	C
0001b		Z
0010b		S
0011b		O
0100b		Reserved
0101b		Reserved
0110b		Reserved
0111b		Reserved
1000b		I
1001b		U
1010b		Reserved
1011b		Reserved
1100b		Reserved
1101b		Reserved
1110b		Reserved
1111b		Reserved

CMP CMP

Code Size

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

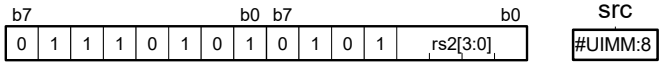
(1) CMP src, src2



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

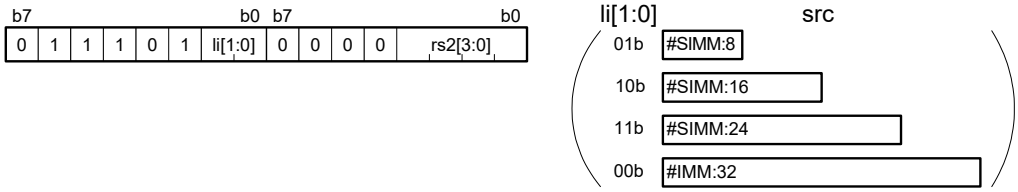
rs2[3:0]	src2	
0000b to 1111b	Rs	R0 (SP) to R15

(2) CMP src, src2



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

(3) CMP src, src2

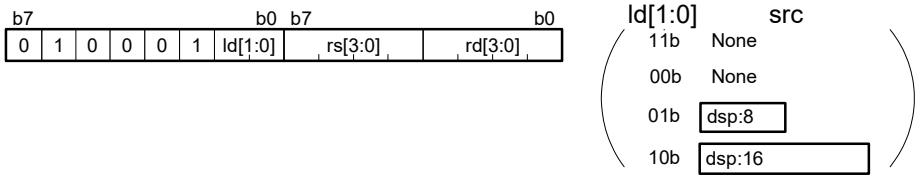


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

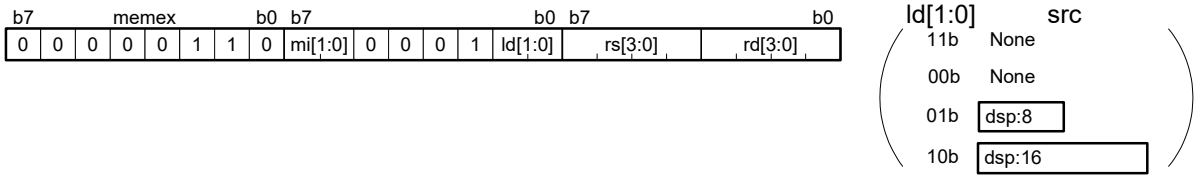
rs2[3:0]	src2	
0000b to 1111b	Rs	R0 (SP) to R15

(4) CMP src, src2

When memex == “UB” or src == Rs



When memex != “UB”



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

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

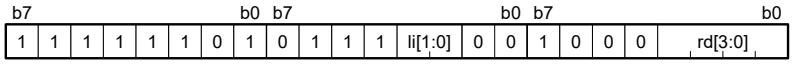
rs[3:0]/rs2[3:0]	src/src2	
0000b to 1111b	Rs/Rs2	R0 (SP) to R15

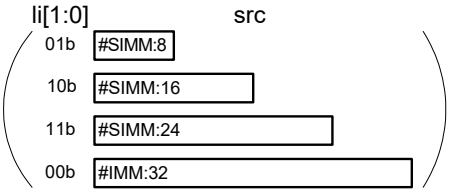
DIV DIV

Code Size

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

(1) DIV src, dest



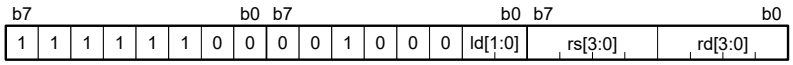


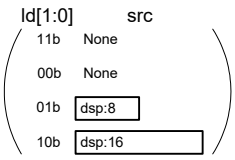
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

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

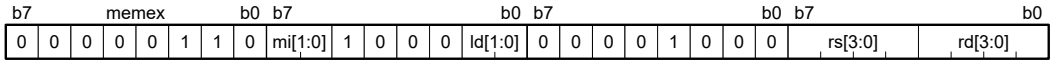
(2) DIV src, dest

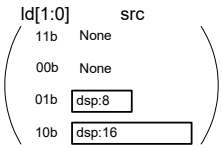
When memex == “UB” or src == Rs





When memex != “UB”





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

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

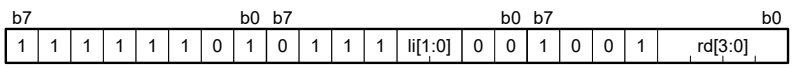
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

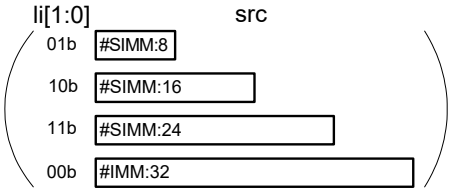
DIVU DIVU

Code Size

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

(1) DIVU src, dest



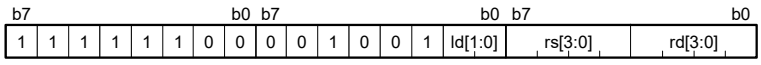


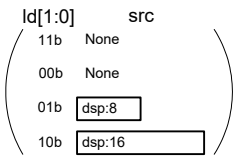
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

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

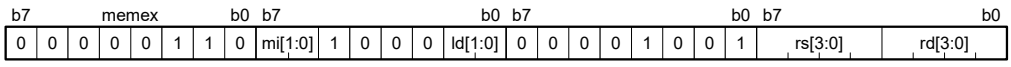
(2) DIVU src, dest

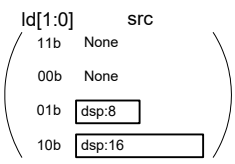
When memex == “UB” or src == Rs





When memex != “UB”





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

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

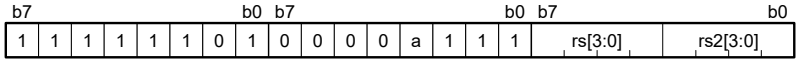
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

EMACA EMACA

Code Size

Syntax	src	dest2	Adest	Code Size (Byte)
(1) EMACA src, src2, Adest	Rs	Rs2	A0, A1	3

(1) EMACA src, src2, Adest



a	Adest
0b	A0
1b	A1

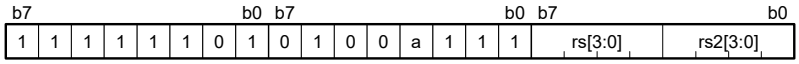
rs[3:0]/rs2[3:0]	src/src2	
0000b to 1111b	Rs/Rs2	R0 (SP) to R15

EMSBA EMSBA

Code Size

Syntax	src	dest2	Adest	Code Size (Byte)
(1) EMSBA src, src2, Adest	Rs	Rs2	A0, A1	3

(1) EMSBA src, src2, Adest



a	Adest
0b	A0
1b	A1

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

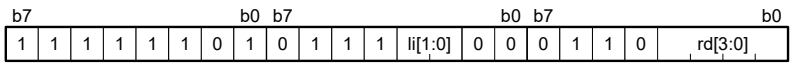
EMUL

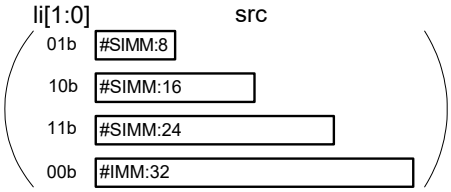
EMUL

Code Size

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

(1) EMUL src, dest



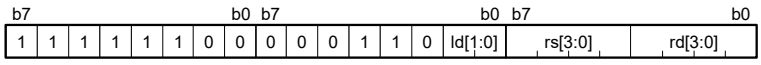


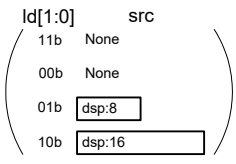
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

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

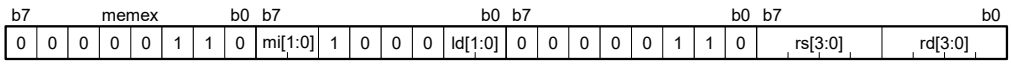
(2) EMUL src, dest

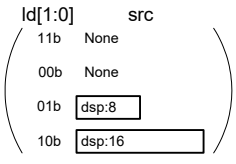
When memex == “UB” or src == Rs





When memex != “UB”





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

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

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

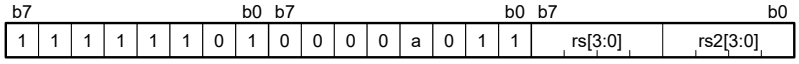
rd[3:0]	dest	
0000b to 1110b	Rd	R0 (SP) to R14

EMULA EMULA

Code Size

Syntax	src	dest2	Adest	Code Size (Byte)
(1) EMULA src, src2, Adest	Rs	Rs2	A0, A1	3

(1) EMULA src, src2, Adest



a	Adest
0b	A0
1b	A1

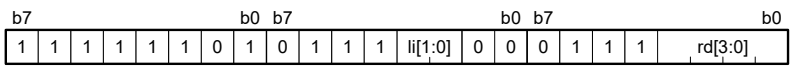
rs[3:0]/rs2[3:0]	src/src2	
0000b to 1111b	Rs/Rs2	R0 (SP) to R15

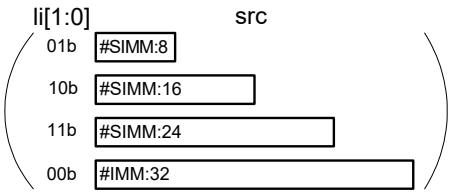
EMULU EMULU

Code Size

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

(1) EMULU src, dest



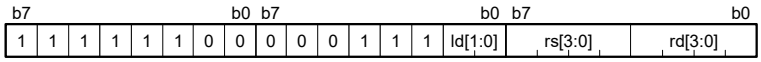


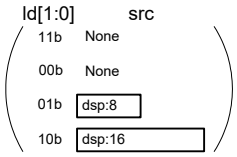
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

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

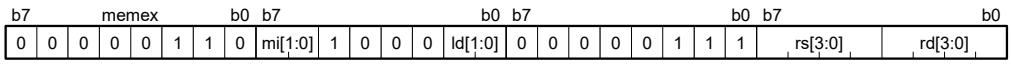
(2) EMULU src, dest

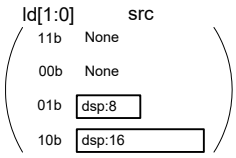
When memex == “UB” or src == Rs





When memex != “UB”





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

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

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

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

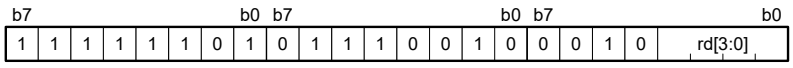
FADD

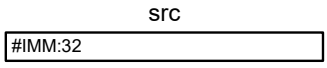
FADD

Code Size

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

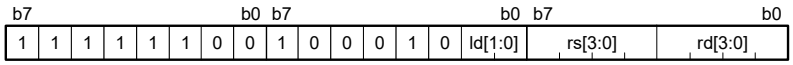
(1) FADD src, dest

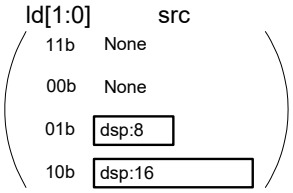




rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) FADD src, dest

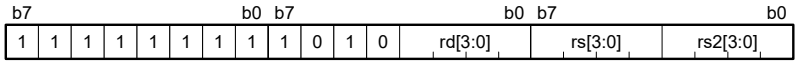




ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

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

(3) FADD 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

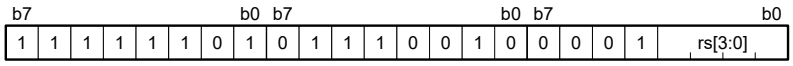
FCMP

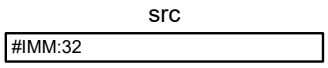
FCMP

Code Size

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

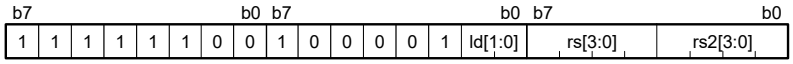
(1) FCMP src, src2

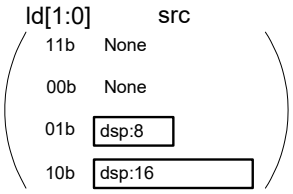




rs[3:0]	src2	
0000b to 1111b	Rs	R0 (SP) to R15

(2) FCMP src, src2





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

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

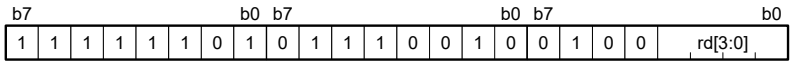
FDIV

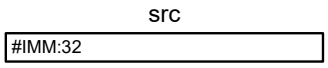
FDIV

Code Size

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

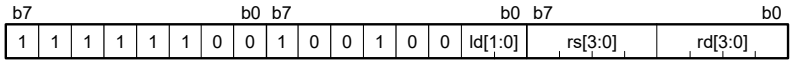
(1) FDIV src, dest

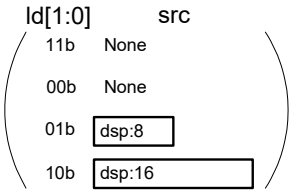




rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) FDIV src, dest





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

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

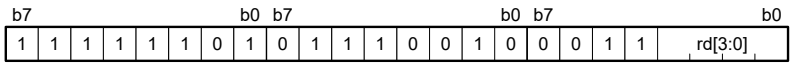
FMUL

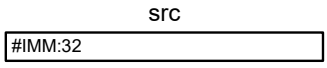
FMUL

Code Size

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

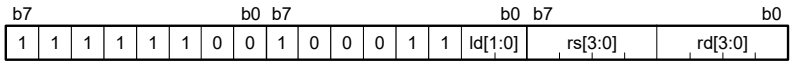
(1) FMUL src, dest

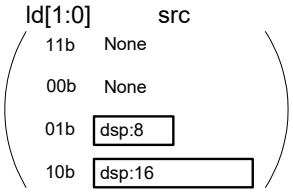




rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) FMUL src, dest

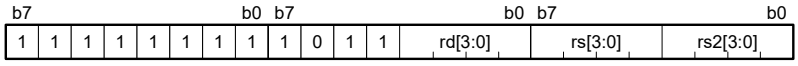




ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

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

(3) FMUL 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

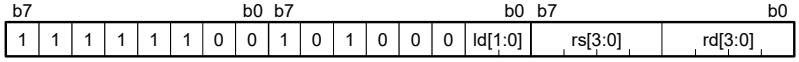
FSQRT

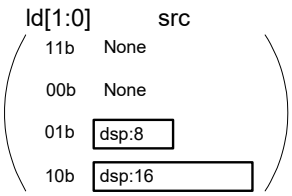
FSQRT

Code Size

Syntax	src	dest	Code Size (Byte)
(1) FSQRT src, dest	Rs	Rd	3
	[Rs].L	Rd	3
	dsp:8[Rs].L	Rd	4
	dsp:16[Rs].L	Rd	5

(1) FSQRT src, dest





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

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

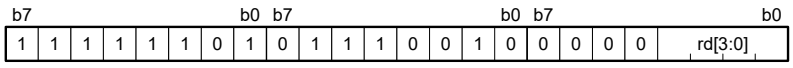
FSUB

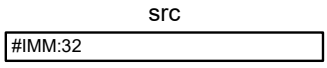
FSUB

Code Size

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

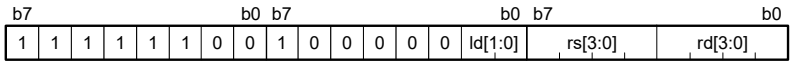
(1) FSUB src, dest

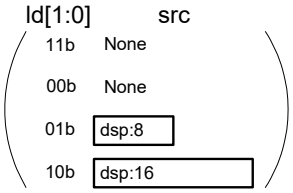




rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) FSUB src, dest

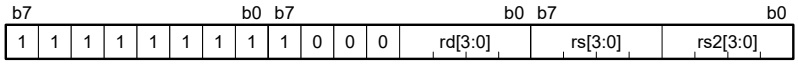




ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

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

(3) FSUB 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

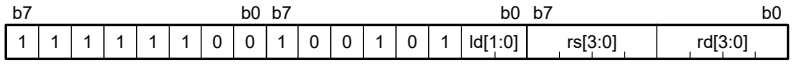
FTOI

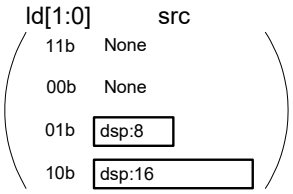
FTOI

Code Size

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

(1) FTOI src, dest





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

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

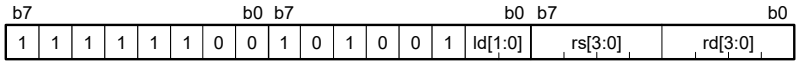
FTOU

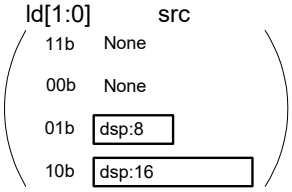
FTOU

Code Size

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

(1) FTOU src, dest





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

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

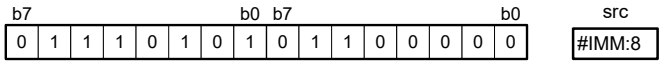
INT

INT

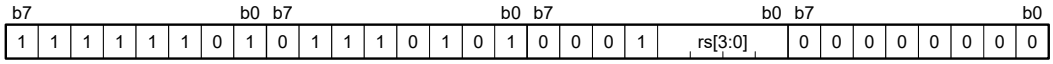
Code Size

Syntax	src	Code Size (Byte)
(1) INT src	#IMM:8	3
(2) INT src	Rs	4

(1) INT src



(2) INT src



ITOF

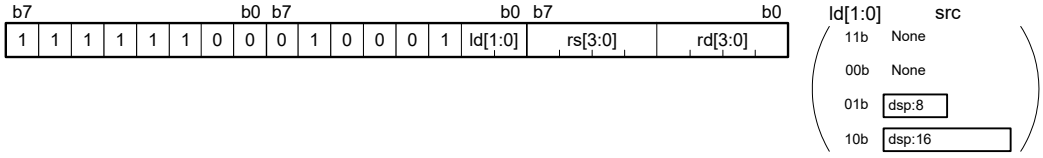
ITOF

Code Size

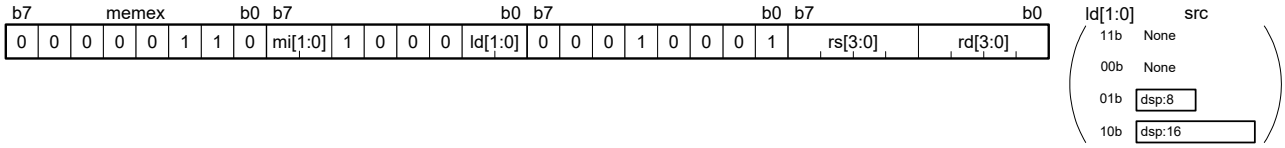
Syntax	src	dest	Code Size (Byte)
(1) ITOF src, dest	Rs	Rd	3
	[Rs].memex	Rd	3 (memex == “UB”) 4 (memex != “UB”)
	dsp:8[Rs].memex	Rd	4 (memex == “UB”) 5 (memex != “UB”)
	dsp:16[Rs].memex	Rd	5 (memex == “UB”) 6 (memex != “UB”)

(1) ITOF src, dest

When memex == “UB” or src == Rs



When memex != “UB”



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

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

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

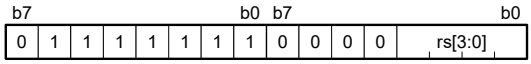
JMP

JMP

Code Size

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

(1) JMP src



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

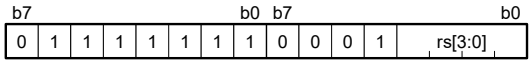
JSR

JSR

Code Size

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

(1) JSR src



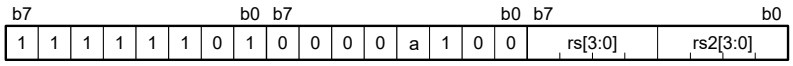
rs[3:0]	src	
0000b to 1111b	Rs	R0 (SP) to R15

MACHI MACHI

Code Size

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

(1) MACHI src, src2, Adest



a	Adest
0b	A0
1b	A1

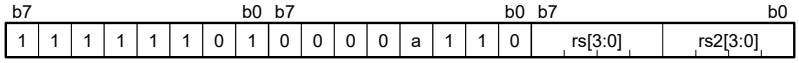
rs[3:0]/rs2[3:0]	src/src2	
0000b to 1111b	Rs/Rs2	R0 (SP) to R15

MACLH MACLH

Code Size

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

(1) MACLH src, src2, Adest



a	Adest
0b	A0
1b	A1

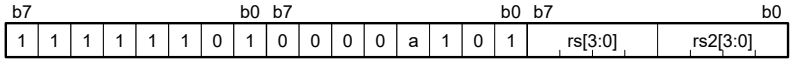
rs[3:0]/rs2[3:0]	src/src2	
0000b to 1111b	Rs/Rs2	R0 (SP) to R15

MACLO MACLO

Code Size

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

(1) MACLO src, src2, Adest



a	Adest
0b	A0
1b	A1

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

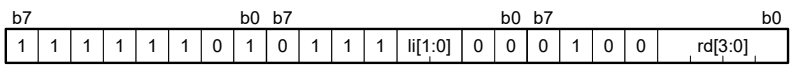
MAX

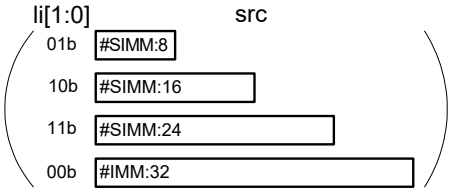
MAX

Code Size

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

(1) MAX src, dest



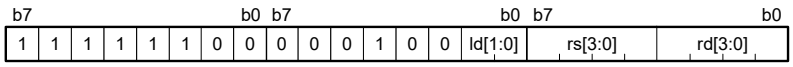


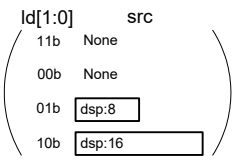
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

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

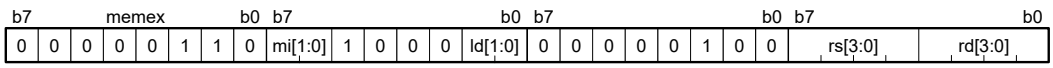
(2) MAX src, dest

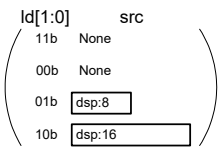
When memex == “UB” or src == Rs





When memex != “UB”





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

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

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

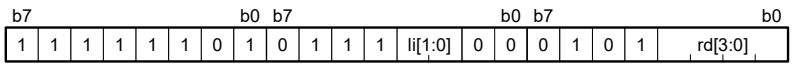
MIN

MIN

Code Size

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

(1) MIN src, dest



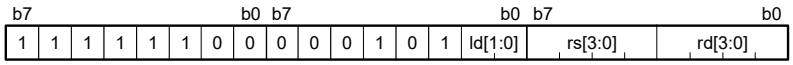


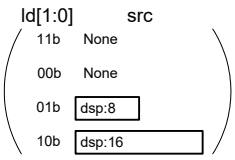
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

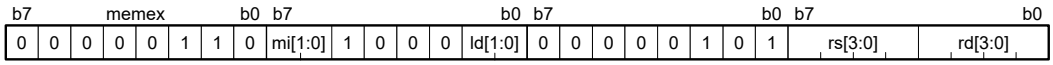
(2) MIN src, dest

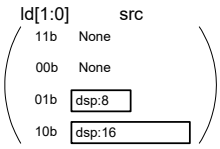
When memex == “UB” or src == Rs





When memex != “UB”





mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

MOV

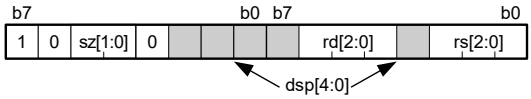
MOV

Code Size

Syntax	Size	Processing Size	src	dest	Code Size (Byte)
(1) MOV.size src, dest	B/W/L	size	Rs (Rs = R0 to R7)	dsp:5[Rd] (Rd = R0 to R7)	2
(2) MOV.size src, dest	B/W/L	L	dsp:5[Rs] (Rs = R0 to R7)	Rd (Rd = R0 to R7)	2
(3) MOV.size src, dest	L	L	#UIMM:4	Rd	2
(4) MOV.size src, dest	B	B	#IMM:8	dsp:5[Rd] (Rd = R0 to R7)	3
	W/L	size	#UIMM:8	dsp:5[Rd] (Rd = R0 to R7)	3
(5) MOV.size src, dest	L	L	#UIMM:8	Rd	3
(6) MOV.size src, dest	L	L	#SIMM:8	Rd	3
	L	L	#SIMM:16	Rd	4
	L	L	#SIMM:24	Rd	5
	L	L	#IMM:32	Rd	6
(7) MOV.size src, dest	B/W	L	Rs	Rd	2
	L	L	Rs	Rd	2
(8) MOV.size src, dest	B	B	#IMM:8	[Rd]	3
	B	B	#IMM:8	dsp:8[Rd]	4
	B	B	#IMM:8	dsp:16[Rd]	5
	W	W	#SIMM:8	[Rd]	3
	W	W	#SIMM:8	dsp:8[Rd]	4
	W	W	#SIMM:8	dsp:16[Rd]	5
	W	W	#IMM:16	[Rd]	4
	W	W	#IMM:16	dsp:8[Rd]	5
	W	W	#IMM:16	dsp:16[Rd]	6
	L	L	#SIMM:8	[Rd]	3
	L	L	#SIMM:8	dsp:8[Rd]	4
	L	L	#SIMM:8	dsp:16 [Rd]	5
	L	L	#SIMM:16	[Rd]	4
	L	L	#SIMM:16	dsp:8[Rd]	5
	L	L	#SIMM:16	dsp:16 [Rd]	6
	L	L	#SIMM:24	[Rd]	5
	L	L	#SIMM:24	dsp:8[Rd]	6
	L	L	#SIMM:24	dsp:16 [Rd]	7
	L	L	#IMM:32	[Rd]	6
	L	L	#IMM:32	dsp:8[Rd]	7
	L	L	#IMM:32	dsp:16 [Rd]	8
(9) MOV.size src, dest	B/W/L	L	[Rs]	Rd	2
	B/W/L	L	dsp:8[Rs]	Rd	3
	B/W/L	L	dsp:16[Rs]	Rd	4
(10) MOV.size src, dest	B/W/L	L	[Ri, Rb]	Rd	3
(11) MOV.size src, dest	B/W/L	size	Rs	[Rd]	2
	B/W/L	size	Rs	dsp:8[Rd]	3
	B/W/L	size	Rs	dsp:16[Rd]	4

Syntax	Size	Processing Size	src	dest	Code Size (Byte)
(12) MOV.size src, dest	B/W/L	size	Rs	[Ri, Rb]	3
(13) MOV.size src, dest	B/W/L	size	[Rs]	[Rd]	2
	B/W/L	size	[Rs]	dsp:8[Rd]	3
	B/W/L	size	[Rs]	dsp:16[Rd]	4
	B/W/L	size	dsp:8[Rs]	[Rd]	3
	B/W/L	size	dsp:8[Rs]	dsp:8[Rd]	4
	B/W/L	size	dsp:8[Rs]	dsp:16[Rd]	5
	B/W/L	size	dsp:16[Rs]	[Rd]	4
	B/W/L	size	dsp:16[Rs]	dsp:8[Rd]	5
	B/W/L	size	dsp:16[Rs]	dsp:16[Rd]	6
(14) MOV.size src, dest	B/W/L	size	Rs	[Rd+]	3
	B/W/L	size	Rs	[–Rd]	3
(15) MOV.size src, dest	B/W/L	L	[Rs+]	Rd	3
	B/W/L	L	[–Rs]	Rd	3

(1) MOV.size src, dest

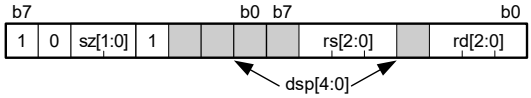


sz[1:0]	Size
00b	B
01b	W
10b	L

dsp[4:0]	dsp:5
00000b to 11111b	0 to 31

rs[2:0]/rd[2:0]	src/dest	
000b to 111b	Rs/Rd	R0 (SP) to R7

(2) MOV.size src, dest

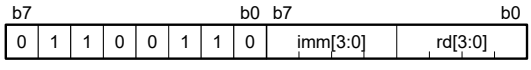


sz[1:0]	Size
00b	B
01b	W
10b	L

dsp[4:0]	dsp:5
00000b to 11111b	0 to 31

rs[2:0]/rd[2:0]	src/dest	
000b to 111b	Rs/Rd	R0 (SP) to R7

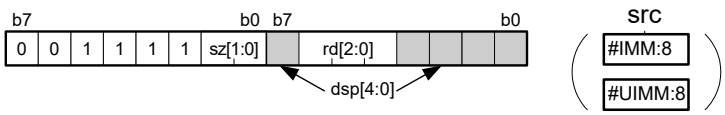
(3) MOV.size src, dest



imm[3:0]	src	
0000b to 1111b	#UIMM:4	0 to 15

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(4) MOV.size src, dest

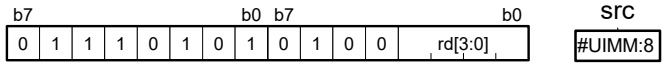


sz[1:0]	Size
00b	B
01b	W
10b	L

dsp[4:0]	dsp:5
00000b to 11111b	0 to 31

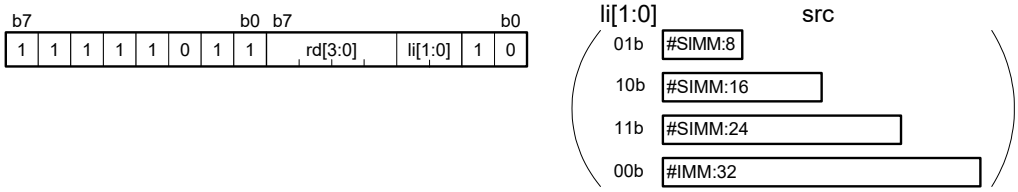
rd[2:0]	dest	
000b to 111b	Rd	R0 (SP) to R7

(5) MOV.size src, dest



rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

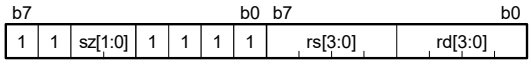
(6) MOV.size src, dest



li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

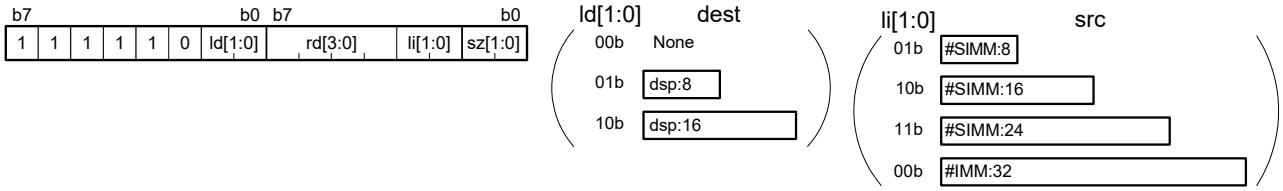
(7) MOV.size src, dest



sz[1:0]	Size
00b	B
01b	W
10b	L

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

(8) MOV.size src, dest



ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

sz[1:0]	Size
00b	B
01b	W
10b	L

(9) MOV.size src, dest

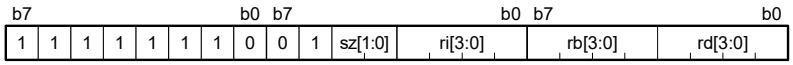


sz[1:0]	Size
00b	B
01b	W
10b	L

ld[1:0]	src
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

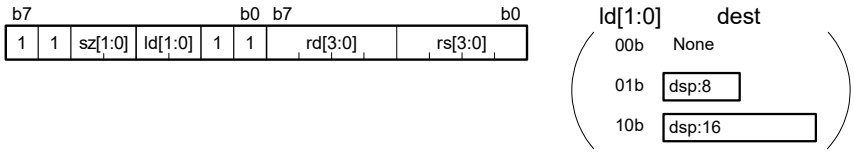
(10) MOV.size src, dest



sz[1:0]	Size
00b	B
01b	W
10b	L

ri[3:0]/rb[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Ri/Rb/Rd	R0 (SP) to R15

(11) MOV.size src, dest

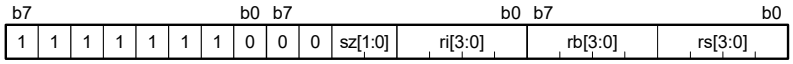


sz[1:0]	Size
00b	B
01b	W
10b	L

ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

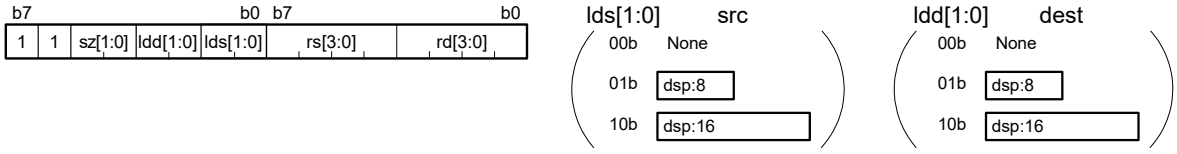
(12) MOV.size src, dest



sz[1:0]	Size
00b	B
01b	W
10b	L

rs[3:0]/ri[3:0]/rb[3:0]	src/dest	
0000b to 1111b	Rs/Ri/Rb	R0 (SP) to R15

(13) MOV.size src, dest

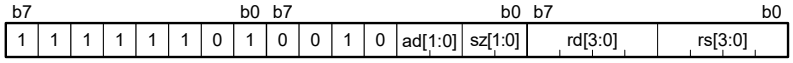


sz[1:0]	Size
00b	B
01b	W
10b	L

lds[1:0]/ldd[1:0]	src/dest
00b	[Rs]/[Rd]
01b	dsp:8[Rs]/dsp:8[Rd]
10b	dsp:16[Rs]/dsp:16[Rd]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

(14) MOV.size src, dest

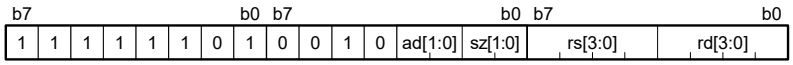


ad[1:0]	Addressing
00b	Rs, [Rd+]
01b	Rs, [-Rd]

sz[1:0]	Size
00b	B
01b	W
10b	L

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

(15) MOV.size src, dest



ad[1:0]	Addressing
10b	[Rs+], Rd
11b	[-Rs], Rd

sz[1:0]	Size
00b	B
01b	W
10b	L

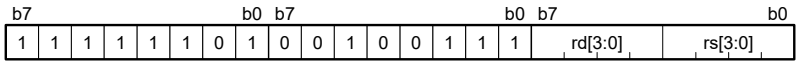
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

MOVCO MOVCO

Code Size

Syntax	Size	Processing Size	src	dest	Code Size (Byte)
(1) MOVCO src, dest	L	L	Rs	[Rd]	3

(1) MOVCO src, dest



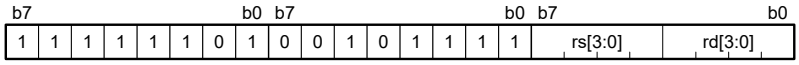
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

MOVLI MOVLI

Code Size

Syntax	Size	Processing Size	src	dest	Code Size (Byte)
(1) MOVLI src, dest	L	L	[Rs]	Rd	3

(1) MOVLI src, dest



rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

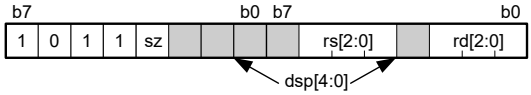
MOVU

MOVU

Code Size

Syntax	Size	Processing Size	src	dest	Code Size (Byte)
(1) MOVU.size src, dest	B/W	L	dsp:5[Rs] (Rs = R0 to R7)	Rd (Rd = R0 to R7)	2
(2) MOVU.size src, dest	B/W	L	Rs	Rd	2
	B/W	L	[Rs]	Rd	2
	B/W	L	dsp:8[Rs]	Rd	3
	B/W	L	dsp:16[Rs]	Rd	4
(3) MOVU.size src, dest	B/W	L	[Ri, Rb]	Rd	3
(4) MOVU.size src, dest	B/W	L	[Rs+]	Rd	3
	B/W	L	[–Rs]	Rd	3

(1) MOVU.size src, dest

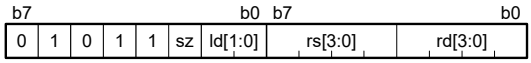


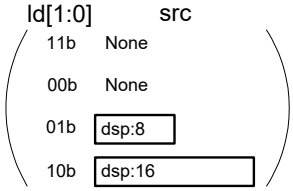
sz	Size
0b	B
1b	W

dsp[4:0]	dsp:5
00000b to 11111b	0 to 31

rs[2:0]/rd[2:0]	src/dest	
000b to 111b	Rs/Rd	R0 (SP) to R7

(2) MOVU.size src, dest



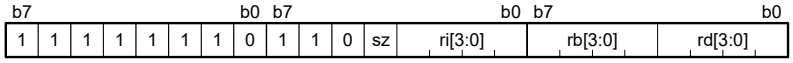


sz	Size
0b	B
1b	W

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

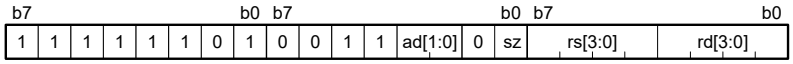
(3) MOVU.size src, dest



sz	Size
0b	B
1b	W

ri[3:0]/rb[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Ri/Rb/Rd	R0 (SP) to R15

(4) MOVU.size src, dest



ad[1:0]	Addressing
10b	[Rs+], Rd
11b	[-Rs], Rd

sz	Size
0b	B
1b	W

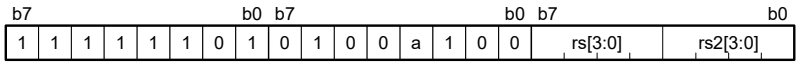
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

MSBHI MSBHI

Code Size

Syntax	src	src2	Adest	Code Size (Byte)
(1) MSBHI src, src2, Adest	Rs	Rs2	A0, A1	3

(1) MSBHI src, src2, Adest



a	Adest
0b	A0
1b	A1

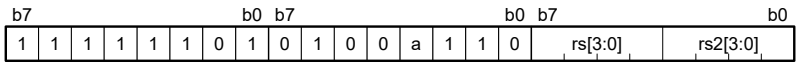
rs[3:0]/rs2[3:0]	src/src2	
0000b to 1111b	Rs/Rs2	R0 (SP) to R15

MSBLH MSBLH

Code Size

Syntax	src	src2	Adest	Code Size (Byte)
(1) MSBLH src, src2, Adest	Rs	Rs2	A0, A1	3

(1) MSBLH src, src2, Adest



a	Adest
0b	A0
1b	A1

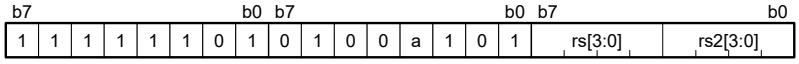
rs[3:0]/rs2[3:0]	src/src2	
0000b to 1111b	Rs/Rs2	R0 (SP) to R15

MSBLO MSBLO

Code Size

Syntax	src	src2	Adest	Code Size (Byte)
(1) MSBLO src, src2, Adest	Rs	Rs2	A0, A1	3

(1) MSBLO src, src2, Adest



a	Adest
0b	A0
1b	A1

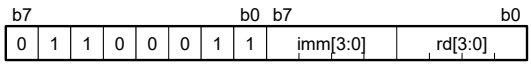
rs[3:0]/rs2[3:0]	src/src2	
0000b to 1111b	Rs/Rs2	R0 (SP) to R15

MUL MUL

Code Size

Syntax	src	src2	dest	Code Size (Byte)
(1) MUL src, dest	#UIMM:4	–	Rd	2
(2) MUL src, dest	#SIMM:8	–	Rd	3
	#SIMM:16	–	Rd	4
	#SIMM:24	–	Rd	5
	#IMM:32	–	Rd	6
(3) MUL src, dest	Rs	–	Rd	2
	[Rs].memex	–	Rd	2 (memex == “UB”) 3 (memex != “UB”)
	dsp:8[Rs].memex	–	Rd	3 (memex == “UB”) 4 (memex != “UB”)
	dsp:16[Rs].memex	–	Rd	4 (memex == “UB”) 5 (memex != “UB”)
(4) MUL src, src2, dest	Rs	Rs2	Rd	3

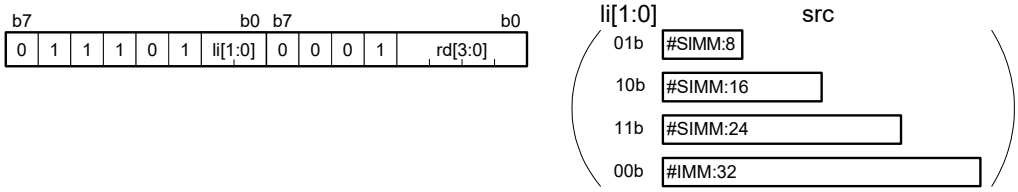
(1) MUL src, dest



imm[3:0]	src	
0000b to 1111b	#UIMM:4	0 to 15

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) MUL src, dest

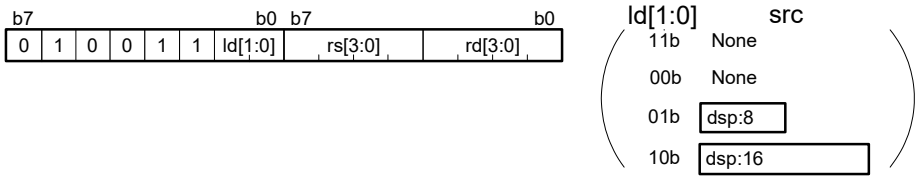


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

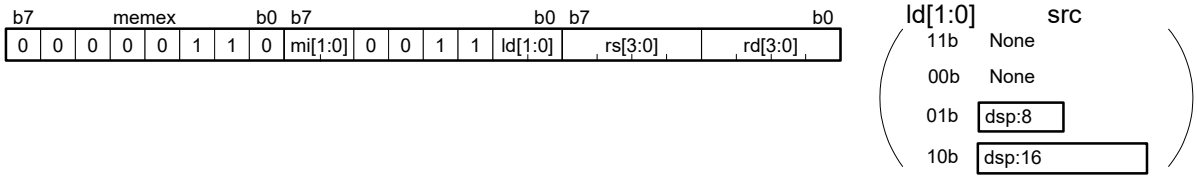
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(3) MUL src, dest

When memex == “UB” or src == Rs



When memex != “UB”

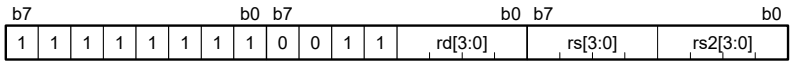


mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

(4) MUL src, src2, dest



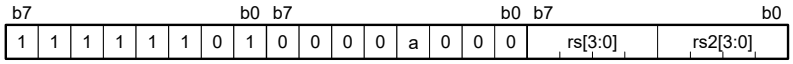
rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b to 1111b	Rs/Rs2/Rd	R0 (SP) to R15

MULHI MULHI

Code Size

Syntax	src	src2	Adest	Code Size (Byte)
(1) MULHI src, src2, Adest	Rs	Rs2	A0, A1	3

(1) MULHI src, src2, Adest



a	Adest
0b	A0
1b	A1

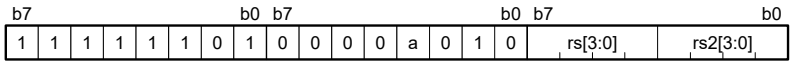
rs[3:0]/rs2[3:0]	src/src2	
0000b to 1111b	Rs/Rs2	R0 (SP) to R15

MULLH MULLH

Code Size

Syntax	src	src2	Adest	Code Size (Byte)
(1) MULLH src, src2, Adest	Rs	Rs2	A0, A1	3

(1) MULLH src, src2, Adest



a	Adest
0b	A0
1b	A1

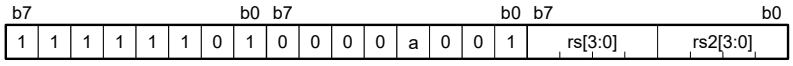
rs[3:0]/rs2[3:0]	src/src2	
0000b to 1111b	Rs/Rs2	R0 (SP) to R15

MULLO MULLO

Code Size

Syntax	src	src2	Adest	Code Size (Byte)
(1) MULLO src, src2, Adest	Rs	Rs2	A0, A1	3

(1) MULLO src, src2, Adest



a	Adest
0b	A0
1b	A1

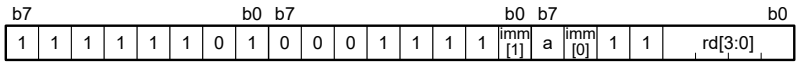
rs[3:0]/rs2[3:0]	src/src2	
0000b to 1111b	Rs/Rs2	R0 (SP) to R15

MVFACGU MVFACGU

Code Size

Syntax	src	Asrc	dest	Code Size (Byte)
(1) MVFACGU src, Asrc, dest	#IMM:2	A0, A1	Rd	3

(1) MVFACGU src, Asrc, dest



a	Asrc
0b	A0
1b	A1

imm[1:0]	src	
00	#IMM:2	2
01		—
10		0
11		1

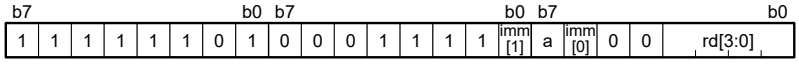
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

MVFACHI MVFACHI

Code Size

Syntax	src	Asrc	dest	Code Size (Byte)
(1) MVFACHI src, Asrc, dest	#IMM:2	A0, A1	Rd	3

(1) MVFACHI src, Asrc, dest



a	Asrc
0b	A0
1b	A1

imm[1:0]	src	
00	#IMM:2	2
01		—
10		0
11		1

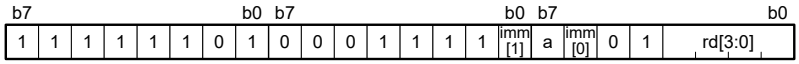
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

MVFACLO MVFACLO

Code Size

Syntax	src	Asrc	dest	Code Size (Byte)
(1) MVFACLO src, Asrc, dest	#IMM:2	A0, A1	Rd	3

(1) MVFACLO src, Asrc, dest



a	Asrc
0b	A0
1b	A1

imm[1:0]	src	
00	#IMM:2	2
01		—
10		0
11		1

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

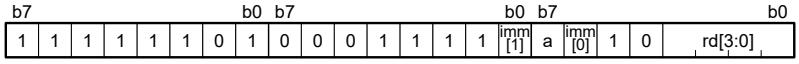
MVFACMI

MVFACMI

Code Size

Syntax	src	Asrc	dest	Code Size (Byte)
(1) MVFACMI src, Asrc, dest	#IMM:2	A0, A1	Rd	3

(1) MVFACMI src, Asrc, dest



a	Asrc
0b	A0
1b	A1

imm[1:0]	src	
00	#IMM:2	2
01		—
10		0
11		1

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

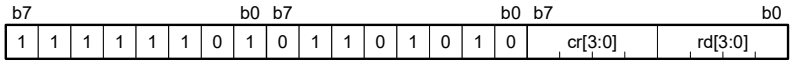
MVFC

MVFC

Code Size

Syntax	src	dest	Code Size (Byte)
(1) MVFC src, dest	Rx	Rd	3

(1) MVFC src, dest



cr[3:0]	src	
0000b	Rx	PSW
0001b		PC
0010b		USP
0011b		FPSW
0100b		Reserved
0101b		IVC
0110b		Reserved
0111b		Reserved
1000b		BPSW
1001b		BPC
1010b		ISP
1011b		FINTV
1100b		INTB
1101b		EXTB
1110b		WPRI
1111b		WID

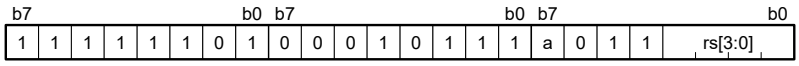
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

MVTACGU MVTACGU

Code Size

Syntax	src	Adest	Code Size (Byte)
(1) MVTACGU src, Adest	Rs	A0, A1	3

(1) MVTACGU src, Adest



a	Adest
0b	A0
1b	A1

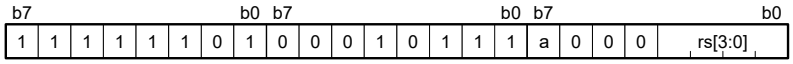
rs[3:0]	src	
0000b to 1111b	Rs	R0 (SP) to R15

MVTACHI MVTACHI

Code Size

Syntax	src	Adest	Code Size (Byte)
(1) MVTACHI src, Adest	Rs	A0, A1	3

(1) MVTACHI src, Adest



a	Adest
0b	A0
1b	A1

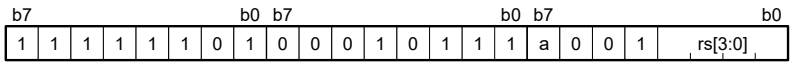
rs[3:0]	src	
0000b to 1111b	Rs	R0 (SP) to R15

MVTACLO MVTACLO

Code Size

Syntax	src	Adest	Code Size (Byte)
(1) MVTACLO src, Adest	Rs	A0, A1	3

(1) MVTACLO src, Adest



a	Adest
0b	A0
1b	A1

rs[3:0]	src	
0000b to 1111b	Rs	R0 (SP) to R15

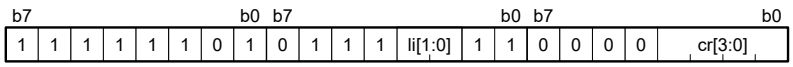
MVTC

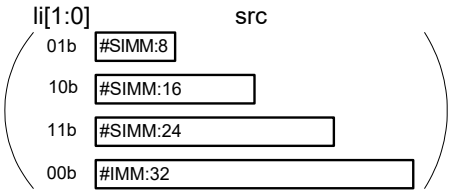
MVTC

Code Size

Syntax	src	dest	Code Size (Byte)
(1) MVTC src, dest	#SIMM:8	Rx	4
	#SIMM:16	Rx	5
	#SIMM:24	Rx	6
	#IMM:32	Rx	7
(2) MVTC src, dest	Rs	Rx	3

(1) MVTC src, dest

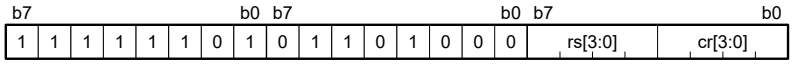




li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

cr[3:0]	dest	
0000b	Rx	PSW
0001b		Reserved
0010b		USP
0011b		FPSW
0100b		Reserved
0101b		IVC
0110b		Reserved
0111b		Reserved
1000b		BPSW
1001b		BPC
1010b		ISP
1011b		FINTV
1100b		INTB
1101b		EXTB
1110b		WPRI
1111b		WID

(2) MVTC src, dest



cr[3:0]	dest	
0000b	Rx	PSW
0001b		Reserved
0010b		USP
0011b		FPSW
0100b		Reserved
0101b		IVC
0110b		Reserved
0111b		Reserved
1000b		BPSW
1001b		BPC
1010b		ISP
1011b		FINTV
1100b		INTB
1101b		EXTB
1110b		WPRI
1111b		WID

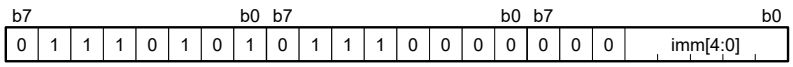
rs[3:0]	src	
0000b to 1111b	Rs	R0 (SP) to R15

MVTIPL MVTIPL

Code Size

Syntax	src	Code Size (Byte)
(1) MVTIPL src	#IMM:5	3

(1) MVTIPL src



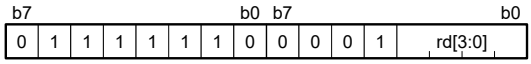
imm[4:0]	#IMM:5
00000b to 11111b	0 to 31

NEG NEG

Code Size

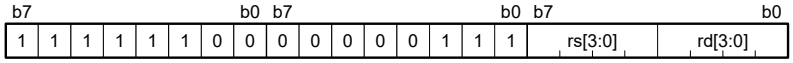
Syntax		src	dest	Code Size (Byte)
(1) NEG	dest	−	Rd	2
(2) NEG	src, dest	Rs	Rd	3

(1) NEG dest



rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) NEG src, dest



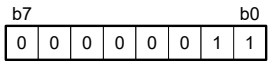
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

NOP NOP

Code Size

Syntax	Code Size (Byte)
(1) NOP	1

(1) NOP



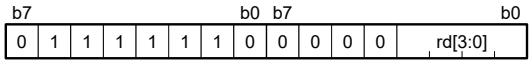
NOT

NOT

Code Size

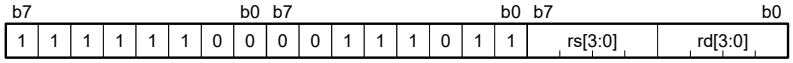
Syntax		src	dest	Code Size (Byte)
(1) NOT	dest	−	Rd	2
(2) NOT	src, dest	Rs	Rd	3

(1) NOT dest



rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) NOT src, dest



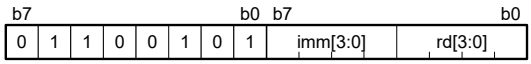
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

OR OR

Code Size

Syntax		src	src2	dest	Code Size (Byte)
(1) OR	src, dest	#UIMM:4	–	Rd	2
(2) OR	src, dest	#SIMM:8	–	Rd	3
		#SIMM:16	–	Rd	4
		#SIMM:24	–	Rd	5
		#IMM:32	–	Rd	6
(3) OR	src, dest	Rs	–	Rd	2
		[Rs].memex	–	Rd	2 (memex == “UB”) 3 (memex != “UB”)
		dsp:8[Rs].memex	–	Rd	3 (memex == “UB”) 4 (memex != “UB”)
		dsp:16[Rs].memex	–	Rd	4 (memex == “UB”) 5 (memex != “UB”)
(4) OR	src, src2, dest	Rs	Rs2	Rd	3

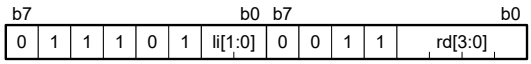
(1) OR src, dest

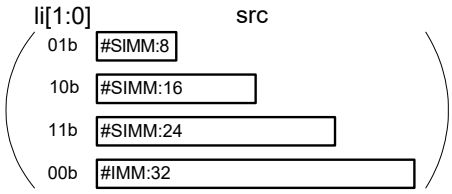


imm[3:0]	src	
0000b to 1111b	#UIMM:4	0 to 15

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) OR src, dest



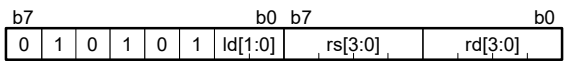


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

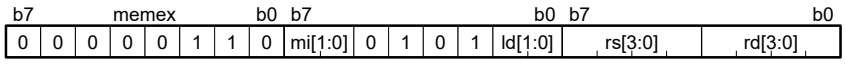
(3) OR src, dest

When memex == “UB” or src == Rs





When memex != “UB”



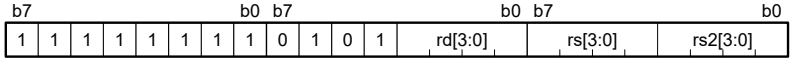


mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

(4) OR src, src2, dest



rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b to 1111b	Rs/Rs2/Rd	R0 (SP) to R15

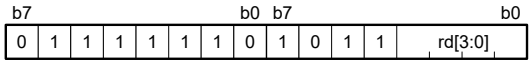
POP

POP

Code Size

Syntax	dest	Code Size (Byte)
(1) POP dest	Rd	2

(1) POP dest



rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

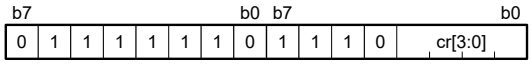
POPC

POPC

Code Size

Syntax	dest	Code Size (Byte)
(1) POPC dest	Rx	2

(1) POPC dest



cr[3:0]	dest	
0000b	Rx	PSW
0001b		Reserved
0010b		USP
0011b		FPSW
0100b		Reserved
0101b		IVC
0110b		Reserved
0111b		Reserved
1000b		BPSW
1001b		BPC
1010b		ISP
1011b		FINTV
1100b		INTB
1101b		EXTB
1110b		WPRI
1111b		WID

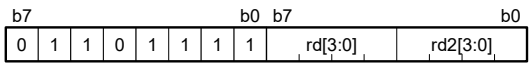
POPM

POPM

Code Size

Syntax	dest	dest2	Code Size (Byte)
(1) POPM dest-dest2	Rd	Rd2	2

(1) POPM dest-dest2



rd[3:0]	dest	
0001b to 1110b	Rd	R1 to R14

rd2[3:0]	dest2	
0010b to 1111b	Rd2	R2 to R15

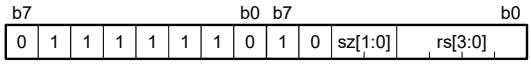
PUSH

PUSH

Code Size

Syntax	src	Code Size (Byte)
(1) PUSH.size src	Rs	2
(2) PUSH.size src	[Rs]	2
	dsp:8[Rs]	3
	dsp:16[Rs]	4

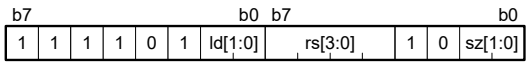
(1) PUSH.size src



sz[1:0]	Size
00b	B
01b	W
10b	L

rs[3:0]	src	
0000b to 1111b	Rs	R0 (SP) to R15

(2) PUSH.size src





ld[1:0]	src
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]	src	
0000b to 1111b	Rs	R0 (SP) to R15

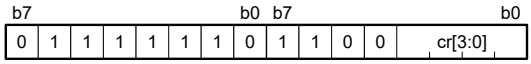
sz[1:0]	Size
00b	B
01b	W
10b	L

PUSHC PUSHC

Code Size

Syntax	src	Code Size (Byte)
(1) PUSHC src	Rx	2

(1) PUSHC src



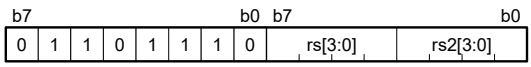
cr[3:0]	src	
0000b	Rx	PSW
0001b		PC
0010b		USP
0011b		FPSW
0100b		Reserved
0101b		IVC
0110b		Reserved
0111b		Reserved
1000b		BPSW
1001b		BPC
1010b		ISP
1011b		FINTV
1100b		INTB
1101b		EXTB
1110b		WPRI
1111b		WID

PUSHM PUSHM

Code Size

Syntax	src	src2	Code Size (Byte)
(1) PUSHM src-src2	Rs	Rs2	2

(1) PUSHM src-src2



rs[3:0]	src	
0001b to 1110b	Rs	R1 to R14

rs2[3:0]	src2	
0010b to 1111b	Rs2	R2 to R15

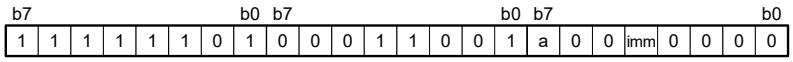
RACL

RACL

Code Size

Syntax	src	Adest	Code Size (Byte)
(1) RACL src, Adest	#IMM:1 (IMM:1 = 1, 2)	A0, A1	3

(1) RACL src, Adest



a	Adest
0b	A0
1b	A1

imm	src/src2	
0b,1b	#IMM:1	1, 2

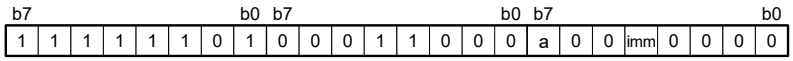
RACW

RACW

Code Size

Syntax	src	Adest	Code Size (Byte)
(1) RACW src, Adest	#IMM:1 (IMM:1 = 1, 2)	A0, A1	3

(1) RACW src, Adest



a	Adest
0b	A0
1b	A1

imm	src	
0b, 1b	#IMM:1	1, 2

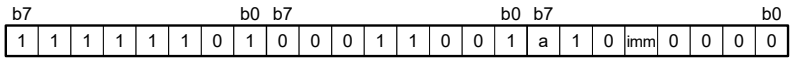
RDACL

RDACL

Code Size

Syntax	src	Adest	Code Size (Byte)
(1) RDACL src, Adest	#IMM:1 (IMM:1 = 1, 2)	A0, A1	3

(1) RDACL src, Adest



a	Adest
0b	A0
1b	A1

imm	src/src2	
0b,1b	#IMM:1	1, 2

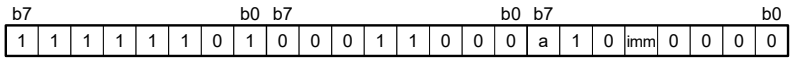
RDACW

RDACW

Code Size

Syntax	src	Adest	Code Size (Byte)
(1) RDACW src, Adest	#IMM:1 (IMM:1 = 1, 2)	A0, A1	3

(1) RDACW src, Adest



a	Adest
0b	A0
1b	A1

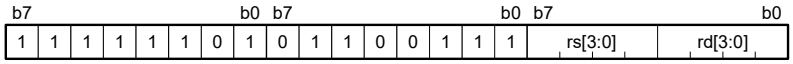
imm	src/src2	
0b, 1b	#IMM:1	1, 2

REVL REVL

Code Size

Syntax	src	dest	Code Size (Byte)
(1) REVL src, dest	Rs	Rd	3

(1) REVL src, dest



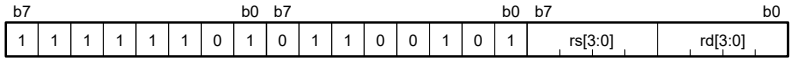
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

REVW REVW

Code Size

Syntax	src	dest	Code Size (Byte)
(1) REVW src, dest	Rs	Rd	3

(1) REVW src, dest



rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

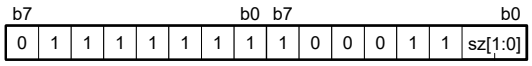
RMPA

RMPA

Code Size

Syntax	Size	Code Size (Byte)
(1) RMPA.size	B	2
	W	2
	L	2

(1) RMPA.size



sz[1:0]	Size
00b	B
01b	W
10b	L

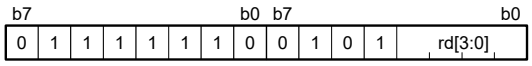
ROLC

ROLC

Code Size

Syntax	dest	Code Size (Byte)
(1) ROLC dest	Rd	2

(1) ROLC dest



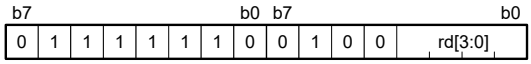
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

RORC RORC

Code Size

Syntax	dest	Code Size (Byte)
(1) RORC dest	Rd	2

(1) RORC dest



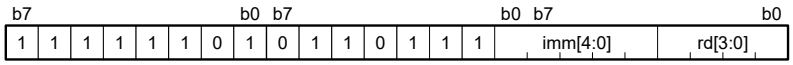
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

ROTL ROTL

Code Size

Syntax	src	dest	Code Size (Byte)
(1) ROTL src, dest	#IMM:5	Rd	3
(2) ROTL src, dest	Rs	Rd	3

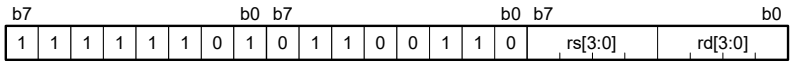
(1) ROTL src, dest



imm[4:0]	src	
00000b to 11111b	#IMM:5	0 to 31

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) ROTL src, dest



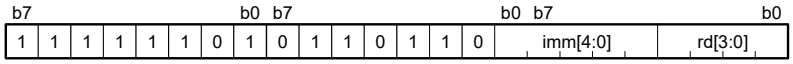
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

ROTR ROTR

Code Size

Syntax	src	dest	Code Size (Byte)
(1) ROTR src, dest	#IMM:5	Rd	3
(2) ROTR src, dest	Rs	Rd	3

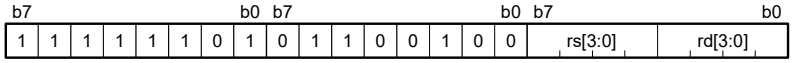
(1) ROTR src, dest



imm[4:0]	src	
00000b to 11111b	#IMM:5	0 to 31

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) ROTR src, dest



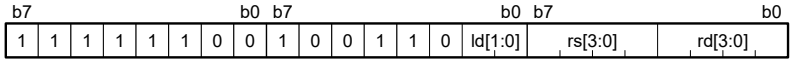
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

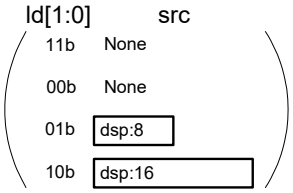
ROUND ROUND

Code Size

Syntax	src	dest	Code Size (Byte)
(1) ROUND src, dest	Rs	Rd	3
	[Rs].L	Rd	3
	dsp:8[Rs].L	Rd	4
	dsp:16[Rs].L	Rd	5

(1) ROUND src, dest





ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

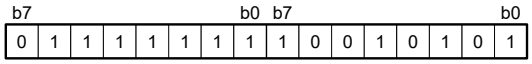
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

RTE RTE

Code Size

Syntax	Code Size (Byte)
(1) RTE	2

(1) RTE



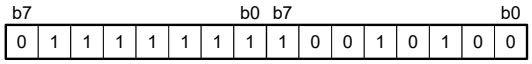
RTFI

RTFI

Code Size

Syntax	Code Size (Byte)
(1) RTFI	2

(1) RTFI



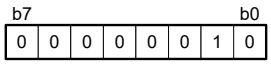
RTS

RTS

Code Size

Syntax	Code Size (Byte)
(1) RTS	1

(1) RTS



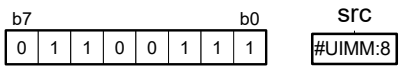
RTSD

RTSD

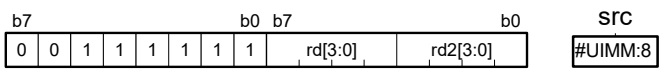
Code Size

Syntax	src	dest	dest2	Code Size (Byte)
(1) RTSD src	#UIMM:8	–	–	2
(2) RTSD src, dest-dest2	#UIMM:8	Rd	Rd2	3

(1) RTSD src



(2) RTSD src, dest-dest2



rd[3:0]/rd2[3:0]	dest/dest2	
0001b to 1111b	Rd/Rd2	R1 to R15

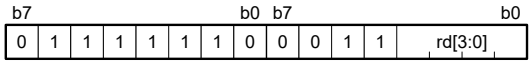
SAT

SAT

Code Size

Syntax	dest	Code Size (Byte)
(1) SAT dest	Rd	2

(1) SAT dest



rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

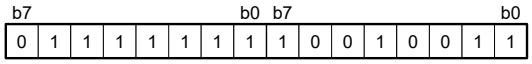
SATR

SATR

Code Size

Syntax	Code Size (Byte)
(1) SATR	2

(1) SATR

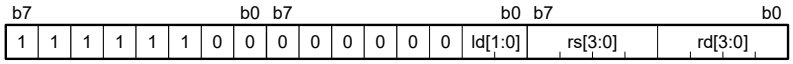


SBB SBB

Code Size

Syntax		src	dest	Code Size (Byte)
(1) SBB	src, dest	Rs	Rd	3
(2) SBB	src, dest	[Rs].L	Rd	4
		dsp:8[Rs].L	Rd	5
		dsp:16[Rs].L	Rd	6

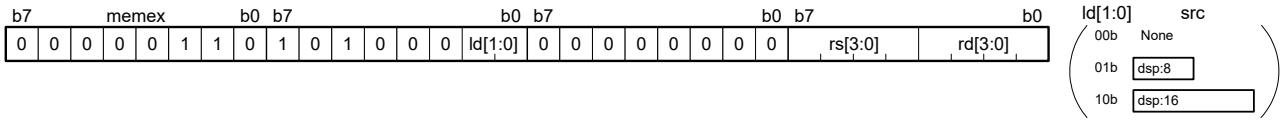
(1) SBB src, dest



ld[1:0]	src
11b	Rs

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

(2) SBB src, dest



ld[1:0]	src
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

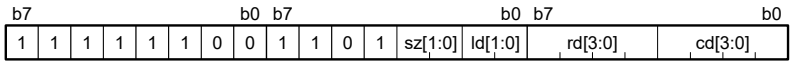
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

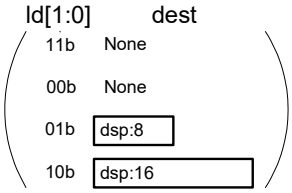
SC*Cnd* SC*Cnd*

Code Size

Syntax	Size	dest	Code Size (Byte)
(1) SC*Cnd*.size dest	L	Rd	3
	B/W/L	[Rd]	3
	B/W/L	dsp:8[Rd]	4
	B/W/L	dsp:16[Rd]	5

(1) SC*Cnd*.size dest





sz[1:0]	Size
00b	B
01b	W
10b	L

ld[1:0]	dest
11b	Rd
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

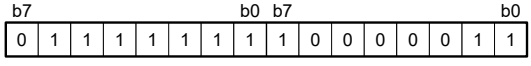
cd[3:0]	SC*Cnd*	cd[3:0]	SC*Cnd*
0000b	SCEQ, SCZ	1000b	SCGE
0001b	SCNE, SCNZ	1001b	SCLT
0010b	SCGEU, SCC	1010b	SCGT
0011b	SCLTU, SCNC	1011b	SCLE
0100b	SCGTU	1100b	SCO
0101b	SCLEU	1101b	SCNO
0110b	SCPZ	1110b	Reserved
0111b	SCN	1111b	Reserved

SCMPU SCMPU

Code Size

Syntax	Code Size (Byte)
(1) SCMPU	2

(1) SCMPU

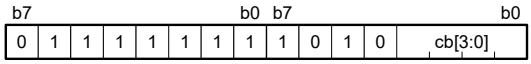


SETPSW SETPSW

Code Size

Syntax	dest	Code Size (Byte)
(1) SETPSW dest	flag	2

(1) SETPSW dest



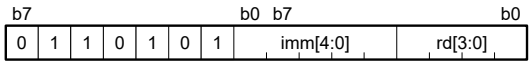
cb[3:0]	dest	
0000b	flag	C
0001b		Z
0010b		S
0011b		O
0100b		Reserved
0101b		Reserved
0110b		Reserved
0111b		Reserved
1000b		I
1001b		U
1010b		Reserved
1011b		Reserved
1100b		Reserved
1101b		Reserved
1110b		Reserved
1111b		Reserved

SHAR SHAR

Code Size

Syntax	src	src2	dest	Code Size (Byte)
(1) SHAR src, dest	#IMM:5	–	Rd	2
(2) SHAR src, dest	Rs	–	Rd	3
(3) SHAR src, src2, dest	#IMM:5	Rs	Rd	3

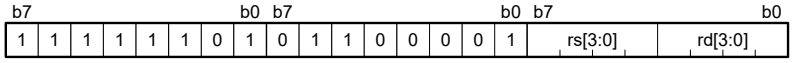
(1) SHAR src, dest



imm[4:0]	src	
00000b to 11111b	#IMM:5	0 to 31

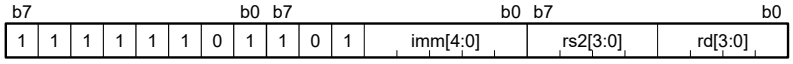
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) SHAR src, dest



rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

(3) SHAR src, src2, dest



imm[4:0]	src	
00000b to 11111b	#IMM:5	0 to 31

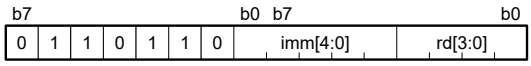
rs2[3:0]/rd[3:0]	src2/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

SHLL SHLL

Code Size

Syntax	src	src2	dest	Code Size (Byte)
(1) SHLL src, dest	#IMM:5	–	Rd	2
(2) SHLL src, dest	Rs	–	Rd	3
(3) SHLL src, src2, dest	#IMM:5	Rs	Rd	3

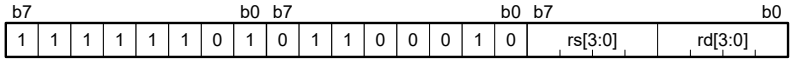
(1) SHLL src, dest



imm[4:0]	src	
00000b to 11111b	#IMM:5	0 to 31

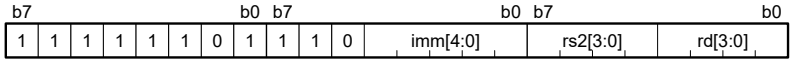
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) SHLL src, dest



rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

(3) SHLL src, src2, dest



imm[4:0]	src	
00000b to 11111b	#IMM:5	0 to 31

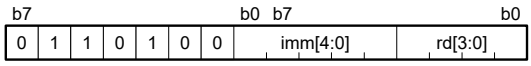
rs2[3:0]/rd[3:0]	src2/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

SHLR SHLR

Code Size

Syntax	src	src2	dest	Code Size (Byte)
(1) SHLR src, dest	#IMM:5	–	Rd	2
(2) SHLR src, dest	Rs	–	Rd	3
(3) SHLR src, src2, dest	#IMM:5	Rs	Rd	3

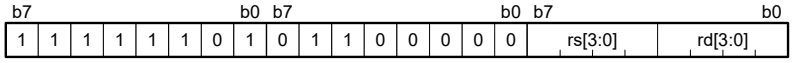
(1) SHLR src, dest



imm[4:0]	src	
00000b to 11111b	#IMM:5	0 to 31

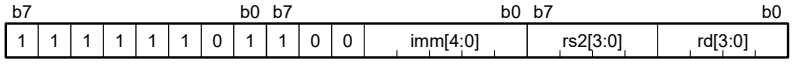
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) SHLR src, dest



rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

(3) SHLR src, src2, dest



imm[4:0]	src	
00000b to 11111b	#IMM:5	0 to 31

rs2[3:0]/rd[3:0]	src2/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

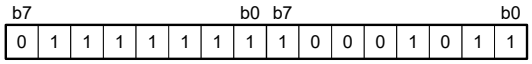
SMOVB

SMOVB

Code Size

Syntax	Code Size (Byte)
(1) SMOVB	2

(1) SMOVB



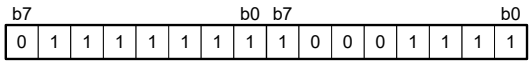
SMOVF

SMOVF

Code Size

Syntax	Code Size (Byte)
(1) SMOVF	2

(1) SMOVF



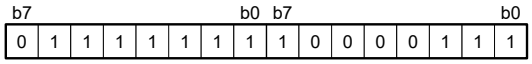
SMOVU

SMOVU

Code Size

Syntax	Code Size (Byte)
(1) SMOVU	2

(1) SMOVU

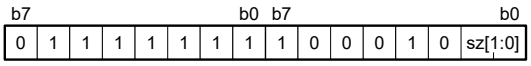


SSTR SSTR

Code Size

Syntax	Size	Processing Size	Code Size (Byte)
(1) SSTR.size	B	B	2
	W	W	2
	L	L	2

(1) SSTR.size



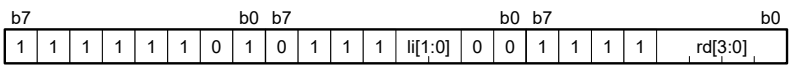
sz[1:0]	Size
00b	B
01b	W
10b	L

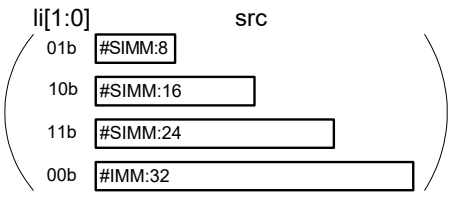
STNZ STNZ

Code Size

Syntax	src	dest	Code Size (Byte)
(1) STNZ src, dest	#SIMM:8	Rd	4
	#SIMM:16	Rd	5
	#SIMM:24	Rd	6
	#IMM:32	Rd	7
(2) STNZ src, dest	Rs	Rd	3

(1) STNZ src, dest

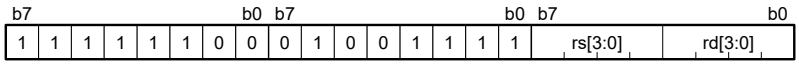




li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) STNZ src, dest



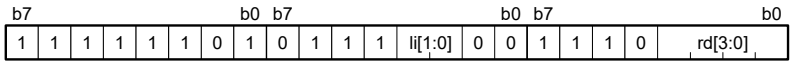
rs[3:0/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

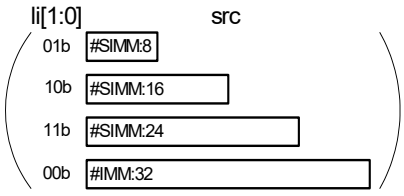
STZ STZ

Code Size

Syntax	src	dest	Code Size (Byte)
(1) STZ src, dest	#SIMM:8	Rd	4
	#SIMM:16	Rd	5
	#SIMM:24	Rd	6
	#IMM:32	Rd	7
(2) STZ src, dest	Rs	Rd	3

(1) STZ src, dest

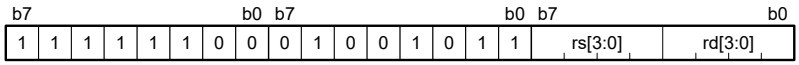




li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) STZ src, dest



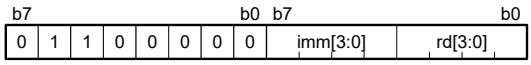
rs[3:0/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

SUB SUB

Code Size

Syntax	src	src2	dest	Code Size (Byte)
(1) SUB src, dest	#UIMM:4	–	Rd	2
(2) SUB src, dest	Rs	–	Rd	2
	[Rs].memex	–	Rd	2 (memex == “UB”) 3 (memex != “UB”)
	dsp:8[Rs].memex	–	Rd	3 (memex == “UB”) 4 (memex != “UB”)
	dsp:16[Rs].memex	–	Rd	4 (memex == “UB”) 5 (memex != “UB”)
(3) SUB src, src2, dest	Rs	Rs2	Rd	3

(1) SUB src, dest

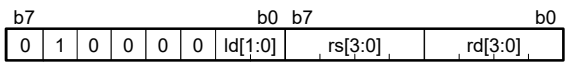


imm[3:0]	src	
0000b to 1111b	#UIMM:4	0 to 15

rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

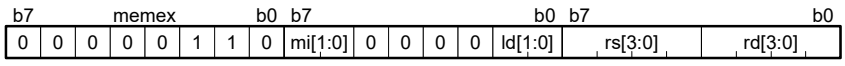
(2) SUB src, dest

When memex == “UB” or src == Rs





When memex != “UB”



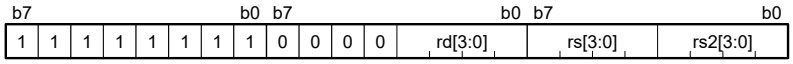


mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

(3) SUB src, src2, dest



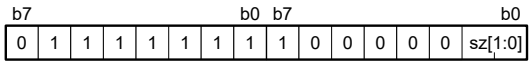
rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b to 1111b	Rs/Rs2/Rd	R0 (SP) to R15

SUNTIL SUNTIL

Code Size

Syntax	Size	Processing Size	Code Size (Byte)
(1) SUNTIL.size	B	B	2
	W	W	2
	L	L	2

(1) SUNTIL.size



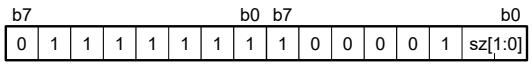
sz[1:0]	Size
00b	B
01b	W
10b	L

SWHILE SWHILE

Code Size

Syntax	Size	Processing Size	Code Size (Byte)
(1) SWHILE.size	B	B	2
	W	W	2
	L	L	2

(1) SWHILE.size



sz[1:0]	Size
00b	B
01b	W
10b	L

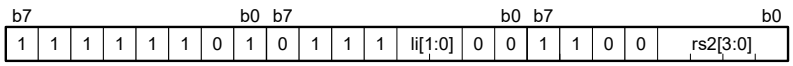
TST

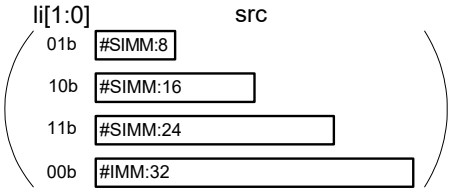
TST

Code Size

Syntax	src	src2	Code Size (Byte)
(1) TST src, src2	#SIMM:8	Rs	4
	#SIMM:16	Rs	5
	#SIMM:24	Rs	6
	#IMM:32	Rs	7
(2) TST src, src2	Rs	Rs2	3
	[Rs].memex	Rs2	3 (memex == “UB”) 4 (memex != “UB”)
	dsp:8[Rs].memex	Rs2	4 (memex == “UB”) 5 (memex != “UB”)
	dsp:16[Rs].memex	Rs2	5 (memex == “UB”) 6 (memex != “UB”)

(1) TST src, src2



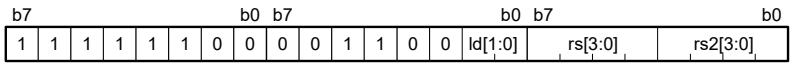


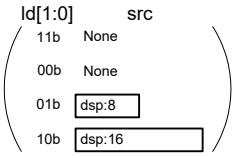
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

rs2[3:0]	src2	
0000b to 1111b	Rs	R0 (SP) to R15

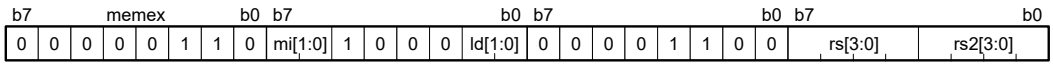
(2) TST src, src2

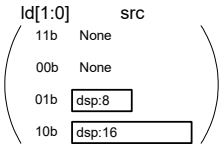
When memex == “UB” or src == Rs





When memex != “UB”





mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rs2[3:0]	src/src2	
0000b to 1111b	Rs/Rs2	R0 (SP) to R15

UTOF

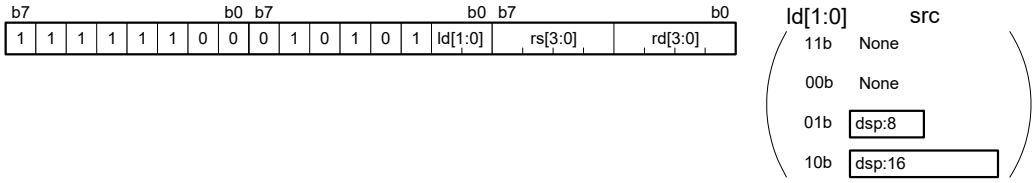
UTOF

Code Size

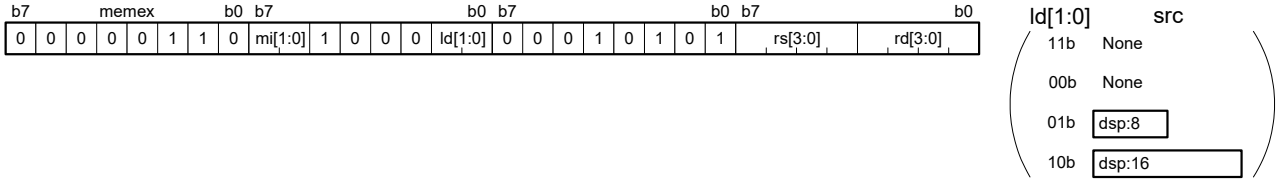
Syntax	src	dest	Code Size (Byte)
(1) UTOF src, dest	Rs	Rd	3
	[Rs].memex	Rd	3 (memex == “UB”) 4 (memex != “UB”)
	dsp:8[Rs].memex	Rd	4 (memex == “UB”) 5 (memex != “UB”)
	dsp:16[Rs].memex	Rd	5 (memex == “UB”) 6 (memex != “UB”)

(1) UTOF src, dest

When memex == “UB” or src == Rs



When memex != “UB”



mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

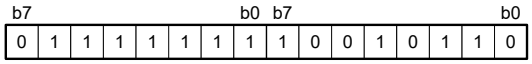
WAIT

WAIT

Code Size

Syntax	Code Size (Byte)
(1) WAIT	2

(1) WAIT



XCHG

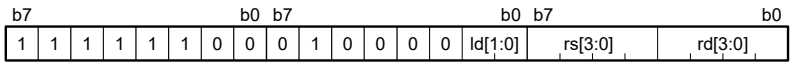
XCHG

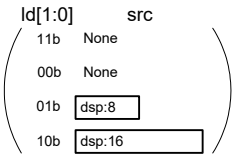
Code Size

Syntax	src	dest	Code Size (Byte)
(1) XCHG src, dest	Rs	Rd	3
	[Rs].memex	Rd	3 (memex == “UB”) 4 (memex != “UB”)
	dsp:8[Rs].memex	Rd	4 (memex == “UB”) 5 (memex != “UB”)
	dsp:16[Rs].memex	Rd	5 (memex == “UB”) 6 (memex != “UB”)

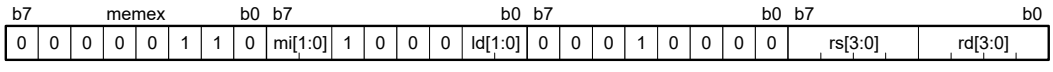
(1) XCHG src, dest

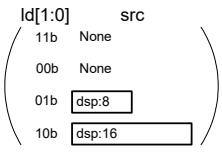
When memex == “UB” or src == Rs





When memex != “UB”





mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

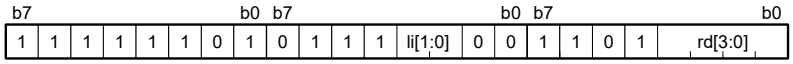
XOR

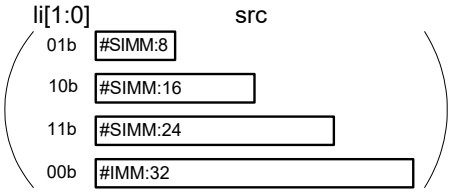
XOR

Code Size

Syntax	src	src2	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”)
(3) XOR src, src2, dest	Rs	Rs2	Rd	3

(1) XOR src, dest



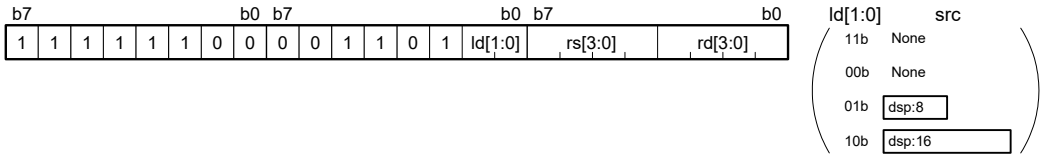


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

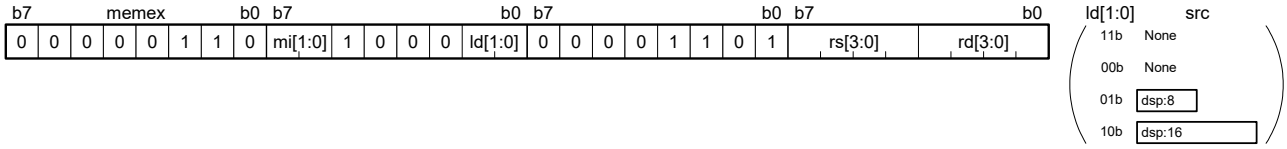
rd[3:0]	dest	
0000b to 1111b	Rd	R0 (SP) to R15

(2) XOR src, dest

When memex == “UB” or src == Rs



When memex != “UB”

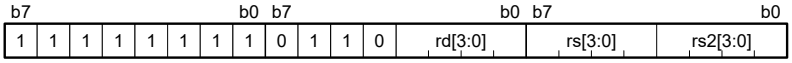


mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	Rs/Rd	R0 (SP) to R15

(3) XOR 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

4.2.2 Instructions for Register Bank Save Function

The following pages give details of the instructions for register bank save function.

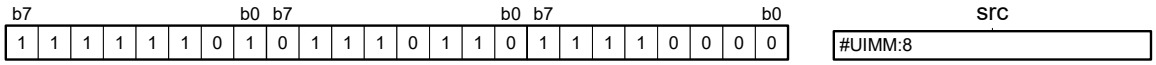
RSTR

RSTR

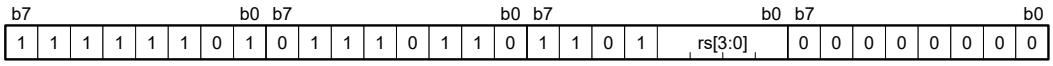
Code Size

Syntax	src	Code Size (Byte)
(1) RSTR src	#UIMM:8	4
(2) RSTR src	Rs	4

(1) RSTR src



(2) RSTR src



rs[3:0]	src	
0000b to 1111b	Rs	R0 (SP) to R15

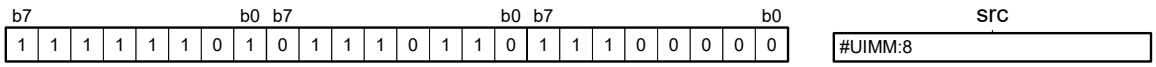
SAVE

SAVE

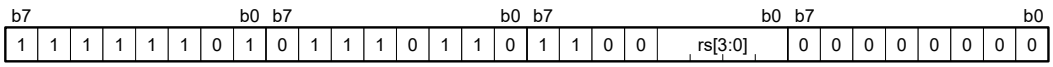
Code Size

Syntax	src	Code Size (Byte)
(1) SAVE src	#UIMM:8	4
(2) SAVE src	Rs	4

(1) SAVE src



(2) SAVE src



rs[3:0]	src	
0000b to 1111b	Rs	R0 (SP) to R15

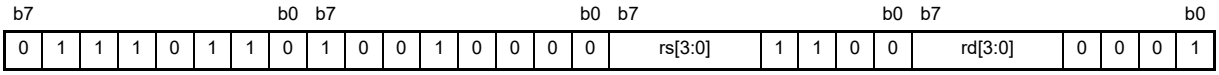
4.2.3 Double-Precision Floating-Point Instructions

The following pages give details of the instruction codes for the double-precision floating-point instructions.

DABS DABS

Code Size

Syntax	src	dest	Code Size (Byte)
DABS src, dest	DRs	DRd	4

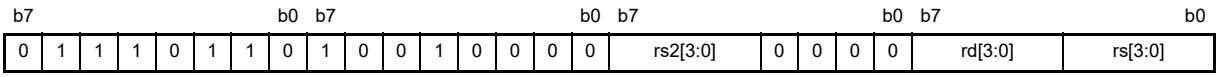


rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	DRs/DRd	DR0 to DR15

DADD DADD

Code Size

Syntax	src	src2	dest	Code Size (Byte)
DADD src, src2, dest	DRs	DRs2	DRd	4

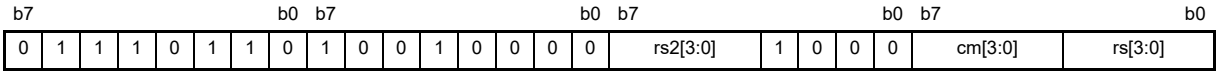


rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b to 1111b	DRs/DRs2/DRd	DR0 to DR15

DCMPcm DCMPcm

Code Size

Syntax		src	src2	Code Size (Byte)
DCMPcm	src, src2	DRs	DRs2	4



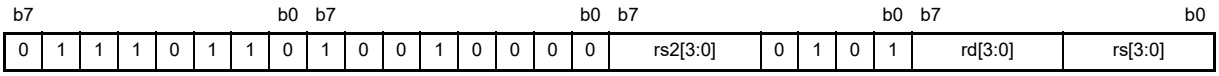
rs[3:0]/rs2[3:0]	src/src2	
0000b to 1111b	DRs/DRs2	DR0 to DR15

cm	Condition
0001b	UN
0010b	EQ
0100b	LT
0110b	LE

DDIV DDIV

Code Size

Syntax	src	src2	dest	Code Size (Byte)
DDIV src, src2, dest	DRs	DRs2	DRd	4



rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b to 1111b	DRs/DRs2/DRd	DR0 to DR15

DMOV DMOV

Code Size

Syntax	Format	Size	Processing Size	Operand src	Operand dest	Code Size (Byte)
DMOV.size src, dest	Data transfer between registers					
	(1)	D	D	Rs	DRHd	4
	(2)	L	L	Rs	DRHd	4
	(3)	L	L	Rs	DRLd	4
	(4)	L	L	DRHs	Rd	4
	(5)	L	L	DRLs	Rd	4
	(6)	D	D	DRs	DRd	4
	Store					
	(7)	D	D	DRs	[Rd]	4
		D	D	DRs	dsp:8[Rd]	5
		D	D	DRs	dsp:16[Rd]	6
	Load					
	(8)	D	D	[Rs]	DRd	4
		D	D	dsp:8[Rs]	DRd	5
		D	D	dsp:16[Rs]	DRd	6
	Set immediate value					
	(9)	D	D	#IMM:32	DRHd	7
	(10)	L	L	#IMM:32	DRHd	7
	(11)	L	L	#IMM:32	DRLd	7

(1) DMOV.size src, dest

b7							b0	b7							b0	b7							b0	b7				b0
1	1	1	1	1	1	0	1	0	1	1	1	0	1	1	1	1	0	0	0	rs[3:0]				rd[3:0]				0	0	1	1

rs[3:0]	src	
0000b to 1111b	Rs	R0(SP) to R15

rd[3:0]	dest	
0000b to 1111b	DRHd	DRH0 to DRH15

(2) DMOV.size src, dest

b7							b0	b7							b0	b7							b0	b7				b0
1	1	1	1	1	1	0	1	0	1	1	1	0	1	1	1	1	0	0	0	rs[3:0]				rd[3:0]				0	0	1	0

rs[3:0]	src	
0000b to 1111b	Rs	R0(SP) to R15

rd[3:0]	dest	
0000b to 1111b	DRHd	DRH0 to DRH15

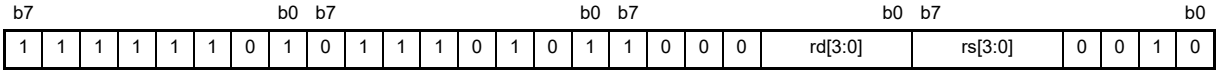
(3) DMOV.size src, dest

b7							b0	b7							b0	b7							b0	b7				b0
1	1	1	1	1	1	0	1	0	1	1	1	0	1	1	1	1	0	0	0	rs[3:0]				rd[3:0]				0	0	0	0

rs[3:0]	src	
0000b to 1111b	Rs	R0(SP) to R15

rd[3:0]	dest	
0000b to 1111b	DRLd	DRL0 to DRL15

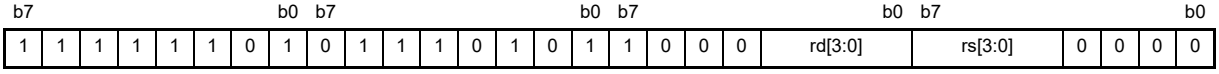
(4) DMOV.size src, dest



rs[3:0]	src	
0000b to 1111b	DRHs	DRH0 to DRH15

rd[3:0]	dest	
0000b to 1111b	Rd	R0(SP) to R15

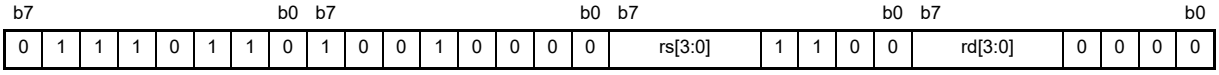
(5) DMOV.size src, dest



rs[3:0]	src	
0000b to 1111b	DRLd	DRL0 to DRL15

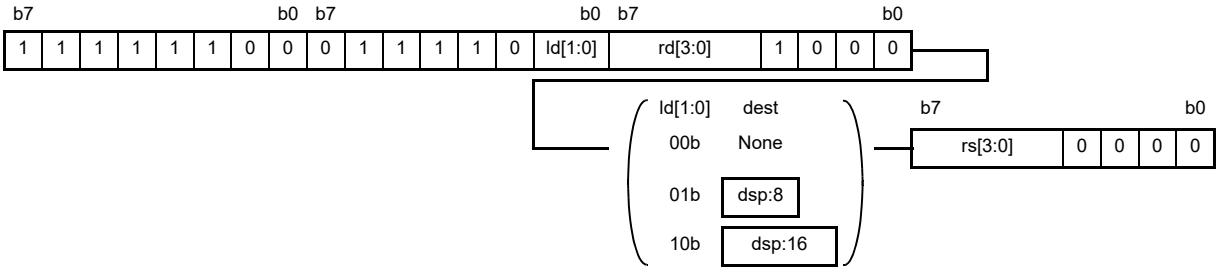
rd[3:0]	dest	
0000b to 1111b	Rd	R0(SP) to R15

(6) DMOV.size src, dest



rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	DRs/DRd	DR0 to DR15

(7) DMOV.size src, dest

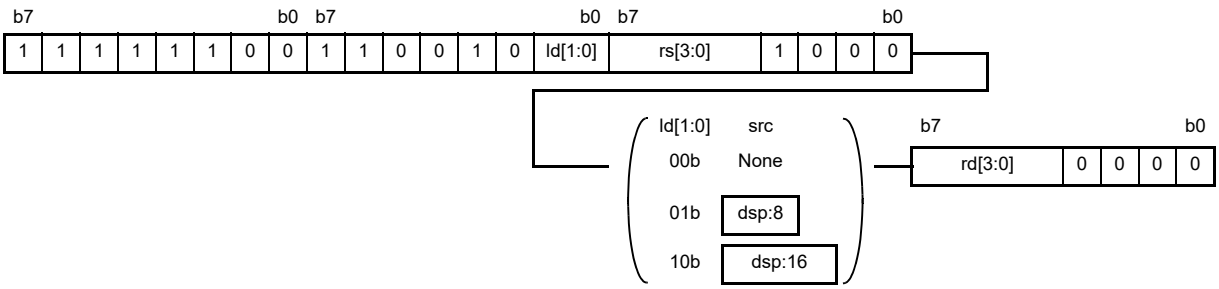


ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

rs[3:0]	src	
0000b to 1111b	DRs	DR0 to DR15

rd[3:0]	dest	
0000b to 1111b	Rd	R0(SP) to R15

(8) DMOV.size src, dest

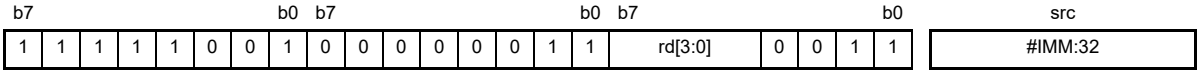


ld[1:0]	src
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]	src	
0000b to 1111b	Rs	R0(SP) to R15

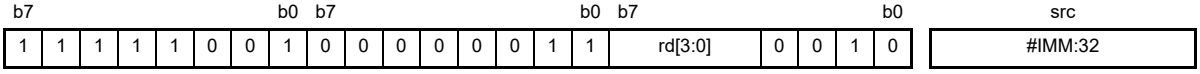
rd[3:0]	dest	
0000b to 1111b	Rd	DR0 to DR15

(9) DMOV.size src, dest



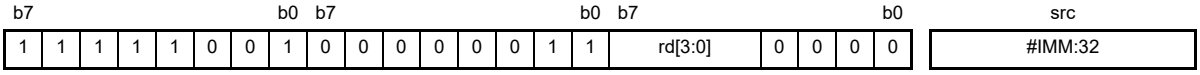
rd[3:0]	dest	
0000b to 1111b	DRHd	DRH0 to DRH15

(10) DMOV.size src, dest



rd[3:0]	dest	
0000b to 1111b	DRHd	DRH0 to DRH15

(11) DMOV.size src, dest



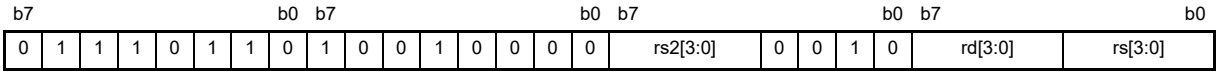
rd[3:0]	dest	
0000b to 1111b	DRLd	DRL0 to DRL15

DMUL

DMUL

Code Size

Syntax	src	src2	dest	Code Size (Byte)
DMUL src, src2, dest	DRs	DRs2	DRd	4



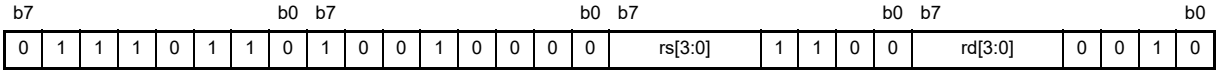
rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b to 1111b	DRs/DRs2/DRd	DR0 to DR15

DNEG

DNEG

Code Size

Syntax	dest	dest	Code Size (Byte)
DNEG src, dest	DRd	DRd	4



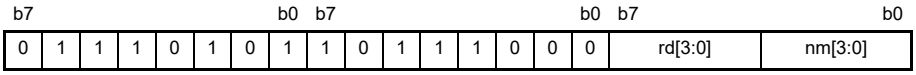
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	DRs/DRd	DR0 to DR15

DPOPM DPOPM

Code Size

Syntax	Format	Processing Size	Operand		Code Size (Byte)
			dest	dest2	
DPOPM.size dest-dest2	(1)	D	DRd	DRd2	3
	(2)	L	DCRd	DCRd2	3

(1) DPOPM.size dest-dest2

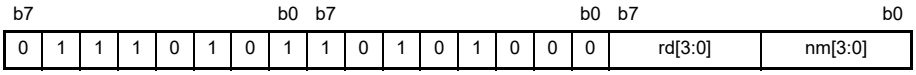


rs[3:0]	dest	
0000b to 1111b	DRd	DR0 to DR15

nm[3:0]	num(dest2 – dest)
0000b to 1111b	d2 – d*

Note: * d2 – d indicates the number of registers between DRd and DRd2, including DRd and DRd2, minus 1.

(2) DPOPM.size dest-dest2



rs[3:0]	dest	
0000b	DCRd	DCR0/DPSW
0001b		DCR1/DCMR
0010b		DCR2/DECNT
0011b		DCR3/DEPC

nm[3:0]	num(dest2 – dest)
0000b to 0011b	d2 – d*

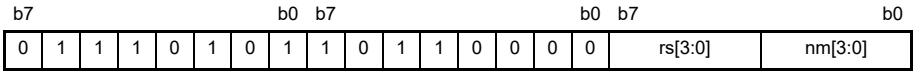
Note: * d2 – d indicates the number of registers between DCRd and DCRd2, including DCRd and DCRd2, minus 1.

DPUSHM DPUSHM

Code Size

Syntax	Format	Processing Size	Operand		Code Size (Byte)
			dest	dest2	
DPUSHM.size src-src2	(1)	D	DRs	DRs2	3
	(2)	L	DCRs	DCRs2	3

(1) DPUSHM.size src-src2

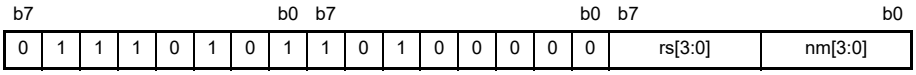


rs[3:0]	src	
0000b to 1111b	DRs	DR0 to DR15

nm[3:0]	num(src2 – src)
0000b to 1111b	s2 – s*

Note: * s2 – s indicates the number of registers between DRs and DRs2, including DRs and DRs2, minus 1.

(2) DPUSHM.size src-src2



rs[3:0]	src	
0000b	DCRs	DCR0/DPSW
0001b		DCR1/DCMR
0010b		DCR2/DECNT
0011b		DCR3/DEPC

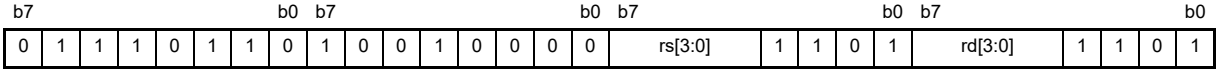
nm[3:0]	num(src2 – src)
0000b to 0011b	s2 – s*

Note: * s2 – s indicates the number of registers between DCRs and DCRs2, including DCRs and DCRs2, minus 1.

DROUND DROUND

Code Size

Syntax	src	dest	Code Size (Byte)
DROUND src, dest	DRd	DRd	4

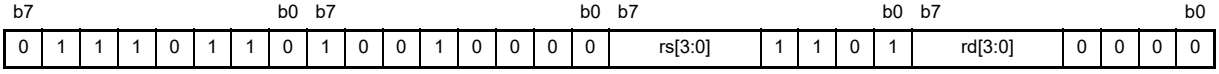


rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	DRs/DRd	DR0 to DR15

DSQRT DSQRT

Code Size

Syntax	src	dest	Code Size (Byte)
DSQRT src, dest	DRs	DRd	4

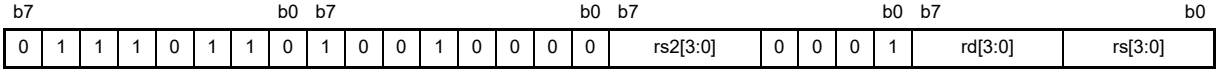


rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	DRs/DRd	DR0 to DR15

DSUB DSUB

Code Size

Syntax	src	src2	dest	Code Size (Byte)
DSUB src, src2, dest	DRs	DRs2	DRd	4

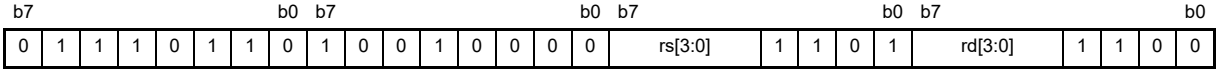


rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b to 1111b	DRs/DRs2/DRd	DR0 to DR15

DTOF DTOF

Code Size

Syntax		src	dest	Code Size (Byte)
DTOF	src, dest	DRs	DRd	4

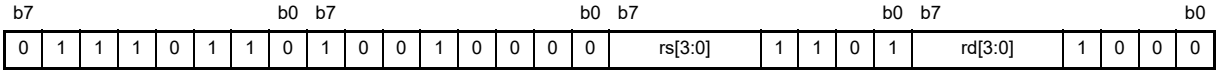


rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	DRs/DRd	DR0 to DR15

DTOI DTOI

Code Size

Syntax		src	dest	Code Size (Byte)
DTOI	src, dest	DRs	DRd	4

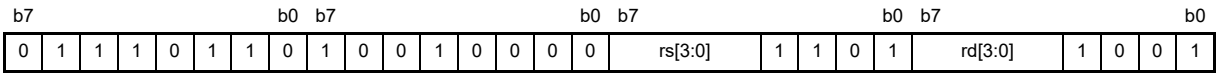


rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	DRs/DRd	DR0 to DR15

DTOU DTOU

Code Size

Syntax		src	dest	Code Size (Byte)
DTOU	src, dest	DRs	DRd	4



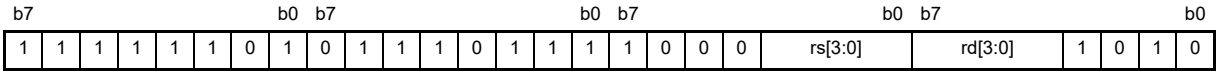
rs[3:0]/rd[3:0]	src/dest	
0000b to 1111b	DRs/DRd	DR0 to DR15

FTOD

FTOD

Code Size

Syntax	src	dest	Code Size (Byte)
FTOD src, dest	Rs	DRd	4



rs[3:0]	src	
0000b to 1111b	Rs	R0(SP) to R15

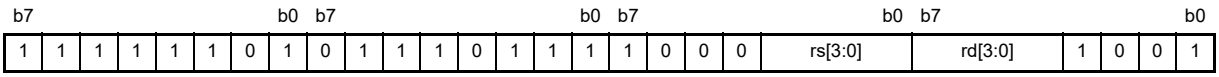
rd[3:0]	dest	
0000b to 1111b	DRd	DR0 to DR15

ITOD

ITOD

Code Size

Syntax	src	dest	Code Size (Byte)
ITOD src, dest	Rs	DRd	4



rs[3:0]	src	
0000b to 1111b	Rs	R0(SP) to R15

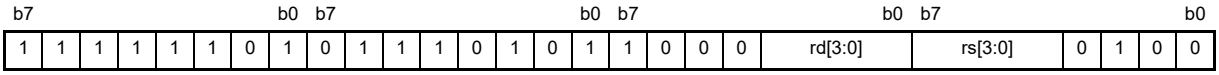
rd[3:0]	dest	
0000b to 1111b	DRd	DR0 to DR15

MVFDC

MVFDC

Code Size

Syntax		src	dest	Code Size (Byte)
MVFDC	src, dest	DCRs	Rd	4



rs[3:0]	src	
0000b	DCRs	DCR0/DPSW
0001b		DCR1/DCMR
0010b		DCR2/DECNT
0011b		DCR3/DEPC

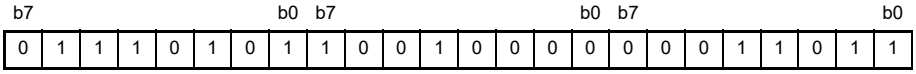
rd[3:0]	dest	
0000b to 1111b	Rd	R0(SP) to R15

MVFDR

MVFDR

Code Size

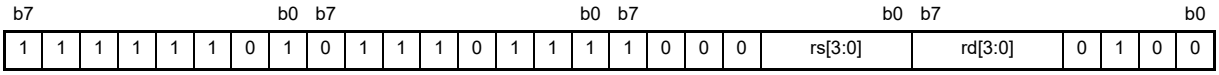
Syntax	Code Size (Byte)
MVFDR	3



MVTDC MVTDC

Code Size

Syntax		src	dest	Code Size (Byte)
MVTDC	src, dest	Rs	DCRd	4



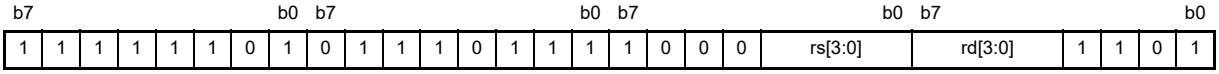
rs[3:0]	src	
0000b to 1111b	Rs	R0(SP) to R15

rd[3:0]	dest	
0000b	DCRd	DCR0/DPSW
0001b		DCR1/DCMR
0010b		DCR2/DECNT
0011b		DCR3/DEPC

UTOD UTOD

Code Size

Syntax		src	dest	Code Size (Byte)
UTOD	src, dest	Rs	DRd	4



rs[3:0]	src	
0000b to 1111b	Rs	R0(SP) to R15

rd[3:0]	dest	
0000b to 1111b	DRd	DR0 to DR15

4.2.4 Protect Zone Instructions

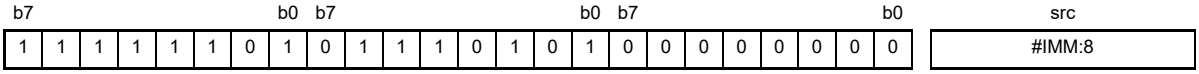
The following pages give details of the instruction codes of the Protect Zone instructions.

TRCALL

TRCALL

Code Size

Syntax	Operand src	Code Size (Byte)
TRCALL src	#IMM:8	4



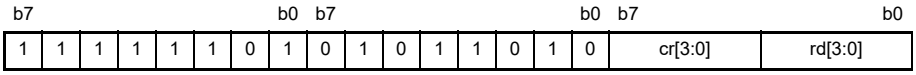
TRMVFC

TRMVFC

Code Size

Syntax	Operand src	Operand dest	Code Size (Byte)
TRMVFC src, dest	Rx	Rd	3

(1) TRMVFC src, dest



cr[3:0]	src	
0000b	Rx	Reserved
0001b		Reserved
0010b		Reserved
0011b		Reserved
0100b		Reserved
0101b		IVC
0110b		Reserved
0111b		Reserved
1000b		BPSW
1001b		BPC
1010b		ISP
1011b		Reserved
1100b		INTB
1101b		EXTB
1110b		TRCTB
1111b		WINTB

rd[3:0]	dest	
0000b to 1111b	Rd	R0(SP) to R15

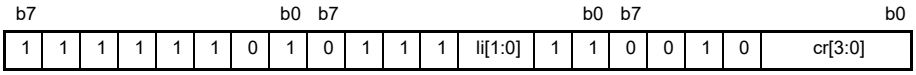
TRMVTC

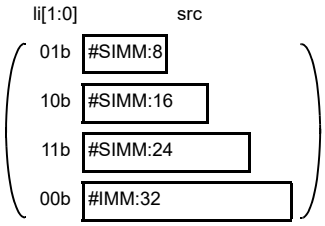
TRMVTC

Code Size

Syntax	Operand		Code Size (Byte)
	src	dest	
(1) TRMVTC src, dest	#SIMM:8	Rx	4
	#SIMM:16	Rx	5
	#SIMM:24	Rx	6
	#IMM:32	Rx	7
(2) TRMVTC src, dest	Rs	Rx	3

(1) TRMVTC src, dest

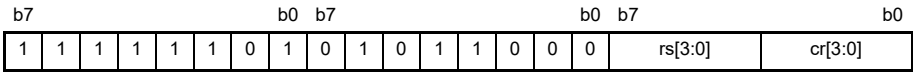




li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

cr[3:0]	dest	
0000b	Rx	Reserved
0001b		Reserved
0010b		Reserved
0011b		Reserved
0100b		Reserved
0101b		IVC
0110b		Reserved
0111b		Reserved
1000b		BPSW
1001b		BPC
1010b		ISP
1011b		Reserved
1100b		INTB
1101b		EXTB
1110b		TRCTB
1111b		WINTB

(2) TRMVTC src, dest



rs[3:0]	src	
0000b to 1111b	Rs	R0(SP) to R15

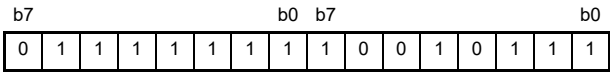
cr[3:0]	dest	
0000b	Rx	Reserved
0001b		Reserved
0010b		Reserved
0011b		Reserved
0100b		Reserved
0101b		IVC
0110b		Reserved
0111b		Reserved
1000b		BPSW
1001b		BPC
1010b		ISP
1011b		Reserved
1100b		INTB
1101b		EXTB
1110b		TRCTB
1111b		WINTB

TRRTE

TRRTE

Code Size

Syntax	Code Size (Byte)
TRRTE	2



5. Exceptions

5.1 Types of Exception

During the execution of a program by the CPU, the occurrence of certain events may necessitate suspending execution of the main flow of the program and starting the execution of another flow. Such events are called exceptions.

Figure 5.1 shows the types of exception.

The occurrence of an exception causes the processor mode to switch to supervisor mode.

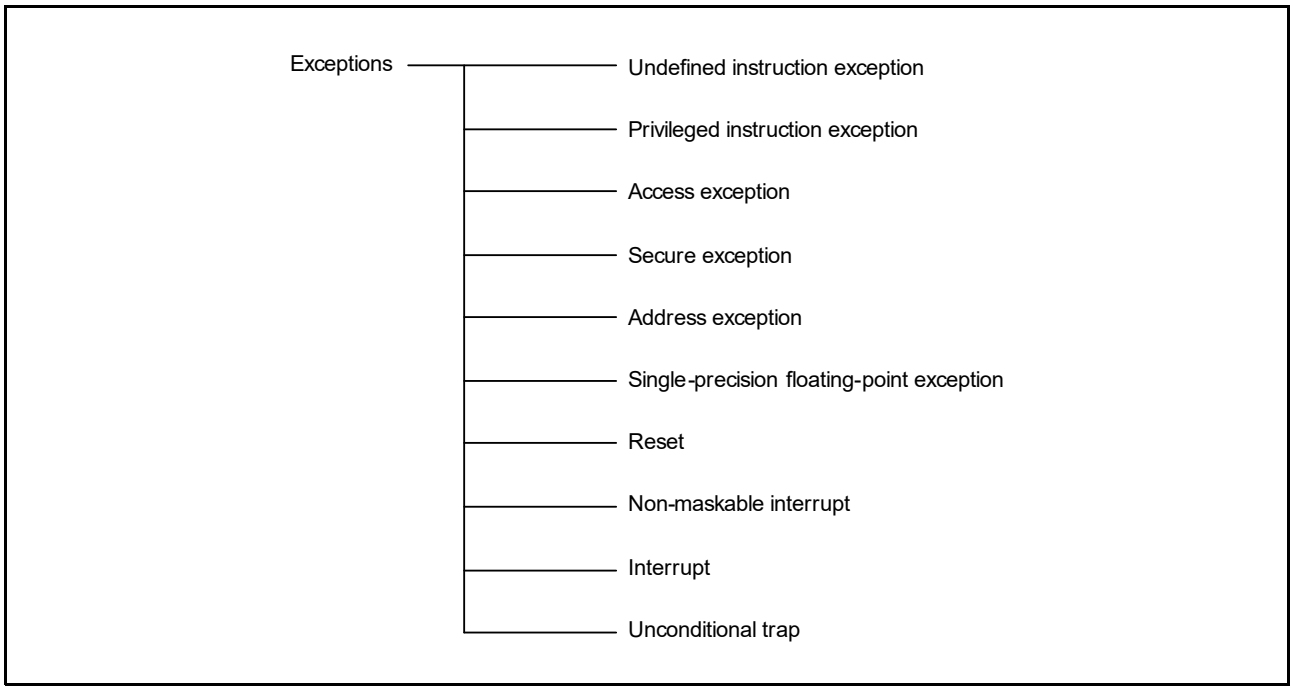


Figure 5.1 Types of Exception

5.1.1 Undefined Instruction Exception

An undefined instruction exception occurs when execution of an undefined instruction (an instruction not implemented) is detected. This exception also occurs in the case of an attempt to execute an instruction that is exclusively for use in the root world from outside the root world.

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

An access exception occurs when the memory protection unit detects an error in memory access by the CPU. If the error is due to instruction fetching, an instruction access exception occurs, and if the error is due to operand access, an operand access exception occurs.

5.1.4 Secure Exception

A secure exception occurs when a security violation is detected in memory access by the CPU. If an error occurs due to an instruction fetch, an instruction secure exception occurs, and if an error occurs due to an operand access, an operand secure exception occurs.

The CPU core is placed in the root world on generation of a secure exception.

Even when a security violation is detected in memory access, the CPU may not generate a secure exception. This depends on the location and timing of the detection.

5.1.5 Address Exceptions

Address exceptions are generated in response to 64-bit operand access to addresses that are not multiples of four.

5.1.6 Single-Precision Floating-Point Exceptions

Single-precision floating-point exceptions are generated when any of the five exceptions specified in the IEEE754 standard, namely overflow, underflow, inexact, division-by-zero, or invalid operation, or an attempts to use processing that is not implemented, is detected upon execution of a single-precision floating-point arithmetic instruction. Exception handling by the CPU only proceeds when any among the EX, EU, EZ, EO, or EV bits in the FPSW, which corresponding to the five types of exception, is set to 1.

5.1.7 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.

5.1.8 Non-Maskable Interrupt

Input of the non-maskable interrupt signal to the CPU generates a non-maskable interrupt. This interrupt is assumed to be used when a fatal problem with the system is judged to have arisen. In such cases, use it under the condition that return to the original program being executed when the exception was generated will not proceed after the exception handling routine. In addition, unless a Flat-Project is in use, returning to the original program being executed when the exception was generated is not possible.

Non-maskable interrupts are accepted independently of the type of the world that is currently in use.

The behavior upon acceptance of a non-maskable interrupt is divided into the following two cases according to the situation at the time of acceptance.

	Situation on Acceptance of a Non-Maskable Interrupt	World Transition
Normal acceptance	The ID of the destination world for a non-maskable interrupt is the same as the ID of the current world.	No world transition
World-transition acceptance	The ID of the destination world for a non-maskable interrupt is different from the ID of the current world.	Transition to the root world occurs

In the case of normal acceptance, a transition from the current world to another world does not occur.

In the case of world-transition acceptance, a transition to the root world occurs. After that, a transition to the destination world of the non-maskable interrupt is produced by software running in the root world.

5.1.9 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.

An interrupt can only be accepted when the conditions in the table below are met. The conditions for acceptance depend on the current world.

Table 5.1 Conditions of Interrupt Acceptance

Current World		
Root	**Secure**	**Non-secure**
• ITW is root and • PSW.I is 1 and • (PSW.IPL < IPR) and • (WPRI.WIPL < IPR)	• PSW.I is 1 and • (PSW.IPL < IPR) and • (WPRI.WIPL < IPR)	• PSW.I is 1 and • (PSW.IPL < IPR) and • (WPRI.WIPL < IPR) However, when the IVC.NSIC bit is 1, an interrupt will even be accepted under the following condition in addition to the above conditions. • (15 < IPR) and • (WPRI.WIPL < IPR)

IPR: The priority level of a requested interrupt

ITW: The destination world ID for a requested interrupt

The behavior upon acceptance of an interrupt is divided into the following two cases according to the situation at the time of acceptance.

	Situation on Acceptance of Interrupt	World Transition
Normal acceptance	The ID of the destination world for an interrupt is the same as the ID of the current world.	No world transition
World-transition acceptance	The ID of the destination world for an interrupt is different from the ID of the current world.	Transition to the root world occurs

In the case of normal acceptance, a transition from the current world to another world does not occur.

In the case of world-transition acceptance, a transition to the root world occurs. After that, a transition to the destination world of the interrupt is produced by software running in the root world.

Interrupt priority levels 1 to 31 can be used for interrupts having root or secure as the destination world. Interrupt priority levels 1 to 15 can be used for interrupts having non-secure as the destination world.

Fast interrupts are only usable in Flat-Projects.

5.1.10 Unconditional Trap

An unconditional trap is generated when the INT, BRK, or TRCALL instruction is executed.

The behavior upon acceptance depends on the instruction and is divided into the following two cases.

	Instructions	World Transition
Normal acceptance	INT, BRK	No world transition
World-transition acceptance	TRCALL	Transition to the root world occurs

5.2 Exception Handling Procedure

Exception handling procedures are of two types. In one, the exception event is processed without world transitions. In the other, the exception event is processed with world transitions.

Table 5.2 shows which exception events generate world transitions.

Table 5.2 Exception Events and World Transition

Exception Event	Presence of World Transitions
Undefined instruction exception	No world transitions
Privileged instruction exception	No world transitions
Access exception	No world transitions
Secure exception	World transitions
Address exception	No world transitions
Single-precision floating-point exception	No world transitions
Reset	— (always causes a transition to the root world)
Non-maskable interrupt (with normal acceptance)	No world transitions
Non-maskable interrupt (with world-transition acceptance)	World transitions
Interrupt (with normal acceptance)	No world transitions
Interrupt (with world-transition acceptance)	World transitions
Unconditional trap (with normal acceptance)	No world transitions
Unconditional trap (with world-transition acceptance)	World transitions

5.2.1 Exception Handling Procedure without World Transitions

For exception handling, part of the processing is handled automatically by hardware and part is handled by a program (the exception handling routine) that has been written by the user. Figure 5.2 shows the handling procedure when an exception other than a reset is accepted.

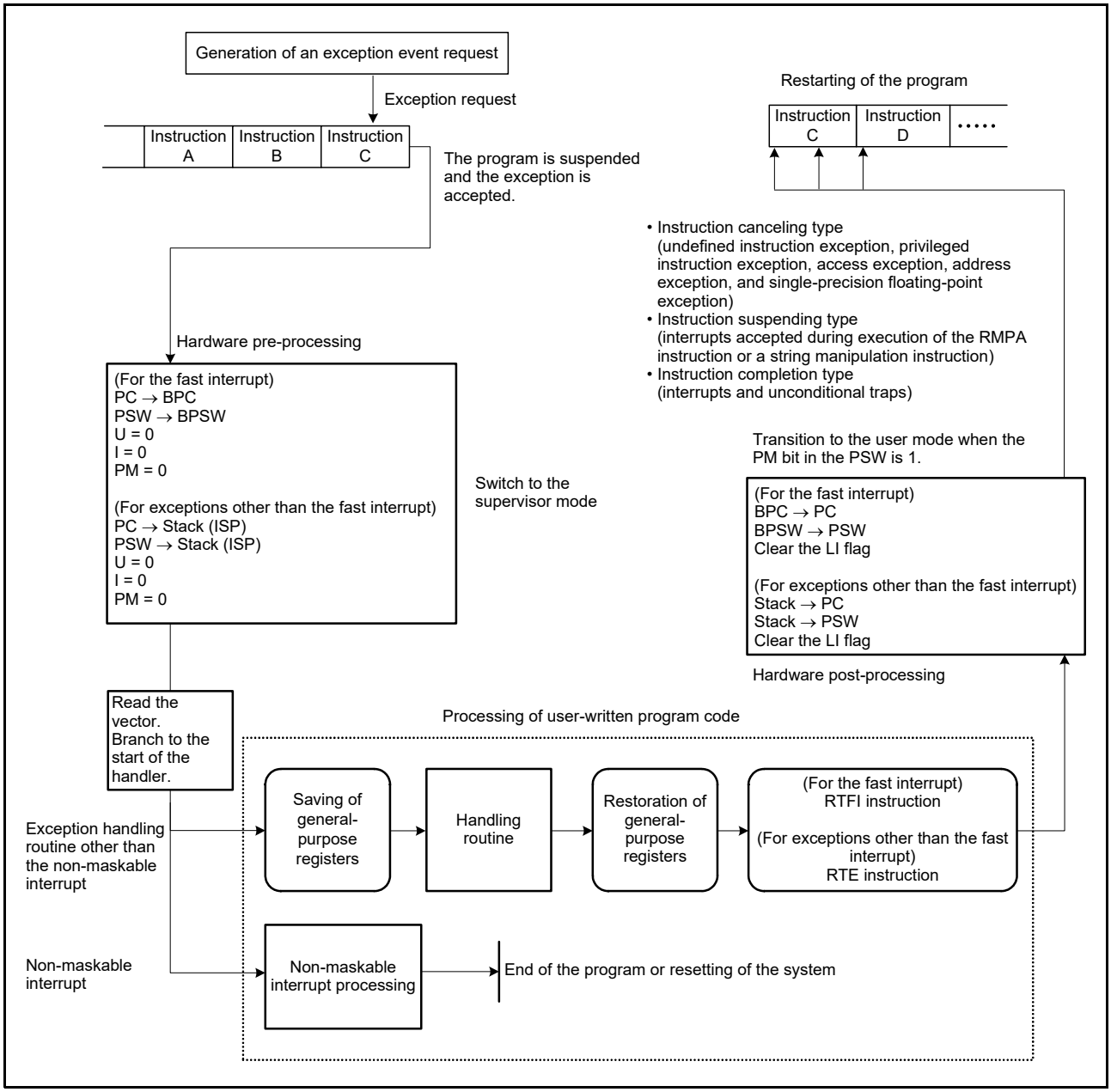


Figure 5.2 Outline of the Exception Handling Procedure without World Transitions

When an exception is accepted, hardware processing by the CPU is followed by vector table access to acquire the address of the branch destination. A vector address is allocated to each exception. The branch destination address of the exception handling routine for the given exception is written to each vector address.

Hardware pre-processing by the CPU handles saving of the contents of the program counter (PC) and processor status word (PSW). In the case of the fast interrupt, the contents are saved in the backup PC (BPC) and the backup PSW (BPSW), respectively. In the case of other exceptions, the contents are saved on the stack. General purpose registers and control registers other than the PC and PSW that are to be used within the exception handling routine must be saved by user program code at the start of the exception handling routine.

At the end of exception handling routine, after the restoration of registers saved by the user, the RTE instruction is executed to return from the exception handling routine to the original program. For return from the fast interrupt, the RTFI instruction is used instead. In the case of the non-maskable interrupt, end the program or reset the system without returning to the original program.

Hardware post-processing by the CPU handles restoration of the pre-exception contents of the PC and PSW. In the case of the fast interrupt, the contents of the BPC and BPSW are restored to the PC and PSW, respectively. In the case of other exceptions, the contents are restored from the stack to the PC and PSW.

The stack or the register-saving bank can be used to save and restore the general-purpose and other registers at the start and end of an exception handling routine.

(1) Saving to and restoring from the register-saving bank is executed by using the SAVE and RSTR instructions if a corresponding product incorporates register-saving banks.

(2) When a corresponding product does not incorporate register-saving banks or when a product with register-saving banks saves or restores a register that is not intended for saving or restoring with the SAVE or RSTR instruction, use the PUSH or POP instruction for saving to and restoring from the stack.

Using the register-saving bank is usually faster than using the stack, except when the number of registers that require saving and restoring in transitions to and from an exception-handling routine is extremely small.

5.2.2 Exception Handling Procedure with World Transitions

An exception handling procedure has a part that is automatically processed by hardware and parts that are processed by user-created programs (processing programs to be executed in the root world, processing programs to be executed in the destination world of the exception event, and exception handling routines). Figure 5.3 shows an exception handling procedure that includes world transitions.

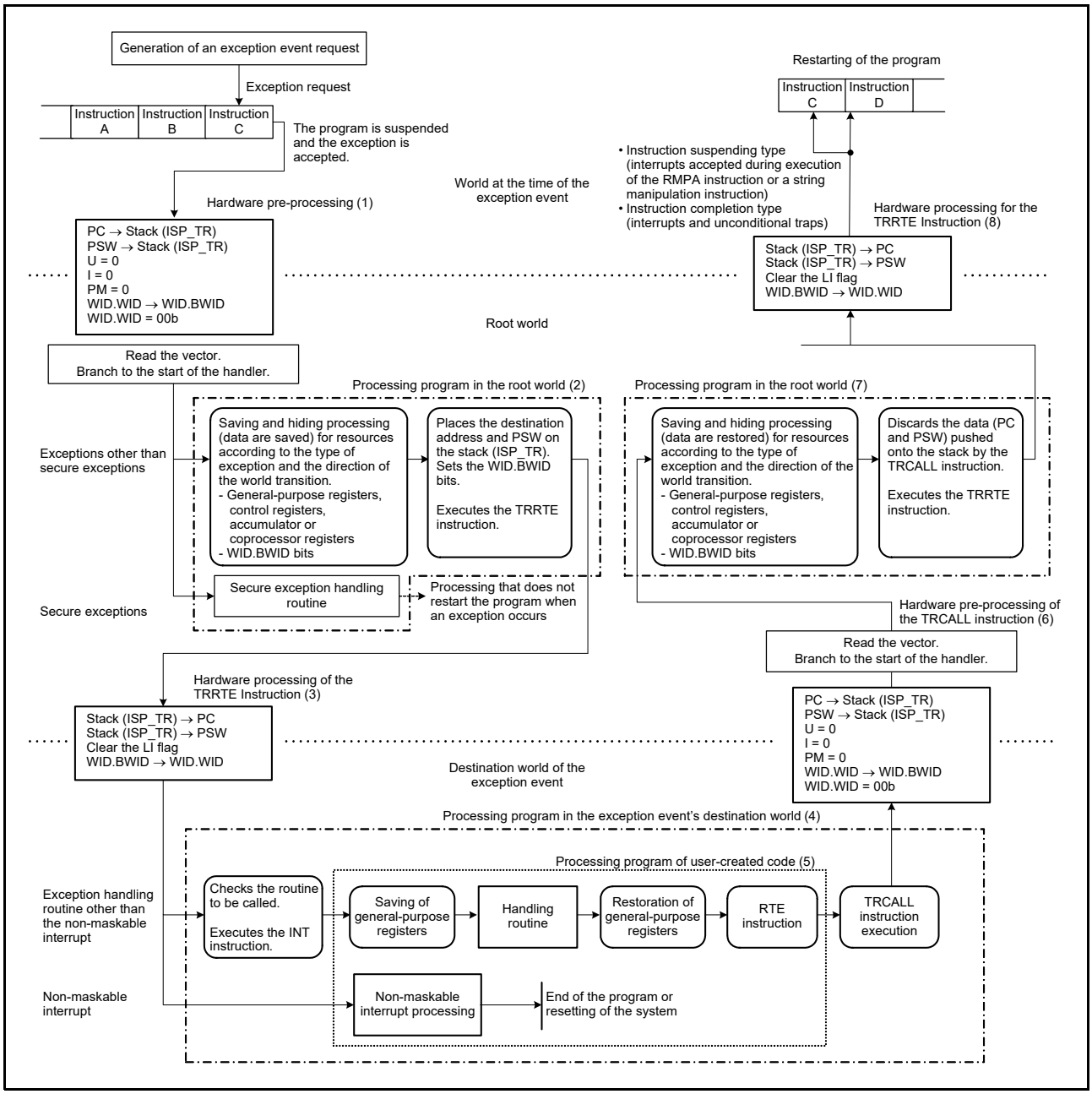


Figure 5.3 Overview of an Exception Handling Procedure with World Transitions

When an exception that includes world transitions is accepted, the CPU performs hardware pre-processing (1) and access to the vector table, then obtains the branch destination address. A vector address is assigned to the vector for each exception, and the branch destination address to the processing program in the root world (2) is written to that address.

In hardware preprocessing by the CPU (1), the values of the PC and PSW are placed on the stack (ISP_TR). Also, the value of the WID.WID bits is saved in the WID.BWID bits, and a transition is made to the root world with the WID.WID bits set to 0.

In the processing program in the root world (2), the flow of processing depends on whether the exception is secure (in other words, processing of the original program will not resume on return) or not (in other words, processing of the original program in the original world resumes after the exception handling routine in another world has been run via the root world). In the case of a secure exception, after appropriate processing by the secure exception handling routine in the root world, the program is terminated or a jump is made to processing that does not involve resuming processing by the program at the time the exception occurred. In the case of an exception other than a secure exception, consider the type of the exception and the direction of world transitions due to that exception, and perform processing to save or hide resources required by the system (general-purpose registers, control registers, and accumulator or coprocessor registers). After that, set the WID.BWID bits to match the destination world, place the start address and PSW of the processing program in the exception event's destination world (4) on the stack (ISP_TR), and execute the TRRTE instruction.

The hardware processing of the TRRTE instruction (3) causes a transition to the processing program in the exception event's destination world (4).

With the processing program in the exception event's destination world (4), if the interrupt is not non-maskable, the INT instruction is used to cause a transition to processing of the user-created code (5). In processing an interrupt, the value of the IVC register is used as an argument of the INT instruction. Programming in this manner in the destination world of the exception event enables (5) to be used in common with cases that involve no world transitions (this method is shown here as an example of a processing procedure, but can also be used to make a common program except for the processing to save and restore the general-purpose registers). After that, execute the TRCALL instruction to execute the processing program in the root world (7).

In the hardware pre-processing of the TRCALL instruction (6), the values of the PC and PSW are placed on the stack (ISP_TR). Also, the value of the WID.WID bits is saved in the WID.BWID bits, and a transition is made to the root world with the WID.WID bits set to 0.

The processing program in the root world (7), after restoring the resources saved in (2), including the value of the WID.BWID bits, and discarding the PC and PSW that were pushed onto the stack (ISP_TR) by the immediately preceding TRCALL instruction, executes the TRRTE instruction.

The hardware processing of the TRRTE instruction (8) causes a transition to the location for resuming processing by the program at the point where the exception was accepted.

The flow of the processing procedure is explained above, but the processing program in the root world ((2) or (7)), should also run checking according to the current security requirements, such as whether the world from which the TRCALL instruction was called is appropriate, or whether there are any inconsistencies in the number of world transitions going back and forth.

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 5.3 lists the timing of acceptance and program counter (PC) value to be saved for each type of exception event.

For interrupts, non-maskable interrupts, and unconditional traps, the behavior shown in the table below will not change regardless of whether the exception is accepted through world-transition acceptance or normal acceptance.

Table 5.3 Timing of Acceptance and Saved PC Value

Exception		Type of Handling	Timing of Acceptance	Value Saved in the BPC/ on the Stack
Undefined instruction exception		Instruction canceling type	During instruction execution	PC value of the instruction that generated the exception
Privileged instruction exception		Instruction canceling type	During instruction execution	PC value of the instruction that generated the exception
Access exception		Instruction canceling type	During instruction execution	PC value of the instruction that generated the exception
Secure exception		Instruction canceling type	During instruction execution	PC value of the instruction that generated the exception
Address exception		Instruction canceling type	During instruction execution	PC value of the instruction that generated the exception
Single-precision floating-point exceptions		Instruction canceling type	During instruction execution	PC value of the instruction that generated the exception
Reset		Program abandonment type	Any machine cycle	None
Non-maskable interrupt	During execution of the RMPA, SCMPU, SMOVB, SMOVF, SMOVU, SSTR, SUNTIL, and SWHILE instructions	Instruction suspending type	During instruction execution	PC value of the instruction being executed
	Other than the above	Instruction completion type	At the next break between instructions	PC value of the next instruction
Interrupts	During execution of the RMPA, SCMPU, SMOVB, SMOVF, SMOVU, SSTR, SUNTIL, and SWHILE instructions	Instruction suspending type	During instruction execution	PC value of the instruction being executed
	Other than the above	Instruction completion type	At the next break between instructions	PC value of the next instruction
Unconditional trap		Instruction completion type	At the next break between instructions	PC value of the next instruction

5.3.2 Vector and Location for Saving the PC and PSW

The vector for each type of exception and the location for saving the contents of the program counter (PC) and processor status word (PSW) are listed in Table 5.4 and Table 5.5.

Table 5.4 Vector and Location for Saving the PC and PSW for an Exception without World Transitions

Exception		Vector	Location for Saving the PC and PSW
Undefined instruction exception		Exception vector table in the current world	Stack in the current world
Privileged instruction exception		Exception vector table in the current world	Stack in the current world
Access exception		Exception vector table in the current world	Stack in the current world
Address exception		Exception vector table in the current world	Stack in the current world
Single-precision floating-point exceptions		Exception vector table in the current world	Stack in the current world
Reset		FFFF FFFCh	Nowhere
Non-maskable interrupt		Exception vector table in the current world	Stack in the current world
Interrupts	Fast interrupt	FINTV	BPC and BPSW in the root world
	Other than the above	Interrupt vector table in the current world	Stack in the current world
Unconditional trap		Interrupt vector table in the current world	Stack in the current world

Table 5.5 Vector and Location for Saving the PC and PSW for an Exception with World Transitions

Exception		Vector	Location for Saving the PC and PSW
Non-maskable interrupt		Exception vector table in the root world (EXTB_TR)	Stack in the root world (ISP_TR)
Interrupts	Fast interrupt	—	—
	Other than the above	Vector table for interrupts that cause world transitions (WINTB)	Stack in the root world (ISP_TR)
Secure exception		Exception vector table in the root world (EXTB_TR)	Stack in the root world (ISP_TR)
Unconditional trap		Root call vector table (TRCTB)	Stack in the root world (ISP_TR)

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.

5.4.1 Hardware Processing for Accepting and Returning from Exceptions Not Involving World Transitions

(1) Hardware pre-processing for accepting an exception

(a) Saving the PSW

(For the fast interrupt)

PSW → BPSW

(For other exceptions)

PSW → Stack

Note: The FPSW is not saved by the hardware preprocessing. If single-precision floating-point arithmetic instructions are to be used within an exception handling routine, save the FPSW on the stack from within the exception handling routine.

(b) Updating the PM, U, and I bits in the PSW

I: Cleared to 0

U: Cleared to 0

PM: Cleared to 0

(c) Saving the PC

(For the fast interrupt)

PC → BPC

(For other exceptions)

PC → Stack

(d) Setting the branch-destination address of the exception handling routine in the PC

Processing is shifted to the exception handling routine by acquiring the vector corresponding to the exception and branching accordingly.

(2) Hardware post-processing for executing RTE and RTFI instructions

(a) Restoring the PSW

(For the fast interrupt)

BPSW → PSW

(For other exceptions)

Stack → PSW

(b) Restoring the PC

(For the fast interrupt)

BPC → PC

(For other exceptions)

Stack → PC

(c) Clearing the LI flag

5.4.2 Hardware Processing for Accepting and Returning from Exceptions Involving World Transitions

(1) Hardware pre-processing for accepting an exception

(a) Saving the WID

WID.WID → WID.BWID

(b) Updating the WID

WID.WID: Cleared to 00b

(c) Saving the PSW

PSW → Stack (ISP_TR)

(d) Updating the PM, U, and I bits in the PSW

I: Cleared to 0

U: Cleared to 0

PM: Cleared to 0

(e) Saving the PC

PC → Stack (ISP_TR)

(f) Setting 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 TRRTE instruction

(a) Restoring the WID

WID.BWID → WID.WID

(b) Restoring the PSW

Stack (ISP_TR) → PSW

(c) Restoring the PC

Stack (ISP_TR) → PC

(d) Clearing the LI flag

5.5 Hardware Pre-processing

The sequences of hardware pre-processing from reception of each exception request to execution of the associated exception handling routine are explained below.

5.5.1 Undefined Instruction Exception

(1) The value of the processor status word (PSW) is saved on the stack (ISP).
(2) The processor mode select bit (PM), the stack pointer select bit (U), and the interrupt enable bit (I) in the PSW are cleared to 0.
(3) The value of the program counter (PC) is saved on the stack (ISP).
(4) The vector is fetched from the value of EXTB + address 0000 005Ch.
(5) The PC is set to the fetched address and processing branches to the start of the exception handling routine.

5.5.2 Privileged Instruction Exception

(1) The value of the processor status word (PSW) is saved on the stack (ISP).
(2) The processor mode select bit (PM), the stack pointer select bit (U), and the interrupt enable bit (I) in the PSW are cleared to 0.
(3) The value of the program counter (PC) is saved on the stack (ISP).
(4) The vector is fetched from the value of EXTB + address 0000 0050h.
(5) The PC is set to the fetched address and processing branches to the start of the exception handling routine.

5.5.3 Access Exception

(1) The value of the processor status word (PSW) is saved on the stack (ISP).
(2) The processor mode select bit (PM), the stack pointer select bit (U), and the interrupt enable bit (I) in the PSW are cleared to 0.
(3) The value of the program counter (PC) is saved on the stack (ISP).
(4) The vector is fetched from the value of EXTB + address 0000 0054h.
(5) The PC is set to the fetched address and processing branches to the start of the exception handling routine.

5.5.4 Secure Exception

(1) The value of the WID.WID bits is saved to the WID.BWID bits.
(2) The WID.WID bits are set to 0 and a transition is made to the root world.
(3) The value of the processor status word (PSW) is saved on the stack (ISP_TR).
(4) 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.
(5) The value of the program counter (PC) is saved on the stack (ISP_TR).
(6) The vector is fetched from the value of EXTB + address 0000 0008h.
(7) The PC is set to the fetched address and processing branches to the start of the exception handling routine.

5.5.5 Address Exceptions

(1) The value of the processor status word (PSW) is saved on the stack (ISP).
(2) The processor mode select bit (PM), the stack pointer select bit (U), and the interrupt enable bit (I) in the PSW are cleared to 0.
(3) The value of the program counter (PC) is saved on the stack (ISP).
(4) The vector is fetched from the value of EXTB + address 0000 0060h.
(5) The PC is set to the fetched address and processing branches to the start of the exception handling routine.

5.5.6 Single-Precision Floating-Point Exceptions

(1) The value of the processor status word (PSW) is saved on the stack (ISP).
(2) The processor mode select bit (PM), the stack pointer select bit (U), and the interrupt enable bit (I) in the PSW are cleared to 0.
(3) The value of the program counter (PC) is saved on the stack (ISP).
(4) The vector is fetched from the value of EXTB + address 0000 0064h.
(5) The PC is set to the fetched address and processing branches to the start of the exception handling routine.

5.5.7 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.

5.5.8 Non-Maskable Interrupts (with Normal Acceptance)

(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[4:0]) in the PSW register are set to 0Fh if the current instruction is being executed in the non-secure world; if not, they are set to 1Fh.
(5) The vector is fetched from the value of EXTB + address 0000 0078h.
(6) The PC is set to the fetched address and processing branches to the start of the exception handling routine.

5.5.9 Non-Maskable Interrupts (with World-Transition Acceptance)

(1) The value in WID.WID is saved to WID.BWID.
(2) Set WID.WID to 0 and transition to the root world.
(3) The destination world ID of the non-maskable interrupt is stored in the WID.BITW bits.
(4) The value of the processor status word (PSW) is saved on the stack (ISP_TR).
(5) 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.
(6) 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_TR). For other instructions, the PC value of the next instruction is saved.
(7) The processor interrupt priority level bits (IPL[4:0]) in the PSW are set to 1Fh.
(8) The vector is fetched from the value of EXTB + address 0000 0018h.
(9) The PC is set to the fetched address and processing branches to the start of the exception handling routine.

5.5.10 Interrupts (with Normal Acceptance)

(1) The value of the processor status word (PSW) is saved on the stack (ISP) or, for the fast interrupt, in the backup PSW (BPSW).
(2) The processor mode select bit (PM), the stack pointer select bit (U), and the interrupt enable bit (I) in the PSW are cleared to 0.
(3) If the interrupt was generated during the execution of an RMPA, SCMPU, SMOVB, SMOVF, SMOVU, SSTR, SUNTIL, or SWHILE instruction, the value of the program counter (PC) for that instruction is saved on the stack

(ISP). For other instructions, the PC value of the next instruction is saved. Saving of the PC is in the backup PC (BPC) for fast interrupts and on the stack for other interrupts.
(4) The interrupt priority level of the accepted interrupt is set in the processor interrupt priority level bits (IPL[4:0]) in the PSW.
(5) The vector for an interrupt source other than the fast interrupt is fetched from the interrupt vector table. For the fast interrupt, the address is fetched from the fast interrupt vector register (FINTV).
(6) The PC is set to the fetched address and processing branches to the start of the exception handling routine.

5.5.11 Interrupts (with World-Transition Acceptance)

(1) The value in WID.WID is saved to WID.BWID.
(2) Set WID.WID to 0 and transition to the root world.
(3) The destination world ID of the interrupt is stored in the WID.BITW bits.
(4) The interrupt source number is stored in the IVC.BWINM bits in the root world and in the destination world of the interrupt.
(5) The value of the processor status word (PSW) is saved on the stack (ISP_TR).
(6) 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.
(7) 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_TR). For other instructions, the PC value of the next instruction is saved.
(8) The interrupt priority level of the accepted interrupt is set in the processor interrupt priority level bits (IPL[4:0]) in the PSW.
(9) The vector corresponding to the destination world of the interrupt is fetched from the vector table for the interrupt that caused the world transition.
(10) The PC is set to the fetched address and processing branches to the start of the exception handling routine.

5.5.12 Unconditional Trap (with Normal Acceptance)

(1) The value of the processor status word (PSW) is saved on the stack (ISP).
(2) The processor mode select bit (PM), the stack pointer select bit (U), and the interrupt enable bit (I) in the PSW are cleared to 0.
(3) The value of the program counter (PC) is saved on the stack (ISP).
(4) For the INT instruction, the value at the vector corresponding to the INT instruction number is fetched from the interrupt vector table.
For the BRK instruction, the value at the vector from the start address is fetched from the interrupt vector table.
(5) The PC is set to the fetched address and processing branches to the start of the exception handling routine.

5.5.13 Unconditional Trap (with World-Transition Acceptance)

(1) The value in WID.WID is saved to WID.BWID.
(2) Set WID.WID to 0 and transition to the root world.
(3) The value of the PSW.I bit is stored in the WID.BWI bit.
(4) The value of the processor status word (PSW) is saved on the stack (ISP_TR).
(5) 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.
(6) The value of the program counter (PC) for the next instruction is saved on the stack (ISP_TR).
(7) The vector corresponding to the TRCALL instruction number is fetched from the root call vector table.
(8) 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 5.6 at the end of the corresponding exception handling routines restores the values of the program counter (PC) and processor status word (PSW) that were saved on the stack or in the backup PC (BPC) or the backup PSW (BPSW) by the hardware preprocessing.

Table 5.6 Return from Exception Handling Routines

Exception		Instruction for Return
Undefined instruction exception		RTE
Privileged instruction exception		RTE
Access exception		RTE
Secure exception		TRRTE*1
Address exception		RTE
Single-precision floating-point exceptions		RTE
Reset		Return is impossible
Non-maskable interrupt (with normal acceptance)		RTE*2
Non-maskable interrupt (with world-transition acceptance)		TRRTE*2
Interrupts (with normal acceptance)	Fast interrupt	RTFI
	Other than the above	RTE
Interrupts (with world-transition acceptance)	Fast interrupt	—
	Other than the above	TRRTE
Unconditional trap (with normal acceptance)		RTE
Unconditional trap (with world-transition acceptance)		TRRTE

Note: 1. Returning to the original program is not possible.

Note: 2. Returning to the original program is not recommended. In addition, unless a Flat-Project is in use, returning to the original program being executed when the exception was generated is not possible.

5.7 Order of Priority for Exceptions

The order of priority for exceptions is given in Table 5.7. When multiple exceptions are generated at the same time, the exception with the highest priority is accepted first.

Table 5.7 Order of Priority for Exceptions

Order of Priority		Exception
High	1	Reset
↑	2	Non-maskable interrupt
	3	Interrupts
	4	Instruction access exception
	5	Instruction secure exception
	6	Undefined instruction exception
		Privileged instruction exception
	7	Unconditional trap
	8	Address exception
	9	Operand access exception
	10	Operand secure exception
Low	11	Single-precision floating-point exceptions

5.8 Exception Generated in Coprocessor

This section describes exceptions generated in the coprocessor and handling of them.

5.8.1 Double-Precision Floating-Point Exceptions

Double-precision floating-point exceptions are generated when any of the five exceptions specified in the IEEE 754 standard, namely overflow, underflow, inexact, division-by-zero, or invalid operation, or an attempt to use processing that is not implemented, is detected upon execution of a double-precision floating-point arithmetic instruction. When a double-precision floating-point exception is generated, an interrupt request is sent to the interrupt controller according to the settings (the DEX, DEU, DEZ, DEO, and DEV bits of the DPSW corresponding to the five exceptions, and the EHM bit of DECNT) without the CPU performing exception handling.

For the vector numbers allocated to the interrupt controller, refer to the user's manuals: hardware for the respective products.

Index

Numerics

0 flush bit of denormalized number (DDN bit) 55
0 flush bit of denormalized number (DN bit) 29

A

ACC0, ACC1 (accumulator) 36
access exception 391
accumulator (ACC0, ACC1) 36
accumulator direct 68
address exception 391

B

backup destination world ID (BITW[1:0] bits) 32
backup interrupt enable for source world of transition (BWI bit) 32
backup PC (BPC) 26
backup PSW (BPSW) 26
backup world ID (BWID[1:0] bits) 32
backup world transition interrupt source number (BWINM[7:0] bits) 31
BITW[1:0] bits (backup destination world ID) 32
bitwise operations 43
BPC (backup PC) 26
BPSW (backup PSW) 26
BWI bit (backup interrupt enable for source world of transition) 32
BWID[1:0] bits (backup world ID) 32
BWINM[7:0] bits (backup world transition interrupt source number) 31

C

C flag (carry flag) 24
carry flag (C flag) 24
CE flag (unimplemented processing cause flag) 28
CO flag (overflow cause flag) 28
control register direct 67
control registers 19
CU flag (underflow cause flag) 28
CV flag (invalid operation cause flag) 28
CX flag (inexact cause flag) 28
CZ flag (division-by-zero cause flag) 28

D

DCE flag (unimplemented processing cause flag) 54
DCMR (double-precision floating-point comparison result register) 55
DCO flag (overflow cause flag) 54
DCU flag (underflow cause flag) 54
DCV flag (invalid operation cause flag) 54
DCX flag (inexact cause flag) 54
DCZ flag (division-by-zero cause flag) 54
DDN bit (0 flush bit of denormalized number) 55
DECNT (double-precision floating-point exception handling control register) 56
DEO bit (overflow exception enable bit) 55
DEPC (double-precision floating-point exception program counter) 56
DEU bit (underflow exception enable bit) 55
DEV bit (invalid operation exception enable bit) 55
DEX bit (inexact exception enable bit) 55
DEZ bit (division-by-zero exception enable bit) 55
DFO flag (overflow flag) 55
DFS flag (double-precision floating-point error summary flag) 55
DFU flag (underflow flag) 55
DFV flag (invalid operation flag) 55
DFX flag (inexact flag) 55
DFZ flag (division-by-zero flag) 55
division-by-zero cause flag (CZ flag) 28
division-by-zero cause flag (DCZ flag) 54
division-by-zero exception enable bit (DEZ bit) 55
division-by-zero exception enable bit (EZ bit) 29
division-by-zero flag (DFZ flag) 55
division-by-zero flag (FZ flag) 29
DN bit (0 flush bit of denormalized number) 29
double-precision floating-point compare instruction result flag (RES flag) 55
double-precision floating-point comparison result register (DCMR) 55
double-precision floating-point control register direct 68
double-precision floating-point coprocessor 51
double-precision floating-point data register direct 68
double-precision floating-point error summary flag (DFS flag) 55
double-precision floating-point exception handling control register (DECNT) 56
double-precision floating-point exception information saving mode bit (EHM bit) 56
double-precision floating-point exception information saving status bit (EHS bit) 56
double-precision floating-point exception program counter (DEPC) 56
double-precision floating-point exceptions 57, 407
double-precision floating-point number 60

double-precision floating-point rounding-mode setting bits (DRM[1:0] bits) 54
double-precision floating-point status word (DPSW) 53
DPSW (double-precision floating-point status word) 53
DRM[1:0] bits (double-precision floating-point rounding-mode setting bits) 54

E

EHM bit (double-precision floating-point exception information saving mode bit) 56
EHS bit (double-precision floating-point exception information saving status bit) 56
EO bit (overflow exception enable bit) 29
EU bit (underflow exception enable bit) 29
EV bit (invalid operation exception enable bit) 29
EX bit (inexact exception enable bit) 29
exception vector table 45
exception vector table register (EXTB) 29
EXCSE bit (secure exception detection during hardware pre-processing) 31
EXTB (exception vector table register) 29
EZ bit (division-by-zero exception enable bit) 29

F

fast interrupt vector register (FINTV) 26
FINTV (fast interrupt vector register) 26
FO flag (overflow flag) 29
FPSW (single-precision floating-point status word) 27
FS flag (single-precision floating-point error summary flag) 29
FU flag (underflow flag) 29
FV flag (invalid operation flag) 29
FX flag (inexact flag) 29
FZ flag (division-by-zero flag) 29

G

general-purpose registers (R0 (SP) to R15) 19

I

I bit (interrupt enable bit) 25
immediate 64
indexed register indirect 66
inexact cause flag (CX flag) 28
inexact cause flag (DCX flag) 54
inexact exception enable bit (DEX bit) 55
inexact exception enable bit (EX bit) 29
inexact flag (DFX flag) 55
inexact flag (FX flag) 29
INTB (interrupt table register) 22
integer 42
interrupt enable bit (I bit) 25
interrupt stack pointer (ISP) 22
interrupt table register (INTB) 22
interrupt vector control register (IVC) 30
interrupt vector table 46
interrupts 392
invalid operation cause flag (CV flag) 28
invalid operation cause flag (DCV flag) 54
invalid operation exception enable bit (DEV bit) 55
invalid operation exception enable bit (EV bit) 29
invalid operation flag (DFV flag) 55
invalid operation flag (FV flag) 29
IPL[4:0] bits (processor interrupt priority level) 25
ISP (interrupt stack pointer) 22
IVC (interrupt vector control register) 30

N

NaN (Not a Number) 39
non-maskable interrupt 392
non-secure interrupt enable control (NSIC bit) 31
NSIC bit (non-secure interrupt enable control) 31

O

O flag (overflow flag) 25
order of priority for exceptions 406
overflow cause flag (CO flag) 28
overflow cause flag (DCO flag) 54
overflow exception enable bit (DEO bit) 55
overflow exception enable bit (EO bit) 29
overflow flag (DFO flag) 55
overflow flag (FO flag) 29
overflow flag (O flag) 25

P

PC (program counter) 22
PM bit (processor mode select bit) 25
post-increment register indirect 66
pre-decrement register indirect 66
privileged instruction 40
privileged instruction exception 391
processor interrupt priority level (IPL[4:0] bits) 25
processor mode 40
processor mode select bit (PM bit) 25
processor status word (PSW) 23
program counter (PC) 22

program counter relative 67
Protect Zone technology 17
PSW (processor status word) 23
PSW direct 67

Q

QNaN (Quiet NaN) 39

R

R0 (SP) to R15 (general-purpose registers) 19
register (n) 79
register bank save function 49
register direct 65
register indirect 65
register relative 65
register_num (Rn) 79
RES flag (double-precision floating-point compare instruction result flag) 55
reset 391
RM[1:0] bits (single-precision floating-point rounding-mode setting bits) 28
root call vector table 47
root call vector table register (TRCTB) 33
Rounding to the nearest value 28, 54
Rounding towards –∞ 28, 54
Rounding towards +∞ 28, 54
Rounding towards 0 ..... 28, 54

S

S flag (sign flag) 24
secure exception 391
secure exception detection during hardware pre-processing (EXCSE bit) 31
sign flag (S flag) 24
single-precision floating-point error summary flag (FS flag) . 29
single-precision floating-point exceptions 37, 391
single-precision floating-point number 42
single-precision floating-point rounding-mode setting bits (RM[1:0] bits) 28
single-precision floating-point status word (FPSW) 27
size extension specifier 81
size specifier 79
SNaN (Signaling NaN) 39
software interrupt restriction (UIR[1:0] bits) 31
stack pointer (R0 (SP)) 19
stack pointer select bit (U bit) 25
strings 43
supervisor mode 40

T

TRCTB (root call vector table register) 33

U

U bit (stack pointer select bit) 25
UIR[1:0] bits (software interrupt restriction) 31
unconditional trap 393
undefined instruction exception 391
underflow cause flag (CU flag) 28
underflow cause flag (DCU flag) 54
underflow exception enable bit (DEU bit) 55
underflow exception enable bit (EU bit) 29
underflow flag (DFU flag) 55
underflow flag (FU flag) 29
unimplemented processing cause flag (CE flag) 28
unimplemented processing cause flag (DCE flag) 54
user mode 40
user stack pointer (USP) 22
USP (user stack pointer) 22

V

vector table 45
vector table for interrupts that cause world transitions 47
vector table register for interrupts that cause world transitions (WINTB) 33

W

WID (world ID register) 32
WID[1:0] bits (world ID) 32
WINTB (vector table register for interrupts that cause world transitions) 33
WIPL[4:0] bits (world-shared processor interrupt priority level) 34
world ID (WID[1:0] bits) 32
world ID register (WID) 32
world-shared processor interrupt priority level (WIPL[4:0] bits) 34
world-shared processor interrupt priority level register (WPRI) 34
WPRI (world-shared processor interrupt priority level register) 34

Z

Z flag (zero flag) 24
zero flag (Z flag) 24

REVISION HISTORY	RX Family RXv3e Instruction Set Architecture User's Manual: Software

		Description	
Rev.	**Date**	**Page**	**Summary**
1.01	Aug. 06, 2026	—	First edition issued

RX Family RXv3e Instruction Set Architecture User’s Manual: Software

Publication Date: Rev.1.01 Aug 06, 2026

Published by: Renesas Electronics Corporation

RX Family RXv3e Instruction Set Architecture

Renesas Electronics Corporation

R01US0745EJ0101