#### Old Company Name in Catalogs and Other Documents

On April 1<sup>st</sup>, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1<sup>st</sup>, 2010 Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

#### Notice

- All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
- Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
- 3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
- 4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
- 5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
- 6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
- 7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics atta abooks, etc.
  - "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.
  - "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anticrime systems; safety equipment; and medical equipment not specifically designed for life support.
  - "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
- 8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
- 9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system 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. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU ROHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
- 11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
- Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.
- (Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majorityowned subsidiaries.
- (Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.



The revision list can be viewed directly by clicking the title page.

The revision list summarizes the locations of revisions and additions. Details should always be checked by referring to the relevant text.

# 16

# H8S/2600 Series, H8S/2000 Series

Software Manual

Renesas 16-Bit Single-Chip Microcomputer H8S Family

Renesas Electronics www.renesas.com

Rev.4.00 2006.02

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

#### Notes regarding these materials

- These materials are intended as a reference to assist our customers in the selection of the Renesas Technology Corp. product best suited to the customer's application; they do not convey any license under any intellectual property rights, or any other rights, belonging to Renesas Technology Corp. or a third party.
- 2. Renesas Technology Corp. assumes no responsibility for any damage, or infringement of any thirdparty's rights, originating in the use of any product data, diagrams, charts, programs, algorithms, or circuit application examples contained in these materials.
- 3. All information contained in these materials, including product data, diagrams, charts, programs and algorithms represents information on products at the time of publication of these materials, and are subject to change by Renesas Technology Corp. without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corp. or an authorized Renesas Technology Corp. product distributor for the latest product information before purchasing a product listed herein.

The information described here may contain technical inaccuracies or typographical errors. Renesas Technology Corp. assumes no responsibility for any damage, liability, or other loss rising from these inaccuracies or errors.

Please also pay attention to information published by Renesas Technology Corp. by various means, including the Renesas Technology Corp. Semiconductor home page (http://www.renesas.com).

- 4. When using any or all of the information contained in these materials, including product data, diagrams, charts, programs, and algorithms, please be sure to evaluate all information as a total system before making a final decision on the applicability of the information and products. Renesas Technology Corp. assumes no responsibility for any damage, liability or other loss resulting from the information contained herein.
- 5. Renesas Technology Corp. semiconductors are not designed or manufactured for use in a device or system that is used under circumstances in which human life is potentially at stake. Please contact Renesas Technology Corp. or an authorized Renesas Technology Corp. product distributor when considering the use of a product contained herein for any specific purposes, such as apparatus or systems for transportation, vehicular, medical, aerospace, nuclear, or undersea repeater use.
- 6. The prior written approval of Renesas Technology Corp. is necessary to reprint or reproduce in whole or in part these materials.
- 7. If these products or technologies are subject to the Japanese export control restrictions, they must be exported under a license from the Japanese government and cannot be imported into a country other than the approved destination.
  Any diversion or reexport control to the export control laws and regulations of Japan and/or the

Any diversion or reexport contrary to the export control laws and regulations of Japan and/or the country of destination is prohibited.

8. Please contact Renesas Technology Corp. for further details on these materials or the products contained therein.



# Preface

The H8S/2600 Series and the H8S/2000 Series are built around an H8S/2000 CPU core.

The H8S/2600 and H8S/2000 CPUs have the same internal 32-bit architecture. Both CPUs execute basic instructions in one state, have sixteen 16-bit registers, and have a concise, optimized instruction set. They can address a 16-Mbyte linear address space.Programs coded in the high-level language C can be compiled to high-speed executable code.

For easy migration, the instruction set is upward-compatible with the H8/300H, H8/300, and H8/300L Series at the object-code level.

The H8S/2600 CPU is upward-compatible with the H8S/2000 CPU at the object-code level, and supports sum of products instructions.

This manual gives details of the H8S/2600 and H8S/2000 instructions and can be sued with all microcontrollers in the H8S/2600 Series and the H8S/2000 Series.

For hardware details, refer to the relevant microcontroller hardware manuals.

Rev. 4.00 Feb 24, 2006 page iii of xiv

Rev. 4.00 Feb 24, 2006 page iv of xiv



# Main Revisions for This Edition

Item	Page	Revisions (See Manual for Details)									
1.1.1 Features	2	Note * add	led								
		— Maximu	ım clock fr	equency: 2	20 MHz*						
		Note: * Th execution t					ruction				
2.2.22 CLRMAC	90	Further ex	planation a	added to n	ote						
Operand Format and Number of States Required for Execution		The numbe For details question.		-							
2.2.24 DAA	94	Table ame	nded								
Description		C Flag before Adjustment	Upper 4 Bits before Adjustment	H Flag before Adjustment	Lower 4 Bits before Adjustment	Value Added (Hexadecimal)	C Flag after Adjustment				
		0	A to F	1	0 to 3	66	1				
		1	0 to 2 0 to 2	0	0 to 9 A to F	60 66	1 1				
		1	0 to 3	1	0 to 3	66	1				
2.2.37 LDMAC	130	Further ex	planation a	added to n	ote						
Operand Format and Number of States Required for Execution		The number of states may differ depending on the product. For details, refer to the hardware manual of the product in question.									
2.2.42 (1) MULXS (B)	151										
Operand Format and Number of States Required for Execution											
2.2.42 (2) MULXS (W)	152	_									
Operand Format and Number of States Required for Execution											
2.2.43 (1) MULXU (B)	153	_									
Operand Format and Number of States Required for Execution											
2.2.43 (2) MULXU (W)	_										
Operand Format and Number of States Required for Execution											

Item	Page	Revisions (See Manual for Details)								
2.2.64 STMAC	232	Table amended								
Operand Format and		Instruction Format								
Number of States		1st byte 2nd byte 3rd byte 4th byte								
Required for Execution		0 2 2 0 erd								
		0 2 3 0 erd								
		Further explanation added to note								
		The number of states may differ depending on the product.								
		For details, refer to the hardware manual of the product in								
		question.								
2.3 Instruction Set	250,	Note 7 amended and note 10 added								
Table 2.1 Instruction	251,	No. of								
Set	260, 262	States <sup>*1</sup>								
	202	Mnemonic								
		Normal								
		DAS         DAS Rd         1           MULXU         MULXU.B Rs,Rd         3 (12*7)								
		MULXU.W Rs,ERd 4 (20*7)								
		MULXS MULXSB Rs,Rd 4 (13*7)								
		MULXS WRS, ERd 5 (21+7)								
		*5*t0								
		Mnemonic Intersection Mnemonic								
		CLRMAC**0         CLRMAC         2*6*10           LDMAC**0         LDMAC ERS,MACH         2*6*10								
		LDMAC ERS,MACL 2*6 *10								
		STMAC <sup>*9</sup> STMAC MACH,ERd 1*6*10 STMAC MACL,ERd 1*6*10								
		No. of States <sup>+1</sup>								
		Mnemonic								
		Normal								
		2         2           TRAPA         TRAPA #x:2         7[9]*7[8]9]*7								
		RTE RTE 5(9) <sup>766</sup>								
		Notes: 7. Values in parentheses () are for the H8S/2000 CPU. Values in square brackets [] apply to interrupt control modes 2 and 3. 10. The number of states may differ depending on the product. For details, refer to the hardware manual of the product in								

Rev. 4.00 Feb 24, 2006 page vi of xiv

# RENESAS

question.

Item	Page	Revisi	ons (See	Manu	al for D	etails	)			
2.6 Number of States Required for Instruction Execution	283, 285, 286,	Note 6	added		Instruction Fetch	Branch Address Read	Stack Operation	Byte Data Access	Word Data Access	Internal Operation
Table 2.5 Number of	288	Instruction CLRMAC* <sup>5</sup>	Mnemonic CLRMAC		1	J 1	ĸ	L	м	N *3 *6
Cycles in Instruction Execution					Instruction Fetch	Branch Address Read	Stack Operation	Byte Data Access	Word Data Access	Internal Operation
		Instruction	Mnemonic			J	ĸ	. L	M	N
		LDMAC*5	LDMAC ERS,MACH		1 1	1 1				*3 *6
					Instruction Fetch	Branch Address Read	Stack Operation	Byte Data Access	Word Data Access	Internal Operation
		Instruction	Mnemonic			J	к	L	м	N
		MULXS		H8S/2600 H8S/2000	2	2 1				*3*6 1
			MULXS.W Rs,ERd		2	3				\$3 \$6
				H8S/2000	2	1				9
		MULXU		H8S/2600	1	2				3%E
			MULXU.W Rs,ERd	H8S/2000	1	1				1
				H8S/2000	1	1				9
					Instruction Fetch	Branch Address Read	Stack Operation	Byte Data Access	Word Data Access	Internal Operation
		Instruction	Mnemonic			J	к	. L	м	N
		STMAC*5	STMAC MACH,ERd STMAC MACL,ERd		1					0 <sup>*3*6</sup>
			OTMAO MAOL,ERG							
		produc	6. The nu ct. For deta ct in questio	ils, re						the
2.7 Bus States During Instruction Execution	290	:M del	eted from I	egenc	k					
Table 2.6 Instruction Execution Cycles	293 to 302	All inst	ances of :I	M dele	eted from	m table	Э			
3.3.5 Usage Notes	312, 313	Newly	added							





# Contents

Secti	on 1	CPU	1
1.1	Overvi	ew	1
	1.1.1	Features	1
	1.1.2	Differences between H8S/2600 CPU and H8S/2000 CPU	2
	1.1.3	Differences from H8/300 CPU	3
	1.1.4	Differences from H8/300H CPU	4
1.2	CPU O	perating Modes	5
1.3	Addres	s Space	10
1.4	Registe	er Configuration	11
	1.4.1	Overview	11
	1.4.2	General Registers	12
	1.4.3	Control Registers	13
	1.4.4	Initial Register Values	15
1.5	Data F	ormats	16
	1.5.1	General Register Data Formats	16
	1.5.2	Memory Data Formats	18
1.6	Instruc	tion Set	19
	1.6.1	Overview	19
	1.6.2	Instructions and Addressing Modes	20
	1.6.3	Table of Instructions Classified by Function	22
	1.6.4	Basic Instruction Formats	32
1.7	Addres	sing Modes and Effective Address Calculation	33
Secti	$n^{2}$	Instruction Descriptions	41
2.1		and Symbols	41
2.1	2.1.1	Assembly-Language Format	42
	2.1.1	Operation	43
	2.1.2	Condition Code	44
	2.1.5	Instruction Format	44
	2.1.4	Register Specification	45
	2.1.6	Bit Data Access in Bit Manipulation Instructions	46
2.2		tion Descriptions	47
2.2	2.2.1 (1		48
	2.2.1 (2		49
	2.2.1 (3		50
	2.2.1 (.	ADD (L)	51
	2.2.2	ADDX	52
	2.2.3		53
		Rev. 4.00 Feb 24, 2006 page ix of	

2.2.4 (2)	AND (W)	54
2.2.4 (3)	AND (L)	55
2.2.5 (1)	ANDC	56
2.2.5 (2)	ANDC	57
2.2.6	BAND	58
2.2.7	Bcc	60
2.2.8	BCLR	62
2.2.9	BIAND	64
2.2.10	BILD	66
2.2.11	BIOR	68
2.2.12	BIST	70
2.2.13	BIXOR	72
2.2.14	BLD	74
2.2.15	BNOT	76
2.2.16	BOR	78
2.2.17	BSET	80
2.2.18	BSR	82
2.2.19	BST	84
2.2.20	BTST	86
2.2.21	BXOR	88
2.2.22	CLRMAC	90
2.2.23 (1)	CMP (B)	91
2.2.23 (2)	CMP (W)	92
2.2.23 (3)	CMP (L)	93
2.2.24	DAA	94
2.2.25	DAS	96
2.2.26(1)	DEC (B)	98
2.2.26 (2)	DEC (W)	99
2.2.26 (3)	DEC (L)	100
2.2.27 (1)	DIVXS (B)	101
2.2.27 (2)	DIVXS (W)	103
2.2.28 (1)	DIVXU (B)	105
2.2.28 (2)	DIVXU (W)	107
2.2.29 (1)	EEPMOV (B)	109
2.2.29 (2)	EEPMOV (W)	110
2.2.30(1)	EXTS (W)	112
2.2.30 (2)	EXTS (L)	113
2.2.31 (1)	EXTU (W)	114
2.2.31 (2)	EXTU (L)	115
2.2.32 (1)	INC (B)	116
2.2.32 (2)	INC (W)	117

Rev. 4.00 Feb 24, 2006 page x of xiv

2.2.32 (3)	INC (L)	118
2.2.33	JMP	119
2.2.34	JSR	120
2.2.35 (1)	LDC (B)	122
2.2.35 (2)	LDC (B)	123
2.2.35 (3)	LDC (W)	124
2.2.35 (4)	LDC (W)	126
2.2.36	LDM	128
2.2.37	LDMAC	130
2.2.38	MAC	131
2.2.39 (1)	MOV (B)	134
2.2.39 (2)	MOV (W)	135
2.2.39 (3)	MOV (L)	136
2.2.39 (4)	MOV (B)	137
2.2.39 (5)	MOV (W)	139
2.2.39 (6)	MOV (L)	141
2.2.39 (7)	MOV (B)	143
2.2.39 (8)	MOV (W)	145
	MOV (L)	
2.2.40	MOVFPE	149
2.2.41	MOVTPE	150
2.2.42 (1)	MULXS (B)	151
2.2.42 (2)	MULXS (W)	152
2.2.43 (1)	MULXU (B)	153
2.2.43 (2)	MULXU (W)	154
2.2.44 (1)	NEG (B)	155
2.2.44 (2)	NEG (W)	156
2.2.44 (3)	NEG (L)	157
2.2.45	NOP	158
2.2.46(1)	NOT (B)	159
2.2.46 (2)	NOT (W)	160
	NOT (L)	
2.2.47 (1)	OR (B)	162
2.2.47 (2)	OR (W)	163
2.2.47 (3)	OR (L)	164
	ORC	
	ORC	
2.2.49 (1)	POP (W)	167
	POP (L)	
	PUSH (W)	
2.2.50 (2)	PUSH (L)	170

Rev. 4.00 Feb 24, 2006 page xi of xiv

2.2.51 (2) ROTL (B)	•••••	171
		172
2.2.51 (3) ROTL (W)		173
2.2.51 (4) ROTL (W)		174
2.2.51 (5) ROTL (L)		175
2.2.51 (6) ROTL (L)		176
2.2.52 (1) ROTR (B)		177
2.2.52 (2) ROTR (B)		178
2.2.52 (3) ROTR (W)		179
2.2.52 (4) ROTR (W)		180
2.2.52 (5) ROTR (L)		181
2.2.52 (6) ROTR (L)		182
2.2.53 (1) ROTXL (B)		183
2.2.53 (2) ROTXL (B)		184
2.2.53 (3) ROTXL (W)		185
2.2.53 (4) ROTXL (W)		186
2.2.53 (5) ROTXL (L)		187
2.2.53 (6) ROTXL (L)		188
2.2.54 (1) ROTXR (B)		189
2.2.54 (2) ROTXR (B)		190
2.2.54 (3) ROTXR (W)		191
2.2.54 (4) ROTXR (W)		192
2.2.54 (5) ROTXR (L)		193
2.2.54 (6) ROTXR (L)		194
2.2.55 RTE		195
2.2.56 RTS		197
2.2.57 (1) SHAL (B)		198
2.2.57 (2) SHAL (B)		199
2.2.57 (3) SHAL (W)		200
2.2.57 (4) SHAL (W)		201
2.2.57 (5) SHAL (L)		202
2.2.57 (6) SHAL (L)		203
2.2.58 (1) SHAR (B)		204
2.2.58 (2) SHAR (B)		205
2.2.58 (3) SHAR (W)		206
2.2.58 (4) SHAR (W)		207
2.2.58 (5) SHAR (L)		
		209
2.2.58 (6) SHAR (L)		
2.2.58 (6) SHAR (L) 2.2.59 (1) SHLL (B)		210

Rev. 4.00 Feb 24, 2006 page xii of xiv

	2.2.59 (4) SHLL (W)	213
	2.2.59 (5) SHLL (L)	214
	2.2.59 (6) SHLL (L)	215
	2.2.60 (1) SHLR (B)	216
	2.2.60 (2) SHLR (B)	217
	2.2.60 (3) SHLR (W)	218
	2.2.60 (4) SHLR (W)	219
	2.2.60 (5) SHLR (L)	220
	2.2.60 (6) SHLR (L)	221
	2.2.61 SLEEP	222
	2.2.62 (1) STC (B)	223
	2.2.62 (2) STC (B)	224
	2.2.62 (3) STC (W)	225
	2.2.62 (4) STC (W)	227
	2.2.63 STM	229
	2.2.64 STMAC	231
	2.2.65 (1) SUB (B)	233
	2.2.65 (2) SUB (W)	235
	2.2.65 (3) SUB (L)	236
	2.2.66 SUBS	237
	2.2.67 SUBX	238
	2.2.68 TAS	239
	2.2.69 TRAPA	240
	2.2.70 (1) XOR (B)	242
	2.2.70 (2) XOR (W)	243
	2.2.70 (3) XOR (L)	244
	2.2.71 (1) XORC	245
	2.2.71 (2) XORC	246
2.3	Instruction Set	247
2.4	Instruction Code	263
2.5	Operation Code Map	274
2.6	Number of States Required for Instruction Execution	278
2.7	Bus States During Instruction Execution	290
2.8	Condition Code Modification	304
	on 3 Processing States	
3.1	Overview	
3.2	Reset State	
3.3	Exception-Handling State	
	3.3.1 Types of Exception Handling and Their Priority	
	3.3.2 Reset Exception Handling	312

Rev. 4.00 Feb 24, 2006 page xiii of xiv

	3.3.3	Trace	312
	3.3.4	Interrupt Exception Handling and Trap Instruction Exception Handling	312
	3.3.5	Usage Notes	312
3.4	Program	n Execution State	314
3.5	Bus-Rel	eased State	315
3.6	Power-I	Down State	315
	3.6.1	Sleep Mode	315
	3.6.2	Software Standby Mode	315
	3.6.3	Hardware Standby Mode	316
Section	on 4 E	Basic Timing	317
4.1	Overvie	W	317
4.2	On-Chij	p Memory (ROM, RAM)	317
4.3	On-Chij	p Supporting Module Access Timing	319
4.4	Externa	Address Space Access Timing	320



# Section 1 CPU

#### 1.1 Overview

The H8S/2600 CPU and the H8S/2000 CPU are high-speed central processing units with a common an internal 32-bit architecture. Each CPU is upward-compatible with the H8/300 and H8/300H CPUs. The H8S/2600 CPU and H8S/2000 CPU have sixteen 16-bit general registers, can address a 4-Gbyte linear address space, and are ideal for realtime control.

#### 1.1.1 Features

The H8S/2600 CPU and H8S/2000 CPU have the following features.

- Upward-compatible with H8/300 and H8/300H CPUs
  - Can execute H8/300 and H8/300H object programs
- General-register architecture
  - Sixteen 16-bit general registers (also usable as sixteen 8-bit registers or eight 32-bit registers)
- Sixty-nine basic instructions (H8S/2000 CPU has sixty-five)
  - 8/16/32-bit arithmetic and logic instructions
  - Multiply and divide instructions
  - Powerful bit-manipulation instructions
  - Multiply-and-accumulate instruction (H8S/2600 CPU only)
- Eight addressing modes
  - Register direct [Rn]
  - Register indirect [@ERn]
  - Register indirect with displacement [@(d:16,ERn) or @(d:32,ERn)]
  - Register indirect with post-increment or pre-decrement [@ERn+ or @-ERn]
  - Absolute address [@aa:8, @aa:16, @aa:24, or @aa:32]
  - Immediate [#xx:8, #xx:16, or #xx:32]
  - Program-counter relative [@(d:8,PC) or @(d:16,PC)]
  - Memory indirect [@@aa:8]
- 4-Gbyte address space
  - Program: 16 Mbytes
  - Data: 4 Gbytes

- High-speed operation
  - All frequently-used instructions execute in one or two states
  - Maximum clock frequency: 20 MHz\*
  - 8/16/32-bit register-register add/subtract: 50 ns
  - 8  $\times$  8-bit register-register multiply:
  - 16 ÷ 8-bit register-register divide:
  - 16 × 16-bit register-register multiply: 200 ns (H8S/2000 CPU: 1000 ns)
  - $-32 \div 16$ -bit register-register divide:
- Two CPU operating modes
  - Normal mode
  - Advanced mode
- Power-down modes
  - Transition to power-down state by SLEEP instruction
  - CPU clock speed selection
- Note: \* The maximum operating frequency and instruction execution time differ depending on the product.

600 ns

1000 ns

150 ns (H8S/2000 CPU: 600 ns)

#### 1.1.2 Differences between H8S/2600 CPU and H8S/2000 CPU

Differences between the H8S/2600 CPU and the H8S/2000 CPU are as follows.

- Register configuration
  - The MAC register is supported only by the H8S/2600 CPU.

For details, see section 1.4, Register Configuration.

- Basic instructions
  - The MAC, CLRMAC, LDMAC, and STMAC instructions are supported only by the H8S/2600 CPU.
    - For details, see section 1.6, Instruction Set, and Section 2, Instruction Descriptions.
- Number of states required for execution
  - The number of states required for execution of the MULXU and MULXS instructions.
     For details, see section 2.6, Number of States Required for Execution.

In addition, there may be defferences in address spaces, EXR register functions, power-down states, and so on. For details, refer to the relevant microcontroller hardware manual.

#### 1.1.3 Differences from H8/300 CPU

In comparison with the H8/300 CPU, the H8S/2600 CPU and H8S/2000 CPU have the following enhancements.

- More general registers and control registers
  - Eight 16-bit registers, one 8-bit and two 32-bit control registers have been added.
- Expanded address space
  - Normal mode supports the same 64-kbyte address space as the H8/300 CPU.
  - Advanced mode supports a maximum 4-Gbyte address space.
- Enhanced addressing
  - The addressing modes have been enhanced to make effective use of the 4-Gbyte address space.
- Enhanced instructions
  - Addressing modes of bit-manipulation instructions have been enhanced.
  - Signed multiply and divide instructions have been added.
  - A multiply-and-accumulate instruction has been added. (H8S/2600CPU only)
  - Two-bit shift and rotate instructions have been added.
  - Instructions for saving and restoring multiple registers have been added.
  - A test and set instruction has been added.
- Higher speed
  - Basic instructions execute twice as fast.

#### 1.1.4 Differences from H8/300H CPU

In comparison with the H8/300H CPU, the H8S/2600 CPU and H8S/2000 CPU have the following enhancements.

- Additional control register
  - One 8-bit and two 32-bit control registers have been added.
- Expanded address space
  - Advanced mode supports a maximum 4-Gbyte data address space.
- Enhanced instructions
  - Addressing modes of bit-manipulation instructions have been enhanced.
  - A multiply-and-accumulate instruction has been added (H8S/2600 CPU only).
  - Two-bit shift and rotate instructions have been added.
  - Instructions for saving and restoring multiple registers have been added.
  - A test and set instruction has been added.
- Higher speed
  - Basic instructions execute twice as fast.



# **1.2 CPU Operating Modes**

Like the H8/300H CPU, the H8S/2600 CPU has two operating modes: normal and advanced. Normal mode supports a maximum 64-kbyte address space. Advanced mode supports a maximum 4-Gbyte total address space, of which up to 16 Mbytes can be used for program code and up to 4 Gbytes for data. The mode is selected with the mode pins of the microcontroller. For further information, refer to the relevant microcontroller hardware manual.



Figure 1.1 CPU Operating Modes

#### (1) Normal Mode

The exception vector table and stack have the same structure as in the H8/300 CPU.

Address Space: A maximum address space of 64 kbytes can be accessed, as in the H8/300 CPU.

**Extended Registers (En):** The extended registers (E0 to E7) can be used as 16-bit registers, or as the upper 16-bit segments of 32-bit registers. When En is used as a 16-bit register it can contain any value, even when the corresponding general register (R0 to R7) is used as an address register. If the general register is referenced in the register indirect addressing mode with pre-decrement (@-Rn) or post-increment (@Rn+) and a carry or borrow occurs, however, the value in the corresponding extended register will be affected.

**Instruction Set:** All additional instructions and addressing modes not found in the H8/300 CPU can be used. Only the lower 16 bits of effective addresses (EA) are valid.

**Exception Vector Table and Memory Indirect Branch Addresses:** In normal mode the top area starting at H'0000 is allocated to the exception vector table. One branch address is stored per 16 bits (figure 1.2). The exception vector table differs depending on the microcontroller. Refer to the relevant microcontroller hardware manual for further information.



Figure 1.2 Exception Vector Table (Normal Mode)

The memory indirect addressing mode (@@aa:8) employed in the JMP and JSR instructions uses an 8-bit absolute address included in the instruction code to specify a memory operand that contains a branch address. In normal mode the operand is a 16-bit word operand, providing a 16-bit branch address. Branch addresses can be stored in the top area from H'0000 to H'00FF. Note that this area is also used for the exception vector table.



**Stack Structure:** When the program counter (PC) is pushed onto the stack in a subroutine call, and the PC, condition-code register (CCR), and extended control register (EXR) are pushed onto the stack in exception handling, they are stored as shown in figure 1.3. When EXR is invalid, it is not pushed onto the stack. For details, see the relevant hardware manual.



Figure 1.3 Stack Structure in Normal Mode

#### (2) Advanced Mode

In advanced mode the data address space is larger than for the H8/300H CPU.

Address Space: The 4-Gbyte maximum address space provides linear access to a maximum 16 Mbytes of program code and maximum 4 Gbytes of data.

**Extended Registers (En):** The extended registers (E0 to E7) can be used as 16-bit registers, or as the upper 16-bit segments of 32-bit registers or address registers.

Instruction Set: All instructions and addressing modes can be used.

**Exception Vector Table and Memory Indirect Branch Addresses:** In advanced mode the top area starting at H'00000000 is allocated to the exception vector table in units of 32 bits. In each 32 bits, the upper 8 bits are ignored and a branch address is stored in the lower 24 bits (figure 1.4). The exception vector table differs depending on the microcontroller. Refer to the relevant microcontroller hardware manual for further information.



Figure 1.4 Exception Vector Table (Advanced Mode)

The memory indirect addressing mode (@@aa:8) employed in the JMP and JSR instructions uses an 8-bit absolute address included in the instruction code to specify a memory operand that contains a branch address. In advanced mode the operand is a 32-bit longword operand, providing a 32-bit branch address. The upper 8 bits of these 32 bits are a reserved area that is regarded as H'00. Branch addresses can be stored in the top area from H'00000000 to H'000000FF. Note that this area is also used for the exception vector table.

**Stack Structure:** In advanced mode, when the program counter (PC) is pushed onto the stack in a subroutine call, and the PC, condition-code register (CCR), and extended control register (EXR) are pushed onto the stack in exception handling, they are stored as shown in figure 1.5. When EXR is invalid, it is not pushed onto the stack. For details, see the relevant hardware manual.



Figure 1.5 Stack Structure in Advanced Mode



# 1.3 Address Space

Figure 1.6 shows a memory map of the H8S/2600 CPU. The H8S/2600 CPU provides linear access to a maximum 64-kbyte address space in normal mode, and a maximum 4-Gbyte address space in advanced mode. The address space differs depending on the operating mode. For details, refer to the relevant microcontroller hardware manual.



Figure 1.6 Memory Map

#### **1.4 Register Configuration**

#### 1.4.1 Overview

The CPUs have the internal registers shown in figure 1.7. There are two types of registers: general registers and control registers. The H8S/2000 CPU does not support the MAC register.



#### Figure 1.7 CPU Registers

#### 1.4.2 General Registers

The CPUs have eight 32-bit general registers. These general registers are all functionally alike and can be used as both address registers and data registers. When a general register is used as a data register, it can be accessed as a 32-bit, 16-bit, or 8-bit register. When the general registers are used as 32-bit registers or address registers, they are designated by the letters ER (ER0 to ER7).

The ER registers divide into 16-bit general registers designated by the letters E (E0 to E7) and R (R0 to R7). These registers are functionally equivalent, providing a maximum sixteen 16-bit registers. The E registers (E0 to E7) are also referred to as extended registers.

The R registers divide into 8-bit general registers designated by the letters RH (R0H to R7H) and RL (R0L to R7L). These registers are functionally equivalent, providing a maximum sixteen 8-bit registers.

Figure 1.8 illustrates the usage of the general registers. The usage of each register can be selected independently.



Figure 1.8 Usage of General Registers

General register ER7 has the function of stack pointer (SP) in addition to its general-register function, and is used implicitly in exception handling and subroutine calls. Figure 1.9 shows the stack.





Figure 1.9 Stack

#### 1.4.3 Control Registers

The control registers are the 24-bit program counter (PC), 8-bit extended control register (EXR), 8-bit condition-code register (CCR), and 64-bit multiply-accumulate register (MAC: H8S/2600 CPU only).

#### (1) Program Counter (PC)

This 24-bit counter indicates the address of the next instruction the CPU will execute. The length of all CPU instructions is 16 bits (one word) or a multiple of 16 bits, so the least significant PC bit is ignored. When an instruction is fetched, the least significant PC bit is regarded as 0.

#### (2) Extended Control Register (EXR)

This 8-bit register contains the trace bit (T) and three interrupt mask bits (I2 to I0).

**Bit 7—Trace Bit (T):** Selects trace mode. When this bit is cleared to 0, instructions are executed in sequence. When this bit is set to 1, a trace exception is generated each time an instruction is executed.

Bits 6 to 3—Reserved: These bits are reserved, always read as 1.

**Bits 2 to 0—Interrupt Mask Bits (I2 to I0):** These bits designate the interrupt mask level (0 to 7). For details refer to the relevant microcontroller hardware manual.

Operations can be performed on the EXR bits by the LDC, STC, ANDC, ORC, and XORC instructions. All interrupts, including NMI, are disabled for three states after one of these instructions is executed, except for STC.

#### (3) Condition-Code Register (CCR)

This 8-bit register contains internal CPU status information, including an interrupt mask bit (I) and half-carry (H), negative (N), zero (Z), overflow (V), and carry (C) flags.

**Bit 7—Interrupt Mask Bit (I):** Masks interrupts other than NMI when set to 1. (NMI is accepted regardless of the I bit setting.) The I bit is set to 1 by hardware at the start of an exception-handling sequence.

**Bit 6—User Bit or Interrupt Mask Bit (UI):** Can be written and read by software using the LDC, STC, ANDC, ORC, and XORC instructions. This bit can also be used as an interrupt mask bit. For details refer to the relevant microcontroller hardware manual.

**Bit 5—Half-Carry Flag (H):** When the ADD.B, ADDX.B, SUB.B, SUBX.B, CMP.B, or NEG.B instruction is executed, this flag is set to 1 if there is a carry or borrow at bit 3, and cleared to 0 otherwise. When the ADD.W, SUB.W, CMP.W, or NEG.W instruction is executed, the H flag is set to 1 if there is a carry or borrow at bit 11, and cleared to 0 otherwise. When the ADD.L, SUB.L, CMP.L, or NEG.L instruction is executed, the H flag is set to 1 if there is a carry or borrow at bit 27, and cleared to 0 otherwise.

**Bit 4—User Bit (U):** Can be written and read by software using the LDC, STC, ANDC, ORC, and XORC instructions.

Bit 3—Negative Flag (N): Stores the value of the most significant bit (sign bit) of data.

Bit 2—Zero Flag (Z): Set to 1 to indicate zero data, and cleared to 0 to indicate non-zero data.

**Bit 1—Overflow Flag (V):** Set to 1 when an arithmetic overflow occurs, and cleared to 0 at other times.

Bit 0—Carry Flag (C): Set to 1 when a carry occurs, and cleared to 0 otherwise. Used by:

- Add instructions, to indicate a carry
- Subtract instructions, to indicate a borrow
- Shift and rotate instructions, to store the value shifted out of the end bit

The carry flag is also used as a bit accumulator by bit manipulation instructions.

Some instructions leave some or all of the flag bits unchanged. For the action of each instruction on the flag bits, refer to the detailed descriptions of the instructions starting in section 2.2.1.

Operations can be performed on the CCR bits by the LDC, STC, ANDC, ORC, and XORC instructions. The N, Z, V, and C flags are used as branching conditions for conditional branch (Bcc) instructions.

#### (4) Multiply-Accumulate Register (MAC)

The MAC register is supported only by the H8S/2600 CPU. This 64-bit register stores the results of multiply-and-accumulate operations. It consists of two 32-bit registers denoted MACH and MACL. The lower 10 bits of MACH are valid; the upper bits are a sign extension.

#### 1.4.4 Initial Register Values

Reset exception handling loads the CPU's program counter (PC) from the vector table, clears the trace bit in EXR to 0, and sets the interrupt mask bits in CCR and EXR to 1. The other CCR bits and the general registers are not initialized. In particular, the stack pointer (ER7) is not initialized. The stack pointer should therefore be initialized by an MOV.L instruction executed immediately after a reset.

# 1.5 Data Formats

The CPUs can process 1-bit, 4-bit (BCD), 8-bit (byte), 16-bit (word), and 32-bit (longword) data. Bit-manipulation instructions operate on 1-bit data by accessing bit n (n = 0, 1, 2, ..., 7) of byte operand data. The DAA and DAS decimal-adjust instructions treat byte data as two digits of 4-bit BCD data.

#### **1.5.1** General Register Data Formats

Data Type	Register Number	Data Format
1-bit data	RnH	7 0 7 6 5 4 3 2 1 0 Don't care
1-bit data	RnL	7         0           Don't care         7         6         5         4         3         2         1         0
4-bit BCD data	RnH	7 4 3 0 Upper Lower Don't care
4-bit BCD data	RnL	7 4 3 0 Don't care Upper Lower
Byte data	RnH	7 0 Don't care MSB LSB
Byte data	RnL	7     0       Don't care

Figure 1.10 shows the data formats in general registers.

Figure 1.10 General Register Data Formats



Figure 1.10 General Register Data Formats (cont)



#### 1.5.2 Memory Data Formats

Figure 1.11 shows the data formats in memory. The CPU can access word data and longword data in memory, but word or longword data must begin at an even address. If an attempt is made to access word or longword data at an odd address, no address error occurs but the least significant bit of the address is regarded as 0, so the access starts at the preceding address. This also applies to instruction fetches.

Data Type				Dat	a Fo	orma	ıt		
	Address				_	_			
		7							0
1-bit data	Address L	7	6	5	4	3	2	1	0
Byte data	Address L	MSB				- - -	1		LSB
Word data	Address 2M	MSB							
	Address 2M + 1								LSB
Longword data	Address 2N	MSB				- - -	1		
	Address 2N + 1			-	1	-	-	-	1 1 1
	Address 2N + 2	1							
	Address 2N + 3			,     					LSB
					_	<u> </u>			

Figure 1.11 Memory Data Formats

When the stack pointer (ER7) is used as an address register to access the stack, the operand size should be word size or longword size.



#### 1.6 Instruction Set

#### 1.6.1 Overview

The H8S/2600 CPU has 69types of instructions, while the H8S/2000 CPU has 65 types. The instructions are classified by function as shown in table 1.1. For a detailed description of each instruction, see section 2.2, Instruction Descriptions.

Function	Instructions	Size	Types
Data transfer	MOV	BWL	5
	POP <sup>*2</sup> , PUSH <sup>*2</sup>	WL	
	LDM, STM	L	
	MOVFPE, MOVTPE	В	
Arithmetic	ADD, SUB, CMP, NEG	BWL	19
operations	ADDX, SUBX, DAA, DAS	В	
	INC, DEC	BWL	
	ADDS, SUBS	L	
	MULXU, DIVXU, MULXS, DIVXS	BW	
	EXTU, EXTS	WL	
	TAS <sup>*4</sup>	В	_
	MAC, LDMAC, STMAC, CLRMAC <sup>*1</sup>	_	4 <sup>*1</sup>
Logic operations	AND, OR, XOR, NOT	BWL	4
Shift	SHAL, SHAR, SHLL, SHLR, ROTL, ROTR, ROTXL, ROTXR	BWL	8
Bit manipulation	BSET, BCLR, BNOT, BTST, BLD, BILD, BST, BIST, BAND, BIAND, BOR, BIOR, BXOR, BIXOR	В	14
Branch	Bcc*3, JMP, BSR, JSR, RTS	_	5
System control	TRAPA, RTE, SLEEP, LDC, STC, ANDC, ORC, XORC, NOP	_	9
Block data transfer	EEPMOV	_	1
	rd size gword size		
H8S/26 2. POP.W @–SP. ERn, @	-	10V.W F	Rn,
	he generic designation of a conditional branch instruction.	TAS inot	ruction

#### Table 1.1 Instruction Classification

4. Only register ER0, ER1, ER4, or ER5 should be used when using the TAS instruction.

#### 1.6.2 Instructions and Addressing Modes

Table 1.2 indicates the combinations of instructions and addressing modes that the H8S/2600 CPU and H8S/2000 CPU can use.

_																			
	_	1	WL	Γ	I	Ι	Ι	—	Ι	Ι	I	I	I	Ι	I	I		0	I
Addressing Modes	8:66 @ @	1	I	I	I	I	I	I	Ι	Ι	I	I	I	I	I	I		I	T
	@(d:16,PC)	1	I	-	I	Ι	I	-	I	Ι	I	I	I	Ι	I	I	1	I	I
	(JA,8:b)@	1	I	Ι	I	I	Ι	Ι	Ι	Ι	I	I	I	I	I	I	1	I	Ι
	26:66@	BWL	I	Ι	I	Ι	Ι	Ι	Ι	Ι	I	I	I	Ι	I	I	1	I	Ι
	42:66@	1	I	Ι	I	I	I	Ι	I	Ι	I	I	I	I	I	I	1	I	Ι
	91:66@	BWL	I	-	в	I	I	-	Ι	Ι	I	I	I	I	I	I	1	I	Ι
	8:66@	æ	I	I	I	I	Ι	I	Ι	Ι	I	I	I	I	I	I	1	I	Ι
	+nЯ∃@\nЯ∃-@	BWL	I	Ι	I	I	I	Ι	I	I	I	I	I	I	I	I	0	I	Ι
	@(d:32,ERn)	BWL	Ι	Ι	I	I	I	Ι	Ι	Ι	I	I	I	I	I	I	I	Ι	Ι
	(nЯ∃,ð1:b)@	BWL	I	Ι	I	I	I	Ι	I	Ι	I	I	I	I	I	1	1	I	Ι
	@E&u	BWL	I	Ι	I	I	I	Ι	Ι	I	I	I	I	I	I	в	1	I	Ι
	uy	BWL	I	Ι	I	BWL	BWL	В	Г	BWL	в	BW	BW	BWL	WL	1	1	I	L
	xx#	BWL	I	-	I	BWL	WL	В	I	Ι	I	I	I	-	I	I		I	Ι
Instruction			POP, PUSH	LDM, STM	MOVEPE, MOVTPE	ADD, CMP	SUB	ADDX, SUBX	ADDS, SUBS	INC, DEC	DAA, DAS	DIVXU, MULXU,	MULXS, DIVXS	NEG	EXTU, EXTS	TAS*2	MAC*1	CLRMAC*1	LDMAC <sup>*1</sup> , STMAC <sup>*1</sup>
Function			transfer			Arithmetic operations													

#### Table 1.2 Combinations of Instructions and Addressing Modes


$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$									Addressing Modes	g Modes						
AND, OR, XOR         BWL         U	ç	Instruction	xx#	uŊ	u8∃@	(nЯ∃,ð1:b)@	@(d:32,ERn)	+u83@\n83-@	8:66@	91:66@	@33:24	26:66@	(Jq,8:b)@	(Jq,ðr:b)@	8:66@@	_
NOT         =         BWL         = <td>suc</td> <td>AND, OR, XOR</td> <td>BWL</td> <td>BWL</td> <td> </td> <td> </td> <td> </td> <td> </td> <td> </td> <td>I</td> <td> </td> <td> </td> <td> </td> <td> </td> <td> </td> <td> </td>	suc	AND, OR, XOR	BWL	BWL						I						
		NOT	Ι	BWL	I	1	1	1	I	I	I	1	I	I	I	I
			I	BWL	1					1	1		1	1	1	1
Bcc, BSR	nipulat	tion	I	в	в	1	1	1	в	в	I	в	I	I	I	I
	_	Bcc, BSR	Ι	Ι	I	I	1	1	I	I	I	1	0	0	I	I
RTS <td></td> <td>JMP, JSR</td> <td>I</td> <td>1</td> <td>1</td> <td> </td> <td> </td> <td> </td> <td> </td> <td>1</td> <td>0</td> <td> </td> <td>1</td> <td>1</td> <td>0</td> <td>1</td>		JMP, JSR	I	1	1					1	0		1	1	0	1
TRAPA </th <td></td> <td>RTS</td> <td>Ι</td> <td>Ι</td> <td>I</td> <td>1</td> <td>1</td> <td>1</td> <td>I</td> <td>I</td> <td>I</td> <td>1</td> <td>I</td> <td>I</td> <td>I</td> <td>0</td>		RTS	Ι	Ι	I	1	1	1	I	I	I	1	I	I	I	0
RTE <td>_</td> <td>TRAPA</td> <td>I</td> <td>I</td> <td>I</td> <td>1</td> <td>1</td> <td>1</td> <td>I</td> <td>I</td> <td>I</td> <td>1</td> <td>I</td> <td>I</td> <td>I</td> <td>0</td>	_	TRAPA	I	I	I	1	1	1	I	I	I	1	I	I	I	0
EP       -	-	RTE	1	1	1			1				1			1	0
B       B       W       W       W       I       I         C, C, C, C, C, C, C, C, C, C, C, C, C, C		SLEEP	I	I	1	1	1	1	I	1	I	1	1	1	I	0
C, C, B       H       W       H       H         C, C, C       B       W       H       H       H         H       H       H       H       H       H       H         H       H       H       H       H       H       H       H         H       H       H       H       H       H       H       H       H         H <td></td> <td>LDC</td> <td>в</td> <td>۵</td> <td>8</td> <td>×</td> <td>8</td> <td>×</td> <td> </td> <td>&gt;</td> <td>I</td> <td>×</td> <td>I</td> <td>I</td> <td>I</td> <td>I</td>		LDC	в	۵	8	×	8	×		>	I	×	I	I	I	I
C, C	-	STC	1	۵	8	×	8	×	1	3		×	1	1	1	
1     1       1     1	-	ANDC, ORC, XORC	ш	I	I	I	I	I	I	I	I	I	I	I	I	I
		NOP	I	1	1	1	1	1	1	1	1	1	1	1	1	0
	lata tra	ansfer	Ι	Ι		1		1			1	1		Ι		BW

W: Word L: Longword

Notes: 1. Supported only by the H8S/2600 CPU 2. Only register ER0, ER1, ER4, or ER5 should be used when using the TAS instruction.

# RENESAS

# Rev. 4.00 Feb 24, 2006 page 21 of 322 REJ09B0139-0400

### **1.6.3** Table of Instructions Classified by Function

Table 1.3 summarizes the instructions in each functional category. The notation used in table 1.3 is defined next.

### **Operation Notation**

Rd	General register (destination)*
Rs	General register (source)*
Rn	General register*
ERn	General register (32-bit register)
MAC	Multiply-accumulate register (32-bit register)
(EAd)	Destination operand
(EAs)	Source operand
EXR	Extended control register
CCR	Condition-code register
N	N (negative) flag in CCR
Z	Z (zero) flag in CCR
V	V (overflow) flag in CCR
С	C (carry) flag in CCR
PC	Program counter
SP	Stack pointer
#IMM	Immediate data
disp	Displacement
+	Addition
_	Subtraction
×	Multiplication
÷	Division
^	Logical AND
V	Logical OR
$\oplus$	Logical exclusive OR
$\rightarrow$	Move
-	Logical not (logical complement)
:8/:16/:24/:32	8-, 16-, 24-, or 32-bit length

to R7, E0 to E7), and 32-bit registers (ER0 to ER7).

Туре	Instruction	Size <sup>*1</sup>	Function
Data transfer	MOV	B/W/L	$(EAs) \to Rd, \ Rs \to (EAd)$
			Moves data between two general registers or between a general register and memory, or moves immediate data to a general register.
	MOVFPE	В	$(EAs) \rightarrow Rd$
			Moves external memory contents (addressed by @aa:16) to a general register in synchronization with an E clock.
	MOVTPE	В	$Rs \to (EAs)$
			Moves general register contents to an external memory location (addressed by @aa:16) in synchronization with an E clock.
	POP	W/L	$@SP+ \rightarrow Rn$
			Pops a register from the stack. POP.W Rn is identical to MOV.W @SP+, Rn. POP.L ERn is identical to MOV.L @SP+, ERn.
	PUSH	W/L	$Rn \rightarrow @-SP$
			Pushes a register onto the stack. PUSH.W Rn is identical to MOV.W Rn, @–SP. PUSH.L ERn is identical to MOV.L ERn, @–SP.
	LDM	L	@SP+ $\rightarrow$ Rn (register list)
			Pops two or more general registers from the stack.
	STM	L	Rn (register list) $\rightarrow$ @–SP
			Pushes two or more general registers onto the stack.

### Table 1.3 Instructions Classified by Function

Туре	Instruction	Size <sup>*1</sup>	Function
Arithmetic	ADD	B/W/L	$Rd\pmRs\toRd,\ Rd\pm\#IMM\toRd$
operations	SUB		Performs addition or subtraction on data in two general registers, or on immediate data and data in a general register. (Immediate byte data cannot be subtracted from byte data in a general register. Use the SUBX or ADD instruction.)
	ADDX	В	$Rd\pmRs\pmC\toRd,\ Rd\pm\#IMM\pmC\toRd$
	SUBX		Performs addition or subtraction with carry or borrow on byte data in two general registers, or on immediate data and data in a general register.
	INC	B/W/L	$Rd\pm 1\toRd,\ Rd\pm 2\toRd$
	DEC		Increments or decrements a general register by 1 or 2. (Byte operands can be incremented or decremented by 1 only.)
	ADDS	L	$Rd\pm 1\toRd,\ Rd\pm 2\toRd,\ Rd\pm 4\toRd$
	SUBS		Adds or subtracts the value 1, 2, or 4 to or from data in a 32-bit register.
	DAA	В	Rd decimal adjust $\rightarrow$ Rd
	DAS		Decimal-adjusts an addition or subtraction result in a general register by referring to the CCR to produce 4-bit BCD data.
	MULXU	B/W	$Rd \times Rs \to Rd$
			Performs unsigned multiplication on data in two general registers: either 8 bits $\times$ 8 bits $\rightarrow$ 16 bits or 16 bits $\times$ 16 bits $\rightarrow$ 32 bits.
	MULXS	B/W	$Rd \times Rs \to Rd$
			Performs signed multiplication on data in two general registers: either 8 bits $\times$ 8 bits $\rightarrow$ 16 bits or 16 bits $\times$ 16 bits $\rightarrow$ 32 bits.
	DIVXU	B/W	$Rd \div Rs \to Rd$
			Performs unsigned division on data in two general registers: either 16 bits $\div$ 8 bits $\rightarrow$ 8-bit quotient and 8-bit remainder or 32 bits $\div$ 16 bits $\rightarrow$ 16-bit quotient and 16-bit remainder.
	DIVXS	B/W	$Rd \div Rs \to Rd$
			Performs signed division on data in two general registers: either 16 bits $\div$ 8 bits $\rightarrow$ 8-bit quotient and 8-bit remainder or 32 bits $\div$ 16 bits $\rightarrow$ 16-bit quotient and 16-bit remainder.

Section 1 CPU

Туре	Instruction	Size <sup>*1</sup>	Function
Arithmetic	CMP	B/W/L	Rd – Rs, Rd – #IMM
operations			Compares data in a general register with data in another general register or with immediate data, and sets CCR bits according to the result.
	NEG	B/W/L	$0 - Rd \rightarrow Rd$
			Takes the two's complement (arithmetic complement) of data in a general register.
	EXTU	W/L	Rd (zero extension) $\rightarrow$ Rd
			Extends the lower 8 bits of a 16-bit register to word size, or the lower 16 bits of a 32-bit register to longword size, by padding with zeros on the left.
	EXTS	W/L	Rd (sign extension) $\rightarrow$ Rd
			Extends the lower 8 bits of a 16-bit register to word size, or the lower 16 bits of a 32-bit register to longword size, by extending the sign bit.
	TAS	В	$@ERd - 0, 1 \rightarrow (\text{ of } @ERd)^{*2}$
			Tests memory contents, and sets the most significant bit (bit 7) to 1.
	MAC	_	$(EAs) \times (EAd) + MAC \rightarrow MAC$
			Performs signed multiplication on memory contents and adds the result to the multiply-accumulate register. The following operations can be performed:
			16 bits $\times$ 16 bits +32 bits $\rightarrow$ 32 bits, saturating 16 bits $\times$ 16 bits + 42 bits $\rightarrow$ 42 bits, non-saturating
			Supported by H8S/2600 CPU only.
	CLRMAC	_	$0 \rightarrow MAC$
			Clears the multiply-accumulate register to zero.
			Supported by H8S/2600 CPU only.
	LDMAC	L	$Rs \to MAC,  MAC \to Rd$
	STMAC		Transfers data between a general register and the multiply-accumulate register.
			Supported by H8S/2600 CPU only.

Туре	Instruction	Size <sup>*1</sup>	Function
Logic operations	AND	B/W/L	$Rd \land Rs \to Rd, \ Rd \land \#IMM \to Rd$
			Performs a logical AND operation on a general register and another general register or immediate data.
	OR	B/W/L	$Rd \lor Rs \to Rd, \ Rd \lor \#IMM \to Rd$
			Performs a logical OR operation on a general register and another general register or immediate data.
	XOR	B/W/L	$Rd \oplus Rs \to Rd, \ Rd \oplus \#IMM \to Rd$
			Performs a logical exclusive OR operation on a general register and another general register or immediate data.
	NOT	B/W/L	$\neg$ (Rd) $\rightarrow$ (Rd)
			Takes the one's complement of general register contents.
Shift operations	SHAL	B/W/L	$Rd \text{ (shift)} \rightarrow Rd$
	SHAR		Performs an arithmetic shift on general register contents.
			1-bit or 2-bit shift is possible.
	SHLL	B/W/L	$Rd \text{ (shift)} \rightarrow Rd$
	SHLR		Performs a logical shift on general register contents.
			1-bit or 2-bit shift is possible.
	ROTL	B/W/L	Rd (rotate) $\rightarrow$ Rd
	ROTR		Rotates general register contents.
			1-bit or 2-bit rotation is possible.
	ROTXL	B/W/L	$Rd$ (rotate) $\rightarrow Rd$
	ROTXR		Rotates general register contents through the carry bit.
			1-bit or 2-bit rotation is possible.



Bit-manipulation instructionsBSETB $1 \rightarrow ( of )$ Sets a specified bit in a general register or memory operand to 1. The bit number is specified by 3-bit immediate data or the lower three bits of a general register.BCLRB $0 \rightarrow ( of )$ Clears a specified bit in a general register or memory operand to 0. The bit number is specified by 3-bit immediate data or the lower three bits of a general register.BNOTB $\neg ( of )$ ( <bit-no.> of <ead>) Inverts a specified bit in a general register or memory operand. The bit number is specified by 3-bit immediate data or the lower three bits of a general register.BNOTB<math>\neg (<bit-no.> of <ead>) \rightarrow (<bit-no.> of <ead>)</ead></bit-no.></ead></bit-no.></math> Inverts a specified bit in a general register or memory operand. The bit number is specified by 3-bit immediate data or the lower three bits of a general register.BTSTB<math>\neg (<bit-no.> of <ead>) \rightarrow Z</ead></bit-no.></math> Tests a specified bit in a general register or memory operand and sets or clears the Z flag accordingly. Th bit number is specified by 3-bit immediate data or the data or the lower three bits of a general register.</ead></bit-no.>	
Sets a specified bit in a general register of memory operand to 1. The bit number is specified by 3-bit immediate data or the lower three bits of a general register.BCLRB $0 \rightarrow ( of )$ Clears a specified bit in a general register or memory operand to 0. The bit number is specified by 3-bit immediate data or the lower three bits of a general register.BNOTB $\neg ( of ) \rightarrow ( of )$ Inverts a specified bit in a general register or memory operand. The bit number is specified by 3-bit immediate data or the lower three bits of a general register.BNOTB $\neg ( of ) \rightarrow ( of )$ Inverts a specified bit in a general register or memory operand. The bit number is specified by 3-bit immediate data or the lower three bits of a general register.BTSTB $\neg ( of ) \rightarrow Z$ Tests a specified bit in a general register or memory operand and sets or clears the Z flag accordingly. Th	
Clears a specified bit in a general register or memory operand to 0. The bit number is specified by 3-bit immediate data or the lower three bits of a general register.BNOTB $\neg$ ( <bit-no.> of <ead>) <math>\rightarrow</math> (<bit-no.> of <ead>) Inverts a specified bit in a general register or memory operand. The bit number is specified by 3-bit immediate data or the lower three bits of a general register.BTSTB<math>\neg</math> (<bit-no.> of <ead>) <math>\rightarrow</math> Z Tests a specified bit in a general register or memory operand and sets or clears the Z flag accordingly. The</ead></bit-no.></ead></bit-no.></ead></bit-no.>	iry
operand to 0. The bit number is specified by 3-bit immediate data or the lower three bits of a general register.BNOTB $\neg$ ( <bit-no.> of <ead>) <math>\rightarrow</math> (<bit-no.> of <ead>)Inverts a specified bit in a general register or memory operand. The bit number is specified by 3-bit immediate data or the lower three bits of a general register.BTSTB<math>\neg</math> (<bit-no.> of <ead>) <math>\rightarrow</math> Z Tests a specified bit in a general register or memory operand and sets or clears the Z flag accordingly. The</ead></bit-no.></ead></bit-no.></ead></bit-no.>	ory
Inverts a specified bit in a general register or memory operand. The bit number is specified by 3-bit immediate data or the lower three bits of a general register.BTSTB $\neg$ ( <bit-no.> of <ead>) <math>\rightarrow</math> ZTests a specified bit in a general register or memory operand and sets or clears the Z flag accordingly. The</ead></bit-no.>	
operand. The bit number is specified by 3-bit immediate data or the lower three bits of a general register. BTST B $\neg$ ( <bit-no.> of <ead>) <math>\rightarrow</math> Z Tests a specified bit in a general register or memory operand and sets or clears the Z flag accordingly. The</ead></bit-no.>	
Tests a specified bit in a general register or memory operand and sets or clears the Z flag accordingly. The	ory
operand and sets or clears the Z flag accordingly. The	
lower three bits of a general register.	The
$\begin{array}{ccc} BAND & B & C \land (<\!bit\!-\!No.\!> of <\!\mathsf{EAd\!\!>}) \to C \end{array}$	
ANDs the carry flag with a specified bit in a general register or memory operand and stores the result in t carry flag.	
$\begin{array}{ccc} BIAND & B & C \land \neg \mbox{ ( of < EAd>)} \rightarrow C \end{array}$	
ANDs the carry flag with the inverse of a specified bi a general register or memory operand and stores the result in the carry flag.	
The bit number is specified by 3-bit immediate data.	۱.
$\begin{array}{ccc} BOR & B & C \lor (<\!bit\!-\!No.\!> of <\!\mathsf{EAd\!\!>}) \to C \end{array}$	
ORs the carry flag with a specified bit in a general register or memory operand and stores the result in t carry flag.	the
$BIOR \qquad B \qquad C \lor \neg (<\!bit\!-\!No.\!> of <\!\mathsf{EAd\!\!>}) \to C$	
ORs the carry flag with the inverse of a specified bit in a general register or memory operand and stores the result in the carry flag.	
The bit number is specified by 3-bit immediate data.	

Туре	Instruction	Size <sup>*1</sup>	Function
Bit-manipulation	BXOR	В	$C \oplus (<\!bit\!-\!No.\!> of <\!\mathsf{EAd\!\!>}) \to C$
instructions			Exclusive-ORs the carry flag with a specified bit in a general register or memory operand and stores the result in the carry flag.
	BIXOR	В	$C \oplus \neg \text{ ( of )} \rightarrow C$
			Exclusive-ORs the carry flag with the inverse of a specified bit in a general register or memory operand and stores the result in the carry flag.
			The bit number is specified by 3-bit immediate data.
	BLD	В	( <bit-no.> of <ead>) <math>\rightarrow</math> C</ead></bit-no.>
			Transfers a specified bit in a general register or memory operand to the carry flag.
	BILD	В	$\neg$ ( <bit-no.> of <ead>) <math>\rightarrow</math> C</ead></bit-no.>
			Transfers the inverse of a specified bit in a general register or memory operand to the carry flag.
			The bit number is specified by 3-bit immediate data.
	BST	В	$C \rightarrow (\text{ of })$
			Transfers the carry flag value to a specified bit in a general register or memory operand.
	BIST	В	$\neg$ C $\rightarrow$ ( <bit-no.> of <ead>)</ead></bit-no.>
			Transfers the inverse of the carry flag value to a specified bit in a general register or memory operand.
			The bit number is specified by 3-bit immediate data.

Section 1 CPU



Туре	Instruction	Size <sup>*1</sup>	Function				
Branch instructions	Bcc		Branches to a specified address if a specified condit is true. The branching conditions are listed below.				
			Mnemonic	Description	Condition		
			BRA(BT)	Always (true)	Always		
			BRN(BF)	Never (false)	Never		
			BHI	High	$C \lor Z = 0$		
			BLS	Low or same	C ∨ Z = 1		
			BCC(BHS)	Carry clear (high or same)	C = 0		
			BCS(BLO)	Carry set (low)	C = 1		
			BNE	Not equal	Z = 0		
			BEQ	Equal	Z = 1		
			BVC	Overflow clear	V = 0		
			BVS	Overflow set	V = 1		
			BPL	Plus	N = 0		
			BMI	Minus	N = 1		
			BGE	Greater or equal	$N \oplus V = 0$		
			BLT	Less than	N ⊕ V = 1		
			BGT	Greater than	$Z \lor (N \oplus V) = 0$		
			BLE	Less or equal	$Z \lor (N \oplus V) = 1$		
	JMP	_	Branches unco	onditionally to a specif	fied address.		
	BSR		Branches to a	subroutine at a specif	fied address.		
	JSR	_	Branches to a	subroutine at a specif	fied address.		
	RTS	—	Returns from a	a subroutine			

Туре	Instruction	Size <sup>*1</sup>	Function
System control	TRAPA	_	Starts trap-instruction exception handling.
instructions	RTE	—	Returns from an exception-handling routine.
	SLEEP		Causes a transition to a power-down state.
	LDC	B/W	$(EAs) \rightarrow CCR, (EAs) \rightarrow EXR$
			Moves the source operand contents or immediate data to CCR or EXR. Although CCR and EXR are 8-bit registers, word-size transfers are performed between them and memory. The upper 8 bits are valid.
	STC	B/W	$CCR \rightarrow (EAd), EXR \rightarrow (EAd)$
			Transfers CCR or EXR contents to a general register or memory. Although CCR and EXR are 8-bit registers, word-size transfers are performed between them and memory. The upper 8 bits are valid.
	ANDC	В	$CCR \land \#IMM \to CCR,  EXR \land \#IMM \to EXR$
			Logically ANDs the CCR or EXR contents with immediate data.
	ORC	В	$CCR \lor \#IMM \to CCR, EXR \lor \#IMM \to EXR$
			Logically ORs the CCR or EXR contents with immediate data.
	XORC	В	$CCR \oplus \#IMM \to CCR, EXR \oplus \#IMM \to EXR$
			Logically exclusive-ORs the CCR or EXR contents with immediate data.
	NOP	_	$PC + 2 \rightarrow PC$
			Only increments the program counter.

# Rev. 4.00 Feb 24, 2006 page 30 of 322 REJ09B0139-0400

Section 1 CPU



Туре	Instruction	Size <sup>*1</sup>	Function
Block data transfer instruction	EEPMOV.B		if R4L $\neq$ 0 then Repeat @ER5+ $\rightarrow$ @ER6+ R4L - 1 $\rightarrow$ R4L Until R4L = 0 else next;
	EEPMOV.W	_	if R4 ≠ 0 then Repeat @ER5+ → @ER6+ R4 – 1 → R4 Until R4 = 0 else next;
			Transfers a data block according to parameters set in general registers R4L or R4, ER5, and ER6.
			R4L or R4: size of block (bytes)ER5:starting source addressER6:starting destination address
			Execution of the next instruction begins as soon as the transfer is completed.
Notes: 1. Size	efers to the ope	rand size	
D. D.	.4 -		

B: Byte

W: Word

- L: Longword
- 2. Only register ER0, ER1, ER4, or ER5 should be used when using the TAS instruction.

# 1.6.4 Basic Instruction Formats

The H8S/2600 or H8S/2000 instructions consist of 2-byte (1-word) units. An instruction consists of an operation field (op field), a register field (r field), an effective address extension (EA field), and a condition field (cc).

**Operation Field:** Indicates the function of the instruction, the addressing mode, and the operation to be carried out on the operand. The operation field always includes the first four bits of the instruction. Some instructions have two operation fields.

**Register Field:** Specifies a general register. Address registers are specified by 3 bits, data registers by 3 bits or 4 bits. Some instructions have two register fields. Some have no register field.

Effective Address Extension: Eight, 16, or 32 bits specifying immediate data, an absolute address, or a displacement.

Condition Field: Specifies the branching condition of Bcc instructions.

Figure 1.12 shows examples of instruction formats.



Figure 1.12 Instruction Formats

# 1.7 Addressing Modes and Effective Address Calculation

# (1) Addressing Modes

The CPUs support the eight addressing modes listed in table 1.4. Each instruction uses a subset of these addressing modes. Arithmetic and logic instructions can use the register direct and immediate modes. Data transfer instructions can use all addressing modes except program-counter relative and memory indirect. Bit manipulation instructions use register direct, register indirect, or absolute addressing mode to specify an operand, and register direct (BSET, BCLR, BNOT, and BTST instructions) or immediate (3-bit) addressing mode to specify a bit number in the operand.

No.	Addressing Mode	Symbol
1	Register direct	Rn
2	Register indirect	@ERn
3	Register indirect with displacement	@(d:16,ERn)/@(d:32,ERn)
4	Register indirect with post-increment	@ERn+
	Register indirect with pre-decrement	@-ERn
5	Absolute address	@aa:8/@aa:16/@aa:24/@aa:32
6	Immediate	#xx:8/#xx:16/#xx:32
7	Program-counter relative	@(d:8,PC)/@(d:16,PC)
8	Memory indirect	@@aa:8

### Table 1.4Addressing Modes

**1. Register Direct—Rn:** The register field of the instruction specifies an 8-, 16-, or 32-bit general register containing the operand. R0H to R7H and R0L to R7L can be specified as 8-bit registers. R0 to R7 and E0 to E7 can be specified as 16-bit registers. ER0 to ER7 can be specified as 32-bit registers.

**2. Register Indirect**—@**ERn:** The register field of the instruction code specifies an address register (ERn) which contains the address of the operand in memory. If the address is a program instruction address, the lower 24 bits are valid and the upper 8 bits are all assumed to be 0 (H'00).

**3. Register Indirect with Displacement**—@(**d:16, ERn**) or @(**d:32, ERn**): A 16-bit or 32-bit displacement contained in the instruction is added to an address register (ERn) specified by the register field of the instruction, and the sum gives the address of a memory operand. A 16-bit displacement is sign-extended when added.

# Renesas

### 4. Register Indirect with Post-Increment or Pre-Decrement—@ERn+ or @-ERn:

• Register indirect with post-increment—@ERn+

The register field of the instruction code specifies an address register (ERn) which contains the address of a memory operand. After the operand is accessed, 1, 2, or 4 is added to the address register contents and the sum is stored in the address register. The value added is 1 for byte access, 2 for word access, or 4 for longword access. For word or longword access, the register value should be even.

• Register indirect with pre-decrement—@-ERn

The value 1, 2, or 4 is subtracted from an address register (ERn) specified by the register field in the instruction code, and the result becomes the address of a memory operand. The result is also stored in the address register. The value subtracted is 1 for byte access, 2 for word access, or 4 for longword access. For word or longword access, the register value should be even.

**5.** Absolute Address—@aa:8, @aa:16, @aa:24, or @aa:32: The instruction code contains the absolute address of a memory operand. The absolute address may be 8 bits long (@aa:8), 16 bits long (@aa:16), 24 bits long (@aa:24), or 32 bits long (@aa:32).

To access data, the absolute address should be 8 bits (@aa:8), 16 bits (@aa:16), or 32 bits (@aa:32) long. For an 8-bit absolute address, the upper 24 bits are all assumed to be 1 (H'FFFFFF). For a 16-bit absolute address the upper 16 bits are a sign extension. A 32-bit absolute address can access the entire address space.

A 24-bit absolute address (@aa:24) indicates the address of a program instruction. The upper 8 bits are all assumed to be 0 (H'00).

Table 1.5 indicates the accessible absolute address ranges.

Table 1.5	Absolute Address Access Ranges
-----------	--------------------------------

Absolute Address		Normal Mode	Advanced Mode
Data address	8 bits (@aa:8)	H'FF00 to H'FFFF	H'FFFFFF00 to H'FFFFFFFF
	16 bits (@aa:16)	H'0000 to H'FFFF	H'00000000 to H'00007FFF, H'FFFF8000 to H'FFFFFFFF
	32 bits (@aa:32)		H'00000000 to H'FFFFFFFF
Program instruction address	24 bits (@aa:24)		H'00000000 to H'00FFFFF

For further details on the accessible range, refer to the relevant microcontroller hardware manual.

**6. Immediate**—**#xx:8**, **#xx:16**, **or #xx:32**: The instruction contains 8-bit (#xx:8), 16-bit (#xx:16), or 32-bit (#xx:32) immediate data as an operand.

The ADDS, SUBS, INC, and DEC instructions contain immediate data implicitly. Some bit manipulation instructions contain 3-bit immediate data in the instruction code, specifying a bit number. The TRAPA instruction contains 2-bit immediate data in its instruction code, specifying a vector address.

7. Program-Counter Relative—@(d:8, PC) or @(d:16, PC): This mode is used in the Bcc and BSR instructions. An 8-bit or 16-bit displacement contained in the instruction is sign-extended and added to the 24-bit PC contents to generate a branch address. Only the lower 24 bits of this branch address are valid; the upper 8 bits are all assumed to be 0 (H'00). The PC value to which the displacement is added is the address of the first byte of the next instruction, so the possible branching range is -126 to +128 bytes (-63 to +64 words) or -32766 to +32768 bytes (-16383 to +16384 words) from the branch instruction. The resulting value should be an even number.

**8. Memory Indirect**—@@**aa:8:** This mode can be used by the JMP and JSR instructions. The second byte of the instruction specifies a memory operand by an 8-bit absolute address. This memory operand contains a branch address. The upper bits of the absolute address are all assumed to be 0, so the address range is 0 to 255 (H'0000 to H'00FF in normal mode, H'00000000 to H'00000FF in advanced mode). In normal mode the memory operand is a word operand and the branch address is 16 bits long. In advanced mode the memory operand is a longword operand, the first byte of which is assumed to be all 0 (H'00).

Note that the first part of the address range is also the exception vector area. For further details refer to the relevant microcontroller hardware manual.



Figure 1.13 Branch Address Specification in Memory Indirect Mode

# Renesas

If an odd address is specified in word or longword memory access, or as a branch address, the least significant bit is regarded as 0, causing data to be accessed or an instruction code to be fetched at the address preceding the specified address. (For further information, see section 1.5.2, Memory Data Formats.)

# (2) Effective Address Calculation

Table 1.6 indicates how effective addresses are calculated in each addressing mode. In normal mode the upper 8 bits of the effective address are ignored in order to generate a 16-bit address.





#### Table 1.6 Effective Address Calculation



# Section 1 CPU

Rev. 4.00 Feb 24, 2006 page 38 of 322 REJ09B0139-0400





# Section 2 Instruction Descriptions

# 2.1 Tables and Symbols

This section explains how to read the tables in section 2.2, describing each instruction. Note that the descriptions of some instructions extend over more than one page.

[1] Mnemonic (Full Name)		[2] <b>Туре</b>
[3] Operation	[6] Condition Code	
[4] Assembly-Language Format	_	
[5] Operand Size	_	
[7] Description		
[8] Available Registers		
[9] Operand Format and Number of	States Required for Execution	on
[10] <b>Notes</b>		

- [1] Mnemonic (Full Name): Gives the full and mnemonic names of the instruction.
- [2] **Type:** Indicates the type of instruction.
- [3] Operation: Describes the instruction in symbolic notation. (See section 2.1.2, Operation.)
- [4] Assembly-Language Format: Indicates the assembly-language format of the instruction. (See section 2.1.1, Assembler Format.)
- [5] Operand Size: Indicates the available operand sizes.
- [6] **Condition Code:** Indicates the effect of instruction execution on the flag bits in the CCR. (See section 2.1.3, Condition Code.)
- [7] Description: Describes the operation of the instruction in detail.
- [8] Available Registers: Indicates which registers can be specified in the register field of the instruction.
- [9] Operand Format and Number of States Required for Execution: Shows the addressing modes and instruction format together with the number of states required for execution.
- [10] Notes: Gives notes concerning execution of the instruction.

# Renesas

# 2.1.1 Assembly-Language Format



The operand size is byte (B), word (W), or longword (L). Some instructions are restricted to a limited set of operand sizes.

The symbol <EA> indicates that two or more addressing modes can be used. The H8S/2600 CPU supports the eight addressing modes listed next. Effective address calculation is described in section 1.7, Addressing Modes and Effective Address Calculation.

Symbol	Addressing Mode
Rn	Register direct
@ERn	Register indirect
@(d:16, ERn)/@(d:32, ERn)	Register indirect with displacement (16-bit or 32-bit)
@ERn+/@-ERn	Register indirect with post-increment or pre-decrement
@aa:8/@aa:16/@aa:24/@aa:32	Absolute address (8-bit, 16-bit, 24-bit, or 32-bit)
#xx:8/#xx:16/#xx:32	Immediate (8-bit, 16-bit, or 32-bit)
@(d:8, PC)/@(d:16, PC)	Program-counter relative (8-bit or 16-bit)
@@aa:8	Memory indirect

The suffixes :8, :16, :24, and :32 may be omitted. In particular, if the :8, :16, :24, or :32 designation is omitted in an absolute address or displacement, the assembler will optimize the length according to the value range. For details, refer to the H8S, H8/300 Series cross assembler user's manual.

Note: ":2" and ":3" in "#xx (:2)" and "#xx (:3)" indicate the specifiable bit length. Do not include (:2) or (:3) in the assembler notation. Example: TRAPA #3

# 2.1.2 Operation

The symbols used in the operation descriptions are defined as follows.

Rd	General register (destination)*
Rs	General register (source)*
Rn	General register*
ERn	General register (32-bit register)
MAC	Multiply-accumulate register (32-bit register)
(EAd)	Destination operand
(EAs)	Source operand
EXR	Extended control register
CCR	Condition-code register
Ν	N (negative) flag in CCR
Z	Z (zero) flag in CCR
V	V (overflow) flag in CCR
С	C (carry) flag in CCR
PC	Program counter
SP	Stack pointer
#IMM	Immediate data
disp	Displacement
+	Add
-	Subtract
×	Multiply
÷	Divide
^	Logical AND
$\vee$	Logical OR
$\oplus$	Logical exclusive OR
$\rightarrow$	Transfer from the operand on the left to the operand on the right, or transition from the state on the left to the state on the right
7	Logical NOT (logical complement)
() < >	Contents of effective address of the operand
:8/:16/ :24/:32	8-, 16-, 24-, or 32-bit length
Note: *	General registers include 8-bit registers (R0H to R7H and R0L to R7L), 16-bit registers (R0 to R7 and E0 to E7), and 32-bit registers (ER0 to ER7).

### 2.1.3 Condition Code

The symbols used in the condition-code description are defined as follows.

Symbol	Meaning
\$	Changes according to the result of instruction execution
*	Undetermined (no guaranteed value)
0	Always cleared to 0
1	Always set to 1
_	Not affected by execution of the instruction
Δ	Varies depending on conditions; see the notes

For details on changes of the condition code, see section 2.8, Condition Code Modification.

# 2.1.4 Instruction Format

The symbols used in the instruction format descriptions are listed below.

Symbol	Meaning
IMM	Immediate data (2, 3, 8, 16, or 32 bits)
abs	Absolute address (8, 16, 24, or 32 bits)
disp	Displacement (8, 16, or 32 bits)
rs, rd, rn	Register field (4 bits). The symbols rs, rd, and rn correspond to operand symbols Rs, Rd, and Rn.
ers, erd, ern	Register field (3 bits). The symbols ers, erd, and ern correspond to operand symbols ERs, ERd, and ERn.



#### 2.1.5 Register Specification

Address Register Specification: When a general register is used as an address register [@ERn, @(d:16, ERn), @(d:32, ERn), @ERn+, or @-ERn], the register is specified by a 3-bit register field (ers or erd).

**Data Register Specification:** A general register can be used as a 32-bit, 16-bit, or 8-bit data register.

When used as a 32-bit register, it is specified by a 3-bit register field (ers, erd, or ern).

When used as a 16-bit register, it is specified by a 4-bit register field (rs, rd, or rn). The lower 3 bits specify the register number. The upper bit is set to 1 to specify an extended register (En) or cleared to 0 to specify a general register (Rn).

When used as an 8-bit register, it is specified by a 4-bit register field (rs, rd, or rn). The lower 3 bits specify the register number. The upper bit is set to 1 to specify a low register (RnL) or cleared to 0 to specify a high register (RnH). This is shown next.

	ddress Register 2-Bit Register 16-Bit Register			8-Bit Register		
Register Field	General Register Register Field		General Register	Register Field	General Register	
000	ER0	0000	R0	0000	R0H	
001	ER1	0001	R1	0001	R1H	
111	ER7	0111	R7	0111	R7H	
		1000	E0	1000	R0L	
		1001	E1	1001	R1L	
		1111	E7	1111	R7L	

# Renesas

# 2.1.6 Bit Data Access in Bit Manipulation Instructions

Bit data is accessed as the n-th bit (n = 0, 1, 2, 3, ..., 7) of a byte operand in a general register or memory. The bit number is given by 3-bit immediate data, or by the lower 3 bits of a general register value.

Example 1: To set bit 3 in R2H to 1



Example 2: To load bit 5 at address H'FFFF02 into the bit accumulator



BLD #5, @H'FFFF02

The operand size and addressing mode are as indicated for register or memory operand data.

# 2.2 Instruction Descriptions

The instructions are described starting in section 2.2.1.

### **2.2.1** (1) **ADD** (**B**)

### ADD (ADD Binary)

#### **Add Binary**

Operation	Condition Code					
$Rd + (EAs) \rightarrow Rd$	$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$					
Assembly-Language Format ADD.B <eas>, Rd</eas>	<ul><li>H: Set to 1 if there is a carry at bit 3; otherwise cleared to 0.</li><li>N: Set to 1 if the result is negative; otherwise cleared to 0.</li></ul>					
<b>Operand Size</b> Byte	<ul> <li>Z: Set to 1 if the result is zero; otherwise cleared to 0.</li> <li>V: Set to 1 if an overflow occurs; otherwise cleared to 0.</li> <li>C: Set to 1 if there is a carry at bit 7; otherwise cleared to 0.</li> </ul>					

# Description

This instruction adds the source operand to the contents of an 8-bit register Rd (destination operand) and stores the result in the 8-bit register Rd.

### **Available Registers**

Rd: R0L to R7L, R0H to R7H Rs: R0L to R7L, R0H to R7H

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands		Instruction Format					No. of
Mode	Millemonic	Operanus	1st byte		2nd	byte	3rd byte	4th byte	States
Immediate	ADD.B	#xx:8, Rd	8	rd	IMM				1
Register direct	ADD.B	Rs, Rd	0	8	rs	rd			1



### 2.2.1 (2) ADD (W)

### ADD (ADD Binary)

#### **Add Binary**

Operation	Condition Code						
$Rd + (EAs) \rightarrow Rd$	I UI H U N Z V C						
	$ - -  \updownarrow  -  \updownarrow   \updownarrow   \updownarrow   \updownarrow  $						
Assembly-Language Format ADD.W <eas>, Rd</eas>	<ul> <li>H: Set to 1 if there is a carry at bit 11; otherwise cleared to 0.</li> <li>N: Set to 1 if the result is negative; otherwise cleared to 0.</li> <li>7: Set to 1 if the result is zero; otherwise</li> </ul>						
<b>Operand Size</b> Word	<ul> <li>Z: Set to 1 if the result is zero; otherwise cleared to 0.</li> <li>V: Set to 1 if an overflow occurs; otherwise cleared to 0.</li> <li>C: Set to 1 if there is a carry at bit 15; otherwise cleared to 0.</li> </ul>						

# Description

This instruction adds the source operand to the contents of a 16-bit register Rd (destination operand) and stores the result in the 16-bit register Rd.

### **Available Registers**

Rd: R0 to R7, E0 to E7 Rs: R0 to R7, E0 to E7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands	Instruction Format					No. of	
Mode	Whenome	Operanus	1st byte		2nd byte		3rd byte	4th byte	States
Immediate	ADD.W	#xx:16, Rd	7	9	1	rd	IMM		2
Register direct	ADD.W	Rs, Rd	0	9	rs	rd			1

Notes

### 2.2.1 (3) ADD (L)

### ADD (ADD Binary)

#### **Add Binary**

Operation	Condition Code
$\text{ERd} + (\text{EAs}) \rightarrow \text{ERd}$	I UI H U N Z V C
	$\left -\right -\left \left \left \right -\right \left \left \left \right \right \right \left \left \left \right \right \right \right$
Assembly-Language Format ADD.L <eas>, ERd</eas>	<ul><li>H: Set to 1 if there is a carry at bit 27; otherwise cleared to 0.</li><li>N: Set to 1 if the result is negative; otherwise cleared to 0.</li></ul>
Operand Size Longword	<ul> <li>Z: Set to 1 if the result is zero; otherwise cleared to 0.</li> <li>V: Set to 1 if an overflow occurs; otherwise cleared to 0.</li> <li>C: Set to 1 if there is a carry at bit 31; otherwise cleared to 0.</li> </ul>

# Description

This instruction adds the source operand to the contents of a 32-bit register ERd (destination operand) and stores the result in the 32-bit register ERd.

### **Available Registers**

ERd: ER0 to ER7 ERs: ER0 to ER7

### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands		Instruction Format									
Mode	Millenionic	Operands	1st	byte	2nd byte	3rd byte	4th byte	5th byte	6th byte	States			
Immediate	ADD.L	#xx:32, ERd	7	А	1 0 erc		IN	1M		3			
Register direct	ADD.L	ERs, ERd	0	А	1 ers 0 erc					1			



#### 2.2.2 ADDS

#### ADDS (ADD with Sign extension)

### Operation

 $Rd + 1 \rightarrow ERd$  $Rd + 2 \rightarrow ERd$  $Rd + 4 \rightarrow ERd$ 

#### **Assembly-Language Format**

ADDS #1, ERd ADDS #2, ERd ADDS #4, ERd

#### **Operand Size**

Longword

### Condition Code



- H: Previous value remains unchanged.
- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Previous value remains unchanged.

### Description

This instruction adds the immediate value 1, 2, or 4 to the contents of a 32-bit register ERd (destination operand). Unlike the ADD instruction, it does not affect the condition code flags.

#### **Available Registers**

ERd: ER0 to ER7

### **Operand Format and Number of States Required for Execution**

Addressing Mode	Mnemonic	Operands			In	str	uctio	n Format		No. of
		Operanus	1st I	byte	2nd	l by	/te	3rd byte	4th byte	States
Register direct	ADDS	#1, ERd	0	В	0	0	erd			1
Register direct	ADDS	#2, ERd	0	В	8	0	erd			1
Register direct	ADDS	#4, ERd	0	В	9	0	erd			1

#### Notes

# RENESAS

# Add Binary Address Data

### 2.2.3 ADDX

### ADDX (ADD with eXtend carry)

# Add with Carry

Operation	Condition Code
$Rd + (EAs) + C \rightarrow Rd$	$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$
Assembly-Language Format ADDX <eas>, Rd</eas>	<ul><li>H: Set to 1 if there is a carry at bit 3; otherwise cleared to 0.</li><li>N: Set to 1 if the result is negative; otherwise cleared to 0.</li></ul>
<b>Operand Size</b> Byte	Z: Set to 1 if the result is zero; otherwise cleared to 0. V: Set to 1 if an overflow occurs; otherwise cleared to 0.
	C: Set to 1 if there is a carry at bit 7; otherwise cleared to 0.

# Description

This instruction adds the source operand and carry flag to the contents of an 8-bit register Rd (destination operand) and stores the result in the 8-bit register Rd.

### **Available Registers**

Rd: R0L to R7L, R0H to R7H Rs: R0L to R7L, R0H to R7H

### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands			No. of				
Mode	Millemonic	Operanus	1st byte		2nd byte		3rd byte	4th byte	States
Immediate	ADDX	#xx:8, Rd	9	rd	IMM				1
Register direct	ADDX	Rs, Rd	0	Е	rs	rd			1



Logical AND

# 2.2.4 (1) AND (B)

# AND (AND logical)

# Operation

 $Rd \land (EAs) \rightarrow Rd$ 

# Assembly-Language Format

AND.B <EAs>, Rd

# **Operand Size**

Byte

# Description

This instruction ANDs the source operand with the contents of an 8-bit register Rd (destination operand) and stores the result in the 8-bit register Rd.

# **Available Registers**

Rd: R0L to R7L, R0H to R7H Rs: R0L to R7L, R0H to R7H

# **Operand Format and Number of States Required for Execution**

Addressing	Addressing Mode	Mnemonic	Operands			Ins	tructio	on Format		No. of
Mode	winemonic	Operatius	1st byte		2nd byte		3rd byte	4th byte	States	
Immediate	AND.B	#xx:8, Rd	Е	rd	IMM				1	
Register direct	AND.B	Rs, Rd	1	6	rs	rd			1	

Cor	nditio	n Co	ode						
	Ι	UI	Н	U	Ν	Ζ	V	С	
	_	—		—	$\Leftrightarrow$	$\updownarrow$	0	—	
H:	Prev	ious	valu	e rem	ains	unch	nange	ed.	

- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Previous value remains unchanged.

#### 2.2.4 (2) AND (W)

#### AND (AND logical)

#### Logical AND

Operation	Condition Code
$\mathrm{Rd} \wedge (\mathrm{EAs}) \rightarrow \mathrm{Rd}$	$\begin{array}{c ccccccccccccccccccccccccccccccccccc$
Assembly-Language Format AND.W <eas>, Rd</eas>	H: Previous value remains unchanged. N: Set to 1 if the result is negative; otherwise cleared to 0.
	Z: Set to 1 if the result is zero; otherwise cleared to 0.
<b>Operand Size</b> Word	<ul><li>V: Always cleared to 0.</li><li>C: Previous value remains unchanged.</li></ul>

# Description

This instruction ANDs the source operand with the contents of a 16-bit register Rd (destination operand) and stores the result in the 16-bit register Rd.

#### **Available Registers**

Rd: R0 to R7, E0 to E7 Rs: R0 to R7, E0 to E7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands			Ins	tructio	on Format		No. of
Mode	Witemonic	Operands	1st byte		2nd byte		3rd byte	4th byte	States
Immediate	AND.W	#xx:16, Rd	7	9	6	rd	IMM		2
Register direct	AND.W	Rs, Rd	6	6	rs	rd			1



#### 2.2.4 (3) AND (L)

#### AND (AND logical)

#### Operation

 $\operatorname{ERd} \wedge (\operatorname{EAs}) \rightarrow \operatorname{ERd}$ 

#### **Assembly-Language Format**

AND.L <EAs>, ERd

#### **Operand Size**

Longword

# Description

This instruction ANDs the source operand with the contents of a 32-bit register ERd (destination operand) and stores the result in the 32-bit register ERd.

#### **Available Registers**

ERd: ER0 to ER7 ERs: ER0 to ER7

### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands		Instruction Format									
Mode	Witterfiorfic	Operations	1st byte		2nd byte		3rd k	3rd byte 4th byte		5th byte	6th byte	States	
Immediate	AND.L	#xx:32, ERd	7	А	6	0 erd			IM	IM		3	
Register direct	AND.L	ERs, ERd	0	1	F	0	6	6	0 ers 0 erd			2	

#### Notes

# RENESAS

#### Logical AND



# Z: Set to 1 if the result is zero; otherwise cleared to 0.

- V: Always cleared to 0.
- C: Previous value remains unchanged.

#### 2.2.5 (1) ANDC

#### ANDC (AND Control register)

#### Operation

 $CCR \land \#IMM \rightarrow CCR$ 

#### **Assembly-Language Format**

ANDC #xx:8, CCR

#### **Operand Size**

Byte

### **Condition Code**

Ι	UI	Н	U	Ν	Ζ	V	С
$\updownarrow$	$\updownarrow$	$\updownarrow$	$\Rightarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$

I: Stores the corresponding bit of the result.

UI: Stores the corresponding bit of the result.

H: Stores the corresponding bit of the result.

U: Stores the corresponding bit of the result.

N: Stores the corresponding bit of the result.

Z: Stores the corresponding bit of the result.

V: Stores the corresponding bit of the result.

C: Stores the corresponding bit of the result.

### Description

This instruction ANDs the contents of the condition-code register (CCR) with immediate data and stores the result in the condition-code register. No interrupt requests, including NMI, are accepted immediately after execution of this instruction.

### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands		Instruction Format					
Mode	Millenionic	Operando	1st byte		2nd byte	3rd byte	4th byte	States	
Immediate	ANDC	#xx:8, CCR	0 6		IMM			1	

Notes



# Logical AND with CCR
Logical AND with EXR

### 2.2.5 (2) ANDC

### ANDC (AND Control register)

### Operation

 $\mathrm{EXR} \land \#\mathrm{IMM} \to \mathrm{EXR}$ 

### Assembly-Language Format

ANDC #xx:8, EXR

### **Operand Size**

Byte

### Description

This instruction ANDs the contents of the extended control register (EXR) with immediate data and stores the result in the extended control register. No interrupt requests, including NMI, are accepted for three states after execution of this instruction.

### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands			Ins	tructio	on Forr	nat		No. of
Mode	Witemonic	Operanus	1st I	byte	2nd	byte	3rd	byte	4th byte	States
Immediate	ANDC	#xx:8, EXR	0	1	4	1	0	6	IMM	2

Notes

### Rev. 4.00 Feb 24, 2006 page 57 of 322 REJ09B0139-0400

# RENESAS

Ι	UI	Н	U	Ν	Ζ	V	С
	—	—			—		—

H: Previous value remains unchanged.

**Condition Code** 

- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Previous value remains unchanged.

### 2.2.6 BAND

### BAND (Bit AND)

### Operation

 $C \land (\langle bit No. \rangle of \langle EAd \rangle) \rightarrow C$ 

### **Assembly-Language Format**

BAND #xx:3, <EAd>

### **Bit Logical AND**

Ι	UI	Н	U	Ν	Ζ	V	С
_							$\updownarrow$

H: Previous value remains unchanged.

**Condition Code** 

- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Stores the result of the operation.

### **Operand Size**

Byte

### Description

This instruction ANDs a specified bit in the destination operand with the carry flag and stores the result in the carry flag. The bit number is specified by 3-bit immediate data. The destination operand contents remain unchanged.



### **Available Registers**

Rd: R0L to R7L, R0H to R7H ERd: ER0 to ER7

Minetronic BANDTet byte2nd byte3rd byte3rd byte5th byte6th byte7th byte8th byteBAND $\#xx:3$ , Rd76 $0$ :MMrd76 $0$ :MMrd76BAND $\#xx:3$ , @ERd7C $0$ :erd076 $0$ :MM076 $0$ :MM0BAND $\#xx:3$ , @aa:87Eabs76 $0$ :MM076 $0$ :MM0BAND $\#xx:3$ , @aa:166A10 $abs$ 76 $0$ :MM076 $0$ :MM0BAND $\#xx:3$ , @aa:326A30 $abs$ 76 $0$ :MM076 $0$ :MM0	Addressing	M							Instructio	Instruction Format				No. of
er         BAND         #xx:3, Rd         7         6         0iMi         rd         rd	Mode*	MINEMONIC	Operands	1st b	yte	2nd byte		yte	4th byte					States
BAND         #xx:3, @ERd         7         C         0 erd         0         7         6         0 MM         0           BAND         #xx:3, @aa:8         7         C         0 erd         0         7         6         0 MM         0           BAND         #xx:3, @aa:8         7         E         abs         7         6         0 MM         0           BAND         #xx:3, @aa:16         6         A         1         0         abs         7         6         0 MM         0           BAND         #xx:3, @aa:32         6         A         1         0         7         6         0 MM         0	Register direct	BAND	#xx:3, Rd	2	9	0 IMM rd								-
BAND         #xx:3, @aa:8         7         E         abs         7         6         0         MM         0         1 <th1< th=""> <th1< th="">         1</th1<></th1<>	Register indirect	BAND		~	U	0 erd 0	~	9	0 MWI 0					ю
BAND         #xx:3, @aa:16         6         A         1         0         abs         7         6         0iMi         0           BAND         #xx:3, @aa:32         6         A         3         0         abs         7         6         0iMi         0	Absolute address	BAND	#хх:3, @аа:8	~	ш	abs	7	9	0 MWI 0					e
BAND #xx:3, @aa:32 6 A 3 0 abs 7 6 0 MM 0	Absolute address	BAND	#xx:3, @aa:16		٨	1		ab	S	7 6	0 MWI 0			4
	Absolute address	BAND	#xx:3, @aa:32	و		0 3			at	Sc		7 6	0 MMI 0	

Notes

For the @aa:8/@aa:16 access range, refer to the relevant microcontroller hardware manual.

### **BAND (Bit AND)**

Section 2 Instruction Descriptions

#### 2.2.7 Bcc

### **Bcc (Branch conditionally)**

# ~

### **Conditional Branch**

Operation	Condition Code
If condition is true, then $PC + disp \rightarrow PC$ else next;	I UI H U N Z V C 
Assembly-Language Format B <u>cc</u> disp → Condition field	<ul> <li>H: Previous value remains unchanged.</li> <li>N: Previous value remains unchanged.</li> <li>Z: Previous value remains unchanged.</li> <li>V: Previous value remains unchanged.</li> <li>C: Previous value remains unchanged.</li> </ul>
Operand Size	6

## Description

If the condition specified in the condition field (cc) is true, a displacement is added to the program counter (PC) and execution branches to the resulting address. If the condition is false, the next instruction is executed. The PC value used in the address calculation is the starting address of the instruction immediately following the Bcc instruction. The displacement is a signed 8-bit or 16-bit value. The branch destination address can be located in the range from -126 to +128 bytes or -32766 to +32768 bytes from the Bcc instruction.

Mnemonic	Meaning	сс	Condition	Signed/Unsigned*
BRA (BT)	Always (true)	0000	True	
BRN (BF)	Never (false)	0001	False	
BHI	Hlgh	0010	$C \lor Z = 0$	X > Y (unsigned)
BLS	Low or Same	0011	C∨Z = 1	$X \leq Y$ (unsigned)
BCC (BHS)	Carry Clear (High or Same)	0100	C = 0	$X \ge Y$ (unsigned)
BCS (BLO)	Carry Set (LOw)	0101	C = 1	X < Y (unsigned)
BNE	Not Equal	0110	Z = 0	$X \neq Y$ (unsigned or signed)
BEQ	EQual	0111	Z = 1	X = Y (unsigned or signed)
BVC	oVerflow Clear	1000	V = 0	
BVS	oVerflow Set	1001	V = 1	
BPL	PLus	1010	N = 0	
BMI	MInus	1011	N = 1	
BGE	Greater or Equal	1100	N⊕V = 0	$X \ge Y$ (signed)
BLT	Less Than	1101	N⊕V = 1	X < Y (signed)
BGT	Greater Than	1110	Z∨(N⊕V) = 0	X > Y (signed)
BLE	Less or Equal	1111	$Z_{\vee}(N \oplus V) = 1$	$X \le Y$ (signed)

Note: \* If the immediately preceding instruction is a CMP instruction, X is the general register contents (destination operand) and Y is the source operand.

# Renesas

### **Bcc (Branch conditionally)**

### **Conditional Branch**

### **Operand Format and Number of States Required for Execution**

Addressing		0			Instr	uctio	on Format		No. of
Mode	Mnemonic	Operands	1st	byte	2nd by	yte	3rd byte	4th byte	States
Program-counter		d:8	4	0	disp	)			2
relative	BRA (BT)	d:16	5	8	0	0	di	sp	3
Program-counter	BRN (BF)	d:8	4	1	disp	)			2
relative	DRIN (DF)	d:16	5	8	1	0	di	sp	3
Program-counter	BHI	d:8	4	2	disp	)			2
relative	ып	d:16	5	8	2	0	di	sp	3
Program-counter	BLS	d:8	4	3	disp	)			2
relative	BLS	d:16	5	8	3	0	di	sp	3
Program-counter	Bcc (BHS)	d:8	4	4	disp	)			2
relative	всс (впо)	d:16	5	8	4	0	di	sp	3
Program-counter		d:8	4	5	disp	)			2
relative	BCS (BLO)	d:16	5	8	5	0	di	sp	3
Program-counter	BNE	d:8	4	6	disp	)			2
relative	DINE	d:16	5	8	6	0	di	sp	3
Program-counter	BEQ	d:8	4	7	disp	)			2
relative	DEQ	d:16	5	8	7	0	di	sp	3
Program-counter	BVC	d:8	4	8	disp	)			2
relative	BVC	d:16	5	8	8	0	di	sp	3
Program-counter	BVS	d:8	4	9	disp	(			2
relative	DV3	d:16	5	8	9	0	di	sp	3
Program-counter	BPL	d:8	4	А	disp	)			2
relative	DPL	d:16	5	8	A	0	di	sp	3
Program-counter	BMI	d:8	4	В	disp	)			2
relative	DIVII	d:16	5	8	В	0	di	sp	3
Program-counter	BGE	d:8	4	С	disp	)			2
relative	BGE	d:16	5	8	С	0	di	sp	3
Program-counter	BLT	d:8	4	D	disp	)			2
relative	DLI	d:16	5	8	D	0	di	sp	3
Program-counter	BGT	d:8	4	Е	disp	)			2
relative	DGI	d:16	5	8	E	0	di	sp	3
Program-counter	BLE	d:8	4	F	disp	)			2
relative	DLE	d:16	5	8	F	0	di	sp	3

### Notes

- 1. The branch destination address must be even.
- 2. In machine language BRA, BRN, BCC, and BCS are identical to BT, BF, BHS, and BLO, respectively.

### 2.2.8 BCLR

### BCLR (Bit CLeaR)

### Operation

 $0 \rightarrow (\langle \text{bit No.} \rangle \text{ of } \langle \text{EAd} \rangle)$ 

### **Assembly-Language Format**

BCLR #xx:3, <EAd> BCLR Rn, <EAd>

### **Condition Code**



- H: Previous value remains unchanged.
- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Previous value remains unchanged.

# Operand Size

Byte

### Description

This instruction clears a specified bit in the destination operand to 0. The bit number can be specified by 3-bit immediate data, or by the lower three bits of an 8-bit register Rn. The specified bit is not tested. The condition-code flags are not altered.



### **Available Registers**

Rd:R0L to R7L, R0H to R7HERd:ER0 to ER7Rn:R0L to R7L, R0H to R7H

Addressing	M							Instructic	Instruction Format				No. of
Mode*	MINEMONIC	Operands	1st byte	oyte	2nd byte	yte	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	States
Register direct	BCLR	#xx:3, Rd	2	5	0 IMM	P							-
Register indirect	BCLR	#xx:3, @ERd	7	٥	0 erd	0	7 2	0 MMI 0					4
Absolute address	BCLR	#xx:3, @aa:8	7	ш	abs		7 2	0 MMI 0					4
Absolute address	BCLR	#xx:3, @aa:16	9	A	-	ω	<u></u>	abs	7 2	0 WWI 0			5
Absolute address	BCLR	#xx:3, @aa:32	9	A	e	œ			abs		7 2	0 MMM 0	9
Register direct	BCLR	Rn, Rd	9	2	E	P							-
Register indirect	BCLR	Rn, @ERd	7	٥	0 erd	0	6	0 E					4
Absolute address	BCLR	Rn, @aa:8	7	ш	abs	s	6 2	0 E					4
Absolute address	BCLR	Rn, @aa:16	9	A	-	ω	σ.	abs	6 2	o E			5
Absolute address	BCLR	Rn, @aa:32	و	A	ю	œ		5	abs		6	0 L	و

RENESAS

# Notes

For the @aa:8/@aa:16 access range, refer to the relevant microcontroller hardware manual.

## BCLR (Bit CLeaR)

**Bit Clear** 

Rev. 4.00 Feb 24, 2006 page 63 of 322 REJ09B0139-0400

### 2.2.9 **BIAND**

### **BIAND (Bit Invert AND)**

### Operation

 $C \land [\neg (<\!\!\text{bit No.}\!\!> \!\text{of} <\!\!\text{EAd}\!\!>)] \to C$ 

### **Assembly-Language Format**

BIAND #xx:3, <EAd>

### **Condition Code**



- H: Previous value remains unchanged.
- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Stores the result of the operation.

### **Operand Size**

Byte

### Description

This instruction ANDs the inverse of a specified bit in the destination operand with the carry flag and stores the result in the carry flag. The bit number is specified by 3-bit immediate data. The destination operand contents remain unchanged.



### **Available Registers**

Rd: ROL to R7L, R0H to R7H ERd: ER0 to ER7

Addressing	Mnomonio	Onorande				Instructic	Instruction Format					No. of
Mode*			1st byte	2nd byte	3rd byte	4th byte	5th byte	6th byte	7th byte		8th byte	States
Register direct	BIAND	#xx:3, Rd	7 6	1 IMM								-
Register indirect	BIAND	#xx:3, @ERd 7 C 0 erd 0	7 C	0 erd 0	7 6	1 IMM 0						e
Absolute address	BIAND	BIAND #xx:3, @aa:8 7	7 E	abs	7 6	1 IMM 0						с
Absolute address	BIAND	BIAND #xx:3, @aa:16 6	9 9	-	, and the second	abs	7 6	1 IMM 0				4
Absolute address	BIAND	BIAND #xx:3, @aa:32 6	6 A	3		0 0	abs		7	- - -	6 1 IMM 0	5
Note: * The	addressin	Note: * The addressing mode is the addressing mode of the destination operand <ead></ead>	addressin	a mode of th	e destination	n onerand	< EAd>					

The addressing mode is the addressing mode of the destination operand <EAd> NOIE.

# Notes

For the @aa:8/@aa:16 access range, refer to the relevant microcontroller hardware manual.

### **BIAND (Bit Invert AND)**

Section 2 Instruction Descriptions

### 2.2.10 BILD

### BILD (Bit Invert LoaD)

### Operation

 $\neg (\langle bit No. \rangle of \langle EAd \rangle) \rightarrow C$ 

### **Assembly-Language Format**

BILD #xx:3, <EAd>

### **Condition Code**

Ι	UI	Η	U	Ν	Ζ	V	С
—							$\updownarrow$

- H: Previous value remains unchanged.
- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Loaded with the inverse of the specified bit.

### **Operand Size**

Byte

### Description

This instruction loads the inverse of a specified bit from the destination operand into the carry flag. The bit number is specified by 3-bit immediate data. The destination operand contents remain unchanged.



### **Available Registers**

Rd: R0L to R7L, R0H to R7H ERd: ER0 to ER7

Mode*MittionOptimized1st byte2nd byte2nd byte3rd byte4th byte5th byte6th byte7th byte8th byte8th byte8th statesRegisterBILD $\pmxx:3. \text{ Rd}$ 771 \mmode	Addressing	Macmonio	0000000							Instruction Format	n Form	lat				No. of
BILD         #xx:3, @ERd         7         7         1 iMM         rd         M <thm< th=""><th>Mode*</th><th></th><th>Operations</th><th>1st b</th><th>yte</th><th>2nd by</th><th>vte</th><th>3rd by</th><th>/te</th><th>4th byte</th><th></th><th>yte</th><th>6th byte</th><th></th><th></th><th>States</th></thm<>	Mode*		Operations	1st b	yte	2nd by	vte	3rd by	/te	4th byte		yte	6th byte			States
BILD         #xx:3, @ERd         7         C         0         7         7         1 iMM         0         7         1 iMM         0           BILD         #xx:3, @aa:8         7         E         abs         7         7         1 iMM         0         7         7         7         1 iMM         0         7         7         7         1 iMM         0         7         7         1 iMM         0         7         7         1 iMM         0         7         7         7         1 iMM         0         7         7         7         1 iMM         0         7         7         1 iMM         0         7         7         1 iMM         0         7         7         7         7         1 iMM         0         7	Register direct	BILD	#xx:3, Rd	2	7	1 IMM	P									-
BILD       #xx:3, @aa:8       7       E       abs       7       7       1 iMM       0       1         BILD       #xx:3, @aa:16       6       A       1       0       abs       7       7       1 iMM       0         BILD       #xx:3, @aa:32       6       A       3       0       abs       7       7       1 iMM       0	Register indirect	BILD	#xx:3, @ERd	7	U	0 erd	0	7	~	1 IMM 0						ю
BILD         #xx:3, @aa:16         6         A         1         0         abs         7         7         1 iMM         0           BILD         #xx:3, @aa:32         6         A         3         0         abs         7         7         1 iMM         0	Absolute address	BILD	#xx:3, @aa:8	2	ш	abs		2	~	1 IMM 0						e
BILD         #xx:3, @aa:32         6         A         3         0         abs         7         7         1 iMM         0	Absolute address	BILD	#xx:3, @aa:16		A	-	0		ab	S	7	~	0 MMI I			4
	Absolute address	BILD	#xx:3, @aa:32	9	A		0			at	sc			7 7	1 IMM 0	5

# Notes

For the @aa:8/@aa:16 access range, refer to the relevant microcontroller hardware manual.

### **BILD (Bit Invert LoaD)**

### 2.2.11 BIOR

### **BIOR (Bit Invert inclusive OR)**

### Operation

 $\mathrm{C} \lor [\neg (<\!\! \mathrm{bit} \; \mathrm{No.}\!\! > \!\mathrm{of} <\!\! \mathrm{EAd}\!\! >)] \to \mathrm{C}$ 

### **Assembly-Language Format**

BIOR #xx:3, <EAd>

### **Condition Code**



- H: Previous value remains unchanged.
- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Stores the result of the operation.

### **Operand Size**

Byte

### Description

This instruction ORs the inverse of a specified bit in the destination operand with the carry flag and stores the result in the carry flag. The bit number is specified by 3-bit immediate data. The destination operand contents remain unchanged.



### **Available Registers**

Rd: R0L to R7L, R0H to R7H ERd: ER0 to ER7

# RENESAS

## **Bit Logical OR**

Addressing	Mnomonio	Onorande						Instructic	Instruction Format				No. of
Mode*			1st byte	6	nd by	te	1st byte 2nd byte 3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	States
Register direct	BIOR	#xx:3, Rd	7 4		1 IMM	p							-
Register indirect	BIOR	#xx:3, @ERd 7 C 0 erd 0	7	0	erd		7 4	7 4 1 IMM 0					з
Absolute address	BIOR	#xx:3, @aa:8 7	7 E		abs		7 4	4 1 IMM 0					3
Absolute address	BIOR	#xx:3, @aa:16 6 A	9			0	, D	abs	7 4	4 1 IMM 0			4
Absolute address	BIOR	#xx:3, @aa:32 6	6 A		 m	0		ō	abs		7 4	4 1 IMM 0	5
Noto: * Th		Note: * The eddressing mode is the eddressing mode of the destination wave [204]	iodarooo	200	0000	0 q +  +	dootioot		10 44				

Note: \* The addressing mode is the addressing mode of the destination operand <EAd>.

# Notes

For the @aa:8/@aa:16 access range, refer to the relevant microcontroller hardware manual.

### **BIOR (Bit Invert inclusive OR)**

Section 2 Instruction Descriptions

### 2.2.12 BIST

### **BIST (Bit Invert STore)**

### Operation

 $\neg C \rightarrow (\langle bit No. \rangle of \langle EAd \rangle)$ 

### **Assembly-Language Format**

BIST #xx:3, <EAd>

### **Condition Code**



- H: Previous value remains unchanged.
- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Previous value remains unchanged.

### **Operand Size**

Byte

### Description

This instruction stores the inverse of the carry flag in a specified bit location in the destination operand. The bit number is specified by 3-bit immediate data. Other bits in the destination operand remain unchanged.



### **Available Registers**

Rd: R0L to R7L, R0H to R7H ERd: ER0 to ER7

Addressing	Mnomonio	Macmonic Onorande					Instruction Format	on Format				No. of
Mode*		Operations	1st byte	<u> </u>	2nd byte	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	States
Register direct	BIST	#xx:3, Rd	9	7 1	1 IMM							-
Register indirect	BIST	#xx:3, @ERd 7 D 0 erd 0	7	0	) erd 0	6 7	1 IMM 0					4
Absolute address	BIST	#xx:3, @aa:8 7	7 F	ш	abs	6 7	7 1 IMM 0					4
Absolute address	BIST	#xx:3, @aa:16 6		۲	1 8	<del>.</del>	abs	6 7	1 IMM 0			5
Absolute address	BIST	#xx:3, @aa:32 6		A	3		at	abs		6 7	7 1 IMM 0	9
Note: * The	e addressin	Note: * The addressing mode is the addressing mode of the destination operand < EAd>	address	ina r	mode of th	e destinatio	n operand	<ead>.</ead>				

2 D D 2 υ Billoo υ ס aud 0 VOIE.

# Notes

For the @aa:8/@aa:16 access range, refer to the relevant microcontroller hardware manual.

### **BIST (Bit Invert STore)**

### 2.2.13 BIXOR

### BIXOR (Bit Invert eXclusive OR)

### Operation

 $C \oplus [\neg (<\!bit \text{ No.}\!> of <\!\!EAd\!>)] \rightarrow C$ 

### **Assembly-Language Format**

BIXOR #xx:3, <EAd>

### **Bit Exclusive Logical OR**

### **Condition Code**



- H: Previous value remains unchanged.
- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Stores the result of the operation.

### **Operand Size**

Byte

### Description

This instruction exclusively ORs the inverse of a specified bit in the destination operand with the carry flag and stores the result in the carry flag. The bit number is specified by 3-bit immediate data. The destination operand contents remain unchanged.



### **Available Registers**

Rd: R0L to R7L, R0H to R7H ERd: ER0 to ER7

Addressing	Mnomonio	Onorande				Instruction Format	on Format				No. of
Mode*			1st byte	2nd byte	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	States
Register direct	BIXOR	#xx:3, Rd	7 5	1 IMM rd							-
Register indirect	BIXOR	#xx:3, @ERd	7 C	7 C 0 erd 0	7 5	1 IMM 0					з
Absolute address	BIXOR	BIXOR #xx:3, @aa:8 7	7 E	abs	7 5	5 1 IMM 0					ю
Absolute address	BIXOR	BIXOR #xx:3, @aa:16 6	9 9	1 0	Ø	abs	7 5	5 1 IMM 0			4
Absolute address	BIXOR	BIXOR #xx:3, @aa:32 6	6 A	3 0		at	abs		7 5	1 IMM 0	5
Noto: * Th	oddroccin	Noto: * The addressing mode is the addressing mode of the destination presend /EAd-		t mode of th			7071				

\* The addressing mode is the addressing mode of the destination operand <EAd>. Note:

# Notes

For the @aa:8/@aa:16 access range, refer to the relevant microcontroller hardware manual.

### BIXOR (Bit Invert eXclusive OR)

Section 2 Instruction Descriptions

#### 2.2.14 BLD

### BLD (Bit LoaD)

### C

### A

### **Bit Load**

Operation	Condition Code
$(\langle Bit No. \rangle \text{ of } \langle EAd \rangle) \rightarrow C$	I UI H U N Z V C — — — — — — — ↓
Assembly-Language Format BLD #xx:3, <ead></ead>	<ul> <li>H: Previous value remains unchanged.</li> <li>N: Previous value remains unchanged.</li> <li>Z: Previous value remains unchanged.</li> <li>V: Previous value remains unchanged.</li> <li>C: Loaded from the specified bit.</li> </ul>
Operand Size	
Byte	

### Description

This instruction loads a specified bit from the destination operand into the carry flag. The bit number is specified by 3-bit immediate data. The destination operand contents remain unchanged.



### **Available Registers**

Rd: R0L to R7L, R0H to R7H ERd: ER0 to ER7



Addressing	Mnomonio	Macmonio Onorrande				Instructic	Instruction Format				No. of
Mode*		Operations	1st byte	2nd byte	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	States
Register direct	BLD	#xx:3, Rd	7 7	7 0 IMM rd							-
Register indirect	BLD	#xx:3, @ERd 7 C 0 erd 0 7 7 0 1MM 0	7 C	0 erd 0	7 7	0 MWI 0					з
Absolute address	BLD	#xx:3, @aa:8 7	7 E	abs	7 7	7 0 IMM 0					3
Absolute address	BLD	#xx:3, @aa:16 6	9 9	-		abs	7 7	7 0 IMM 0			4
Absolute address	BLD	#xx:3, @aa:32 6	6 A	0 %		, a	abs		7 7	7 7 0:IMM 0	5
Note: * Th∈	e addressin	Note: * The addressing mode is the addressing mode of the destination operand <ead></ead>	addressir	ng mode of th	he destinati	on operand	<ead>.</ead>				

Notes

For the @aa:8/@aa:16 access range, refer to the relevant microcontroller hardware manual.

### BLD (Bit LoaD)

### 2.2.15 BNOT

### **BNOT (Bit NOT)**

### Operation

 $\neg$  (<bit No.> of <EAd>)  $\rightarrow$  (bit No. of <EAd>)

### **Assembly-Language Format**

BNOT #xx:3, <EAd> BNOT Rn, <EAd>

### **Operand Size**

Byte

# Description

This instruction inverts a specified bit in the destination operand. The bit number is specified by 3bit immediate data or by the lower 3 bits of an 8-bit register Rn. The specified bit is not tested. The condition code remains unchanged.

# Available Registers

Rd:R0L to R7L, R0H to R7HERd:ER0 to ER7Rn:R0L to R7L, R0H to R7H

### **Condition Code**

Ι	UI	Н	U	Ν	Ζ	V	С
—		_		—	_		

- H: Previous value remains unchanged.
- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Previous value remains unchanged.





### **Bit NOT**

Addressing						Instructic	Instruction Format				No. of
Mode*	Minemonic	Operands	1st byte	e 2nd byte	ard byte	4th byte	5th byte	6th byte	7th byte	8th byte	States
Register direct	BNOT	#xx:3, Rd	7	1 0 IMM							-
Register indirect	BNOT	#xx:3, @ERd	7	D 0 erd 0	7 1	0 IMM					4
Absolute address	BNOT	#xx:3, @aa:8	7	Fabs	7 1	0 MMI 0					4
Absolute address	BNOT	#xx:3, @aa:16	9	A 1 8		abs	7 1	0 MMM 0			5
Absolute address	BNOT	#xx:3, @aa:32	9	A 3		a	abs		7 1	0 MMI:0	9
Register direct	BNOT	Rn, Rd	9	L L							-
Register indirect	BNOT	Rn, @ERd	2	D 0 erd 0	6	0 L					4
Absolute address	BNOT	Rn, @aa:8	7	Fabs	6 1	0 L					4
Absolute address	BNOT	Rn, @aa:16	9	A 1 8		abs	6 1	rn 0			5
Absolute address	BNOT	Rn, @aa:32	9	A 3		<del>.</del>	abs		6	0 L	9

RENESAS

# Notes

For the @aa:8/@aa:16 access range, refer to the relevant microcontroller hardware manual.

### **BNOT (Bit NOT)**

**Bit NOT** 

Rev. 4.00 Feb 24, 2006 page 77 of 322 REJ09B0139-0400

### 2.2.16 BOR

### BOR (Bit inclusive OR)

### Operation

 $\mathrm{C} \lor (<\!\! \mathrm{bit} \: \mathrm{No.}\!\! > \! \mathrm{of} <\!\! \mathrm{EAd}\!\! >) \to \mathrm{C}$ 

### **Assembly-Language Format**

BOR #xx:3, <EAd>

### **Condition Code**



- H: Previous value remains unchanged.
- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Stores the result of the operation.

### **Operand Size**

Byte

### Description

This instruction ORs a specified bit in the destination operand with the carry flag and stores the result in the carry flag. The bit number is specified by 3-bit immediate data. The destination operand contents remain unchanged.



### **Available Registers**

Rd: ROL to R7L, R0H to R7H ERd: ER0 to ER7

# RENESAS

### **Bit Logical OR**

Addressing	Mnomonio	Onorande						Instructio	Instruction Format				No. of
Mode*		Operations	1st by	te	1st byte 2nd byte		3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	States
Register direct	BOR	#xx:3, Rd	2	4	4 0 IMM rd	P							-
Register indirect	BOR	#xx:3, @ERd 7 C 0 erd 0	~	0	) erd	0	7 4	7 4 0 MM 0					з
Absolute address	BOR	#xx:3, @aa:8 7		ш	abs		7 4	4 0 IMM 0					з
Absolute address	BOR	#xx:3, @aa:16 6		۲	-	0	ō	abs	7 4	4 0:IMM 0			4
Absolute address	BOR	#xx:3, @aa:32 6	9	A	e	0		at	abs		7 4	7 4 0 MM 0	5
Note: * Th∈	e addressin	Note: * The addressing mode is the addressing mode of the destination operand <ead></ead>	address	sing 1	mode c	of the	e destinatio	on operand	<ead>.</ead>				

Notes

For the @aa:8/@aa:16 access range, refer to the relevant microcontroller hardware manual.

### BOR (Bit inclusive OR)

### 2.2.17 BSET

### **BSET (Bit SET)**

### Operation

 $1 \rightarrow (\langle bit No. \rangle of \langle EAd \rangle)$ 

### **Assembly-Language Format**

BSET #xx:3, <EAd> BSET Rn, <EAd>

### **Operand Size**

Byte

### Description

This instruction sets a specified bit in the destination operand to 1. The bit number can be specified by 3-bit immediate data, or by the lower three bits of an 8-bit register Rn. The specified bit is not tested. The condition code flags are not altered.



### **Available Registers**

Rd:R0L to R7L, R0H to R7HERd:ER0 to ER7Rn:R0L to R7L, R0H to R7H

## RENESAS



H: Previous value remains unchanged.

**Condition Code** 

- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Previous value remains unchanged.

1												No. of
Mode*		Operations	1st byte	yte	2nd byte	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	States
Register direct	BSET	#xx:3, Rd	2	0	0 IMM rd							-
Register indirect	BSET	#xx:3, @ERd	2	۵	0 erd 0	7 0	0 MMI 0					4
Absolute address	BSET	#xx:3, @aa:8	2	ш	abs	7 0	0 MMM 0					4
Absolute address	BSET	#xx:3, @aa:16	9	A	1 8	ធ	abs	7 0	0 MMI 0			5
Absolute address	BSET	#xx:3, @aa:32	9	A	8 8		at	abs		7 0	0 IMM	9
Register direct	BSET	Rn, Rd	9	0	r L							-
Register indirect	BSET	Rn, @ERd	2	۵	0 erd 0	9	0 L					4
Absolute address	BSET	Rn, @aa:8	7	ш	abs	6 0	0 L					4
Absolute address	BSET	Rn, @aa:16	9	A	1 8	al	abs	6 0	0 L			5
Absolute address	BSET	Rn, @aa:32	و	A	8 3		at	abs		0 9	0 E	9

RENESAS

# Notes

For the @aa:8/@aa:16 access range, refer to the relevant microcontroller hardware manual.

### **BSET (Bit SET)**

**Bit Set** 

Rev. 4.00 Feb 24, 2006 page 81 of 322 REJ09B0139-0400

### 2.2.18 BSR

### **BSR (Branch to SubRoutine)**

### Operation

 $PC \rightarrow @-SP$  $PC + disp \rightarrow PC$ 

### **Assembly-Language Format**

BSR disp

### **Operand Size**

### **Condition Code**



- H: Previous value remains unchanged.
- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Previous value remains unchanged.

### Description

This instruction branches to a subroutine at a specified address. It pushes the program counter (PC) value onto the stack as a restart address, then adds a specified displacement to the PC value and branches to the resulting address. The PC value pushed onto the stack is the address of the instruction following the BSR instruction. The displacement is a signed 8-bit or 16-bit value, so the possible branching range is -126 to +128 bytes or -32766 to +32768 bytes from the address of the BSR instruction.

### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands			Instr	uctio	n Format		No. of	fStates
Mode	Milenonie	operando	1st	byte	2nd	byte	3rd byte	4th byte	Normal	Advanced
Program-counter	BSR	d:8	5	5	di	sp			3	4
relative	DOIN	d:16	5	С	0	0	di	sp	4	5

# RENESAS

### **Branch to Subroutine**

### **BSR (Branch to SubRoutine)**

### **Branch to Subroutine**

### Notes

The stack structure differs between normal mode and advanced mode. In normal mode only the lower 16 bits of the program counter are pushed onto the stack.

Ensure that the branch destination address is even.



### 2.2.19 BST

### BST (Bit STore)

### Operation

 $C \rightarrow (\langle bit No. \rangle of \langle EAd \rangle)$ 

### **Assembly-Language Format**

BST #xx:3, <EAd>

#### **Bit Store**

Cor	nditio	on Co	ode						
	Ι	UI	Н	U	Ν	Ζ	V	С	
	_		—					—	
		vious vious					U		

Z: Previous value remains unchanged.

V: Previous value remains unchanged.

C: Previous value remains unchanged.

### **Operand Size**

Byte

### Description

This instruction stores the carry flag in a specified bit location in the destination operand. The bit number is specified by 3-bit immediate data.



### **Available Registers**

Rd: ROL to R7L, R0H to R7H ERd: ER0 to ER7



Addressing	Mnomonio	Onorande					Instruction Format	n Format				No. of
Mode*		Operations	1st byte		2nd byte	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	States
Register direct	BST	#xx:3, Rd	6 7	0	7 0 IMM rd							-
Register indirect	BST	#xx:3, @ERd 7 D 0 erd 0	2	0	erd 0	9	7 0 IMM 0					4
Absolute address	BST	#xx:3, @aa:8 7	7 F		abs	6 7	7 0 MM 0					4
Absolute address	BST	#xx:3, @aa:16 6	9 9		- 0		abs	6 7	7 0 IMM 0			5
Absolute address	BST	#xx:3, @aa:32 6	6 A		8 9		at	abs		6 7	7 0 IMM 0	9
Note: * The	eddressin (	Note: * The addressing mode is the addressing mode of the destination operand <ead></ead>	addressi	ing m	ode of ti	he destinati	ion operand	<ead>.</ead>				

Notes

For the @aa:8/@aa:16 access range, refer to the relevant microcontroller hardware manual.

### **BST (Bit STore)**

### 2.2.20 BTST

### BTST (Bit TeST)

### Operation

 $\neg (\langle Bit No. \rangle of \langle EAd \rangle) \rightarrow Z$ 

### Assembly-Language Format

BTST #xx:3, <EAd> BTST Rn, <EAd>

### **Operand Size**

Byte

### **Condition Code**

Ι	UI	Η	U	Ν	Ζ	V	С
—					$\updownarrow$		

- H: Previous value remains unchanged.
- N: Previous value remains unchanged.
- Z: Set to 1 if the specified bit is zero; otherwise cleared to 0.
- V: Previous value remains unchanged.
- C: Previous value remains unchanged.

### Description

This instruction tests a specified bit in the destination operand and sets or clears the zero flag according to the result. The bit number can be specified by 3-bit immediate data, or by the lower three bits of an 8-bit register Rn. The destination operand contents remain unchanged.



### **Available Registers**

Rd: R0L to R7L, R0H to R7H ERd: ER0 to ER7 Rn: R0L to R7L, R0H to R7H

ddressing	Manada						Instructic	Instruction Format				No. of
Mode*	Minemonic	Operands	1st byte	oyte	2nd byte	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	States
Register direct	BTST	#xx:3, Rd	~	ю	0 IMM rd							-
Register indirect	BTST	#xx:3, @ERd	~	ပ	0 erd 0	7 3	0 MMM 0					с
Absolute address	BTST	#xx:3, @aa:8	7	ш	abs	7 3	0 MMI 0					ю
Absolute address	BTST	#xx:3, @aa:16	9	A	1	<u></u>	abs	7 3	0 MMI 0			4
Absolute address	BTST	#xx:3, @aa:32	9	A	3		at	abs		7 3	0 MMI 0	5
Register direct	BTST	Rn, Rd	9	ю	E							-
Register indirect	BTST	Rn, @ERd	2	ပ	0 erd 0	9	u u					ю
Absolute address	BTST	Rn, @aa:8	2	ш	abs	9	rn 0					ю
Absolute address	BTST	Rn, @aa:16	9	A	1	<u>a</u>	abs	9 9	0 5			4
Absolute address	BTST	Rn, @aa:32	ە	A	0 സ		at	abs		я 9	0 L	2

RENESAS

Notes

For the @aa:8/@aa:16 access range, refer to the relevant microcontroller hardware manual.

### BTST (Bit TeST)

**Bit Test** 

Rev. 4.00 Feb 24, 2006 page 87 of 322 REJ09B0139-0400

### 2.2.21 BXOR

### BXOR (Bit eXclusive OR)

### Operation

 $C \oplus (<\!\!\text{bit No.}\!\!> \!\text{of} <\!\!\!\text{EAd}\!\!>) \rightarrow C$ 

### **Assembly-Language Format**

BXOR #xx:3, <EAd>

### **Bit Exclusive Logical OR**

### **Condition Code**



- H: Previous value remains unchanged.
- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Stores the result of the operation.

### **Operand Size**

Byte

### Description

This instruction exclusively ORs a specified bit in the destination operand with the carry flag and stores the result in the carry flag. The bit number is specified by 3-bit immediate data. The destination operand contents remain unchanged.



### **Available Registers**

Rd: R0L to R7L, R0H to R7H ERd: ER0 to ER7

Addressing	Mnomonio	Onorande				Instructic	Instruction Format				No. of
Mode*			1st byte	2nd byte	ard byte	4th byte	5th byte	6th byte	7th byte	8th byte	States
Register direct	BXOR	#xx:3, Rd	7 5	5 0 IMM rd							-
Register indirect	BXOR	#xx:3, @ERd	7 C	7 C 0 erd 0	7 5	0 WWI					с
Absolute address	BXOR	#xx:3, @aa:8	7 E	abs	7 5	5 0 IMM 0					3
Absolute address	BXOR	BXOR #xx:3, @aa:16 6	6 A	- 0		abs	7 5	0 MMM 0			4
Absolute address	BXOR	BXOR #xx:3, @aa:32 6	9 9	3		9 9	abs		7 5	5 0 IMM 0	5
Note: * The	e addressin	Note: * The addressing mode is the addressing mode of the destination operand <fad></fad>	addressin	a mode of	the destination	on operand	<ead>.</ead>				

# Notes

For the @aa:8/@aa:16 access range, refer to the relevant microcontroller hardware manual.

### **BXOR (Bit eXclusive OR)**

Section 2 Instruction Descriptions

### 2.2.22 CLRMAC

### CLRMAC (CLeaR MAC register)

### Operation

 $0 \rightarrow MACH, MACL$ 

### **Assembly-Language Format**

CLRMAC

### Initialize Multiply-Accumulate Register

### **Condition Code**



- H: Previous value remains unchanged.
- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Previous value remains unchanged.

### **Operand Size**

### Description

This instruction simultaneously clears registers MACH and MACL.

It is supported only by the H8S/2600 CPU.

### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands			Ins	tructio	on Format		No. of
Mode	Willemonic	Operanus	1st	byte	2nd	byte	3rd byte	4th byte	States
—	CLRMAC	—	0	1	А	0			2*

Note: \* A maximum of three additional states are required for execution of this instruction within three states after execution of a MAC instruction. For example, if there is a one-state instruction (such as NOP) between the MAC instruction and this instruction, this instruction will be two states longer.

The number of states may differ depending on the product. For details, refer to the relevant microcontroller hardware manual of the product in question.

### Notes

Execution of this instruction also clears the overflow flag in the multiplier to 0.



### 2.2.23 (1) CMP (B)

### CMP (CoMPare)

### Compare

Operation	Condition Code
Rd – (EAs), set/clear CCR	$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$
Assembly-Language Format CMP.B <eas>, Rd</eas>	<ul> <li>H: Set to 1 if there is a borrow at bit 3; otherwise cleared to 0.</li> <li>N: Set to 1 if the result is negative; otherwise cleared to 0.</li> <li>Z: Set to 1 if the result is zero; otherwise</li> </ul>
<b>Operand Size</b> Byte	<ul><li>cleared to 0.</li><li>V: Set to 1 if an overflow occurs; otherwise cleared to 0.</li><li>C: Set to 1 if there is a borrow at bit 7; otherwise cleared to 0.</li></ul>

### Description

This instruction subtracts the source operand from the contents of an 8-bit register Rd (destination operand) and sets or clears the condition code bits according to the result. The contents of the 8-bit register Rd remain unchanged.

### **Available Registers**

Rd: R0L to R7L, R0H to R7H Rs: R0L to R7L, R0H to R7H

### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands			Ins	tructio	on Format		No. of
Mode	Witterfiorfic	Operanus	1st I	byte	2nd	byte	3rd byte	4th byte	States
Immediate	CMP.B	#xx:8, Rd	А	rd	IM	IM			1
Register direct	CMP.B	Rs, Rd	1	С	rs	rd			1

### Notes

### 2.2.23 (2) CMP (W)

### CMP (CoMPare)

Operation	Condition Code
Rd – (EAs), set/clear CCR	$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$
Assembly-Language Format CMP.W <eas>, Rd</eas>	<ul> <li>H: Set to 1 if there is a borrow at bit 11; otherwise cleared to 0.</li> <li>N: Set to 1 if the result is negative; otherwise cleared to 0.</li> <li>Z: Set to 1 if the result is zero; otherwise</li> </ul>
<b>Operand Size</b> Word	<ul><li>cleared to 0.</li><li>V: Set to 1 if an overflow occurs; otherwise cleared to 0.</li><li>C: Set to 1 if there is a borrow at bit 15; otherwise cleared to 0.</li></ul>

### Description

This instruction subtracts the source operand from the contents of a 16-bit register Rd (destination operand) and sets or clears the condition code bits according to the result. The contents of the 16-bit register Rd remain unchanged.

### **Available Registers**

Rd: R0 to R7, E0 to E7 Rs: R0 to R7, E0 to E7

### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands			Ins	tructio	on Format		No. of
Mode	Millemonic	Operanus	1st	byte	2nd	byte	3rd byte	4th byte	States
Immediate	CMP.W	#xx:16, Rd	7	9	2	rd	IM	М	2
Register direct	CMP.W	Rs, Rd	1	D	rs	rd			1

### Notes


#### 2.2.23 (3) CMP (L)

#### CMP (CoMPare)

#### Compare

Operation	Condition Code
ERd – (EAs), set/clear CCR	$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$
Assembly-Language Format         CMP.L <eas>, ERd         Operand Size         Longword</eas>	<ul> <li>H: Set to 1 if there is a borrow at bit 27; otherwise cleared to 0.</li> <li>N: Set to 1 if the result is negative; otherwise cleared to 0.</li> <li>Z: Set to 1 if the result is zero; otherwise cleared to 0.</li> <li>V: Set to 1 if an overflow occurs; otherwise cleared to 0.</li> <li>C: Set to 1 if there is a borrow at bit 31; otherwise cleared to 0.</li> </ul>

### Description

This instruction subtracts the source operand from the contents of a 32-bit register ERd (destination operand) and sets or clears the condition code bits according to the result. The contents of the 32-bit register ERd remain unchanged.

#### **Available Registers**

ERd: ER0 to ER7 ERs: ER0 to ER7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands				Instructio	n Format			No. of
Mode	Milenionic	Operations	1st	byte	2nd byte	3rd byte	4th byte	5th byte	6th byte	States
Immediate	CMP.L	#xx:32, ERd	7	А	2 0 erd		IN	1M		3
Register direct	CMP.L	ERs, ERd	1	F	1 ers 0 erd					1

#### Notes

#### 2.2.24 DAA

#### DAA (Decimal Adjust Add)

#### **Decimal Adjust**

Operation	Condition Code
Rd (decimal adjust) $\rightarrow$ Rd	$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$
Assembly-Language Format DAA Rd	<ul> <li>H: Undetermined (no guaranteed value).</li> <li>N: Set to 1 if the adjusted result is negative; otherwise cleared to 0.</li> <li>Z: Set to 1 if the adjusted result is zero; otherwise cleared to 0.</li> </ul>
<b>Operand Size</b> Byte	<ul><li>V: Undetermined (no guaranteed value).</li><li>C: Set to 1 if there is a carry at bit 7; otherwise left unchanged.</li></ul>

#### Description

Given that the result of an addition operation performed by an ADD.B or ADDX instruction on 4-bit BCD data is contained in an 8-bit register Rd and the carry and half-carry flags, the DAA instruction adjusts the contents of the 8-bit register Rd (destination operand) by adding H'00, H'06, H'60, or H'66 according to the table below.

C Flag before Adjustment	Upper 4 Bits before Adjustment	H Flag before Adjustment	Lower 4 Bits before Adjustment	Value Added (Hexadecimal)	C Flag after Adjustment
0	0 to 9	0	0 to 9	00	0
0	0 to 8	0	A to F	06	0
0	0 to 9	1	0 to 3	06	0
0	A to F	0	0 to 9	60	1
0	9 to F	0	A to F	66	1
0	A to F	1	0 to 3	66	1
1	0 to 2	0	0 to 9	60	1
1	0 to 2	0	A to F	66	1
1	0 to 3	1	0 to 3	66	1

#### DAA (Decimal Adjust Add)

#### **Decimal Adjust**

#### **Available Registers**

Rd: R0L to R7L, R0H to R7H

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands			Ins	tructic	on Format		No. of
Mode	Milenonie	operando	1st byte		2nd	byte	3rd byte	4th byte	States
Register direct	DAA	Rd	0	F	0	rd			1

#### Notes

Valid results (8-bit register Rd contents and C, V, Z, N, and H flags) are not assured if this instruction is executed under conditions other than those described above.



#### 2.2.25 DAS

#### DAS (Decimal Adjust Subtract)

#### Operation

Rd (decimal adjust)  $\rightarrow$  Rd

#### Assembly-Language Format

DAS Rd

#### **Operand Size**

Byte

#### **Condition Code**

Ι	UI	Η	U	Ν	Ζ	V	С
		*		$\uparrow$	$\uparrow$	*	0

- H: Undetermined (no guaranteed value).
- N: Set to 1 if the adjusted result is negative; otherwise cleared to 0.
- Z: Set to 1 if the adjusted result is zero; otherwise cleared to 0.
- V: Undetermined (no guaranteed value).
- C: Previous value remains unchanged.

#### Description

Given that the result of a subtraction operation performed by a SUB.B, SUBX.B, or NEG.B instruction on 4-bit BCD data is contained in an 8-bit register Rd and the carry and half-carry flags, the DAS instruction adjusts the contents of the 8-bit register Rd (destination operand) by adding H'00, H'FA, H'A0, or H'9A according to the table below.

C Flag before Adjustment	Upper 4 Bits before Adjustment	H Flag before Adjustment	Lower 4 Bits before Adjustment	Value Added (Hexadecimal)	C Flag after Adjustment
0	0 to 9	0	0 to 9	00	0
0	0 to 8	1	6 to F	FA	0
1	7 to F	0	0 to 9	A0	1
1	6 to F	1	6 to F	9A	1

#### **Available Registers**

Rd: R0L to R7L, R0H to R7H

#### DAS (Decimal Adjust Subtract)

#### **Decimal Adjust**

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands		Instructio	on Format		No. of	
Mode	Witemonic	Operanus	1st byte	2nd byte	3rd byte 4th byte		States	
Register direct	DAS	Rd	1 F	0 rd			1	

#### Notes

Valid results (8-bit register Rd contents and C, V, Z, N, and H flags) are not assured if this instruction is executed under conditions other than those described above.

#### **2.2.26** (1) **DEC** (**B**)

#### **DEC (DECrement)**

#### Decrement

Operation	Condition Code
$Rd - 1 \rightarrow Rd$	$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$
Assembly-Language Format	<ul><li>H: Previous value remains unchanged.</li><li>N: Set to 1 if the result is negative; otherwise cleared to 0.</li></ul>
	Z: Set to 1 if the result is zero; otherwise cleared to 0.
<b>Operand Size</b> Byte	<ul><li>V: Set to 1 if an overflow occurs; otherwise cleared to 0.</li><li>C: Previous value remains unchanged.</li></ul>

#### Description

This instruction decrements an 8-bit register Rd (destination operand) and stores the result in the 8-bit register Rd.

#### **Available Registers**

Rd: R0L to R7L, R0H to R7H

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands			Ins	tructio	on Format		No. of
Mode	Mileinoino	operando	1st I	byte	2nd	byte	3rd byte	4th byte	States
Register direct	DEC.B	Rd	1	A	0	rd			1

#### Notes

An overflow is caused by the operation  $H'80 - 1 \rightarrow H'7F$ .



#### 2.2.26 (2) DEC (W)

#### **DEC (DECrement)**

#### Operation

 $\begin{array}{l} Rd-1 \rightarrow Rd \\ Rd-2 \rightarrow Rd \end{array}$ 

#### **Assembly-Language Format**

DEC.W #1, Rd DEC.W #2, Rd

#### **Operand Size**

Word

# Condition CodeIUIHUNZVC--- $\uparrow$ $\uparrow$ $\uparrow$ -

- H: Previous value remains unchanged.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Set to 1 if an overflow occurs; otherwise cleared to 0.
- C: Previous value remains unchanged.

#### Description

This instruction subtracts the immediate value 1 or 2 from the contents of a 16-bit register Rd (destination operand) and stores the result in the 16-bit register Rd.

#### **Available Registers**

Rd: R0 to R7, E0 to E7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic Operands			No. of					
Mode	Witemonic	Operatios	1st I	byte	2nd	byte	3rd byte	4th byte	States
Register direct	DEC.W	#1, Rd	1	В	5	rd			1
Register direct	DEC.W	#2, Rd	1	В	D	rd			1

#### Notes

An overflow is caused by the operations H'8000 – 1  $\rightarrow$  H'7FFF, H'8000 – 2  $\rightarrow$  H'7FFE, and H'8001 – 2  $\rightarrow$  H'7FFF.

## Renesas

#### Decrement

#### 2.2.26 (3) DEC (L)

#### **DEC (DECrement)**

#### Decrement

Operation	Condition Code
$ERd - 1 \rightarrow ERd$ $ERd - 2 \rightarrow ERd$	$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$
Assembly-Language Format DEC.L #1, ERd DEC.L #2, ERd	<ul><li>H: Previous value remains unchanged.</li><li>N: Set to 1 if the result is negative; otherwise cleared to 0.</li><li>Z: Set to 1 if the result is zero; otherwise cleared to 0.</li></ul>
<b>Operand Size</b> Longword	<ul><li>V: Set to 1 if an overflow occurs; otherwise cleared to 0.</li><li>C: Previous value remains unchanged.</li></ul>

#### Description

This instruction subtracts the immediate value 1 or 2 from the contents of a 32-bit register ERd (destination operand) and stores the result in the 32-bit register ERd.

#### **Available Registers**

ERd: ER0 to ER7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands			No. of					
Mode	Millemonic	Operanus	1st	byte	2nd	by	/te	3rd byte	4th byte	States
Register direct	DEC.L	#1, ERd	1	В	7	0	erd			1
Register direct	DEC.L	#2, ERd	1	В	F	0	erd			1

#### Notes

An overflow is caused by the operations H'80000000 – 1  $\rightarrow$  H'7FFFFFFF, H'80000000 – 2  $\rightarrow$  H'7FFFFFFE, and H'80000001 – 2  $\rightarrow$  H'7FFFFFFF.

**Divide Signed** 

#### 2.2.27 (1) DIVXS (B)

#### **DIVXS (DIVide eXtend as Signed)**

#### Operation

 $Rd \div Rs \rightarrow Rd$ 

#### Assembly-Lan

DIVXS.B Rs

#### **Operand Size**

Byte

#### Description

This instruction divides the contents of a 16-bit register Rd (destination operand) by the contents of an 8-bit register Rs (source operand) and stores the result in the 16-bit register Rd. The division is signed. The operation performed is 16 bits  $\div$  8 bits  $\rightarrow$  8-bit quotient and 8-bit remainder. The quotient is placed in the lower 8 bits of Rd. The remainder is placed in the upper 8 bits of Rd. The sign of the remainder matches the sign of the dividend.



Valid results are not assured if division by zero is attempted or an overflow occurs.

#### **Available Registers**

Rd: R0 to R7, E0 to E7 Rs: R0L to R7L, R0H to R7H

## Renesas

	Condition Code
1	I UI H U N Z V C
<b>nguage Format</b> s, Rd	<ul> <li> - - -  ↓ ↓  - - </li> <li>H: Previous value remains unchanged.</li> <li>N: Set to 1 if the quotient is negative; otherwise cleared to 0.</li> <li>Z: Set to 1 if the divisor is zero; otherwise cleared to 0.</li> <li>V: Previous value remains unchanged.</li> <li>C: Previous value remains unchanged.</li> </ul>
	c. Trevious value remains unenangeu.

#### DIVXS (DIVide eXtend as Signed)

#### **Divide Signed**

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic Operands			Instruction Format								
Mode	Witemonic	Operanus	1st byte		2nd byte		3rd byte		4th byte		States	
Register direct	DIVXS.B	Rs, Rd	0	1	D	0	5	1	rs	rd	13	

#### Notes

The N flag is set to 1 if the dividend and divisor have different signs, and cleared to 0 if they have the same sign. The N flag may therefore be set to 1 when the quotient is zero.



**Divide Signed** 

#### 2.2.27 (2) DIVXS (W)

#### DIVXS (DIVide eXtend as Signed)

#### Operation

 $ERd \div Rs \rightarrow ERd$ 

#### **Assembly-Language Format**

DIVXS.W Rs, ERd

#### **Operand Size**

Word

#### Description

This instruction divides the contents of a 32-bit register ERd (destination operand) by the contents of a 16-bit register Rs (source operand) and stores the result in the 32-bit register ERd. The division is signed. The operation performed is 32 bits  $\div$  16 bits  $\rightarrow$  16-bit quotient and 16-bit remainder. The quotient is placed in the lower 16 bits (Rd) of the 32-bit register ERd. The remainder is placed in the upper 16 bits (Ed). The sign of the remainder matches the sign of the dividend.

Valid results are not assured if division by zero is attempted or an overflow occurs.

#### **Available Registers**

ERd: ER0 to ER7 Rs: R0 to R7, E0 to E7

## Renesas



- otherwise cleared to 0.Z: Set to 1 if the divisor is zero; otherwise cleared to 0.
- V: Previous value remains unchanged.
- C: Previous value remains unchanged.

#### DIVXS (DIVide eXtend as Signed)

#### **Divide Signed**

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands			Ins	tructio	on Forr	nat			No. of
Mode	Millemonic	Operanus	1st byte		2nd byte		3rd byte		4th byte		States
Register direct	DIVXS.W	Rs, ERd	0	1	D	0	5	3	rs	0 erd	21

#### Notes

The N flag is set to 1 if the dividend and divisor have different signs, and cleared to 0 if they have the same sign. The N flag may therefore be set to 1 when the quotient is zero.



#### 2.2.28 (1) DIVXU (B)

DIVXU	(DIVide	eXtend	as	Unsigned)
-------	---------	--------	----	-----------

#### Operation

 $Rd \div Rs \to Rd$ 

#### **Assembly-Language Format**

DIVXU.B Rs, Rd

#### **Operand Size**

Byte

#### Description

This instruction divides the contents of a 16-bit register Rd (destination operand) by the contents of an 8-bit register Rs (source operand) and stores the result in the 16-bit register Rd. The division is unsigned. The operation performed is 16 bits  $\div$  8 bits  $\rightarrow$  8-bit quotient and 8-bit remainder. The quotient is placed in the lower 8 bits of Rd. The remainder is placed in the upper 8 bits of Rd.



Valid results are not assured if division by zero is attempted or an overflow occurs.

#### **Available Registers**

Rd: R0 to R7, E0 to E7 Rs: R0L to R7L, R0H to R7H

## Renesas

#### Divide



- H: Previous value remains unchanged.
- N: Set to 1 if the divisor is negative; otherwise cleared to 0.
- Z: Set to 1 if the divisor is zero; otherwise cleared to 0.
- V: Previous value remains unchanged.
- C: Previous value remains unchanged.

#### DIVXU (DIVide eXtend as Unsigned)

#### Divide

#### **Operand Format and Number of States Required for Execution**

Addressing	Addressing Mnemonic Operands Instruction Format								No. of
Mode	Witemonic	Operanus	1st I	byte	2nd	byte	3rd byte	4th byte	States
Register direct	DIVXU.B	Rs, Rd	5	1	rs	rd			12

Notes



#### 2.2.28 (2) **DIVXU** (W)

DIVXU	(DIVide	eXtend a	as Unsigned)
-------	---------	----------	--------------

#### Operation

 $\mathsf{ERd} \div \mathsf{Rs} \to \mathsf{ERd}$ 

#### **Assembly-Language Format**

DIVXU.W Rs, ERd

#### **Operand Size**

Word

#### Description

This instruction divides the contents of a 32-bit register ERd (destination operand) by the contents of a 16-bit register Rs (source register) and stores the result in the 32-bit register ERd. The division is unsigned. The operation performed is 32 bits  $\div$  16 bits  $\rightarrow$  16-bit quotient and 16-bit remainder. The quotient is placed in the lower 16 bits (Rd) of the 32-bit register ERd. The remainder is placed in the upper 16 bits of (Ed).



Valid results are not assured if division by zero is attempted or an overflow occurs.

#### **Available Registers**

ERd: ER0 to ER7 Rs: R0 to R7, E0 to E7

## Renesas

#### Divide



- H: Previous value remains unchanged.
- N: Set to 1 if the divisor is negative; otherwise cleared to 0.
- Z: Set to 1 if the divisor is zero; otherwise cleared to 0.
- V: Previous value remains unchanged.
- C: Previous value remains unchanged.

#### DIVXU (DIVide eXtend as Unsigned)

#### Divide

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands			Instructio	on Format		No. of
Mode	Witemonic	Operanus	1st I	byte	2nd byte	3rd byte	4th byte	States
Register direct	DIVXU.W	Rs, ERd	5	3	rs 0 erd			20

Notes



Ζ

#### **2.2.29 (1) EEPMOV (B)**

#### **EEPMOV (MOVe data to EEPROM)**

#### Operation

if  $R4L \neq 0$  then repeat @ER5+  $\rightarrow$  @ER6+  $R4L - 1 \rightarrow R4L$ until R4L = 0 else next;

#### Assembly-Language Format

EEPMOV.B

#### **Operand Size**

## Condition Code



- H: Previous value remains unchanged.
- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Previous value remains unchanged.

#### Description

This instruction performs a block data transfer. It moves data from the memory location specified in ER5 to the memory location specified in ER6, increments ER5 and ER6, decrements R4L, and repeats these operations until R4L reaches zero. Execution then proceeds to the next instruction. The data transfer is performed a byte at a time, with R4L indicating the number of bytes to be transferred. The byte symbol in the assembly-language format designates the size of R4L (and limits the maximum number of bytes that can be transferred to 255). No interrupts are detected while the block transfer is in progress.

When the EEPMOV.B instruction ends, R4L contains 0 (zero), and ER5 and ER6 contain the last transfer address + 1.

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands			Ins	tructic	on Forr	nat			No. of
Mode	Witterfiorfic	Operanus	1st byte		2nd byte		3rd byte		4th byte		States
_	EEPMOV.B		7	В	5	С	5	9	8	F	4 + 2n*

Note: \* n is the initial value of R4L. Although n bytes of data are transferred, 2(n + 1) data accesses are performed, requiring 2(n + 1) states. (n = 0, 1, 2, ..., 255).

#### Notes

This instruction first reads the memory locations indicated by ER5 and ER6, then carries out the block data transfer.

## Renesas

#### **Block Data Transfer**

#### **2.2.29 (2) EEPMOV (W)**

#### **EEPMOV (MOVe data to EEPROM)**

#### Operation

if  $R4 \neq 0$  then repeat @ER5+  $\rightarrow$  @ER6+  $R4 - 1 \rightarrow R4$ until R4 = 0 else next;

#### Assembly-Language Format

EEPMOV.W

#### **Operand Size**

#### **Condition Code**

Ι	UI	Η	U	Ν	Ζ	V	С
—	—	_		—			

- H: Previous value remains unchanged.
- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Previous value remains unchanged.

#### Description

This instruction performs a block data transfer. It moves data from the memory location specified in ER5 to the memory location specified in ER6, increments ER5 and ER6, decrements R4, and repeats these operations until R4 reaches zero. Execution then proceeds to the next instruction. The data transfer is performed a byte at a time, with R4 indicating the number of bytes to be transferred. The word symbol in the assembly-language format designates the size of R4 (allowing a maximum 65535 bytes to be transferred). All interrupts are detected while the block transfer is in progress.

If no interrupt occurs while the EEPMOV.W instruction is executing, when the EEPMOV.W instruction ends, R4 contains 0 (zero), and ER5 and ER6 contain the last transfer address + 1.

If an interrupt occurs, interrupt exception handling begins after the current byte has been transferred. R4 indicates the number of bytes remaining to be transferred. ER5 and ER6 indicate the next transfer addresses. The program counter value pushed onto the stack in interrupt exception handling is the address of the next instruction after the EEPMOV.W instruction.

See the note on EEPMOV.W instruction and interrupt.



#### **Block Data Transfer**

#### **EEPMOV (MOVe data to EEPROM)**

#### **Block Data Transfer**

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands			Ins	tructio	on Forr	nat			No. of
Mode	Witterfiorfic	Operanus	1st I	byte	2nd	byte	3rd	byte	4th	byte	States
	EEPMOV.W		7	В	D	4	5	9	8	F	4 + 2n*

Note: \* n is the initial value of R4. Although n bytes of data are transferred, 2(n + 1) data accesses are performed, requiring 2(n + 1) states. (n = 0, 1, 2, ..., 65535).

#### Notes

This instruction first reads memory at the addresses indicated by ER5 and ER6, then carries out the block data transfer.

#### **EEPMOV.W Instruction and Interrupt**

If an interrupt request occurs while the EEPMOV.W instruction is being executed, interrupt exception handling is carried out after the current byte has been transferred. Register contents are then as follows:

- ER5: address of the next byte to be transferred
- ER6: destination address of the next byte
- R4: number of bytes remaining to be transferred

The program counter value pushed on the stack in interrupt exception handling is the address of the next instruction after the EEPMOV.W instruction. Programs should be coded as follows to allow for interrupts during execution of the EEPMOV.W instruction.

#### **Example:**

L1: EEPMOV.W MOV.W R4,R4 BNE L1

Interrupt requests other than NMI are not accepted if they are masked in the CPU.

During execution of the EEPMOV.B instruction no interrupts are accepted, including NMI.

## Renesas

#### 2.2.30 (1) EXTS (W)

#### EXTS (EXTend as Signed)

#### Sign Extension

Operation	Condition Code
$(\langle Bit 7 \rangle \text{ of } Rd) \rightarrow (\langle bits 15 \text{ to } 8 \rangle \text{ of } Rd)$	$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$
Assembly-Language Format	- H: Previous value remains unchanged.
EXTS.W Rd	N: Set to 1 if the result is negative; otherwise cleared to 0.
	Z: Set to 1 if the result is zero; otherwise
Operand Size	<ul><li>cleared to 0.</li><li>V: Always cleared to 0.</li></ul>
Word	C: Previous value remains unchanged.

#### Description

This instruction copies the sign of the lower 8 bits in a 16-bit register Rd in the upward direction (copies Rd bit 7 to bits 15 to 8) to extend the data to signed word data.



#### **Available Registers**

Rd: R0 to R7, E0 to E7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands		No. of					
Mode	lode Minemonic Operands		1st byte		2nd byte		3rd byte	4th byte	States
Register direct	EXTS.W	Rd	1	7	D	rd			1

#### Notes

#### 2.2.30 (2) EXTS (L)

#### EXTS (EXTend as Signed)

#### Sign Extension

## Operation

(<Bit 15> of ERd)  $\rightarrow$  (<bits 31 to 16> of ERd)

#### Assembly-Language Format

EXTS.L ERd

#### **Operand Size**

Longword

#### **Condition Code**



- H: Previous value remains unchanged.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Previous value remains unchanged.

## Description

This instruction copies the sign of the lower 16 bits in a 32-bit register ERd in the upward direction (copies ERd bit 15 to bits 31 to 16) to extend the data to signed longword data.



#### **Available Registers**

ERd: ER0 to ER7

#### **Operand Format and Number of States Required for Execution**

Addressing	Addressing Mode Mnemonic Operands			No. of			
Mode			1st byte	2nd byte	3rd byte	4th byte	States
Register direct	EXTS.L	ERd	1 7	F 0 erd			1

#### Notes

## Renesas

#### 2.2.31 (1) EXTU (W)

#### EXTU (EXTend as Unsigned)

#### Operation

 $0 \rightarrow (< bits 15 \text{ to } 8 > of \text{ Rd})$ 

#### Assembly-Language Format

EXTU.W Rd

#### **Operand Size**

Word

## Condition Code

Ι	UI	Η	U	Ν	Ζ	V	С
				0	$\updownarrow$	0	

- H: Previous value remains unchanged.
- N: Always cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Previous value remains unchanged.

#### Description

This instruction extends the lower 8 bits in a 16-bit register Rd to word data by padding with zeros. That is, it clears the upper 8 bits of Rd (bits 15 to 8) to 0.



#### **Available Registers**

Rd: R0 to R7, E0 to E7

#### **Operand Format and Number of States Required for Execution**

Addressing	Addressing Mode Mnemonic Operands –			Instruction Format							
Mode			1st byte		2nd byte		3rd byte	4th byte	States		
Register direct	EXTU.W	Rd	1	7	5	rd			1		

#### Notes

## RENESAS

#### **Zero Extension**

#### 2.2.31 (2) EXTU (L)

#### EXTU (EXTend as Unsigned)

#### Operation

 $0 \rightarrow (< bits 31 \text{ to } 16 > of ERd)$ 

#### Assembly-Language Format

EXTU.L ERd

#### **Operand Size**

Longword

#### Description

This instruction extends the lower 16 bits (general register Rd) in a 32-bit register ERd to longword data by padding with zeros. That is, it clears the upper 16 bits of ERd (bits 31 to 16) to 0.

**Condition Code** 

N: Always cleared to 0.

cleared to 0. V: Always cleared to 0.

H: Previous value remains unchanged.

C: Previous value remains unchanged.

Z: Set to 1 if the result is zero: otherwise

I UI H U N Z — — — — 0 ↓



#### **Available Registers**

ERd: ER0 to ER7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands		No. of			
Mode	Mode Minemonic Operands		1st byte	2nd byte	3rd byte	4th byte	States
Register direct	EXTU.L	ERd	1 7	7 0 erd			1

#### Notes

## RENESAS

#### **Zero Extension**

0

#### 2.2.32 (1) INC (B)

#### INC (INCrement)

#### Increment

Operation	Condition Code
$Rd + 1 \rightarrow Rd$	$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$
Assembly-Language Format	<ul><li>H: Previous value remains unchanged.</li><li>N: Set to 1 if the result is negative; otherwise cleared to 0.</li></ul>
	Z: Set to 1 if the result is zero; otherwise cleared to 0.
<b>Operand Size</b> Byte	<ul><li>V: Set to 1 if an overflow occurs; otherwise cleared to 0.</li><li>C: Previous value remains unchanged.</li></ul>

#### Description

This instruction increments an 8-bit register Rd (destination operand) and stores the result in the 8-bit register Rd.

#### **Available Registers**

Rd: R0L to R7L, R0H to R7H

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands		No. of					
Mode	Millenionic	operando	1st I	1st byte		byte	3rd byte	4th byte	States
Register direct	INC.B	Rd	0	A	0	rd			1

#### Notes

An overflow is caused by the operation  $H'7F + 1 \rightarrow H'80$ .



#### 2.2.32 (2) INC (W)

#### INC (INCrement)

#### Operation

 $\begin{array}{l} Rd+1 \rightarrow Rd \\ Rd+2 \rightarrow Rd \end{array}$ 

#### **Assembly-Language Format**

INC.W #1, Rd INC.W #2, Rd

#### **Operand Size**

Word

#### Increment



#### Description

This instruction adds the immediate value 1 or 2 to the contents of a 16-bit register Rd (destination operand) and stores the result in the 16-bit register Rd.

#### **Available Registers**

Rd: R0 to R7, E0 to E7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands		No. of					
Mode	Witterfiorfic	Operanus	1st byte		2nd byte		3rd byte	4th byte	States
Register direct	INC.W	#1, Rd	0	В	5	rd			1
Register direct	INC.W	#2, Rd	0	В	D	rd			1

#### Notes

An overflow is caused by the operations H'7FFF + 1  $\rightarrow$  H'8000, H'7FFF + 2  $\rightarrow$  H'8001, and H'7FFE + 2  $\rightarrow$  H'8000.

## Renesas

#### 2.2.32 (3) INC (L)

#### INC (INCrement)

#### Increment

Operation	Condition Code
$ERd + 1 \rightarrow ERd$ $ERd + 2 \rightarrow ERd$	$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$
Assembly-Language Format INC.L #1, ERd INC.L #2, ERd	<ul><li>H: Previous value remains unchanged.</li><li>N: Set to 1 if the result is negative; otherwise cleared to 0.</li><li>Z: Set to 1 if the result is zero; otherwise cleared to 0.</li></ul>
<b>Operand Size</b> Longword	<ul><li>V: Set to 1 if an overflow occurs; otherwise cleared to 0.</li><li>C: Previous value remains unchanged.</li></ul>

#### Description

This instruction adds the immediate value 1 or 2 to the contents of a 32-bit register ERd (destination operand) and stores the result in the 32-bit register ERd.

#### **Available Registers**

ERd: ER0 to ER7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands		Instruction Format								
Mode	Millemonic	Operanus	1st byte		2nd byte		/te	3rd byte	4th byte	States		
Register direct	INC.L	#1, ERd	0	В	7	0	erd			1		
Register direct	INC.L	#2, ERd	0	В	F	0	erd			1		

#### Notes

An overflow is caused by the operations H'7FFFFFF + 1  $\rightarrow$  H'80000000, H'7FFFFFFF + 2  $\rightarrow$  H'80000001, and H'7FFFFFFE + 2  $\rightarrow$  H'80000000.

**Unconditional Branch** 

#### 2.2.33 JMP

#### JMP (JuMP)

#### Operation

Effective address  $\rightarrow$  PC

#### **Assembly-Language Format**

JMP <EA>

#### **Operand Size**

#### Condition Code



- H: Previous value remains unchanged.
- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Previous value remains unchanged.

#### Description

This instruction branches unconditionally to a specified effective address.

#### **Available Registers**

ERn: ER0 to ER7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands			Instru	No. of States				
Mode	winemonic	Operands	1st I	1st byte 2nd byte 3		3rd byte	4th byte	Normal	Advanced	
Register indirect	JMP	@ERn	5	9	0 ern	0 ern 0			2	
Absolute address	JMP	@aa:24	5	Α			abs		3	
Memory indirect	JMP	@@aa:8	5	В	abs				4	5

#### Notes

The structure of the branch address and the number of states required for execution differ between normal mode and advanced mode.

Ensure that the branch destination address is even.

#### 2.2.34 JSR

#### JSR (Jump to SubRoutine)

#### Operation

 $PC \rightarrow @-SP$ Effective address  $\rightarrow PC$ 

#### Assembly-Language Format

JSR <EA>

#### **Condition Code**



- H: Previous value remains unchanged.
- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Previous value remains unchanged.

#### **Operand Size**

#### Description

This instruction pushes the program counter onto the stack as a return address, then branches to a specified effective address. The program counter value pushed onto the stack is the address of the instruction following the JSR instruction.

#### **Available Registers**

ERn: ER0 to ER7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Inemonic Operands			Instructio	No. of States			
Mode	whethonic	Operands	1st I	byte	2nd byte	3rd byte	4th byte	Normal	Advanced
Register indirect	JSR	@ERn	5	D	0 ern 0			3	4
Absolute address	JSR	@aa:24	5	Е		abs		4	5
Memory indirect	JSR	@@aa:8	5	F	abs			4	6



#### Jump to Subroutine

#### JSR (Jump to SubRoutine)

#### Jump to Subroutine

#### Notes

The stack structure differs between normal mode and advanced mode. In normal mode only the lower 16 bits of the program counter are pushed onto the stack.

Ensure that the branch destination address is even.



#### 2.2.35 (1) LDC (B)

#### LDC (LoaD to Control register)

Operation	Condition Code
$\langle EAs \rangle \rightarrow CCR$	$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$
Assembly-Language Format LDC.B <eas>, CCR</eas>	<ul><li>I: Loaded from the corresponding bit in the source operand.</li><li>H: Loaded from the corresponding bit in the source operand.</li><li>N: Loaded from the corresponding bit in the</li></ul>
<b>Operand Size</b> Byte	<ul> <li>source operand.</li> <li>Z: Loaded from the corresponding bit in the source operand.</li> <li>V: Loaded from the corresponding bit in the source operand.</li> <li>C: Loaded from the corresponding bit in the source operand.</li> </ul>

#### Description

This instruction loads the source operand contents into the condition-code register (CCR).

No interrupt requests, including NMI, are accepted immediately after execution of this instruction.

#### **Available Registers**

Rs: R0L to R7L, R0H to R7H

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands			Ins	tructio	on Format		No. of
Mode	winemonic	Operatius	1st I	byte	2nd	byte	3rd byte	4th byte	States
Immediate	LDC.B	#xx:8, CCR	0	7	IM	IM			1
Register direct	LDC.B	Rs, CCR	0	3	0	rs			1

#### Notes

#### 2.2.35 (2) LDC (B)

#### LDC (LoaD to Control register)

#### Operation

 $\langle EAs \rangle \rightarrow EXR$ 

#### **Assembly-Language Format**

LDC.B <EAs>, EXR

## Condition Code



- H: Previous value remains unchanged.
- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Previous value remains unchanged.

#### **Operand Size**

Byte

#### Description

This instruction loads the source operand contents into the extended control register (EXR).

No interrupt requests, including NMI, are accepted for three states after execution of this instruction.

#### Available Registers

Rs: R0L to R7L, R0H to R7H

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands			Ins	tructio	on Forr	nat		No. of
Mode	Witterfiorfic	Operanus	1st I	byte	2nd	byte	3rd	byte	4th byte	States
Immediate	LDC.B	#xx:8, EXR	0	1	4	1	0	7	IMM	2
Register direct	LDC.B	Rs, EXR	0	3	1	rs				1

Notes

## RENESAS

#### Load EXR

#### 2.2.35 (3) LDC (W)

#### LDC (LoaD to Control register)

Operation	Condition Code
$(EAs) \rightarrow CCR$	I UI H U N Z V C
Assembly-Language Format	I: Loaded from the corresponding bit in the
LDC.W <eas>, CCR</eas>	source operand. H: Loaded from the corresponding bit in the source operand.
Operand Size	N: Loaded from the corresponding bit in the source operand.
Word	Z: Loaded from the corresponding bit in the source operand.
	V: Loaded from the corresponding bit in the source operand.
	C: Loaded from the corresponding bit in the source operand.

#### Description

This instruction loads the source operand contents into the condition-code register (CCR). Although CCR is a byte register, the source operand is word size. The contents of the even address are loaded into CCR.

No interrupt requests, including NMI, are accepted immediately after execution of this instruction.

**Available Registers** ERs: ER0 to ER7



**Operand Format and Number of States Required for Execution** 

Addressing											Instruction Format	n Forn	lat					No. of
Mode	Mnemonic	Operands	1st b	1st byte	2nd byte 3rd byte	yte	3rd by		4th byte		5th byte		yte	7th byte	6th byte 7th byte 8th byte	9th byte	9th byte 10th byte States	States
Register indirect	LDC.W	@ERs, CCR	0	-	4	0	9	6	0: ers	0								e
Register indirect with	LDC.W	@(d:16, ERs), CCR	0	-	4	0	9	ш	0 ers 0	0	đi	disp						4
displace- ment	LDC.W	@(d:32, ERs), CCR 0		-	4	0	2	8	0 ers	0	B 9	N	0		P	disp		9
Register indirect with post- increment	LDC.W	@ERs+, CCR	0	-	4	0	9	0	0 ers	0								4
Absolute	LDC.W	@aa:16, CCR	0	-	4	0	9	в	0	0	σ	abs						4
address	LDC.W	@aa:32, CCR	0	1	4 0	0	9	В	7	0			abs					5

LDC (LoaD to Control register)

Notes

Load CCR

2.2.35 (4) LDC (W)

#### LDC (LoaD to Control register)

#### Operation

 $(EAs) \rightarrow EXR$ 

#### Assembly-Language Format

LDC.W <EAs>, EXR

#### **Condition Code**



- H: Previous value remains unchanged.
- N: Previous value remains unchanged.

Z: Previous value remains unchanged.

V: Previous value remains unchanged.

C: Previous value remains unchanged.

#### **Operand Size**

Word

#### Description

This instruction loads the source operand contents into the extended control register (EXR). Although EXR is a byte register, the source operand is word size. The contents of the even address are loaded into EXR.

No interrupt requests, including NMI, are accepted for three states after execution of this instruction.

**Available Registers** ERs: ER0 to ER7 **Operand Format and Number of States Required for Execution** 

Addressing										Instructio	Instruction Format					No. of
Mode	MILIEIMOULIC	Operarius	1st byte		2nd byte	yte	3rd byte	vte	4th byte	5th byte	6th byte	7th byte	8th byte	9th byte	10th byte States	States
Register indirect	LDC.W	@ERs, EXR	0	-	4	-	9	ი	0: ers 0							3
Register indirect with	LDC.W	@(d:16, ERs), EXR	0	-	4	-	9	Ľ.	0 ers 0		disp					4
displace- ment	LDC.W	@(d:32, ERs), EXR	0	-	4	+	7	8	0 ers 0	9 9	2 0		q	disp		9
Register indirect with post- increment	LDC.W	@ERs+, EXR	o	-	4	-	٥		0 ers							4
Absolute	LDC.W	@aa:16, EXR	0	-	4	-	9	۵	0		abs					4
address	LDC.W	@aa:32, EXR	0	-	4	-	9	в	2 0		at	abs				5

LDC (LoaD to Control register)

Notes

#### Load EXR

#### 2.2.36 LDM

#### LDM (LoaD to Multiple registers)

#### Operation

 $@SP+ \rightarrow ERn$  (register list)

#### **Assembly-Language Format**

LDM.L @SP+, <register list>

#### **Restore Data from Stack**

#### **Condition Code**



- H: Previous value remains unchanged.
- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Previous value remains unchanged.

#### **Operand Size**

Longword

#### Description

This instruction restores data saved on the stack to a specified list of registers. Registers are restored in descending order of register number.

Two, three, or four registers can be restored by one LDM instruction. The following ranges can be specified in the register list.

Two registers:ER0–ER1, ER2–ER3, ER4–ER5, or ER6–ER7Three registers:ER0–ER2 or ER4–ER6Four registers:ER0–ER3 or ER4–ER7

#### **Available Registers**

ERn: ER0 to ER7


#### LDM (LoaD to Multiple registers)

#### **Restore Data from Stack**

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	omenia Onerendo		Instruction Format								
Mode	whemonic	Operands	1st I	oyte	2nd	byte	3rd	byte	4tl	h byte	States	
_	LDM.L	@SP+, (ERn–ERn+1)	0	1	1	0	6	D	7	0 ern+1	7	
_	LDM.L	@SP+, (ERn–ERn+2)	0	1	2	0	6	D	7	0 ern+2	9	
_	LDM.L	@SP+, (ERn–ERn+3)	0	1	3	0	6	D	7	0 ern+3	11	

Notes

#### 2.2.37 LDMAC

#### LDMAC (LoaD to MAC register)

#### Operation

 $\begin{array}{l} \text{ERs} \rightarrow \text{MACH} \\ \text{or} \\ \text{ERs} \rightarrow \text{MACL} \end{array}$ 

#### **Assembly-Language Format**

LDMAC ERs, MAC registe	AC register
------------------------	-------------

#### **Operand Size**

Longword

#### Description

This instruction moves the contents of a general register to a multiply-accumulate register (MACH or MACL). If the transfer is to MACH, only the lowest 10 bits of the general register are transferred.

Supported only by the H8S/2600 CPU.

#### **Available Registers**

ERs: ER0 to ER7

#### **Operand Format and Number of States Required for Execution**

Addressing		Operands		No. of					
Mode	WINEIHOINC	Operations	1st byte		2nd byte		3rd byte	4th byte	States
Register direct	LDMAC	ERs, MACH	0	3	2	0 ers			2*
Register direct	LDMAC	ERs, MACL	0	3	3	0 ers			2*

Note: \* A maximum of three additional states are required for execution of this instruction within three states after execution of a MAC instruction. For example, if there is a one-state instruction (such as NOP) between the MAC instruction and this instruction, this instruction will be two states longer.

The number of states may differ depending on the product. For details, refer to the relevant microcontroller hardware manual of the product in question.

#### Notes

Execution of this instruction clears the overflow flag in the multiplier to 0.

## Load MAC Register

Ι	UI	Н	U	Ν	Ζ	V	С
—	—	_		—	_	—	

H: Previous value remains unchanged.

**Condition Code** 

- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Previous value remains unchanged.



#### 2.2.38 MAC

#### MAC (Multiply and ACcumulate)

#### Operation

 $(EAn) \times (EAm) + MAC$  register  $\rightarrow$ MAC register ERn  $+ 2 \rightarrow ERn$ ERm  $+ 2 \rightarrow ERm$ 

#### Assembly-Language Format

MAC @ERn+, @ERm+

## Condition Code



- H: Previous value remains unchanged.
- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Previous value remains unchanged.

## Description

**Operand Size** 

This instruction performs signed multiplication on two 16-bit operands at addresses given by the contents of general registers ERn and ERm, adds the 32-bit product to the contents of the MAC register, and stores the sum in the MAC register. After this operation, ERn and ERm are both incremented by 2.

The operation can be carried out in saturating or non-saturating mode, depending on the MACS bit in a system control register. (SYSCR)

See the relevant hardware manual for further information.

In non-saturating mode, MACH and MACL are concatenated to store a 42-bit result. The value of bit 41 is copied into the upper 22 bits of MACH as a sign extension.

In saturating mode, only MACL is valid, and the result is limited to the range from H'80000000 (minimum value) to H'7FFFFFF (maximum value). If the result overflows in the negative direction, H'80000000 (the minimum value) is stored in MACL. If the result overflows in the positive direction, H'7FFFFFFF (the maximum value) is stored in MACL. The LSB of the MACH register indicates the status of the overflow flag (V-MULT) in the multiplier. Other bits retain their previous contents.

This instruction is supported only by the H8S/2600 CPU.

## Renesas

#### Multiply and Accumulate

#### MAC (Multiply and ACcumulate)

#### Multiply and Accumulate

Addressing	Addressing Mode Mnemonic Operands	Onorondo	Instruction Format								No. of		
Mode		1st k	oyte	2nd	byte	3rd	byte		4th	byt	e	States	
Register indirect with post-increment	MAC	@ERn+, @ERm+	0	1	6	0	6	D	0	ern	0	erm	4

#### **Operand Format and Number of States Required for Execution**

#### Notes

- 1. Flags (N, Z, V) indicating the result of the MAC instruction can be set in the condition-code register (CCR) by the STMAC instruction.
- 2. If ERn and ERm are the same register, the execution addresses are ERn and ERn + 2. After execution, the value of ERn is ERn + 4.
- 3. If MACS is modified during execution of a MAC instruction, the result cannot be guaranteed. It is essential to wait for at least three states after a MAC instruction before modifying MACS.

#### Further Explanation of Instructions Using Multiplier

1. Modification of flags

The multiplier has N-MULT, Z-MULT, and V-MULT flags that indicate the results of MAC instructions. These flags are separated from the condition-code register (CCR). The values of these flags can be set in the N, Z, and V flags of the CCR only by the STMAC instruction. N-MULT and Z-MULT are modified only by MAC instructions. V-MULT retains a value indicating whether an overflow has occurred in the past, until it is cleared by execution of the CLRMAC or LDMAC instruction.

The setting and clearing conditions for these flags are given below.

• N-MULT (negative flag)

Saturating mode	Set when bit 31 of register MACL is set to 1 by execution of a MAC instruction
	Cleared when bit 31 of register MACL is cleared to 0 by execution of a MAC instruction
Non-saturating mode	Set when bit 41 of register MACH is set to 1 by execution of a MAC instruction
	Cleared when bit 41 of register MACH is cleared to 0 by execution of a MAC instruction

#### MAC (Multiply and ACcumulate)

#### **Multiply and Accumulate**

Saturating mode	Set when register MACL is cleared to 0 by execution of a MAC instruction
	Cleared when register MACL is not cleared to 0 by execution of a MAC instruction
Non-saturating mode	Set when registers MACH and MACL are both cleared to 0 by execution of a MAC instruction
	Cleared when register MACH or MACL is not cleared to 0 by execution of a MAC instruction

#### • V-MULT (overflow flag)

	-
Saturating mode	Set when the result of the MAC instruction overflows the range from H'80000000 (minimum) to H'7FFFFFFF (maximum)
	Cleared when a CLRMAC or LDMAC instruction is executed
	Note: Not cleared when the result of the MAC instruction is within the above range
Non-saturating mode	Set when the result of the MAC instruction overflows the range from H'20000000000 (minimum) to H'1FFFFFFFFF (maximum)
	Cleared when a CLRMAC or LDMAC instruction is executed
	Note: Not cleared when the result of the MAC instruction is within the above range

The N-MULT, Z-MULT, and V-MULT flags are not modified by switching between saturating and non-saturating modes, or by execution of a multiply instruction (MULXU or MULXS).

#### 2. Example

CLRMAC

```
MAC @ER1+,@ER2+

MAC @ER1+,@ER2+ \leftarrow Overflow occurs

:

MAC @ER1+,@ER2+ \leftarrow Result = 0

NOP

STMAC MACH,ER3 \leftarrow CCR (N = 0, Z = 1, V = 1)

CLRMAC

STMAC MACH,ER3 \leftarrow CCR (N = 0, Z = 1, V = 0)
```

#### 2.2.39 (1) MOV (B)

#### MOV (MOVe data)

Operation	Condition Code
$Rs \rightarrow Rd$	$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$
Assembly-Language Format MOV.B Rs, Rd	<ul> <li>H: Previous value remains unchanged.</li> <li>N: Set to 1 if the transferred data is negative; otherwise cleared to 0.</li> <li>Z: Set to 1 if the transferred data is zero;</li> </ul>
Operand Size Byte	otherwise cleared to 0.         V:       Always cleared to 0.         C:       Previous value remains unchanged.

#### Description

This instruction transfers one byte of data from an 8-bit register Rs to an 8-bit register Rd, tests the transferred data, and sets condition-code flags according to the result.

#### **Available Registers**

Rs: R0L to R7L, R0H to R7H Rd: R0L to R7L, R0H to R7H

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands		No. of					
Mode	Milenonic	operando	1st byte		2nd byte		3rd byte	4th byte	States
Register direct	MOV.B	Rs, Rd	0	С	rs	rd			1

#### Notes



#### 2.2.39 (2) MOV (W)

#### MOV (MOVe data)

#### Move

Operation	Condition Code
$Rs \rightarrow Rd$	$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$
Assembly-Language Format MOV.W Rs, Rd	<ul><li>H: Previous value remains unchanged.</li><li>N: Set to 1 if the transferred data is negative; otherwise cleared to 0.</li></ul>
	Z: Set to 1 if the transferred data is zero; otherwise cleared to 0.
<b>Operand Size</b> Word	<ul><li>V: Always cleared to 0.</li><li>C: Previous value remains unchanged.</li></ul>

#### Description

This instruction transfers one word of data from a 16-bit register Rs to a 16-bit register Rd, tests the transferred data, and sets condition-code flags according to the result.

#### **Available Registers**

Rd: R0 to R7, E0 to E7 Rs: R0 to R7, E0 to E7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic		No. of						
Mode	Witemonic	Operands	1st byte		2nd byte		3rd byte	4th byte	States
Register direct	MOV.W	Rs, Rd	0	D	rs	rd			1

Notes

#### 2.2.39 (3) MOV (L)

#### MOV (MOVe data)

#### Operation **Condition Code** $ERs \rightarrow ERd$ I UI Η U Ν Ζ V ↕ ↑ 0 H: Previous value remains unchanged. **Assembly-Language Format** N: Set to 1 if the transferred data is negative; MOV.L ERs. ERd otherwise cleared to 0. Z: Set to 1 if the transferred data is zero; otherwise cleared to 0. V: Always cleared to 0. **Operand Size** C: Previous value remains unchanged. Longword

#### Description

This instruction transfers one word of data from a 32-bit register ERs to a 32-bit register ERd, tests the transferred data, and sets condition-code flags according to the result.

#### **Available Registers**

ERd: ER0 to ER7 ERs: ER0 to ER7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands			Instructio	n Format		No. of
Mode	Willemonic	Operanus	1st I	byte	2nd byte	3rd byte	4th byte	States
Register direct	MOV.L	ERs, ERd	0	F	1 ers 0 erd			1

#### Notes



#### 2.2.39 (4) MOV (B)

#### MOV (MOVe data)

#### Move

Operation	Condition Code
$(EAs) \rightarrow Rd$	$\begin{array}{c ccccccccccccccccccccccccccccccccccc$
Assembly-Language Format MOV.B <eas>, Rd</eas>	<ul> <li>H: Previous value remains unchanged.</li> <li>N: Set to 1 if the transferred data is negative; otherwise cleared to 0.</li> <li>Z: Set to 1 if the transferred data is zero; otherwise cleared to 0.</li> </ul>
Operand Size Byte	<ul><li>V: Always cleared to 0.</li><li>C: Previous value remains unchanged.</li></ul>

#### Description

This instruction transfers the source operand contents to an 8-bit register Rd, tests the transferred data, and sets condition-code flags according to the result.

#### **Available Registers**

Rd: R0L to R7L, R0H to R7H ERs: ER0 to ER7

**Operand Format and Number of States Required for Execution** 

Addressing Magazia	Macanio							Instruction Format	n Format				No. of
Mode		Operands	1st k	1st byte	2nd byte	yte	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	States
Immediate	MOV.B	#xx:8, Rd	ш	Þ	MMI	5							~
Register indirect	MOV.B	@ERs, Rd	9	ω	0 ers	Ð							7
Register indirect	MOV.B	@(d:16, ERs), Rd	ە	ш	0 ers	Ð	disp	dş					ю
with displace- ment	MOV.B	@(d:32, ERs), Rd	7	œ	0 ers	0	6 A	2 rd		q	disp		ъ
Register indirect with post- increment	MOV.B	@ERs+, Rd	9	C	0 ers	rd							з
	MOV.B	@aa:8, Rd	2	rd	abs	(0							5
Absolute address	MOV.B	@aa:16, Rd	9	A	0	Ð	abs	S					ю
	MOV.B	@aa:32, Rd	9	A	2	p		at	abs				4

Rev. 4.00 Feb 24, 2006 page 138 of 322

REJ09B0139-0400

# Notes

RENESAS

The MOV.B @ER7+, Rd instruction should never be used, because it leaves an odd value in the stack pointer (ER7). For details refer to section 3.3, Exception-Handling State, or to the relevant hardware manual.

For the @aa:8/@aa:16 access range, refer to the relevant microcontroller hardware manual

#### MOV (MOVe data)

Move

#### 2.2.39 (5) MOV (W)

#### MOV (MOVe data)

#### Move

Operation	Condition Code
$(EAs) \rightarrow Rd$	$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$
Assembly-Language Format MOV.W <eas>, Rd</eas>	<ul> <li>H: Previous value remains unchanged.</li> <li>N: Set to 1 if the transferred data is negative; otherwise cleared to 0.</li> <li>Z: Set to 1 if the transferred data is zero; otherwise cleared to 0.</li> </ul>
Operand Size Word	<ul><li>V: Always cleared to 0.</li><li>C: Previous value remains unchanged.</li></ul>

#### Description

This instruction transfers the source operand contents to a 16-bit register Rd, tests the transferred data, and sets condition-code flags according to the result.

#### **Available Registers**

Rd: R0 to R7, E0 to E7 ERs: ER0 to ER7

**Operand Format and Number of States Required for Execution** 

Addressing							Instructio	Instruction Format				No. of
Mode Mnemonic	MINEMONIC	Operands	1st byte		2nd byte	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	States
Immediate	MOV.W	#xx:16, Rd	7 9	0	Ð	2	MMI					7
Register indirect	MOV.W	@ERs, Rd	6 9	0 ers	r s							7
Register indirect	MOV.W	@(d:16, ERs), Rd	ц 9	0 ers	ъ S	<u>ס</u>	disp					m
with displace- ment	MOV.W	@(d:32, ERs), Rd	7 8	0 ers	0 0	۵ ب	2		<u>ס</u>	disp		ى ا
Register indirect with post- increment	MOV.W	@ERs+, Rd	0 9	0 ers	p S							e
Absolute	MOV.W	@aa:16, Rd	В 9	0	Ð	<b>D</b>	abs					ъ
address	MOV.W	@aa:32, Rd	В 9	7	Ð		<b>ਡਾਂ</b>	abs				4

Rev. 4.00 Feb 24, 2006 page 140 of 322

REJ09B0139-0400

# Notes

- The source operand <EAs> must be located at an even address.
- In machine language, MOV.W @ER7+, Rd is identical to POP.W Rd. . . .

#### MOV (MOVe data)

#### 2.2.39 (6) MOV (L)

#### MOV (MOVe data)

#### Move

Operation	Condition Code
$(EAs) \rightarrow ERd$	$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$
Assembly-Language Format MOV.L <eas>, ERd</eas>	<ul> <li>H: Previous value remains unchanged.</li> <li>N: Set to 1 if the transferred data is negative; otherwise cleared to 0.</li> <li>Z: Set to 1 if the transferred data is zero; otherwise cleared to 0.</li> </ul>
Operand Size Longword	<ul><li>V: Always cleared to 0.</li><li>C: Previous value remains unchanged.</li></ul>

#### Description

This instruction transfers the source operand contents to a specified 32-bit register (ERd), tests the transferred data, and sets condition-code flags according to the result. The first memory word located at the effective address is stored in extended register Ed. The next word is stored in general register Rd.



#### **Available Registers**

ERs: ER0 to ER7 ERd: ER0 to ER7



Addressing									-	nstructio	Instruction Format					No. of
Mode	Mnemonic	Operands	1st byte		2nd byte		3rd byte	4th byte		5th byte	6th byte	7th byte	8th byte	9th byte	10th byte States	States
Immediate	WOV.L	#xx:32, Rd	4	A	0 0 erd	g			MMI							ъ
Register indirect	MOV.L	@ERs, ERd	0	-	0	9	ര	0.ers.0.erd	erd							4
Register indirect with	MOV.L	@(d:16, ERs), ERd	0	-	0	9	ш	0 ers 0 erd	erd	<u>ਰ</u>	disp					5
displace- ment	MOV.L	@(d:32, ERs), ERd	0	-	0	7	∞	0 ers	•	в 9	2 0 erd		<del>ס</del>	disp		7
Register indirect with post- increment	MOV.L	@ERs+, ERd	o	-	0 0	9	۵	0: ers: 0: erd	erd							2
Absolute	MOV.L	@aa:16, ERd	0	-	0	9	<u>م</u>	0	0 erd	ធ	abs					5
address	MOV.L	@aa:32, ERd	0	-	0	9	<u>م</u>	2	0:erd		at	abs				6

# Notes

- 1. The source operand <EAs> must be located at an even address.
- 2. In machine language, MOV.L @R7+, ERd is identical to POP.L ERd.

#### MOV (MOVe data)

Rev. 4.00 Feb 24, 2006 page 142 of 322 REJ09B0139-0400

#### 2.2.39 (7) MOV (B)

#### MOV (MOVe data)

#### Move

Condition Code
$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$
<ul><li>H: Previous value remains unchanged.</li><li>N: Set to 1 if the transferred data is negative; otherwise cleared to 0.</li></ul>
Z: Set to 1 if the transferred data is zero; otherwise cleared to 0.
<ul><li>V: Always cleared to 0.</li><li>C: Previous value remains unchanged.</li></ul>

#### Description

This instruction transfers the contents of an 8-bit register Rs (source operand) to a destination location, tests the transferred data, and sets condition-code flags according to the result.

#### **Available Registers**

Rs: R0L to R7L, R0H to R7H ERd: ER0 to ER7

**Operand Format and Number of States Required for Execution** 

Addressing	Mucmould							Instruction Format	n Format				No. of
	MITERIORIC		1st byte	yte	2nd byte	yte	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	States
Register indirect	MOV.B	Rs, @ERd	9	00	1 erd	ſS							7
Register indirect	MOV.B	Rs, @(d:16, ERd)	۵	ш	1 erd	S	di;	disp					m
displace- ment	MOV.B	Rs, @(d:32, ERd)	7	8	0 erd	0	6 A	A rs		qi	disp		5
Register indirect with pre- decrement	MOV.B	Rs, @-Erd	Q	U	1 erd	S							3
	MOV.B	Rs, @aa:8	с	ß	abs	(0							2
Absolute address	MOV.B	Rs, @aa:16	9	A	80	ſS	at	abs					3
	MOV.B	Rs, @aa:32	9	A	A	S		abs	S				4

Rev. 4.00 Feb 24, 2006 page 144 of 322

REJ09B0139-0400

# Notes

RENESAS

- The MOV.B Rs, @-ER7 instruction should never be used, because it leaves an odd value in the stack pointer (ER7). For details refer to section 3.3, Exception-Handling State, or to the relevant hardware manual.
- Execution of MOV.B RnL, @-ERn or MOV.B RnH, @-ERn first decrements ERn by one, then transfers the designated part (RnL or RnH) of the resulting ERn value. ä

#### MOV (MOVe data)

#### 2.2.39 (8) MOV (W)

#### MOV (MOVe data)

#### Move

Operation	Condition Code
$Rs \rightarrow (EAd)$	$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$
Assembly-Language Format MOV.W Rs, <ead></ead>	<ul> <li>H: Previous value remains unchanged.</li> <li>N: Set to 1 if the transferred data is negative; otherwise cleared to 0.</li> <li>Z: Set to 1 if the transferred data is zero; otherwise cleared to 0.</li> </ul>
<b>Operand Size</b> Word	<ul><li>V: Always cleared to 0.</li><li>C: Previous value remains unchanged.</li></ul>

#### Description

This instruction transfers the contents of a 16-bit register Rs (source operand) to a destination location, tests the transferred data, and sets condition-code flags according to the result.

#### **Available Registers**

Rs: R0 to R7, E0 to E7 ERd: ER0 to ER7



Addressing	Macmonio	Coccordo						Instructio	Instruction Format				No. of
Mode			1st byte	yte	2nd byte	yte	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	States
Register indirect	MOV.W	Rs, @ERd	g	ത	1 erd rs	s							2
Register indirect	MOV.W	Rs, @(d:16, ERd)	Q	ш	1 erd rs	S	đ	disp					ъ
displace- ment	MOV.W	Rs, @(d:32, ERd)	7	8	0 erd	0	6 6	A rs		đ	disp		5
Register indirect with pre- decrement	MOV.W	Rs, @-ERd	9	D	1 erd	S							3
Absolute	MOV.W	Rs, @aa:16	9	В	80	ſS	a	abs					3
address	MOV.W	Rs, @aa:32	9	ш	۲	ß		at	abs				4

# Notes

- The destination operand <EAd> must be located at an even address.
- In machine language, MOV.W Rs, @-ER7 is identical to PUSH.W Rs.
- When MOV.W Rn, @-ERn is executed, the transferred value comes from (value of ERn before execution) 2.

## MOV (MOVe data)



Rev. 4.00 Feb 24, 2006 page 146 of 322 REJ09B0139-0400

#### 2.2.39 (9) MOV (L)

#### MOV (MOVe data)

#### Move

Operation	Condition Code
$\text{ERs} \rightarrow (\text{EAd})$	$\begin{array}{c ccccccccccccccccccccccccccccccccccc$
Assembly-Language Format MOV.L ERs, <ead></ead>	<ul> <li>H: Previous value remains unchanged.</li> <li>N: Set to 1 if the transferred data is negative; otherwise cleared to 0.</li> <li>Z: Set to 1 if the transferred data is zero; otherwise cleared to 0.</li> </ul>
<b>Operand Size</b> Longword	<ul><li>V: Always cleared to 0.</li><li>C: Previous value remains unchanged.</li></ul>

#### Description

This instruction transfers the contents of a 32-bit register ERs (source operand) to a destination location, tests the transferred data, and sets condition-code flags according to the result. The extended register (Es) contents are stored at the first word indicated by the effective address. The general register (Rs) contents are stored at the next word.



Available Registers ERs: ER0 to ER7 ERd: ER0 to ER7

## Renesas



Addressing										Instruction Format	n Format					No. of
Mode	Mnemonic	Operands	1st byte	-	2nd byte	-	3rd byte		4th byte	5th byte	6th byte	7th byte	8th byte	9th byte	10th byte States	States
Register indirect	MOV.L	ERs, @ERd	0	-	0	0	9	<del>ر</del>	l erd:0 ers							4
Register indirect with	MOV.L	ERs, @(d:16, ERd)	0	-	0	0		<del>.</del>	erd 0 ers		disp					5
displace- ment	MOVIL	ERs, @(d:32, ERd)	0	-	0	0	2	8	0 erd 0	B Q	A 0 ers		σ	disp		7
Register indirect with pre- decrement	MOV.L	ERs, @-ERd	ο	-	0	0	9	D 	l erd 0 ers							a
Absolute	MOV.L	ERs, @aa:16	0	-	0	0	9	m	8 0 ers		abs					5
address	MOVL	ERs, @aa:32	0	-	0	0	9	۵	A 0 ers		at	abs				9

Rev. 4.00 Feb 24, 2006 page 148 of 322

REJ09B0139-0400

RENESAS

- The destination operand <EAd> must be located at an even address. <u>\_</u>:
- In machine language, MOVL ERs, @-ER7 is identical to PUSH.L ERs.
- When MOVL ERn, @-ERn is executed, the transferred value is (value of ERn before execution) 4. સં સં

#### MOV (MOVe data)

#### 2.2.40 MOVFPE

#### **MOVFPE (MOVe From Peripheral with E clock)**

#### Operation

 $(EAs) \rightarrow Rd$ Synchronized with E clock

#### **Assembly-Language Format**

MOVFPE @aa:16, Rd

#### **Operand Size**

Byte

#### Condition Code



- H: Previous value remains unchanged.
- N: Set to 1 if the transferred data is negative; otherwise cleared to 0.
- Z: Set to 1 if the transferred data is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Previous value remains unchanged.

#### Description

This instruction transfers memory contents specified by a 16-bit absolute address to a general register Rd in synchronization with an E clock, tests the transferred data, and sets condition-code flags according to the result.

Note: Avoid using this instruction in microcontrollers without an E clock output pin, or in single-chip mode.

#### **Available Registers**

Rd: R0L to R7L, R0H to R7H

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands			Ins	tructic	on Format		No. of
Mode	Willemonic	Operanus	1st I	byte	2nd	byte	3rd byte	4th byte	States
Absolute address	MOVFPE	@aa:16, Rd	6	А	4	rd	at	DS	*

Note: \* For details, refer to the relevant microcontroller hardware manual.

#### Notes

- 1. This instruction cannot be used with addressing modes other than the above, and cannot transfer word data or longword data.
- 2. The number of states required for execution is variable. For details, refer to the relevant microcontroller hardware manual.

## Renesas

#### Move Data with E Clock

#### 2.2.41 MOVTPE

#### **MOVTPE (MOVe To Peripheral with E clock)**

#### Operation

 $Rs \rightarrow (EAd)$ Synchronized with E clock

#### Assembly-Language Format

MOVTPE Rs, @aa:16

#### **Operand Size**

Byte

#### **Condition Code**

Ι	UI	Η	U	Ν	Ζ	V	С
				$\uparrow$	$\updownarrow$	0	

- H: Previous value remains unchanged.
- N: Set to 1 if the transferred data is negative; otherwise cleared to 0.
- Z: Set to 1 if the transferred data is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Previous value remains unchanged.

#### Description

This instruction transfers the contents of a general register Rs (source operand) to a destination location specified by a 16-bit absolute address in synchronization with an E clock, tests the transferred data, and sets condition-code flags according to the result.

Note: Avoid using this instruction in microcontrollers without an E clock output pin, or in single-chip mode.

#### **Available Registers**

Rs: R0L to R7L, R0H to R7H

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands			Ins	tructio	on Format		No. of
Mode	Willemonic	Operanus	1st I	oyte	2nd	byte	3rd byte	4th byte	States
Absolute address	MOVTPE	Rs, @aa:16	6	А	С	rs	at	)S	*

Note: \* For details, refer to the relevant microcontroller hardware manual.

#### Notes

- 1. This instruction cannot be used with addressing modes other than the above, and cannot transfer word data or longword data.
- 2. The number of states required for execution is variable. For details, refer to the relevant microcontroller hardware manual.

## RENESAS

#### Move Data with E Clock

#### 2.2.42 (1) MULXS (B)

#### MULXS (MULtiply eXtend as Signed)

#### **Multiply Signed**

Operation	Condition Code
$Rd \times Rs \rightarrow Rd$	I UI H U N Z V C
	1 + 1 +
Assembly-Language Format	H: Previous value remains unchanged.
MULXS.B Rs, Rd	N: Set to 1 if the result is negative; otherwise
	cleared to 0.
Onerend Size	Z: Set to 1 if the result is zero; otherwise
Operand Size	cleared to 0.
Byte	V: Previous value remains unchanged.
	C: Previous value remains unchanged.

#### Description

This instruction multiplies the lower 8 bits of a 16-bit register Rd (destination operand) by the contents of an 8-bit register Rs (source operand) as signed data and stores the result in the 16-bit register Rd. If Rd is one of general registers R0 to R7, Rs can be the upper part (RdH) or lower part (RdL) of Rd. The operation performed is 8 bits  $\times$  8 bits  $\rightarrow$  16 bits signed multiplication.



#### **Available Registers**

Rd: R0 to R7, E0 to E7 Rs: R0L to R7L, R0H to R7H

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands			Ins	tructic	on Forr	nat			No. of
Mode	Witterfiorfic	Operanus	1st I	byte	2nd	byte	3rd	byte	4th	byte	States
Register direct	MULXS.B	Rs, Rd	0	1	С	0	5	0	rs	rd	4*

Note: \* The number of states in the H8S/2000 CPU is 13.

A maximum of three additional states are required for execution of this instruction within three states after execution of a MAC instruction. For example, if there is a one-state instruction (such as NOP) between the MAC instruction and this instruction, this instruction will be two states longer.

The number of states may differ depending on the product. For details, refer to the relevant microcontroller hardware manual of the product in question.

#### Notes

## Renesas

#### 2.2.42 (2) MULXS (W)

#### MULXS (MULtiply eXtend as Signed)

#### **Multiply Signed**

Operation	Condition Code
$ERd \times Rs \rightarrow ERd$	I UI H U N Z V C
	$\boxed{-  -  -   \updownarrow   \updownarrow   -   -}$
Assembly-Language Format	H: Previous value remains unchanged.
MULXS.W Rs, ERd	N: Set to 1 if the result is negative; otherwise
	cleared to 0.
0 10'	Z: Set to 1 if the result is zero; otherwise
Operand Size	cleared to 0.
Word	V: Previous value remains unchanged.
	C: Previous value remains unchanged.

#### Description

This instruction multiplies the lower 16 bits of a 32-bit register ERd (destination operand) by the contents of a 16-bit register Rs (source operand) as signed data and stores the result in the 32-bit register ERd. Rs can be the upper part (Ed) or lower part (Rd) of ERd. The operation performed is 16 bits  $\times$  16 bits  $\rightarrow$  32 bits signed multiplication.



#### **Available Registers**

ERd: ER0 to ER7

Rs: R0 to R7, E0 to E7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands			Ins	tructio	on Forr	nat			No. of
Mode	WITEINOTIC	Operanus	1st I	byte	2nd	byte	3rd	byte	4th	byte	States
Register direct	MULXS.W	Rs, ERd	0	1	С	0	5	2	rs	0 erd	5*

Note: \* The number of states in the H8S/2000 CPU is 21.

A maximum of three additional states are required for execution of this instruction within three states after execution of a MAC instruction. For example, if there is a one-state instruction (such as NOP) between the MAC instruction and this instruction, this instruction will be two states longer.

The number of states may differ depending on the product. For details, refer to the relevant microcontroller hardware manual of the product in question.

#### Notes

#### 2.2.43 (1) MULXU (B)

#### MULXU (MULtiply eXtend as Unsigned)

#### Operation

 $Rd \times Rs \rightarrow Rd$ 

#### Assembly-Language Format

MULXU.B Rs, Rd

#### **Operand Size**

Byte

#### 

H: Previous value remains unchanged.

**Condition Code** 

- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Previous value remains unchanged.

#### Description

This instruction multiplies the lower 8 bits of a 16-bit register Rd (destination operand) by the contents of an 8-bit register Rs (source operand) as unsigned data and stores the result in the 16-bit register Rd. If Rd is one of general registers R0 to R7, Rs can be the upper part (RdH) or lower part (RdL) of Rd. The operation performed is 8 bits  $\times$  8 bits  $\rightarrow$  16 bits unsigned multiplication.



#### **Available Registers**

Rd: R0 to R7, E0 to E7 Rs: R0L to R7L, R0H to R7H

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands			Ins	tructio	n Format		No. of
Mode	Witemonic	Operanus	1st I	byte	2nd	byte	3rd byte	4th byte	States
Register direct	MULXU.B	Rs, Rd	5	0	rs	rd			3*

Note: \* The number of states in the H8S/2000 CPU is 12.

A maximum of three additional states are required for execution of this instruction within three states after execution of a MAC instruction. For example, if there is a one-state instruction (such as NOP) between the MAC instruction and this instruction, this instruction will be two states longer.

The number of states may differ depending on the product. For details, refer to the relevant microcontroller hardware manual of the product in question.

#### Notes

## Renesas

#### Multiply

#### 2.2.43 (2) MULXU (W)

#### MULXU (MULtiply eXtend as Unsigned)

#### Operation

 $ERd \times Rs \rightarrow ERd$ 

#### Assembly-Language Format

MULXU.W Rs, ERd

#### **Operand Size**

Word

#### Condition Code

Ι	UI	Η	U	Ν	Ζ	V	С
—	—	_		—			—

- H: Previous value remains unchanged.
- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Previous value remains unchanged.

#### Description

This instruction multiplies the lower 16 bits of a 32-bit register ERd (destination operand) by the contents of a 16-bit register Rs (source operand) as unsigned data and stores the result in the 32-bit register ERd. Rs can be the upper part (Ed) or lower part (Rd) of ERd. The operation performed is 16 bits  $\times$  16 bits  $\rightarrow$  32 bits unsigned multiplication.



#### **Available Registers**

ERd: ER0 to ER7

Rs: R0 to R7, E0 to E7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands			In	str	uctio	n Format		No. of
Mode	Willemonic	Operanus	1st I	byte	2nd	l by	/te	3rd byte	4th byte	States
Register direct	MULXU.W	Rs, ERd	5	2	rs	0	erd			4*

Note: \* The number of states in the H8S/2000 CPU is 20.

A maximum of three additional states are required for execution of this instruction within three states after execution of a MAC instruction. For example, if there is a one-state instruction (such as NOP) between the MAC instruction and this instruction, this instruction will be two states longer.

The number of states may differ depending on the product. For details, refer to the relevant microcontroller hardware manual of the product in question.

#### Notes

#### **2.2.44** (1) **NEG** (**B**)

#### NEG (NEGate)

#### **Negate Binary Signed**

Operation	Condition Code
$0 - \mathrm{Rd} \rightarrow \mathrm{Rd}$	I UI H U N Z V C
	$\boxed{-   -   \updownarrow   -   \updownarrow   \updownarrow   \updownarrow   \updownarrow }$
Assembly-Language Format NEG.B Rd Operand Size	<ul><li>H: Set to 1 if there is a borrow at bit 3; otherwise cleared to 0.</li><li>N: Set to 1 if the result is negative; otherwise cleared to 0.</li><li>Z: Set to 1 if the result is zero; otherwise cleared to 0.</li></ul>
Byte	V: Set to 1 if an overflow occurs; otherwise cleared to 0.
	C: Set to 1 if there is a borrow at bit 7; otherwise cleared to 0.

#### Description

This instruction takes the two's complement of the contents of an 8-bit register Rd (destination operand) and stores the result in the 8-bit register Rd (subtracting the register contents from H'00). If the original contents of Rd were H'80, however, the result remains H'80.

#### Available Registers

Rd: R0L to R7L, R0H to R7H

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands		No. of					
Mode	Milenonic	operanus	1st byte		2nd	byte	3rd byte	4th byte	States
Register direct	NEG.B	Rd	1	7	8	rd			1

#### Notes

An overflow occurs if the original contents of Rd were H'80.

2.2.44 (2) NEG (W)

Negate Binary Signed								
Condition Code								
$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$								
<ul><li>H: Set to 1 if there is a borrow at bit 11; otherwise cleared to 0.</li><li>N: Set to 1 if the result is negative; otherwise cleared to 0.</li></ul>								
<ul> <li>Z: Set to 1 if the result is zero; otherwise cleared to 0.</li> <li>V: Set to 1 if an overflow occurs; otherwise cleared to 0.</li> <li>C: Set to 1 if there is a borrow at bit 15; otherwise cleared to 0.</li> </ul>								

#### Description

This instruction takes the two's complement of the contents of a 16-bit register Rd (destination operand) and stores the result in the 16-bit register Rd (subtracting the register contents from H'0000). If the original contents of Rd were H'8000, however, the result remains H'8000.

#### **Available Registers**

Rd: R0 to R7, E0 to E7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands		No. of					
Mode	Willemonic	Operanus	1st byte		2nd	byte	3rd byte	4th byte	States
Register direct	NEG.W	Rd	1	7	9	rd			1

#### Notes

An overflow occurs if the original contents of Rd were H'8000.

#### 2.2.44 (3) NEG (L)

#### NEG (NEGate)

#### **Negate Binary Signed**

Operation	Condition Code											
$0 - \text{ERd} \rightarrow \text{ERd}$	$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$											
Assembly-Language Format NEG.L ERd Operand Size Longword												

#### Description

This instruction takes the two's complement of the contents of a 32-bit register ERd (destination operand) and stores the result in the 32-bit register ERd (subtracting the register contents from H'00000000). If the original contents of ERd were H'80000000, however, the result remains H'80000000.

#### **Available Registers**

ERd: ER0 to ER7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic			No. of					
Mode	Witterfiorfic	Operands	1st byte		2nd byte		3rd byte	4th byte	States
Register direct	NEG.L	ERd	1	7	В	0 erd			1

#### Notes

An overflow occurs if the original contents of ERd were H'80000000.

#### 2.2.45 NOP

#### NOP (No OPeration)

#### **No Operation**

Operation	
-----------	--

 $\text{PC} + 2 \rightarrow \text{PC}$ 

#### **Assembly-Language Format**

NOP

## **Operand Size**

#### **Condition Code**



- H: Previous value remains unchanged.
- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Previous value remains unchanged.

#### Description

This instruction only increments the program counter, causing the next instruction to be executed. The internal state of the CPU does not change.

#### **Operand Format and Number of States Required for Execution**

Addre	essing	Mnemonic	onic Operands		Instruction Format						
Mo	ode	Milenonic	operanus	1st	1st byte		byte	3rd byte	4th byte	States	
_		NOP		0	0	0	0			1	

Notes



\$ \$

U N Z V

Η

#### **2.2.46** (1) NOT (B)

<b>NOT (NOT = logical complement)</b>
---------------------------------------

#### Operation

 $\neg \operatorname{Rd} \rightarrow \operatorname{Rd}$ 

#### **Assembly-Language Format**

NOT.B Rd

#### **Operand Size**

Byte

#### Description

This instruction takes the one's complement of the contents of an 8-bit register Rd (destination operand) and stores the result in the 8-bit register Rd.

#### **Available Registers**

Rd: R0L to R7L, R0H to R7H

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands		No. of					
Mode	Milenonie	operanus	1st byte		2nd	byte	3rd byte	4th byte	States
Register direct	NOT.B	Rd	1	7	0	rd			1

Notes

#### Rev. 4.00 Feb 24, 2006 page 159 of 322 REJ09B0139-0400

## RENESAS

t	H:	Previous value remains unchanged.
	N:	Set to 1 if the result is negative; otherwise
		cleared to 0.
	Z:	Set to 1 if the result is zero; otherwise
		cleared to 0.
	V:	Always cleared to 0.
	C:	Previous value remains unchanged.

**Condition Code** 

I UI

#### Logical Complement

0

С

#### 2.2.46 (2) NOT (W)

#### **NOT (NOT = logical complement)**

#### Operation

 $\neg \operatorname{Rd} \rightarrow \operatorname{Rd}$ 

#### Assembly-Language Format

NOT.W Rd

#### **Operand Size**

Word

#### Condition Code

Ι	UI	Η	U	Ν	Ζ	V	С
—				$\uparrow$	$\updownarrow$	0	

- H: Previous value remains unchanged.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Previous value remains unchanged.

#### Description

This instruction takes the one's complement of the contents of a 16-bit register Rd (destination operand) and stores the result in the 16-bit register Rd.

#### **Available Registers**

Rd: R0 to R7, E0 to E7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic Oper			No. of					
Mode	Milenonie	operando	1st byte		2nd	byte	3rd byte	4th byte	States
Register direct	NOT.W	Rd	1	7	1	rd			1

Notes

### RENESAS

## Logical Complement

2.2.46 (3) NOT (L)

<b>NOT</b> ( <b>NOT</b> = logical complement)	NOT	(NOT	= logical	complement)	
---	-----	------	-----------	-------------	--

#### Operation

 $\neg$  ERd  $\rightarrow$  ERd

#### **Assembly-Language Format**

NOT.L ERd

#### **Operand Size**

Longword

#### Description

This instruction takes the one's complement of the contents of a 32-bit register ERd (destination operand) and stores the result in the 32-bit register ERd.

#### **Available Registers**

ERd: ER0 to ER7

#### **Operand Format and Number of States Required for Execution**

Addressing Mode	Mnemonic	Operands		No. of							
	Milemonie	operando	1st	byte	2nc	l by	/te	3rd byte	4th byte	States	
Register direct	NOT.L	ERd	1	7	3	0	erd			1	

Notes

Ι	UI	Н	U	Ν	Z	V	С
—				\$	$\updownarrow$	0	—

- H: Previous value remains unchanged.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Previous value remains unchanged.

## Condition Code



## Logical Complement

2.2.47 (1) OR (B)

#### **OR** (inclusive **OR** logical)

#### Logical OR

Operation	Condition Code
$Rd \lor (EAs) \rightarrow Rd$	$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$
Assembly-Language Format OR.B <eas>, Rd</eas>	<ul><li>H: Previous value remains unchanged.</li><li>N: Set to 1 if the result is negative; otherwise cleared to 0.</li></ul>
	Z: Set to 1 if the result is zero; otherwise cleared to 0.
Operand Size	<ul><li>V: Always cleared to 0.</li><li>C: Previous value remains unchanged.</li></ul>

#### Description

Byte

This instruction ORs the source operand with the contents of an 8-bit register Rd (destination operand) and stores the result in the 8-bit register Rd.

#### **Available Registers**

Rd: R0L to R7L, R0H to R7H Rs: R0L to R7L, R0H to R7H

#### **Operand Format and Number of States Required for Execution**

Addressing Mode	Mnemonic	Operands		No. of					
	Millemonic	Operatios	1st byte		2nd byte		3rd byte	4th byte	States
Immediate	OR.B	#xx:8, Rd	С	rd	IMM				1
Register direct	OR.B	Rs, Rd	1	4	rs	rd			1

#### Notes



Logical OR

#### 2.2.47 (2) OR (W)

#### **OR** (inclusive **OR** logical)

#### Operation

 $Rd \lor (EAs) \rightarrow Rd$ 

#### **Assembly-Language Format**

OR.W <EAs>, Rd

#### **Operand Size**

Word

#### Description

This instruction ORs the source operand with the contents of a 16-bit register Rd (destination operand) and stores the result in the 16-bit register Rd.

#### **Available Registers**

Rd: R0 to R7, E0 to E7 Rs: R0 to R7, E0 to E7

#### **Operand Format and Number of States Required for Execution**

Addressing Mode	Mnemonic	Operands		No. of					
	Witemonic	Operanus	1st byte		2nd byte		3rd byte	4th byte	States
Immediate	OR.W	#xx:16, Rd	7	9	4	rd	IMM		2
Register direct	OR.W	Rs, Rd	6	4	rs	rd			1

Notes

#### Rev. 4.00 Feb 24, 2006 page 163 of 322 REJ09B0139-0400

## RENESAS

## H: Previous value remains unchanged.

- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Always cleared to 0.

**Condition Code** 

C: Previous value remains unchanged.



2.2.47 (3) OR (L)

#### **OR** (inclusive **OR** logical)

#### Logical OR

Operation	Condition Code							
$\operatorname{ERd} \lor (\operatorname{EAs}) \to \operatorname{ERd}$	$\begin{array}{c ccccccccccccccccccccccccccccccccccc$							
Assembly-Language Format OR.L <eas>, ERd</eas>	<ul><li>H: Previous value remains unchanged.</li><li>N: Set to 1 if the result is negative; otherwise cleared to 0.</li><li>Z: Set to 1 if the result is zero; otherwise cleared to 0.</li></ul>							
Operand Size	Cleared to 0. V: Always cleared to 0. C: Previous value remains unchanged.							

Longword

#### Description

This instruction ORs the source operand with the contents of a 32-bit register ERd (destination operand) and stores the result in the 32-bit register ERd.

#### **Available Registers**

ERd: ER0 to ER7 ERs: ER0 to ER7

#### **Operand Format and Number of States Required for Execution**

Addressing Mode Mr	Mnemonic	Operands		Instruction Format								
	whenome operands	Operands	1st I	byte	2nd	byte	3rd byte	4th byte	5th byte	6th byte	States	
Immediate	OR.L	#xx:32, ERd	7	A	4	0 erd		IN	М		3	
Register direct	OR.L	ERs, ERd	0	1	F	0	6 4	0 ers 0 erd			2	

#### Notes


#### 2.2.48 (1) ORC

#### Operation

 $\mathrm{CCR} \lor \#\mathrm{IMM} \to \mathrm{CCR}$ 

#### **Assembly-Language Format**

ORC #xx:8, CCR

#### **Operand Size**

Byte

#### **Condition Code**

Ι	UI	Н	U	Ν	Ζ	V	С
$\updownarrow$	\$						

I: Stores the corresponding bit of the result.

UI: Stores the corresponding bit of the result.

H: Stores the corresponding bit of the result.

U: Stores the corresponding bit of the result.

N: Stores the corresponding bit of the result.

Z: Stores the corresponding bit of the result.

V: Stores the corresponding bit of the result.

C: Stores the corresponding bit of the result.

#### Description

This instruction ORs the contents of the condition-code register (CCR) with immediate data and stores the result in the condition-code register. No interrupt requests, including NMI, are accepted immediately after execution of this instruction.

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic		No. of					
Mode	Milenonic	Operands	1st I	byte	2nd byte	3rd byte	4th byte	States
Immediate	ORC	#xx:8, CCR	0	4	IMM			1

Notes

# RENESAS

### Logical OR with CCR

#### 2.2.48 (2) ORC

#### **ORC** (inclusive **OR** Control register)

#### Operation

 $\text{EXR} \lor \#\text{IMM} \to \text{EXR}$ 

#### **Assembly-Language Format**

ORC #xx:8, EXR

#### Logical OR with EXR

#### **Condition Code**



- H: Stores the corresponding bit of the result.
- N: Stores the corresponding bit of the result.
- Z: Stores the corresponding bit of the result.
- V: Stores the corresponding bit of the result.
- C: Stores the corresponding bit of the result.

#### **Operand Size**

Byte

#### Description

This instruction ORs the contents of the extended control register (EXR) with immediate data and stores the result in the extended control register. No interrupt requests, including NMI, are accepted for three states after execution of this instruction.

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands		No. of							
Mode	Millenionic		1st I	oyte	2nd	byte	3rd	byte	4th byte	States	
Immediate	ORC #xx:8, EXR		0	1	4	1	0	4	IMM	2	

Notes



**Pop Data from Stack** 

#### 2.2.49 (1) POP (W)

#### POP (POP data)

#### Operation

 $@SP+ \rightarrow Rn$ 

#### **Assembly-Language Format**

POP.W Rn

#### **Operand Size**

Word

#### Description

This instruction restores data from the stack to a 16-bit general register Rn, tests the restored data, and sets condition-code flags according to the result.

#### **Available Registers**

Rn: R0 to R7. E0 to E7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic		No. of								
Mode	Millenionic	Operands	1st	byte	2nd	byte	3rd byte	4th byte	States		
_	POP.W	Rn	6	D	7 rn				3		

#### Notes

POP.W Rn is identical to MOV.W @SP+, Rn.

	1	UI	Н	U	Ν	Z	V	C				
	_	—			$\updownarrow$	$\updownarrow$	0	—				
 H:	Prev	ious	valu	e rem	nains	unch	nange	ed.				
N:		Set to 1 if the transferred data is negative; otherwise cleared to 0.										
7.	Sat t	<u> </u>	ftha	trong	forro	d dat	o ic	zoro.				

- Z: Set to 1 if the transferred data is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Previous value remains unchanged.

# **Condition Code**



#### 2.2.49 (2) POP (L)

#### POP (POP data)

#### **Pop Data from Stack**

Operation	Condition Code							
$@SP+ \rightarrow ERn$	$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$							
Assembly-Language Format POP.L ERn	<ul> <li>H: Previous value remains unchanged.</li> <li>N: Set to 1 if the transferred data is negative; otherwise cleared to 0.</li> <li>Z: Set to 1 if the transferred data is zero; otherwise cleared to 0.</li> </ul>							
Operand SizeV: Always cleared to 0.LongwordC: Previous value remains unchanged.								

## Description

This instruction restores data from the stack to a 32-bit general register ERn, tests the restored data, and sets condition-code flags according to the result.

#### **Available Registers**

ERn: ER0 to ER7

#### **Operand Format and Number of States Required for Execution**

Ad	Addressing	Mnemonic	Operands		No. of							
	Mode	Millenionic		1st I	byte	2nd	byte	3rd	byte	4th	byte	States
	—	POP.L	ERn	0	1	0	0	6	D	7	0 ern	5

#### Notes

POP.L ERn is identical to MOV.L @SP+, ERn.



#### 2.2.50 (1) PUSH (W)

#### PUSH (PUSH data)

#### Push Data on Stack

Operation	Condition Code								
$Rn \rightarrow @-SP$	$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$								
Assembly-Language Format	<ul> <li>H: Previous value remains unchanged.</li> <li>N: Set to 1 if the transferred data is negative; otherwise cleared to 0.</li> <li>Z: Set to 1 if the transferred data is zero; otherwise cleared to 0.</li> </ul>								
<b>Operand Size</b> Word	<ul><li>V: Always cleared to 0.</li><li>C: Previous value remains unchanged.</li></ul>								

#### Description

This instruction saves data from a 16-bit register Rn onto the stack, tests the saved data, and sets condition-code flags according to the result.

#### **Available Registers**

Rn: R0 to R7, E0 to E7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic Operands			No. of							
Mode	witterflorflo	Operanus	1st I	byte	2nd	byte	3rd byte	4th byte	States		
_	PUSH.W	Rn	6	D	F	rn			3		

#### Notes

- 1. PUSH.W Rn is identical to MOV.W Rn, @-SP.
- 2. When PUSH.W R7 or PUSH.W E7 is executed, the value saved on the stack is the R7 or E7 value after effective address calculation (after ER7 is decremented by 2).

#### 2.2.50 (2) PUSH (L)

#### PUSH (PUSH data)

#### Push Data on Stack

Operation	Condition Code							
$ERn \rightarrow @-SP$	$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$							
Assembly-Language Format	H: Previous value remains unchanged.							
Assembly-Language Format	N: Set to 1 if the transferred data is negative; otherwise cleared to 0.							
	Z: Set to 1 if the transferred data is zero; otherwise cleared to 0.							
Operand Size	V: Always cleared to 0.							
Longword	C: Previous value remains unchanged.							

#### Description

This instruction pushes data from a 32-bit register ERn onto the stack, tests the saved data, and sets condition-code flags according to the result.

#### **Available Registers**

ERn: ER0 to ER7

#### **Operand Format and Number of States Required for Execution**

	Addressing Mode	Mnemonic	Operands		No. of							
		Milenonie		1st I	byte	2nd	byte	3rd	byte	4th	byte	States
		PUSH.L	ERn	0	1	0	0	6	D	F	0 ern	5

#### Notes

- 1. PUSH.L ERn is identical to MOV.L ERn, @-SP.
- 2. When PUSH.L ER7 is executed, the value saved on the stack is the ER7 value after effective address calculation (after ER7 is decremented by 4).

Ζ

U Ν

H: Previous value remains unchanged. N: Set to 1 if the result is negative; otherwise

Z: Set to 1 if the result is zero; otherwise

C: Receives the previous value in bit 7.

ᡗ <u>↑</u>

Η

#### 2.2.51 (1) ROTL (B)

#### **ROTL (ROTate Left)**

#### Operation

Rd (left rotation)  $\rightarrow$  Rd

#### **Assembly-Language Format**

ROTL.B Rd

#### **Operand Size**

Byte

#### Description

This instruction rotates the bits in an 8-bit register Rd (destination operand) one bit to the left. The most significant bit (bit 7) is rotated to the least significant bit (bit 0), and also copied to the carry flag.

**Condition Code** 

cleared to 0.

cleared to 0.

V: Always cleared to 0.

I UI



#### **Available Registers**

Rd: R0L to R7L, R0H to R7H

#### **Operand Format and Number of States Required for Execution**

Addressing Mode	Mnemonic	Inemonic Operands		Instruction Format							
	Witterfiorfic	Operations	1st I	byte	2nd	byte	3rd byte	4th byte	States		
Register direct	ROTL.B	Rd	1	2	8	rd			1		

#### Notes

# RENESAS

#### Rotate

С

1 0

#### 2.2.51 (2) ROTL (B)

#### **ROTL (ROTate Left)**

Operation	Condition Code
Rd (left rotation) $\rightarrow$ Rd	$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$
Assembly-Language Format	H: Previous value remains unchanged.
ROTL.B #2, Rd	N: Set to 1 if the result is negative; otherwise cleared to 0.
	Z: Set to 1 if the result is zero; otherwise
Operand Size	cleared to 0.
Byte	V: Always cleared to 0.
2,00	C: Receives the previous value in bit 6.

#### Description

This instruction rotates the bits in an 8-bit register Rd (destination operand) two bits to the left. The most significant two bits (bits 7 and 6) are rotated to the least significant two bits (bits 1 and 0), and bit 6 is also copied to the carry flag.



#### **Available Registers**

Rd: R0L to R7L, R0H to R7H

#### **Operand Format and Number of States Required for Execution**

Addressing Mode	Mnemonic	Operands		Instruction Format							
	WITEINOTIC	Operanus	1st I	byte	2nd	byte	3rd byte	4th byte	States		
Register direct	ROTL.B	#2, Rd	1	2	С	rd			1		

#### Notes

#### 2.2.51 (3) ROTL (W)

#### ROTL (ROTate Left)

#### Operation

Rd (left rotation)  $\rightarrow$  Rd

#### **Assembly-Language Format**

ROTL.W Rd

#### **Operand Size**

Word

# Condition CodeIUIHUNZVC--- $\uparrow$ $\uparrow$ 0 $\uparrow$

- H: Previous value remains unchanged.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Receives the previous value in bit 15.

# Description

This instruction rotates the bits in a 16-bit register Rd (destination operand) one bit to the left. The most significant bit (bit 15) is rotated to the least significant bit (bit 0), and also copied to the carry flag.



#### **Available Registers**

Rd: R0 to R7, E0 to E7

#### **Operand Format and Number of States Required for Execution**

Addressing Mode	Mnemonic	Mnemonic Operands			Instruction Format							
	witterflorflo	Operations	1st I	byte	2nd	byte	3rd byte	4th byte	States			
Register direct	ROTL.W	Rd	1	2	9	rd			1			

#### Notes

# RENESAS

#### Rotate

#### 2.2.51 (4) ROTL (W)

#### **ROTL (ROTate Left)**

Operation	Condition Code								
Rd (left rotation) $\rightarrow$ Rd	$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$								
Assembly-Language Format	H: Previous value remains unchanged.								
ROTL.W #2, Rd	N: Set to 1 if the result is negative; otherwise cleared to 0.								
	Z: Set to 1 if the result is zero; otherwise								
Operand Size	cleared to 0.								
Word	<ul><li>V: Always cleared to 0.</li><li>C: Receives the previous value in bit 14.</li></ul>								

#### Description

This instruction rotates the bits in a 16-bit register Rd (destination operand) two bits to the left. The most significant two bits (bits 15 and 14) are rotated to the least significant two bits (bits 1 and 0), and bit 14 is also copied to the carry flag.



#### **Available Registers**

Rd: R0 to R7, E0 to E7

#### **Operand Format and Number of States Required for Execution**

Addressing Mode	Mnemonic	Mnemonic Operands		Instruction Format							
	Willemonic	Operatios	1st I	byte	2nd	byte	3rd byte	4th byte	States		
Register direct	ROTL.W	#2, Rd	1	2	D	rd			1		

#### Notes

#### 2.2.51 (5) ROTL (L)

#### ROTL (ROTate Left)

#### Operation

ERd (left rotation)  $\rightarrow$  ERd

#### **Assembly-Language Format**

ROTL.L ERd

#### **Operand Size**

Longword

# Condition CodeIUIHUNZVC--- $\uparrow$ $\uparrow$ 0 $\uparrow$

- H: Previous value remains unchanged.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Receives the previous value in bit 31.

# Description

This instruction rotates the bits in a 32-bit register ERd (destination operand) one bit to the left. The most significant bit (bit 31) is rotated to the least significant bit (bit 0), and also copied to the carry flag.



#### **Available Registers**

ERd: ER0 to ER7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic Operands			Instruction Format							
Mode	Witterfiorfic	Operanus	1st byte		2nd byte		3rd byte	4th byte	States		
Register direct	ROTL.L	ERd	1	2	В	0 erd			1		

#### Notes

# RENESAS

#### Rotate

#### 2.2.51 (6) ROTL (L)

#### **ROTL (ROTate Left)**

#### Operation

ERd (left rotation)  $\rightarrow$  ERd

#### Assembly-Language Format

ROTL.L #2, ERd

#### **Operand Size**

Longword

# Condition Code

Ι	UI	Η	U	Ν	Ζ	V	С
—				$\uparrow$	$\updownarrow$	0	$\updownarrow$

- H: Previous value remains unchanged.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Receives the previous value in bit 30.

# Description

This instruction rotates the bits in a 32-bit register ERd (destination operand) two bits to the left. The most significant two bits (bits 31 and 30) are rotated to the least significant two bits (bits 1 and 0), and bit 30 is also copied to the carry flag.



#### **Available Registers**

ERd: ER0 to ER7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands		Instruction Format							
Mode	Willemonic	Operanus	1st byte		2no	d by	/te	3rd byte	4th byte	States	
Register direct	ROTL.L	#2, ERd	1	2	F	0	erd			1	

#### Notes

Ζ

U N

H: Previous value remains unchanged.N: Set to 1 if the result is negative; otherwise

Z: Set to 1 if the result is zero; otherwise

C: Receives the previous value in bit 0.

1 1

Η

#### 2.2.52 (1) ROTR (B)

#### **ROTR (ROTate Right)**

#### Operation

Rd (right rotation)  $\rightarrow$  Rd

#### **Assembly-Language Format**

ROTR.B Rd

#### **Operand Size**

Byte

#### Description

This instruction rotates the bits in an 8-bit register Rd (destination operand) one bit to the right. The least significant bit (bit 0) is rotated to the most significant bit (bit 7), and also copied to the carry flag.

**Condition Code** 

cleared to 0.

cleared to 0.

V: Always cleared to 0.

I UI



#### **Available Registers**

Rd: R0L to R7L, R0H to R7H

#### **Operand Format and Number of States Required for Execution**

Addressing Mode	Mnemonic Operands			Instruction Format							
	Witterfiorfic	Operations	1st I	byte	2nd	byte	3rd byte	4th byte	States		
Register direct	ROTR.B	Rd	1	3	8	rd			1		

#### Notes

# RENESAS

#### Rotate

С

0 1

#### 2.2.52 (2) ROTR (B)

#### **ROTR (ROTate Right)**

Operation	Condition Code								
Rd (right rotation) $\rightarrow$ Rd	$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$								
Assembly-Language Format	H: Previous value remains unchanged.								
ROTR.B #2, Rd	N: Set to 1 if the result is negative; otherwise cleared to 0.								
	Z: Set to 1 if the result is zero; otherwise								
Operand Size	cleared to 0.								
•	V: Always cleared to 0.								
Byte	C: Receives the previous value in bit 1.								

#### Description

This instruction rotates the bits in an 8-bit register Rd (destination operand) two bits to the right. The least significant two bits (bits 1 and 0) are rotated to the most significant two bits (bits 7 and 6), and bit 1 is also copied to the carry flag.



#### **Available Registers**

Rd: R0L to R7L, R0H to R7H

#### **Operand Format and Number of States Required for Execution**

Addressing Mode	Mnemonic Operands			Instruction Format							
	WITEINOTIC	Operanus	1st I	byte	2nd	byte	3rd byte	4th byte	States		
Register direct	ROTR.B	#2, Rd	1	3	С	rd			1		

#### Notes

#### 2.2.52 (3) ROTR (W)

#### **ROTR (ROTate Right)**

#### Operation

Rd (right rotation)  $\rightarrow$  Rd

#### **Assembly-Language Format**

ROTR.W Rd

#### **Operand Size**

Word

# I UI H U N Z V C — — — 1 ↓

- H: Previous value remains unchanged.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Receives the previous value in bit 0.

#### Description

This instruction rotates the bits in a 16-bit register Rd (destination operand) one bit to the right. The least significant bit (bit 0) is rotated to the most significant bit (bit 15), and also copied to the carry flag.



#### **Available Registers**

Rd: R0 to R7, E0 to E7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic Operands			Instruction Format							
Mode	Witterfiorfic	Operanus	1st byte		2nd byte		3rd byte	4th byte	States		
Register direct	ROTR.W	Rd	1	3	9	rd			1		

#### Notes

# RENESAS

#### Rotate

#### 2.2.52 (4) ROTR (W)

#### **ROTR (ROTate Right)**

Operation	Condition Code
Rd (right rotation) $\rightarrow$ Rd	$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$
Assembly-Language Format	H: Previous value remains unchanged.
ROTR.W #2, Rd	N: Set to 1 if the result is negative; otherwise cleared to 0.
	Z: Set to 1 if the result is zero; otherwise
Operand Size	cleared to 0.
•	V: Always cleared to 0.
Word	C: Receives the previous value in bit 1.

#### Description

This instruction rotates the bits in a 16-bit register Rd (destination operand) two bits to the right. The least significant two bits (bits 1 and 0) are rotated to the most significant two bits (bits 15 and 14), and bit 1 is also copied to the carry flag.



#### **Available Registers**

Rd: R0 to R7, E0 to E7

#### **Operand Format and Number of States Required for Execution**

Addressing Mode Mnemonic	Operands		No. of						
Mode	Willemonic	Operanus	1st byte		2nd byte		3rd byte	4th byte	States
Register direct	ROTR.W	#2, Rd	1	3	D	rd			1

#### Notes



#### 2.2.52 (5) ROTR (L)

#### **ROTR** (**ROTate Right**)

#### Rotate

Operation	Condition Code
ERd (right rotation) $\rightarrow$ ERd	$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$
Assembly-Language Format	H: Previous value remains unchanged.
ROTR.L ERd	N: Set to 1 if the result is negative; otherwise cleared to 0.
	Z: Set to 1 if the result is zero; otherwise
Operand Size Longword	<ul><li>cleared to 0.</li><li>V: Always cleared to 0.</li><li>C: Receives the previous value in bit 0.</li></ul>

#### Description

This instruction rotates the bits in a 32-bit register ERd (destination operand) one bit to the right. The least significant bit (bit 0) is rotated to the most significant bit (bit 31), and also copied to the carry flag.



#### **Available Registers**

ERd: ER0 to ER7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands		No. of					
Mode	Witemonic	Operanus	1st byte		2no	d byte	3rd byte	4th byte	States
Register direct	ROTR.L	ERd	1	3	В	0 erd			1

#### Notes

#### 2.2.52 (6) ROTR (L)

#### **ROTR (ROTate Right)**

#### Operation

ERd (right rotation)  $\rightarrow$  ERd

#### Assembly-Language Format

ROTR.L #2, ERd

#### **Operand Size**

Longword

# Condition Code

Ι	UI	Η	U	Ν	Ζ	V	С
—				$\uparrow$	$\updownarrow$	0	$\updownarrow$

- H: Previous value remains unchanged.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Receives the previous value in bit 1.

#### Description

This instruction rotates the bits in a 32-bit register ERd (destination operand) two bits to the right. The least significant two bits (bits 1 and 0) are rotated to the most significant two bits (bits 31 and 30), and bit 1 is also copied to the carry flag.



#### **Available Registers**

ERd: ER0 to ER7

#### **Operand Format and Number of States Required for Execution**

Addressing	Operands			In	No. of					
Mode	Willemonic	Operanus	1st byte		2nd byte		/te	3rd byte	4th byte	States
Register direct	ROTR.L	#2, ERd	1	3	F	0	erd			1

#### Notes

#### 2.2.53 (1) ROTXL (B)

#### **ROTXL (ROTate with eXtend carry Left)**

#### Operation

Rd (left rotation through carry flag)  $\rightarrow$  Rd

#### **Assembly-Language Format**

ROTXL.B Rd

#### **Operand Size**

Byte

#### **Condition Code**



- H: Previous value remains unchanged.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Receives the previous value in bit 7.

#### Description

This instruction rotates the bits in an 8-bit register Rd (destination operand) one bit to the left through the carry flag. The carry flag is rotated into the least significant bit (bit 0). The most significant bit (bit 7) rotates into the carry flag.



#### **Available Registers**

Rd: R0L to R7L, R0H to R7H

#### **Operand Format and Number of States Required for Execution**

Addressing Mode Mnemonic	Operands		No. of						
Mode	Witterfiorfic	Operanus	1st byte		2nd byte		3rd byte	4th byte	States
Register direct	ROTXL.B	Rd	1	2	0	rd			1

#### Notes

# RENESAS

#### 2.2.53 (2) ROTXL (B)

#### **ROTXL (ROTate with eXtend carry Left)**

#### Operation

Rd (left rotation through carry flag)  $\rightarrow$  Rd

#### Assembly-Language Format

ROTXL.B #2, Rd

#### **Operand Size**

Byte

#### **Condition Code**

Ι	UI	Η	U	Ν	Ζ	V	С
—				$\Leftrightarrow$	$\updownarrow$	0	$\leftrightarrow$

- H: Previous value remains unchanged.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Receives the previous value in bit 6.

#### Description

This instruction rotates the bits in an 8-bit register Rd (destination operand) two bits to the left through the carry flag. The carry flag rotates into bit 1, bit 7 rotates into bit 0, and bit 6 rotates into the carry flag.



#### **Available Registers**

Rd: R0L to R7L, R0H to R7H

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands		No. of					
Mode	Millenionic	operando	1st byte		2nd byte		3rd byte	4th byte	States
Register direct	ROTXL.B	#2, Rd	1	2	4	rd			1

#### Notes

# RENESAS

#### 2.2.53 (3) ROTXL (W)

#### **ROTXL (ROTate with eXtend carry Left)**

#### Operation

Rd (left rotation through carry flag)  $\rightarrow$  Rd

#### **Assembly-Language Format**

ROTXL.W Rd

#### **Operand Size**

Word

#### **Condition Code**



- H: Previous value remains unchanged.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Receives the previous value in bit 15.

#### Description

This instruction rotates the bits in a 16-bit register Rd (destination operand) one bit to the left through the carry flag. The carry flag is rotated into the least significant bit (bit 0). The most significant bit (bit 15) rotates into the carry flag.



#### **Available Registers**

Rd: R0 to R7, E0 to E7

#### **Operand Format and Number of States Required for Execution**

Addressing Mode Mnemonic	Operands		No. of						
Mode	Witterfiorfic	Operanus	1st byte		2nd byte		3rd byte	4th byte	States
Register direct	ROTXL.W	Rd	1	2	1	rd			1

#### Notes

# RENESAS

#### 2.2.53 (4) ROTXL (W)

#### **ROTXL (ROTate with eXtend carry Left)**

#### Operation

Rd (left rotation through carry flag)  $\rightarrow$  Rd

#### Assembly-Language Format

ROTXL.W #2, Rd

#### **Operand Size**

Word

#### **Condition Code**

Ι	UI	Η	U	Ν	Ζ	V	С
				$\uparrow$	$\updownarrow$	0	$\leftrightarrow$

- H: Previous value remains unchanged.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Receives the previous value in bit 14.

#### Description

This instruction rotates the bits in a 16-bit register Rd (destination operand) two bits to the left through the carry flag. The carry flag rotates into bit 1, bit 15 rotates into bit 0, and bit 14 rotates into the carry flag.



#### **Available Registers**

Rd: R0 to R7, E0 to E7

#### **Operand Format and Number of States Required for Execution**

Addressing Mode	Mnemonic Operands			Instruction Format							
	Millemonic	Operanus	1st	byte	2nd	byte	3rd byte	4th byte	States		
Register direct	ROTXL.W	#2, Rd	1	2	5	rd			1		

#### Notes

# RENESAS

#### 2.2.53 (5) ROTXL (L)

#### **ROTXL (ROTate with eXtend carry Left)**

#### Operation

ERd (left rotation through carry flag)  $\rightarrow$  ERd

#### **Assembly-Language Format**

ROTXL.L ERd

#### **Operand Size**

Longword

#### **Condition Code**



- H: Previous value remains unchanged.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Receives the previous value in bit 31.

#### Description

This instruction rotates the bits in a 32-bit register ERd (destination operand) one bit to the left through the carry flag. The carry flag is rotated into the least significant bit (bit 0). The most significant bit (bit 31) rotates into the carry flag.



#### **Available Registers**

ERd: ER0 to ER7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands			In	structio	n Format		No. of
Mode	Witterfiorfic	Operanus	1st byte		2no	d byte	3rd byte	4th byte	States
Register direct	ROTXL.L	ERd	1	2	3	0 erd			1

#### Notes

# RENESAS

# 2.2.53 (6) ROTXL (L)

## **ROTXL (ROTate with eXtend carry Left)**

#### Operation

ERd (left rotation through carry flag)  $\rightarrow$  ERd

## **Assembly-Language Format**

ROTXL.L #2, ERd

### **Operand Size**

Longword

# **Condition Code**

Ι	UI	Η	U	Ν	Ζ	V	С
				$\uparrow$	$\updownarrow$	0	$\uparrow$

- H: Previous value remains unchanged.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Receives the previous value in bit 30.

# Description

This instruction rotates the bits in a 32-bit register ERd (destination operand) two bits to the left through the carry flag. The carry flag rotates into bit 1, bit 31 rotates into bit 0, and bit 30 rotates into into the carry flag.



# **Available Registers**

ERd: ER0 to ER7

# **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	nemonic Operands		Instruction Format							
Mode	Willemonic	Operanus	1st I	byte	2n	d byte	3rd byte	4th byte	States		
Register direct	ROTXL.L	#2, ERd	1	2	7	0 erd			1		

#### Notes

# RENESAS

#### 2.2.54 (1) ROTXR (B)

#### **ROTXR** (**ROTate with eXtend carry Right**)

#### Operation

Rd (right rotation through carry flag)  $\rightarrow$  Rd

#### **Assembly-Language Format**

ROTXR.B Rd

#### **Operand Size**

Byte

#### **Condition Code**



- H: Previous value remains unchanged.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Receives the previous value in bit 0.

#### Description

This instruction rotates the bits in an 8-bit register Rd (destination operand) one bit to the right through the carry flag. The carry flag is rotated into the most significant bit (bit 7). The least significant bit (bit 0) rotates into the carry flag.



#### **Available Registers**

Rd: R0L to R7L, R0H to R7H

#### **Operand Format and Number of States Required for Execution**

Addressing Mode Mn	Mnemonic Operands		Instruction Format							
		Operanus	1st I	byte	2nd	byte	3rd byte	4th byte	States	
Register direct	ROTXR.B	Rd	1	3	0	rd			1	

#### Notes

# RENESAS

#### 2.2.54 (2) ROTXR (B)

#### **ROTXR** (ROTate with eXtend carry Right)

#### Operation

Rd (right rotation through carry flag)  $\rightarrow$  Rd

#### Assembly-Language Format

ROTXR.B #2, Rd

#### **Operand Size**

Byte

#### **Condition Code**

Ι	UI	Η	U	Ν	Ζ	V	С
				$\uparrow$	$\updownarrow$	0	$\leftrightarrow$

- H: Previous value remains unchanged.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Receives the previous value in bit 1.

#### Description

This instruction rotates the bits in an 8-bit register Rd (destination operand) two bits to the right through the carry flag. The carry flag rotates into bit 6, bit 0 rotates into bit 7, and bit 1 rotates into the carry flag.



#### **Available Registers**

Rd: R0L to R7L, R0H to R7H

#### **Operand Format and Number of States Required for Execution**

Addressing Mode	Mnemonic	Operands		Instruction Format							
	WITEITIOTTIC		1st I	byte	2nd	byte	3rd byte	4th byte	States		
Register direct	ROTXR.B	#2, Rd	1	3	4	rd			1		

#### Notes

# RENESAS

#### 2.2.54 (3) ROTXR (W)

#### **ROTXR** (**ROTate with eXtend carry Right**)

#### Operation

Rd (right rotation through carry flag)  $\rightarrow$  Rd

#### Assembly-Language Format

ROTXR.W Rd

#### **Operand Size**

Word

#### **Condition Code**



- H: Previous value remains unchanged.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Receives the previous value in bit 0.

#### Description

This instruction rotates the bits in a 16-bit register Rd (destination operand) one bit to the right through the carry flag. The carry flag is rotated into the most significant bit (bit 15). The least significant bit (bit 0) rotates into the carry flag.



#### **Available Registers**

Rd: R0 to R7, E0 to E7

#### **Operand Format and Number of States Required for Execution**

Addressing Mode M	Mnemonic Operands			Instruction Format							
	WITEINOTIC	Operanus	1st I	byte	2nd	byte	3rd byte	4th byte	States		
Register direct	ROTXR.W	Rd	1	3	1	rd			1		

#### Notes

# RENESAS

#### 2.2.54 (4) ROTXR (W)

#### **ROTXR (ROTate with eXtend carry Right)**

#### Operation

Rd (right rotation through carry flag)  $\rightarrow$  Rd

#### Assembly-Language Format

ROTXR.W #2, Rd

#### **Operand Size**

Word

#### **Condition Code**

Ι	UI	Η	U	Ν	Ζ	V	С
				$\uparrow$	$\updownarrow$	0	$\leftrightarrow$

- H: Previous value remains unchanged.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Receives the previous value in bit 1.

#### Description

This instruction rotates the bits in a 16-bit register Rd (destination operand) two bits to the right through the carry flag. The carry flag rotates into bit 14, bit 0 rotates into bit 15, and bit 1 rotates into the carry flag.



#### **Available Registers**

Rd: R0 to R7, E0 to E7

#### **Operand Format and Number of States Required for Execution**

Addressing Mode	Mnemonic O	Operands		Instruction Format							
		Operanus	1st I	byte	2nd	byte	3rd byte	4th byte	States		
Register direct	ROTXR.W	#2, Rd	1	3	5	rd			1		

#### Notes

# RENESAS

#### 2.2.54 (5) ROTXR (L)

#### **ROTXR (ROTate with eXtend carry Right)**

#### Operation

ERd (right rotation through carry flag)  $\rightarrow$  ERd

#### **Assembly-Language Format**

ROTXR.L ERd

#### **Operand Size**

Longword

#### **Condition Code**



- H: Previous value remains unchanged.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Receives the previous value in bit 0.

### Description

This instruction rotates the bits in a 32-bit register ERd (destination operand) one bit to the right through the carry flag. The carry flag is rotated into the most significant bit (bit 31). The least significant bit (bit 0) rotates into the carry flag.



#### **Available Registers**

ERd: ER0 to ER7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Mnemonic Operands		Instruction Format						
Mode	winemonic	Operanus	1st I	byte	2no	d byte	3rd byte	4th byte	States	
Register direct	ROTXR.L	ERd	1	3	3	0 erd			1	

#### Notes

# RENESAS

# 2.2.54 (6) ROTXR (L)

## **ROTXR (ROTate with eXtend carry Right)**

#### Operation

ERd (right rotation through carry flag)  $\rightarrow$  ERd

### **Assembly-Language Format**

ROTXR.L #2, ERd

### **Operand Size**

Longword

# **Condition Code**

Ι	UI	Η	U	Ν	Ζ	V	С
_				$\uparrow$	$\updownarrow$	0	$\updownarrow$

- H: Previous value remains unchanged.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Receives the previous value in bit 1.

# Description

This instruction rotates the bits in a 32-bit register ERd (destination operand) two bits to the right through the carry flag. The carry flag rotates into bit 30, bit 0 rotates into bit 31, and bit 1 rotates into the carry flag.



# **Available Registers**

ERd: ER0 to ER7

# **Operand Format and Number of States Required for Execution**

Addressing Mode	Mnemonic	Operands		No. of					
	Willemonic		1st I	byte	2n	d byte	3rd byte	4th byte	States
Register direct	ROTXR.L	#2, ERd	1	3	7	0 erd			1

#### Notes

# RENESAS

#### 2.2.55 RTE

#### Operation

- When EXR is invalid  $@SP+ \rightarrow CCR$  $@SP+ \rightarrow PC$
- When EXR is valid
   @SP+ → EXR
   @SP+ → CCR
  - $@SP+ \rightarrow PC$

#### **Assembly-Language Format**

RTE

#### **Operand Size**

#### **Return from Exception Handling**

Cor	Condition Code									
	I UI H U N Z V C									
	$\begin{array}{ c c c c c c c c c c c c c c c c c c c$									
I:	Restored from the corresponding bit on									
	the stack.									
UI:	Restored from the corresponding bit on									
	the stack.									
H:	Restored from the corresponding bit on									
	the stack.									
U:	Restored from the corresponding bit on									
	the stack.									
N:	Restored from the corresponding bit on									
	the stack.									
Z:										
	the stack.									
V:	Restored from the corresponding bit on									
	the stack.									
C:										
	the stack.									

#### Description

This instruction returns from an exception-handling routine by restoring the EXR, condition-code register (CCR) and program counter (PC) from the stack. Program execution continues from the address restored to the program counter. The CCR and PC contents at the time of execution of this instruction are lost. If the extended control regiser (EXR) is valid, it is also restored (and the existing EXR contents are lost).

#### **Operand Format and Number of States Required for Execution**

Addressing Mode	Mnemonic Operands	Operands		No. of					
		Operations	1st	byte	2nd	byte	3rd byte	4th byte	States
_	RTE		5	6	7	0			5*

Note: \* Six states when EXR is valid.

# Renesas

## **RTE (ReTurn from Exception)**

#### **Return from Exception Handling**

#### Notes

The stack structure differs between normal mode and advanced mode.





#### 2.2.56 RTS

#### **RTS (ReTurn from Subroutine)**

#### Operation

 $@SP+ \rightarrow PC$ 

#### **Assembly-Language Format**

RTS

#### **Operand Size**

**Return from Subroutine** 



- H: Previous value remains unchanged.
- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Previous value remains unchanged.

#### Description

This instruction returns from a subroutine by restoring the program counter (PC) from the stack. Program execution continues from the address restored to the program counter. The PC contents at the time of execution of this instruction are lost.

#### **Operand Format and Number of States Required for Execution**

Addressing Mode	Mnemonic	Operands			Instr	No. of States				
			1st I	byte	2nd byte		2nd byte 3rd byte		Normal	Advanced
_	RTS		5	4	7	0			4	5

#### Notes

The stack structure and number of states required for execution differ between normal mode and advanced mode. In normal mode, only the lower 16 bits of the program counter are restored.



Renesas

#### 2.2.57 (1) SHAL (B)

#### SHAL (SHift Arithmetic Left)

#### **Shift Arithmetic**

Operation	Condition Code								
Rd (left arithmetic shift) $\rightarrow$ Rd	$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$								
Assembly-Language Format	H: Previous value remains unchanged.								
SHAL.B Rd	N: Set to 1 if the result is negative; otherwise cleared to 0.								
	Z: Set to 1 if the result is zero; otherwise								
Operand Size	cleared to 0.								
•	V: Set to 1 if an overflow occurs; otherwise								
Byte	cleared to 0.								
	C: Receives the previous value in bit 7.								

#### Description

This instruction shifts the bits in an 8-bit register Rd (destination operand) one bit to the left. The most significant bit (bit 7) shifts into the carry flag. The least significant bit (bit 0) is cleared to 0.



#### Available Registers

Rd: R0L to R7L, R0H to R7H

#### **Operand Format and Number of States Required for Execution**

Addressing Mode	Mnemonic	Operands		No. of					
			1st	byte	2nd	byte	3rd byte	4th byte	States
Register direct	SHAL.B	Rd	1	0	8	rd			1

#### Notes

The SHAL instruction differs from the SHLL instruction in its effect on the overflow flag.



#### 2.2.57 (2) SHAL (B)

#### SHAL (SHift Arithmetic Left)

#### Operation

Rd (left arithmetic shift)  $\rightarrow$  Rd

#### **Assembly-Language Format**

SHAL.B #2, Rd

#### **Operand Size**

Byte

#### 

H: Previous value remains unchanged.

**Condition** Code

- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Set to 1 if an overflow occurs; otherwise cleared to 0.
- C: Receives the previous value in bit 6.

#### Description

This instruction shifts the bits in an 8-bit register Rd (destination operand) two bits to the left. Bit 6 shifts into the carry flag. Bits 0 and 1 are cleared to 0.



#### **Available Registers**

Rd: R0L to R7L, R0H to R7H

#### **Operand Format and Number of States Required for Execution**

Addressing Mode	Mnemonic	Operands		No. of					
			1st	byte	2nd	byte	3rd byte	4th byte	States
Register direct	SHAL.B	#2, Rd	1	0	С	rd			1

#### Notes

The SHAL instruction differs from the SHLL instruction in its effect on the overflow flag.

# RENESAS

#### Shift Arithmetic

#### 2.2.57 (3) SHAL (W)

#### SHAL (SHift Arithmetic Left)

#### Shift Arithmetic

Operation	Condition Code							
Rd (left arithmetic shift) $\rightarrow$ Rd	$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$							
Assembly-Language Format	H: Previous value remains unchanged.							
SHAL.W Rd	N: Set to 1 if the result is negative; otherwise cleared to 0.							
	Z: Set to 1 if the result is zero; otherwise							
Operand Size	cleared to 0.							
•	V: Set to 1 if an overflow occurs; otherwise							
Word	cleared to 0.							
	C: Receives the previous value in bit 15.							

#### Description

This instruction shifts the bits in a 16-bit register Rd (destination operand) one bit to the left. The most significant bit (bit 15) shifts into the carry flag. The least significant bit (bit 0) is cleared to 0.



#### **Available Registers**

Rd: R0 to R7, E0 to E7

#### **Operand Format and Number of States Required for Execution**

Addressing Mode	Mnemonic	Operands		No. of					
		Operanus	1st	byte	2nd	byte	3rd byte	4th byte	States
Register direct	SHAL.W	Rd	1	0	9	rd			1

#### Notes

The SHAL instruction differs from the SHLL instruction in its effect on the overflow flag.
#### 2.2.57 (4) SHAL (W)

#### SHAL (SHift Arithmetic Left)

#### Operation

Rd (left arithmetic shift)  $\rightarrow$  Rd

#### **Assembly-Language Format**

SHAL.W #2, Rd

#### **Operand Size**

Word

# Condition CodeIUIHUNZVC--++<td

- H: Previous value remains unchanged.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Set to 1 if an overflow occurs; otherwise cleared to 0.
- C: Receives the previous value in bit 14.

#### Description

This instruction shifts the bits in a 16-bit register Rd (destination operand) two bits to the left. Bit 14 shifts into the carry flag. Bits 0 and 1 are cleared to 0.



#### **Available Registers**

Rd: R0 to R7, E0 to E7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands	Instruction Format						No. of	
Mode			1st byte		2nd byte		3rd byte 4th byte		States	
Register direct	SHAL.W	#2, Rd	1	0	D	rd			1	

#### Notes

The SHAL instruction differs from the SHLL instruction in its effect on the overflow flag.

#### RENESAS

#### 2.2.57 (5) SHAL(L)

#### SHAL (SHift Arithmetic Left)

#### Shift Arithmetic

Operation	Condition Code								
ERd (left arithmetic shift) $\rightarrow$ ERd	$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$								
Assembly-Language Format	H: Previous value remains unchanged.								
SHAL.L ERd	N: Set to 1 if the result is negative; otherwise cleared to 0.								
	Z: Set to 1 if the result is zero; otherwise								
Operand Size	cleared to 0.								
•	V: Set to 1 if an overflow occurs; otherwise								
Longword	cleared to 0.								
	C: Receives the previous value in bit 31.								

#### Description

This instruction shifts the bits in a 32-bit register ERd (destination operand) one bit to the left. The most significant bit (bit 31) shifts into the carry flag. The least significant bit (bit 0) is cleared to 0.



#### **Available Registers**

ERd: ER0 to ER7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands	Instruction Format						No. of	
Mode	Millenionic	operando	1st byte		2nd byte		3rd byte 4th byte		States	
Register direct	SHAL.L	ERd	1	0	В	0 erd			1	

#### Notes

The SHAL instruction differs from the SHLL instruction in its effect on the overflow flag.

#### 2.2.57 (6) SHAL (L)

#### SHAL (SHift Arithmetic Left)

#### Operation

ERd (left arithmetic shift)  $\rightarrow$  ERd

#### **Assembly-Language Format**

SHAL.L #2, ERd

#### **Operand Size**

Longword

# I UI H U N Z V C I I I I I I I I

- H: Previous value remains unchanged.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Set to 1 if an overflow occurs; otherwise cleared to 0.
- C: Receives the previous value in bit 30.

#### Description

This instruction shifts the bits in a 32-bit register ERd (destination operand) two bits to the left. Bit 30 shifts into the carry flag. Bits 0 and 1 are cleared to 0.



#### **Available Registers**

ERd: ER0 to ER7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands	Instruction Format						No. of
Mode	Witterfiorfic			byte	2no	d byte	3rd byte	4th byte	States
Register direct	SHAL.L	#2, ERd	1	0	F	0 erd			1

#### Notes

The SHAL instruction differs from the SHLL instruction in its effect on the overflow flag.

## RENESAS

#### 2.2.58 (1) SHAR (B)

#### SHAR (SHift Arithmetic Right)

#### Operation

Rd (right arithmetic shift)  $\rightarrow$  Rd

#### Assembly-Language Format

SHAR.B Rd

#### **Operand Size**

Byte

#### **Condition Code**

Ι	UI	Η	U	Ν	Ζ	V	С
				$\uparrow$	$\updownarrow$	0	$\leftrightarrow$

- H: Previous value remains unchanged.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Receives the previous value in bit 0.

#### Description

This instruction shifts the bits in an 8-bit register Rd (destination operand) one bit to the right. Bit 0 shifts into the carry flag. Bit 7 shifts into itself. Since bit 7 remains unaltered, the sign does not change.



#### **Available Registers**

Rd: R0L to R7L, R0H to R7H

#### **Operand Format and Number of States Required for Execution**

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of	
	winemonic	Operands	1st	byte	2nd	byte	3rd byte	4th byte	States	
Register direct	SHAR.B	Rd	1	1	8	rd			1	

#### Notes

## RENESAS

#### 2.2.58 (2) SHAR (B)

#### SHAR (SHift Arithmetic Right)

#### Operation

Rd (right arithmetic shift)  $\rightarrow$  Rd

#### Assembly-Language Format

SHAR.B #2, Rd

#### **Operand Size**

Byte

# Condition Code



- H: Previous value remains unchanged.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Receives the previous value in bit 1.

#### Description

This instruction shifts the bits in an 8-bit register Rd (destination operand) two bits to the right. Bit 1 shifts into the carry flag. Bits 7 and 6 receive the previous value of bit 7. Since bit 7 remains unaltered, the sign does not change.



#### **Available Registers**

Rd: R0L to R7L, R0H to R7H

#### **Operand Format and Number of States Required for Execution**

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of	
	WITEITIOTTIC	Operanus	1st I	byte	2nd	byte	3rd byte	4th byte	States	
Register direct	SHAR.B	#2, Rd	1	1	С	rd			1	

#### Notes

## RENESAS

#### 2.2.58 (3) SHAR (W)

#### SHAR (SHift Arithmetic Right)

#### Operation

Rd (right arithmetic shift)  $\rightarrow$  Rd

#### Assembly-Language Format

SHAR.W Rd

#### **Operand Size**

Word

#### **Condition Code**

Ι	UI	Η	U	Ν	Ζ	V	С
—				$\uparrow$	$\updownarrow$	0	$\leftrightarrow$

- H: Previous value remains unchanged.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Receives the previous value in bit 0.

#### Description

This instruction shifts the bits in a 16-bit register Rd (destination operand) one bit to the right. Bit 0 shifts into the carry flag. Bit 15 shifts into itself. Since bit 15 remains unaltered, the sign does not change.



#### **Available Registers**

Rd: R0 to R7, E0 to E7

#### **Operand Format and Number of States Required for Execution**

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of	
	Witemonic	Operatios	1st I	byte	2nd	byte	3rd byte	4th byte	States	
Register direct	SHAR.W	Rd	1	1	9	rd			1	

#### Notes

## RENESAS

#### 2.2.58 (4) SHAR (W)

#### SHAR (SHift Arithmetic Right)

#### Operation

Rd (right arithmetic shift)  $\rightarrow$  Rd

#### Assembly-Language Format

SHAR.W #2, Rd

#### **Operand Size**

Word

## Condition Code



- H: Previous value remains unchanged.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Receives the previous value in bit 1.

#### Description

This instruction shifts the bits in a 16-bit register Rd (destination operand) two bits to the right. Bit 1 shifts into the carry flag. Bits 15 and 14 receive the previous value of bit 15. Since bit 15 remains unaltered, the sign does not change.



#### **Available Registers**

Rd: R0 to R7, E0 to E7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands	Instruction Format						No. of
Mode	Mode Minemonic Ope		1st byte		2nd byte		3rd byte	4th byte	States
Register direct	SHAR.W	#2, Rd	1	1	D	rd			1

#### Notes

#### RENESAS

#### 2.2.58 (5) SHAR (L)

#### SHAR (SHift Arithmetic Right)

#### Operation

ERd (right arithmetic shift)  $\rightarrow$  ERd

#### Assembly-Language Format

SHAR.L ERd

#### **Operand Size**

Longword

#### **Condition Code**

Ι	UI	Η	U	Ν	Ζ	V	С
—				$\uparrow$	$\updownarrow$	0	$\leftrightarrow$

- H: Previous value remains unchanged.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Receives the previous value in bit 0.

#### Description

This instruction shifts the bits in a 32-bit register ERd (destination operand) one bit to the right. Bit 0 shifts into the carry flag. Bit 31 shifts into itself. Since bit 31 remains unaltered, the sign does not change.



#### **Available Registers**

ERd: ER0 to ER7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands	Instruction Format						No. of	
Mode	Millenionic			byte	2nd byte		yte	3rd byte	4th byte	States
Register direct	SHAR.L	ERd	1	1	В	0	erd			1

#### Notes

## RENESAS

#### 2.2.58 (6) SHAR (L)

#### SHAR (SHift Arithmetic Right)

#### Operation

ERd (right arithmetic shift)  $\rightarrow$  ERd

#### **Assembly-Language Format**

SHAR.L #2, ERd

#### **Operand Size**

Longword

#### Condition Code



- H: Previous value remains unchanged.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Receives the previous value in bit 1.

#### Description

This instruction shifts the bits in a 32-bit register ERd (destination operand) two bits to the right. Bit 1 shifts into the carry flag. Bits 31 and 30 receive the previous value of bit 31. Since bit 31 remains unaltered, the sign does not change.



#### **Available Registers**

ERd: ER0 to ER7

#### **Operand Format and Number of States Required for Execution**

Addressing Mode Mnem	Mnemonic	Operands		No. of						
	Witterfiorfic		1st I	byte	2no	l byte	3rd byte	4th byte	States	
Register direct	SHAR.L	#2, ERd	1	1	F	0 erd			1	

#### Notes

## RENESAS

#### 2.2.59 (1) SHLL (B)

#### SHLL (SHift Logical Left)

#### Shift Logical

Operation	Condition Code
Rd (left logical shift) $\rightarrow$ Rd	$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$
Assembly-Language Format	H: Previous value remains unchanged.
SHLL.B Rd	N: Set to 1 if the result is negative; otherwise cleared to 0.
	Z: Set to 1 if the result is zero; otherwise
Operand Size	cleared to 0.
Derte	V: Always cleared to 0.
Byte	C: Receives the previous value in bit 7.

#### Description

This instruction shifts the bits in an 8-bit register Rd (destination operand) one bit to the left. The most significant bit (bit 7) shifts into the carry flag. The least significant bit (bit 0) is cleared to 0.



#### Available Registers

Rd: R0L to R7L, R0H to R7H

#### **Operand Format and Number of States Required for Execution**

Addressing Mode	Mnemonic	Operands		No. of					
	Willemonic	Operanus	1st	byte	2nd	byte	3rd byte	4th byte	States
Register direct	SHLL.B	Rd	1	0	0	rd			1

#### Notes

The SHLL instruction differs from the SHAL instruction in its effect on the overflow flag.

#### 2.2.59 (2) SHLL (B)

#### SHLL (SHift Logical Left)

#### Operation

Rd (left logical shift)  $\rightarrow$  Rd

#### **Assembly-Language Format**

SHLL.B #2.Rd

#### **Operand Size**

Byte

#### **Condition** Code UI U С Η Ν Ζ 1 1 1 0

- H: Previous value remains unchanged.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Always cleared to 0.

I

C: Receives the previous value in bit 6.

#### Description

This instruction shifts the bits in an 8-bit register Rd (destination operand) two bits to the left. Bit 6 shifts into the carry flag. Bits 0 and 1 are cleared to 0.



#### **Available Registers**

Rd: R0L to R7L, R0H to R7H

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands		No. of					
Mode	Milenonic	operando	1st byte	2nd	byte	3rd byte	4th byte	States	
Register direct	SHLL.B	#2, Rd	1	0	4	rd			1

#### Notes

The SHLL instruction differs from the SHAL instruction in its effect on the overflow flag.

#### RENESAS

#### Shift Logical

#### 2.2.59 (3) SHLL (W)

#### SHLL (SHift Logical Left)

#### Shift Logical

Operation	Condition Code									
Rd (left logical shift) $\rightarrow$ Rd	I UI H U N Z V C									
	$\boxed{-  -  -   \uparrow   \uparrow   0   \uparrow}$									
Assembly-Language Format	H: Previous value remains unchanged.									
SHLL.W Rd	N: Set to 1 if the result is negative; otherwise cleared to 0.									
	Z: Set to 1 if the result is zero; otherwise									
Operand Size	cleared to 0.									
-	V: Always cleared to 0.									
Word	C: Receives the previous value in bit 15.									

#### Description

This instruction shifts the bits in a 16-bit register Rd (destination operand) one bit to the left. The most significant bit (bit 15) shifts into the carry flag. The least significant bit (bit 0) is cleared to 0.



#### **Available Registers**

Rd: R0 to R7, E0 to E7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands		No. of					
Mode	Millenionic	operando	1st	byte	2nd	byte	3rd byte	4th byte	States
Register direct	SHLL.W	Rd	1	0	1	rd			1

#### Notes

The SHLL instruction differs from the SHAL instruction in its effect on the overflow flag.

#### 2.2.59 (4) SHLL (W)

#### SHLL (SHift Logical Left)

#### Operation

Rd (left logical shift)  $\rightarrow$  Rd

#### Assembly-Language Format

SHLL.W #2, Rd

#### **Operand Size**

Word

# Condition CodeIUIHUNZVC--- $\uparrow$ $\uparrow$ 0 $\uparrow$

- H: Previous value remains unchanged.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Receives the previous value in bit 14.

#### Description

This instruction shifts the bits in a 16-bit register Rd (destination operand) two bits to the left. Bit 14 shifts into the carry flag. Bits 0 and 1 are cleared to 0.



#### **Available Registers**

Rd: R0 to R7, E0 to E7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands		No. of					
Mode	Witterfiorfic	Operanus	1st byte		2nd byte		3rd byte	4th byte	States
Register direct	SHLL.W	#2, Rd	1	0	5	rd			1

#### Notes

The SHLL instruction differs from the SHAL instruction in its effect on the overflow flag.

#### RENESAS

#### Shift Logical

#### 2.2.59 (5) SHLL (L)

#### SHLL (SHift Logical Left)

#### Shift Logical

Operation	Co	nditio	on Co	ode						
ERd (left logical shift) $\rightarrow$ ERd		Ι	UI	Н	U	Ν	Ζ	v	С	
		_	—			$\updownarrow$	$\uparrow$	0	\$	
Assembly-Language Format	H: Previous value remains unchanged.									
SHLL.L ERd	N:	N: Set to 1 if the result is negative; otherwise cleared to 0.							vise	
	Z:	Set	to 1 i	f the	resu	lt is z	zero;	othe	rwise	
Operand Size	cleared to 0.									
•	V: Always cleared to 0.									
Longword	C:	Rec	eives	the j	previ	ous v	alue	in bi	it 31.	

#### Description

This instruction shifts the bits in a 32-bit register ERd (destination operand) one bit to the left. The most significant bit (bit 31) shifts into the carry flag. The least significant bit (bit 0) is cleared to 0.



#### **Available Registers**

ERd: ER0 to ER7

#### **Operand Format and Number of States Required for Execution**

Addressing Mode	Mnemonic	Operands		No. of						
	Willemonic	Operands	1st	byte	2no	d byte	3rd byte	4th byte	States	
Register direct	SHLL.L	ERd	1	0	3	0 erd			1	

#### Notes

The SHLL instruction differs from the SHAL instruction in its effect on the overflow flag.

#### 2.2.59 (6) SHLL (L)

#### SHLL (SHift Logical Left)

#### Operation

ERd (left logical shift)  $\rightarrow$  ERd

#### Assembly-Language Format

SHLL.L #2, ERd

#### **Operand Size**

Longword

# I UI H U N Z V C — — — — ↓ ↓ 0 ↓

- H: Previous value remains unchanged.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Receives the previous value in bit 30.

#### Description

This instruction shifts the bits in a 32-bit register ERd (destination operand) two bits to the left. Bit 30 shifts into the carry flag. Bits 0 and 1 are cleared to 0.



#### **Available Registers**

ERd: ER0 to ER7

#### **Operand Format and Number of States Required for Execution**

Addressing Mode	Mnemonic	Operands		No. of						
	WITEHTOTTIC	Operanus	1st	byte	2n	d byte	3rd byte	4th byte	States	
Register direct	SHLL.L	#2, ERd	1	0	7	0 erd			1	

#### Notes

The SHLL instruction differs from the SHAL instruction in its effect on the overflow flag.

#### RENESAS

#### Shift Logical

#### 2.2.60 (1) SHLR (B)

#### SHLR (SHift Logical Right)

#### Operation

Rd (right logical shift)  $\rightarrow$  Rd

#### Assembly-Language Format

SHLR.B Rd

#### **Operand Size**

Byte

#### Description

This instruction shifts the bits in an 8-bit register Rd (destination operand) one bit to the right. The least significant bit (bit 0) shifts into the carry flag. The most significant bit (bit 7) is cleared to 0.



#### Available Registers

Rd: R0L to R7L, R0H to R7H

#### **Operand Format and Number of States Required for Execution**

Addressing Mode	Mnemonic	Operands		No. of					
	WITEINOINC	Operando	1st	byte	2nd	byte	3rd byte	4th byte	States
Register direct	SHLR.B	Rd	1	1	0	rd			1

#### Notes

#### RENESAS



- H: Previous value remains unchanged.
- N: Always cleared to 0.

**Condition Code** 

- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Receives the previous value in bit 0.

♪

0

#### 2.2.60 (2) SHLR (B)

#### SHLR (SHift Logical Right)

#### Operation

Rd (right logical shift)  $\rightarrow$  Rd

#### **Assembly-Language Format**

SHLR.B #2, Rd

#### **Operand Size**

**Condition Code** UI Η U Ν С Ζ

0

- H: Previous value remains unchanged.
- N: Always cleared to 0.

I

- Z: Set to 1 if the result is zero: otherwise cleared to 0
- V: Always cleared to 0.
- C: Receives the previous value in bit 1.

Byte

#### Description

This instruction shifts the bits in an 8-bit register Rd (destination operand) two bits to the right. Bit 1 shifts into the carry flag. Bits 7 and 6 are cleared to 0.



#### **Available Registers**

Rd: R0L to R7L, R0H to R7H

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands	Instruction Format								No. of
Mode	Witemonic	Operanus	1st byte		2nd byte		3rd byte	4th byte	States		
Register direct	SHLR.B	#2, Rd	1	1	4	rd			1		

#### Notes

#### RENESAS

#### Shift Logical

↕

#### 2.2.60 (3) SHLR (W)

#### SHLR (SHift Logical Right)

#### Operation

Rd (right logical shift)  $\rightarrow$  Rd

#### Assembly-Language Format

SHLR.W Rd

#### **Operand Size**

Word

# Condition Code

Ι	UI	Η	U	Ν	Ζ	V	С
				0	$\updownarrow$	0	$\updownarrow$

- H: Previous value remains unchanged.
- N: Always cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Receives the previous value in bit 0.

#### Description

This instruction shifts the bits in a 16-bit register Rd (destination operand) one bit to the right. The least significant bit (bit 0) shifts into the carry flag. The most significant bit (bit 15) is cleared to 0.



#### **Available Registers**

Rd: R0 to R7, E0 to E7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic Operands			No. of						
Mode	winemonic	Operands	1st	byte	2nd	byte	3rd byte	4th byte	States	
Register direct	SHLR.W	Rd	1	1	1	rd			1	

#### Notes

Z ↑

0

H: Previous value remains unchanged.

Z: Set to 1 if the result is zero: otherwise

C: Receives the previous value in bit 1.

#### 2.2.60 (4) SHLR (W)

#### SHLR (SHift Logical Right)

#### Operation

Rd (right logical shift)  $\rightarrow$  Rd

#### **Assembly-Language Format**

SHLR.W #2, Rd

#### **Operand Size**

Word

#### Description

This instruction shifts the bits in a 16-bit register Rd (destination operand) two bits to the right. Bit 1 shifts into the carry flag. Bits 15 and 14 are cleared to 0.

**Condition Code** 

N: Always cleared to 0.

cleared to 0. V: Always cleared to 0.

I UI H U N



#### **Available Registers**

Rd: R0 to R7, E0 to E7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic Operands				No. of					
Mode	WITEHTOTTC	Operatius	1st I	byte	2nd	byte	3rd byte	4th byte	States	
Register direct	SHLR.W	#2, Rd	1	1	5	rd			1	

#### Notes

#### RENESAS

#### Shift Logical

С

↕

0

#### 2.2.60 (5) SHLR (L)

#### SHLR (SHift Logical Right)

#### Operation

ERd (right logical shift)  $\rightarrow$  ERd

#### Assembly-Language Format

SHLR.L ERd

#### **Operand Size**

## Condition Code

Ι	UI	Η	U	Ν	Ζ	V	С
		_		0	$\uparrow$	0	\$

- H: Previous value remains unchanged.
- N: Always cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Receives the previous value in bit 0.

Longword

#### Description

This instruction shifts the bits in a 32-bit register ERd (destination operand) one bit to the right. The least significant bit (bit 0) shifts into the carry flag. The most significant bit (bit 31) is cleared to 0.



#### **Available Registers**

ERd: ER0 to ER7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands		Instruction Format							
Mode	Witemonic	Operanus	1st	byte	2no	d byt	e	3rd byte	4th byte	States	
Register direct	SHLR.L	ERd	1	1	3	0 6	erd			1	

#### Notes

#### 2.2.60 (6) SHLR (L)

#### SHLR (SHift Logical Right)

#### Operation

ERd (right logical shift)  $\rightarrow$  ERd

#### Assembly-Language Format

SHLR.L #2, ERd

#### **Operand Size**

Longword

# Condition Code



- H: Previous value remains unchanged.
- N: Always cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Receives the previous value in bit 1.

#### Description

This instruction shifts the bits in a 32-bit register ERd (destination operand) two bits to the right. Bit 1 shifts into the carry flag. Bits 31 and 30 are cleared to 0.



#### **Available Registers**

ERd: ER0 to ER7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands			No. of				
Mode	WITEHIOTIC	Operanus	1st	byte	2n	d byte	3rd byte	4th byte	States
Register direct	SHLR.L	#2, ERd	1	1	7	0 erd			1

#### Notes

## RENESAS

#### Shift Logical

#### 2.2.61 SLEEP

#### **SLEEP (SLEEP)**

#### **Power-Down Mode**

#### Operation

Program execution state  $\rightarrow$  power-down mode

#### Assembly-Language Format

SLEEP

#### **Operand Size**

#### **Condition Code**



- H: Previous value remains unchanged.
- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Previous value remains unchanged.

#### Description

When the SLEEP instruction is executed, the CPU enters a power-down mode. Its internal state remains unchanged, but the CPU stops executing instructions and waits for an exception-handling request. When it receives an exception-handling request, the CPU exits the power-down mode and begins the exception-handling sequence. Interrupt requests other than NMI cannot end the power-down mode if they are masked in the CPU.

#### **Available Registers**

#### **Operand Format and Number of States Required for Execution**

ſ	Addressing Mode	Mnemonic	Operands			No. of		
				1st byte	2nd byte	3rd byte	4th byte	States
		SLEEP		0 1	8 0			2

#### Notes

For information about power-down modes, see the relevant microcontroller hardware manual.

Store CCR

#### **2.2.62** (1) **STC** (**B**)

#### STC (STore from Control register)

#### Operation

 $\mathrm{CCR} \rightarrow \mathrm{Rd}$ 

#### Assembly-Language Format

STC.B CCR, Rd

#### **Operand Size**

Byte

#### Description

This instruction copies the CCR contents to an 8-bit register Rd.

#### **Available Registers**

Rd: R0L to R7L, R0H to R7H

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	emonic Operands			Instruction Format							
Mode	winemonic	operanas	1st I	byte	2nd	byte	3rd byte	4th byte	States			
Register direct	STC.B	CCR, Rd	0	2	0	rd			1			

Notes

#### RENESAS

Ι	UI	Н	U	N	Z	V	С
	—						

H: Previous value remains unchanged.

**Condition Code** 

- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Previous value remains unchanged.

#### 2.2.62 (2) STC (B)

#### STC (STore from Control register)

#### Operation

 $\text{EXR} \rightarrow \text{Rd}$ 

#### Assembly-Language Format

STC.B EXR, Rd

#### **Operand Size**

Byte

#### Description

This instruction copies the EXR contents to an 8-bit register Rd.

#### **Available Registers**

Rd: R0L to R7L, R0H to R7H

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands		Instruction Format							
Mode	e Minemonic Operand		1st byte		2nd byte		3rd byte 4th byte		States		
Register direct	STC.B	EXR, Rd	0	2	1	rd			1		

#### Notes

#### **Condition Code**

Ι	UI	Η	U	Ν	Ζ	V	С
		_		—	_		

- H: Previous value remains unchanged.
- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Previous value remains unchanged.





Store CCR

#### 2.2.62 (3) STC (W)

#### STC (STore from Control register)

#### Operation

 $CCR \rightarrow (EAd)$ 

#### **Assembly-Language Format**

STC.W CCR, <EAd>

#### **Operand Size**

Word

#### Description

This instruction copies the CCR contents to a destination location. Although CCR is a byte register, the destination operand is a word operand. The CCR contents are stored at the even address. Undetermined data is stored at the odd address.

**Available Registers** ERd: ER0 to ER7

#### RENESAS



# H: Previous value remains unchanged.

N: Previous value remains unchanged.

Z: Previous value remains unchanged.

V: Previous value remains unchanged.

C: Previous value remains unchanged.



**Operand Format and Number of States Required for Execution** 

#### Rev. 4.00 Feb 24, 2006 page 226 of 322 REJ09B0139-0400

#### STC (STore from Control register)

Section 2 Instruction Descriptions

RENESAS

Notes

Store EXR

#### 2.2.62 (4) STC (W)

#### STC (STore from Control register)

#### Operation

 $EXR \rightarrow (EAd)$ 

#### **Assembly-Language Format**

STC.W EXR, <EAd>

#### **Operand Size**

Word

#### Description

This instruction copies the EXR contents to a destination location. Although EXR is a byte register, the destination operand is a word operand. The EXR contents are stored at the even address. Undetermined data is stored at the odd address.

**Available Registers** ERd: ER0 to ER7

#### RENESAS



#### H: Previous value remains unchanged.

N: Previous value remains unchanged.

Z: Previous value remains unchanged.

V: Previous value remains unchanged.

C: Previous value remains unchanged.



Notes

RENESAS

## STC (STore from Control register)

Store EXR

Store Data on Stack

#### 2.2.63 STM

#### STM (STore from Multiple registers)

#### Operation

ERn (register list)  $\rightarrow$  @-SP

#### **Assembly-Language Format**

STM.L <register list>, @-SP

#### **Operand Size**

Longword

#### Description

This instruction saves a group of registers specified by a register list onto the stack. The registers are saved in ascending order of register number.

Two, three, or four registers can be saved by one STM instruction. The following ranges can be specified in the register list.

Two registers:ER0–ER1, ER2–ER3, ER4–ER5, or ER6–ER7Three registers:ER0–ER2 or ER4–ER6Four registers:ER0–ER3 or ER4–ER7

#### **Available Registers**

ERn: ER0 to ER7

#### Renesas

#### 

H: Previous value remains unchanged.

**Condition Code** 

- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Previous value remains unchanged.

#### **STM (STore from Multiple registers)**

#### Store Data on Stack

Addressing	Mnemonic	Operands			I	nstructio	on Forma	t			No. of
Mode	whemonic	Operatios	1st	byte	2nd	byte	3rd	byte	4th	byte	States
_	STM.L	(ERn–ERn+1), @–SP	0	1	1	0	6	D	F	0 ern	7
_	STM.L	(ERn–ERn+2), @–SP	0	1	2	0	6	D	F	0 ern	9
_	STM.L	(ERn–ERn+3), @–SP	0	1	3	0	6	D	F	0 ern	11

#### **Operand Format and Number of States Required for Execution**

#### Notes

When ER7 is saved, the value after effective address calculation (after ER7 is decremented by 4) is saved on the stack.



#### 2.2.64 STMAC

STMAC	(STore	from	MAC	register)
-------	--------	------	-----	-----------

#### Operation

 $\begin{array}{l} MACH \rightarrow ERd \\ or \\ MACL \rightarrow ERd \end{array}$ 

#### **Assembly-Language Format**

STMAC MAC register, ERd

#### **Operand Size**

Longword

#### Store Data from MAC Register

#### **Condition Code**



- H: Previous value remains unchanged.
- N: Set to 1 if a MAC instruction resulted in a negative MAC register value; otherwise cleared to 0.
- Z: Set to 1 if a MAC instruction resulted in a zero MAC register value; otherwise cleared to 0.
- V: Set to 1 if a MAC instruction resulted in an overflow; otherwise cleared to 0.
- C: Previous value remains unchanged.
- Note: \* Execution of this instruction copies the N, Z, and V flag values from the multiplier to the condition-code register (CCR). If the STMAC instruction is executed after a CLRMAC or LDMAC instruction with no intervening MAC instruction, the V flag will be 0 and the N and Z flags will have undetermined values.

#### Description

This instruction moves the contents of a multiply-accumulate register (MACH or MACL) to a general register. If the transfer is from MACH, the upper 22 bits transferred to the general register are a sign extension.

This instruction is supported by the H8S/2600 CPU only.

#### **Available Registers**

ERd: ER0 to ER7

#### Renesas

#### STMAC (STore from MAC register)

#### Store Data from MAC Register

#### **Operand Format and Number of States Required for Execution**

Addressing Mode	Mnemonic	Operands		No. of						
	winemonic	Operations	1st byte		2nd byte			3rd byte	4th byte	States
Register direct	STMAC	MACH, ERd	0	2	2	0 er	ď			1*
Register direct	STMAC	MACL, ERd	0	2	3	0 er	ď			1*

Note: \* A maximum of three additional states are required for execution of this instruction within three states after execution of a MAC instruction. For example, if there is a one-state instruction (such as NOP) between the MAC instruction and this instruction, this instruction will be two states longer. The number of states may differ depending on the product. For details, refer to the relevant microcontroller hardware manual of the product in question.

Notes



#### 2.2.65 (1) SUB (B)

#### SUB (SUBtract binary)

#### **Subtract Binary**

Operation	Condition Code
$Rd - Rs \rightarrow Rd$	I UI H U N Z V C
	$ - -  \updownarrow  -  \updownarrow   \updownarrow   \updownarrow   \updownarrow  $
Assembly-Language Format SUB.B Rs, Rd	<ul> <li>H: Set to 1 if there is a borrow at bit 3; otherwise cleared to 0.</li> <li>N: Set to 1 if the result is negative; otherwise cleared to 0.</li> <li>Z: Set to 1 if the result is zero; otherwise</li> </ul>
<b>Operand Size</b> Byte	<ul><li>cleared to 0.</li><li>V: Set to 1 if an overflow occurs; otherwise cleared to 0.</li></ul>
	C: Set to 1 if there is a borrow at bit 7; otherwise cleared to 0.

#### Description

This instruction subtracts the contents of an 8-bit register Rs (source operand) from the contents of an 8-bit register Rd (destination operand) and stores the result in the 8-bit register Rd.

#### **Available Registers**

Rd: R0L to R7L, R0H to R7H Rs: R0L to R7L, R0H to R7H

#### **Operand Format and Number of States Required for Execution**

Addressing Mode	Mnemonic	Operands		No. of						
	Milenonic	Operando	1st I	byte	2nd	byte	3rd byte	4th byte	States	
Register direct	SUB.B	Rs, Rd	1	8	rs	rd			1	

#### SUB (SUBtract binary)

#### Notes

The SUB.B instruction can operate only on general registers. Immediate data can be subtracted from general register contents by using the SUBX instruction. Before executing SUBX #xx:8, Rd, first set the Z flag to 1 and clear the C flag to 0. The following coding examples can also be used to subtract nonzero immediate data #IMM.

(1)	ORC	#H'05,CCR
	SUBX	#(IMM-1),Rd
(2)	ADD	#(0-IMM),Rd
	XORC	#H'01,CCR



#### 2.2.65 (2) SUB (W)

#### SUB (SUBtract binary)

#### **Subtract Binary**

Operation	Condition Code									
$Rd - (EAs) \rightarrow Rd$	$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$									
Assembly-Language Format         SUB.W <eas>, Rd         Operand Size         Word</eas>	<ul> <li>H: Set to 1 if there is a borrow at bit 11; otherwise cleared to 0.</li> <li>N: Set to 1 if the result is negative; otherwise cleared to 0.</li> <li>Z: Set to 1 if the result is zero; otherwise cleared to 0.</li> <li>V: Set to 1 if an overflow occurs; otherwise cleared to 0.</li> <li>C: Set to 1 if there is a borrow at bit 15; otherwise cleared to 0.</li> </ul>									

#### Description

This instruction subtracts a source operand from the contents of a 16-bit register Rd (destination operand) and stores the result in the 16-bit register Rd.

#### **Available Registers**

Rd: R0 to R7, E0 to E7 Rs: R0 to R7, E0 to E7

#### **Operand Format and Number of States Required for Execution**

Addressing Mode	Mnemonic	Operands		No. of					
	Witterfiorfic	Operanus	1st I	byte	2nd	byte	3rd byte	4th byte	States
Immediate	SUB.W	#xx:16, Rd	7	9	3	rd	IMM		2
Register direct	SUB.W	Rs, Rd	1	9	rs	rd			1

Notes

#### 2.2.65 (3) SUB (L)

#### SUB (SUBtract binary)

#### **Subtract Binary**

Operation	Condition Code									
$ERd - (EAs) \rightarrow ERd$	I UI H U N Z V C									
	$ - -  \updownarrow  -  \updownarrow   \updownarrow   \updownarrow   \updownarrow  $									
Assembly-Language Format	H: Set to 1 if there is a borrow at bit 27;									
SUB.L <eas>, ERd</eas>	otherwise cleared to 0. N: Set to 1 if the result is negative; otherwise cleared to 0.									
	Z: Set to 1 if the result is zero; otherwise									
Operand Size	cleared to 0.									
Longword	V: Set to 1 if an overflow occurs; otherwise cleared to 0.									
	C: Set to 1 if there is a borrow at bit 31; otherwise cleared to 0.									

#### Description

This instruction subtracts a source operand from the contents of a 32-bit register ERd (destination operand) and stores the result in the 32-bit register ERd.

#### **Available Registers**

ERd: ER0 to ER7 ERs: ER0 to ER7

#### **Operand Format and Number of States Required for Execution**

Addressing Mode Mnemonic	Mnomonio	Operands	Instruction Format									
	Operatios	1st	byte	2nd	byte	3rd byte	4th byte	5th byte	6th byte	States		
Immediate	SUB.L	#xx:32, ERd	7	А	3	0 erd		IN	IM		3	
Register direct	SUB.L	ERs, ERd	1	А	1 ers	0 erd					1	

#### Notes


#### 2.2.66 SUBS

#### SUBS (SUBtract with Sign extension)

#### Operation

 $Rd - 1 \rightarrow ERd$  $Rd - 2 \rightarrow ERd$  $Rd - 4 \rightarrow ERd$ 

#### **Assembly-Language Format**

SUBS #1, ERd SUBS #2, ERd SUBS #4, ERd

#### **Operand Size**

Longword

#### Subtract Binary Address Data

#### **Condition Code**



- H: Previous value remains unchanged.
- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Previous value remains unchanged.

#### Description

This instruction subtracts the immediate value 1, 2, or 4 from the contents of a 32-bit register ERd (destination operand). Unlike the SUB instruction, it does not affect the condition-code flags.

#### **Available Registers**

ERd: ER0 to ER7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands			In	str	uctio	n Format	No. of	
Mode	winemonic	Operatius	1st byte		2nd byte		/te	3rd byte	4th byte	States
Register direct	SUBS	#1, ERd	1	В	0	0	erd			1
Register direct	SUBS	#2, ERd	1	В	8	0	erd			1
Register direct	SUBS	#4, ERd	1	В	9	0	erd			1

#### Notes

#### 2.2.67 SUBX

#### Operation

 $Rd - (EAs) - C \rightarrow Rd$ 

#### **Assembly-Language Format**

SUBX <EAs>, Rd

#### **Operand Size**

Byte

# **Condition Code**

Ι	UI	Η	U	Ν	Ζ	V	С
—		$\updownarrow$		$\uparrow$	$\updownarrow$	$\updownarrow$	$\uparrow$

- H: Set to 1 if there is a borrow at bit 3; otherwise cleared to 0.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Previous value remains unchanged when the result is zero; otherwise cleared to 0.
- V: Set to 1 if an overflow occurs; otherwise cleared to 0.
- C: Set to 1 if there is a borrow at bit 7; otherwise cleared to 0.

#### Description

This instruction subtracts the source operand and carry flag from the contents of an 8-bit register Rd (destination operand) and stores the result in the 8-bit register Rd.

#### **Available Registers**

Rd: R0L to R7L, R0H to R7H

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands		Instruction Format							
Mode	Witemonic	Operatios	1st byte		2nd byte		3rd byte	4th byte	States		
Immediate	SUBX	#xx:8, Rd	В	rd	IMM				1		
Register direct	SUBX	Rs, Rd	1	E	rs	rd			1		

#### Notes

# RENESAS

#### Subtract with Borrow

#### 2.2.68 TAS

#### TAS (Test And Set)

#### Operation

 $@ERd - 0 \rightarrow set/clear CCR$ 1  $\rightarrow$  (<bit 7> of @ERd)

#### **Assembly-Language Format**

TAS @ERd

#### **Operand Size**

Byte

#### Test and Set



#### Description

This instruction tests a memory operand by comparing it with zero, and sets the condition-code register according to the result. Then it sets the most significant bit (bit 7) of the operand to 1.

#### **Available Registers**

ERd: ER0, ER1, ER4, ER5

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands		Instruction Format							
Mode	Milenionic	operando	1st byte		2nd byte		3rd byte		4th byte		States
Register indirect	TAS	@ERd	0	1	Е	0	7	В	0 erd	С	4

Notes

#### 2.2.69 TRAPA

#### TRAPA (TRAP Always)

#### Operation

- When EXR is invalid  $PC \rightarrow @-SP$   $CCR \rightarrow @-SP$  $<Vector> \rightarrow PC$
- When EXR is valid  $PC \rightarrow @-SP$   $CCR \rightarrow @-SP$   $EXR \rightarrow @-SP$  $<Vector > \rightarrow PC$

#### Assembly-Language Format

TRAPA #x:2

#### **Operand Size**

#### **Condition Code**

Ι	UI	Η	U	Ν	Ζ	V	С
1	*						

I: Always set to 1.

UI: See note.

- H: Previous value remains unchanged.
- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Previous value remains unchanged.
- Note: \* The UI bit is set to 1 when used as an interrupt mask bit, but retains its previous value when used as a user bit. For details, see the relevant microcontroller hardware manual.

#### Description

This instruction pushes the program counter (PC) and condition-code register (CCR) onto the stack, then sets the I bit to 1. If the extended control register (EXR) is valid, EXR is also saved onto the stack, but bits I2 to I0 are not modified. Next execution branches to a new address given by the contents of the vector address corresponding to the specified vector number. The PC value pushed onto the stack is the starting address of the next instruction after the TRAPA instruction.

#x	Vector Address							
#X	Normal Mode	Advanced Mode						
0	H'0010 to H'0011	H'000020 to H'000023						
1	H'0012 to H'0013	H'000024 to H'000027						
2	H'0014 to H'0015	H'000028 to H'00002B						
3	H'0016 to H'0017	H'00002C to H'00002F						

# RENESAS

#### Trap Unconditionally

#### TRAPA (TRAP Always)

#### **Trap Unconditionally**

#### **Operand Format and Number of States Required for Execution**

Addressing Mnemonic	Oporande		No. of						
Mode	WITEITIOTTIC	Operatius	1st	1st byte		yte	3rd byte	4th byte	States
Register direct	TRAPA	#x:2	5	7	00 MM	0			7*

Note: \* Eight states when EXR is valid.

#### Notes

The stack and vector structure differ between normal mode and advanced mode, and depending on whether EXR is valid or invalid.

#### 2.2.70 (1) XOR (B)

#### XOR (eXclusive OR logical)

#### Operation

 $\operatorname{Rd} \oplus (\operatorname{EAs}) \to \operatorname{Rd}$ 

#### **Assembly-Language Format**

XOR.B <EAs>, Rd

#### **Operand Size**

Byte

#### **Condition Code**

Ι	UI	Η	U	Ν	Ζ	V	С
				$\uparrow$	$\updownarrow$	0	

- H: Previous value remains unchanged.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Previous value remains unchanged.

#### Description

This instruction exclusively ORs the source operand with the contents of an 8-bit register Rd (destination operand) and stores the result in the 8-bit register Rd.

#### **Available Registers**

Rd: R0L to R7L, R0H to R7H Rs: R0L to R7L, R0H to R7H

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands		Instruction Format							
Mode	Willemonic	Operatios	1st byte		2nd byte		3rd byte	4th byte	States		
Immediate	XOR.B	#xx:8, Rd	D	rd	IMM				1		
Register direct	XOR.B	Rs, Rd	1	5	rs	rd			1		

#### Notes

# RENESAS

# **Exclusive Logical OR**

#### 2.2.70 (2) XOR (W)

#### XOR (eXclusive OR logical)

#### Operation

 $\operatorname{Rd} \oplus (\operatorname{EAs}) \to \operatorname{Rd}$ 

#### **Assembly-Language Format**

XOR.W <EAs>, Rd

#### **Operand Size**

Word

# Condition Code



- H: Previous value remains unchanged.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Previous value remains unchanged.

#### Description

This instruction exclusively ORs the source operand with the contents of a 16-bit register Rd (destination operand) and stores the result in the 16-bit register Rd.

#### **Available Registers**

Rd: R0 to R7, E0 to E7 Rs: R0 to R7, E0 to E7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic Operan				No. of				
Mode	Witemonic	Operanus	1st byte		2nd byte		3rd byte 4th by		States
Immediate	XOR.W	#xx:16, Rd	7	9	5	rd	IMM		2
Register direct	XOR.W	Rs, Rd	6	5	rs	rd			1

Notes

# RENESAS

#### **Exclusive Logical OR**

#### 2.2.70 (3) XOR (L)

#### XOR (eXclusive OR logical)

#### Operation

 $\operatorname{ERd} \oplus (\operatorname{EAs}) \to \operatorname{ERd}$ 

#### **Assembly-Language Format**

XOR.L <EAs>, ERd

#### **Operand Size**

Longword

#### **Condition Code**

Ι	UI	Η	U	Ν	Ζ	V	С
—				$\updownarrow$	$\updownarrow$	0	

- H: Previous value remains unchanged.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Always cleared to 0.
- C: Previous value remains unchanged.

#### Description

This instruction exclusively ORs the source operand with the contents of a 32-bit register ERd (destination operand) and stores the result in the 32-bit register ERd.

#### **Available Registers**

ERd: ER0 to ER7 ERs: ER0 to ER7

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands					Instruction	on Format			No. of
Mode	winemonic	Operands	1st	byte	2nd	byte	3rd byte	4th byte	5th byte	6th byte	States
Immediate	XOR.L	#xx:32, ERd	7	А	5	0 erd		IN	IM		3
Register direct	XOR.L	ERs, ERd	0	1	F	0	6 5	0 ers 0 erd			2

#### Notes

# RENESAS

#### **Exclusive Logical OR**

#### 2.2.71 (1) XORC

#### Operation

 $\mathrm{CCR} \oplus \#\mathrm{IMM} \to \mathrm{CCR}$ 

#### **Assembly-Language Format**

XORC #xx:8, CCR

#### **Operand Size**

Byte

#### **Exclusive Logical OR with CCR**

#### **Condition Code**

Ι	UI	Н	U	Ν	Ζ	V	С
$\updownarrow$	\$						

I: Stores the corresponding bit of the result.

- UI: Stores the corresponding bit of the result.
- H: Stores the corresponding bit of the result.
- U: Stores the corresponding bit of the result.
- N: Stores the corresponding bit of the result.
- Z: Stores the corresponding bit of the result.
- V: Stores the corresponding bit of the result.
- C: Stores the corresponding bit of the result.

#### Description

This instruction exclusively ORs the contents of the condition-code register (CCR) with immediate data and stores the result in the condition-code register. No interrupt requests, including NMI, are accepted immediately after execution of this instruction.

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands			Instructio	n Format		No. of
Mode	Milenie	operanas	1st I	byte	2nd byte	3rd byte	4th byte	States
Immediate	XORC	#xx:8, CCR	0	5	IMM			1

Notes

#### 2.2.71 (2) XORC

#### Operation

 $\mathrm{EXR} \oplus \mathrm{\#IMM} \to \mathrm{EXR}$ 

#### Assembly-Language Format

XORC #xx:8, EXR

#### **Exclusive Logical OR with EXR**

#### **Condition Code**



- H: Previous value remains unchanged.
- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Previous value remains unchanged.

#### **Operand Size**

Byte

#### Description

This instruction exclusively ORs the contents of the extended control register (EXR) with immediate data and stores the result in the extended control register. No interrupt requests, including NMI, are accepted for three states after execution of this instruction.

#### **Operand Format and Number of States Required for Execution**

Addressing	Mnemonic	Operands			Ins	tructio	n Forr	nat		No. of
Mode	Milenionio	operanas	1st I	byte	2nd	byte	3rd	byte	4th byte	States
Immediate	XORC	#xx:8, EXR	0	1	4	1	0	5	IMM	2

Notes



# (1) Data Transfer Instructions

1			Adc	Iressing	Addressing Mode and Instruction Length (Bytes)	nd Ins	tructic	on Lenç	gth (B)	rtes)								2	7
			<u> </u>			+u83								Condition Code	lition	Cod	Ð	No. or States*1	s*1
		A710			EBn) u			PC)				Operation						al	pəpu
			xx#	uŊ	(d,b) ФЕR		66 Ø	"p)@	?@@	-			-	I	z	> z	ပ	Norm	вурА
	MOV.B #xx:8,Rd	в	5								3:xx#	#xx:8→Rd8	Ι		↔	0 ↔	1	-	
	MOV.B Rs, Rd	æ		2							Rs8-	Rs8→Rd8	Ι	I	$\leftrightarrow$	0 ↔	Ι	-	
	MOV.B @ERs,Rd	æ			5						@ER	@ERs→Rd8	1	1	$\leftrightarrow$	0 ↔	1	2	
	MOV.B @(d:16, ERs), Rd	æ			4						@(d:	@(d:16,ERs)→Rd8	Ι	I	$\leftrightarrow$	0 ↔	Ι	e	
	MOV.B @(d:32,ERs),Rd	в			80						@(d:	@(d:32,ERs)→Rd8	Ι	1	$\leftrightarrow$	0 ↔	Ι	5	
	MOV.B @ERs+,Rd	æ				2					@ER	@ERs→Rd8,ERs32+1 →ERs32	Ι	1	$\leftrightarrow$	0 ↔	1	e	
	MOV.B @aa:8,Rd	m					2				@aa:	@aa:8→Rd8	Ι	1	$\leftrightarrow$	0	1	2	
	MOV.B @aa:16,Rd	в					4				@aa:	@aa:16→Rd8	Ι	1	$\leftrightarrow$	0 4	Ι	Э	
	MOV.B @aa:32,Rd	в					9				@aa:	@aa:32→Rd8	Ι	1	$\leftrightarrow$	0		4	
	MOV.B Rs, @ERd	в			2						Rs8-	Rs8→@ERd	Ι	1	$\leftrightarrow$	0 ↓	Ι	2	
	MOV.B Rs, @(d: 16,ERd)	в			4						Rd8-	Rd8→@(d:16,ERd)	Ι	1	↔	0 ↓		3	
	MOV.B Rs, @(d:32,ERd)	в			8						Rd8-	Rd8→@(d:32,ERd)	Ι	Ī	\` ↓	0	Ι	5	
	MOV.B Rs, @-ERd	в				2					ERd	ERd32-1→ERd32,Rs8→@ERd	Ι	Ī	\` ↓	0 ↓	Ι	3	
	MOV.B Rs, @aa:8	в					2				Rs8-	Rs8→@aa:8	I	1	$\leftrightarrow$	0 ↓	Ι	2	
	MOV.B Rs, @aa:16	В					4				Rs8-	Rs8→@aa:16	Ι	1	\	0	Ι	3	
	MOV.B Rs, @aa:32	В					6				Rs8-	Rs8→@aa:32	Ι	Ī	↔	¢ 0	Ι	4	
	MOV.W #xx:16,Rd	M	4								#xx:1	#xx:16→Rd16	Ι	1	↔	0 \$	Ι	2	
	MOV.W Rs,Rd	Ν		2							Rs16	Rs16→Rd16	Ι	1	$\leftrightarrow$	0 ↓	Ι	-	
	MOV.W @ERs,Rd	≥			2						@ER	@ERs→Rd16	Ι	1	$\leftrightarrow$	0 ()	Ι	2	
	MOV.W @(d:16,ERs),Rd	M		-	4						@(d:	@(d:16,ERs)→Rd16	Ι	Ì	`´ ↔	0	Ι	3	
	MOV.W @(d:32,ERs),Rd	M		-	8						:p)@	@(d:32,ERs)→Rd16	Ι	1	↔	0 ↓	1	5	
	MOV.W @ERs+,Rd	M				2					@ER	@ERs→Rd16,ERs32+2→@ERs32	Ι		$\leftrightarrow$	0 ↓	Ι	3	
	MOV.W @aa:16,Rd	N		-	_	_	4	_			@aa:	@aa:16→Rd16	Ι	1	$\leftrightarrow$	0 ↔	1	3	
	MOV.W @aa:32,Rd	N		-	_	_	9	_			@aa:	@aa:32→Rd16	Ι		$\leftrightarrow$	0 ↔	1	4	
	MOV.W Rs, @ERd	N			2						Rs16	Rs16→@ERd	Ι		$\leftrightarrow$	0 ↓	Ι	2	
	MOV.W Rs,@(d:16,ERd)	3			4						Rs16	Rs16→@(d:16,ERd)	Ι	1	$\leftrightarrow$	0 ↔	1	e	
	MOV.W Rs,@(d:32,ERd)	3			8						Rs16	Rs16→@(d:32,ERd)	Ι	Ī	$\leftrightarrow$	0 ↔	1	5	
	MOV.W Rs, @-ERd	3				7					ERd	ERd32-2→ERd32,Rs16→@ERd	Ι	I	$\leftrightarrow$	0 ↔	1	e	
	MOV.W Rs,@aa:16	3					4				Rs16	Rs16→@aa:16	Ι	Ī	$\leftrightarrow$	0 ↔	1	e	
	MOV.W Rs,@aa:32	N			_		9				Rs16	Rs16→@aa:32	Ι	Ì	$\leftrightarrow$	0 ↔	Ι	4	

# RENESAS

#### 2.3 **Instruction Set**

#### Table 2.1 **Instruction Set**

			Ă	ddressi	ng Mo	and	Instrue	Addressing Mode and Instruction Length (Bytes)	ength (	(Bytes)								2	5
	cin c mo cu						+u83						-	Condition Code	ition	Cod	Ð	No. or States*1	۵. 1.
		ATIC			u	(uya	]@/uと		(Da	ee		Operation						al	рәри
			xx#	uЯ	ØЕК	l'p)@	13-@	66 D	l'p)@	200	_	1	-	2 1	z	> N	ပ	Norm	вурА
MOV	MOV.L #xx:32,ERd		9									#xx:32→ERd32	Ī	↔ 	$\leftrightarrow$	0 ↔	1	e	
	MOV.L ERS, ERd	_		2				-				ERs32→ERd32	Ι	↔	$\leftrightarrow$	0 ↔	Ι	-	
	MOV.L @ERS,ERd	-			4							@ERs→ERd32	1	↔ 	$\leftrightarrow$	0 ↔		4	
	MOV.L @(d:16,ERs),ERd	_				9						@ (d:16,ERs)→ERd32	1	↔		0 ↔	1	5	
	MOV.L @(d:32,ERs),ERd	-				10						@ (d:32,ERs)→ERd32	1	↔	$\leftrightarrow$	0 ↔	1	7	
	MOV.L @ERs+,ERd	_					4					@ERs→ERd32,ERs32+4→@ERs32	1	↔	· ∕ ↔	0 ↔	Ι	5	
	MOV.L @aa:16,ERd	_						9				@aa:16→ERd32	Ι	+	4	0 ↓	I	2	
	MOV.L @aa:32,ERd	L						8				@aa:32→ERd32		÷ 	4	0 ↓	Ι	9	
	MOV.L ERS,@ERd	L			4							ERs32→@ERd		÷ 	↔	0 ↓	Ι	4	
	MOV.L ERs, @ (d: 16, ERd)	L				9						ERs32→@(d:16,ERd)		↔ 	L L	0 ↓		5	
	MOV.L ERs, @ (d: 32, ERd)	L				10						ERs32→@(d:32,ERd)		↔ 	$\leftrightarrow$	0 ↓	1	7	
	MOV.L ERs, @-ERd	L					4					ERd32–4→ERd32,ERs32→@ERd	Ι	↔ 	L'	0 ↔	1	5	
	MOV.L ERs,@aa:16	-						9				ERs32→@aa:16		↔ 	$\leftrightarrow$	0 ≎		5	
	MOV.L ERs,@aa:32	-						8				ERs32→@aa:32			$\leftrightarrow$	0 ↓		9	
РОР	POP.W Rn	Ν				-					2	@SP→Rn16,SP+2→SP		↔ 	$\leftrightarrow$	0 ↓		3	
	POP.L ERn	L									4	@SP→ERn32,SP+4→SP		1	↔	0 \$	Ι	5	
HSU	PUSH.W Rn	W									2	SP–2→SP,Rn16→@SP		1	↔	0 \$		3	
	PUSH.L ERn	L									4	SP-4→SP,ERn32→@SP		÷ 	↔	0 \$	Ι	5	
LDM	LDM.L @SP+,(ERm-ERn)	_									4	(@SP→ERn32,SP+4→SP) Repeated for each register restored	Ι		1		1	7/9/11*3	1*3
STM	STM.L (ERm-ERn),@-SP	L									4	(SP-4→SP,ERn32→@SP) Repeated for each register saved	Ι	 			1	7/9/11*3	1*3
MOVFPE	MOVFPE@aa:16,Rd	В						4				@aa:16→Rd (synchronized with E clock)	Ι		$\leftrightarrow$	0 ↔	Ι	(1)	_
MOVTPE	MOVTPE Rs, @aa:16	в						4				Rs→@aa:16 (synchronized with E clock)	Ι	↔ 	$\leftrightarrow$	0 ↔	Ι	(1)	_
					1	1	1			-			1	1	1				1

# Rev. 4.00 Feb 24, 2006 page 248 of 322 REJ09B0139-0400

	č	ي». *	pəɔu	вvbA																																
	Ň	States*1	lsi	Norm	1	٢	2	-	3	1	٢	٢	-	-	-	1	٢	-	-	-	-	-	2	-	3	1	1	٢	-	-	-	1	-	-	-	-
				с	↔	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	↔	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$										$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$								
		ode		>	↔	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	Ι	Ι	Ι	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	*	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	Ι	1	Ι	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$
		Condition Code		N	↔	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	(4)	(4)	Ι	Ι	Ι	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	(4)	(4)	Ι	Ι	Ι	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$
		ditio		z	↔	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	Ι	Ι	Ι	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	Ι	Ι	Ι	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$
		Cor		т	↔	$\leftrightarrow$	(2)	(2)	(3)	(3)	$\leftrightarrow$	$\leftrightarrow$	Ι	I	Ι	Ι	Ι	Ι	Ι	Ι	*	$\leftrightarrow$	(2)	(3	(3)	(3)	$\leftrightarrow$	$\leftrightarrow$	Ι	Ι	Ι	Ι	I	Ι	Ι	1
				-	Ι	Ι	Ι	I	Ι	Ι		Ι	I	I	I	Ι	Ι	I	I	Ι	I	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	I	I	Ι		I	Ι	I
		Oneration			Rd8+#xx:8→Rd8	Rd8+Rs8→Rd8	Rd16+#xx:16→Rd16	Rd16+Rs16→Rd16	ERd32+#xx:32→ERd32	ERd32+ERs32→ERd32	Rd8+#xx:8+C→Rd8	Rd8+Rs8+C→Rd8	ERd32+1→ERd32	ERd32+2→ERd32	ERd32+4→ERd32	Rd8+1→Rd8	Rd16+1→Rd16	Rd16+2→Rd16	ERd32+1→ERd32	ERd32+2→ERd32	Rd8 decimal adjust → Rd8	Rd8–Rs8→Rd8	Rd16–#xx:16→Rd16	Rd16–Rs16→Rd16	ERd32–#xx:32→ERd32	ERd32–ERs32→ERd32	Rd8–#xx:8–C→Rd8	Rd8–Rs8–C→Rd8	ERd32–1→ERd32	ERd32–2→ERd32	ERd32–4→ERd32	Rd8–1→Rd8	Rd16–1→Rd16	Rd16–2→Rd16	ERd32–1→ERd32	ERd32-2→ERd32
	rtes)			_																																
!	igth (B)			,b)@ ,0@																																
-	ion Ler			66.0a																																
	Addressing Mode and Instruction Length (Bytes)	+u83																						_												
	e and		(uya	ʻp)@																																
:	ooM gr		u;	@EB																																
	dressi			uЯ		2		2		2		2	2	2	2	2	2	2	2	2	2	2		2		2		2	2	2	2	2	2	2	2	2
	8 A			xx#	2		4		6		2												4		6		2									
		Size			в	в	Ν	N	Г	Γ	В	в	_	_	_	в	Ν	N	_	_	ш	в	Ν	N	Γ	Γ	В	в	_	_	_	В	Ν	≥	_	_
		Maemonic			ADD.B #xx:8,Rd	ADD.B Rs,Rd	ADD.W #xx:16,Rd	ADD.W Rs,Rd	ADD.L #xx:32,ERd	ADD.L ERS,ERd	ADDX #xx:8,Rd	ADDX Rs,Rd	ADDS #1, ERd	ADDS #2, ERd	ADDS #4, ERd	INC.B Rd	INC.W #1,Rd	INC.W #2,Rd	INC.L #1,ERd	INC.L #2,ERd	DAA Rd	SUB.B Rs,Rd	SUB.W #xx:16,Rd	SUB.W RS,Rd	SUB.L #xx:32,ERd	SUB.L ERS, ERd	SUBX #xx:8,Rd	SUBX Rs,Rd	SUBS #1,ERd	SUBS #2,ERd	SUBS #4,ERd	DEC.B Rd	DEC.W #1,Rd	DEC.W #2,Rd	DEC.L #1,ERd	DEC.L #2,ERd
					ADD						ADDX		ADDS			INC					DAA	SUB					SUBX		SUBS			DEC				

(2) Arithmetic Operation Instructions

Image: constrained by the section of the sectin of the section of the section o				Ă	Addressing Mode and Instruction Length (Bytes)	g Mod	e and l	nstruct	tion Le	ngth (E	tytes)									
Mutcholic         See (4,6,7c)         Mutcholic         Commander (1,1,1,1,1,1,2,5,1,5,1,5,1,5,1,5,1,5,1,5,			i					+u8	<u> </u>						Cond	lition	Code		No. o State:	م <u>،</u> ج
MUXUN Rs.ERd         B         C <thc< th="">         C         <thc< th="">         C         <thc< th=""> <thc<< th=""><th></th><th>Mnemonic</th><th>SIZE</th><th></th><th></th><th>u</th><th></th><th>3@/u2</th><th>.00</th><th></th><th>pt</th><th></th><th>Operation</th><th></th><th></th><th></th><th></th><th></th><th>al</th><th>рәри</th></thc<<></thc<></thc<></thc<>		Mnemonic	SIZE			u		3@/u2	.00		pt		Operation						al	рәри
DMSRd         B         1         2         1 <th></th> <th></th> <th></th> <th>xx#</th> <th>uЯ</th> <th>83@</th> <th></th> <th>_</th> <th></th> <th></th> <th>: @@</th> <th>_</th> <th></th> <th>-</th> <th></th> <th></th> <th></th> <th>ပ</th> <th>Norm</th> <th>вvbA</th>				xx#	uЯ	83@		_			: @@	_		-				ပ	Norm	вvbA
MUXUB Rs,Rd         B         2         1         1         Robic/seb-Ref         1 <th1< th=""> <th1< th=""></th1<></th1<>	DAS	DAS Rd	в		2							Ř	d8 decimal adjust →Rd8	Ι	*	↔	*	Ι	-	
MUXUNRSERd         W         2         I         I         C         I         C <thc< td=""><td>MULXU</td><td>MULXU.B Rs,Rd</td><td>۵</td><td></td><td>2</td><td></td><td></td><td></td><td></td><td></td><td></td><td>Ϋ́</td><td>d8×Rs8→Rd16 .nsigned multiplication)</td><td>I</td><td> </td><td>1</td><td>1</td><td>Ι</td><td>3 (12<sup>*</sup></td><td>دم) 0</td></thc<>	MULXU	MULXU.B Rs,Rd	۵		2							Ϋ́	d8×Rs8→Rd16 .nsigned multiplication)	I		1	1	Ι	3 (12 <sup>*</sup>	دم) 0
MULXSBRs.Rd         B         4         A <th< td=""><td></td><td>MULXU.W Rs,ERd</td><td>≥</td><td></td><td>7</td><td></td><td></td><td></td><td></td><td></td><td></td><td>ЖЭ</td><td>d16×Rs16→ERd32 nsigned multiplication)</td><td>Ι</td><td>· ·</td><td>· ·</td><td></td><td>Ι</td><td>4 (20<sup>*</sup></td><td>(1)</td></th<>		MULXU.W Rs,ERd	≥		7							ЖЭ	d16×Rs16→ERd32 nsigned multiplication)	Ι	· ·	· ·		Ι	4 (20 <sup>*</sup>	(1)
MULXSW Rs.ERd         W         I         <	MULXS	MULXS.B Rs,Rd	۵		4							ы К К	d8×Rs8→Rd16 igned multiplication)	Ι				Ι	4 (13 <sup>*</sup> *5 *1	(1)
DIVXLB Rs, Rd         B         2         P         Cal: quotient) (unsigned division)         P         C <thc< th=""> <thc< t<="" td=""><td></td><td>MULXS.W Rs, ERd</td><td>≥</td><td></td><td>4</td><td></td><td></td><td></td><td></td><td></td><td></td><td>Si R</td><td>d16×Rs16→ERd32 igned multiplication)</td><td>Ι</td><td></td><td></td><td>· ·</td><td>Ι</td><td>5 (21* *5 *1</td><td>(1)</td></thc<></thc<>		MULXS.W Rs, ERd	≥		4							Si R	d16×Rs16→ERd32 igned multiplication)	Ι			· ·	Ι	5 (21* *5 *1	(1)
DIVX.U.W.R.ERd         W         2         H         L         E Rd32-Fs f6 - E Rd32 (Ed: remainder, icmainder, icmaind	DIVXU	DIVXU.B Rs,Rd	۵		2							<u> </u>	d16÷Rs8→Rd16 (RdH: remainder, dL: quotient) (unsigned division)	Ι		<u> </u>	· ·	Ι	12	
		DIVXU.W RS,ERd	≥		7							шĸ	Rd32÷Rs16→ERd32 (Ed: remainder, d: quotient) (unsigned division)	Ι	-			Ι	20	
DIVXS.W Rs, ERd         W         I         4         I         I         ERd32: FR 16→E Rd32 (Ed: remainder, icon)         I <thi< th="">         I         I</thi<>	DIVXS	DIVXS.B Rs,Rd	۵		4							<u> </u>	d16÷Rs8→Rd16 (RdH: remainder, dL: quotient) (signed division)	Ι				Ι	13	
CMPB #xx:8,rd         B         2         1         <		DIVXS.W Rs,ERd	≥		4							шĸ	Rd32÷Rs16→ERd32 (Ed: remainder, d: quotient) (signed division)	I				I	21	
CMPB Rs, Rd         B         2         0         0         RdB-RsB         CdP-RsB          2 <th2< th=""> <th2< td="" th<=""><td>CMP</td><td>CMP.B #xx:8,Rd</td><td>в</td><td>2</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>R</td><td>d8-#xx:8</td><td>Ι</td><td><math>\leftrightarrow</math></td><td>↔</td><td></td><td><math>\leftrightarrow</math></td><td>1</td><td></td></th2<></th2<>	CMP	CMP.B #xx:8,Rd	в	2								R	d8-#xx:8	Ι	$\leftrightarrow$	↔		$\leftrightarrow$	1	
		CMP:B Rs,Rd	ш		2							ğ	d8–Rs8	I	$\leftrightarrow$	$\leftrightarrow$		$\leftrightarrow$	-	
$ \begin{array}{c c c c c c c c c c c c c c c c c c c $		CMP.W #xx:16,Rd	≥	4								ž	d16-#xx:16	Ι	(2)	$\leftrightarrow$		$\leftrightarrow$	7	
CMPL #Xx.32.ERd         L         6          ERd32-#Xx.32         -         (3)         (2)		CMP.W Rs,Rd	≥		2						-	Ř	d16–Rs16	-	6	$\leftrightarrow$		$\leftrightarrow$	-	
CMPL ERs.Erd         L         2         P         ERd32-ERs32         -         (3)         (2) <t< td=""><td></td><td>CMP.L #xx:32,ERd</td><td>_</td><td>9</td><td></td><td></td><td>+</td><td></td><td></td><td>-</td><td>+</td><td>山</td><td>Rd32–#xx:32</td><td>-</td><td></td><td>-</td><td></td><td>↔ •</td><td>e</td><td></td></t<>		CMP.L #xx:32,ERd	_	9			+			-	+	山	Rd32–#xx:32	-		-		↔ •	e	
NEGE Mat         B         Z         0         0-003→108         -         +		CMP:LERs,ERd	(		0	+	+	+		+	+	击 。 一	Rd32–ERs32	Ι		-		↔ (		
Network         <	NEG	NEG.B KO	≩מ		N C	+	+	+	+	+	+	5 0	-K08→K08 D416 \D416	I	_	-		→ ←		
EXTUWRd         W         2         O         O         (chits 15 to 8> of Rd16)         -         -         0         2         0         -           EXTULERd         L         2         0         O         O         O         O         0         10         1         0		NEG.L ERd	: _		1 0						+	- -	-ERd32→ERd32	1	$\rightarrow \leftrightarrow$	$\rightarrow \leftrightarrow$		$\leftrightarrow$		
EXTULERd         L         2         0         (-dits 31 to 16> of Ed32)          0         2         0          0         2         0          0         2         0          0         2         0          0         2         0          0         2         0          0         2         0          0         2         0          0         2         0          0         2         0          0         2         0          0         2         0          0         2         0          0         2         0          0         2         0          0         2         0          0         2         0          0         2         0          1         2         1	EXTU	EXTU.W Rd	>		2		-				-	0	→( bits 15 to 8> of Rd16)	I	1			1	-	
EXTS.W Rd         W         2         Q         (cbit 7> of Rd16)→(cbits 15 to 8>          1         1         2         0            EXTS.W Rd         W         2         O         O         Rd16)         O         O         P         0         O		EXTU.L ERd	_		7							0	→( bits 31 to 16> of ERd32)	Ι	<u> </u>			Ι	-	
EXTSL ERd         L         2         ( of ERd32)         ( cle Rd32)         ( 	EXTS	EXTS.W Rd	≥		2							€€	:bit 7> of Rd16)→( <bits 15="" 8="" to=""> . Rd16)</bits>	I				I	-	
TAS @ERd <sup>*6</sup> B     4     @ERd <sup>-0-5et</sup> CCR, 1→( $\bigcirc (2 + 1)^{-1})$ $\bigcirc (2 + 1)^{-1}$ $\bigcirc (2 + 1)^{-1}$		EXTS.L ERd	_		7							of (<	bit 15> of ERd32) →( bits 31 to 16> ERd32)	I					-	
	TAS	TAS @ERd <sup>*8</sup>	в			4						88	tERd–0→set CCR, 1→( bit 7> of ERd)						4	

ţ	States*1	pəວເ	вурА	4		2*6 *10	2*6 *10	2*6 *10	*6 *10	1 % K % 10
2	Sta	le	Morm			Š,	2*	2*	-	
			ပ	I			I	1	I	I
	Condition Code		>	I	(8)		1	1	$\leftrightarrow$	~
	tion (		N Z I	1	8		1	1	$\leftrightarrow$	~
	ondit		z		(8)				$\leftrightarrow$	÷
	ŏ		I						1	
			-				1	1	1	
	noiseast			@ERn×@ERm+MAC→MAC (signed	multiplication) ERn+2→ERn,ERm+2→ERm	0→MACH, MACL	ERs→MACH	ERs→MACL	MACH→ERd	
()			_			2				
(Byte:		e	200							
ength.		(၁4	l'p)@							
ction L			66 Ø							
Addressing Mode and Instruction Length (Bytes)	+u83	@/uչ	13-@	4						
de and		(u83	l'p)@							
ing Mo		u	83@							
Idressi			uЯ				2	2	2	~
Ad			xx#							
	Sizo	210				1	_	_	_	_
	Mnemonic			MAC @ERn+,@ERm+		CLRMAC	LDMAC ERS, MACH	LDMAC ERS, MACL	STMAC MACH, ERd	STMAC MACLERd
				MAC*9		CLRMAC*9	LDMAC*9		STMAC*9	

Addressing Mode and Instruction Isolution 2         Rin Fig         Mode Fig         Mode Fig <th>(3) Logic Operation Instructions</th> <th></th> <th></th> <th></th> <th></th> <th></th> <th>·  </th> <th></th> <th>ŀ</th> <th></th> <th></th> <th></th> <th></th> <th></th> <th></th> <th></th> <th>Γ</th>	(3) Logic Operation Instructions						·		ŀ								Γ
Condition         Condition         Condition           1 </th <th></th> <th>Addre</th> <th>essing N</th> <th>lode an</th> <th>d Instru</th> <th>ction L</th> <th>ength</th> <th>(Bytes)</th> <th></th> <th></th> <th></th> <th></th> <th></th> <th></th> <th></th> <th>No.</th> <th></th>		Addre	essing N	lode an	d Instru	ction L	ength	(Bytes)								No.	
2         3         4	Size				+u83					Oneration		Cond	ition	Cod	0	State	÷.
n         n	040		u	(uya	@/uչ		(Da	e								al	рәри
2         1         1         1         Rd8.4x:18-Rd8         1         <				( <b>p)</b>	I∃-@	66 Ø	( <b>p</b> )	20 0	_		-			-	ပ	Morm	вурА
2	B 2									Rd8∧#xx:8→Rd8	Ι	1			Ι	٢	
	В		2							Rd8∧Rs8→Rd8	Ι					٢	
2	W 4									Rd16∧#xx:16→Rd16	Ι	I				2	
	M		~						_	Rd16∧Rs16→Rd16	Ι					-	
	9 -									ERd32∧#xx:32→ERd32	Ι	I		-		e	
	_	1								ERd32∧ERs32→ERd32	Ι					2	
Image: black         Image: black <th< td=""><td>B 2</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>Rd8∨#xx:8→Rd8</td><td>I</td><td></td><td></td><td></td><td></td><td>٢</td><td></td></th<>	B 2									Rd8∨#xx:8→Rd8	I					٢	
Image: black state	В		2							Rd8∨Rs8→Rd8	I	1				٢	
$ \left  \begin{array}{c c c c c c c c c c c c c c c c c c c $	W 4									Rd16∨#xx:16→Rd16	Ι	-				2	
	M		2							Rd16∨Rs16→Rd16	Ι					•	
	L 6									ERd32∨#xx:32→ERd32	Ι					3	
Image: mark of the state of the st	L	7	4							ERd32∨ERs32→ERd32		1		_	-	2	
	B 2									Rd8⊕#xx:8→Rd8	Ι	1		_		٢	
2         Rd160#XX:16→Rd16          1         1         2         0            2         -         -         -         1         1         1         0            4         -         -         -         -         1         1         1         0            2         -         -         -         -         1         1         1         0            4         -         -         -         -         1         1         1         0            2         -         -         -         -         1         1         1         0            2         -         -         -         -         1         1         1         0            2         -         -         -         1 <td>в</td> <td>. 1</td> <td>~</td> <td></td> <td></td> <td></td> <td></td> <td>_</td> <td></td> <td>Rd8⊕Rs8→Rd8</td> <td>Ι</td> <td>_</td> <td></td> <td>_</td> <td>-</td> <td>-</td> <td></td>	в	. 1	~					_		Rd8⊕Rs8→Rd8	Ι	_		_	-	-	
2         Rd160Rs16-Rd16          2         2         0            4          -         -         -         -         -         2         0            2          -	W		_					_	-	Rd16⊕#xx:16→Rd16	Ι	1		_	Ι	7	
4         ERd32@#xx32~ERd32          2         2         0            2         -         -         -         -         -         -         2         0            2         -         -         -         -         -         -         1         1         1         0            2         -         -         -         -         -         1         1         1         0            2         -         -         -         -         -         1         1         1         0            2         -         -         -         -         -         1         1         0            2         -         -         -         -         -         1         1         0            2         -         -         -         -         -         1         1         0            2         -         -         -         -         1         1         1         1         1         1         1         1         1         1         1         1 </td <td>N</td> <td>. 1</td> <td>~</td> <td></td> <td></td> <td></td> <td></td> <td>_</td> <td>-</td> <td>Rd16⊕Rs16→Rd16</td> <td>Ι</td> <td>_</td> <td>-</td> <td>_</td> <td>-</td> <td>٢</td> <td></td>	N	. 1	~					_	-	Rd16⊕Rs16→Rd16	Ι	_	-	_	-	٢	
ERd32@ERs32_ERd32	L 6									ERd32⊕#xx:32→ERd32	Ι	1	_			3	
□         □         □         Rd8→Rd8         □<	L	7	+							ERd32⊕ERs32→ERd32	Ι		_	_	-	2	
¬ Rd16→Rd16         −         −         □ <th□< th="">         □         □         <th< td=""><td>В</td><td></td><td>~</td><td></td><td></td><td></td><td></td><td></td><td></td><td>⊐ Rd8→Rd8</td><td>Ι</td><td>-</td><td></td><td></td><td></td><td>1</td><td></td></th<></th□<>	В		~							⊐ Rd8→Rd8	Ι	-				1	
¬ Rd32→Rd32         −   −   ‡   ‡   0	Ν		~							- Rd16→Rd16		1				~	
	_		0							⊐ Rd32→Rd32	Ι					-	

Section 2 Instruction Descriptions

# Rev. 4.00 Feb 24, 2006 page 252 of 322 REJ09B0139-0400

	No. of States <sup>*1</sup>	3163		вурА	1	-	1	-	1	+	1	1	-	1	1	1	1	1	-	1	1	1	1	1	1	1	-	-	+	1	1	-	-	-
	z st	้อ้	ls	Norm																														
		æ		ပ	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	÷
		Cod		>	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4
		Condition Code		N Z	$\leftrightarrow$ $\leftrightarrow$	$\leftrightarrow$ $\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$ $\leftrightarrow$	$\leftrightarrow \leftrightarrow \Rightarrow$	$\leftrightarrow$ $\leftrightarrow$	$\Rightarrow$	⇔	$\leftrightarrow \leftrightarrow \leftrightarrow$	¢ ¢	↔ ¢	$\leftrightarrow \leftrightarrow \Rightarrow$	↔ ¢	$\leftrightarrow$	$\leftrightarrow$	¢ ↓	\$ \$	¢ ↓	¢ 0	≎ 0	↔ 0	⇔ 0	↔ 0	↔ 0	$\leftrightarrow \Rightarrow$	¢∣⊅	¢ ¢	$\leftrightarrow$ $\leftrightarrow$	$\leftrightarrow \leftrightarrow \Rightarrow$	* *
		Cond		I					1									1					-	-	1	1	1	1					1	ľ
		0		_	1							-	1	-	-	- 							- 	' 	1		1				-	1		
		Oberation												MSBCC				-		C MSB - LSB	_	-					2					C MSB + LSB		
tec)	(ea)			_																														
1th (Bv				200 200																														-
neluc			PC)	ʻp)@																														
Addressing Mode and Instruction Length (Bytes)		+u83	@/UY	@ 99																				_										
and In		23	(uya																					_										
Mode				974 МЭФ																														
ressing				u N	2	5	5	5	2	5	2	2	2	2	2	2	2	2	2	2	2	5	2	2	2	2	2	2	2	2	2	2	2	
Add				xx#																														
		Size			в	в	N	×	_	_	в	В	8	Ν	_	_	в	в	Ν	Ν	L	_	в	в	M	Ν	_	_	в	в	Ν	Ν	_	
		Mnemonic	2		SHAL.B Rd	SHAL.B #2,Rd	SHAL.W Rd	SHAL.W #2,Rd	SHAL.L ERd	SHAL.L #2,ERd	SHAR.B Rd	SHAR.B #2,Rd	SHAR.W Rd	SHAR.W #2,Rd	SHAR.L ERd	SHAR.L #2,ERd	SHLL.B Rd	SHLL.B #2,Rd	SHLL.W Rd	SHLL.W #2,Rd	SHLL.L ERd	SHLL.L #2,ERd	SHLR.B Rd	SHLR.B #2,Rd	SHLR.W Rd	SHLR.W #2,Rd	SHLR.L ERd	SHLR.L #2,ERd	ROTXL.B Rd	ROTXL.B #2,Rd	ROTXL.W Rd	ROTXL.W #2,Rd	ROTXL.L ERd	
					SHAL						SHAR						SHLL						SHLR						ROTXL					

7	States <sup>*1</sup>	pəsu	вурА		_	_	_	_	-	_	_	_	_	_	_	-	-	-	_	_	
2	Stat	le	Norm	Ì						Ì									Ì	·	
			ပ	$\leftrightarrow$																	
	Condition Code		>	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	tion (		N	$\leftrightarrow$																	
	ondi		z	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	↔ -	↔ -	$\leftrightarrow$	+										
	Ŭ		т _													 					 
	Oneration				1				-		1					1	1				-
													  							MSB	
3ytes)		pr	- 00																		
ength (E			'p)@																		
ction Le			66 D																		
Instru	+u83	@/u⊁	I∃-@																		
ode and		(uya	ʻp)@																		
Addressing Mode and Instruction Length (Bytes)		u	<b>В</b> ЕR																		
ddres			uЯ	7	7	7	2	2	2	7	2	2	2	7	2	2	2	2	2	2	2
◄			xx#																		
	Siza	210		в	ш	N	Ν	Γ	Γ	в	В	M	Ν	Г	L	В	В	Ν	W	Γ	L
	Mnemorio			ROTXR.B Rd	ROTXR.B #2,Rd	ROTXR.W Rd	ROTXR.W #2,Rd	ROTXR.L ERd	ROTXR.L #2,ERd	ROTL.B Rd	ROTL.B #2,Rd	ROTL.W Rd	ROTL.W #2,Rd	ROTL.LERd	ROTL.L #2,ERd	ROTR.B Rd	ROTR.B #2,Rd	ROTR.W Rd	ROTR.W #2,Rd	ROTR.L ERd	ROTR.L #2, ERd
				ROTXR						ROTL						ROTR					

#### Rev. 4.00 Feb 24, 2006 page 254 of 322 REJ09B0139-0400

ž	s*a	рәри	вурА																														
Ň	States *1	ls	Norm	-	4	4	5	9	-	4	4	5	9	1	4	4	5	9	-	4	4	5	9	-	4	4	5	9	-	4	4	5	9
			ပ	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι
	Condition Code		>	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	I	Ι	Ι	Ι	Ι	Ι	Ι
	ion C		N	1	Ι	Ι	Ι	Ι	Ι	Ι	Ι		1	Ι	Ι	Ι	Ι	Ι	Ι	Ι	Ι	I		Ι	Ι	Ι	I	1	Ι	Ι	Ι	Ι	Ι
	ndit		z	1		1	1										1								1				1	Ι			Ι
	ပိ		I				1				1		1				1							-			1	1		1			1
			-																		Ι								1				
	Oneration			(#xx:3 of Rd8)←1	(#xx:3 of @ERd)←1	(#xx:3 of @aa:8)←1	(#xx:3 of @aa:16)←1	(#xx:3 of @aa:32)←1	(Rn8 of Rd8)←1	(Rn8 of @ERd)←1	(Rn8 of @aa:8)←1	(Rn8 of @aa:16)←1	(Rn8 of @aa:32)←1	(#xx:3 of Rd8)←0	(#xx:3 of @ERd)←0	(#xx:3 of @aa:8)←0	(#xx:3 of @aa:16)←0	(#xx:3 of @aa:32)←0	(Rn8 of Rd8)←0	(Rn8 of @ERd)←0	(Rn8 of @aa:8)←0	(Rn8 of @aa:16)←0	(Rn8 of @aa:32)←0	(#xx:3 of Rd8)← [¬ (#xx:3 of Rd8)]	(#xx:3 of @ERd) ← [¬ (#xx:3 of @ERd)]	(#xx:3 of @aa:8)← [¬ (#xx:3 of @aa:8)]	(#xx:3 of @aa:16)← [¬ (#xx:3 of @aa:16)]	(#xx:3 of @aa:32)← [¬ (#xx:3 of @aa:32)]	(Rn8 of Rd8)← [¬ (Rn8 of Rd8)]	(Rn8 of @ERd)← [¬ (Rn8 of @ERd)]	(Rn8 of @aa:8) ← [¬ (Rn8 of @aa:8)]	(Rn8 of @aa:16) ← [¬ (Rn8 of @aa:16)]	(Rn8 of @aa:32)← [¬ (Rn8 of @aa:32)]
iytes)		pr	- 200							-																							
Addressing Mode and Instruction Length (Bytes)													_																				
n Len		LC)	l'p)@																														
tructio			66 D			4	9	80			4	9	8			4	9	80			4	9	80			4	9	8			4	6	8
sul bu	+u83	@/uչ	13-@																														
ode al		(u83	l'p)@																														
ing M		u	@ЕВ		4					4					4					4					4					4			
dress			uЯ	2					2					2					2					2					7				
Ä			xx#																														
	Size			۵	В	в	в	в	в	В	В	в	В	В	В	В	в	в	В	в	в	۵	в	В	в	В	ю	۵	۵	в	В	В	В
	Maemonic			BSET #xx:3,Rd	BSET #xx:3,@ERd	BSET #xx:3,@aa:8	BSET #xx:3,@aa:16	BSET #xx:3,@aa:32	BSET Rn, Rd	BSET Rn, @ERd	BSET Rn, @aa:8	BSET Rn, @aa:16	BSET Rn, @aa:32	BCLR #xx:3,Rd	BCLR #xx:3, @ERd	BCLR #xx:3, @aa:8	BCLR #xx:3, @aa:16	BCLR #xx:3, @aa:32	BCLR Rn, Rd	BCLR Rn, @ERd	BCLR Rn, @aa:8	BCLR Rn, @aa: 16	BCLR Rn, @aa: 32	BNOT #xx:3,Rd	BNOT #xx:3,@ERd	BNOT #xx:3,@aa:8	BNOT #xx:3,@aa:16	BNOT #xx:3,@aa:32	BNOT Rn, Rd	BNOT Rn, @ERd	BNOT Rn, @aa:8	BNOT Rn, @aa:16	BNOT Rn, @aa:32
				BSET										BCLR										BNOT									

Г

# RENESAS

Т Т Т Т Т 

Node and Ins		
(u원 (미원크	(uya	(uya
ʻp)@	@(q' @Ев	ʻp)@
		2
4	4	4
_		
_		
_		2
	4	4
_		
_		
_		
		2
-	4	4
		2
	4	4
		2
	4	4
		2
	4	4
	-	

No of	States <sup>*1</sup>	nced al	Morm 6vbA	-	3	3	4	5	-	3	3	4	5	+		з	е С	ю к 4	3 3	- 2 4 3 3	3 -1 5 4 3 3	3 3 3 5 6 7 8 3 3	ω ω <b>4</b> Ω <del>1</del> ω ω <b>4</b>	0 m 4 0 - 0 m 4 0	- 1 1 + 1 1 - 1 1 + 1 1 -	ω ω 4 ω τ ω ω 4 ω τ ω	ω ω 4 ω τ ω ω 4 ω τ ω	ω ω 4 ω τ ω ω 4 ω τ ω ω	x α 4 α - α x 4 α - α 4 α 7 4 α 3 4 8 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7	~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~	~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~	~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~	~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~	~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
			U	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$		↔	$\leftrightarrow$	↔	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	↔	↔	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	<;	_	_							+ $+$ $+$ $+$ $+$ $+$ $+$	+ $+$ $+$ $+$ $+$ $+$ $+$ $+$ $+$	+ $+$ $+$ $+$ $+$ $+$ $+$ $+$ $+$ $+$	+ + + + + + + + + + + + + + + + + + + +
	Condition Code		> N										 	 		 							-	_										
	dition		z							H	 		<u> </u> 	<u> </u>	1		 																	
	Conc		т	1	Ī	1	I	Ī	1	Ì		Ì	Ī	Ī	1			1						1										
			-	1	Ι	Ι	Ι		Ι			Ι	Ι	Ι	I	Ι	Ι	Ι					I		1	11	11	111						
	Onseration	0000		C∧(#xx:3 of Rd8)→C	C∧(#xx:3 of @ERd24)→C	C∧(#xx:3 of @aa:8)→C	C∧(#xx:3 of @aa:16)→C	C∧(#xx:3 of @aa:32)→C	C∧ [- (#xx:3 of Rd8)]→C	C∧ [¬ (#xx:3 of @ERd24)]→C	C∧ [¬ (#xx:3 of @aa:8)]→C	C∧ [¬ (#xx:3 of @aa:16)]→C	C∧ [¬ (#xx:3 of @aa:32)]→C	C∨(#xx:3 of Rd8)→C	C√(#xx:3 of @ERd24)→C	C∨(#xx3: of @aa:8)→C	C∨(#xx3: of @aa:16)→C	C∨(#xx3: of @aa:32)→C	C∨ [¬ (#xx:3 of Rd8)]→C	C∨ [¬ (#xx:3 of @ERd24)]→C	C∨ [¬ (#xx:3 of @aa:8)]→C	C∨ [¬ (#xx:3 of @aa:16)]→C	C∨ [¬ (#xx:3 of @aa:32)]→C		C⊕(#XX:3 OT Kd8)→C	C⊕(#xx:3 of Kd8) →C C⊕(#xx:3 of @ERd24) →C	C⊕(#xx:3 of @ERd24)→C C⊕(#xx:3 of @ERd24)→C	C⊕(#xx:3 or ka8) →C C⊕(#xx:3 of @ERd24) →C C⊕(#xx:3 of @aa:8) →C	C⊕(#xx:3 of Ro8)→C C⊕(#xx:3 of @ERd24)→C C⊕(#xx:3 of @aa:8)→C C⊕(#xx:3 of @aa:16)→C	C@(#xx:3 of Kd9)→C C@(#xx:3 of @ERd24)→C C@(#xx:3 of @aa:8)→C C@(#xx:3 of @aa:32)→C C@(#xx:3 of @aa:32)→C	C⊕(#xx:3 of KdB)→C C⊕(#xx:3 of @ERd24)→C C⊕(#xx:3 of @aa:9)→C C⊕(#xx:3 of @aa:16)→C C⊕(#xx:3 of @aa:16)→C C⊕(#xx:3 of RdB)]→C C⊕(- (#xx:3 of RdB)]→C	C⊕(#xx:3 or KdB)→C C⊕(#xx:3 of @ERd24)→C C⊕(#xx:3 of @aa:9)→C C⊕(#xx:3 of @aa:16)→C C⊕(#xx:3 of @aa:16)→C C⊕(+ (#xx:3 of @ab])→C C⊕(- (#xx:3 of @ERd24))→C	C⊕(#xx:3 of KdB)→C C⊕(#xx:3 of @ERd24)→C C⊕(#xx:3 of @aa:16)→C C⊕(#xx:3 of @aa:16)→C C⊕(#xx:3 of @aa:22)→C C⊕(1 (#xx:3 of @aa:32)→C C⊕(- (#xx:3 of @aa:8)]→C C⊕(- (#xx:3 of @aa:8)]→C C⊕(- (#xx:3 of @aa:8)]→C	C@(#xx:3 or Kd9)→C C@(#xx:3 of @ ERd24)→C C@(#xx:3 of @ aa:16)→C C@(#xx:3 of @ aa:22)→C C@(+#xx:3 of @ aa:32)→C C@(-1 (#xx:3 of @ aa:31)→C C@(-1 (#xx:3 of @ aa:16))→C C@(-1 (#xx:3 of @ aa:16))→C C@(-1 (#xx:3 of @ aa:16))→C
			-																															
		ee	00																															
		PC)	ʻp)@																															
			66 Ø			4	6	8			4	6	8			4	9	œ			4	6	8					4	4 0	6 6 8	4 0 8	4 0 8	4 0 8 4	4 0 8 4 0
	+u83	@/uଧ	∃-@																															
		(uya	ʻp)@																															
2		u	9ER		4					4					4					4					4							4	4	4
רמתו בפצוווה ושסתב מוות ווופנו תכנוסוו דבווהנוו (ה) נכפו			uЯ	2					2					2					2					2							2	7	2	N
			xx#																															
	Si70	2710		m	ш	В	В	в	m	в	в	В	В	В	в	В	в	B	m	в	В	В	В	в	в	8	۵		в	<u></u> в	<u> </u>	<u> </u>	<u> </u>	<u> </u>
	Maemonic			BAND #xx:3,Rd	BAND #xx:3,@ERd	BAND #xx:3,@aa:8	BAND #xx:3,@aa:16	BAND #xx:3,@aa:32	BIAND #xx:3,Rd	BIAND #xx:3, @ERd	BIAND #xx:3, @aa:8	BIAND #xx:3, @aa:16	BIAND #xx:3, @aa:32	BOR #xx:3,Rd	BOR #xx:3,@ERd	BOR #xx:3,@aa:8	BOR #xx:3,@aa:16	BOR #xx:3,@aa:32	BIOR #xx:3,Rd	BIOR #xx:3,@ERd	BIOR #xx:3,@aa:8	BIOR #xx:3,@aa:16	BIOR #xx:3,@aa:32	BXOR #xx:3,Rd	BXOR #xx:3,@ERd	BXOR #xx:3,@aa:8	BXOR #xx:3,@aa:8		BXOR #xx:3,@aa:16	BXOR #xx:3,@aa:16 BXOR #xx:3,@aa:32	BXOR #xx:3,@aa:16 BXOR #xx:3,@aa:32 BIXOR #xx:3,Rd	BXOR #xx:3,@aa:16 BXOR #xx:3,@aa:32 BIXOR #xx:3,Rd BIXOR #xx:3,@ERd	BXOR #xx:3, @aa: 16 BXOR #xx:3, @aa: 32 BIXOR #xx:3, Rd BIXOR #xx:3, @ERd BIXOR #xx:3, @aa:8 BIXOR #xx:3, @aa:8	BXOR #xx:3,@aa:16 BXOR #xx:3,@aa:32 BIXOR #xx:3,Rd BIXOR #xx:3,@ERd BIXOR #xx:3,@aa:8 BIXOR #xx:3,@aa:16 BIXOR #xx:3,@aa:16
				BAND					BIAND					BOR					BIOR					BXOR							ŚOR	XOR	BIXOR	XOR

			Add	dressin	g Mode	and Insti	ruction	Length	Addressing Mode and Instruction Length (Bytes)								2	ž
		0				+u83							Ŭ	Condition Code	tion Co	ode	<sup>2</sup> ở	States <sup>*1</sup>
		azic				1@/uչ		(၁८	e		Operation						le	
			xx#	uЯ	@Eצ @נמיו		66 D	l,b)@	200	_		Branch Condition	-	z I	и	>	Norma	вурА
Bcc	BRA d:8(BT d:8)	I						7			if condition is true then	Always	1		1			2
	BRA d:16(BT d:16)	I						4			PC←PC+d		1		1			e
	BRN d:8(BF d:8)	1						7			else next;	Never	1	1	1	1	1	5
	BRN d: 16(BF d: 16)	I						4					1		1	1		e
	BHI d:8	I						7				Cvz=0	1		1			2
	BHI d:16	I						4					1		I	1		e
	BLS d:8	I						2				Cvz=1	1		1	1		5
	BLS d:16	Ι						4					1		1		1	в
	BCC d:8(BHS d:8)	I						2				C=0	1		1		1	2
	BCC d: 16(BHS d: 16)	Ι						4										3
	BCS d:8(BLO d:8)	Ι						2				C=1	I		1			2
	BCS d:16(BLO d:16)	Ι						4							I			з
	BNE d:8	Ι						2				Z=0	İ		Ι	- 		2
	BNE d:16	Ι						4					1					3
	BEQ d:8	Ι						2				Z=1	I		1		_	2
	BEQ d:16	I						4					1		1	1		e
	BVC d:8	I						7				V=0	1		I	1		5
	BVC d:16	I						4					1		1			<del>ر</del>
	BVS d:8	Ι						2				V=1	1					2
	BVS d:16	Ι						4					1					3
	BPL d:8	Ι						2				N=0	I		1			2
	BPL d:16	Ι						4							1			3
	BMI d:8	I						2				N=1			1	 		2
	BMI d:16	Ι						4							I			в
	BGE d:8	Ι						2				N⊕V=0	1					2
	BGE d:16	Ι						4					I		1			3
	BLT d:8	Ι						2				N⊕V=1	I		1			2
	BLT d:16	I						4					İ	 	Ι	1		e
	BGT d:8	Ι						2				Z√(N⊕V)=0	İ	 	Ι	- 		2
	BGT d:16	Ι						4					I	1	1	1	1	в
	BLE d:8	I				_		2				Z√(N⊕V)=1	1	1	1	1	1	2
	BLE d:16	Ι	_	_	_			4	_	_			İ	 	1	' 		с,

Rev. 4.00 Feb 24, 2006 page 258 of 322 REJ09B0139-0400

			Adc	dressir	g Mod	Addressing Mode and Instruction Length (Bytes)	struct	ion Ler	ngth (E	sytes)										7
	ci no mon	Cito Cito					+ия=					noitenen			Cond	ition	Condition Code	0	No. of States <sup>*1</sup>	ss*1
		ATIC			u		1@/uչ			PI		Operation							le	pəɔı
			xx#	uЯ	@ЕВ	ı'p)@		66.00	l,b)@	•@@			Branch Condition	-	Ŧ	7	> N Z	ပ	Norm	levbA
JMP	JMP @ERn	Ι			2						₫.	PC←Ern		Ι	1			Ι	2	
	JMP @aa:24	I						4			₫.	PC←aa:24		Ι				Ι		e
	JMP @@aa:8	I								2	₽.	PC← @aa:8		Ι					4	5
BSR	BSR d:8	I							5		₽.	PC→@-SP,PC←PC+d:8		Ι					e	4
	BSR d:16	I						7	4		۵.	PC→ @-SP,PC←PC+d:16	0	Ι				I	4	5
JSR	JSR @ERn	I			2						۵.	PC→@-SP,PC←ERn		Ι	-		-	I	3	4
	JSR @aa:24	I					-	4			₫.	PC→@-SP,PC←aa:24		Ι				Ι	4	5
	JSR @@aa:8	I								2	₽.	PC→@–SP,PC←aa:8		Ι		 			4	9
RTS	RTS	Ι								5	2 P	PC←@SP+						I	4	5

	of	5*3	pəsu	івурА	7 8[9]*7	5[9]*7																	
	No. of	States*1	al	Morm	7[9]*7	2[9	2	-	2	1	1	3	ę	4	4	9	9	4	4	4	4	5	ıс.
				ပ	Ι	$\leftrightarrow$	Ι	$\leftrightarrow$	Ι	$\leftrightarrow$	I	$\leftrightarrow$		$\leftrightarrow$	I	$\leftrightarrow$	I	$\leftrightarrow$	Ι	$\leftrightarrow$	I	$\leftrightarrow$	1
l		Condition Code		>	1	$\leftrightarrow$	Ι	$\leftrightarrow$	Ι	$\leftrightarrow$		$\leftrightarrow$	Ι	$\leftrightarrow$	Ι	$\leftrightarrow$	Ι	$\leftrightarrow$	Ι	$\leftrightarrow$	I	$\leftrightarrow$	1
l		on C		N	1	$\leftrightarrow$	1	$\leftrightarrow$	Ι	$\leftrightarrow$	Ι	$\leftrightarrow$	Ι	$\leftrightarrow$	Ι	$\leftrightarrow$	1	$\leftrightarrow$	1	$\leftrightarrow$		$\leftrightarrow$	1
l		nditi		z	1	$\leftrightarrow$	1	$\leftrightarrow$	Ι	$\leftrightarrow$	Ι	$\leftrightarrow$	1	$\leftrightarrow$	Ι	$\leftrightarrow$	1	$\leftrightarrow$	1	$\leftrightarrow$		$\leftrightarrow$	1
l		ပိ		I		$\leftrightarrow$	1	$\leftrightarrow$		$\leftrightarrow$		$\leftrightarrow$	1	$\leftrightarrow$		$\leftrightarrow$	1	$\leftrightarrow$	1	$\leftrightarrow$		$\leftrightarrow$	1
┝				-	-	$\leftrightarrow$		$\leftrightarrow$		$\leftrightarrow$		$\leftrightarrow$		$\leftrightarrow$	Ι	$\leftrightarrow$	Ι	$\leftrightarrow$		$\leftrightarrow$		$\leftrightarrow$	1
		Oneration			PC→@-SPCCR→@-SP, EXR→@-SP, <vector>→PC</vector>	EXR←@SP+,CCR←@SP+, PC←@SP+	Transition to power-down state	#xx:8→CCR	#xx:8→EXR	Rs8→CCR	Rs8→EXR	@ERs→CCR	@ERs→EXR	@(d:16,ERs)→CCR	@(d:16,ERs)→EXR	@(d:32,ERs)→CCR	@(d:32,ERs)→EXR	@ERs→CCR,ERs32+2→ERs32	@ERs→EXR,ERs32+2→ERs32	@aa:16→CCR	@aa:16→EXR	@aa:32→CCR	@aa:32→FXR
	ss)			_	2																		
	Addressing Mode and Instruction Length (Bytes)		66	°00																			
	Lengt		(D4	l'p)@																			
	uction			66 Ø																9	9	8	œ
	d Instr	+u83	@/uչ	13-@														4	4				
	de an		(uy3	l'p)@										9	9	10	10						
	ing Ma		u	<b>@Е</b> В								4	4										
	dress			uЯ						2	2												
	Ă			xx#				2	4														
		Size	010		1	I	I	B	в	в	в	Ν	×	×	Μ	Ν	N	×	×	≥	≥	Ν	>
		Mnemonic			TRAPA #x:2	RTE	SLEEP	LDC #xx:8,CCR	LDC #xx:8,EXR	LDC Rs,CCR	LDC Rs,EXR	LDC @ERs,CCR	LDC @ERs,EXR	LDC @(d:16,ERs),CCR	LDC @(d:16,ERs),EXR	LDC @(d:32,ERs),CCR	LDC @(d:32,ERs),EXR	LDC @ERs+,CCR	LDC @ERs+,EXR	LDC @aa:16,CCR	LDC @aa:16,EXR	LDC @aa:32,CCR	LDC @aa:32.EXR
					TRAPA	RTE	SLEEP	LDC															

(7) System Control Instructions

			Add	Addressing Mode and Instruction Length (Bytes)	Mode	and Ins	structi	on Len	gth (By	tes)							_	1
- month						+u8=							0	Condition Code	ion C	ode	- 0	No. of States <sup>*1</sup>
		ATIC				1@/uչ		()d			Operation						l	
			xx#	u Sn	@Eצ @Eצ			l'p)@	200	_			_	z I	N	>	Norma	ISVDA
STC CCR,Rd	:R,Rd	в		2							CCR→Rd8		1		Ι	Ι	1	-
STC EXR,Rd	R,Rd	m		5							EXR→Rd8		1		1	Ι		-
STC CC	STC CCR,@ERd	≥		1	4						CCR→@ERd		1		1	Ι	1	e
STC EX.	STC EXR,@ERd	M		7	4						EXR→@ERd				Ι	Ι	Ι	з
STC CC	STC CCR, @ (d: 16, ERd)	M			9						CCR→@(d:16,ERd)				I	Ι	Ι	4
STC EX.	STC EXR,@(d:16,ERd)	M			9						EXR→@(d:16,ERd)		1		Ι	Ι		4
STC CC	STC CCR, @ (d: 32, ERd)	N			10	0					CCR→@(d:32,ERd)				1	Ι		9
STC EX	STC EXR,@(d:32,ERd)	N			10	0					EXR→@(d:32,ERd)				1	Ι		9
STC CC	STC CCR, @-ERd	W				4					ERd32-2→ERd32,CCR→@ERd	۶d		 	Ι	Ι		4
STC EX.	STC EXR,@-ERd	M				4					ERd32-2→ERd32,EXR→@ERd	sd	1		I	I	Ι	4
STC CC	STC CCR,@aa:16	M					9				CCR→@aa:16				1	I		4
STC EX	STC EXR,@aa:16	M					9				EXR→@aa:16		' 		Ι	Ι		4
STC CC	STC CCR, @ aa:32	N					8				CCR→@aa:32		<u> </u>	   		Ι		5
STC EX	STC EXR,@aa:32	N					80				EXR→@aa:32		<u> </u>	   	1	I		5
ANDC ANDC#	ANDC #xx:8,CCR	В	2								CCR∧#xx:8→CCR		$\leftrightarrow$	$\Rightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	1
ANDC #	ANDC #xx:8,EXR	В	4								EXR∧#xx:8→EXR		1		Ι	I	Ι	2
ORC #x:	ORC #xx:8,CCR	В	2								CCR√#xx:8→CCR		$\leftrightarrow$	¢ ¢	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	-
ORC #x:	ORC #xx:8,EXR	в	4								EXR∨#xx:8→EXR		- 	 	Ι	Ι		2
XORC XORC #	XORC #xx:8,CCR	В	2								CCR⊕#xx:8→CCR		$\leftrightarrow$	$\leftrightarrow \leftrightarrow \Rightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	-
XORC #	XORC #xx:8,EXR	В	4								EXR⊕#xx:8→EXR		-		Ι	Ι		2
NOP		I								2	PC←PC+2		1		Ι			٢

Blo	(8) Block Transfer Instructions	s														ľ		
		1	Ρq	ldressir	ng Moc	le and	Instruc	tion Le	Addressing Mode and Instruction Length (Bytes)	ytes)							Ň	ţ
	SincmenM	Size					+u83					Oneration		Condition Code	on Code	-	States <sup>*1</sup>	s*1
		010	xx#	uЯ	u83@	(uЯ∃,b)@	@/u83-@	66 B.6	@ @ 99 0 (q,PC)				-	z	> 2	ပ	Normal	beonsvbA
EEPMOV	/ EEPMOVB	1								4		if R4L ≠ 0 Repost @ER5+→@ER6+ ER5+1→ER5 ER6+1→ER5 R4L-1→R4L Until R4L=0 Until R4L=0 else next:	1			1	4+2n*2	*2
	EEPMOV.W	1								4		if R4 ≠ 0			1	1	4+2n*2	*2
Notes:	<ol> <li>The number of states is the numbe</li> <li>n is the initial setting of R4L or R4.</li> <li>Seven states for saving or restoring</li> <li>One additional state is required for are required for execution of a MUI</li> </ol>	the nur 84L or   5r restc 3quired n of a h	mber ( R4. oring to I for e) MULX	of state wo reg kecutio U instr	is requiring the second	uired fe , nine : nediate withir	or exe states I afte three	cution for threat a ML states	when t ee regi: JLXU, f after e	he ins sters, MULX execu	tructio or elev S, or S tion of	The number of states is the number of states required for execution when the instruction and its operands are located in on-chip memory. In is the initial setting of R4L or R4. Seven states for saving or restoring two registers, nine states for three registers, or eleven states for four registers. One additional state is required for execution immediately after a MULXU, MULXS, or STMAC instruction. Also, a maximum of three additional states are required for execution of a MULXU instruction within three states after execution of a MAC instruction. For example, if there is a one-state instruction	ed in on naximun nple, if t	-chip me n of thre here is (	emory. ee addit a one-s	tional state i	state	s ction
~	(such as NOP) between 5. A maximum of two additi For example, if there is a state longer.	a MAC ional st i one-s	C instru tates a tate in	uction ; are req istruction	and a uired on (st	MULX for exe ich as	U inst cutior NOP)	ructior of a N betwe	i, the N AULXS en a M	IULXL instru AC in	J instru uction v structio	(such as NOP) between a MAC instruction and a MULXU instruction, the MULXU instruction will be two states longer. A maximum of two additional states are required for execution of a MULXS instruction within two states after execution of a MAC instruction. For example, if there is a one-state instruction (such as NOP) between a MAC instruction and a MULXS instruction, the MULXS instruction will be one state longer.	er. ion of a the MU	MAC in LXS ins	Istructic	n. Will	be on	Ð
÷	<ol> <li>A maximum of three add For example, if there is a longer.</li> </ol>	litional t one-s	states itate in	s are re istructi	equired on (su	d for e: ich as	kecutik NOP)	on of o betwe	ne of th en a M	AC in	nstruct	A more region of three additional states are required for execution of one of these instructions within three states after execution of a MAC instruction. A maximum of there is a one-state instruction (such as NOP) between a MAC instruction and one of these instructions, that instruction will be two states concert.	executi ons, tha	on of a l t instruc	MAC in tion wil	I be t	tion. vo sta	ates
1- 00 0.	<ol> <li>T. Values in parentheses () are for the H8S/2000 CPU. Values in square brackets [] ar</li> <li>Only register ER0, ER1, ER4, or ER5 should be used when using the TAS instruction.</li> <li>These instructions are supported only by the H8S/2600 CPU.</li> </ol>	) are fi ER4, c upporte	or the or ER5 only	H8S/2 5 shoul	d be u BBS	PU. V sed w 2600	alues hen u: CPU.	in squi sing th	are bra e TAS	ckets instru	[] app ction.	ougor. Values in parentheses()are for the H8S/2000 CPU. Values in square brackets[]apply to interrupt control modes 2 and 3. Only register ER0, ER1, ER4, or ER5 should be used when using the TAS instruction. These instructions are supported only by the H8S/2600 CPU.	2 and 3	m				
		ay diffe quired t borrow	for exervice	ecution st bi	on the of an it 11; c	e prod instru otherw	uct. Fc ction t ise cle	or deta hat tra	ils, refe nsfers 2 0.	er to th data i	n sync	The number of states may differ depending on the product. For details, refer to the relevant microcontroller hardware manual of the product in question. The number of states required for execution of an instruction that transfers data in synchronization with the E clock is variable. Set to 1 when a carry or borrow occurs at bit 11; otherwise cleared to 0.	e manua is variab	al of the ole.	produc	ct in q	uestic	U
		borrow e wher is neg	v occu n the r jative;	irs at b esult is otherw	it 27; ( s zero vise cl	otherw ; other eared	ise clé wise c to 0.	eared t <sub>i</sub> leared	o 0. to 0.									
	<ul> <li>(6) Set to 1 when the divisor is zero; otherwise cleared to 0.</li> <li>(7) Set to 1 when the quotient is negative; otherwise cleared to 0.</li> <li>(8) MAC instruction results are indicated in the flags when FXD is valid instruction is executed.</li> <li>(9) One additional evants is convinced for execution when FXD is valid</li> </ul>	r is zeru nt is ne are indi	o; oth( egative icated	erwise e; othe in the	clear rwise flags	ed to 0 cleare when t	d to 0. he ST	MAC ii	nstructi	ion is	execut	ted.						
-		naiinba	10 10	Xecure		IL LA	k Is va											

Rev. 4.00 Feb 24, 2006 page 262 of 322 REJ09B0139-0400

		F						Instruction Format	ר Format				
Instruction	Mnemonic	Size	1st	1st byte	2nd byte	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	9th byte	10th byte
ADD	ADD.B #xx:8,Rd	в	∞	Þ	IMM			•			•		
	ADD.B Rs,Rd	ю	0	80	rs								
	ADD.W #xx:16,Rd	8	7	6	1 rd	=	IMM						
	ADD.W Rs,Rd	Ν	0	6	rs rd								
	ADD.L #xx:32,ERd	L	7	A	1 0 erd		2	IMM					
	ADD.L ERS,ERd	-	0	A	1 ers 0 erd								
ADDS	ADDS #1,ERd		0	æ	0 0: erd								
	ADDS #2,ERd	_	0	æ	8 :0: erd								
	ADDS #4,ERd	L	0	В	9 0 erd								
ADDX	ADDX #xx:8,Rd	m	6	Þ	IMM								
	ADDX Rs,Rd	B	0	ш	rd s								
AND	AND.B #xx:8,Rd	æ	ш	Þ	IMM								
	AND.B Rs,Rd	в	-	9	rs rd								
	AND.W #xx:16,Rd	3	~	ი	9	=	IMM						
	AND.W Rs,Rd	3	9	9	rd s								
	AND.L #xx:32,ERd	L	7	A	6 : 0: erd		2	IMM					
	AND.L ERS, ERd	_	0	-	0 L	9	0: ers : 0: erd						
ANDC	ANDC #xx:8,CCR	æ	0	9	IMM								
	ANDC #xx:8,EXR	в	0	-	4	0 6	MMI						
BAND	BAND #xx:3,Rd	в	2	9	0.IMM rd								
	BAND #xx:3,@ERd	m	~	ပ	0 erd 0	7 6	0 IMMI 0						
	BAND #xx:3,@aa:8	B	~	ш	abs	7 6	0 IMM 0						
	BAND #xx:3,@aa:16	в	9	A	1		abs	2 6	0 IMMi 0				
	BAND #xx:3,@aa:32	в	9	A	3 0		a	abs		7 6	0 IMMI 0		
Bcc	BRA d:8 (BT d:8)	1	4	0	disp								
	BRA d:16 (BT d:16)	I	ഹ	80	0	U	disp						
	BRN d:8 (BF d:8)		4	-	disp								
	BRN d:16 (BF d:16)	1	ъ	œ	-	0	disp						
	BHI d:8	1	4	2	disp								
	BHI d:16		ъ	8	2 0	0	disp						
	BLS d:8	Ι	4	з	disp								
	BLS d:16	Ι	2	8	3 0	υ	disp						
	BCC d:8 (BHS d:8)	Ι	4	4	disp								

# 2.4 Instruction Code

# Table 2.2Instruction Codes

		į						Instructio	Instruction Format				
Instruction	Mnemonic		1st byte	fe	2nd byte	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	9th byte	10th byte
Bcc	BCC d:16 (BHS d:16)		5	8	4 : 0	dis	disp						
	BCS d:8 (BLO d:8)		4	5	disp								
	BCS d:16 (BLO d:16)	1	 2	8	5	di	disp						
	BNE d:8		4	9	disp								
	BNE d:16	1	5	8	6 0	di	disp						
	BEQ d:8	I	4	7	disp								
	BEQ d:16	1	5	8	7 : 0	disp	sp						
	BVC d:8	I	4	8	disp								
	BVC d:16	1	 2	8	8	disp	ds						
	BVS d:8	1	4	6	disp								
	BVS d:16	1	 2	8	0	dis	disp						
	BPL d:8	1	4	A	disp								
	BPL d:16	1	 20	8	0 	dis	disp						
	BMI d:8	1	4	æ	disp								
	BMI d:16	1	 2	8	о  В	ġ	disp						
	BGE d:8	1	4	υ	disp								
	BGE d:16	1	 2	8	0 C	dis	disp						
	BLT d:8	1		۵	disp								
	BLT d:16	1	 го	8	о 	di	disp						
	BGT d:8	1	4	ш	disp								
	BGT d:16	1	 2	8	о  Ш	di	disp						
	BLE d:8	1	4	ш	disp								
	BLE d:16	1	 2	8	0 	đị	disp						
BCLR	BCLR #xx:3,Rd	m		2	0 IMM rd								
	BCLR #xx:3,@ERd	В	2	٥	0 erd 0	7 2	0 [MM] 0						
	BCLR #xx:3,@aa:8	ш	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~	ш	abs	7 2	0 IMMI 0						
	BCLR #xx:3,@aa:16	m	9	A	-	at a	abs	7 2	0 IMMI 0				
	BCLR #xx:3,@aa:32	m		A	8 		abs	S		7 2	0 IMMI: 0		
	BCLR Rn,Rd	В	9	2	rn 								
	BCLR Rn,@ERd	В	7	٥	0 erd 0	6 2	0 1						
	BCLR Rn,@aa:8	ш	~~~~	ш	abs	6 2	0 						
	BCLR Rn,@aa:16	B		۲	1	at	abs	6	0 				
	BCLR Rn,@aa:32	ß	9	A	з 		at	abs		6 2	m 0		

acito intera	Macmonia								Instructic	Instruction Format				
		270	1st	1st byte	2nd	2nd byte	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	9th byte	10th byte
BIAND	BIAND #xx:3,Rd	m	7	9	1 IMM	P								
	BIAND #xx:3,@ERd	в	7	с 	0 erd	0	7 6	1 IMM 0						
	BIAND #xx:3,@aa:8	m	4	ш	B	abs	7 6	1 IMM 0						
	BIAND #xx:3,@aa:16	ш	9	∢	-	0		abs	7 6	1 : IMM: 0				
	BIAND #xx:3,@aa:32	в	9	Α	3	0		G	abs		7 6	1 IMM 0		
BILD	BILD #xx:3,Rd	m	2	~	1 IMM	5								
	BILD #xx:3,@ERd	ш	7	<u>с</u>	0 erd	0	7 2 7	1 IMM 0						
	BILD #xx:3,@aa:8	m	2	ш	8 D	abs	7 7	1 IMM 0						
	BILD #xx:3,@aa:16	۵	9	∢	-	0		abs	7 7	1 IMM 0				
-	BILD #xx:3,@aa:32	m	9	₹	e	0		.0	abs		7 : 7	1 IMM 0		
BIOR	BIOR #xx:3,Rd	۵	2	4	1 IMM	5								
	BIOR #xx:3,@ERd	m	~	<u>с</u>	0 erd	0	7 4	1 IMM 0						
	BIOR #xx:3,@aa:8	۵	2	ш	a	abs	7 4	1 IMM 0						
-	BIOR #xx:3,@aa:16	m	9	<	-	0		abs	7 4	1 IMM 0				
	BIOR #xx:3,@aa:32	۵	9	<	e	0		0	abs		7 : 4	1 IMM		
BIST	BIST #xx:3,Rd	۵	9	~	1 IMM	Ð								
-	BIST #xx:3,@ERd	۵	2		0 erd	0	6 7	1 [IMM] 0						
	BIST #xx:3,@aa:8	۵	7	<b>ц</b>	50 I	abs	6 7	1 IMM 0						
-	BIST #xx:3,@aa:16	m	9	<	-	∞		abs	6 7	1 IMM 0				
	BIST #xx:3,@aa:32	в	9	¥	3	8		σ,	abs		2 9	1 IMM 0		
BIXOR	BIXOR #xx:3,Rd	m	2	22 	1 IMM	5								
	BIXOR #xx:3,@ERd	в	7	C	0 erd	0	7 5	1 IMM 0						
	BIXOR #xx:3,@aa:8	ю	7	ш 	g	abs	7 5	1 IMM 0						
	BIXOR #xx:3,@aa:16	ш	9	4	-	0		abs	7 5	1 IMM 0				
	BIXOR #xx:3,@aa:32	в	9	Α	3	0		G	abs		7 5	1 IMM 0		
BLD	BLD #xx:3,Rd	в	7	. 7	0 IMM	1: rd								
	BLD #xx:3,@ERd	m	4	U	0 erd	0	7 7	0 [MMI] 0						
	BLD #xx:3,@aa:8	В	7	ш	а	abs	7 7	0 IMM 0						
	BLD #xx:3,@aa:16	в	9	A	٢	0		abs	7 7	0 IMMI 0				
	BLD #xx:3,@aa:32	ш	9	4	e	0		10	abs		7 7	0 IMM 0		
BNOT	BNOT #xx:3,Rd	ш	7	<del>.</del>	0 IMM	5								
	BNOT #xx:3,@ERd	۵	7		0 erd	0	7 1	0 IMMI 0						
	BNOT #xx:3,@aa:8	ш	7	ш. 	a	abs	7 1	0 IMM 0						
	BNOT #xx:3,@aa:16	۵	9	4	-	80		abs	7 1	0 [IMM] 0				
	BNOT #xx:3,@aa:32	ш	9	4	з	80		G	abs		7 1	0 IMMI 0		
	BNOT Rn,Rd	ш	9		5	ъ								

BNOT Rn.@ ERd           BNOT Rn.@ aa:8           BNOT Rn.@ aa:3           BNOT Rn.@ aa:3           BOR #xx:3, Rd           BOR #xx:3, @ aa:8           BOR #xx:3, @ aa:8           BOR #xx:3, @ aa:16           BOR #xx:3, @ aa:16           BOR #xx:3, @ aa:32           BOR #xx:3, @ aa:16           BOR #xx:3, @ aa:16           BOR #xx:3, @ aa:16           BOR #xx:3, @ aa:16           BSET #xx:3, @ aa:16           BSET #xx:3, @ aa:16           BSET #xx:3, @ aa:16           BSET #xx:3, @ aa:16           BSET #xx:3, @ aa:16           BSET #xx:3, @ aa:16           BSET #xx:3, @ aa:16           BSET #xx:3, @ aa:16           BSET #xx:3, @ aa:16           BSET #xx:3, @ aa:16           BSET #xx:3, @ aa:16           BST #xx:3, @ aa:16           BST #xx:3, @ aa:16           BST #xx:3, @ aa:16           BST #xx:3, @ aa:16           BST #xx:3, @ aa:16           BST #xx:3, @ aa:16           BST #xx:3, @ aa:16           BST #xx:3, @ aa:16           BST #xx:3, @ aa:16           BST #xx:3, @ aa:16           BST #xx:3, @ aa:16           BST #xx:3, @ aa:16           B	Instruction	Mnemonic	Siza							Instructi	Instruction Format				
BNOTRAGENCIA         B         T         D </th <th></th> <th></th> <th></th> <th>1st b</th> <th>yte</th> <th>2nd by</th> <th>te</th> <th>3rd byte</th> <th>4th byte</th> <th>5th byte</th> <th>6th byte</th> <th>7th byte</th> <th>8th byte</th> <th>9th byte</th> <th>10th byte</th>				1st b	yte	2nd by	te	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	9th byte	10th byte
BVOT Fine arease         B         7         F         abs         F         abs         F         abs         F         abs         F         abs         F         f </td <td>BNOT</td> <td>BNOT Rn,@ERd</td> <td>в</td> <td>7</td> <td></td> <td>0 erd</td> <td>0</td> <td>6 1</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td>	BNOT	BNOT Rn,@ERd	в	7		0 erd	0	6 1							
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$		BNOT Rn,@aa:8	ш	7	ш	abs		6							
BNOT Thrundsmar32         B         A         3         8         A         3         6         1         4         1           BON WAX:30EHd         B         7         4         0.1MM         10         7         4         0.1MM         1         4         1         4         1         4         1         4         0.1MM         1         1         4         0.1MM		BNOT Rn,@aa:16	в		A	1	8		abs	6 1					
BOK RAX-3/REM         B         7         4         0/1M/i         0         1		BNOT Rn,@aa:32	ш	9	٨	 С	8		U	abs					
BOR kwc.3 @erd         B         7         C         0 end         0         7         4         0 iMM         0         7         4         0         7         4         0         7         4         0         7         4         0         1         0         1         0         1<	BOR	BOR #xx:3,Rd	в	7		MMI: 0	P								
BOR kwc3.(8)ast         B         7         4         0.10M.         0         1         0         1         0         1         0         1         0         1         0         1         0         1         0         1         0         1         0         1         0         1         0         1		BOR #xx:3,@ERd	в	7		0 erd	0								
BOR #xx3.@artif         B         A         I         0         abs         T         A         P		BOR #xx:3,@aa:8	в	7	ш	abs									
EDC #xx3.06 ara32         B         C         A         3         0         A         4         3         0         M           EDC #xx3.06 ara3         B         7         0         M         M         A         <		BOR #xx:3,@aa:16	в	 9	A	<del>.</del>	0		abs *1						
BET #xx:3,@ERd         B         7         0         0;MMi         0         1		BOR #xx:3,@aa:32	в	9	A	 С	0			abs					
BET #xx3.0 eEtd         B         7         D         0 iMM         0         iMM           BET RAX3.08ar30         B         C         D	BSET	BSET #xx:3,Rd	æ	2		0 IMM	P								
BET #xx3.@aar6         B         7         6         0		BSET #xx:3,@ERd	в	7		0 erd	0								
BET #xx3,@aar16         B         A         1         B         0         1         C         0         1         0         1         0         1         0         1         0         1         0         1         0         1         0         1         0         1         0         1         0         0         1         0         0         1         0         0         1         0         0         1         0         0         1         0         0         1         0         0         1         0         0         1         0         0         1         0         0         1         0         0         1         0         0         1         0         0         1         0         0         1         0         1         0         1         0         1         1         0         1         1         0         1 <th1< th="">         1         1</th1<>		BSET #xx:3,@aa:8	m	2	ш	abs									
BET #xx3.@aar.32         B         C         A         3         8         A         A         3         8         A         A         0         (i)         A         A         0         (i)         A		BSET #xx:3,@aa:16	m	9	A		8		abs						
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$		BSET #xx:3,@aa:32	ш	9	A	 ო	80		. U	abs					
$ \begin{array}{l c c c c c c c c c c c c c c c c c c c$		BSET Rn,Rd	m	9	0		p								
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$		BSET Rn,@ERd	ш	~		0 erd	0								
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$		BSET Rn,@aa:8	m	7	ш	abs									
BET Ru, @aa.32         B         6         3         8         A         abs         abs <td></td> <td>BSET Rn,@aa:16</td> <td>ш</td> <td>9</td> <td>A</td> <td> <del></del></td> <td>@</td> <td></td> <td>abs</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td>		BSET Rn,@aa:16	ш	9	A	 <del></del>	@		abs						
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$		BSET Rn,@aa:32	m	9	A	с С	80			abs					
	BSR	BSR d:8	I	2	5	disp									
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$		BSR d:16	I	 ۲	ပ	0	0	5	disp						
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$	BST	BST #xx:3,Rd	ш	 9		0 IMM	P								
		BST #xx:3,@ERd	в	2		0 erd	0								
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$		BST #xx:3,@aa:8	в	7	ш	abs									
BTT #xx3.@aa:32         B         6         A         3         8         A         3         8         A         3         9         A         0.100         C         1         0.100         C         1         0.100         C         1         0.100         C         1         0.100		BST #xx:3,@aa:16	в	9	A	1	8	-	abs						
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$		BST #xx:3,@aa:32	ю	9	٨	 С	8		U	abs					
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$	BTST	BTST #xx:3,Rd	в	7		0 IMM	P								
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$		BTST #xx:3,@ERd	в	7		0 erd	0								
		BTST #xx:3,@aa:8	в	7	ш	abs									
		BTST #xx:3,@aa:16	m	9	A		0		abs		l				
		BTST #xx:3,@aa:32	۵	9	A	с.	0		.0	abs			l		
		BTST Rn,Rd	в	9	e		p								
B         7         E         abs         6         3         m         0           B         6         A         1         0         abs         6         3         m         0         m           B         6         A         1         0         abs         6         3         m         0         7         6         3         m         1         0         1 </td <td></td> <td>BTST Rn,@ERd</td> <td>в</td> <td>7</td> <td></td> <td>0 erd</td> <td>0</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td>		BTST Rn,@ERd	в	7		0 erd	0								
B         6         A         1         0         abs         6         3         m         0         m		BTST Rn,@aa:8	в	7	Е	abs									
B 6 A 3 0 abs 6 3 m		BTST Rn,@aa:16	ю	9	۷		0		abs						
		BTST Rn,@aa:32	m	9	4	 e	0			sde					

Instruction	Macmonic	0.10 0.10							Instruction Format	n Format				
		3126	1st byte	oyte	2nd byte	oyte	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	9th byte	10th byte
BXOR	BXOR #xx:3,Rd	в	7	5	0 IMM	p								
	BXOR #xx:3,@ERd	B	~	υ	0 erd	0	7 5	0 [IMM] 0						
	BXOR #xx:3,@aa:8	ш	7	ш	abs	s	7 5	0 [IMM] 0						
1	BXOR #xx:3,@aa:16	m	٥	×	-	0		abs	7 : 5	0 : IMM: 0				
_	BXOR #xx:3,@aa:32	ш	9	A	e	0			abs		7 : 5	0 :IMM: 0		
CLRMAC*1	CLRMAC	Ι	0	٢	A	0								
CMP	CMP.B #xx:8,Rd	В	A	Þ	MMI	Μ								
	CMP:B Rs,Rd	В	+	ပ	rs	rd								
	CMP.W #xx:16,Rd	×	7	6	5	p	-	IMM						
	CMP.W Rd,Rd	Ν	۲	D	rs	rd								
	CMP.L #xx:32, ERd	Г	7	A	2	0 erd		_	IMM					
	CMP.L ERS, ERd	_	-	ш	1 ers 0 erd	0 erd								
DAA	DAA Rd	В	0	ш	0	rd								
DAS	DAS Rd	ш	-	ш	0	p								
DEC	DEC.B Rd	В	-	A	0	p								
	DEC.W #1,Rd	≥	-	æ	2	p								
	DEC.W #2,Rd	×	-	в	۵	rd								
	DEC.L #1,ERd	L	-	в	7	0 erd								
	DEC.L #2,ERd	Г	+	в	ш	0: erd								
DIVXS	DIVXS.B Rs,Rd	в	0	-	۵	0	5	rs rd						
	DIVXS.W Rs,ERd	≥	0	-		0	3	rs 0 erd						
DIVXU	DIVXU.B Rs,Rd	в	<u>د</u>	-	s	p								
	DIVXU.W Rs,ERd	≥	ъ	ю	s	0: erd								
EEPMOV	EEPMOV.B	Ι	7	ш	2	υ	5	8						
	EEPMOV.W	I	4	ю	۵	4	2 	⊥  8						
EXTS	EXTS.W Rd	Ν	۲	7	۵	p								
	EXTS.L ERd	-	-	2	ш	0: erd								
EXTU	EXTU.W Rd	≥	-	7	ŝ	p								
	EXTU.L ERd	-	-	2	2	0: erd								
INC	INC.B Rd	æ	0	A	0	Þ								
	INC.W #1,Rd	×	0	æ	2	p								
	INC.W #2,Rd	Ν	0	в	۵	rd								
	INC.L #1,ERd	-	0	ю	7	0 erd								
	INC.L #2,ERd	_	0	в	ш	0 erd								

											Instruc	Instruction Format	rmat				
Instruction	Mnemonic	Size	1st	1st byte	2nd byte	byte	3rd byte	vte	4th byte	đ	5th byte	-	6th byte	7th byte	8th byte	9th byte	10th byte
IMD	MD @FDs		u	0	o or o					2							
			, I				-	]				+					
	JMF @aa.24	Ι	0	۲ I		Ī	ans			+		+					
	JMP @aa:8	Ι	2	m	abs	S											
JSR	JSR @ERn	Ι	5	٥	0 : ern :	0											
	JSR @aa:24	Ι	5	ш			abs										
	JSR @@aa:8	Ι	5	ш.	abs	s											
LDC	LDC #xx:8,CCR	ш	0	7	MMI	Σ											
	LDC #xx:8,EXR	æ	0	-	4	-	0	7	MMI								
	LDC Rs,CCR	æ	0	<i>с</i>	0	S											
	LDC RS,EXR	ш	0	e	-	s											
	LDC @ERs,CCR	≥	0	-	4	0	9	6	0 ers	0							
	LDC @ERS,EXR	≥	0	-	4	-	9	6	0: ers	0							
	LDC @(d:16,ERs),CCR	≥	0	-	4	0	9	ш	0 ers	0		disp					
	LDC @(d:16,ERs),EXR	≥	0	-	4	-	9	ш	0 ers	0		disp					
	LDC @(d:32,ERs),CCR	≥	0	-	4	0	~	œ	0 ers	0	B  9	8	0		đ	disp	
	LDC @(d:32,ERs),EXR	≥	0	-	4	-	7	œ	0 ers	0	В  9	3	0		Ð	disp	
	LDC @ERs+,CCR	≥	0	-	4	0	9	۵	0 ers	0							
	LDC @ERs+,EXR	≥	0	-	4	-	9	۵	0 ers	0							
	LDC @aa: 16,CCR	≥	0	-	4	0	9	m	0	0		abs					
	LDC @aa:16,EXR	٨	0	-	4	٢	9	в	0	0		abs					
	LDC @aa:32,CCR	≥	0	-	4	0	9	ш	5	0			abs	Ş			
	LDC @aa:32,EXR	×	0	-	4	-	9	æ	2	0			abs	Ş			
LDM	LDM.L @SP+,(ERn-ERn+1)	_	0	-	-	0	9	۵	7 0	0 ern+1							
	LDM.L @SP+,(ERn-ERn+2)	-	0	-	5	0	9	٥	7 :0	0 ern+2							
	LDM.L @SP+,(ERn-ERn+3)	L	0	-	3	0	9	D	2 0	0 em+3							
LDMAC*1	LDMAC ERS, MACH	L	0	3	2	0 ers											
	LDMAC ERS, MACL	Г	0	е 	e	0 ers											
MAC*1	MAC @ERn+,@ERm+	I	0	-	9	0	9	۵	0 ern 0 erm	erm							
NOV	MOV.B #xx:8,Rd	в	ш	p	IMM	Σ											
	MOV.B Rs,Rd	В	0	с 	rs	p											
	MOV.B @ERs,Rd	в	9	8	0 ers	p											
	MOV.B @(d:16,ERs),Rd	ш	9	ш	0 ers	p		disp	ġ.								
	MOV.B @(d:32,ERs),Rd	В	7	8	0 ers	0	9	A	2	p			disp	đ			
	MOV.B @ERs+,Rd	ш	9	ပ	0 ers	p											
	MOV.B @aa:8,Rd	В	2	Id	abs	s											
	MOV.B @aa:16,Rd	ш	9	ح	0	Ð		at	abs								

Ath         Ath byte         Bth byte											Instructic	Instruction Format				
INOUG Result         B         6         A         2         1d	Instruction	Mnemonic	Size	1st	byte	2nd	byte	3rd by	rte	4th byte	5th byte	6th byte	7th byte	8th byte	9th byte	10th byte
	MOV	MOV.B @aa:32,Rd	8	9	۷	2				a	sd					
RM         B         C         S         C		MOV.B Rs,@ERd	ш	9	∞ 	1 erd										
Rð)         B         7         8         1		MOV.B Rs,@(d:16,ERd)	m	9	ш	1 erd			disp							
Image: line intermed inte		MOV.B Rs,@(d:32,ERd)	в	7	8	0: erd		9	A			dis	ds			
I         I		MOV.B Rs, @-ERd	в	9	с 	1 erd										
IB         6         A         8         13		MOV.B Rs,@aa:8	в	ю	s.	а	sq									
Index         Index <th< td=""><td></td><td>MOV.B Rs,@aa:16</td><td>ю</td><td>9</td><td>4</td><td>80</td><td>s</td><td></td><td>ab</td><td>6</td><td></td><td></td><td></td><td></td><td></td><td></td></th<>		MOV.B Rs,@aa:16	ю	9	4	80	s		ab	6						
W         T         9         0         1d         Imm         1mm		MOV.B Rs,@aa:32	۵	9	۷	۷	s			a	bs					
W         C		MOV.W #xx:16,Rd	≥	2	<b>б</b>	0	2		Ň	5						
NH         6         9         0         64         1		MOV.W Rs,Rd	3	0		2	Ð									
Jkb         W         6         F         0 (e)         rd         display         M         T         8         0 (e)         0         e         F         0 (e)         0         e         e         1         e         e         1         e         e         1         e         e         1         e         i         e         i		MOV.W @ERs,Rd	≥	9	б 	0 ers										
NRd         W         T         B         0         eres         1         disp         disp           W         6         D         0         eres         rd         rd         rds         rds<		MOV.W @(d:16,ERs),Rd	≥	9	<u>ш</u>	0 ers			dist							
		MOV.W @(d:32,ERs),Rd	≥	2	∞	0: ers		9	в			dis	ď			
		MOV.W @ERs+,Rd	≥	9	<u>_</u>	0: ers										
		MOV.W @aa:16,Rd	≥	9	8	0	Þ		ab	6						
		MOV.W @aa:32,Rd	3	9	<b>m</b>	2	Þ			3	sq					
		MOV.W Rs,@ERd	≥	9	6 	1 erd										
$ \left[ \begin{array}{cccccccccccccccccccccccccccccccccccc$		MOV.W Rs,@(d:16,ERd)	≥	9	ш. 	1 erd			disp							
		MOV.W Rs,@(d:32,ERd)	>	2	∞	0: erd		9	в			dis	d			
		MOV.W Rs,@-ERd	8	9		1 erd										
		MOV.W Rs,@aa:16	>	9	<u>6</u>	8	s		ab.	0						
		MOV.W Rs,@aa:32	8	9	в 	۷	LS.			a	bs					
		MOV.L #xx:32,Rd	_	2	4	0	0 erd			≧	1M					
		MOV.L ERS, ERd	_	0	ш. 	1 ers	0 erd									
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$		MOV.L @ERS,ERd	Г	0	-	0	0	9		0 ers 0 erd						
$ \left[ \begin{array}{cccccccccccccccccccccccccccccccccccc$		MOV.L @(d:16,ERs),ERd	-	0	-	0	0	9		0: ers : 0: erd		lisp				
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$		MOV.L @(d:32,ERs),ERd	_	0	-	0	0	7				0		q	isp	
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$		MOV.L @ERs+,ERd	-	0	-	0	0	9		0 ers 0 erd						
L         0         1         0         0         6         B         2         0         etcl         abs           L         0         1         0         0         6         9         1         etcl         abs           ERd)         L         0         1         0         6         7         1         etcl         abs           ERd)*2         L         0         1         0         0         6         7         1         etcl         abs           L         0         1         0         0         6         7         8         0         etcl         abs           L         0         1         0         0         6         7         8         0         etcl         abs           L         0         1         0         0         6         7         8         0         etcl         abs           L         0         1         0         0         6         7         3         abs         ats           L         0         1         0         0         6         7         3         abs         abs       <		MOV.L @aa:16,ERd	-	0	-	0	0	9	В			abs				
		MOV.L @aa:32,ERd		0		0	0		в			ab	S			
ERd)         L         0         1         0         6         F         1         erd 0         ers         disp         disp <t< td=""><td></td><td>MOV.L ERs, @ERd</td><td>_</td><td>0</td><td>-</td><td>0</td><td>0</td><td>9</td><td></td><td>1 erd 0 ers</td><td></td><td></td><td></td><td></td><td></td><td></td></t<>		MOV.L ERs, @ERd	_	0	-	0	0	9		1 erd 0 ers						
ERd)*2         L         0         1         0         0         7         8         0         erd         0         6         B         A         00         ers           L         0         1         0         0         6         D         1         erd         ers         ers           L         0         1         0         0         6         D         1         erd         ers         er		MOV.L ERs, @ (d:16, ERd)	-	0	-	0	0	 9		1: erd: 0: ers	q	lisp				
L         0         1         0         6         D         1! erd         0: ers         1           L         0         1         0         0         6         D         1! erd         0: ers         abs           L         0         1         0         0         6         B         8         0: ers         abs           L         0         1         0         0         6         B         A         0: ers         abs		MOV.L ERs, @ (d:32, ERd)*2		0	-	0	0	7				0		d	isp	
L         0         1         0         0         6         B         8         0; ers         abs           L         0         1         0         0         6         B         A         0; ers         abs		MOV.L ERs, @-ERd	-	0	-	0	0	9		1 erd 0 ers						
L 0 i 1 0 i 0 6 B A 0 ers		MOV.L ERs,@aa:16	_	0	-	0	0	9	в			abs				
		MOV.L ERs,@aa:32	_	0		0	0	9	в			ab	S			

										Instruction Format	n Format				
Instruction	Mnemonic	Size	1st byte	yte	2nd byte	byte	3rd byte		4th byte	5th byte	6th byte	7th byte	8th byte	9th byte	10th byte
MOVFPE	MOVFPE @aa:16,Rd	B	9	٨	4	Þ		abs							
MOVTPE	MOVTPE Rs,@aa:16	в		۲	с	ß		abs							
MULXS	MULXS.B Rs,Rd	ш	0	-	υ	0	2 2	0	rs rd						
	MULXS.W Rs,ERd	3	0	-	с	0	 2	2	rs :0: erd						
MULXU	MULXU.B Rs,Rd	۵	 2	0	2	p									
	MULXU.W Rs, ERd	3	2	2	2	0: erd									
NEG	NEG.B Rd	В	+	7	8	rd									
	NEG.W Rd	N	-	7	6	P									
	NEG.L ERd	_	-	7	<u>_</u>	0: erd									
NOP	NOP	Ι	0	0	0	0									
NOT	NOT.B Rd	œ	·····	7	0	Ð									
	NOT.W Rd	N	-	7	-	P									
	NOT.L ERd	_	·····	7	e	0 erd									
OR	OR.B #xx:8,Rd	œ	 U	Þ	≧	MMI									
	OR.B Rs, Rd	ш	-	4	S	Þ									
	OR.W #xx:16,Rd	≥	~	6	4	Þ		MMI							
	OR.W Rs,Rd	3	9	4	2	P									
	OR.L #xx:32,ERd	_	~	A	4	0 erd			2	IMM					
	OR.L ERS, ERd	L	0	-	LL.	0	9	4 0	0 ers 0 erd						
ORC	ORC #xx:8,CCR	m	0	4	≧	MMI									
	ORC #xx:8,EXR	ш	0	-	4	-	0	4	MMI						
РОР	POP.W Rn	≥	9	٥	~	٤									
	POP.L ERn	_	0	-	0	0	9	0	7 :0: ern						
PUSH	PUSH.W Rn	3	 9	۵	ш.	E									
	PUSH.L ERn	Γ	0	-	0	0	9	D	F :0: ern						
ROTL	ROTL.B Rd	в	-	2	80	p									
	ROTL.B #2,Rd	В	+	2	с	p									
	ROTL.W Rd	3	-	2	6	Þ									
	ROTL.W #2,Rd	3	-	2	۵	Þ									
	ROTL.L ERd	_	<del></del>	2	<u>6</u>	0 erd									
	ROTL.L #2,ERd	_		2	LL.	0 erd									
ROTR	ROTR.B Rd	в		3	80	Þ									
	ROTR.B #2,Rd	В	+	3	с	rd									
	ROTR.W Rd	3	-	3	6	p									
	ROTR.W #2,Rd	8	-	3	۵	p									

#### Section 2 Instruction Descriptions

Image: design of the stat									Instruction Format	n Format				
INDELLERG         L         1         7         0	Instruction	Mnemonic	Size	1st k	oyte	2nd byte	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	9th byte	10th byte
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$	ROTR	ROTR.L ERd	_	-	e e	B 0 erd						•	•	
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$		ROTR.L #2,ERd	-	-	e									
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$	ROTXL	ROTXL.B Rd	в	-	2									
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$		ROTXL.B #2,Rd	æ	-	2									
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$		ROTXL.W Rd	Ν	-	2									
$\begin{tabular}{ c c c c c c c c c c c c c c c c c c c$		ROTXL.W #2,Rd	3	-	2									
ROTXLL #2.ERd         L         1         2         7           ROTXLL #2.ERd         B         1         3         0           ROTXR.BAC         B         1         3         4           ROTXR.BAC.A         B         1         3         4           ROTXR.WEd         W         1         3         5           ROTXR.LERd         W         1         3         5           ROTXR.WELERd         L         1         3         7           RTE         -         5         6         7           RTS         -         5         6         7           RTL         -         5         6         7           RTS         -         5         6         7           RTL         -         5         6         7           RTL         -         5         6         7           SHALB#Z.Rd         B         1         0         7           SHALL#Z.ERd         L         1         1         1         1           SHALL#Z.ERd         W         1         1         1         1         1           SHALL#Z.ERd         L		ROTXL.L ERd	-	-	2									
ROTXR.B Rd         B         1         3         0           ROTXR.B #2, Rd         B         1         3         4           ROTXR.W Rd         W         1         3         5           ROTXR.W #2, Rd         W         1         3         5           ROTXR.W #2, Rd         W         1         3         5           ROTXR.L ERd         L         1         3         7           RTS          5         6         7           RTS          5         6         7           SHAL.B #2, Rd         B         1         0         9           SHAL.B #2, Rd         W         1         0         9           SHAL.B #2, Rd         W         1         0         9           SHAL.W #2, Rd         W         1         0         9           SHAL.W #2, Rd         W         1         1         1         1           SHAL.W #2, Rd         W         1         1         1         1           SHAL.W #2, Rd         W         1         1         1         1         1           SHAL.W #2, Rd         W         1         1		ROTXL.L #2,ERd	-	-	5									
ROTXR.B #2.Rd         B         1         3         4           ROTXR.W Rd         W         1         3         5           ROTXR.W #2.Rd         W         1         3         5           ROTXR.W #2.Rd         W         1         3         7           ROTXR.LERd         L         1         3         7           RTE          5         4         7           RTS          5         4         7           RTS          5         4         7           SHAL.B #2.Rd         B         1         0         8           SHAL.WRC         W         1         1         0         7           SHAL.WR2         W         1         1         0         7           SHAL.WR2.Rd         W         1         1         1         1              SHAL.WR2.Rd         W         1         1         1         1           SHAL.B #2.Rd         W         1         1         1         1         1         1         1         1         1         1         1         1         1         1         1         1         1 <td>ROTXR</td> <td>ROTXR.B Rd</td> <td>ш</td> <td>-</td> <td>з</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td>	ROTXR	ROTXR.B Rd	ш	-	з									
$\begin{tabular}{ c c c c c c c c c c c c c c c c c c c$		ROTXR.B #2,Rd	۵	-	e									
ROTXR.W #2,Rd         W         1         3         5           RTXR.L ERd         L         1         3         3         3           RTE         L         1         3         7         7           RTE         L         1         3         7         7           RTS         CUTXR.L #2.ERd         L         1         3         7           SHAL.Brd         B         1         0         6         7           SHAL.Wrd         W         1         0         8         7           SHAL.Wrd         W         1         0         8         9           SHAL.Wrd         W         1         0         8         9           SHAL.Wrd         W         1         1         0         9           SHAL.Wrd         W         1         1         1         8           SHAL.Wrd         W         1         1         1         1         1           SHAL.Wrd         W         1         1         1         1         1         1         1         1         1         1         1         1         1         1         1         1		ROTXR.W Rd	≥	-	e									
$\begin{tabular}{ c c c c c c c c c c c c c c c c c c c$		ROTXR.W #2,Rd	≥	-	e									
ROTXR.L#2.ERd         L         1         3         7           RTE          5         6         7           RTS          5         6         7           SHAL.BRd         B         1         0         8           SHAL.B#2.Rd         B         1         0         8           SHAL.B#2.Rd         W         1         0         9           SHAL.B#2.Rd         W         1         0         8           SHAL.B#2.Rd         L         1         0         8           SHAL.B#2.Rd         B         1         1         9           SHAR.B#2.Rd         B         1         1         9           SHAR.B#2.Rd         B         1         1         1         9           SHAR.B#2.Rd         L         1         1         1         9           SHAR.B#2.Rd         L         1         1         1         1         1           SHAR.B#2.Rd         L         L         1         1         1         1         1         1         1         1         1         1         1         1         1         1         1         <		ROTXR.L ERd	-	-	e									
RTE          5         6         7           RTS          5         4         7           SHAL.BrAC         B         1         0         8           SHAL.BrZ.Rd         B         1         0         9           SHAL.BrZ.Rd         B         1         0         9           SHAL.BrZ.Rd         L         1         1         0         9           SHAL.BrZ.Rd         L         1         1         0         8           SHAL.BrZ.Rd         L         1         1         0         8           SHAR.BrZ.Rd         B         1         1         1         9           SHAR.BrZ.Rd         B         1         1         1         1         1           SHAR.BrZ.Rd         B         1		ROTXR.L #2,ERd	-	-	e									
RTS          5         4         7           SHAL.B.Rd         B         1         0         8           SHAL.B.#Z.Rd         B         1         0         8           SHAL.W.Rd         W         1         0         9           SHAL.WEd         W         1         0         9           SHAL.WAC         W         1         0         9           SHAL.WAC         W         1         0         8           SHAL.WAC         N         1         1         8           SHAL.WAC         B         1         1         8           SHAR.MEd         N         1         1         9           SHAR.WEd         W         1         1         1         9           SHAR.WEd         W         1         1         1         1         1           SHAR.WEd         L         1         1         1         1         1         1           SHAR.WEd         L         L         1         1         1         1         1         1         1           SHAR.WEd         L         L         1         1         1<	RTE	RTE	Ι	ŝ	9									
SHALB Rd         B         1         0         8           SHALB #Z, Rd         B         1         0         C           SHALB #Z, Rd         W         1         0         C           SHALB #Z, Rd         W         1         0         D           SHAL.WZ, Rd         W         1         0         D           SHAL.WZ, Rd         W         1         0         D           SHAL.WZ, Rd         B         1         1         0         P           SHAR. Rd         B         1         1         0         P           SHAR. M &Z, Rd         B         1         1         1         P           SHAR. W #Z, Rd         W         1         1         1         P           SHAR. W #Z, Rd         W         1         1         1         P           SHAR. W #Z, Rd         L         1         1         1         P         P           SHAR. W #Z, Rd         L         1         1         1         P         P           SHAR. M #Z, Rd         L         L         1         1         1         P         P           SHAR. M #Z, Rd         L	RTS	RTS	Ι	ŝ	4									
SHALLB #2,Rd         B         1         0         C           SHALLB #2,Rd         W         1         0         9           SHAL.W#2,Rd         W         1         0         9           SHALLERd         L         1         0         8           SHALLERd         L         1         0         8           SHALLERd         L         1         0         7           SHALLERd         B         1         1         8           SHAR.WEd         W         1         1         8           SHAR.WED         W         1         1         8           SHAR.WED         W         1         1         1         8           SHAR.WED         W         1         1         1         1         1           SHAR.WED         W         1	SHAL	SHAL.B Rd	æ	-	0									
SHAL.WRd         W         1         0         9           SHAL.WAZ.Rd         W         1         0         D           SHAL.LERd         L         1         0         B           SHAL.LERd         L         1         0         B           SHAL.LAZ.ERd         L         1         0         F           SHAR.BR         B         1         1         0         F           SHAR.BR         B         1         1         1         B           SHAR.WRJ         W         1         1         1         B           SHAR.BRJ         B         1         1         1         B           SHLL.BAZRD         B         1         1         0         3           SHLL.BAZR		SHAL.B #2,Rd	В	٢	0									
SHALLW #2.Rd       W       1       0       D         SHALL #2.Rd       L       1       0       B         SHALL #2.ERd       L       1       0       F         SHALL #2.ERd       L       1       1       0       F         SHAR.BAG       B       1       1       1       8         SHAR.BAG       W       1       1       1       2         SHAR.BAG       W       1       1       1       2         SHAR.BAG       W       1       1       1       2         SHAR.LERd       L       1       1       1       2         SHAR.LBAG       L       L       1       1       7       2         SHLL.BAZ.Rd       B       1       0       1       1       7       3         SHLL.BAZ.Rd       B       1       1       0       3 <t< td=""><td></td><td>SHAL.W Rd</td><td>3</td><td>-</td><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></t<>		SHAL.W Rd	3	-	0									
SHALLERd         L         1         0         B           SHALL#Z.ERd         L         1         0         F           SHALL#Z.ERd         L         1         0         F           SHALL#Z.ERd         B         1         1         8           SHAR.B#Z.Rd         B         1         1         8           SHAR.B#Z.Rd         B         1         1         9           SHAR.W#Z.Rd         W         1         1         9           SHAR.W#Z.Rd         W         1         1         1         1           SHAR.W#Z.Rd         B         1         1         1         1         1           SHAR.W#Z.Rd         W         1         1         0         1		SHAL.W #2,Rd	Ν	+	0									
SHALL#2,ERd         L         1         0         F           SHAL.B2,ERd         B         1         1         8         8           SHAR.B.Kd         B         1         1         1         8           SHAR.B.Z.Rd         B         1         1         1         9           SHAR.B.Z.Rd         W         1         1         1         9           SHAR.W.Fd         W         1         1         1         9           SHAR.W.#2.Rd         W         1         1         1         1         1           SHAR.L.B.Z.Rd         L         1		SHAL.L ERd	-	-	0									
SHAR.B Rd         B         1         1         8           SHAR.B #2.Rd         B         1         1         1         C           SHAR.B #2.Rd         B         1         1         1         C           SHAR.B #2.Rd         W         1         1         1         9           SHAR.W #2.Rd         W         1         1         1         9           SHAR.W #2.Rd         L         1         1         1         1         1           SHAR.L #2.ERd         L         1		SHAL.L #2,ERd	-	-	0									
SHAR.B.#.2,Rd         B         1         1         C           SHAR.B.#2,Rd         W         1         1         9           SHAR.WFd         W         1         1         1         9           SHAR.W#2,Rd         W         1         1         1         9           SHAR.W#2,Rd         W         1         1         1         B           SHAR.LERd         L         1         1         1         B           SHLL.BRd         B         1         0         1         1         F           SHLL.BRd         W         1         0         1         0         3           SHLL.BR2.Rd         W         1         0         7         3           SHLL.WERD         L         1         1         0         7           SHLL.B#2.Rd         L         1         1         0         7           SHLR.B#2.Rd         B         1         1         1         4	SHAR	SHAR.B Rd	æ	-	-									
SHAR.W Rd         W         1         1         9           SHAR.W #2.Rd         W         1         1         1         D           SHAR.W #2.Rd         W         1         1         1         D           SHAR.W #2.Rd         L         1         1         1         B           SHAR.L #2.ERd         L         1         1         1         F           SHLLB #2.Rd         B         1         0         1         1         F           SHLL.B.#2.Rd         W         1         0         7         1         1         1         1         1         1         1           SHLL.B.#2.Rd         L         L         1         1         0         1		SHAR.B #2,Rd	ш	-	-									
SHAR.W #2.Rd     W     1     1     D       SHAR.L #Z.Rd     L     1     1     B       SHAR.L #Z.ERd     L     1     1     F       SHAR.L #Z.ERd     L     1     1     F       SHLLB #Z.Rd     B     1     0     4       SHLLL #Z.ERd     W     1     0     1       SHLL WRd     W     1     0     7       SHLL #Z.Rd     L     1     1     7       SHLL #Z.ERd     L     1     1     0       SHLL #Z.ERd     L     1     1     0       SHLL #Z.ERd     B     1     1     0       SHLR.B #Z.Rd     B     1     1     0		SHAR.W Rd	≥	-	-									
SHAR.L.ERd         L         1         1         B           SHAR.L.#2.ERd         L         1         1         7         8           SHAR.L.#2.ERd         L         1         1         1         7         9           SHAL.B.#2.Rd         B         1         0         0         0         0         1           SHLL.B.#2.Rd         W         1         0         1         0         1         1           SHLL.WAG         W         1         0         1         0         3         3           SHLL.WAG         L         1         1         0         3         <		SHAR.W #2,Rd	N	-	1									
SHARL#Z_ERd         L         1         1         F           SHAL.BC         B         1         0         0         0           SHL.BR2.Rd         B         1         0         0         4           SHL.LB#Z.Rd         B         1         0         1         7           SHLL.WEd         W         1         0         1         7           SHLL.WEd         W         1         0         7         5           SHLL.WEAC         L         1         1         0         7           SHLL.B#Z.Rd         L         1         1         0         7           SHLR.B#Z.Rd         B         1         1         0         7		SHAR.L ERd	L	-	1									
SHLLB#         B         1         0         0           SHLLB#2,Rd         B         1         0         4           SHLLB#2,Rd         W         1         0         4           SHLLW#         W         1         0         1           SHLLW#         W         1         0         1           SHLLW#         L         W         1         0         1           SHLLERd         L         1         1         0         3           SHLLERd         L         1         1         0         7           SHLLERd         L         1         1         0         7           SHLR.B.#Z,Rd         B         1         1         4		SHAR.L #2,ERd	_	-	-									
SHLLB#2,Rd         B         1         0         4           SHLLWRd         W         1         0         1           SHLLW#2,Rd         W         1         0         5           SHLLERd         L         1         0         5           SHLLERd         L         1         0         7           SHLLERd         L         1         1         0         7           SHLLERd         L         1         1         0         7           SHLL.B#2,ERd         B         1         1         0         7           SHLR.B.#2,Rd         B         1         1         4	SHLL	SHLL.B Rd	в	-	0									
SHLL.W Rd         W         1         0         1           SHLL.W #2.Rd         W         1         0         5           SHLL.ERd         L         1         0         5           SHLL.ERd         L         1         0         7           SHLL.B.#2.ERd         L         1         0         7           SHLL.B.#2.Rd         B         1         1         0         7           SHLR.B.#2.Rd         B         1         1         4		SHLL.B #2,Rd	В	+	0									
SHLLW #2,Rd         W         1         0         5           SHLLLERd         L         1         0         3           SHLLL#2,ERd         L         1         0         7           SHLL.#2,ERd         L         1         1         0         7           SHLL.#2,ERd         B         1         1         0         7           SHLR.B.#2,Rd         B         1         1         1         4		SHLL.W Rd	Ν	+	0									
SHLLLERd         L         1         0         3           SHLLL#2ERd         L         1         0         7           SHLLB#ZERd         B         1         1         0         7           SHLR.B.#ZRd         B         1         1         0         7		SHLL.W #2,Rd	Ν	-	0									
SHLLL #2,ERd         L         1         0         7           SHLR.B.Rd         B         1         1         0         7           SHLR.B.#2,Rd         B         1         1         0		SHLL.L ERd	-	-	0									
SHLR.B.Rd         B         1         1         0           SHLR.B.#2.Rd         B         1         1         4		SHLL.L #2,ERd	-	-	0									
B 1 1 4	SHLR	SHLR.B Rd	ш	-	-									
		SHLR.B #2,Rd	в	-	۱									

		i								Instructio	Instruction Format				
			1st	1st byte	2nd	2nd byte	3rd byte	ŧ	4th byte	5th byte	6th byte	7th byte	8th byte	9th byte	10th byte
SHLR	SHLR.W Rd	N	۲	-	-	q									
	SHLR.W #2,Rd	N	٦	-	5	rd									
	SHLR.L ERd	_	-	-	е	0 erd									
	SHLR.L #2,ERd	_	-	-	7	0 erd									
SLEEP	SLEEP	Ι	0	-	8	0									
STC	STC.B CCR,Rd	m	0	2	0	p									
	STC.B EXR,Rd	в	0	2	-	p									
	STC.W CCR,@ERd	Ν	0	-	4	0	9	9 1	1 erd 0						
	STC.W EXR,@ERd	8	0	-	4	-	 9	9	1 erd 0						
	STC.W CCR,@(d:16,ERd)	≥	0	-	4	0	9	н Г	1 erd 0	q	disp				
	STC.W EXR,@(d:16,ERd)	≥	0	<del>.</del>	4	-	9	н	1 erd 0	q	disp				
	STC.W CCR,@(d:32,ERd)	Ν	0	-	4	0	7	8 C	0 erd 0	6 B	A 0		disp	ds	
	STC.W EXR,@(d:32,ERd)	≥	0	-	4	-	7	8	0 erd 0	в 9	0 V		disp	ď	
	STC.W CCR, @-ERd	Ν	0	+	4	0	9	D 1	1 erd 0						
	STC.W EXR,@-ERd	≥	0	-	4	-	 9	0	1 erd 0						
	STC.W CCR,@aa:16	≥	0	-	4	0	9	ш	8	a	abs				
	STC.W EXR,@aa:16	≥	0	-	4	-	9	в	8	a	abs				
	STC.W CCR,@aa:32	N	0	-	4	0	 9	в	A : 0		abs	S			
	STC.W EXR,@aa:32	×	0	-	4	+	9	в	A 0		abs	S			
STM	STM.L (ERn-ERn+1), @-SP	_	0	-	-	0	 9	٥	F 0 ern						
	STM.L (ERn-ERn+2), @-SP	_	0	-	2	0	9	D	F :0; ern						
	STM.L (ERn-ERn+3), @-SP	_	0	-	e	0	9	۵	F 0 ern						
STMAC*1	STMAC MACH, ERd	_	0	2	2	0 ers									
	STMAC MACL, ERd	L	0	2	3	0 ers									
SUB	SUB.B Rs,Rd	ш	-	∞	rs	p		_							
	SUB.W #xx:16,Rd	N	7	6	3	p		IMM							
	SUB.W Rs,Rd	8	-	6	rs	p									
	SUB.L #xx:32,ERd	L	7	A	3	0 erd			II	IMM					
	SUB.L ERS, ERd	_	-	۲	1 ers	1 ers 0 erd									
SUBS	SUBS #1,ERd	_	-	æ	0	0 erd									
	SUBS #2,ERd	_	-	B	8	0 erd									
	SUBS #4,ERd	L	٢	8	6	:0 erd									
SUBX	SUBX #xx:8,Rd	в	в	p	≥	IMM									
	SUBX Rs,Rd	m	-	ш	s	Þ									
TAS	TAS @ERd*3	ш	0	-	ш	0	~	<u>в</u>	0 ; erd ; C						
TRAPA	TRAPA #x:2	Ι	5	~	00: IMM	0									

#### Section 2 Instruction Descriptions
XOR	_	A710					4th hvte	5th byte	6th byte	7th byte	8th byte	9th byte	
XOR			-	1 st byte	2nd byte	3rd byte				-			10th byte
	XOR.B #xx:8,Rd	۵		5	MMI								
	XOR.B Rs,Rd	۵	-	2 	rs rd								
	XOR.W #xx:16,Rd	≥	~	ര 	5 	É	MMI						
	XOR.W Rs,Rd	8	9	2	rs 								
	XOR.L #xx:32,ERd	_	2	۲	5 :0: erd		IMM	5					
	XOR.L ERS, ERd	-	0	-	0 	6	0 ers : 0 erd						
XORC	XORC #xx:8,CCR	۵	0	<u>ء</u>	MMI								
	XORC #xx:8,EXR	B	0	-	4	0 5	MMI						
Notes: 1	<ol> <li>These instructions are supported by the H8S/2600 CPU only.</li> <li>Bit 7 of the 4th byte of the MOV.L ERs, @(d:32,ERd) instruction can be either 1 or 0.</li> <li>Only register ER0, ER1, ER4, or ER5 should be used when using the TAS instruction.</li> </ol>	are su e of the ER1, I	∋ MO ER4,	ted by N.L.ER or ER5	the H8S/26( 8s, @(d:32,E 5 should be	00 CPU onl ERd) instruc used when	nstructions are supported by the H8S/2600 CPU only. the 4th byte of the MOV.L ERs, @(d:32,ERd) instruction can be either 1 or 0. gister ER0, ER1, ER4, or ER5 should be used when using the TAS instruction	ither 1 or 0. S instruction	. c				
Legend: IMM: abs:	Immediate data (2, 3, 8, 16, or 32 bits) Absolute address (8, 16, 24, or 32 bits)	e data ( iddress	5 (8, <sup>.</sup>	8, 16, 6 16, 24,	or 32 bits) or 32 bits)								
disp: rs, rd, rn:		ient (8, eld (4 ł	, 16, i bits s	or 32 b pecifyir	its) ng an 8-bit c	or 16-bit reg	Displacement (8, 16, or 32 bits) Register field (4 bits specifying an 8-bit or 16-bit register. The symbols rs, rd, and rn correspond to operand symbols Rs, Rd,	nbols rs, rd,	and rn corr	espond to (	operand syr	nbols Rs, R	čd,
ırs, erd,	ers, erd, ern, erm: Register field (3 bits specifying an ac symbols ERs, ERd, ERn, and ERm.)	eld (3 I ERs, EF	bits s ≷d, E	specifyii Rn, and	ng an addre: d ERm.)	ss register -	supports	ster. The syr	mbols ers, e	erd, ern, and	d erm corre:	spond to op	berand
The regi	The register fields specify general registers as follows.	neral r∈	egiste	ers as fo	ollows.								
32 Ac	Address Register 32-Bit Register			16-	16-Bit Register	Ļ	œ	8-Bit Register	er				
Register Field	· General Register	I	Regis Field	Register Field	General Register	eral ster	Register Field	General Register	eral ister				
000	ERO		0000	0	RO		0000	ROH					
001	ER1		0001	Ξ	F.		0001	R1H					
			• •					••					
111	ER7		011	<del>~</del>	R7		0111	R7H					
			100	0	EO		1000	ROL					
			1001	۲	Ξ·		1001	R1L					
			• •		•••								
			1111	-	E7		1111	R7L					

# 2.5 Operation Code Map

Table 2.3 shows an operation code map.

#### Table 2.3Operation Code Map (1)

ation Code: 1st byte 2nd byte 4H BL Instruct AH AL BH BL Instruct noP Table 23 (2) Table 23 (2) Table 23 (2) OR XOR AND Table 23 (2) Ta	B     SUBX       C     OR       D     OR       F     AND       Note: * These instructions are supported by the HSS/2600 CPU only.
--	---



2nd byte 1st byte Operation Code:

В ВН AL AH

AH AL BH	0	-	2	ю	4	5	9	7	8	6	A	в	ပ	٥	ш	Ŀ
01	MOV	КDМ		STM	LDC STC		MAC*		SLEEP		CLRMAC*		Table 2.3 (3) Table 2.3 (3)	Table 2.3 (3)	TAS	Table 2.3 (3)
0A	INC											AI	ADD			
0B	ADDS					INC		INC	ADDS	SC				INC		INC
ΟF	DAA											ž	MOV			
10	Sh	SHLL			SHLL			SHLL	SHAL	٦L			SHAL			SHAL
11	HS	SHLR			SHLR			SHLR	SHAR	٩R			SHAR			SHAR
12	.03	ROTXL			ROTXL			ROTXL	ROTL	1			ROTL			ROTL
13	RO <sup>-</sup>	ROTXR			ROTXR			ROTXR	ROTR	IR			ROTR			ROTR
17	Ň	NOT		NOT		EXTU		EXTU	NEG	IJ		NEG		EXTS		EXTS
1A	DEC											S	SUB			
1B	SUBS					DEC		DEC	SUBS	3S				DEC		DEC
1F	DAS											C	CMP			
58	BRA	BRN	BHI	BLS	BCC	BCS	BNE	BEQ	BVC	BVS	BPL	BMI	BGE	BLT	BGT	BLE
6A	MOV	Table 2.3 (4)	MOV	Table 2.3 (4)	Table 2.3 (4) MOVFPE				MOV		MOV		MOVTPE			
79	MOV	ADD	CMP	SUB	OR	XOR	AND									
ΤA	NOM	ADD	CMP	SUB	OR	XOR	AND									

Note: \* These instructions are supported by the H8S/2600 CPU only.

#### **Operation Code Map (2)** Table 2.3

#### Section 2 Instruction Descriptions

#### **Operation Code Map (3)** Table 2.3

Instruction when most significant bit of DH is 0. 4th byte 3rd byte 2nd byte 1st byte **Operation Code:** 

—Instruction when most significant bit of DH is 1.	ш												
fficant bit c	ш												
most signi	۵												
tion when	U												
	B												
•	A												
	თ												
	8												
	7					BLD BILD	BST BIST			BLD BILD	BST BIST		
Ы	9			AND		BAND R BIAND				BIAND			
Н	2ı			XOR		BIXOR				BIXOR			
с т	4			OR		BORBIOR				BORBIOR			
CH	ю		DIVXS		BTST	BTST			BTST	BTST			
BHBL	2	MULXS					BCLR	BCLR			BCLR	BCLR	
AL	-		DIVXS				BNOT	BNOT			BNOT	BNOT	
ΗA	0	MULXS					BSET	BSET			BSET	BSET	
	AHAL BHBLCH CL	01C05	01D05	01F06	7Cr06 *1	7Cr07*1	7Dr06 *1	7Dr07 *1	7Eaa6*2	7Eaa7*2	7Faa6* <sup>2</sup>	7Faa7*2	

Notes: 1. The letter "r" indicates a register field. 2. The letters "aa" indicate an absolute address field.

Rev. 4.00 Feb 24, 2006 page 276 of 322 REJ09B0139-0400





#### Table 2.3 **Operation Code Map (4)**

## 2.6 Number of States Required for Instruction Execution

The tables in this section can be used to calculate the number of states required for instruction execution by the CPU. Table 2.5 indicates the number of instruction fetch, data read/write, and other cycles occurring in each instruction. Table 2.4 indicates the number of states required for each cycle, depending on its size. The number of states required for each cycle depends on the product. See the hardware manual named for the relevant product for details. The number of states required for execution of an instruction can be calculated from these two tables as follows:

Execution states =  $I \times S_I + J \times S_J + K \times S_K + L \times S_L + M \times S_M + N \times S_N$ 

**Examples:** Advanced mode, program code and stack located in external memory, on-chip supporting modules accessed in two states with 8-bit bus width, external devices accessed in three states with one wait state and 16-bit bus width.

1. BSET #0, @FFFFC7:8

From table 2.5:

 $I=L=2, \quad J=K=M=N{=}0$ 

From table 2.4:

 $S_{I} = 4$ ,  $S_{I} = 2$ 

Number of states required for execution =  $2 \times 4 + 2 \times 2 = 12$ 

2. JSR @@30

From table 2.5:

 $I=J=K=2, \quad L=M=N=0$ 

From table 2.4:

$$S_{I} = S_{J} = S_{K} = 4$$

Number of states required for execution =  $2 \times 4 + 2 \times 4 + 2 \times 4 = 24$ 

## Renesas

					Access	Conditions		
			On-Chi	p Supporting		Extern	al Device	
			Module		8-E	lit Bus	16-E	Bit Bus
Cycle		On-Chip Memory	8-Bit Bus	16-Bit Bus	2-State Access	3-State Access	2-State Access	3-State Access
Instruction fetch	S	1	2n	n	4	6 + 2m	2	3 + m*
Branch address rea	d S <sub>J</sub>							
Stack operation	Sκ							
Byte data access	$S_{L}$		n		2	3 + m		
Word data access	S <sub>м</sub>		2n		4	6 + 2m		
Internal operation	S <sub>N</sub>	1	1	1	1	1	1	1

#### Table 2.4Number of States per Cycle

Note: \* For the MOVFPE and MOVTPE instructions, refer to the relevant microcontroller hardware manual.

Legend:

m: Number of wait states inserted into external device access

n: Number of states required for access to an on-chip supporting module. For the specific number, refer to the relevant microcontroller hardware manual.

#### Table 2.5 Number of Cycles in Instruction Execution

		Instruction Fetch	Branch Address Read	Stack Operation	Byte Data Access	Word Data Access	Internal Operation
Instruction	Mnemonic	I	J	К	L	М	Ν
ADD	ADD.B #xx:8,Rd	1					
	ADD.B Rs,Rd	1					
	ADD.W #xx:16,Rd	2					
	ADD.W Rs,Rd	1					
	ADD.L #xx:32,ERd	3					
	ADD.L ERs,ERd	1					
ADDS	ADDS #1/2/4,ERd	1					
ADDX	ADDX #xx:8,Rd	1					
	ADDX Rs,Rd	1					
AND	AND.B #xx:8,Rd	1					
	AND.B Rs,Rd	1					
	AND.W #xx:16,Rd	2					
	AND.L #xx:32,ERd	3					
	AND.L ERs,ERd	2					
ANDC	ANDC #xx:8,CCR	1					
	ANDC #xx:8,EXR	2					
BAND	BAND #xx:3,Rd	1					
	BAND #xx:3,@ERd	2			1		
	BAND #xx:3,@aa:8	2			1		
	BAND #xx:3,@aa:16	3			1		
	BAND #xx:3,@aa:32	4			1		
	BRA d:8 (BT d:8)	2					
	BRN d:8 (BF d:8)	2					
	BHI d:8	2					
	BLS d:8	2					
	BCC d:8 (BHS d:8)	2					
	BCS d:8 (BLO d:8)	2					
	BNE d:8	2					
	BEQ d:8	2					
	BVC d:8	2					
	BVS d:8	2					
	BPL d:8	2					
	BMI d:8	2					
	BGE d:8	2					
	BLT d:8	2					
	BGT d:8	2					

		Instruction Fetch	Branch Address Read	Stack Operation	Byte Data Access	Word Data Access	Internal Operation
Instruction	Mnemonic	I	J	К	L	М	Ν
Bcc	BLE d:8	2					
	BRA d:16 (BT d:16)	2					1
	BRN d:16 (BF d:16)	2					1
	BHI d:16	2					1
	BLS d:16	2					1
	BCC d:16 (BHS d:16)	2					1
	BCS d:16 (BLO d:16)	2					1
	BNE d:16	2					1
	BEQ d:16	2					1
	BVC d:16	2					1
	BVS d:16	2					1
	BPL d:16	2					1
	BMI d:16	2					1
	BGE d:16	2					1
	BLT d:16	2					1
	BGT d:16	2					1
	BLE d:16	2					1
BCLR	BCLR #xx:3,Rd	1					
	BCLR #xx:3,@ERd	2			2		
	BCLR #xx:3,@aa:8	2			2		
	BCLR #xx:3,@aa:16	3			2		
	BCLR #xx:3,@aa:32	4			2		
	BCLR Rn,Rd	1					
	BCLR Rn,@ERd	2			2		
	BCLR Rn,@aa:8	2			2		
	BCLR Rn,@aa:16	3			2		
	BCLR Rn,@aa:32	4			2		
BIAND	BIAND #xx:3,Rd	1					
	BIAND #xx:3,@ERd	2			1		
	BIAND #xx:3,@aa:8	2			1		
	BIAND #xx:3,@aa:16	3			1		
	BIAND #xx:3,@aa:32	4			1		
BILD	BILD #xx:3,Rd	1					
	BILD #xx:3,@ERd	2			1		
	BILD #xx:3,@aa:8	2			1		
	BILD #xx:3,@aa:16	3			1		
	BILD #xx:3,@aa:32	4			1		

#### Section 2 Instruction Descriptions

		Instruction Fetch	Branch Address Read	Stack Operation	Byte Data Access	Word Data Access	Internal Operation
Instruction	Mnemonic	I	J	К	L	Μ	Ν
BIOR	BIOR #xx:8,Rd	1					
	BIOR #xx:8,@ERd	2			1		
	BIOR #xx:8,@aa:8	2			1		
	BIOR #xx:8,@aa:16	3			1		
	BIOR #xx:8,@aa:32	4			1		
BIST	BIST #xx:3,Rd	1					
	BIST #xx:3,@ERd	2			2		
	BIST #xx:3,@aa:8	2			2		
	BIST #xx:3,@aa:16	3			2		
	BIST #xx:3,@aa:32	4			2		
BIXOR	BIXOR #xx:3,Rd	1					
	BIXOR #xx:3,@ERd	2			1		
	BIXOR #xx:3,@aa:8	2			1		
	BIXOR #xx:3,@aa:16	3			1		
	BIXOR #xx:3,@aa:32	4			1		
BLD	BLD #xx:3,Rd	1					
	BLD #xx:3,@ERd	2			1		
	BLD #xx:3,@aa:8	2			1		
	BLD #xx:3,@aa:16	3			1		
	BLD #xx:3,@aa:32	4			1		
BNOT	BNOT #xx:3,Rd	1					
	BNOT #xx:3,@ERd	2			2		
	BNOT #xx:3,@aa:8	2			2		
	BNOT #xx:3,@aa:16	3			2		
	BNOT #xx:3,@aa:32	4			2		
	BNOT Rn,Rd	1					
	BNOT Rn,@ERd	2			2		
	BNOT Rn,@aa:8	2			2		
	BNOT Rn,@aa:16	3			2		
	BNOT Rn,@aa:32	4			2		
BOR	BOR #xx:3,Rd	1					
	BOR #xx:3,@ERd	2			1		
	BOR #xx:3,@aa:8	2			1		
	BOR #xx:3,@aa:16	3			1		
	BOR #xx:3,@aa:32	4			1		
BSET	BSET #xx:3,Rd	1					
	BSET #xx:3,@ERd	2			2		
	BSET #xx:3,@aa:8	2			2		

			Instruction Fetch	Branch Address Read	Stack Operation	Byte Data Access	Word Data Access	Internal Operation
Instruction	Mnemonic		I	J	К	L	М	Ν
BSET	BSET #xx:3,@	2aa:16	3			2		
	BSET #xx:3,@	aa:32	4			2		
	BSET Rn,Rd		1					
	BSET Rn,@El	Rd	2			2		
	BSET Rn,@aa	a:8	2			2		
	BSET Rn,@aa	a:16	3			2		
	BSET Rn,@aa	a:32	4			2		
BSR	BSR d:8	Normal	2		1			
		Advanced	2		2			
	BSR d:16	Normal	2		1			1
		Advanced	2		2			1
BST	BST #xx:3,Rd		1					
	BST #xx:3,@E	ERd	2			2		
	BST #xx:3,@a	aa:8	2			2		
	BST #xx:3,@a	aa:16	3			2		
	BST #xx:3,@a	aa:32	4			2		
BTST	BTST #xx:3,R	d	1					
	BTST #xx:3,@	ERd	2			1		
	BTST #xx:3,@	aa:8	2			1		
	BTST #xx:3,@	aa:16	3			1		
	BTST #xx:3,@	aa:32	4			1		
	BTST Rn,Rd		1					
	BTST Rn,@El	Rd	2			1		
	BTST Rn,@aa	a:8	2			1		
	BTST Rn,@aa	a:16	3			1		
	BTST Rn,@aa:32		4			1		
BXOR	BXOR #xx:3,R	۲d	1					
	BXOR #xx:3,@	2 ERd	2			1		
	BXOR #xx:3,@	2aa:8	2			1		
	BXOR #xx:3,@	2aa:16	3			1		
	BXOR #xx:3,@	2aa:32	4			1		
CLRMAC*5	CLRMAC		1					1 <sup>*3 *6</sup>
CMP	CMP.B #xx:8,I	Rd	1					
	CMP.B Rs,Rd		1					
	CMP.W #xx:10	6,Rd	2					
	CMP.W Rs,Ro	ł	1					
	CMP.L #xx:32	,ERd	3					
	CMP.L ERs,E	Rd	1					

#### Section 2 Instruction Descriptions

			Instruction Fetch	Branch Address Read	Stack Operation	Byte Data Access	Word Data Access	Internal Operation
Instruction	Mnemonic		I	J	К	L	М	Ν
DAA	DAA Rd		1					
DAS	DAS Rd		1					
DEC	DEC.B Rd		1					
	DEC.W #1/2,Rd		1					
	DEC.L #1/2,ERd		1					
DIVXS	DIVXS.B Rs,Rd		2					11
	DIVXS.W Rs,ER	d	2					19
DIVXU	DIVXU.B Rs,Rd		1					11
	DIVXU.W Rs,ER	d	1					19
EEPMOV	EEPMOV.B		2			2n + 2*1		
	EEPMOV.W		2			2n + 2*1		
EXTS	EXTS.W Rd		1					
	EXTS.L ERd		1					
EXTU	EXTU.W Rd		1					
	EXTU.L ERd		1					
INC	INC.B Rd		1					
	INC.W #1/2,Rd		1					
	INC.L #1/2,ERd		1					
JMP	JMP @ERn		2					
	JMP @aa:24		2					1
	JMP @@aa:8	Normal	2	1				1
		Advanced	2	2				1
JSR	JSR @ERn	Normal	2		1			
		Advanced	2		2			
	JSR @aa:24	Normal	2		1			1
		Advanced	2		2			1
	JSR @@aa:8	Normal	2	1	1			
		Advanced	2	2	2			
LDC	LDC #xx:8,CCR		1					
	LDC #xx:8,EXR		2					
	LDC Rs,CCR		1					
	LDC Rs,EXR		1					
	LDC @ERs,CCR	1	2				1	
	LDC @ERs,EXR		2				1	
	LDC @(d:16,ERs	s),CCR	3				1	
	LDC @(d:16,ERs	s),EXR	3				1	
	LDC @(d:32,ERs	s),CCR	5				1	
	LDC @(d:32,ERs	s),EXR	5				1	

		Instruction Fetch	Branch Address Read	Stack Operation	Byte Data Access	Word Data Access	Internal Operation
Instruction	Mnemonic	I	J	К	L	М	N
LDC	LDC @ERs+,CCR	2				1	1
	LDC @ERs+,EXR	2				1	1
	LDC @aa:16,CCR	3				1	
	LDC @aa:16,EXR	3				1	
	LDC @aa:32,CCR	4				1	
	LDC @aa:32,EXR	4				1	
LDM	LDM.L @SP+,(ERn-ERn+1)	2		4			1
	LDM.L @SP+,(ERn-ERn+2)	2		6			1
	LDM.L @SP+,(ERn-ERn+3)	2		8			1
LDMAC*5	LDMAC ERs,MACH	1					1 <sup>*3*6</sup>
	LDMAC ERs,MACL	1					1 <sup>*3*6</sup>
MAC <sup>*5</sup>	MAC @ERn+,@ERm+	2				2	
MOV	MOV.B #xx:8,Rd	1					
	MOV.B Rs,Rd	1					
	MOV.B @ERs,Rd	1			1		
	MOV.B @(d:16,ERs),Rd	2			1		
	MOV.B @(d:32,ERs),Rd	4			1		
	MOV.B @ERs+,Rd	1			1		1
	MOV.B @aa:8,Rd	1			1		
	MOV.B @aa:16,Rd	2			1		
	MOV.B @aa:32,Rd	3			1		
	MOV.B Rs,@ERd	1			1		
	MOV.B Rs,@(d:16,ERd)	2			1		
	MOV.B Rs,@(d:32,ERd)	4			1		
	MOV.B Rs,@-ERd	1			1		1
	MOV.B Rs,@aa:8	1			1		
	MOV.B Rs,@aa:16	2			1		
	MOV.B Rs,@aa:32	3			1		
	MOV.W #xx:16,Rd	2					
	MOV.W Rs,Rd	1					
	MOV.W @ERs,Rd	1				1	
	MOV.W @(d:16,ERs),Rd	2				1	
	MOV.W @(d:32,ERs),Rd	4				1	
	MOV.W @ERs+,Rd	1				1	1
	MOV.W @aa:16,Rd	2				1	
	MOV.W @aa:32,Rd	3				1	
	MOV.W Rs,@ERd	1				1	

#### Section 2 Instruction Descriptions

			Instruction Fetch	Branch Address Read	Stack Operation	Byte Data Access	Word Data Access	Internal Operation
Instruction	Mnemonic		I	J	к	L	М	N
MOV	MOV.W Rs,@(d:16	,ERd)	2				1	
	MOV.W Rs,@(d:32	,ERd)	4				1	
	MOV.W Rs,@-ER	ł	1				1	1
	MOV.W Rs,@aa:16	6	2				1	
	MOV.W Rs,@aa:32	2	3				1	
	MOV.L #xx:32,ERd		3					
	MOV.L ERs,ERd		1					
	MOV.L @ERs,ERd		2				2	
	MOV.L @(d:16,ER	s),ERd	3				2	
	MOV.L @(d:32,ER	s),ERd	5				2	
	MOV.L @ERs+,ER	d	2				2	1
	MOV.L @aa:16,ER	d	3				2	
	MOV.L @aa:32,ER	d	4				2	
	MOV.L ERs,@ERd		2				2	
	MOV.L ERs,@(d:16	6,ERd)	3				2	
	MOV.L ERs,@(d:32	2,ERd)	5				2	
	MOV.L ERs,@-ER	d	2				2	1
	MOV.L ERs,@aa:1	6	3				2	
	MOV.L ERs,@aa:3	2	4				2	
MOVFPE	MOVFPE @:aa:16,	Rd	2			1 <sup>*2</sup>		
MOVTPE	MOVTPE Rs,@:aa	:16	2			1 <sup>*2</sup>		
MULXS	MULXS.B Rs,Rd	H8S/2600	2					2 <sup>*3 *6</sup>
		H8S/2000	2					11
	MULXS.W Rs,ERd	H8S/2600	2					3 <sup>*3 *6</sup>
		H8S/2000	2					19
MULXU	MULXU.B Rs,Rd	H8S/2600	1					2 <sup>*3 *6</sup>
		H8S/2000	1					11
	MULXU.W Rs,ERd	H8S/2600	1					3 <sup>*3 *6</sup>
		H8S/2000	1					19
NEG	NEG.B Rd		1					
	NEG.W Rd		1					
	NEG.L ERd		1					
NOP	NOP		1					
NOT	NOT.B Rd		1					
	NOT.W Rd		1					
	NOT.L ERd		1					

			Instruction Fetch	Branch Address Read	Stack Operation	Byte Data Access	Word Data Access	Internal Operation
Instruction	Mnemonic		I	J	К	L	М	Ν
OR	OR.B #xx:8,Rd		1					
	OR.B Rs,Rd		1					
	OR.W #xx:16,R	d	2					
	OR.W Rs,Rd		1					
	OR.L #xx:32,ER	d	3					
	OR.L ERs,ERd		2					
ORC	ORC #xx:8,CCF	2	1					
	ORC #xx:8,EXR	ł	2					
POP	POP.W Rn		1				1	1
	POP.L ERn		2				2	1
PUSH	PUSH.W Rn		1				1	1
	PUSH.L ERn		2				2	1
ROTL	ROTL.B Rd		1					
	ROTL.B #2,Rd		1					
	ROTL.W Rd		1					
	ROTL.W #2,Rd		1					
	ROTL.L ERd		1					
	ROTL.L #2,ERd		1					
ROTR	ROTR.B Rd		1					
	ROTR.B #2,Rd		1					
	ROTR.W Rd		1					
	ROTR.W #2,Rd		1					
	ROTR.L ERd		1					
	ROTR.L #2,ERG	ł	1					
ROTXL	ROTXL.B Rd		1					
	ROTXL.B #2,Rc	1	1					
	ROTXL.W Rd		1					
	ROTXL.W #2,R	d	1					
	ROTXL.L ERd		1					
	ROTXL.L #2,ER	d	1					
ROTXR	ROTXR.B Rd		1					
	ROTXR.B #2,Ro	d	1					
	ROTXR.W Rd		1					
	ROTXR.W #2,R	d	1					
	ROTXR.L ERd		1					
	ROTXR.L #2,EF	Rd	1					
RTE	RTE		2		2/3*1			1
RTS	RTS	Normal	2		1			1
		Advanced	2		2			1

#### Section 2 Instruction Descriptions

		Instruction Fetch	Branch Address Read	Stack Operation	Byte Data Access	Word Data Access	Internal Operation
Instruction	Mnemonic	I	J	К	L	М	Ν
SHAL	SHAL.B Rd	1					
	SHAL.B #2,Rd	1					
	SHAL.W Rd	1					
	SHAL.W #2,Rd	1					
	SHAL.L ERd	1					
	SHAL.L #2,ERd	1					
SHAR	SHAR.B Rd	1					
	SHAR.B #2,Rd	1					
	SHAR.W Rd	1					
	SHAR.W #2,Rd	1					
	SHAR.L ERd	1					
	SHAR.L #2,ERd	1					
SHLL	SHLL.B Rd	1					
	SHLL.B #2,Rd	1					
	SHLL.W Rd	1					
	SHLL.W #2,Rd	1					
	SHLL.L ERd	1					
	SHLL.L #2,ERd	1					
SHLR	SHLR.B Rd	1					
	SHLR.B #2,Rd	1					
	SHLR.W Rd	1					
	SHLR.W #2,Rd	1					
	SHLR.L ERd	1					
	SHLR.L #2,ERd	1					
SLEEP	SLEEP	1					1
STC	STC.B CCR,Rd	1					
	STC.B EXR,Rd	1					
	STC.W CCR,@ERd	2				1	
	STC.W EXR,@ERd	2				1	
	STC.W CCR,@(d:16,ERd)	3				1	
	STC.W EXR,@(d:16,ERd)	3				1	
	STC.W CCR,@(d:32,ERd)	5				1	
	STC.W EXR,@(d:32,ERd)	5				1	
	STC.W CCR,@-ERd	2				1	1
	STC.W EXR,@-ERd	2				1	1
	STC.W CCR,@aa:16	3				1	
	STC.W EXR,@aa:16	3				1	
	STC.W CCR,@aa:32	4				1	
	STC.W EXR,@aa:32	4				1	

			Instruction Fetch	Branch Address Read	Stack Operation	Byte Data Access	Word Data Access	Internal Operation
Instruction	Mnemonic		I	J	К	L	М	Ν
STM	STM.L (ERn-ER	n+1),@-SP	2		4			1
	STM.L(ERn-ER	n+2),@−SP	2		6			1
	STM.L(ERn-ER	n+3),@–SP	2		8			1
STMAC*5	STMAC MACH,E	Rd	1					0*3*6
	STMAC MACL,E	Rd	1					0*3*6
SUB	SUB.B Rs,Rd		1					
	SUB.W #xx:16,R	d	2					
	SUB.W Rs,Rd		1					
	SUB.L #xx:32,EF	۶d	3					
	SUB.L ERs,ERd		1					
SUBS	SUBS #1/2/4,ER	d	1					
SUBX	SUBX #xx:8,Rd		1					
	SUBX Rs,Rd		1					
TAS	TAS @ERd*4		2			2		
TRAPA	TRAPA #x:2	Normal	2	1	2/3*1			2
		Advanced	2	2	2/3*1			2
XOR	XOR.B #xx:8,Rd		1					
	XOR.B Rs,Rd		1					
	XOR.W #xx:16,F	Rd	2					
	XOR.W Rs,Rd		1					
	XOR.L #xx:32,EI	Rd	3					
	XOR.L ERs,ERd		2					
XORC	XORC #xx:8,CC	R	1					
XORC	XORC #xx:8,EXI	R	2					

Notes: 1. 2 when EXR is invalid, 3 when EXR is valid.

2. 5 for concatenated execution, 4 otherwise.

3. An internal operation may require between 0 and 3 additional states, depending on the preceding instruction.

4. Only register ER0, ER1, ER4, or ER5 should be used when using the TAS instruction.

5. These instructions are supported by the H8S/2600 CPU only.

6. The number of states may differ depending on the product. For details, refer to the relevant microcontroller hardware manual of the product in question.

## 2.7 Bus States During Instruction Execution

Table 2.6 indicates the types of cycles that occur during instruction execution by the CPU. See table 2.4 for the number of states per cycle.

#### How to Read the Table:



#### Legend

•	
R:B	Byte-size read
R:W	Word-size read
W:B	Byte-size write
W:W	Word-size write
2nd	Address of 2nd word (3rd and 4th bytes)
3rd	Address of 3rd word (5th and 6th bytes)
4th	Address of 4th word (7th and 8th bytes)
5th	Address of 5th word (9th and 10th bytes)
NEXT	Address of next instruction
EA	Effective address
VEC	Vector address

Figure 2.1 shows timing waveforms for the address bus and the  $\overline{RD}$  and  $\overline{WR}$  (HWR or  $\overline{LWR}$ ) signals during execution of the above instruction with an 8-bit bus, using three-state access with no wait states.



Figure 2.1 Address Bus, RD, and WR (HWR or LWR) Timing (8-Bit Bus, Three-State Access, No Wait States)

Table 2.6	Instruction	Execution	Cycles
-----------	-------------	-----------	--------

Instruction	-	2	m	4	5	9	7	8	6
ADD.B #xx:8,Rd	R:W NEXT								
ADD.B Rs,Rd	R:W NEXT								
ADD.W #xx:16,Rd	R:W 2nd	R:W NEXT							
ADD.W Rs,Rd	R:W NEXT								
ADD.L #xx:32,ERd	R:W 2nd	R:W 3rd	R:W NEXT						
ADD.L ERS, ERd	R:W NEXT								
ADDS #1/2/4,ERd	R:W NEXT								
ADDX #xx:8,Rd	R:W NEXT								
ADDX Rs,Rd	R:W NEXT								
AND.B #xx:8,Rd	R:W NEXT								
AND.B Rs,Rd	R:W NEXT								
AND.W #xx:16,Rd	R:W 2nd	R:W NEXT							
AND.W Rs,Rd	R:W NEXT								
AND.L #xx:32,ERd	R:W 2nd	R:W 3rd	R:W NEXT						
AND.L ERS, ERd	R:W 2nd	R:W NEXT							
ANDC #xx:8,CCR	R:W NEXT								
ANDC #xx:8,EXR	R:W 2nd	R:W NEXT							
BAND #xx:3,Rd	R:W NEXT								
BAND #xx:3,@ERd	R:W 2nd	R:B EA	R:W NEXT						
BAND #xx:3,@aa:8	R:W 2nd	R:B EA	R:W NEXT						
BAND #xx:3,@aa:16	R:W 2nd	R:W 3rd	R:B EA	R:W NEXT					
BAND #xx:3,@aa:32	R:W 2nd	R:W 3rd	R:W 4th	R:B EA	R:W NEXT				
BRA d:8 (BT d:8)	R:W NEXT	R:W EA							
BRN d:8 (BF d:8)	R:W NEXT	R:W EA							
BHI d:8	R:W NEXT	R:W EA							
BLS d:8	R:W NEXT	R:W EA							
BCC d:8 (BHS d:8)	R:W NEXT	R:W EA							
BCS d:8 (BLO d:8)	R:W NEXT	R:W EA							
BNE d:8	R:W NEXT	R:W EA							
BEQ d:8	R:W NEXT	R:W EA							
BVC d:8	R:W NEXT	R:W EA							
BVS d:8	R:W NEXT	R:W EA							
BPL d:8	R:W NEXT	R:W EA							
BMI d:8	R:W NEXT	R:W EA							
BGE d:8	R:W NEXT	R:W EA							
BLT d:8	R:W NEXT	R:W EA							
BGT d:8	R:W NEXT	R:W EA							

Rev. 4.00 Feb 24, 2006 page 292 of 322 REJ09B0139-0400

Instruction	-	2	3	4	5	9	7	8	6
BLE d:8	R:W NEXT	R:W EA							
BRA d:16 (BT d:16)	R:W 2nd	Internal operation, R:W EA 1 state	R:W EA						
BRN d:16 (BF d:16)	R:W 2nd	Internal operation, R:W EA 1 state	R:W EA						
BHI d:16	R:W 2nd	Internal operation, R:W EA 1 state	R:W EA						
BLS d:16	R:W 2nd	Internal operation, R:W EA 1 state	R:W EA						
BCC d:16 (BHS d:16)	R:W 2nd	Internal operation, R:W EA 1 state	R:W EA						
BCS d:16 (BLO d:16)	R:W 2nd	Internal operation, R:W EA 1 state	R:W EA						
BNE d:16	R:W 2nd	Internal operation, R:W EA 1 state	R:W EA						
BEQ d:16	R:W 2nd	Internal operation, R:W EA 1 state	R:W EA						
BVC d:16	R:W 2nd	Internal operation, R:W EA 1 state	R:W EA						
BVS d:16	R:W 2nd	Internal operation, R:W EA 1 state	R:W EA						
BPL d:16	R:W 2nd	Internal operation, R:W EA 1 state	R:W EA						
BMI d:16	R:W 2nd	Internal operation, R:W EA 1 state	R:W EA						
BGE d:16	R:W 2nd	Internal operation, R:W EA 1 state	R:W EA						
BLT d:16	R:W 2nd	Internal operation, R:W EA 1 state	R:W EA						
BGT d:16	R:W 2nd	Internal operation, R:W EA 1 state	R:W EA						
BLE d:16	R:W 2nd	Internal operation, R:W EA 1 state	R:W EA						
BCLR #xx:3,Rd	R:W NEXT								
BCLR #xx:3,@ERd	R:W 2nd			W:B EA					
BCLR #xx:3,@aa:8	R:W 2nd		T	W:B EA					
BCLR #xx:3,@aa:16	R:W 2nd	R:W 3rd	R:B EA	R:W NEXT	W:BEA				

Instruction	-	•	~		Ľ	u	~	a	a
				, i i		,		,	,
BCLK #xx:3, @ aa: 32	K:W Znd	K:W 3rd	K:W 4th	K:B EA	K:W NEX I	W:B EA			
BCLR Rn,Rd	R:W NEXT								
BCLR Rn,@ERd	R:W 2nd	R:B EA	R:W NEXT	W:B EA					
BCLR Rn,@aa:8	R:W 2nd	R:B EA	R:W NEXT	W:B EA					
BCLR Rn,@aa:16	R:W 2nd	R:W 3rd	R:B EA	R:W NEXT	W:B EA				
BCLR Rn,@aa:32	R:W 2nd	R:W 3rd	R:W 4th	R:B EA	R:W NEXT	W:B EA			
BIAND #xx:3,Rd	R:W NEXT								
BIAND #xx:3,@ERd	R:W 2nd	R:B EA	R:W NEXT						
BIAND #xx:3,@aa:8	R:W 2nd	R:B EA	R:W NEXT						
BIAND #xx:3,@aa:16	R:W 2nd	R:W 3rd	R:B EA	R:W NEXT					
BIAND #xx:3,@aa:32	R:W 2nd	R:W 3rd	R:W 4th	R:B EA	R:W NEXT				
BILD #xx:3,Rd	R:W NEXT								
BILD #xx:3,@ERd	R:W 2nd	R:B EA	R:W NEXT						
BILD #xx:3,@aa:8	R:W 2nd	R:B EA	R:W NEXT						
BILD #xx:3,@aa:16	R:W 2nd	R:W 3rd	R:B EA	R:W NEXT					
BILD #xx:3,@aa:32	R:W 2nd	R:W 3rd	R:W 4th	R:B EA	R:W NEXT				
BIOR #xx:3,Rd	R:W NEXT								
BIOR #xx:3,@ERd	R:W 2nd	R:B EA	R:W NEXT						
BIOR #xx:3,@aa:8	R:W 2nd	R:B EA	R:W NEXT						
BIOR #xx:3,@aa:16	R:W 2nd	R:W 3rd	R:B EA	R:W NEXT					
BIOR #xx:3,@aa:32	R:W 2nd	R:W 3rd	R:W 4th	R:B EA	R:W NEXT				
BIST #xx:3,Rd	R:W NEXT								
BIST #xx:3,@ERd	R:W 2nd	R:B EA	R:W NEXT	W:B EA					
BIST #xx:3,@aa:8	R:W 2nd	R:B EA	R:W NEXT	W:B EA					
BIST #xx:3,@aa:16	R:W 2nd	R:W 3rd	R:B EA	R:W NEXT	W:B EA				
BIST #xx:3,@aa:32	R:W 2nd	R:W 3rd	R:W 4th	R:B EA	R:W NEXT	W:B EA			
BIXOR #xx:3,Rd	R:W NEXT								
BIXOR #xx:3,@ERd	R:W 2nd	R:B EA	R:W NEXT						
BIXOR #xx:3,@aa:8	R:W 2nd	R:B EA	R:W NEXT						
BIXOR #xx:3,@aa:16	R:W 2nd	R:W 3rd	R:B EA	R:W NEXT					
BIXOR #xx:3,@aa:32	R:W 2nd	R:W 3rd	R:W 4th	R:B EA	R:W NEXT				
BLD #xx:3,Rd	R:W NEXT								
BLD #xx:3,@ERd	R:W 2nd	R:B EA	R:W NEXT						
BLD #xx:3,@aa:8	R:W 2nd	R:B EA	R:W NEXT						
BLD #xx:3,@aa:16	R:W 2nd	R:W 3rd	R:B EA	R:W NEXT					
BLD #xx:3,@aa:32	R:W 2nd	R:W 3rd	R:W 4th	R:B EA	R:W NEXT				
BNOT #xx:3,Rd	R:W NEXT								

Instruction	1	2	3	4	2	9	7	8	6
BNOT #xx:3,@ERd	R:W 2nd	R:B EA	R:W NEXT	W:B EA					
BNOT #xx:3,@aa:8	R:W 2nd	R:B EA	R:W NEXT	W:B EA					
BNOT #xx:3,@aa:16	R:W 2nd	R:W 3rd	R:B EA	R:W NEXT	W:B EA				
BNOT #xx:3,@aa:32	R:W 2nd	R:W 3rd	R:W 4th	R:B EA	R:W NEXT	W:B EA			
BNOT Rn,Rd	R:W NEXT								
BNOT Rn,@ERd	R:W 2nd	R:B EA	R:W NEXT	W:B EA					
BNOT Rn,@aa:8	R:W 2nd	R:B EA	R:W NEXT	W:B EA					
BNOT Rn,@aa:16	R:W 2nd	R:W 3rd	R:B EA	R:W NEXT	W:B EA				
BNOT Rn,@aa:32	R:W 2nd	R:W 3rd	R:W 4th	R:B EA	R:W NEXT	W:B EA			
BOR #xx:3,Rd	R:W NEXT								
BOR #xx:3, @ERd	R:W 2nd	R:B EA	R:W NEXT						
BOR #xx:3, @aa:8	R:W 2nd	R:B EA	R:W NEXT						
BOR #xx:3, @aa:16	R:W 2nd	R:W 3rd	R:B EA	R:W NEXT					
BOR #xx:3, @aa:32	R:W 2nd	R:W 3rd	R:W 4th	R:B EA	R:W NEXT				
BSET #xx:3,Rd	R:W NEXT								
BSET #xx:3,@ERd	R:W 2nd	R:B EA	R:W NEXT	W:B EA					
BSET #xx:3,@aa:8	R:W 2nd	R:B EA	R:W NEXT	W:B EA					
BSET #xx:3,@aa:16	R:W 2nd	R:W 3rd	R:B EA	R:W NEXT	W:B EA				
BSET #xx:3,@aa:32	R:W 2nd	R:W 3rd	R:W 4th	R:B EA	R:W NEXT	W:B EA			
BSET Rn,Rd	R:W NEXT								
BSET Rn,@ERd	R:W 2nd	R:B EA	R:W NEXT	W:B EA					
BSET Rn,@aa:8	R:W 2nd	R:B EA	R:W NEXT	W:B EA					
BSET Rn,@aa:16	R:W 2nd	R:W 3rd	R:B EA	R:W NEXT	W:B EA				
BSET Rn,@aa:32	R:W 2nd	R:W 3rd	R:W 4th	R:B EA	R:W NEXT	W:B EA			
BSR d:8 Normal	R:W NEXT	R:W EA	W:W stack						
Advanced	d R:W NEXT	R:W EA	W:W stack (H)	W:W stack (L)					
BSR d:16 Normal	R:W 2nd	Internal operation, R:W EA 1 state	R:W EA	W:W stack					
Advance	Advanced R:W 2nd	Internal operation, R:W EA 1 state	R:W EA	W:W stack (H)	W:W stack (L)				
BST #xx:3,Rd	R:W NEXT								
BST #xx:3,@ERd	R:W 2nd	R:B EA	R:W NEXT	W:B EA					
BST #xx:3,@aa:8	R:W 2nd	R:B EA	R:W NEXT	W:B EA					
BST #xx:3,@aa:16	R:W 2nd	R:W 3rd	R:B EA	R:W NEXT	W:B EA				
BST #xx:3,@aa:32	R:W 2nd	R:W 3rd	R:W 4th	R:B EA	R:W NEXT	W:B EA			
BTST #xx:3,Rd	R:W NEXT								
BTST #xx:3,@ERd	R:W 2nd	R:B EA	R:W NEXT						

3.8 aar8         F.W.2nd         R.B.EA         R.W.NEXT         R.W.NEXT         R.W.NEXT           3.8 aar32         R.W.Znd         R.W.3nd         R.W.NEXT         R.W.NEXT         R.W.NEXT         R.W.NEXT           3.8 aar32         R.W.NEXT         R.W.NEXT         R.W.NEXT         R.W.NEXT         R.W.NEXT           3.8 aar32         R.W.NEXT         R.W.NEXT         R.W.NEXT         R.W.NEXT         R.W.NEXT           2.9 aar32         R.W.V.Znd         R.B.EA         R.W.NEXT         R.W.NEXT         R.W.NEXT           2.9 aar32         R.W.V.Znd         R.B.EA         R.W.NEXT         R.W.NEXT         R.W.NEXT           2.9 aar32         R.W.V.Znd         R.B.EA         R.W.NEXT         R.W.NEXT         R.W.NEXT           3.0 aar32         R.W.V.Znd         R.W.Ath         R.B.EA         R.W.NE         R.W.NE           3.0 aar32         R.W.NEXT         R.W.NEXT         R.W.NEXT         R.W.NEXT         R.W.NEXT           3.0 aar32         R.W.NEXT         R.W.NEXT         R.W.NEXT         R.W.NE         R.W.NE           3.0 aar32         R.W.NEXT         R.W.NEXT         R.W.NEXT         R.W.NEXT         R.W.NEXT           3.0 aar32         R.W.NEXT         R.W.NEXT         R	Instruction	-	2		4	5	9	7	8	6
3@@#:16         R.W. WAT         R.W. NEXT	BTST #xx:3,@aa:8	R:W 2nd	R:B EA	R:W NEXT						
3@æn32         R.W.NZM         R.W.WZM         R.W.WEXT         R.W.NEXT         R.W.NEXT         R.W.NEXT           BERG         R.W.WZM         R.E.E.E.A         R.W.NEXT         R.W.NEXT         R.W.NEXT           Berl         R.W.Zmd         R.E.E.A         R.W.NEXT         R.W.NEXT         R.W.NEXT           Berl         R.W.Zmd         R.W.NEXT         R.W.NEXT         R.W.NEXT         R.W.NEXT           Berl         R.W.Zmd         R.W.WZM         R.W.NEXT         R.W.NEXT         R.W.NEXT           Berl         R.W.NEXT         R.W.NEXT         R.W.NEXT         R.W.NEXT         R.W.NEXT           3.@estife         R.W.NEXT         R.W.NEXT         R.W.NEXT	BTST #xx:3,@aa:16	R:W 2nd	R:W 3rd	R:B EA	R:W NEXT					
dd         R.W.NEXT         M.N.EXT         R.W.N.EXT         R.W.N.EXT         R.W.N.EXT         M.N.EXT         R.W.N.EXT         <	BTST #xx:3,@aa:32	R:W 2nd	R:W 3rd	R:W 4th	R:B EA	R:W NEXT				
BERd         R.W.Zud         R.B. EA         R.W.NEXT         MEXT           Baa:3         R.W.Zud         R.B. EA         R.W.NEXT         R.W.NEXT           Baa:32         R.W.NEXT         R.W.NEXT         R.W.NEXT         R.W.NEXT           Baa:32         R.W.NEXT         R.W.NEXT         R.W.NEXT         R.W.NEXT           3.6 Ref         R.W.NEXT         R.W.NEXT         R.W.NEXT         R.W.NEXT           3.6 asis         R.W.NEXT         R.W.NEXT         R.W.NEXT         R.	BTST Rn,Rd	R:W NEXT								
Baar3         R.W. Zud         R.W. MEXT         M.W. KT         M.W.KT         <	BTST Rn, @ERd	R:W 2nd	R:B EA	R:W NEXT						
Basa:16         R:W 2nd         R:W With         R:W NEXT         R:W NEXT         R:W NEXT           Baa.32         R:W NEXT         R:W NEXT         R:W NEXT         R:W NEXT         R:W NEXT           3.0.0 ER         R:W NEXT         R:W NEXT         R:W NEXT         R:W NEXT         R:W NEXT           3.0.0 ER         R:W 2nd         R:W NEXT         R:W NEXT         R:W NEXT         R:W NEXT           3.0.0 ER         R:W 2nd         R:W 3nd         R:B EA         R:W NEXT         R:W NEXT           3.0 aar.16         R:W NEXT         R:W NEXT         R:W NEXT         R:W NEXT         R:W NEXT           3.0 aar.32         R:W NEXT         R:W NEXT         R:W NEXT         R:W NEXT         R:W NEXT           3.0 aar.32         R:W NEXT         R:W NEXT         R:W NEXT         R:W NEXT         R:W NEXT           3.0 aar.32         R:W NEXT         R:W NEXT         R:W NEXT         R:W NEXT         R:W NEXT           3.0 aar.32         R:W NEXT         R:W NEXT         R:W NEXT         R:W NEXT         R:W NEXT           3.0 aar.22         R:W NEXT         R:W NEXT         R:W NEXT         R:W NEXT         R:W NEXT           R:0 A.0         R:W NEXT         R:W NEXT         R:W NEXT	BTST Rn, @aa:8	R:W 2nd	R:B EA	R:W NEXT						
Basa32         R.W Mark         R.W Mark         R.W NEXT         R.W NEXT         R.W NEXT           3.Red         R.W NEXT         R.W NEXT         R.W NEXT         R.W NEXT         R.W NEXT           3.0.0 RM         R.W 2nd         R.B EA         R.W NEXT         R.W NEXT         R.W NEXT           3.0.0 RW         R.W 2nd         R.W BEA         R.W NEXT         R.W NEXT         R.W NEXT           3.0.0 RW         R.W NEXT         R.W SAT         R.W NEXT         R.W NEXT         R.W NEXT           3.0.0 RW         R.W NEXT         R.W NEXT         R.M NEXT         R.M NEXT         R.M NEXT           3.0.0 RW         R.W NEXT         R.W NEXT         R.M NEXT         R.M NEXT         R.M NEXT           R.M         R.W NEXT         R.W NEXT         R.M NEXT         R.M NEXT         R.M NEXT           R.M         R.W NEXT         R.W NEXT         R.M NEXT         R.M NEXT         R.M NEXT           R.M         R.W NEXT         R.W NEXT         R.M NEXT         R.M NEXT         R.M NEXT           R.M         R.W NEXT         R.W NEXT         R.M NEXT         R.M NEXT         R.M NEXT           R.M         R.W NEXT         R.W NEXT         R.M NEXT         R.M NEXT         R.M	BTST Rn, @aa:16	R:W 2nd	R:W 3rd	R:B EA	R:W NEXT					
3Rd     RW NEXT     FM NEXT     MEXT     Mext       3.00000     RW Znd     RB EA     RW NEXT     RW NEXT       3.000000     RW Znd     RW MEXT     RW NEXT       3.000000     RW Znd     RW MEXT     RW NEXT       3.000000     RW Znd     RW MEXT     RW NEXT       3.0000000     RW NEXT     RW NEXT     RW NEXT       8.Rd     RW NEXT     RW NEXT     RW NEXT       Rd     RW NEXT     RW NEXT     RW NEXT       Stated     RW NEXT     RW NEXT     RW NEXT       Rd     RW NEXT     RW NEXT     RW RAT       Rd     RW NEXT     RW NEXT     RW NEXT       Stated     RW NEXT     RW NEXT     RW RAT       Stated     RW NEXT     RW NEXT	BTST Rn, @aa:32	R:W 2nd	R:W 3rd	R:W 4th	R:B EA	R:W NEXT				
3.000000000000000000000000000000000000	BXOR #xx:3,Rd	R:W NEXT								
3. @azs     R.W. Znd     R.W. BEA     R.W. NEXT     R.W. NEXT       3. @aar16     R.W. Znd     R.W. Ward     R.W. NEXT     R.W. NEXT       3. @aar15     R.W. NEXT     Internal operation.     R.W. NEXT     R.W. NEXT       3. @aar16     R.W. NEXT     Internal operation.     R.W. NEXT     R.W. NEXT       3. @aar32     R.W. NEXT     Internal operation.     R.W. NEXT     R.W. NEXT       8. Rd     R.W. NEXT     Istate®     R.W. NEXT     R.W. NEXT       8. Rd     R.W. NEXT     R.W. NEXT     R.W. NEXT     R.W. NEXT       8. R.W. NEXT     R.W. NEXT     R.W. NEXT     R.W. NEXT     R.W. NEXT       9. SERd     R.W. NEXT     R.W. NEXT     R.W. NEXT     R.W. NEXT       9. S.R.     R.W. NEXT     Internal operation. 11 states     R.M. States       2. Rd     R.W. NEXT     Internal operation. 13 states     R.W. States       8. Rd     R.W. NEXT     Internal operation. 13 states     R.W. States       8. Rd     R.W. NEXT     Internal operation. 13 states     R.W. States       8. Rd     R.W. NEXT     Internal operation. 13 states     R.W. States       8. Rd     R.W. NEXT     Internal operation. 13 states     R.W. States       8. Rd     R.W. NEXT     R.W. States     R.W. States <td>BXOR #xx:3, @ERd</td> <td>R:W 2nd</td> <td>R:B EA</td> <td>R:W NEXT</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td>	BXOR #xx:3, @ERd	R:W 2nd	R:B EA	R:W NEXT						
3.@aar16         R.W. Znd         R.W. Srd         R.W. NEXT         R	BXOR #xx:3, @aa:8	R:W 2nd	R:B EA	R:W NEXT						
3.@aa:32         R.W Znd         R.W Ath         R.B EA         R.W NEXT           8.W NEXT         Internal operation         1 state*9         8 W NEXT         New NEXT         New NEXT           8.Rd         R.W NEXT         Internal operation         1 state*9         8 W NEXT         New NEXT           8.Rd         R.W NEXT         EXW NEXT         EXW NEXT         EXW NEXT         EXW NEXT           8.Rd         R.W NEXT         EXW NEXT         EXW NEXT         EXW NEXT         EXW NEXT           8.Rd         R.W NEXT         EXW NEXT         EXW NEXT         EXW NEXT         EXW NEXT           8.Rd         R.W NEXT         EXW NEXT         EXW NEXT         EXW NEXT         EXW NEXT           8.Rd         R.W NEXT         Internal operation, 11 states         EXW NEXT         EXW NEXT           8.Rd         R.W NEXT         Internal operation, 11 states         EXW NEXT         EXW NEXT           8.Rd         R.W NEXT         Internal operation, 11 states         EXW NEXT         EXW NEXT           8.Rd         R.W NEXT         Internal operation, 11 states         EXW NEXT         EXW NEXT           8.Rd         R.W NEXT         Internal operation, 11 states         EXW NEXT         EXW NEXT	BXOR #xx:3,@aa:16	R:W 2nd	R:W 3rd	R:B EA	R:W NEXT					
RW NEXT         Internal operation.         Internal operation. <thi< td=""><td>BXOR #xx:3, @aa:32</td><td>R:W 2nd</td><td>R:W 3rd</td><td>R:W 4th</td><td>R:B EA</td><td>R:W NEXT</td><td></td><td></td><td></td><td></td></thi<>	BXOR #xx:3, @aa:32	R:W 2nd	R:W 3rd	R:W 4th	R:B EA	R:W NEXT				
B.Rd     RW NEXT     B.W NEXT<	CLRMAC*	R:W NEXT	Internal operation, 1 state* <sup>9</sup>							
Rd         R:W NEXT         R:M NEXT	CMP.B #xx:8,Rd	R:W NEXT								
x16.Rd         R:W Dad         R:W NEXT         R:M NEXT <t< td=""><td>CMP.B Rs,Rd</td><td>R:W NEXT</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></t<>	CMP.B Rs,Rd	R:W NEXT								
RdR:W NEXTMMMM::2.ERdR:W NEXTR:W NEXTR:W NEXTR:W NEXTR:W NEXT $E.RMR:W NEXTR:W NEXTR:W NEXTR:W NEXTR:W NEXTR:W NEXTR:W NEXTR:W NEXTR:M RR:M R2.RdR:W NEXTR:W NEXTR:M RR:M R2.RdR:W NEXTR:W NEXTR:M RR:M R2.RdR:W NEXTInternal operation.11 states2.RdR:W NEXTInternal operation.13 states5.RdR:W NEXTR:B EAs "1R:B EAs "26R:W NEXTR:W NEXTR:B CAS "17R:W NEXTR:B CAS "1R:B EAs "27R:W NEXTR:W NEXTR:W NEXT6R:W NEXTR:W NEXTR:B CAS "27R:W NEXTR:B CAS "1R:B EAS "27R:W NEXTR:W NEXTR:M EAS "27R:W NEXTR:M R$	CMP.W #xx:16,Rd	R:W 2nd	R:W NEXT							
:32.Edd     R:W 2nd     R:W NEXT     R:B Edd *1     R:B Edd *2       2.E Edd     R:W NEXT     Internal operation, 11 states     R:B Edd *2     W:B Edd *2       3.E. Rd     R:W NEXT     Internal operation, 13 states     R:B Edd *2     W:B Edd *2       3.E. Rd     R:W NEXT     R:B Edd *1     R:B Edd *2     W:B Edd *2       3.E. Rd     R:W NEXT     R:B Edd *1     R:B Edd *2     W:B Edd *2       3.E. Rd     R:W NEXT     R:B Edd *1     R:B Edd *2     W:B Edd *2       3.E. Rd     R:W NEXT     R:B Edd *1     R:B Edd *2     W:B Edd *2       3.E. Rd     R:W NEXT     R:B Edd *1     R:B Edd *2     W:B Edd *2       3.E. Rd     R:W NEXT     R:B Edd *1     R:B Edd *2     M:B	CMP.W Rs,Rd	R:W NEXT								
S,ERd     R;W NEXT     Image: Second	CMP.L #xx:32,ERd	R:W 2nd	R:W 3rd	R:W NEXT						
$\begin{tabular}{ c c c c c c c c c c c c c c c c c c c$	CMP.L ERS, ERd	R:W NEXT								
	DAA Rd	R:W NEXT								
R:W NEXT $R:W NEXT$ $R:W RET = R R R R R R R R R R R R R R R R R R$	DAS Rd	R:W NEXT								
Z.Rd         R:W NEXT         Mex         M	DEC.B Rd	R:W NEXT								
LERd     R:W NEXT     Immal operation, 11 states       (s,Rd     R:W Znd     R:W NEXT     Internal operation, 11 states       (s,Rd     R:W NEXT     Internal operation, 11 states       (s,Rd     R:W NEXT     Internal operation, 13 states       (s,Rd     R:W NEXT     Internal operation, 13 states       (s,Rd     R:W NEXT     Internal operation, 13 states       (s,Rd     R:W NEXT     Internal operation, 13 states       (s,Rd     R:W NEXT     Internal operation, 13 states       (s,Rd     R:W SEAs***     R:B EA8***       (s,Rd     R:B EA8***     R:B EA8***       (s,W NEXT     R:B EA8***     W:B EA4***       (d     R:W NEXT     R:B EA8***       (d     R:W NEXT     R:D EA4***       (d     R:W NEXT     R:M EXT	DEC.W #1/2,Rd	R:W NEXT								
S,Rd         R:W Znd         R:W NEXT         Internal operation, 11 states           Rs,ERd         R:W NEXT         Internal operation, 10 states           S,Rd         R:W NEXT         Internal operation, 10 states           S,Rd         R:W NEXT         Internal operation, 11 states           S,ERd         R:W NEXT         Internal operation, 10 states           S,ERd         R:W NEXT         Internal operation, 10 states           S,ERd         R:W NEXT         Internal operation, 10 states           V         R:W Znd         R:B EAs *1         R:B EAs *2         W:B EAd *2           V         R:W NEXT         R:B EAd *1         R:B EAg *2         W:B EAd *2           d         R:W NEXT         R:B EAd *1         R:B EAg *2         W:B EAd *2           d         R:W NEXT         R:B EAg *1         R:B EAg *2         W:B EAd *2           d         R:W NEXT         R:D EAd *1         R:B EAg *2         W:B EAd *2           d         R:W NEXT         R:D EAd *1         R:B EAg *2         W:B EAd *2           d         R:W NEXT         R:W NEXT         R:W NEXT         R:M EAG         R:M EAG	DEC.L #1/2,ERd	R:W NEXT								
Rs, ERd         R:W Znd         R:W NEXT         Internal operation, 19 states           Is, Rd         R:W NEXT         Internal operation, 11 states         Internal operation, 11 states           Rs, Krad         R:W NEXT         Internal operation, 11 states         Internal operation, 12 states           Rs, Krad         R:W NEXT         Internal operation, 19 states         Internal operation, 19 states           V         R:W Znd         R:B EAs "1         R:B EAs "2         W:B EAd "2           V         R:W Znd         R:B EAs "1         R:B EAs "2         W:B EAd "2           V         R:W NEXT         R:B EAd "1         R:B EAs "2         W:B EAd "2           d         R:W NEXT         R:D R         R:D R         R:D R         R:D R           d         R:W NEXT         R:W NEXT         R:W NEXT         R:W NEXT         R:W NEXT         R:W NEXT	DIVXS.B Rs,Rd	R:W 2nd	R:W NEXT	Internal operation,	11 states					
Is,Rd     R:W NEXT     Internal operation, 11 states       R:W NEXT     Internal operation, 19 states       R:W NEXT     Internal operation, 19 states       R:W NEXT     R:B EAs *1     R:B EAs *2     W:B EAs *2       V     R:W Znd     R:B EAs *1     R:B EAs *2     W:B EAs *2       V     R:W Znd     R:B EAs *1     R:B EAs *2     W:B EAs *2       V     R:W NEXT     R:B EAs *1     R:B EAs *2     W:B EAs *2       V     R:W NEXT     R:D R     R:B EAs *2     W:B EAs *2       V     R:W NEXT     R:D R     R:D R     R:D R       R:W NEXT     R:W NEXT     R:W NEXT     R:M R     R:D R	DIVXS.W Rs,ERd	R:W 2nd	R:W NEXT	Internal operation,	19 states					
Ray         R:W NEXT         Internal operation, 19 states           8:         R:W Znd         R:B EAs *1         R:B EAs *2         W:B EAd *2           V         R:W Znd         R:B EAs *1         R:B EAd *1         R:B EAs *2         W:B EAd *2           V         R:W Znd         R:B EAs *1         R:B EAd *1         R:B EAs *2         W:B EAd *2           V         R:W NEXT         R:B EAd *1         R:B EAg *2         W:B EAd *2           A         R:W NEXT         R:B EAd *1         R:B EAg *2         W:B EAd *2           A         R:W NEXT         R:M NEXT         P         P         R:M R*2           A         R:W NEXT         R:W NEXT         R:W NEXT         R:W NEXT         P         P	DIVXU.B Rs,Rd	R:W NEXT	Internal operation,	11 states						
8         R:W 2nd         R:B EA3 <sup>±1</sup> R:B EAd <sup>±1</sup> W:B EAd <sup>±2</sup> W:B EAd <sup>±2</sup> V         R:W 2nd         R:B EA3 <sup>±1</sup> R:B EAd <sup>±1</sup> R:B EA3 <sup>±2</sup> W:B EAd <sup>±2</sup> d         R:W NEXT         R:B EA3 <sup>±1</sup> R:B EAd <sup>±1</sup> R:B EA3 <sup>±2</sup> W:B EAd <sup>±2</sup> d         R:W NEXT         P         P         P         P           d         R:W NEXT         P         P         P         P           d         R:W NEXT         P         P         P         P           d         R:W NEXT         P         P         P         P           kd         R:W NEXT         P         P         P         P	DIVXU.W Rs,ERd	R:W NEXT	Internal operation,	19 states						
V         R:W 2nd         R:B EA3 *1         R:B EAd *1         R:B EA3 *2         W:B EAd *2           d         R:W NEXT         P         P         P         P         P         P           d         R:W NEXT         P         P         P         P         P         P         P           d         R:W NEXT         P<	EEPMOV.B	R:W 2nd	R:B EAs *1	R:B EAd *1	R:B EAs *2	W:B EAd *2	R:W NEXT			
d R:W NEXT → kd R:W NEXT → d R:W NEXT → kd R:W NEXT → Kd R:W NEXT →	EEPMOV.W	R:W 2nd	R:B EAs *1	R:B EAd *1	R:B EAs *2	W:B EAd *2	R:W NEXT			
d d Xd	EXTS.W Rd	R:W NEXT			← Repeated	n times <sup>∗3</sup>				
g p	EXTS.L ERd	R:W NEXT								
Sd	EXTU.W Rd	R:W NEXT								
	EXTU.L ERd	R:W NEXT								
	INC.B Rd	R:W NEXT								

Instruction		-	2	3	4	5	9	7	8	6
INC.W #1/2,Rd		R:W NEXT								
INC.L #1/2,ERd		R:W NEXT								
JMP @ERn		R:W NEXT	R:W EA							
JMP @aa:24		R:W 2nd	Internal operation, R:W EA 1 state	R:W EA						
JMP @ @aa:8 No	Normal	R:W NEXT	R:Waa:8	Internal operation, R:W EA 1 state	R:W EA					
PA	ivanced	Advanced R:W NEXT	R:W aa:8 (H)	R:W aa:8 (L)	Internal operation, R:W EA 1 state	R:W EA				
JSR @ERn No	Normal	R:W NEXT	R:WEA	W:W stack						
Ad	lvanced	Advanced R:W NEXT	R:WEA	ck (H)	W:W stack (L)					
JSR @aa:24 No	Normal	R:W 2nd	Internal operation, R:W EA 1 state		W:W stack					
PA	Advanced R:W 2nd	R:W 2nd	Internal operation, R:W EA 1 state		W:W stack (H)	W:W stack (L)				
JSR @@aa:8 No	Normal	R:W NEXT	R:W aa:8	W:W stack	R:W EA					
Ad	lvanced	Advanced R:W NEXT	R:W aa:8 (H)	R:W aa:8 (L)	W:W stack (H)	W:W stack (L)	R:W EA			
LDC #xx:8,CCR		R:W NEXT								
LDC #xx:8,EXR		R:W 2nd	R:W NEXT							
LDC Rs,CCR		R:W NEXT								
LDC Rs,EXR		R:W NEXT								
LDC @ERs,CCR		R:W 2nd	R:W NEXT	R:W EA						
LDC @ERs,EXR		R:W 2nd	R:W NEXT	R:W EA						
LDC @ (d: 16, ERs), CCR		R:W 2nd	R:W 3rd	R:W NEXT	R:W EA					
LDC @(d:16,ERs),EXR		R:W 2nd	R:W 3rd	R:W NEXT	R:W EA					
LDC @ (d: 32, ERs), CCR		R:W 2nd	R:W 3rd	R:W 4th	R:W 5th	R:W NEXT	R:W EA			
LDC @(d:32,ERs),EXR		R:W 2nd		R:W 4th	R:W 5th	R:W NEXT	R:W EA			
LDC @ERs+,CCR		R:W 2nd	R:W NEXT	Internal operation, R:W EA 1 state	R:W EA					
LDC @ERs+,EXR		R:W 2nd	R:W NEXT	Internal operation, R:W EA	R:W EA					
LDC @aa:16,CCR		R:W 2nd	R:W 3rd	R:W NEXT	R:W EA					
LDC @aa:16,EXR		R:W 2nd	R:W 3rd	R:W NEXT	R:W EA					
LDC @aa:32,CCR		R:W 2nd	R:W 3rd	R:W 4th		R:W EA				
LDC @aa:32,EXR		R:W 2nd		R:W 4th		R:W EA				
LDM.L @SP+,(ERn-ERn+1)		R:W 2nd	R:W NEXT	Internal operation, R:W stack (H)*3 1 state		R:W stack (L)*3				

Instruction	-	2	3	4	S	9	7	8	6
LDM.L @SP+,(ERn-ERn+2)	R:W 2nd	R:W NEXT	Internal operation, R:W stack (H)*3 1 state		R:W stack (L)*3				
LDM.L @SP+,(ERn-ERn+3)	R:W 2nd	R:W NEXT	Internal operation, R:W stack (H)*3 1 state		R:W stack (L)*3				
LDMAC ERS,MACH*11	R:W NEXT	Internal operation, 1 state*9		← Repeated n times*3	n times $^{*3} \longrightarrow$				
LDMAC ERs,MACL*11	R:W NEXT	Internal operation, 1 state*9							
MAC @ERn+,@ERm+*11	R:W 2nd	R:W NEXT	R:W EAn	R:W EAm					
MOV.B #xx:8,Rd	R:W NEXT								
MOV.B Rs,Rd	R:W NEXT								
MOV.B @ERs,Rd	R:W NEXT	R:B EA							
MOV.B @ (d: 16, ERs), Rd	R:W 2nd	R:W NEXT	R:B EA						
MOV.B @ (d: 32, ERs), Rd	R:W 2nd	R:W 3rd	R:W 4th	R:W NEXT	R:B EA				
MOV.B @ ERs+, Rd	R:W NEXT	Internal operation, R:B EA 1 state	R:B EA						
MOV.B @aa:8,Rd	R:W NEXT	R:B EA							
MOV.B @aa:16,Rd	R:W 2nd	R:W NEXT	R:B EA						
MOV.B @aa:32,Rd	R:W 2nd	R:W 3rd	R:W NEXT	R:B EA					
MOV.B Rs,@ERd	R:W NEXT	W:B EA							
MOV.B Rs,@(d:16,ERd)	R:W 2nd	R:W NEXT	W:B EA						
MOV.B Rs,@(d:32,ERd)	R:W 2nd	R:W 3rd	R:W 4th	R:W NEXT	W:B EA				
MOV.B Rs, @-ERd	R:W NEXT	Internal operation, W:B EA 1 state	W:B EA						
MOV.B Rs,@aa:8	R:W NEXT	W:B EA							
MOV.B Rs,@aa:16	R:W 2nd	R:W NEXT	W:B EA						
MOV.B Rs,@aa:32	R:W 2nd	R:W 3rd	R:W NEXT	W:B EA					
MOV.W #xx:16,Rd	R:W 2nd	R:W NEXT							
MOV.W Rs,Rd	R:W NEXT								
MOV.W @ERs,Rd	R:W NEXT	R:W EA							
MOV.W @(d:16,ERs),Rd	R:W 2nd	R:W NEXT	R:W EA						
MOV.W @(d:32,ERs),Rd	R:W 2nd	R:W 3rd	R:W 4th	R:W NEXT	R:W EA				
MOV.W @ERs+, Rd	R:W NEXT	Internal operation, R:W EA 1 state	R:W EA						
MOV.W @aa:16,Rd	R:W 2nd	R:W NEXT	R:W EA						
MOV.W @aa:32,Rd	R:W 2nd		R:W NEXT	R:B EA					
MOV.W Rs,@ERd	R:W NEXT	W:W EA							

0         RW2dd         RW	Instruction	-	2	9	4	5	9	2	8	6
M.M. Mathematical Schwart         R.W. Mathematical Schwart         R.W. MEXT         W.W. KEA         M.W. KEA         K.W. MEAT         R.W. MEAT         R.W. MEAT         R.W. MEAT         R.W. MEAT         R.W. MEAT         R.W. KEA         R.W. K	MOVW Rs @ (d:16 FRd)	R-W 2nd	R-W NEXT	W-W FA						
16         R.W. Zud         R.W. NEXT         W.W. EA         W.W. EA         W.W. EA         W.W. EA         M.W. EA	MOV.W Rs,@(d:32,ERd)	R:W 2nd	R:W 3rd	R:E 4th	R:W NEXT	W:W EA				
22         R.W. Dadi         R.W. Ward         W.W. Bard         W.W. Bard         W.W. Bard         W.W. Bard         R.W. Werd         Istand         Rev. Werd         Istand         Rev. MeXT         Rev. MeXA	MOV.W Rs,@aa:16	R:W 2nd	R:W NEXT	W:W EA						
dd         R.W. NEXT         Internal operation, W.W. EA         F.W. NEXT         Internal operation, W.W. EA         F.W. NEXT         Internal operation, W.W. EA         F.W. NEXT         R.W.	MOV.W Rs,@aa:32	R:W 2nd	R:W 3rd	R:W NEXT	W:W EA					
td         R.W.2nd         R.W.Bard         R.	MOV.W Rs,@-ERd	R:W NEXT	Internal operation 1 state	, w:w EA						
RW NEXT         KW MEXT         KW EA+2         <	MOV.L #xx:32,ERd	R:W 2nd	R:W 3rd	R:W NEXT						
Rs.Erd         R.W.Znd         R.W.NEXT         R.W.EA         R.W.EA         R.W.EA         R.W.EA           ci.5.EFs).ERd         R.W.Znd         R.W.WIKT         R.W.KT         R.W.EA         R.W.EA           ci.3.2.EFs).ERd         R.W.Znd         R.W.WIKT         R.W.KT         R.W.EA         R.W.EA           ci.3.2.EFs).ERd         R.W.Znd         R.W.WIKT         R.W.KT         R.W.EA         R.W.EA           ci.3.2.EFs).ERd         R.W.Znd         R.W.WIKT         R.W.KT         R.W.KA         R.W.EA           ci.3.2.EFd         R.W.Znd         R.W.NEXT         R.W.KA         R.W.KA         R.W.EA           ci.3.2.EFd         R.W.Znd         R.W.NEXT         R.W.KA         R.W.KA         R.W.EA           s.80(ct)6.Erd)         R.W.Znd         R.W.KT         W.W.EA         R.W.KA         R.W.EA           s.0.0(ct)6.Erd)         R.W.Znd         R.W.KT         W.W.EA         W.W.EA         R.W.EA           s.0.0(ct)6.Erd)         R.W.Znd         R.W.KT         W.W.EA         W.W.EA         R.W.EA           s.0.0(ct)6.Erd)         R.W.Znd         R.W.WEA         W.W.EA         W.W.EA         R.W.EA           s.0.0(ct)6.Erd)         R.W.Znd         R.W.WEA         W.W.EA	MOV.L ERS, ERd	R:W NEXT								
d:16.EFs),ERd         R:W 2nd         R:W 3nd         R:W KeA         R:W MEXT	MOV.L @ERS,ERd	R:W 2nd	R:W NEXT	R:W EA	R:W EA+2					
d:3:2.FRs), Erkd         R:W 2md         R:W MeXT	MOV.L @(d:16,ERs),ERd	R:W 2nd	R:W 3rd		R:W EA	R:W EA+2				
IRN-ERd         R:W Ward         R:W MEXT         Immal operation         R:W EA+2         R:W EA+2           ac:32.ERd         R:W 2nd         R:W NEXT         R:W NEXT         R:W MEXT         R:W EA+2           ac:32.ERd         R:W 2nd         R:W NEXT         W:W EA         R:W EA+2           ac:32.ERd         R:W 2nd         R:W NEXT         W:W EA         R:W EA+2           s@(!16.ERd)         R:W 2nd         R:W NEXT         W:W EA         R:W EA+2           s@(15.ERd)         R:W 2nd         R:W Ath         R:W NEXT         W:W EA+2           s@(15.ERd)         R:W 2nd         R:W Ath         R:W EA         M:W EA           s@(15.ERd)         R:W 2nd         R:W Ath         R:W EA         M:W EA           s@(15.ERd)         R:W 2nd         R:W 16A         M:W EA         M:W EA           s@(15.E.Rd)         R:W 2nd         R:W 16A         M:W EA         M:W EA           s@(15.E.Rd)         R:W 2nd         R:W 16A         M:W EA         M:W EA           s@(15.E.Rd)         R:W 2nd         R:W 16A         M:W EA         M:W EA           s@(15.E.Rd)         R:W 2nd         R:W 16A         M:W EA         M:W EA           S@(15.E.Rd)         R:W 2nd         R:	MOV.L @(d:32,ERs),ERd	R:W 2nd	R:W 3rd	R:W 4th	R:W 5th	R:W NEXT	R:W EA	R:W EA+2		
a:16,ERdR:W 2ndR:W 3ndR:W NEXTR:W EA+2Aa:32,ERdR:W 2ndR:W 3ndR:W 10R:W 10R:W 10R:W 10a:32,ERdR:W 2ndR:W 10R:W 10R:W 10R:W 10R:W 10s.@(1:16,ERd)R:W 2ndR:W 10W:W EAK:W 10R:W 10s.@(1:32,ERd)R:W 2ndR:W 10R:W 10W:W EAM:W EAs.@(1:32,ERd)R:W 2ndR:W 10R:W 10W:W EAM:W EAs.@(1:32,ERd)R:W 2ndR:W 10R:W 10W:W EAM:W EAs.@a:16R:W 2ndR:W 10R:W 10M:W EAM:W EAs.@a:16R:W 2ndR:W 10R:W 10M:W EAM:W EAs.@a:16R:W 2ndR:W 10R:W 10M:W EAM:W EAs.@a:16R:W 2ndR:W 10R:W 10M:W EAM:W EAs.@a:16R:W 2ndR:W 10R:W 10R:W 10M:W EAs.@a:16R:W 2ndR:W 10R:W 10R:W 10M:W 10s.@a:16R:W 2ndR:W 10R:W 10R:W 10M:W 10s.@a:16R:W 2ndR:W 10R:W 10R:W 10M:W 10s.@a:16R:W 2ndR:W 10R:W 10R:W 10M:W 10s.RedH85/200R:W 2ndR:W 10R:W 10R:W 10R:RedH85/200R:W 2ndR:W 10R:W 10R:W 10R:RedH85/200R:W 2ndR:W 10R:W 10R:W 10R:RedH85/200R:W 2ndR:W 10 <td>MOV.L @ERs+,ERd</td> <td>R:W 2nd</td> <td>R:W NEXT</td> <td>Internal operation, 1 state</td> <td>R:W EA</td> <td>R:W EA+2</td> <td></td> <td></td> <td></td> <td></td>	MOV.L @ERs+,ERd	R:W 2nd	R:W NEXT	Internal operation, 1 state	R:W EA	R:W EA+2				
a:3.2.ERdE:W.2ndE:W.3ndE:W.8ndE:W.NEXTE:W.8ndE:W.8nds:@EFdE:W.2ndE:W.NEXTW:WEAW:WEAE:W.8ndE:W.8nds:@(:16, Erd)E:W.2ndE:W.8ndE:W.8ndW:WEAM:WEAE:W.8nds:@(:16, Erd)E:W.2ndE:W.8ndE:W.8ndW:WEAM:WEAE:W.8nds:@(:16, Erd)E:W.2ndE:W.8ndE:W.8ndW:WEAM:WEAE:W.8nds:@(:16, Erd)E:W.2ndE:W.8ndE:W.8ndW:WEAM:WEAM:WEAs:@(:16, Erd)E:W.2ndE:W.8ndE:W.8ndW:WEAM:WEAM:WEAs:@(:16, Erd)E:W.2ndE:W.8ndE:W.8ndW:WEAM:WEAM:WEAs:@(:16, Erd)E:W.2ndE:W.8ndM:WEAM:WEAM:WEAM:WEAs:@(:16, Erd)E:W.2ndE:W.8ndM:WEAM:WEAM:WEAM:WEAs:@(:16, Erd)E:W.2ndE:W.8ndM:WEAM:WEAM:WEAM:WEAs:@(:16, Erd)E:W.2ndE:W.8ndM:WEAM:WEAM:WEAM:WEAs:@(:16, Erd)E:W.2ndE:W.8ndM:WEAM:WEAM:WEAM:WEAs:@(:16, Erd)E:W.2ndE:W.8ndM:WEAM:WEAM:WEAM:WEAs:@(:16, Erd)E:W.2ndE:W.8ndM:WEAM:WEAM:WEAM:WEAs:@(:16, Erd)E:W.2ndE:W.8ndM:WEAM:WEAM:WEAM:WEAs:@(:16, Erd)E:W.8200E:W.8200E:W.NEAM:B.7M:WEAM:WEA	MOV.L @aa:16,ERd	R:W 2nd	R:W 3rd	R:W NEXT	R:W EA	R:W EA+2				
s, @ERd          s, @ERd         E:W Dad         E:W NEXT         W:W EA+2         M:W EA+2         M:W EA+2           s, @(:15, Erd)         R:W 2nd         R:W NEXT         W:W EA         W:W EA+2         W:W EA+2           s, @(:15, Erd)         R:W 2nd         R:W NEXT         W:W EA         W:W EA+2         W:W EA           s, @(:15, Erd)         R:W 2nd         R:W NEXT         W:W EA         W:W EA         W:W EA           s, @ =:16         R:W 2nd         R:W 3rd         R:W NEXT         W:W EA         W:W EA         W:W EA           s, @ =:16         R:W 2nd         R:W 2nd         R:W 14         R:W NEXT         W:W EA         W:W EA           8:@ a:16         R:W 2nd         R:W NEXT         W:W EA         W:W EA         W:W EA           8:@ a:16         R:W 2nd         R:W NEXT         W:W EA         W:W EA         W:W EA           8:@ a:16         R:W 2nd         R:W NEXT         W:W EA         W:W EA         W:W EA           8:@ a:16         R:W NEXT         W:W EA         W:W EA         W:W EA         W:W EA           8:@ a:16         R:W NEXT         W:W EA         W:W EA         W:W EA         W:W EA           8:@ a:16         R:W NEXT         W:W EA         W:W EA	MOV.L @aa:32,ERd	R:W 2nd	R:W 3rd	R:W 4th	R:W NEXT	R:W EA	R:W EA+2			
s,@(d:16,ERd) R.W.2nd R.W.Brd R.W.NEXT W.W.EA, W.W.EA, W.W.EA, R.W.NEXT W.W.EA, S.@(d:32,ERd) R.W.2nd R.W.NEXT R.W.4th R.W.6th R.W.NEXT W.W.EA, S.@aa:16 R.W.2nd R.W.NEXT R.W.4th R.W.EA W.W.EA, W.W.EA, W.W.EA, R.W.NEXT R.W.2nd R.W.2nd R.W.NEXT W.W.EA, W.W.EA, W.W.EA, R.W.NEXT R.W.2nd R.W.NEXT W.W.EA, W.W.EA, M. R.W.2nd R.W.NEXT R.W.4th R.W.NEXT W.W.EA, W.W.EA, S.@aa:16 R.W.2nd R.W.NEXT R.W.4th R.W.NEXT W.W.EA, W.W.EA, S.@aa:16 R.W.2nd R.W.NEXT R.W.4th R.W.NEXT W.W.EA, M. W.W.EA, S.@aa:16 R.W.2nd R.W.NEXT R.W.4th R.W.NEXT W.W.EA, M. W.W.EA, S.@aa:16 R.W.2nd R.W.NEXT R.W.4th R.W.NEXT W.W.EA, M. W.W.EA, S.@aa:16 R.W.Znd R.W.NEXT W.W.EA, M. W.W.EA, M. W.W.EA, S.@aa:16 R.W.Znd R.W.NEXT W.W.EA, M. M.W.EA, S.@aa:16 R.W.NEXT M. R.W.NEXT R.W.4th R.W.NEXT M. R.W.NEXT R.W.4th R.W.NEXT R.W.4th R.W.NEXT R.W.4th R.W.NEXT M. W.W.EA, M. W.W.EA, S.@aa:16 R.W.Znd R.W.NEXT M. R.W.NEXT M. W.W.EA, M. W.W.EA, S.@aa:16 R.W.Znd R.W.NEXT M. R.W.NEXT M. M.W.EA, S.@aa:16 R.W.NEXT M. R.W. R.M. R.M. R.M. R.M. R.M. R.M. R	MOV.L ERs,@ERd	R:W 2nd	R:W NEXT	W:W EA	W:W EA+2					
s,@(d:32,ERd) R:W 2nd R:W 3rd R:W 4th R:W 6th R:W NEXT W:W EA s,@=ERd R:W 2nd R:W NEXT Internal operation, W:W EA s,@aa:16 R:W 2nd R:W NEXT Internal operation, W:W EA S,@aa:16 R:W 2nd R:W 8rX R:W NEXT W:W EA S,@aa:16 R:W 2nd R:W 8rX R:W 4th R:W NEXT W:W EA S,@aa:16 R:W 2nd R:W 3rd R:W 4th R:W NEXT W:W EA S,@aa:16 R:W 2nd R:W 3rd R:W 4th R:W NEXT W:W EA S,@aa:16 R:W 2nd R:W 4th R:W NEXT W:W EA S,@aa:16 R:W 2nd R:W 8rX R:W 4th R:W NEXT W:W EA S,@aa:16 R:W 2nd R:W 8rX R:W 4th R:W NEXT W:W EA S,@aa:16 R:W 2nd R:W 8rX R:W 4th R:W NEXT W:M EA S,@aa:16 R:W 2nd R:W 2nd R:W 8th R:W 8th R:W 2nd R:	MOV.L ERs, @ (d: 16, ERd)	R:W 2nd	R:W 3rd	R:W NEXT	W:W EA	W:W EA+2				
s.@=Erd R:W 2nd R:W NEXT Internal operation, W:W EA, 2 8.@aa:16 R:W 2nd R:W 3rd R:W NEXT W:W EA WINE A W:W EA, 2 8.@aa:16 R:W 2nd R:W 2nd R:W NEXT W:W EA WINE A W:W EA, 2 8.@aa:16 R:W 2nd R:W 2nd R:W NEXT W:W EA WINE A W:W EA, 2 8.@aa:16 R:W 2nd R:W 2nd R:W NEXT R:W "E A WINE A W:W EA, 2 8.@aa:16 R:W 2nd R:W 2nd R:W NEXT R:W "E A MILL A WINE A WINE A MILL A	MOV.L ERs, @ (d: 32, ERd)	R:W 2nd	R:W 3rd	R:W 4th	R:W 5th	R:W NEXT	W:W EA	W:W EA+2		
s.@aa:16         R:W Znd         R:W NEXT         W:W EAA           s.@aa:16, Rd         R:W Znd         R:W NEXT         W:W EAA           @aa:16, Rd         R:W Znd         R:W NEXT         W:W EAA           83, @aa:16         R:W Znd         R:W NEXT         W:W EAA           Rs, Rd         HSS/2000         R:W NEXT         Internal operation. 2 states <sup>-9</sup> Rs, Rd         HSS/2000         R:W NEXT         Internal operation. 1 states           Rs, Rd         HSS/2000         R:W NEXT         Internal operation. 1 states           Rs, Rd         HSS/2000         R:W NEXT         Internal operation. 1 states           Rs, Rd         HSS/2000         R:W NEXT         Internal operation. 1 states           Rs, Rd         HSS/2000         R:W NEXT         Internal operation. 1 states           Rs, Rd         HSS/2000         R:W NEXT         Internal operation. 1 states           HSS/2000         R:W NEXT         Internal operation. 1 states         Internal operation. 1 states           HSS/200	MOV.L ERs,@-ERd	R:W 2nd	R:W NEXT	Internal operation, 1 state	W:W EA	W:W EA+2				
s.@aa:32         R:W Znd         R:W And         R:W WEXT         R:W WEXT         W:W EX           @aa:16,Rd         R:W Znd         R:W NEXT         R:W WEXT         W:W EX         W:W EA           8.a.16,Rd         R:W Znd         R:W NEXT         R:W WEXT         W:W EA         M:W EA           8.a.16,Rd         R:W Znd         R:W NEXT         W:B "4         EA         M:W EA           8.a.16         R:W Znd         R:W NEXT         Internal operation, 2 states"         M:W EA           Rs,Rd         H85/200         R:W NEXT         Internal operation, 1 states         M:W EA           Rs,Rd         H85/200         R:W NEXT         Internal operation, 1 states         M:W EA           Rs,Rd         H85/200         R:W NEXT         Internal operation, 1 states         M:W EA           Rs,Fd         H85/200         R:W NEXT         Internal operation, 1 states         M:W EA           Rs,Fd         H85/200         R:W NEXT         Internal operation, 1 states         M:W EA           Rs,Fd         H85/200         R:W NEXT         Internal operation, 1 states         M:W EA           Rs,Fd         H85/200         R:W NEXT         Internal operation, 1 states         M:W EA           Rs,Fd         H85/200	MOV.L ERs,@aa:16	R:W 2nd	R:W 3rd	R:W NEXT	W:W EA	W:W EA+2				
@aa:16,Rd         R:W 2nd         R:W NEXT           78,@aa:16         R:W 2nd         R:W NEXT           Rs,Rd         H85/2600         R:W 2nd         R:W NEXT           Rs,Edd         H85/2000         R:W 2nd         R:W NEXT           Rs,Edd         H85/2000         R:W NEXT         R:W NEXT           Rs,Edd         H85/2000         R:W NEXT         R:W NEXT           Rs,Rd         H85/2000         R:W NEXT         Internal operation.           Rs,Erd         H85/2000         R:W NEXT         Internal operation.           Rs,Frd         H85/2000         R:W NEXT         Internal operation.           Rs,Frd         H85/2000         R:W NEXT         Internal operation.           Rs,Frd         H85/2000         R:W NEXT         Internal operation.           d         R:W NEXT         Internal operation.         R:W NEXT           d         R:W NEXT         Internal operation.         R:W NEXT         Internal operation.           d         R:W NEXT         R:W NEXT         R:W NEXT         R:W NEXT         R:W NEXT	MOV.L ERs,@aa:32	R:W 2nd	R:W 3rd	R:W 4th	R:W NEXT	W:W EA	W:W EA+2			
Rs, Rad         HBS/2600         R::W 2nd         R::W NEXT           Rs, Fid         HBS/2600         R::W NEXT         R::W NEXT           Rs, Edd         HBS/2600         R::W NEXT         R::W NEXT           Rs, Edd         HBS/2000         R::W NEXT         R::W NEXT           Rs, Fid         HBS/2000         R::W NEXT         R::W NEXT           Rs, Rd         HBS/2000         R::W NEXT         Internal operation.           Rs, FRd         HBS/2000         R::W NEXT         Internal operation.           Rs, FRd         HBS/2000         R::W NEXT         Internal operation.           d         R::W NEXT         Internal operation.         R::W NEXT           d         R::W NEXT         Internal operation.         R::W NEXT           d         R::W NEXT         Internal operation.         R::W NEXT	MOVFPE @aa:16,Rd	R:W 2nd	R:W NEXT	R:W *4 EA						
Rs,Rd         H8S/2600         R:W NEXT           Rs,Erd         H8S/2000         R:W NEXT           Rs,Erd         H8S/2000         R:W NEXT           Rs,Rd         H8S/2000         R:W NEXT           Rs,Fed         H8S/2000         R:W NEXT           H8S/2000         R:W NEXT         Internal operation.           Rs,Erd         H8S/2000         R:W NEXT           d         R:W NEXT         Internal operation.           d         R:W NEXT         Internal operation.           d         R:W NEXT         Internal operation.           R:W NEXT         Internal operation.           d         R:W NEXT         Internal operation.	MOVTPE Rs,@aa:16	R:W 2nd	R:W NEXT	W:B *4 EA						
HBS/2000         R::W NEXT           Rs.ERd         HBS/2000         R::W NEXT           Rs.Rd         HBS/2600         R::W NEXT           ABS/2000         R::W NEXT         Internal operation.           Rs.Rd         HBS/2000         R::W NEXT           Rs.Rd         HBS/2000         R::W NEXT           Rs.Rd         HBS/2000         R::W NEXT           Rs.Etd         HBS/2000         R::W NEXT           HSS/2000         R::W NEXT         Internal operation.           Rs.Etd         HBS/2000         R::W NEXT           d         R::W NEXT         Internal operation.		R:W 2nd	R:W NEXT	Internal operation,	2 states*9					
Rs.ERd         HBS/2600         R::W NEXT           HBS/2000         R::W NEXT           Rs.Rd         HBS/2000         R::W NEXT           Rs.Rd         HBS/2000         R::W NEXT           Rs.Rd         HBS/2000         R::W NEXT           Rs.Edd         HBS/2000         R::W NEXT           Rs.Edd         HBS/2000         R::W NEXT           Rs.Edd         HBS/2000         R::W NEXT           Rs.W NEXT         Internal operation.           d         R::W NEXT         Internal operation.           d         R::W NEXT         Internal operation.           d         R::W NEXT         Internal operation.           d         R::W NEXT         Internal operation.           d         R::W NEXT         Internal operation.	H8S/2000	R:W 2nd	R:W NEXT	Internal operation,	11 states					
H8S/2000         R:W NEXT           Rs,Rd         H8S/2600         R:W NEXT           Internal operation.         H8S/2600         R:W NEXT           H8S/2600         R:W NEXT         Internal operation.           Rs,Erd         H8S/2600         R:W NEXT         Internal operation.           Rs,Erd         H8S/2000         R:W NEXT         Internal operation.           Rs,Erd         H8S/2000         R:W NEXT         Internal operation.           d         R:W NEXT         Internal operation.           d         R:W NEXT         R:M NEXT         Internal operation.           d         R:W NEXT         R:W NEXT         R:M NEXT           d         R:W NEXT         R:W NEXT         R:M NEXT		R:W 2nd	R:W NEXT	Internal operation,	3 states*9					
Rs, Rd HBS/2600 R:W NEXT Rs, Erd HBS/2000 R:W NEXT Rs, Erd HBS/2000 R:W NEXT HBS/2000 R:W NEXT HBS/2000 R:W NEXT d R:W NEXT d R:W NEXT R:W NEXT	H8S/2000	R:W 2nd	R:W NEXT	Internal operation,	19 states					
H8S/2000         R:W NEXT           Rs,ERd         H8S/2600         R:W NEXT           H8S/2000         R:W NEXT           H8S/2000         R:W NEXT           A         R:W NEXT           d         R:W NEXT           R:W NEXT         R:W NEXT		R:W NEXT	Internal operation	, 2 states*9						
Rs,ERd         Hss/2600         R:W NEXT           Hss/2000         R:W NEXT           B         R:W NEXT           A         R:W NEXT           A         R:W NEXT           A         R:W NEXT	H8S/2000	R:W NEXT	Internal operation	, 11 states						
H8S/2000         R.W NEXT           3         R.W NEXT           4         R.W NEXT           8         R.W NEXT           9         R.W NEXT		R:W NEXT	Internal operation	, 3 states*9						
7.9	H8S/2000	R:W NEXT	Internal operation	, 19 states						
υ	NEG.B Rd	R:W NEXT								
g	NEG.W Rd	R:W NEXT								
	NEG.L ERd	R:W NEXT								
	NOP	R:W NEXT								
	NOT.B Rd	R:W NEXT								

Instruction	1	2	e	4	5	9	7	8	6
NOT.W Rd	R:W NEXT								
NOT.L ERd	R:W NEXT								
OR.B #xx:8,Rd	R:W NEXT								
OR.B Rs,Rd	R:W NEXT								
OR.W #xx:16,Rd	R:W 2nd	R:W NEXT							
OR.W Rs,Rd	R:W NEXT								
OR.L #xx:32,ERd	R:W 2nd	R:W 3rd	R:W NEXT						
OR.L ERS, ERd	R:W 2nd	R:W NEXT							
ORC #xx:8,CCR	R:W NEXT								
ORC #xx:8,EXR	R:W 2nd	R:W NEXT							
POP.W Rn	R:W NEXT	Internal operation, R:W EA 1 state	R:W EA						
POP.L ERn	R:W 2nd	R:W NEXT	Internal operation, R:W EA 1 state	R:W EA	R:W EA+2				
PUSH.W Rn	R:W NEXT	Internal operation, W:W EA 1 state	W:W EA						
PUSH.L ERn	R:W 2nd	R:W NEXT	Internal operation W:W EA	W:W EA	W:W EA+2				
ROTL.B Rd	R:W NEXT								
ROTL.B #2,Rd	R:W NEXT								
ROTL.W Rd	R:W NEXT								
ROTL.W #2,Rd	R:W NEXT								
ROTL.L ERd	R:W NEXT								
ROTL.L #2,ERd	R:W NEXT								
ROTR.B Rd	R:W NEXT								
ROTR.B #2,Rd	R:W NEXT								
ROTR.W Rd	R:W NEXT								
ROTR.W #2,Rd	R:W NEXT								
ROTR.L ERd	R:W NEXT								
ROTR.L #2,ERd	R:W NEXT								
ROTXL.B Rd	R:W NEXT								
ROTXL.B #2,Rd	R:W NEXT								
ROTXL.W Rd	R:W NEXT								
ROTXL.W #2,Rd	R:W NEXT								
ROTXL.L ERd	R:W NEXT								
ROTXL.L #2,ERd	R:W NEXT								
ROTXR.B Rd	R:W NEXT								

CUTCR MB2         R.W.WET	Instruction		-	2	3	4	5	9	7	8	6
Image         <	ROTXR.B #2,Rd		R:W NEXT								
Rd         Ewnext         Ewnext <td>ROTXR.W Rd</td> <td>_</td> <td>R:W NEXT</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td>	ROTXR.W Rd	_	R:W NEXT								
Image         <	ROTXR.W #2,Rd	_	R:W NEXT								
ERd         ERd         Erd         Mextr         R:w hextr         R:w stack (Ext)         R:w stack (L)         R:w stack (L) <td>ROTXR.L ERd</td> <td>_</td> <td>R:W NEXT</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td>	ROTXR.L ERd	_	R:W NEXT								
Image: Normal state         R:W stack (EXN)         R:W stack (H)         R:W stack (H) <th< td=""><td>ROTXR.L #2,ERd</td><td>_</td><td>R:W NEXT</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></th<>	ROTXR.L #2,ERd	_	R:W NEXT								
International         Result of the stack (L)         Result of the stack (L)         Internation (L)           Advanced         R:W NEXT         R:W stack (L)         R:W stack (L)           Advanced         R:W NEXT         R:W stack (L)         R:W stack (L)           Advanced         R:W NEXT         R:W NEXT         R:W stack (L)           Advanced         R:W NEXT         R:W NEXT         R:W Stack (L)           R:W NEXT         R:W NEXT         R:W NEXT         R:W Stack (L)           R:W NEXT         R:W NEXT         R:W NEXT         R:W NEXT           R:W NEXT         R:W NEXT         R:W NEXT         R:W NEXT <td>RTE</td> <td></td> <td>R:W NEXT</td> <td>R:W stack (EXR)</td> <td></td> <td>ck (L)</td> <td>Internal operation, F 1 state</td> <td>₹:W <sup>#5</sup></td> <td></td> <td></td> <td></td>	RTE		R:W NEXT	R:W stack (EXR)		ck (L)	Internal operation, F 1 state	₹:W <sup>#5</sup>			
			R:W NEXT		Internal operation, 1 state	R:W *5					
Image: Simple state	*	Advanced	R:W NEXT	R:W stack (H)		Internal operation, 1 state	R:W *5				
J     R:W NEXT       d     R:W NEXT       kd     R:W NEXT	SHAL.B Rd		R:W NEXT								
R:W NEXT       R:W NEXT	SHAL.B #2,Rd		R:W NEXT								
d R:W NEXT RW NEXT RW NEXT R:W	SHAL.W Rd	-	R:W NEXT								
R::W NEXT       R::W NEXT	SHAL.W #2,Rd		R:W NEXT								
kd     R:W NEXT       d     R:W NEXT       d     R:W NEXT       kd     R:W NEXT	SHAL.L ERd	_	R:W NEXT								
d     R:W NEXT       d     R:W NEXT       kd     R:W NEXT       rd     R:W NEXT       rd     R:W NEXT       rd     R:W NEXT       d     R:W NEXT	SHAL.L #2,ERd	_	R:W NEXT								
d R:W NEXT R:W NEXT R:W NEXT R:W NEXT R:W NEXT R:W NEXT R:W NEXT R:W NEXT R:W NEXT R:W NEXT d R:W NEXT d R:W NEXT d R:W NEXT d R:W NEXT d R:W NEXT d R:W NEXT R:W NEX	SHAR.B Rd	-	R:W NEXT								
kd R:W NEXT R:W NEXT R:W NEXT R:W NEXT R:W NEXT R:W NEXT d R:W NEXT KU NEXT d R:W NEXT d R:W NEXT d R:W NEXT d R:W NEXT d R:W NEXT R:W NEX	SHAR.B #2,Rd	-	R:W NEXT								
kd R:W NEXT Rd R:W NEXT R:W NEXT R:W NEXT R:W NEXT d R:W NEXT KU NEXT d R:W NEXT d R:W NEXT d R:W NEXT d R:W NEXT d R:W NEXT R:W	SHAR.W Rd	-	R:W NEXT								
R:W NEXT       R:W NEXT	SHAR.W #2,Rd	-	R:W NEXT								
Ad R:W NEXT A R:W	SHAR.L ERd	-	R:W NEXT								
d R:W NEXT d R:W NEXT d R:W NEXT R:W NEXT R:W NEXT d R:W NEXT d R:W NEXT d R:W NEXT d R:W NEXT d R:W NEXT d R:W NEXT R:W NEXT R:W NEXT R:W NEXT R:W NEXT R:W NEXT R:W NEXT	SHAR.L #2,ERd		R:W NEXT								
d R:W NEXT d R:W NEXT d R:W NEXT R:W NEXT R:W NEXT d R:W NEXT d R:W NEXT d R:W NEXT R:W NEXT R:W NEXT R:W NEXT R:W NEXT R:W NEXT R:W NEXT R:W NEXT R:W NEXT R:W NEXT	SHLL.B Rd	-	R:W NEXT								
d R:W NEXT d R:W NEXT R:W NEXT R:W NEXT d R:W NEXT d R:W NEXT d R:W NEXT R:W NEXT R:W NEXT R:W NEXT R:W NEXT R:W NEXT R:W NEXT R:W NEXT R:W NEXT	SHLL.B #2,Rd	-	R:W NEXT								
d R:W NEXT d R:W NEXT d R:W NEXT d R:W NEXT d R:W NEXT d R:W NEXT d R:W NEXT d R:W NEXT d R:W NEXT R:W NEXT R:W NEXT	SHLL.W Rd	-	R:W NEXT								
R:W NEXT R:W NEXT	SHLL.W #2,Rd	-	R:W NEXT								
kd R:W NEXT d R:W NEXT d R:W NEXT d R:W NEXT d R:W NEXT R:W NEXT R:W NEXT R:W NEXT R:W NEXT R:W NEXT R:W NEXT	SHLL.L ERd	_	R:W NEXT								
d         R:W NEXT           d         R:W NEXT           d         R:W NEXT           d         R:W NEXT           d         R:W NEXT           d         R:W NEXT           d         R:W NEXT           R         R:W NEXT           R         R:W NEXT           R         R:W NEXT	SHLL.L #2,ERd	-	R:W NEXT								
d R:W NEXT d R:W NEXT d R:W NEXT R:W NEXT d R:W NEXT R:W NEXT R:W NEXT R:W NEXT	SHLR.B Rd	-	R:W NEXT								
d R:W NEXT d R:W NEXT R:W NEXT d R:W NEXT R:W NEXT R:W NEXT R:W NEXT	SHLR.B #2,Rd		R:W NEXT								
d R:W NEXT R:W NEXT R:W NEXT R:W NEXT R:W NEXT R:W NEXT	SHLR.W Rd	_	R:W NEXT								
R:W NEXT R:W NEXT R:W NEXT R:W NEXT R:W NEXT	SHLR.W #2,Rd		R:W NEXT								
d R:W NEXT R:W NEXT R:W NEXT R:W NEXT	SHLR.L ERd	-	R:W NEXT								
R:W NEXT R:W NEXT	SHLR.L #2,ERd	-	R:W NEXT								
	SLEEP		R:W NEXT	Internal operation, 1 state							
	STC CCR,Rd	_	R:W NEXT								

Instruction	٢	2	3	4	5	9	7	80	6
STC EXR,Rd	R:W NEXT								
STC CCR, @ERd	R:W 2nd	R:W NEXT	W:W EA						
STC EXR,@ERd	R:W 2nd	R:W NEXT	W:W EA						
STC CCR, @(d:16,ERd)	R:W 2nd	R:W 3rd	R:W NEXT	W:W EA					
STC EXR,@(d:16,ERd)	R:W 2nd	R:W 3rd	R:W NEXT	W:W EA					
STC CCR, @(d:32,ERd)	R:W 2nd		R:W 4th	R:W 5th	R:W NEXT	W:WEA			
STC EXR,@(d:32,ERd)	R:W 2nd	R:W 3rd	R:W 4th	R:W 5th	R:W NEXT	W:WEA			
STC CCR, @-ERd	R:W 2nd	R:W NEXT	Internal operation, W:W EA 1 state	W:W EA					
STC EXR,@-ERd	R:W 2nd	R:W NEXT	Internal operation, W:W EA 1 state	W:W EA					
STC CCR, @aa:16	R:W 2nd	R:W 3rd	R:W NEXT	W:W EA					
STC EXR,@aa:16	R:W 2nd	R:W 3rd	R:W NEXT	W:W EA					
STC CCR, @aa:32	R:W 2nd	R:W 3rd	R:W 4th	R:W NEXT	W:W EA				
STC EXR,@aa:32	R:W 2nd	R:W 3rd	R:W 4th	R:W NEXT	W:W EA				
STM.L(ERn-ERn+1),@-SP	R:W 2nd	R:W NEXT	Internal operation, W:W stack (H)*3 1 state	W:W stack (H)*3	W:W stack (L)*3				
STM.L(ERn-ERn+2),@-SP	R:W 2nd	R:W NEXT	Internal operation, W:W stack (H)*3 1 state	W:W stack (H)*3	W:W stack (L)*3				
STM.L(ERn-ERn+3),@-SP	R:W 2nd	R:W NEXT	Internal operation, W:W stack (H)*3 W:W stack (L)*3 tate	W:W stack (H)*3	W:W stack (L)*3				
STMAC MACH, ERd*11	R:W NEXT	6*		← Repeated n times*3	n times $^{*3} \longrightarrow$				
STMAC MACL, ERd*11	R:W NEXT	6*							
SUB.B Rs,Rd	R:W NEXT								
SUB.W #xx:16,Rd	R:W 2nd	R:W NEXT							
SUB.W RS,Rd	R:W NEXT								
SUB.L #xx:32,ERd	R:W 2nd	R:W 3rd	R:W NEXT						
SUB.L ERS, ERd	R:W NEXT								
SUBS #1/2/4,ERd	R:W NEXT								
SUBX #xx:8,Rd	R:W NEXT								
SUBX Rs,Rd	R:W NEXT								
TAS @ERd*10	R:W 2nd	R:W NEXT	R:B EA	W:B EA					
TRAPA #x:2 Normal	R:W NEXT	Internal operation, W:W stack (L) 1 state	W:W stack (L)	W:W stack (H)	W:W stack (EXR) R:W VEC	R:W VEC	Internal operation, R:W *8 1 state	R:W *8	
Advanceo	Advanced R:W NEXT	Internal operation, W:W stack (L) 1 state	W:W stack (L)	W:W stack (H)	W:W stack (EXR) R:W VEC	R:W VEC	R:W VEC+2	Internal operation, R:W *8 1 state	R:W *8
XOR.B #xx8,Rd	R:W NEXT								

Instruction	Ę	-	2	3	4	5	9	7	8	6
XOR.B Rs,Rd		R:W NEXT								
XOR.W #xx:16,Rd		R:W 2nd	R:W NEXT							
XOR.W Rs,Rd		R:W NEXT								
XOR.L #xx:32,ERd		R:W 2nd	R:W 3rd	R:W NEXT						
XOR.L ERS, ERd		R:W 2nd	R:W NEXT							
XORC #xx:8,CCR		R:W NEXT								
XORC #xx:8,EXR		R:W 2nd	R:W NEXT							
Reset exception handling	Normal	R:W VEC	Internal operation, R:W *6 1 state	R:W *6						
	Advanced	Advanced R:W VEC	R:W VEC+2	Internal operation, R:W *6 1 state	R:W *6					
Interrupt exception handling	Normal	R:W *7	Internal operation, W:W stack (L) 1 state		W:W stack (H)	W:W stack (EXR) R:W VEC	R:W VEC	Internal operation, R:W *8 1 state	R:W *8	
	Advanced R:W *7	R:W *7	Internal operation, W:W stack (L) 1 state		W:W stack (H)	W:W stack (EXR) R:W:M VEC	R:W:M VEC	R:W VEC+2	Internal operation, R:W *8 1 state	R:W *8

Notes: 1. EAs is the contents of ER5. EAd is the contents of ER6.

- EAs is the contents of ER5. EAd is the contents of ER6. Both registers are incremented by 1 after execution of the instruction. n is the initial value of R4L or R4. If n = 0, these bus cycles are not executed. сi
- Repeated two times to save or restore two registers, three times for three registers, or four times for four registers. ы. С
- For the number of states required for byte-size read or write, refer to the relevant microcontroller hardware manual. 4.
- 5. Start address after return.
- Start address of the program.
   Prefetch address, equal to two
- Prefetch address, equal to two plus the PC value pushed onto the stack. In recovery from sleep mode or software standby mode the read operation is replaced by an internal operation.
  - 8. Start address of the interrupt-handling routine.
- An internal operation may require between 0 and 3 additional states, depending on the preceding instruction. . о
- 10. Only register ER0, ER1, ER4, or ER5 should be used when using the TAS instruction.
- 11. These instructions are supported by the H8S/2600 CPU only.

## 2.8 Condition Code Modification

This section indicates the effect of each CPU instruction on the condition code. The notation used in the table is defined below.

m = (	31 for longword operands
ł	15 for word operands
l	7 for byte operands
Si	The i-th bit of the source operand
Di	The i-th bit of the destination operand
Ri	The i-th bit of the result
Dn	The specified bit in the destination operand
_	Not affected
$\updownarrow$	Modified according to the result of the instruction (see definition)
0	Always cleared to 0
1	Always set to 1
*	Undetermined (no guaranteed value)
Z'	Z flag before instruction execution
C'	C flag before instruction execution



Instruction	н	Ν	z	V	С	Definition
ADD	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$H = Sm-4 \cdot Dm-4 + Dm-4 \cdot \overline{Rm-4} + Sm-4 \cdot \overline{Rm-4}$
						N = Rm
						$Z = \overline{Rm} \cdot \overline{Rm-1} \cdot \dots \cdot \overline{R0}$
						$V = Sm \cdot Dm \cdot \overline{Rm} + \overline{Sm} \cdot \overline{Dm} \cdot Rm$
						$C = Sm \cdot Dm + Dm \cdot \overline{Rm} + Sm \cdot \overline{Rm}$
ADDS	_	_	_	_	_	
ADDX	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$H = Sm-4 \cdot Dm-4 + Dm-4 \cdot \overline{Rm-4} + Sm-4 \cdot \overline{Rm-4}$
						N = Rm
						$Z = Z' \cdot \overline{Rm} \cdot \dots \cdot \overline{R0}$
						$V = Sm \cdot Dm \cdot \overline{Rm} + \overline{Sm} \cdot \overline{Dm} \cdot Rm$
						$C = Sm \cdot Dm + Dm \cdot \overline{Rm} + Sm \cdot \overline{Rm}$
AND	_	$\updownarrow$	$\updownarrow$	0	—	N = Rm
						$Z = \overline{Rm} \cdot \overline{Rm} - 1 \cdot \dots \cdot \overline{R0}$
ANDC	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	Stores the corresponding bits of the result.
						No flags change when the operand is EXR.
BAND	_	—	—	—	$\updownarrow$	$C = C' \cdot Dn$
Bcc	—	—	—	—	_	
BCLR	_	—	—	—	_	
BIAND	_	—	—	—	$\updownarrow$	$C = C' \cdot \overline{Dn}$
BILD	_	—	—	—	$\updownarrow$	$C = \overline{Dn}$
BIOR	_	_	_	—	$\updownarrow$	$C = C' + \overline{Dn}$
BIST	_	—	—	—	—	
BIXOR	_	_	_	_	$\updownarrow$	$C = C' \cdot Dn + \overline{C'} \cdot \overline{Dn}$
BLD	_	_	_	_	$\updownarrow$	C = Dn
BNOT	_	—	—	—	_	
BOR	_	_	_	_	$\updownarrow$	C = C' + Dn
BSET	_	_	—	_	_	
BSR	_	_	_	—	_	
BST	_	_	_	—	_	
BTST	_	—	$\updownarrow$	—	_	Z = Dn
BXOR	_	—	—	_	$\updownarrow$	$C = C' \cdot \overline{Dn} + \overline{C'} \cdot Dn$
CLRMAC*	_	_	_	_	_	
CMP	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$H = Sm-4 \cdot \overline{Dm-4} + \overline{Dm-4} \cdot Rm-4 + Sm-4 \cdot Rm-4$
						N = Rm
						$Z = \overline{Rm} \cdot \overline{Rm-1} \cdot \cdot \overline{R0}$
						$V = \overline{Sm} \cdot Dm \cdot \overline{Rm} + Sm \cdot \overline{Dm} \cdot Rm$
						$C = Sm \cdot \overline{Dm} + \overline{Dm} \cdot Rm + Sm \cdot Rm$

#### Table 2.7 Condition Code Modification

Rev. 4.00 Feb 24, 2006 page 305 of 322 REJ09B0139-0400

Section 2	Instruction	Descriptions
-----------	-------------	--------------

Instruction	н	Ν	z	v	С	Definition
DAA	*	$\updownarrow$	$\updownarrow$	*	$\updownarrow$	N = Rm
						$Z = \overline{Rm} \cdot \overline{Rm-1} \cdot \dots \cdot \overline{R0}$
						C: decimal arithmetic carry
DAS	*	$\updownarrow$	$\updownarrow$	*	$\updownarrow$	N = Rm
						$Z = \overline{Rm} \cdot \overline{Rm-1} \cdot \dots \cdot \overline{R0}$
						C: decimal arithmetic borrow
DEC	_	$\updownarrow$	$\updownarrow$	$\updownarrow$	_	N = Rm
						$Z = \overline{Rm} \cdot \overline{Rm-1} \cdot \dots \cdot \overline{R0}$
						$V = Dm \cdot \overline{Rm}$
DIVXS	_	$\updownarrow$	$\updownarrow$	_	_	$N = Sm \cdot \overline{Dm} + \overline{Sm} \cdot Dm$
						$Z = \overline{Sm} \cdot \overline{Sm-1} \cdot \dots \cdot \overline{S0}$
DIVXU	_	$\updownarrow$	$\updownarrow$	_	_	N = Sm
						$Z = \overline{Sm} \cdot \overline{Sm-1} \cdot \dots \cdot \overline{S0}$
EEPMOV	_	_	_	_	_	
EXTS	_	\$	\$	0	_	N = Rm
						$Z = \overline{Rm} \cdot \overline{Rm-1} \cdot \dots \cdot \overline{R0}$
EXTU	_	0	\$	0	_	$Z = \overline{Rm} \cdot \overline{Rm-1} \cdot \dots \cdot \overline{R0}$
INC	_	$\updownarrow$	$\updownarrow$	$\updownarrow$	_	N = Rm
						$Z = \overline{Rm} \cdot \overline{Rm-1} \cdot \dots \cdot \overline{R0}$
						$V = \overline{Dm} \cdot Rm$
JMP	_	_	_	_	_	
JSR	_	—	_	_	_	
LDC	$\uparrow$	\$	$\updownarrow$	$\updownarrow$	$\updownarrow$	Stores the corresponding bits of the result.
						No flags change when the operand is EXR.
LDM	_	_	_	_	_	
LDMAC*	_	_	_	_	_	
MAC*	_	_	_	_	_	
MOV		\$	\$	0		N = Rm
						$Z = \overline{Rm} \cdot \overline{Rm-1} \cdot \dots \cdot \overline{R0}$
MOVFPE	_	$\updownarrow$	$\updownarrow$	0		N = Rm
						$Z = \overline{Rm} \cdot \overline{Rm-1} \cdot \dots \cdot \overline{R0}$
MOVTPE	_	$\updownarrow$	$\updownarrow$	0		N = Rm
						$Z = \overline{Rm} \cdot \overline{Rm-1} \cdot \dots \cdot \overline{R0}$
MULXS	_	\$	\$	_	_	N = R2m
						$Z = \overline{R2m} \cdot \overline{R2m-1} \cdot \dots \cdot \overline{R0}$
						$L = H_{2}(II) \cdot H_{2}(I) - I \cdot \dots \cdot H_{V}$

Instruction	Н	Ν	z	v	с	Definition
MULXU	—	—	—	—	—	
NEG	$\uparrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	H = Dm–4 + Rm–4
						N = Rm
						$Z = \overline{Rm} \cdot \overline{Rm} - 1 \cdot \dots \cdot \overline{R0}$
						$V = Dm \cdot Rm$
						C = Dm + Rm
NOP	_	_	_	_	_	
NOT	—	$\updownarrow$	$\updownarrow$	0	—	N = Rm
						$Z = \overline{Rm} \cdot \overline{Rm} - 1 \cdot \dots \cdot \overline{R0}$
OR	_	$\updownarrow$	$\updownarrow$	0	_	N = Rm
						$Z = \overline{Rm} \cdot \overline{Rm} - 1 \cdot \dots \cdot \overline{R0}$
ORC	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	Stores the corresponding bits of the result.
						No flags change when the operand is EXR.
POP	—	$\updownarrow$	$\updownarrow$	0	—	N = Rm
						$Z = \overline{Rm} \cdot \overline{Rm} - 1 \cdot \dots \cdot \overline{R0}$
PUSH	—	$\updownarrow$	$\updownarrow$	0	—	N = Rm
						$Z = \overline{Rm} \cdot \overline{Rm} - 1 \cdot \dots \cdot \overline{R0}$
ROTL	_	$\updownarrow$	$\updownarrow$	0	$\updownarrow$	N = Rm
						$Z = \overline{Rm} \cdot \overline{Rm} - 1 \cdot \dots \cdot \overline{R0}$
						C = Dm (1-bit shift)  or  C = Dm-1 (2-bit shift)
ROTR	—	$\updownarrow$	$\updownarrow$	0	$\updownarrow$	N = Rm
						$Z = \overline{Rm} \cdot \overline{Rm} - 1 \cdot \dots \cdot \overline{R0}$
						C = D0 (1-bit shift) or $C = D1$ (2-bit shift)
ROTXL	—	$\updownarrow$	$\updownarrow$	0	$\updownarrow$	N = Rm
						$Z = \overline{Rm} \cdot \overline{Rm} - 1 \cdot \dots \cdot \overline{R0}$
						C = Dm (1-bit shift)  or  C = Dm-1 (2-bit shift)
ROTXR	—	$\updownarrow$	$\updownarrow$	0	$\updownarrow$	N = Rm
						$Z = \overline{Rm} \cdot \overline{Rm} - 1 \cdot \dots \cdot \overline{R0}$
						C = D0 (1-bit shift) or $C = D1$ (2-bit shift)
RTE	$\uparrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	Stores the corresponding bits of the result.
RTS	_	—	—	—	—	
SHAL	—	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	N = Rm
						$Z = \overline{Rm} \cdot \overline{Rm} - 1 \cdot \dots \cdot \overline{R0}$
						$V = \overline{Dm \cdot \overline{Dm-1} + \overline{Dm} \cdot \overline{Dm-1}} $ (1-bit shift)
						$V = \overline{Dm \cdot \overline{Dm-1} \cdot \overline{Dm-2} + \overline{Dm} \cdot \overline{Dm-1} \cdot \overline{Dm-2}} $ (2-bit shift)
						C = Dm (1-bit shift)  or  C = Dm-1 (2-bit shift)

Instruction	н	Ν	z	v	С	Definition
SHAR	_	$\updownarrow$	$\updownarrow$	0	$\updownarrow$	N = Rm
						$Z = \overline{Rm} \cdot \overline{Rm-1} \cdot \dots \cdot \overline{R0}$
						C = D0 (1-bit shift) or $C = D1$ (2-bit shift)
SHLL	_	$\updownarrow$	$\updownarrow$	0	$\updownarrow$	N = Rm
						$Z = \overline{Rm} \cdot \overline{Rm-1} \cdot \dots \cdot \overline{R0}$
						C = Dm (1-bit shift) or C = Dm-1 (2-bit shift)
SHLR	_	0	\$	0	$\updownarrow$	N = Rm
						$Z = \overline{Rm} \cdot \overline{Rm-1} \cdot \dots \cdot \overline{R0}$
						C = D0 (1-bit shift) or $C = D1$ (2-bit shift)
SLEEP	_	_	_	_	_	
STC	_	_	_	_	_	
STM	_	_	_	_	_	
STMAC*	_	$\updownarrow$	$\updownarrow$	$\updownarrow$	_	N = 1 if MAC instruction resulted in negative value in MAC register
						Z = 1 if MAC instruction resulted in zero value in MAC register
						V = 1 if MAC instruction resulted in overflow
SUB	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$H = Sm-4 \cdot \overline{Dm-4} + \overline{Dm-4} \cdot Rm-4 + Sm-4 \cdot Rm-4$
						N = Rm
						$Z = \overline{Rm} \cdot \overline{Rm-1} \cdot \dots \cdot \overline{R0}$
						$V = \overline{Sm} \cdot Dm \cdot \overline{Rm} + Sm \cdot \overline{Dm} \cdot Rm$
						$C = Sm \cdot \overline{Dm} + \overline{Dm} \cdot Rm + Sm \cdot Rm$
SUBS	—	—	_	—	—	
SUBX	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$H = Sm-4 \cdot \overline{Dm-4} + \overline{Dm-4} \cdot Rm-4 + Sm-4 \cdot Rm-4$
						N = Rm
						$Z = Z' \cdot \overline{Rm} \cdot \dots \cdot \overline{R0}$
						$V = \overline{Sm} \cdot Dm \cdot \overline{Rm} + Sm \cdot \overline{Dm} \cdot Rm$
						$C = Sm \cdot \overline{Dm} + \overline{Dm} \cdot Rm + Sm \cdot Rm$
TAS	_	$\updownarrow$	$\updownarrow$	0	—	N = Dm
						$Z = \overline{Dm} \cdot \overline{Dm-1} \cdot \dots \dots \cdot \overline{D0}$
TRAPA	—	—	_	—	—	
XOR	_	$\updownarrow$	$\updownarrow$	0	_	N = Rm
						$Z = \overline{Rm} \cdot \overline{Rm} - 1 \cdot \dots \cdot \overline{R0}$
XORC	\$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	Stores the corresponding bits of the result.
						No flags change when the operand is EXR.

Note: \* These instructions are supported by the H8S/2600 CPU only.
# Section 3 Processing States

### 3.1 Overview

The CPU has five main processing states: the reset state, exception handling state, program execution state, bus-released state, and power-down state. Figure 3.1 shows a diagram of the processing states. Figure 3.2 indicates the state transitions.



#### Figure 3.1 Processing States



Figure 3.2 State Transitions

### 3.2 Reset State

When the  $\overline{\text{RES}}$  input goes low all current processing stops and the CPU enters the reset state. Reset exception handling starts when the  $\overline{\text{RES}}$  signal changes from low to high.

The reset state can also be entered by a watchdog timer overflow. For details, refer to the relevant microcontroller hardware manual.

### 3.3 Exception-Handling State

The exception-handling state is a transient state that occurs when the CPU alters the normal processing flow due to a reset, interrupt, or trap instruction. The CPU fetches a start address (vector) from the exception vector table and branches to that address.

#### 3.3.1 Types of Exception Handling and Their Priority

Exception handling is performed for traces, resets, interrupts, and trap instructions. Table 3.1 indicates the types of exception handling and their priority. Trap instruction exception handling is always accepted, in the program execution state.

Exception handling and the stack structure differ according to the interrupt control mode set in SYSCR.

Priority	Type of Exception	Detection Timing	Start of Exception Handling
High	Reset	Synchronized with clock	Exception handling starts immediately when RES changes from low to high
	Trace	End of instruction execution or end of exception-handling sequence <sup>*1</sup>	When the trace (T) bit is set to 1, the trace starts at the end of the current instruction or current exception-handling sequence
	Interrupt	End of instruction execution or end of exception-handling sequence <sup>*2</sup>	When an interrupt is requested, exception handling starts at the end of the current instruction or current exception-handling sequence
	Trap instruction	When TRAPA instruction is executed	Exception handling starts when a trap (TRAPA) instruction is executed <sup>*3</sup>

#### Table 3.1 Exception Handling Types and Priority

Notes: 1. Traces are enabled only in interrupt control modes 2 and 3. Trace exception-handling is not executed at the end of the RTE instruction.

2. Interrupts are not detected at the end of the ANDC, ORC, XORC, and LDC instructions, or immediately after reset exception handling.

3. Trap instruction exception handling is always accepted, in the program execution state.

For details on interrupt control modes, exception sources, and exception handling, refer to the relevant microcontroller hardware manual.

#### 3.3.2 Reset Exception Handling

After the  $\overline{\text{RES}}$  pin has gone low and the reset state has been entered, reset exception handling starts when  $\overline{\text{RES}}$  goes high again. When reset exception handling starts the CPU fetches a start address (vector) from the exception vector table and starts program execution from that address. All interrupts, including NMI, are disabled during reset exception handling and after it ends.

#### 3.3.3 Trace

Traces are enabled only in interrupt control modes 2 and 3. Trace mode is entered when the T bit of EXR is set to 1. When trace mode is established, trace exception handling starts at the end of each instruction.

At the end of a trace exception-handling sequence, the T bit of EXR is cleared to 0 and trace mode is cleared. Interrupt masks are not affected.

The T bit saved on the stack retains its value of 1, and when the RTE instruction is executed to return from the trace exception-handling routine, trace mode is entered again. Trace exception-handling is not executed at the end of the RTE instruction.

Trace mode is not entered in interrupt control modes 0 and 1, regardless of the state of the T bit.

#### 3.3.4 Interrupt Exception Handling and Trap Instruction Exception Handling

When interrupt or trap-instruction exception handling begins, the CPU references the stack pointer (ER7) and pushes the program counter and other control registers onto the stack. Next, the CPU alters the settings of the interrupt mask bits in the control registers. Then the CPU fetches a start address (vector) from the exception vector table and execution branches to that address.

Figure 3.3 shows the stack after exception handling ends, for the case of interrupt mode 1 in advanced mode.

#### 3.3.5 Usage Notes

#### (1) Conflict between Interrupt Generation and Disabling

When an interrupt enable bit is cleared to 0 to disable interrupts, the disabling becomes effective after execution of the instruction.

When an interrupt enable bit is cleared to 0 by an instruction such as BCLR or MOV, and if an interrupt is generated during execution of the instruction, the interrupt concerned will still be

enabled on completion of the instruction, and so interrupt exception handling for that interrupt will be executed on completion of the instruction. However, if there is an interrupt request of higher priority than that interrupt, interrupt exception handling will be executed for the higher-priority interrupt, and the lower-priority interrupt will be ignored. The same also applies when an interrupt source flag is cleared to 0. The above conflict will not occur if an enable bit or interrupt source flag is cleared to 0 while the interrupt is masked by the CPU.

#### (2) Instructions that Disable Interrupts

The instructions that disable interrupts are LDC, ANDC, ORC, and XORC. After any of these instructions are executed, all interrupts including NMI are disabled and the next instruction is always executed. When the I bit is set by one of these instructions, the new value becomes valid two states after execution of the instruction ends.

#### (3) Interrupts during Execution of EEPMOV Instructions

Interrupt operation differs between the EEPMOV.B instruction and the EEPMOV.W instruction.

With the EEPMOV.B instruction, an interrupt request (including NMI) issued during the transfer is not accepted until the transfer is completed.

With the EEPMOV.W instruction, if an interrupt request is issued during the transfer, interrupt exception handling starts at the next break in the transfer cycle. The PC value saved on the stack in this case is the address of the next instruction.

Therefore, if an interrupt is generated during execution of an EEPMOV.W instruction, the following coding should be used.

L1: EEPMOV.W MOV.W R4,R4 BNE L1



Figure 3.3 Stack Structure after Exception Handling (Example)

### **3.4 Program Execution State**

In this state the CPU executes program instructions in sequence.

#### 3.5 Bus-Released State

This is a state in which the bus has been released in response to a bus request from a bus master other than the CPU. While the bus is released, the CPU halts except for internal operations.

Bus masters other than the CPU may include the direct memory access controller (DMAC) and data transfer controller (DTC).

For further details, refer to the relevant microcontroller hardware manual.

### 3.6 Power-Down State

The power-down state includes both modes in which the CPU stops operating and modes in which the CPU does not stop. There are three modes in which the CPU stops operating: sleep mode, software standby mode, and hardware standby mode. There are also two other power-down modes: medium-speed mode and module stop mode. In medium-speed mode the CPU and other bus masters operate on a medium-speed clock. Module stop mode permits halting of the operation of individual modules, other than the CPU. For details, refer to the relevant microcontroller hardware manual.

#### 3.6.1 Sleep Mode

A transition to sleep mode is made if the SLEEP instruction is executed while the software standby bit (SSBY) in the system control register (SYSCR) is cleared to 0. In sleep mode, CPU operations stop immediately after execution of the SLEEP instruction. The contents of CPU registers are retained.

#### 3.6.2 Software Standby Mode

A transition to software standby mode is made if the SLEEP instruction is executed while the SSBY bit in SYSCR is set to 1. In software standby mode, the CPU and clock halt and all on-chip operations stop. The on-chip supporting modules are reset, but as long as a specified voltage is supplied, the contents of CPU registers and on-chip RAM are retained. The I/O ports also remain in their existing states.

#### 3.6.3 Hardware Standby Mode

A transition to hardware standby mode is made when the **STBY** pin goes low. In hardware standby mode, the CPU and clock halt and all on-chip operations stop. The on-chip supporting modules are reset, but as long as a specified voltage is supplied, on-chip RAM contents are retained.



# Section 4 Basic Timing

### 4.1 Overview

The CPU is driven by a system clock, denoted by the symbol  $\phi$ . The period from one rising edge of  $\phi$  to the next is referred to as a "state." The memory cycle or bus cycle consists of one, two, or three states. Different methods are used to access on-chip memory, on-chip supporting modules, and the external address space. Refer to the relevant microcontroller hardware manual for details.

### 4.2 On-Chip Memory (ROM, RAM)

On-chip memory is accessed in one state. The data bus is 16 bits wide, permitting both byte and word access. Figure 4.1 shows the on-chip memory access cycle. Figure 4.2 shows the pin states.



Figure 4.1 On-Chip Memory Access Cycle



Figure 4.2 Pin States during On-Chip Memory Access



### 4.3 On-Chip Supporting Module Access Timing

The on-chip supporting modules are accessed in two states. The data bus is either 8 bits or 16 bits wide, depending on the particular on-chip register being accessed. Figure 4.3 shows the access timing for the on-chip supporting modules. Figure 4.4 shows the pin states.



Figure 4.3 On-Chip Supporting Module Access Timing



Figure 4.4 Pin States during On-Chip Supporting Module Access

## 4.4 External Address Space Access Timing

The external address space is accessed with an 8-bit or 16-bit data bus width in a two-state or three-state bus cycle. Figure 4.5 shows the read timing for two-state and three-state access. Figure 4.6 shows the write timing for two-state and three-state access. In three-state access, wait states can be inserted. For further details, refer to the relevant microcontroller hardware manual.





Figure 4.5 External Device Access Timing (Read Timing)



Figure 4.6 External Device Access Timing (Write Timing)

#### Renesas 16-Bit Single-Chip Microcomputer Software Manual H8S/2600 Series, H8S/2000 Series

Publication Date:	1st Edition, March 1995	
	Rev.4.00, February 24, 2006	
Published by:	Sales Strategic Planning Div.	
	Renesas Technology Corp.	
Edited by:	Customer Support Department	
	Global Strategic Communication Div.	
	Renesas Solutions Corp.	

#### RenesasTechnologyCorp. Sales Strategic Planning Div. Nippon Bldg., 2-6-2, Ohte-machi, Chiyoda-ku, Tokyo 100-0004, Japan



http://www.renesas.com

#### RENESAS SALES OFFICES

#### Refer to "http://www.renesas.com/en/network" for the latest and detailed information.

#### Renesas Technology America, Inc.

450 Holger Way, San Jose, CA 95134-1368, U.S.A Tel: <1> (408) 382-7500, Fax: <1> (408) 382-7501

Renesas Technology Europe Limited Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K. Tel: <44 - (1628) 585-100, Fax: <44 - (1628) 585-900

Renesas Technology (Shanghai) Co., Ltd. Unit 204, 205, AZIACenter, No.1233 Lujiazui Ring Rd, Pudong District, Shanghai, China 200120 Tel: <86> (21) 5877-1818, Fax: <86> (21) 6887-7898

Renesas Technology Hong Kong Ltd. 7th Floor, North Tower, World Finance Centre, Harbour City, 1 Canton Road, Tsimshatsui, Kowloon, Hong Kong Tel: <852> 2265-6688, Fax: <852> 2730-6071

Renesas Technology Taiwan Co., Ltd. 10th Floor, No.99, Fushing North Road, Taipei, Taiwan Tel: <886> (2) 2715-2888, Fax: <886> (2) 2713-2999

Renesas Technology Singapore Pte. Ltd. 1 Harbour Front Avenue, #06-10, Keppel Bay Tower, Singapore 098632 Tel: <65- 6213-0200, Fax: <65- 6278-8001

Renesas Technology Korea Co., Ltd. Kukje Center Bldg. 18th Fl., 191, 2-ka, Hangang-ro, Yongsan-ku, Seoul 140-702, Korea Tel: <82> (2) 796-3115, Fax: <82> (2) 796-2145

#### Renesas Technology Malaysia Sdn. Bhd

Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No.18, Jalan Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia Tel: <603>7955-9390. Fax: <603>7955-9510

# H8S/2600 Series, H8S/2000 Series Software Manual



Renesas Electronics Corporation 1753, Shimonumabe, Nakahara-ku, Kawasaki-shi, Kanagawa 211-8668 Japan

REJ09B0139-0400