User's Manual

32

RX族

用户手册 软件篇

瑞萨32位单片机
RX族

本资料所记载的内容，均为本资料发行时的信息，瑞萨电子对于本资料所记载的产品或者规格可能会作改动，恕不另行通知。
请通过瑞萨电子的主页确认发布的最新信息。

瑞萨电子
www.renesas.com

Rev.1.00 2011.01

Notice

1. 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.
2. 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 categorized as “Specific” without the prior 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. The quality grade of each Renesas Electronics product is “Standard” unless otherwise expressly specified in a Renesas Electronics data sheets or data books, 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; anti-crime 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.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

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

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

略称

以下说明本手册中使用的略称。

分类	略称	含义
符号	IMM	表示立即数 （Immediate）。
	SIMM	表示按处理长度进行符号扩展的立即数 （Signed）。
	UIMM	表示按处理长度进行零扩展的立即数 （Unsigned）。
	src	表示指令的源 （Source）操作数。
	dest	表示指令的目标 （Destination）操作数。
	dsp	表示相对寻址的位移量 （Displacement）。
	pcdsp	表示程序计数器相对寻址的位移量 （Displacement）。
	[ ]	表示间接寻址。
	Rn	表示通用寄存器。在没有特别要求的情况下，能指定 R0 ～ R15。
	Rs	表示通用寄存器 （Source）。在没有特别要求的情况下，能指定 R0 ～ R15。
	Rs2	主要用于 ADD、AND、CMP、MUL、OR、PUSHM、SUB、TST 指令的说明。对于这些指令，能给操作数指定 2 个通用寄存器 （Source），因此将第 1 个通用寄存器 （Source）记述为 Rs，第 2 个通用寄存器 （Source）记述为 Rs2。
	Rd	表示通用寄存器 （Destination）。在没有特别要求的情况下，能指定 R0 ～ R15。
	Rd2	主要用于 POPM 指令和 RTSD 指令的说明。对于这些指令，能给操作数指定 2 个通用寄存器 （Destination），因此将第 1 个通用寄存器 （Destination）记述为 Rd，第 2 个通用寄存器 （Destination）记述为 Rd2。
	Rb	表示通用寄存器 （Base）。在没有特别要求的情况下，能指定 R0 ～ R15。
	Ri	表示通用寄存器 （Index）。在没有特别要求的情况下，能指定 R0 ～ R15。
	Rx	表示控制寄存器。能指定 PC、 ISP、 USP、 INTB、 PSW、 BPC、 BPSW、 FINTV、 FPSW。只能将 PC 指定为 MVFC 指令和 PUSHC 指令的 src。
	flag	表示 PSW 的位 （U、 I）和标志 （O、 S、 Z、 C）。
数值	000b	表示 2 进制数。
	0000h	表示 16 进制数。
位长	#IMM:8 等	表示操作数的有效位长。
	:1	表示有效位长为 1 位。
	:2	表示有效位长为 2 位。
	:3	表示有效位长为 3 位。
	:4	表示有效位长为 4 位。
	:5	表示有效位长为 5 位。
	:8	表示有效位长为 8 位。
	:16	表示有效位长为 16 位。
	:24	表示有效位长为 24 位。
	:32	表示有效位长为 32 位。
数据长度的指定	MOV.W 等	这是指定指令的处理长度的符号。
	.B	指定字节 （8 位）。
	.W	指定字 （16 位）。
	.L	指定长字 （32 位）。

分类	略称	含义
转移距离的指定	BRA.A 等	这是指定转移相对距离的有效位长的符号。
	.S	表示 3 位 PC 前方相对，有效值为 3 ～ 10。
	.B	表示 8 位 PC 相对，有效值为 –128 ～ 127。
	.W	表示 16 位 PC 相对，有效值为 –32768 ～ 32767。
	.A	表示 24 位 PC 相对，有效值为 –8388608 ～ 8388607。
	.L	表示 32 位 PC 相对，有效值为 –2147483648 ～ 2147483647。
给存储器操作数附加的长度扩展指定	dsp:16[Rs].UB 等	这是指定存储器操作数的长度和扩展方法的符号。如果省略，就作为长字进行处理。
	.B	指定字节 （8 位），扩展方法是符号扩展。
	.UB	指定字节 （8 位），扩展方法是零扩展。
	.W	指定字 （16 位），扩展方法是符号扩展。
	.UW	指定字 （16 位），扩展方法是零扩展。
	.L	指定长字 （32 位）。
操作	（原则上遵循 C 语言的语法规则。以下说明本手册中使用的记述。）	
	=	这是赋值运算符，将右边的值赋给左边。
	–	表示一元运算符的负号或者二元运算符的 “减”。
	+	表示二元运算符的 “加”。
	*	表示指针运算符或者二元运算符的 “乘”。
	/	表示二元运算符的 “除”。
	%	表示二元运算符的 “模”。
	~	表示一元位运算符的 “NOT”。
	&	表示二元位运算符的 “AND”。
	|	表示二元位运算符的 “OR”。
	^	表示二元位运算符的 “Exclusive OR”。
	;	表示语句的结束。
	{ }	表示多个语句的开始和结束，能在 { } 内记述多个语句。
	if (表达式) 语句 1 else 语句 2	表示 if 语句。评估表达式，当表达式是真时，执行语句 1 ； 当表达式是假时，执行语句 2。
	for (语句 1; 表达式 ; 语句 2) 语句 3	表示 for 语句。在执行语句 1 后评估表达式，当表达式是真时，执行语句 3。在执行语句 3 后执行语句 2，然后评估表达式。
	do 语句 while (表达式);	表示 do 语句。在表达式为真的期间执行语句。与表达式的真假无关，至少执行 1 次语句。
	while (表达式) 语句	表示 while 语句。在表达式为真的期间执行语句。
	==、 !=	这是比较运算符，分别表示 “等于”和 “不等于”。
	>、 <	这是比较运算符，分别表示 “大于”和 “小于”。
	>=、 <=	这是比较运算符。‘>’、‘<’加上 ‘==’的条件。
	&&	这是逻辑运算符。左侧的条件和右侧的条件的逻辑运算为 “AND”。
	||	这是逻辑运算符。左侧的条件和右侧的条件的逻辑运算为 “OR”。
	<<、 >>	这是移位运算符，表示 “向左移位”和 “向右移位”。
	tmp、 tmp0、 tmp1、 tmp2、 tmp3	表示暂存器。
	!	logical NOT。将变量和表达式的布尔运算值取反。
浮点数	NaN	非数值 （Not a Number）
浮点格式	SNaN	Signaling NaN
	QNaN	Quiet NaN

目　录

RX 族指令一览表

表 A.1　字母 - 页速查表 (1 / 4)

助记符		功能	详细记载指令的页	详细记载指令码的页
ABS		绝对值	29	146
ADC		带进位的加法运算	30	147
ADD		不带进位的加法运算	31	148
AND		逻辑与	33	150
BCLR		位清除	35	152
B*Cnd*	BGEU	相对条件转移	36	154
	BC		36	154
	BEQ		36	154
	BZ		36	154
	BGTU		36	154
	BPZ		36	154
	BGE		36	154
	BGT		36	154
	BO		36	154
	BLTU		36	154
	BNC		36	154
	BNE		36	154
	BNZ		36	154
	BLEU		36	154
	BN		36	154
	BLE		36	154
	BLT		36	154
	BNO		36	154
BM*Cnd*	BMGEU	条件位传送	37	155
	BMC		37	155
	BMEQ		37	155
	BMZ		37	155
	BMGTU		37	155
	BMPZ		37	155
	BMGE		37	155
	BMGT		37	155
	BMO		37	155
	BMLTU		37	155
	BMNC		37	155
	BMNE		37	155
	BMNZ		37	155
	BMLEU		37	155
	BMN		37	155
	BMLE		37	155
	BMLT		37	155
	BMNO		37	155

表 A.1　字母 - 页速查表 (2 / 4)

助记符	功能	详细记载指令的页	详细记载指令码的页
BNOT	位取反	39	156
BRA	相对无条件转移	40	157
BRK	无条件陷阱	41	158
BSET	位置位	42	158
BSR	相对子程序转移	43	160
BTST	位测试	44	161
CLRPSW	PSW 的标志和位的清除	45	162
CMP	比较	46	163
DIV	带符号的除法运算	47	164
DIVU	不带符号的除法运算	48	166
EMUL	带符号的乘法运算	49	167
EMULU	不符号的乘法运算	51	168
FADD	浮点加法运算	53	169
FCMP	浮点比较	55	170
FDIV	浮点除法运算	57	171
FMUL	浮点乘法运算	59	172
FSUB	浮点减法运算	61	173
FTOI	浮点数 → 整数的转换	63	174
INT	软件中断	65	174
ITOF	整数 → 浮点数的转换	66	175
JMP	无条件转移	68	175
JSR	子程序转移	69	176
MACHI	高 16 位的乘加运算	70	176
MACLO	低 16 位的乘加运算	71	176
MAX	最大值选择	72	177
MIN	最小值选择	73	178
MOV	传送	74	179
MOVU	不带符号的数据传送	78	184
MUL	乘法运算	79	185
MULHI	高 16 位的乘法运算	81	186
MULLO	低 16 位的乘法运算	82	187
MVFACHI	从累加器高 32 位的传送	83	187
MVFACMI	从累加器中间 32 位的传送	84	187
MVFC	从控制寄存器的传送	85	188
MVTACHI	向累加器高 32 位的传送	86	188
MVTACLO	向累加器低 32 位的传送	87	189
MVTC	向控制寄存器的传送	88	189
MVTIPL （特权指令）（注）	中断优先级的设定	89	190
NEG	符号取反	90	191
NOP	空操作	91	191
NOT	逻辑取反	92	192
OR	逻辑或	93	193
POP	寄存器的数据退栈	95	194
POPC	控制寄存器的退栈	96	195

表 A.1　字母 - 页速查表 (3 / 4)

助记符		功能	详细记载指令的页	详细记载指令码的页
POPM		多个寄存器的退栈	97	195
PUSH		数据的压栈	98	196
PUSHC		控制寄存器的压栈	99	197
PUSHM		多个寄存器的压栈	100	197
RACW		16 位带符号的累加器舍入处理	101	198
REVL		字节序转换	103	198
REVW		字节序转换	104	198
RMPA		乘加运算	105	199
ROLC		带进位的左循环	107	199
RORC		带进位的右循环	108	199
ROTL		左循环	109	200
ROTR		右循环	110	200
ROUND		浮点数 → 整数的转换	111	201
RTE （特权指令）		异常返回	114	201
RTFI （特权指令）		从高速中断的返回	115	201
RTS		从子程序的返回	116	202
RTSD		栈帧的释放和从子程序的返回	117	202
SAT		32 位带符号的饱和处理	119	202
SATR		RMPA 指令的 64 位带符号的饱和处理	120	203
SBB		带借位的减法运算	121	203
SC*Cnd*	SCGEU	条件设定	122	204
	SCC		122	204
	SCEQ		122	204
	SCZ		122	204
	SCGTU		122	204
	SCPZ		122	204
	SCGE		122	204
	SCGT		122	204
	SCO		122	204
	SCLTU		122	204
	SCNC		122	204
	SCNE		122	204
	SCNZ		122	204
	SCLEU		122	204
	SCN		122	204
	SCLE		122	204
	SCLT		122	204
	SCNO		122	204
SCMPU		字符串比较	123	204
SETPSW		PSW 的标志和位的置位	124	205
SHAR		算术右移	125	205
SHLL		逻辑 / 算术左移	126	206
SHLR		逻辑右移	127	207
SMOVB		字符串反向传送	128	207

表 A.1　字母 - 页速查表 (4 / 4)

助记符	功能	详细记载指令的页	详细记载指令码的页
SMOVF	字符串正向传送	129	208
SMOVU	字符串传送	130	208
SSTR	字符串存储	131	208
STNZ	带条件的传送	132	209
STZ	带条件的传送	133	209
SUB	不带借位的减法运算	134	210
SUNTIL	字符串搜索	135	211
SWHILE	字符串搜索	137	211
TST	测试	139	212
WAIT （特权指令）	等待	140	213
XCHG	交换	141	213
XOR	逻辑异或	142	214

注 .　在 RX610 群中，不能使用 MVTIPL 指令。

表 A.2　功能 - 页速查表 (1 / 5)

指令的种类	助记符	功能	详细记载指令的页	详细记载指令码的页
算术 / 逻辑运算指令	ABS	绝对值	29	146
	ADC	带进位的加法运算	30	147
	ADD	不带进位的加法运算	31	148
	AND	逻辑与	33	150
	CMP	比较	46	163
	DIV	带符号的除法运算	47	164
	DIVU	不带符号的除法运算	48	166
	EMUL	带符号的乘法运算	49	167
	EMULU	不带符号的乘法运算	51	168
	MAX	最大值选择	72	177
	MIN	最小值选择	73	178
	MUL	乘法运算	79	185
	NEG	符号取反	90	191
	NOP	空操作	91	191
	NOT	逻辑取反	92	192
	OR	逻辑或	93	193
	RMPA	乘加运算	105	199
	ROLC	带进位的左循环	107	199
	RORC	带进位的右循环	108	199
	ROTL	左循环	109	200
	ROTR	右循环	110	200
	SAT	32 位带符号的饱和处理	119	202
	SATR	RMPA 指令的 64 位带符号的饱和处理	120	203
	SBB	带借位的减法运算	121	203
	SHAR	算术右移	125	205
	SHLL	逻辑 / 算术左移	126	206
	SHLR	逻辑右移	127	207
	SUB	不带借位的减法运算	134	210
	TST	测试	139	212
	XOR	逻辑异或	142	214
浮点运算指令	FADD	浮点加法运算	53	169
	FCMP	浮点比较	55	170
	FDIV	浮点除法运算	57	171
	FMUL	浮点乘法运算	59	172
	FSUB	浮点减法运算	61	173
	FTOI	浮点数 → 整数的转换	63	174
	ITOF	整数 → 浮点数的转换	66	175
	ROUND	浮点数 → 整数的转换	111	201

表 A.2　功能 - 页速查表 (2 / 5)

指令的种类	助记符		功能	详细记载指令的页	详细记载指令码的页
传送指令	MOV		传送	74	179
	MOVU		不带符号的数据传送	78	184
	POP		寄存器的数据退栈	95	194
	POPC		控制寄存器的退栈	96	195
	POPM		多个寄存器的退栈	97	195
	PUSH		数据的压栈	98	196
	PUSHC		控制寄存器的压栈	99	197
	PUSHM		多个寄存器的压栈	100	197
	REVL		字节序转换	103	198
	REVW		字节序转换	104	198
	SC*Cnd*	SCGEU	条件设定	122	204
		SCC		122	204
		SCEQ		122	204
		SCZ		122	204
		SCGTU		122	204
		SCPZ		122	204
		SCGE		122	204
		SCGT		122	204
		SCO		122	204
		SCLTU		122	204
		SCNC		122	204
		SCNE		122	204
		SCNZ		122	204
		SCLEU		122	204
		SCN		122	204
		SCLE		122	204
		SCLT		122	204
		SCNO		122	204
	STNZ		带条件的传送	132	209
	STZ		带条件的传送	133	209
	XCHG		交换	141	213

表 A.2　功能 - 页速查表 (3 / 5)

指令的种类	助记符		功能	详细记载指令的页	详细记载指令码的页
转移指令	B*Cnd*	BGEU	相对条件转移	36	154
		BC		36	154
		BEQ		36	154
		BZ		36	154
		BGTU		36	154
		BPZ		36	154
		BGE		36	154
		BGT		36	154
		BO		36	154
		BLTU		36	154
		BNC		36	154
		BNE		36	154
		BNZ		36	154
		BLEU		36	154
		BN		36	154
		BLE		36	154
		BLT		36	154
		BNO		36	154
	BRA		相对无条件转移	40	157
	BSR		相对子程序转移	43	160
	JMP		无条件转移	68	175
	JSR		子程序转移	69	176
	RTS		从子程序的返回	116	202
	RTSD		栈帧的释放和从子程序的返回	117	202

表 A.2　功能 - 页速查表 (4 / 5)

指令的种类	助记符		功能	详细记载指令的页	详细记载指令码的页
位操作指令	BCLR		位清除	35	152
	BM*Cnd*	BMGEU	条件位传送	37	155
		BMC		37	155
		BMEQ		37	155
		BMZ		37	155
		BMGTU		37	155
		BMPZ		37	155
		BMGE		37	155
		BMGT		37	155
		BMO		37	155
		BMLTU		37	155
		BMNC		37	155
		BMNE		37	155
		BMNZ		37	155
		BMLEU		37	155
		BMN		37	155
		BMLE		37	155
		BMLT		37	155
		BMNO		37	155
	BNOT		位取反	39	156
	BSET		位置位	42	158
	BTST		位测试	44	161
字符串操作指令	SCMPU		字符串比较	123	204
	SMOVB		字符串反向传送	128	207
	SMOVF		字符串正向传送	129	208
	SMOVU		字符串传送	130	208
	SSTR		字符串存储	131	208
	SUNTIL		字符串搜索	135	211
	SWHILE		字符串搜索	137	211
系统操作指令	BRK		无条件陷阱	41	158
	CLRPSW		PSW 的标志和位的清除	45	162
	INT		软件中断	65	174
	MVFC		从控制寄存器的传送	85	188
	MVTC		向控制寄存器的传送	88	189
	MVTIPL （特权指令）（注）		中断优先级的设定	89	190
	RTE （特权指令）		异常返回	114	201
	RTFI （特权指令）		从高速中断的返回	115	201
	SETPSW		PSW 的标志和位的置位	124	205
	WAIT （特权指令）		等待	140	213

表 A.2　功能 - 页速查表 (5 / 5)

指令的种类	助记符	功能	详细记载指令的页	详细记载指令码的页
DSP 功能指令	MACHI	高 16 位的乘加运算	70	176
	MACLO	低 16 位的乘加运算	71	176
	MULHI	高 16 位的乘法运算	81	186
	MULLO	低 16 位的乘法运算	82	187
	MVFACHI	从累加器高 32 位的传送	83	187
	MVFACMI	从累加器中间 32 位的传送	84	187
	MVTACHI	向累加器高 32 位的传送	86	188
	MVTACLO	向累加器低 32 位的传送	87	189
	RACW	16 位带符号的累加器舍入处理	101	198

注 .　在 RX610 群中，不能使用 MVTIPL 指令。

1. CPU 功能

RX CPU 对于常用指令提供了短格式，能开发出存储容量小而且效率高的程序；具有用 1 个时钟执行的指令，实现快速运算处理；有 73 种基本指令、 8 种浮点运算指令、 9 种 DSP 功能指令共 90 种指令和 10 种寻址方式，能进行寄存器 - 寄存器之间的运算、寄存器 - 存储器之间的运算、位运算以及存储器 - 存储器之间的传送。另外，内置乘法器，能进行快速的乘法运算。

1.1 特点

- 指令的最短执行时间：1个时钟执行1条指令
- 地址空间：4G字节、线性地址
- CPU寄存器组
  通用寄存器：32位×16个
  控制寄存器：32位×9个
  累加器：64位×1个
- 基本指令：73种
  对应转移距离的相对转移指令
  可变长指令格式 （1字节～8字节）
  常用指令有短格式
- 浮点运算指令：8种
- DSP功能指令：9种
  对应16位×16位的乘法指令和乘加指令
  对应累加器的舍入指令
- 寻址方式：10种
- 处理器模式
  管理模式、用户模式
- 浮点运算单元
  对应单精度浮点数 （32位）
  符合IEEE754规格的数据类型以及对应浮点异常
- 存储器保护单元 （选项功能）
- 数据排放
  可选择小端法或者大端法

1.2 CPU 寄存器组

RX CPU 的寄存器有 16 个通用寄存器、 9 个控制寄存器和 1 个由 DSP 功能指令使用的累加器。

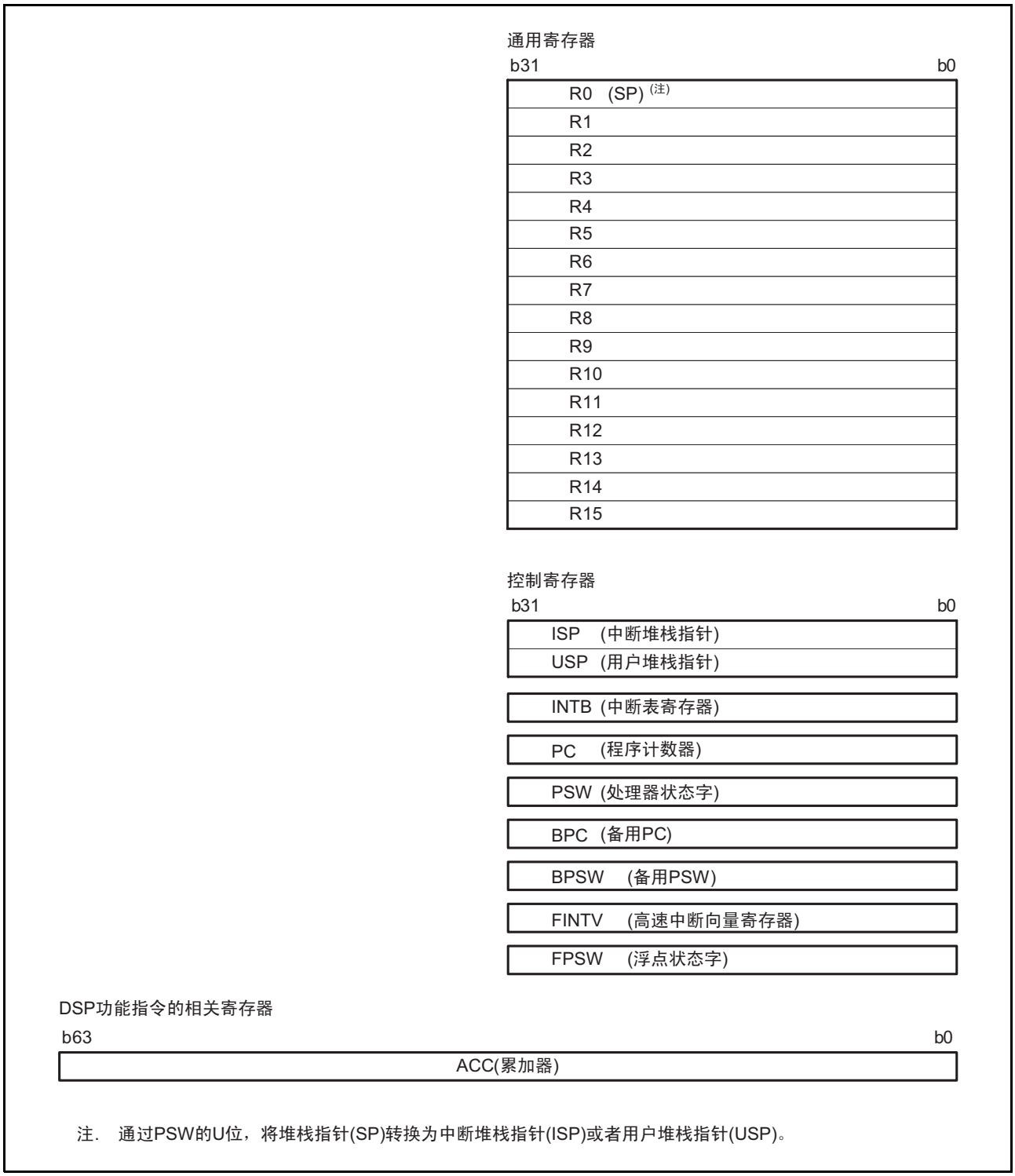


图 1.1 CPU 寄存器组

1.2.1 通用寄存器 （R0 ～ R15）

通用寄存器有 16 个 （R0 ～ R15），用作数据寄存器和地址寄存器。

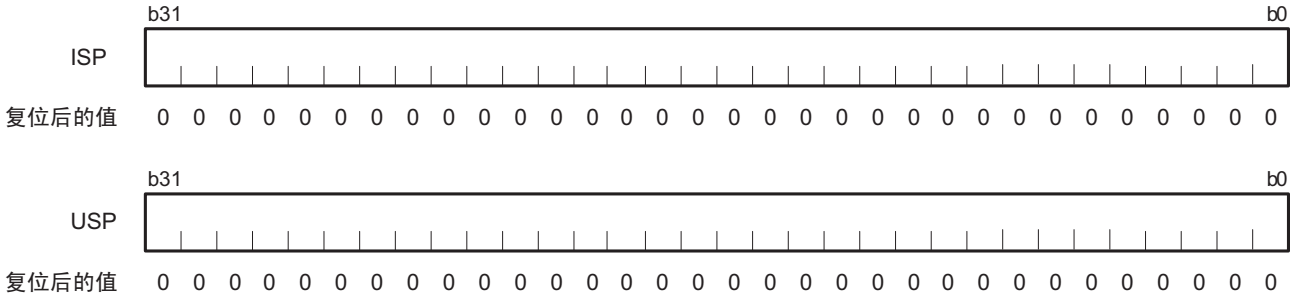
通用寄存器 R0 除了具有通用寄存器的功能以外，还有堆栈指针 （SP）的功能。通过处理器状态字 （PSW）的堆栈指针指定位 （U），将 SP 转换为中断堆栈指针 （ISP）或者用户堆栈指针 （USP）。

1.2.2 控制寄存器

控制寄存器有以下 9 个：

- 中断堆栈指针 （ISP）
- 用户堆栈指针 （USP）
- 中断表寄存器 （INTB）
- 程序计数器 （PC）
- 处理器状态字 （PSW）
- 备用PC （BPC）
- 备用PSW （BPSW）
- 高速中断向量寄存器 （FINTV）
- 浮点状态字 （FPSW）

1.2.2.1 中断堆栈指针 （ISP） / 用户堆栈指针 （USP）



堆栈指针 （SP）有中断堆栈指针 （ISP）和用户堆栈指针 （USP）两种，通过处理器状态字 （PSW）的堆栈指针指定位 （U）转换要使用的堆栈指针 （ISP/USP）。

如果给 ISP 和 USP 设定 4 的倍数，带有堆栈操作的指令和中断响应顺序的周期数就会变短。

1.2.2.2 中断表寄存器 （INTB）

必须给中断表寄存器 （INTB）设定可变向量表的起始地址。

1.2.2.3 程序计数器 （PC）

程序计数器 （PC）表示正在执行的指令的地址。

1.2.2.4 处理器状态字 （PSW）

	b31	b30	b29	b28	b27 (注)	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	IPL[3:0]				—	—	—	PM	—	—	U	I
复位后的值	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	O	S	Z	C
复位后的值	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

注 . 在 RX610 群中，因为中断优先级为 0 ～ 7，所以 b27 为保留位， b27 的写操作无效。

位	符号	位名	功能	R/W
b0	C	进位标志	0：未发生进位 1：发生进位	R/W
b1	Z	零标志	0：运算结果不为 “0” 1：运算结果为 “0”	R/W
b2	S	符号标志	0：运算结果为正数或者 “0” 1：运算结果为负数	R/W
b3	O	上溢标志	0：未发生上溢 1：发生上溢	R/W
b15-b4	—	保留位	读写值都为 “0”。	R/W
b16	I（注 1）	中断允许位	0：禁止中断 1：允许中断	R/W
b17	U（注 1）	堆栈指针指定位	0：指定中断堆栈指针 （ISP） 1：指定用户堆栈指针 （USP）	R/W
b19-b18	—	保留位	读写值都为 “0”。	R/W
b20	PM （注1、注2、注3）	处理器模式设定位	0：设定为管理模式 1：设定为用户模式	R/W
b23-b21	—	保留位	读写值都为 “0”。	R/W
b27-b24	IPL[3:0] （注 1、注 4）	处理器中断优先级	b27 b24 0 0 0 0：0 级 （最低） 0 0 0 1：1 级 0 0 1 0：2 级 0 0 1 1：3 级 0 1 0 0：4 级 0 1 0 1：5 级 0 1 1 0：6 级 0 1 1 1：7 级 1 0 0 0：8 级 1 0 0 1：9 级 1 0 1 0：10 级 1 0 1 1：11 级 1 1 0 0：12 级 1 1 0 1：13 级 1 1 1 0：14 级 1 1 1 1：15 级 （最高）	R/W
b31-b28	—	保留位	读写值都为 “0”。	R/W

注 1. 当设定为用户模式时，忽视用 MVTC 指令和 POPC 指令对 IPL[3:0] 位、 PM 位、 U 位和 I 位的写操作。另外，当用 MVTIPL 指令写 IPL[3:0] 位时，发生特权指令异常。

注 2. 当设定为管理模式时，忽视用 MVTC 指令和 POPC 指令对 PM 位的写操作，但是能写除 PM 位以外的其他位。

注 3. 要从管理模式转换为用户模式时，必须在将被压栈的 PSW 的 PM 位置 “1” 后执行 RTE 指令，或者在将备用 PSW （BPSW) 的 PM 位置 “1” 后执行 RTFI 指令。

注 4. 在 RX610 群中，因为中断优先级为 0 ～ 7，所以 b27 为保留位， b27 的写操作无效。

处理器状态字 （PSW）表示指令的执行结果和 CPU 的状态。

C 标志 （进位标志）
表示运算结果发生进位、借位或者移出。

Z 标志 （零标志）
表示运算结果为 “0”。

S 标志 （符号标志）
表示运算结果为负数。

O 标志 （上溢标志）
表示运算中发生上溢。

I 位 （中断允许位）
此位是允许接受中断请求的位。如果接受异常处理，此位就变为 “0”。

U 位 （堆栈指针指定位）
此位是指定要使用的堆栈指针 （ISP/USP）的位。如果接受异常处理，此位就变为 “0”。如果从管理模式转移到用户模式，此位就变为 “1”。

PM 位 （处理器模式设定位）
此位是设定处理器运行模式的位。如果接受异常处理，此位就变为 “0”。

IPL[3:0] 位 （处理器中断优先级）
IPL[3:0] 位指定 0 级 （最低）～ 15 级 （最高）的 16 个处理器中断优先级。如果发生请求的中断的优先级高于处理器中断优先级，就允许该中断。在将 IPL[3:0] 位设定为 “15 级”（Fh）时，禁止全部的中断。如果发生非屏蔽中断， IPL[3:0] 位就变为 “15 级”（Fh）。如果发生中断，这些位就为所接受中断的优先级。

1.2.2.5 备用 PC （BPC）

复位后的值 不定值

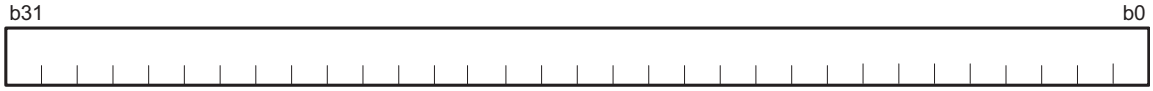
备用 PC （BPC）是为实现中断响应高速化而设计的寄存器。如果发生高速中断，就将程序计数器 （PC）的内容保存到 BPC。

1.2.2.6 备用 PSW （BPSW）

复位后的值 不定值

备用 PSW （BPSW）是为实现中断响应高速化而设计的寄存器。
如果发生高速中断，就将处理器状态字 （PSW）的内容保存到 BPSW。 BPSW 的位配置与 PSW 对应。

1.2.2.7 高速中断向量寄存器 （FINTV）



复位后的值 不定值

高速中断向量寄存器 （FINTV）是为实现中断响应高速化而设计的寄存器。必须设定发生高速中断时的转移目标地址。

1.2.2.8 浮点状态字 （FPSW）

	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	FS	FX	FU	FZ	FO	FV	—	—	—	—	—	—	—	—	—	—
复位后的值	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

位	符号	位名	功能	R/W
b1-b0	RM[1:0]	浮点舍入模式设定位	b1 b0 0 0：向最接近的值舍入 0 1：向 0 方向舍入 1 0：向 +∞ 方向舍入 1 1：向 –∞ 方向舍入	R/W
b2	CV	无效运算源标志	0：未发生无效运算 1：发生无效运算	R/(W) （注 1）
b3	CO	上溢源标志	0：未发生上溢 1：发生上溢	R/(W) （注 1）
b4	CZ	被零除源标志	0：未发生被零除 1：发生被零除	R/(W) （注 1）
b5	CU	下溢源标志	0：未发生下溢 1：发生下溢	R/(W) （注 1）
b6	CX	精度异常源标志	0：未发生精度异常 1：发生精度异常	R/(W) （注 1）
b7	CE	非安装处理源标志	0：未发生非安装处理 1：发生非安装处理	R/(W) （注 1）
b8	DN	非规格化数的 0 刷新位	0：将非规格化数作为非规格化数处理 1：将非规格化数作为 0 处理（注 2）	R/W
b9	—	保留位	读写值都为 “0”。	R/W
b10	EV	无效运算异常处理允许位	0：禁止无效运算引起的异常处理 1：允许无效运算引起的异常处理	R/W
b11	EO	上溢异常处理允许位	0：禁止上溢引起的异常处理 1：允许上溢引起的异常处理	R/W
b12	EZ	被零除异常处理允许位	0：禁止被零除引起的异常处理 1：允许被零除引起的异常处理	R/W

位	符号	位名	功能	R/W
b13	EU	下溢异常处理允许位	0：禁止下溢引起的异常处理 1：允许下溢引起的异常处理	R/W
b14	EX	精度异常处理允许位	0：禁止精度异常引起的异常处理 1：允许精度异常引起的异常处理	R/W
b25-b15	—	保留位	读写值都为 “0”。	R/W
b26	FV（注 3）	无效运算标志	0：未发生无效运算 1：发生无效运算（注 8）	R/W
b27	FO（注 4）	上溢标志	0：未发生上溢 1：发生上溢（注 8）	R/W
b28	FZ（注 5）	被零除标志	0：未发生被零除 1：发生被零除（注 8）	R/W
b29	FU（注 6）	下溢标志	0：未发生下溢 1：发生下溢（注 8）	R/W
b30	FX（注 7）	精度异常标志	0：未发生精度异常 1：发生精度异常（注 8）	R/W
b31	FS	浮点错误概要标志	反映 FU、 FZ、 FO、 FV 标志的逻辑或。	R

注 1. 如果写 “0”，此位就变为 “0”。如果写 “1”，就保持原来的值。
注 2. 正的非规格化数作为 +0 处理，负的非规格化数作为 –0 处理。
注 3. 当 EV 位为 “0” 时， FV 标志有效。
注 4. 当 EO 位为 “0” 时， FO 标志有效。
注 5. 当 EZ 位为 “0” 时， FZ 标志有效。
注 6. 当 EU 位为 “0” 时， FU 标志有效。
注 7. 当 EX 位为 “0” 时， FX 标志有效。
注 8. 一旦该位变为 “1”，就在通过软件置 “0” 前一直保持 “1”。

浮点状态字 （FPSW）表示浮点运算结果。对于没有对应浮点指令的产品， FPSW 的读取值总是为 “00000000h”，并且忽视写操作。

如果通过异常处理允许位 （Ej）允许异常处理 （Ej=1），就能通过异常处理程序检查对应的 Cj 标志，判断异常的发生源。如果禁止异常处理 （Ej=0），就能在一连串处理的最后检查 Fj 标志，确认是否发生异常。 Fj 标志是累积标志 （j=X、 U、 Z、 O、 V）。

RM[1:0] 位 （浮点舍入模式设定位）

这些位是设定浮点舍入模式的位。

【浮点舍入模式的说明】

- 向最接近的值舍入 （默认） ：向接近以无限有效位数进行计算时的结果的值舍入。如果为中间值，就向结果为偶数的方向舍入。
- 向0方向舍入 ：向结果的绝对值变小的方向舍入 （单纯的舍去）。
- 向+∞方向舍入 ：向结果值变大的方向舍入。
- 向–∞方向舍入 ：向结果值变小的方向舍入。

1. “向最接近的值舍入” 是默认模式，返回最正确的值。
2. “向0方向舍入”、“向+∞方向舍入”、“向–∞方向舍入” 用于保证使用了区间运算 （Interval arithmetic）的精度。

CV 标志 （无效运算源标志）、 CO 标志 （上溢源标志）
CZ 标志 （被零除源标志）、 CU 标志 （下溢源标志）
CX 标志 （精度异常源标志）、 CE 标志 （非安装处理源标志）

除 IEEE754 规格规定的 5 种异常 （上溢、下溢、精度异常、被零除、无效运算）以外，如果发生非安装处理，对应的标志就变为 “1”。

- 在对应的标志为 “1”的情况下执行浮点运算指令时，该标志就变为 “0”。
- 如果使用MVTC指令和POPC指令给对应的标志写 “0”，该标志就变为 “0”；如果写 “1”，就保持原来的值。

DN 位 （非规格化数的 0 刷新位）

当此位为 “0”时，将非规格化数作为非规格化数处理。
当此位为 “1”时，将非规格化数作为 0 处理。

EV 位 （无效运算异常处理允许位）、 EO 位 （上溢异常处理允许位）
EZ 位 （被零除异常处理允许位）、 EU 位 （下溢异常处理允许位）
EX 位 （精度异常处理允许位）

在因执行浮点运算指令而发生了 IEEE754 规格规定的 5 种异常时，控制 CPU 是否转移到异常处理。
如果此位为 “0”，就禁止异常处理；如果为 “1”，就允许异常处理。

FV 标志 （无效运算标志）、 FO 标志 （上溢标志）、 FZ 标志 （被零除标志）
FU 标志 （下溢标志）、 FX 标志 （精度异常标志）

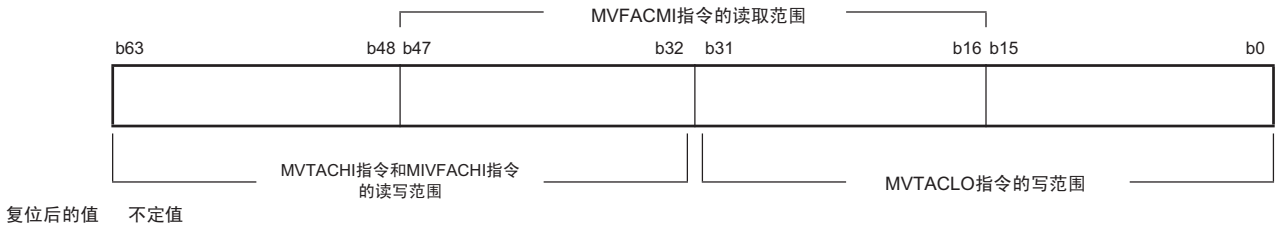
在异常处理允许位 Ej 为 “0”（禁止异常处理）时，如果发生 IEEE754 规格规定的 5 种异常，对应的标志就变为 “1”。

- 当Ej为 “1”（允许异常处理）时，此标志不变。
- 如果该标志变为 “1”，就在通过软件置 “0”前一直保持 “1”（累积标志）。

FS 标志 （浮点错误概要标志）

此标志反映 FU、 FZ、 FO、 FV 标志的逻辑或。

1.2.3 累加器 （ACC）



累加器 （ACC）是 64 位寄存器，用于 DSP 功能指令，也用于乘法指令 （EMUL、 EMULU、 FMUL、 MUL）和乘加运算指令 （RMPA）。在执行这些指令时， ACC 的值被更改。

使用 MVTACHI 指令和 MVTACLO 指令写 ACC。通过 MVTACHI 指令将数据写到高 32 位 （b63 ～ b32），通过 MVTACLO 指令将数据写到低 32 位 （b31 ～ b0）。

使用 MVFACHI 指令和 MVFACMI 指令读 ACC。通过 MVFACHI 指令读高 32 位 （b63 ～ b32）的数据，通过 MVFACMI 指令读中间 32 位 （b47 ～ b16）的数据。

1.3 浮点异常

除 IEEE754 规格规定的 5 种异常事件 （上溢、下溢、精度异常、被零除、无效运算）以外，如果检测到非安装处理，就发生浮点异常。以下说明发生浮点异常的事件概要。

1.3.1 上溢

当运算结果的绝对值大于浮点格式能表现的值时发生上溢异常，发生上溢时的运算结果如**表 1.1** 所示。

表 1.1 发生上溢时的运算结果

浮点舍入模式	结果的符号	运算结果 （目标寄存器的值）	
		EO=0	EO=1
向 –∞ 方向舍入	+	+MAX	不变
	–	–∞	
向 +∞ 方向舍入	+	+∞	
	–	–MAX	
向 0 方向舍入	+	+MAX	
	–	–MAX	
向最接近的值舍入	+	+∞	
	–	–∞	

注 . 如果在 EO 为 “0” 时发生上溢，就发生精度异常。

1.3.2 下溢

当运算结果的绝对值小于浮点格式的正规化数能表现的值 （0 除外）时发生下溢异常，发生下溢时的运算结果如**表 1.2** 所示。

表 1.2 发生下溢时的运算结果

运算结果 （目标寄存器的内容）	
EU=0	EU=1
当 DN 为 “0” 时不变 （发生非安装处理异常）。	不变
当 DN 为 “1” 时返回 “0”。	

1.3.3 精度异常

当假设具有无限有效位数的计算结果与运算结果不同时发生精度异常，精度异常的发生条件和运算结果如**表 1.3** 所示。

表 1.3 精度异常的发生条件和运算结果

发生条件	运算结果 （目标寄存器的内容）	
	EX=0	EX=1
在禁止上溢异常的状态下发生上溢。	参照 “表 1.1 发生上溢时的运算结果”。	不变
发生舍入	舍入后的值	

注 1. 在发生下溢时，不发生精度异常。

注 2. 在允许上溢异常的状态下发生上溢时，与发生舍入无关，不发生精度异常。

1.3.4 被零除

当不为零的有限数被零除时发生被零除异常，发生被零除时的运算结果如**表 1.4** 所示。

表 1.4 发生被零除时的运算结果

被除数	运算结果 （目标寄存器的内容）	
	EZ=0	EZ=1
不为零的有限数	±∞ （符号为除数和被除数的符号的异或）	不变

注 . 在以下情况下，不发生被零除异常。

被除数	运行
0	发生无效运算异常。
∞	不发生异常，结果为 ∞。
非正规化数 （DN=0）	发生非安装处理异常。
QNaN	不发生异常，结果为 QNaN。
SNaN	发生无效运算异常。

1.3.5 无效运算

当执行无效的运算时发生无效运算异常，无效运算的发生条件和运算结果如**表 1.5** 所示。

表 1.5 无效运算的发生条件和运算结果

发生条件	运算结果 （目标寄存器的内容）	
	EV=0	EV=1
对 SNaN 操作数的运算	QNaN	不变
+∞+(–∞) 、 +∞–(+∞) 、 –∞–(–∞)		
0×∞		
0÷0、 ∞÷∞		
在执行 FTOI 指令或者 ROUND 指令时，整数转换发生上溢或者对 NaN、 ∞ 进行了整数转换。	当转换前的符号位为 “0” 时返回 7FFFFFFFh ；当为 “1” 时返回 80000000h。	
对 SNaN 操作数的比较	无目标	

- NaN （Not a Number） ： 非数值
- SNaN （Signaling NaN） ： 这是尾数部的最高位为 “0” 的 NaN。如果将 SNaN 用作运算的源操作数，就发生无效运算异常。通过将 SNaN 用作变量的初始值，有助于发现程序的缺陷。另外， SNaN 不会由硬件生成。
- QNaN （Quiet NaN） ： 这是尾数部的最高位为 “1” 的 NaN。即使将 QNaN 用作运算的源操作数也不发生无效运算异常 （比较和格式转换除外）。因为通过运算 QNaN 被传播，所以不执行异常处理而只看结果也能进行调试。另外，通过运算，由硬件生成 QNaN。

运算结果为 QnaN 时的生成规则如表 1.6 所示。

表 1.6 QNaN 生成规则

源操作数	运算结果 （目标寄存器的内容）
SNaN 和 QnaN	变为 QNaN 的 SNaN 源操作数
都为 SNaN	变为 QNaN 的 dest
都为 QnaN	dest
SNaN 和实数	变为 QNaN 的 SNaN 源操作数
QNaN 和实数	QNaN 源操作数
都不是 NaN 并且发生无效运算时	7FFFFFFFh

注 . 将尾数部的最高位置 “1”，使 SNaN 变为 QNaN。

1.3.6 非安装处理

在 DN 为 “0”并且非正规化数作为运算操作数时或者在 DN 为 “0”并且运算结果发生下溢时，发生非安装处理异常。在 DN 为 “1”时不发生非安装处理异常。

不能禁止由发生非安装处理异常引起的异常处理 （没有异常处理允许位来禁止由发生非安装处理异常引起的异常处理）。目标寄存器的内容不变。

1.4 处理器模式

RX CPU 有管理模式和用户模式两种处理器模式。能通过使用这些处理器模式和存储器保护功能，实现对 CPU 资源和存储器的阶层保护结构。能对各处理器模式规定存储器的存取和可执行的指令的权限，管理模式的权限高于用户模式。在复位后，以管理模式运行。

1.4.1 管理模式

在管理模式中，能存取全部的 CPU 资源，还能执行全部的指令。但是，忽视通过 MVTC 指令和 POPC 指令写处理器状态字 （PSW）的处理器模式设定位 （PM）。有关写 PM 位的方法，请参照 “1.2.2.4 处理器状态字（PSW）”。

1.4.2 用户模式

在用户模式中，限制部分 CPU 资源的写存取。被限制写存取的 CPU 资源如下，限制对象为全部指令的存取。

- 处理器状态字 （PSW）的部分位 （IPL[3:0]、 PM、 U、 I）
- 中断堆栈指针 （ISP）
- 中断表寄存器 （INTB）
- 备用PSW （BPSW）
- 备用PC （BPC）
- 高速中断向量寄存器 （FINTV）

1.4.3 特权指令

特权指令是只能在管理模式中执行的指令。如果在用户模式中执行特权指令，就会发生特权指令异常。特权指令有 RTFI、 MVTIPL、 RTE、 WAIT 指令。

1.4.4 处理器模式之间的转移

通过处理器状态字 （PSW）的处理器模式设定位 （PM）转换处理器模式。但是，通过 MVTC 指令和 POPC 指令对 PM 位的改写无效。必须通过以下所示的方法进行转换。

(1) 用户模式向管理模式的转移

如果发生异常， PSW 的 PM 位就变为 “0”， CPU 转移到管理模式。在管理模式中执行硬件预处理。被保存的 PSW 的 PM 位保持发生异常前的处理器模式。

(2) 管理模式向用户模式的转移

在将被压栈的 PSW 的 PM 位置 “1”后执行 RTE 指令，或者在将被保存到备用 PSW （BPSW）中的 PSW 的 PM 位置 “1”后执行 RTFI 指令，向用户模式转移。一旦转移到用户模式， PSW 的堆栈指针指定位 （U）就变为 “1”。

1.5 数据类型

RX CPU 能处理整数、浮点数、位、字符串共 4 种数据。

1.5.1 整数

整数有带符号整数和不带符号整数，带符号整数的负值用 2 的补数表现。

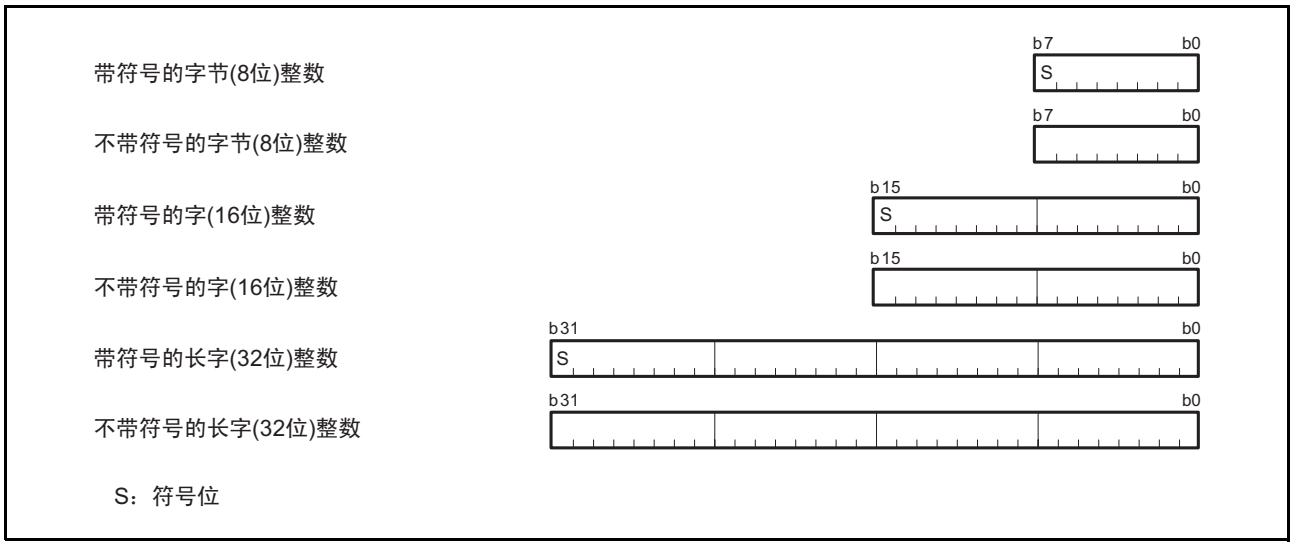


图 1.2 整数

1.5.2 浮点数

浮点数对应 IEEE754 规定的单精度浮点数。浮点数能用于浮点运算指令 FADD、 FCMP、 FDIV、 FMUL、 FSUB、 FTOI、 ITOF、 ROUND 共 8 种指令。

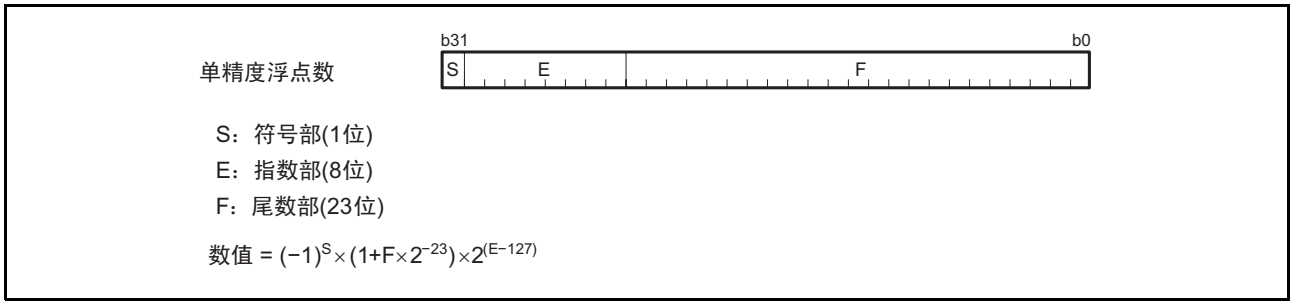


图 1.3 浮点数

浮点数对应以下数值：

- 0<E<255 （规格化数-Normal Numbers）
- E=0 并且 F=0 （零-Signed Zero）
- E=0 并且 F>0 （非规格化数-Subnormal Numbers）（注）
- E=255 并且 F=0 （无穷大-Infinity）
- E=255 并且 F>0 （非数值-NaN：Not a Number）

注 . 当 FPSW 的 DN 位为 “1” 时，作为 0 处理；当 DN 位为 “0” 时，发生非安装处理异常。

1.5.3 位

位用于位操作指令 BCLR、 BM*Cnd*、 BNOT、 BSET、 BTST 共 5 种指令。

通过对象寄存器和 31 ～ 0 的位号指定寄存器的位。

通过对象地址和 7 ～ 0 的位号指定存储器的位。能用于地址指定的寻址方式有寄存器间接和寄存器相对两种。

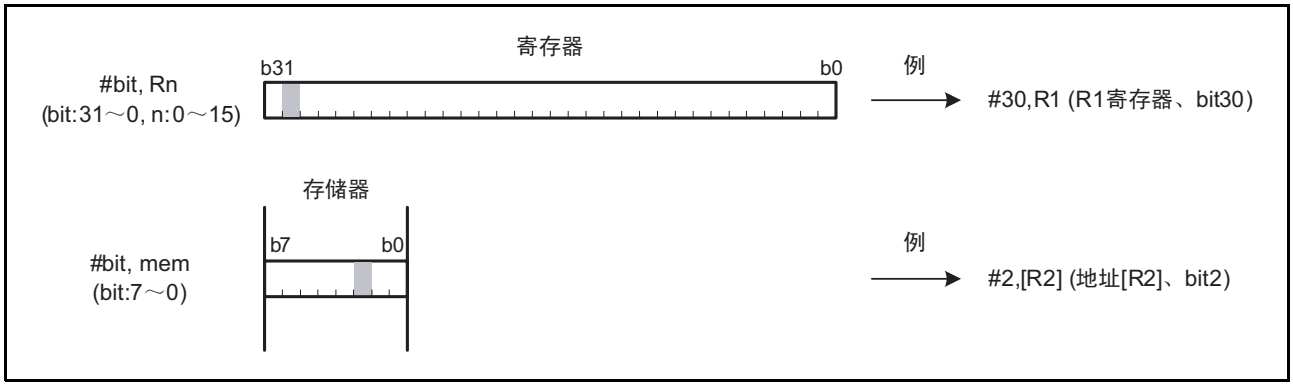


图 1.4 位

1.5.4 字符串

字符串是指只连续排列任意个数的字节 （8 位）、字 （16 位）或者长字 （32 位）数据的数据类型。字符串能用于字符串操作指令 SCMPU、 SMOVB、 SMOVF、 SMOVU、 SSTR、 SUNTIL、 SWHILE 共 7 种指令。

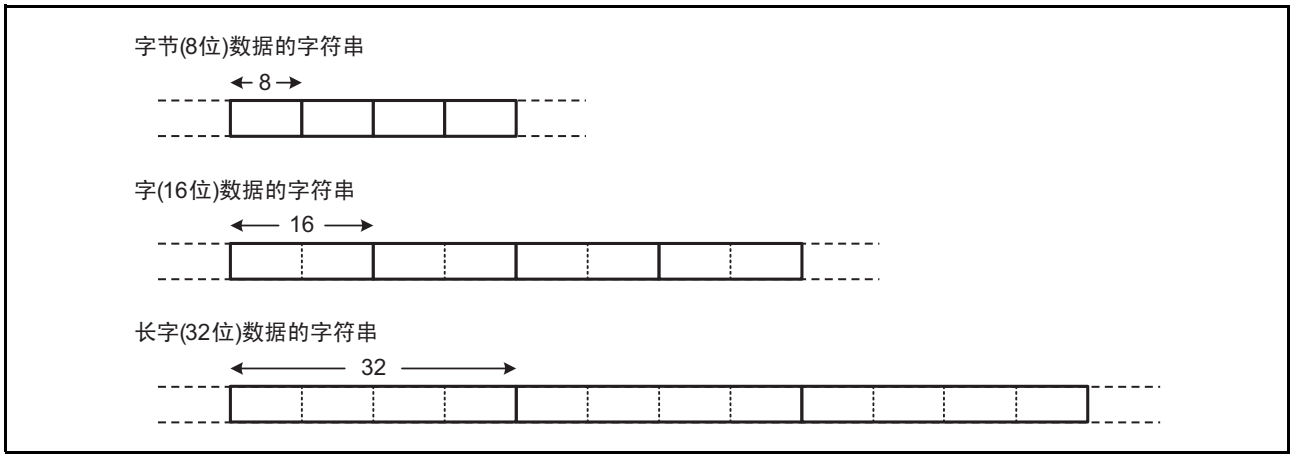


图 1.5 字符串

1.6 数据排放

1.6.1 寄存器的数据排放

寄存器的数据长度和位号的关系如图 1.6 所示。

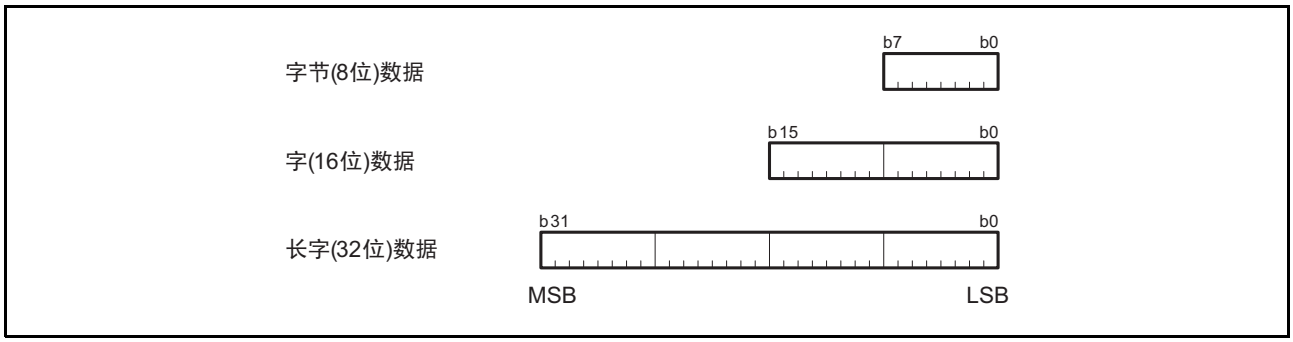


图 1.6 寄存器的数据排放

1.6.2 存储器的数据排放

存储器的数据长度有字节 （8 位）、字 （16 位）和长字 （32 位）共 3 种，能选择小端法或者大端法的数据排放方式。存储器的数据排放如图 1.7 所示。

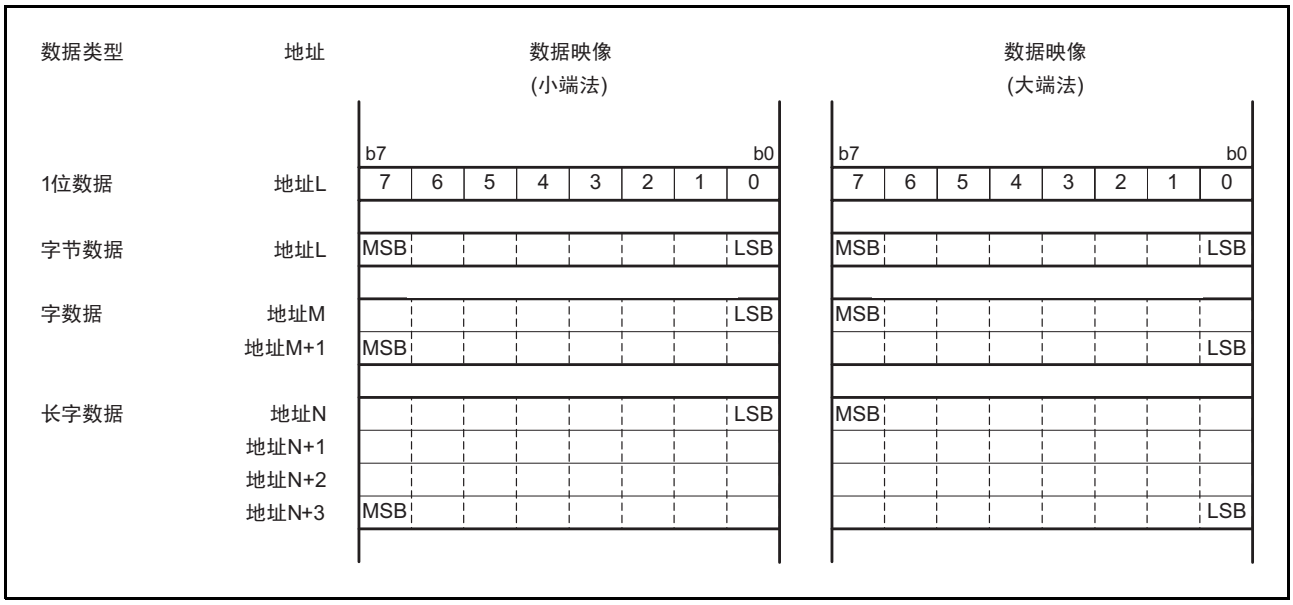


图 1.7 存储器的数据排放

1.7 向量表

向量表有固定向量表和可变向量表。向量表的 1 个向量由 4 字节构成，各向量设有对应的异常处理程序的起始地址。

1.7.1 固定向量表

固定向量表是表的分配地址被固定的向量表。特权指令异常、存取异常、未定义指令异常、浮点异常、非屏蔽中断、复位的各向量分配在地址 FFFFFF80h ～ FFFFFFFFh。固定向量表如图 1.8 所示。

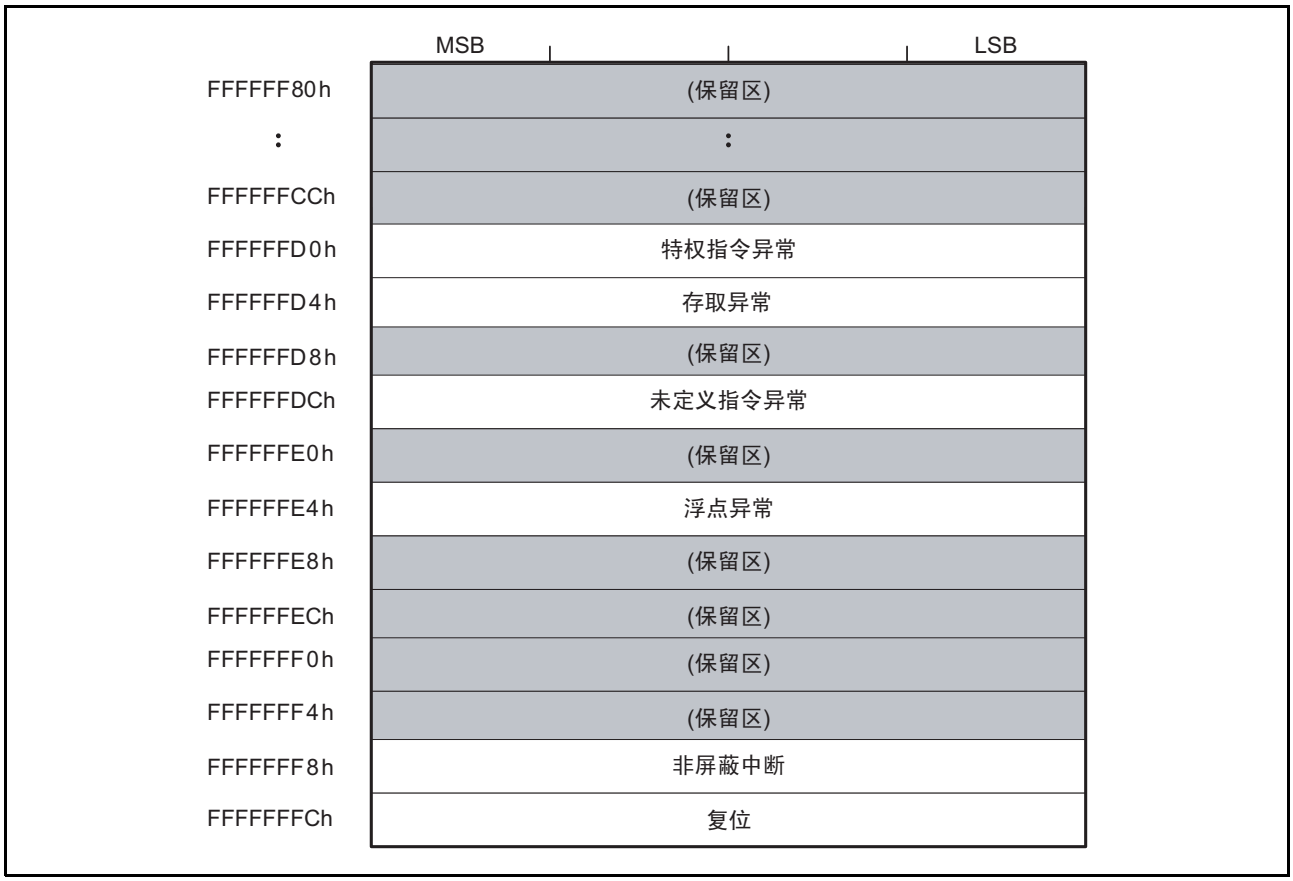


图 1.8 固定向量表

1.7.2 可变向量表

可变向量表是能改变表的分配地址的向量表。无条件陷阱和中断的各向量分配在以中断表寄存器 （INTB）的内容所示的值为起始地址 （IntBase）的 1024 字节的区域。可变向量表如图 1.9 所示。

可变向量表中的每个向量带有序号 （0 ～ 255）。在无条件陷阱发生源的 INT 指令时，分配了与 INT 指令号（0 ～ 255）对应的向量，而在 BRK 指令时，分配了序号 0 的向量。另外，在中断源时，分配了各产品规定的序号 （0 ～ 255）。

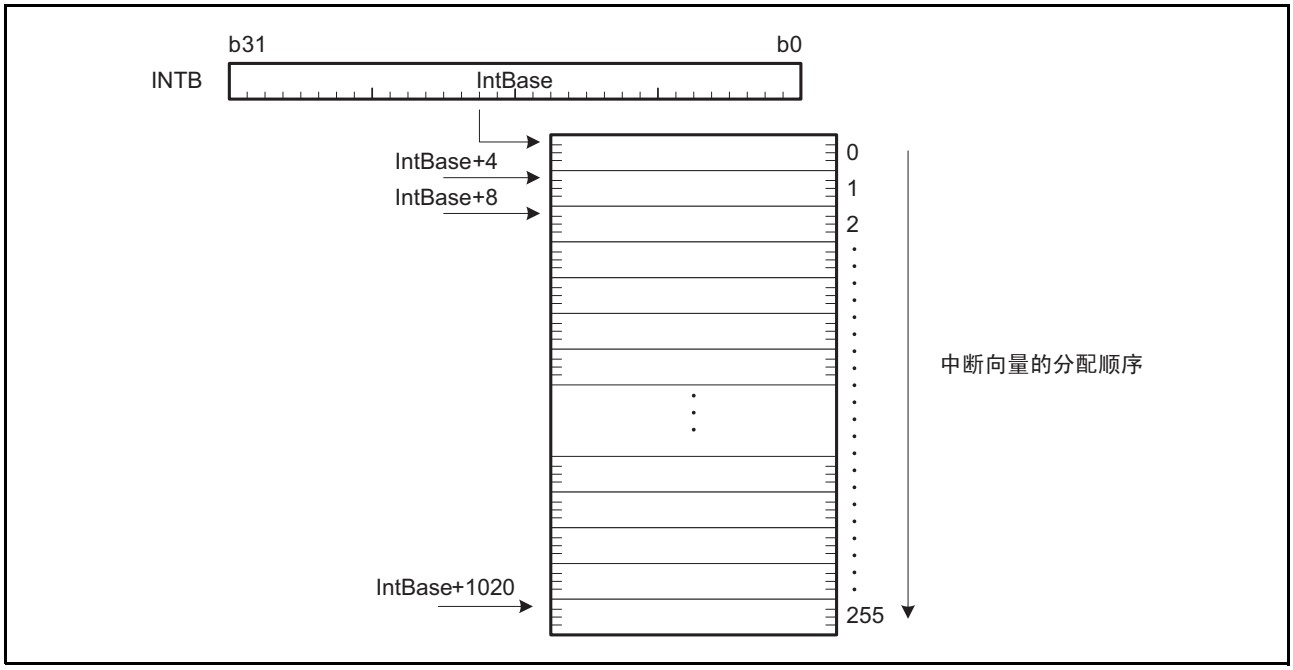


图 1.9 可变向量表

1.8 地址空间

RX CPU 的地址空间有地址 00000000h ～地址 FFFFFFFFh 的 4G 字节，能对程序区和数据区共计最多 4G 字节的空间进行线性存取。RX CPU 的地址空间如图**图 1.10** 所示，各区域因产品和运行模式而不同，详细内容请参照各产品的硬件手册。

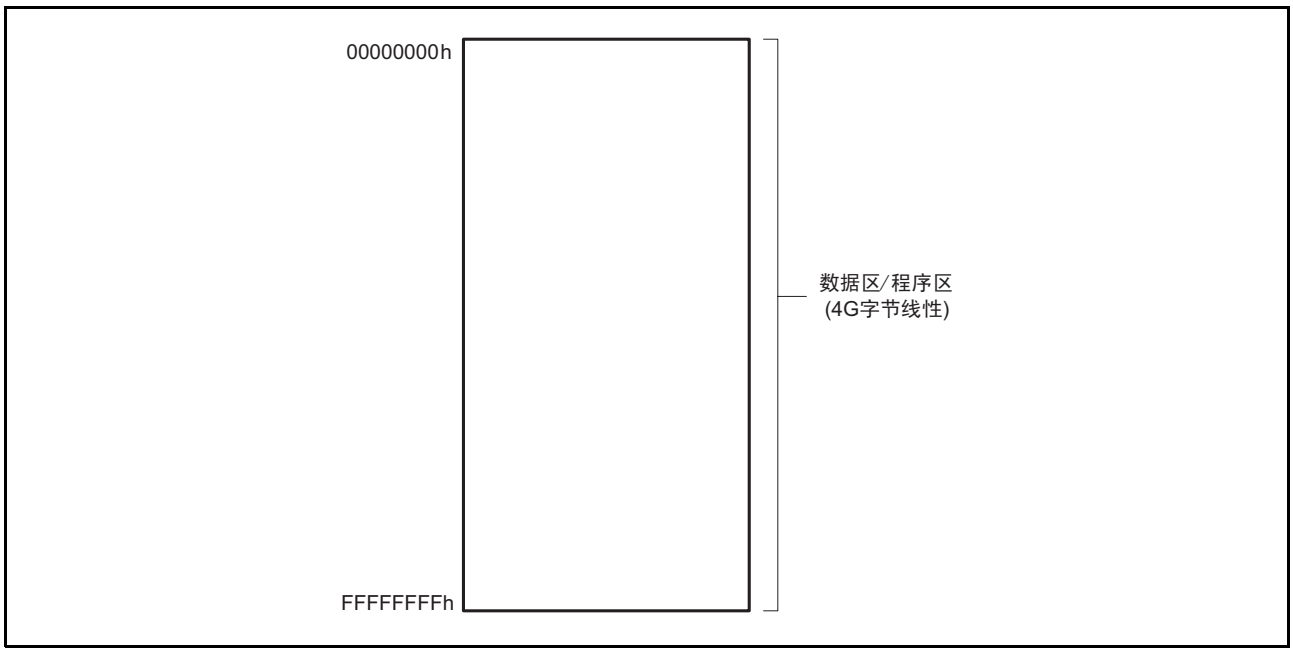


图 1.10 地址空间

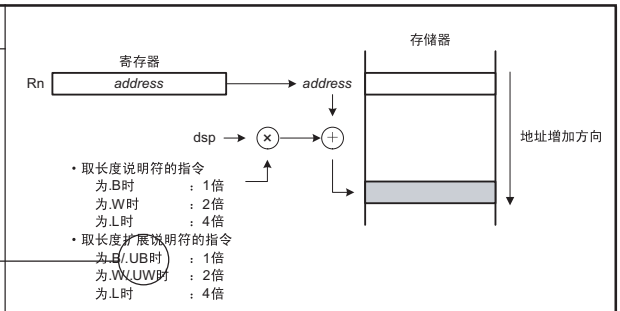
2. 寻址方式

在本章中，按各寻址方式对表示寻址方式的符号和操作进行说明。
寻址方式有以下所示的 10 种类型：

- 立即数
- 寄存器直接
- 寄存器间接
- 寄存器相对
- 后增寄存器间接
- 先减寄存器间接
- 带变址的寄存器间接
- 控制寄存器直接
- PSW 直接
- 程序计数器相对

2.1 本章的阅读方法

通过以下的例子说明本章的阅读方法。

(1) 寄存器相对		
(2) dsp:5[Rn] (Rn=R0 ～ R7)	将位移量 （dsp）的值进行零扩展，在扩展为 32 位后按照规则 （参照右图），将 1/2/4 倍的值和寄存器的值进行加法运算，运算结果的低 32 位为运算对象的有效地址。有效地址的范围为 00000000h ～ FFFFFFFFh。	
(3) dsp:8[Rn] (Rn=R0 ～ R15) dsp:16[Rn] (Rn=R0 ～ R15)	dsp:n 表示 n 位长的位移量。 能指定 dsp:5[Rn] (Rn=R0 ～ R7)、dsp:8[Rn] (Rn=R0 ～ R15)、dsp:16[Rn] (Rn=R0 ～ R15)。 dsp:5[Rn] (Rn=R0 ～ R7) 只能用于 MOV 指令和 MOVU 指令。	
(4) 操作图		

(1) 名称

这是寻址方式的名称。

(2) 符号

这是表示寻址方式的符号。
“:8”、“:16”表示前一个值的有效位数。在手册中需要明确记述有效位数，而在编程时不需要记述。

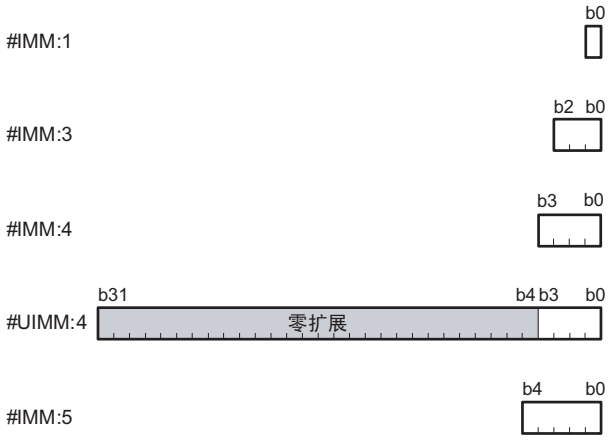
(3) 解说

说明操作和有效地址的范围。

(4) 操作图

用图说明地址的操作。

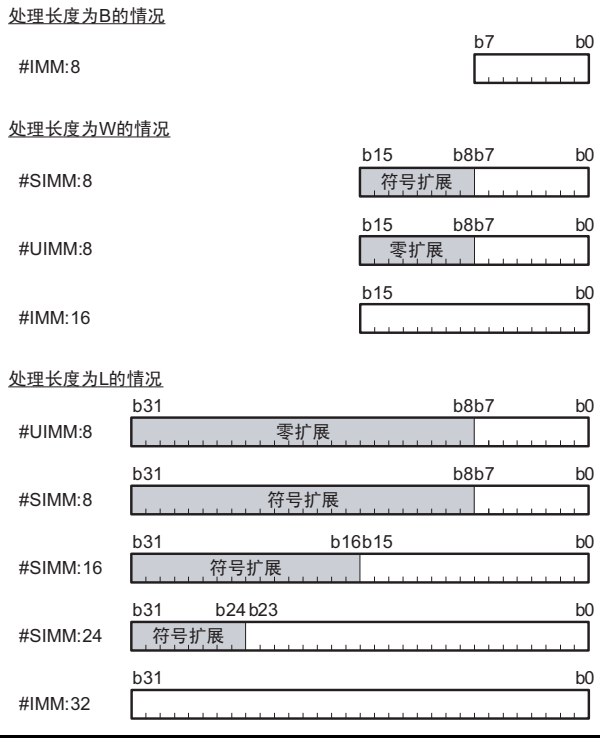
2.2 寻址方式

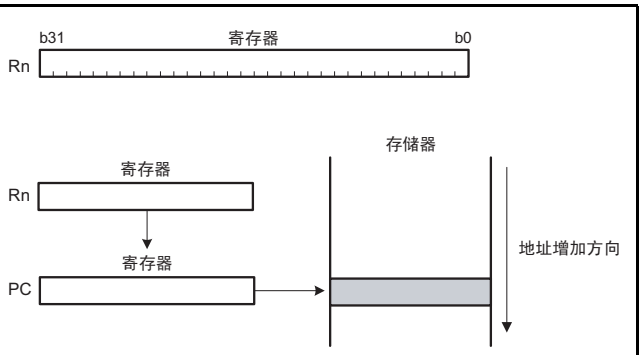
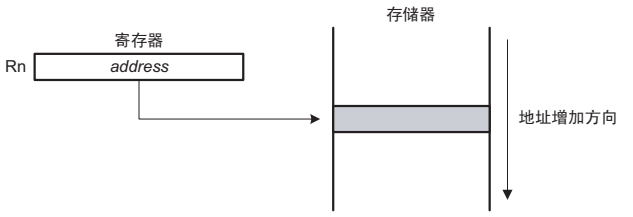
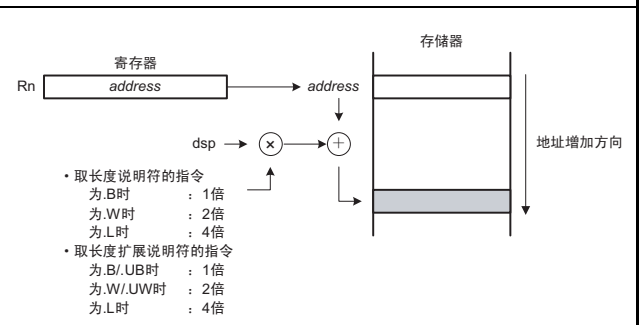
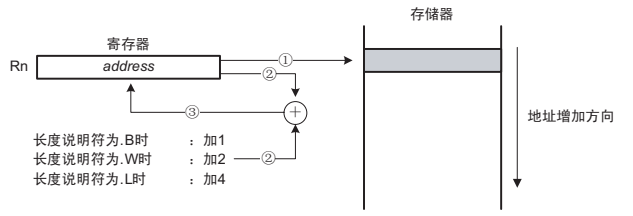
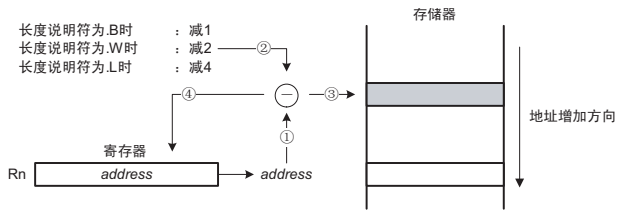
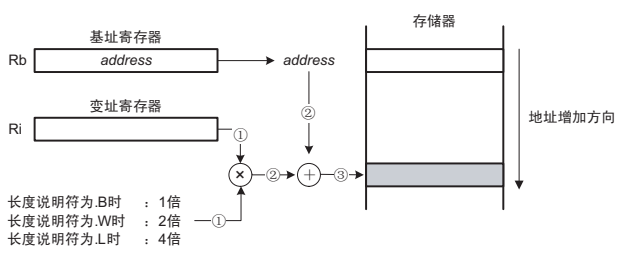
<table>
<tr><td colspan="2">立即数</td><td rowspan="2"> 
</td></tr>
<tr><td>#IMM:1
#IMM:3
#UIMM:4
#IMM:5</td><td>• #IMM:1
#IMM 所示的 1 位立即数为运算对象。此寻址方式用于 RACW 指令的源操作数。

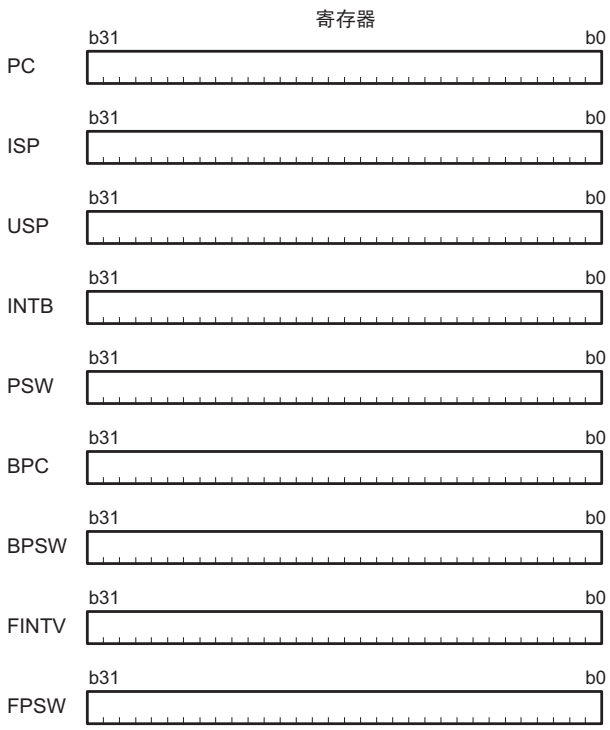
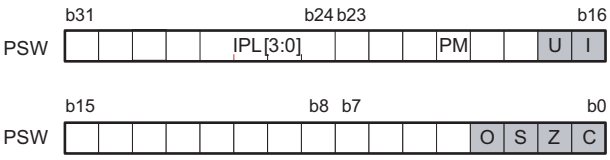
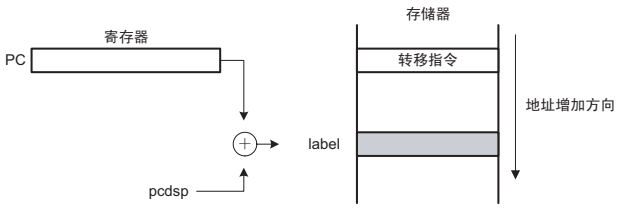
• #IMM:3
#IMM 所示的 3 位立即数为运算对象。此寻址方式用于指定位操作指令 （BCLR、BMCnd、 BNOT、 BSET、 BTST）的位号。

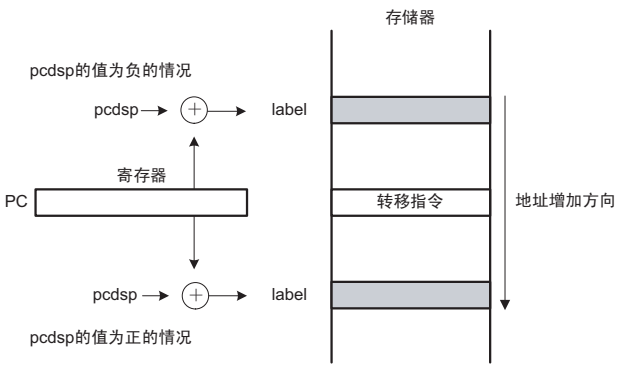
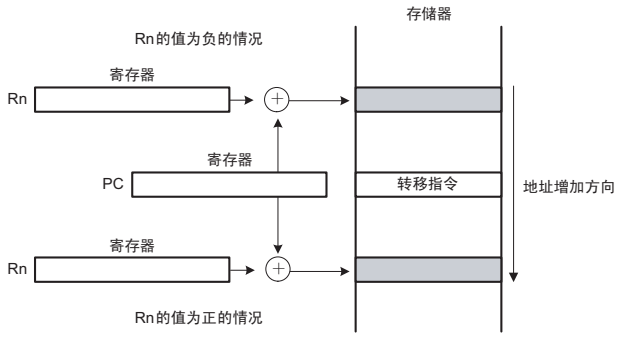
• #IMM:4
#IMM 所示的 4 位立即数为运算对象。此寻址方式用于指定 MVTIPL 指令的中断优先级。

• #UIMM:4
将 #UIMM 所示的 4 位立即数进行零扩展，扩展为 32 位的结果为运算对象。此寻址方式用于 ADD、AND、CMP、MOV、MUL、OR、 SUB 指令的源操作数。

• #IMM:5
#IMM 所示的 5 位立即数为运算对象。此寻址方式用于指定位操作指令 （BCLR、BMCnd、 BNOT、 BSET、 BTST）的位号、算术 / 逻辑运算指令 （SHAR、SHLL、 SHLR）的移位位数以及算术 / 逻辑运算指令 （ROTL、 ROTR）的循环位数。</td></tr>
<tr><td>#IMM:8
#SIMM:8
#UIMM:8
#IMM:16
#SIMM:16
#SIMM:24
#IMM:32</td><td>立即数指定的值为运算对象。但是，将 #UIMM 指定的立即数进行零扩展，扩展为处理长度的结果为运算对象，将 #SIMM 指定的立即数进行符号扩展，扩展为处理长度的结果为运算对象。 #IMM:n、 #UIMM:n、 #SIMM:n 表示 n 位长的立即数。
IMM 的范围请参照 “2.2.1 IMM 的范围”。</td><td> 
</td></tr>
</table>

寄存器直接		
Rn (Rn= R0 ～ R15)	指定的寄存器为运算对象。 或者在 JMP 指令和 JSR 指令的情况下，将 Rn 的值传送到程序计数器 （PC）。 有效地址的范围为 00000000h ～ FFFFFFFFh。 能指定 Rn(Rn=R0 ～ R15)	
寄存器间接		
[Rn] (Rn= R0 ～ R15)	寄存器的值为运算对象的有效地址。有效地址的范围为 00000000h ～ FFFFFFFFh。 能指定 [Rn](Rn=R0 ～ R15)。	
寄存器相对		
dsp:5[Rn] (Rn= R0 ～ R7) dsp:8[Rn] (Rn= R0 ～ R15) dsp:16[Rn] (Rn= R0 ～ R15)	将位移量 （dsp）的值进行零扩展，在扩展为 32 位后按照规则 （参照右图），将 1/2/4 倍的值和寄存器的值进行加法运算，运算结果的低 32 位为运算对象的有效地址。有效地址的范围为 00000000h ～ FFFFFFFFh。 dsp:n 表示 n 位长的位移量。 能指定 dsp:5[Rn] (Rn=R0 ～ R7)、dsp:8[Rn] (Rn=R0 ～ R15)、dsp:16[Rn] (Rn=R0 ～ R15)。 dsp:5[Rn] (Rn=R0 ～ R7) 只能用于 MOV 指令和 MOVU 指令。	
后增寄存器间接		
[Rn+] (Rn= R0 ～ R15)	寄存器的值为运算对象的有效地址。有效地址的范围为 00000000h ～ FFFFFFFFh。在运算后，寄存器的值加上分别与长度说明符 .B/.W/.L 对应的 1/2/4。此寻址方式用于 MOV 指令和 MOVU 指令。	
先减寄存器间接		
[–Rn] (Rn= R0 ～ R15)	寄存器的值减去分别与长度说明符 .B/.W/.L 对应的 1/2/4。减法运算后的值为运算对象的有效地址。有效地址的范围为 00000000h ～ FFFFFFFFh。此寻址方式用于 MOV 指令和 MOVU 指令。	
带变址的寄存器间接		
[Ri,Rb] (Ri,Rb= R0 ～ R15)	将变址寄存器 （Ri）的值乘上分别与长度说明符 .B/.W/.L 对应的 1/2/4 倍，结果和基址寄存器 （Rb）的值相加，运算结果的低 32 位为运算对象的有效地址。有效地址的范围为 00000000h ～ FFFFFFFFh。此寻址方式用于 MOV 指令和 MOVU 指令。	

控制寄存器直接		
PC ISP USP INTB PSW BPC BPSW FINTV FPSW	指定的控制寄存器为运算对象。此寻址方式用于 MVFC、 MVTC、 POPC、 PUSHC 指令。 只能将 PC 指定为 MVFC 指令和 PUSHC 指令的 src。	
PSW 直接		
C Z S O I U	指定的标志或者位为运算对象。此寻址方式用于 CLRPSW 指令和 SETPSW 指令。	
程序计数器相对		
pcdsp:3	在转移距离说明符为 “.S” 时，将程序计数器 （PC）的值和位移量 （pcdsp）的值进行不带符号的加法运算，运算结果的低 32 位为有效地址。转移的范围为 3 ～ 10。有效地址的范围为 00000000h ～ FFFFFFFFh。此寻址方式用于 B*Cnd* 指令 （只 *Cnd*==EQ/Z、NE/NZ）和 BRA 指令。	

程序计数器相对		
pcdsp:8 pcdsp:16 pcdsp:24	在转移距离说明符为 “.B”、“.W” 或者 “.A” 时，将程序计数器 （PC）的值和位移量 （pcdsp）的值进行带符号的加法运算，运算结果为有效地址。 pcdsp 的范围为： 当为 “.B” 时：–128 ≤ pcdsp:8 ≤ 127 当为 “.W” 时： –32768 ≤ pcdsp:16 ≤ 32767 当为 “.A” 时： –8388608 ≤ pcdsp:24 ≤ 8388607 有效地址的范围为 00000000h ～ FFFFFFFFh。 此寻址方式在为 “.B” 时用于 B*Cnd* 指令和 BRA 指令；在为 “.W” 时用于 B*Cnd* （只 *Cnd*==EQ/Z、NE/NZ）、BRA、BSR 指令；在为 “.A” 时用于 BRA 指令和 BSR 指令。	
Rn (Rn= R0 ～ R15)	将程序计数器 （PC）的值和 Rn 的值进行带符号的加法运算，运算结果为有效地址。 Rn 值的范围为 –2147483648 ～ 2147483647。 有效地址的范围为 00000000h ～ FFFFFFFFh。 此寻址方式用于 BRA(.L) 指令和 BSR(.L) 指令。	

2.2.1 IMM 的范围

IMM 的范围如表 2.1 所示。

在 “3.2 指令的详细说明” 的各指令中没有特别记述的情况下， IMM 的范围如下：

表 2.1 IMM 的范围

IMM	10 进制记述	16 进制记述
IMM:1	1 ～ 2	1h ～ 2h
IMM:3	0 ～ 7	0h ～ 7h
IMM:4	0 ～ 15	0h ～ 0Fh
UIMM:4	0 ～ 15	0h ～ 0Fh
IMM:5	0 ～ 31	0h ～ 1Fh
IMM:8	–128 ～ 255	–80h ～ 0FFh
UIMM:8	0 ～ 255	0h ～ 0FFh
SIMM:8	–128 ～ 127	–80h ～ 7Fh
IMM:16	–32768 ～ 65535	–8000h ～ 0FFFFh
SIMM:16	–32768 ～ 32767	–8000h ～ 7FFFh
SIMM:24	–8388608 ～ 8388607	–800000h ～ 7FFFFFh
IMM:32	–2147483648 ～ 4294967295	–80000000h ～ 0FFFFFFFFh

注 . 1. 本公司的 “RX 族的汇编程序” 将 IMM 转换为最佳位长的指令码。

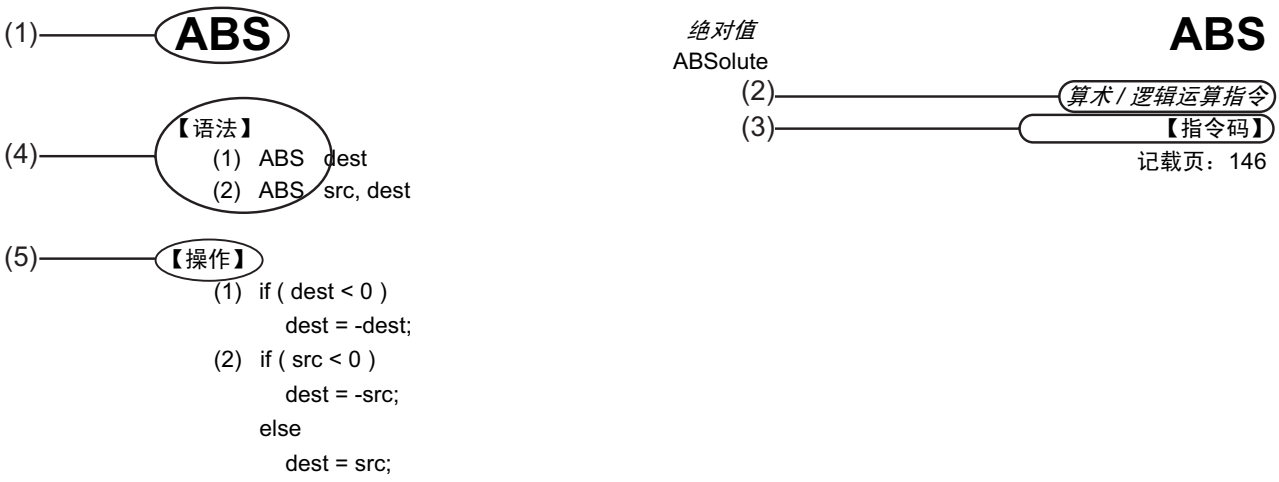
2. 本公司的 “RX 族的汇编程序” 的 16 进制记述也能用 32 位记述。
例如. 能将10进制记述的 “–127” 和16进制记述的 “–7Fh” 记述为 “0FFFFFF81h”。

3. 有关 INT 指令和 RTSD 指令的 IMM 范围，请参照 “3.2 指令的详细说明” 的对应指令。

3. 指令

3.1 本章的阅读方法

在本章中，按各指令对语法、操作、功能、可选择的 src/dest、标志的变化和记述例子进行说明。
通过以下的例子说明本章的阅读方法。



标志	变化	条件
C	—	
Z	○	当运算后的 dest 是 “0” 时为 “1”，否则为 “0”。
S	○	当运算后的 dest 的 MSB 是 “1” 时为 “1”，否则为 “0”。
O	○	1. 当运算前的 dest 是 “80000000h” 时为 “1”，否则为 “0”。 2. 当运算前的 src 是 “80000000h” 时为 “1”，否则为 “0”。

(8) 【指令格式】

语法	处理长度	对象		代码长度（字节）
		src	dest	
(1) ABS dest	L	—	Rd	2
(2) ABS src, dest	L	Rs	Rd	3

(1) 助记符

表示本页中说明的助记符。中间记载了指令的简单操作和全名。

(2) 指令的种类

表示指令的种类。

(3) 指令码

表示指令码的记载页。
有关指令码，请参照此页。

(4) 语法

用符号表示指令的语法。

(a) 助记符

记述助记符。

(b) 长度说明符 .size

对于传送指令、一部分字符串的操作指令和 RMPA 指令，能在助记符的词尾指定长度说明符。通过长度说明符指定要处理的数据长度如下：

.B	字节 （8 位）
.W	字 （16 位）
.L	长字 （32 位）

(c) 操作数 src、 dest

记述操作数。

src	源操作数
dest	目标操作数

(5) 操作

表示指令的操作。操作的记述遵循 C 语言的表述方法。

(a) 数据类型

signed char	带符号的字节 （8 位）整数
signed short	带符号的字 （16 位）整数
signed long	带符号的长字 （32 位）整数
signed long long	带符号的双倍长字 （64 位）整数
unsigned char	不带符号的字节 （8 位）整数
unsigned short	不带符号的字 （16 位）整数
unsigned long	不带符号的长字 （32 位）整数
unsigned long long	不带符号的双倍长字 （64 位）整数
float	单精度浮点数

(b) 伪函数

register(n)	： 返回寄存器 Rn， n 是寄存器号 （n：0 ～ 15）。
register_num(Rn)	： 返回 Rn 的寄存器号 n。

(c) 特殊记述

Rn[i+7:i]	：表示 Rn 的 bit(i+7) ～ biti 的不带符号的字节整数。 （n：0 ～ 15， i：24、 16、 8、 0）
Rm:Rn	：表示 2 个寄存器相连的虚拟 64 位寄存器。 （m、 n：0 ～ 15， Rm 分配到 bit63 ～ bit32， Rn 分配到 bit31 ～ bit0）
Rl:Rm:Rn	：表示 3 个寄存器相连的虚拟 96 位寄存器。 （l、 m、 n：0 ～ 15， Rl 分配到 bit95 ～ bit64， Rm 分配到 bit63 ～ bit32，Rn 分配到 bit31 ～ bit0）
{byte3, byte2, byte1, byte0}	：表示 4 个不带符号的字节整数相连的不带符号的长字整数。

(6) 功能

说明指令的功能和注意事项。

(7) 标志的变化

表示 PSW 的标志 （O、 S、 Z、 C）的变化。

在浮点指令的情况下，也表示 FPSW 的标志（FX、FU、FZ、FO、FV、CE、CX、CU、CZ、CO、CV）的变化。

表中所示符号的含义如下：

—：不变。

○：因条件而变。

(8) 指令格式

表示指令的格式。

【指令格式】

语法	处理长度	对象			代码长度（字节）
		src	src2	dest	
(1) AND src, dest	L	#UIMM:4	—	Rd	2
	L	#SIMM:8	—	Rd	3
	L	#SIMM:16	—	Rd	4
	L	#SIMM:24	—	Rd	5
	L	#IMM:32	—	Rd	6
	L	Rs	—	Rd	2
	L	[Rs].memex	—	Rd	2 (memex == UB) 3 (memex != UB)
	L	dsp:8[Rs].memex（注）	—	Rd	3 (memex == UB) 4 (memex != UB)
	L	dsp:16[Rs].memex（注）	—	Rd	4 (memex == UB) 5 (memex !=UB)
(2) AND src, src2, dest	L	Rs	Rs2	Rd	3

【指令格式】

语法	处理长度	对象		代码长度（字节）
		src	dest（注）	
MVTC src, dest	L	#SIMM:8	Rx	4
	L	#SIMM:16	Rx	5
	L	#SIMM:24	Rx	6
	L	#IMM:32	Rx	7
	L	Rs	Rx	3

【指令格式】

语法	对象	代码长度（字节）
	dest	
SETPSW dest	flag	2

(a) 寄存器

在没有特别要求的情况下， Rs、 Rs2、 Rd、 Rd2、 Ri、 Rb 能指定 R0 ～ R15。

(b) 控制寄存器

Rx 能指定 PC、 ISP、 USP、 INTB、 PSW、 BPC、 BPSW、 FINTV、 FPSW。
只能将 PC 指定为 MVFC 指令和 PUSHC 指令的 src。

(c) 标志和位

能给 flag 指定 PSW 的位 （U、 I）和标志 （O、 S、 Z、 C）。

(d) 立即数

#IMM:n、#UIMM:n、#SIMM:n 表示 n 位长的立即数。在需要扩展时，将 UIMM 进行零扩展，将 SIMM 进行符号扩展。

(e) 给存储器操作数附加的长度扩展说明符 .memex

表示存储器操作数的长度和扩展方法。根据长度说明符指定的扩展方法，在扩展为处理长度后处理各指令。

memex	长度	扩展方法
B	字节	符号扩展
UB	字节	零扩展
W	字	符号扩展
UW	字	零扩展
L	长字	无

如果省略，位操作指令就作为字节进行处理，其他指令作为长字进行处理。

(f) 处理长度

处理长度表示 CPU 内部的传送长度或者运算长度。

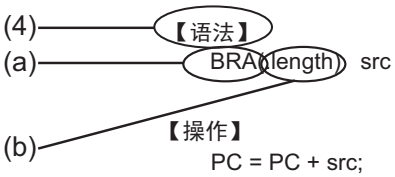
(9) 记述例子

表示指令的记述例子。

通过以下 BRA 指令的例子表示 B*Cnd*、 BRA、 BSR 指令的语法：

BRA *相对无条件的转移* BRanch Always **BRA**

转移指令
【指令码】
记载页：157



【操作】
PC = PC + src;

【功能】
- 向src所示的转移目标进行相对转移。

【标志的变化】
- 标志不变。

【指令格式】

语法	length	对象		代码长度（字节）
		src	pcdsp/Rs 的范围	
BRA(.length) src	S	pcdsp:3	3 ≤ pcdsp ≤ 10	1
	B	pcdsp:8	–128 ≤ pcdsp ≤ 127	2
	W	pcdsp:16	–32768 ≤ pcdsp ≤ 32767	3
	A	pcdsp:24	–8388608 ≤ pcdsp ≤ 8388607	4
	L	Rs	–2147483648 ≤ Rs ≤ 2147483647	2

【记述例子】
BRA label1
BRA.A label2
BRA R1
BRA.L R2

注 . 本公司的 “RX 族的汇编程序” 的位移量的值 （pcdsp:3、 pcdsp:8、 pcdsp:16、 pcdsp:24）必须指定转移目标的标号或者有效地址。在指令码 （pcdsp）里填入指定的地址减去指令的分配地址后的值。

【记述例子】
BRA label
BRA 1000h

（4）语法

用符号表示指令的语法。

(a) 助记符

记述助记符。

(b) 转移距离说明符 .length

对于转移跳转指令，能在助记符之后指定转移距离说明符。通过转移距离说明符指定的转移相对距离的有效位数如下：

.S 表示 3 位 PC 前方相对，有效值为 3 ～ 10。
.B 表示 8 位 PC 相对，有效值为 –128 ～ 127。
.W 表示 16 位 PC 相对，有效值为 –32768 ～ 32767。
.A 表示 24 位 PC 相对，有效值为 –8388608 ～ 8388607。
.L 表示 32 位 PC 相对，有效值为 –2147483648 ～ 2147483647。

3.2 指令的详细说明

从下一页开始详细说明 RX 族的各指令。

ABS

绝对值
ABSolute

ABS

算术 / 逻辑运算指令

【指令码】
记载页：146

【语法】
(1) ABS dest
(2) ABS src, dest

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

【功能】
1. 取dest的绝对值，其结果保存到dest。
2. 取src的绝对值，其结果保存到dest。

【标志的变化】

标志	变化	条件
C	—	
Z	○	当运算后的 dest 是 “0” 时为 “1”，否则为 “0”。
S	○	当运算后的 dest 的 MSB 是 “1” 时为 “1”，否则为 “0”。
O	○	1. 当运算前的 dest 是 “80000000h” 时为 “1”，否则为 “0”。 2. 当运算前的 src 是 “80000000h” 时为 “1”，否则为 “0”。

【指令格式】

语法	处理长度	对象		代码长度（字节）
		src	dest	
(1) ABS dest	L	—	Rd	2
(2) ABS src, dest	L	Rs	Rd	3

【记述例子】
```
ABS  R2
ABS  R1, R2
```

ADC

带进位的加法运算
ADd with Carry

ADC

算术 / 逻辑运算指令

【语法】
ADC src, dest

【指令码】
记载页：147

【操作】
dest = dest + src + C;

【功能】
- 将dest、 src和C标志相加，其结果保存到dest。

【标志的变化】

标志	变化	条件
C	○	当不带符号的运算发生上溢时为 “1”，否则为 “0”。
Z	○	当运算后的 dest 是 “0”时为 “1”，否则为 “0”。
S	○	当运算后的 dest 的 MSB 是 “1”时为 “1”，否则为 “0”。
O	○	当带符号的运算发生上溢时为 “1”，否则为 “0”。

【指令格式】

语法	处理长度	对象		代码长度（字节）
		src	dest	
ADC src, dest	L	#SIMM:8	Rd	4
	L	#SIMM:16	Rd	5
	L	#SIMM:24	Rd	6
	L	#IMM:32	Rd	7
	L	Rs	Rd	3
	L	[Rs].L	Rd	4
	L	dsp:8[Rs].L（注）	Rd	5
	L	dsp:16[Rs].L（注）	Rd	6

注 . 本公司的 “RX 族的汇编程序”的位移量的值 （dsp:8、 dsp:16）必须指定为 4 的倍数。能给 dsp:8 指定 0 ～ 1020（255×4），并且能给 dsp:16 指定 0 ～ 262140 （65535×4）。在指令码里填入位移量除 4 后的值。

【记述例子】
ADC #127, R2
ADC R1, R2
ADC [R1], R2

ADD

不带进位的加法运算
ADD

ADD

算术 / 逻辑运算指令

【语法】
(1) ADD src, dest
(2) ADD src, src2, dest

【指令码】
记载页：148

【操作】
(1) dest = dest + src;
(2) dest = src + src2;

【功能】
1. 将dest和src相加，其结果保存到dest。
2. 将src和src2相加，其结果保存到dest。

【标志的变化】

标志	变化	条件
C	○	当不带符号的运算发生上溢时为 “1”，否则为 “0”。
Z	○	当运算后的 dest 是 “0” 时为 “1”，否则为 “0”。
S	○	当运算后的 dest 的 MSB 是 “1” 时为 “1”，否则为 “0”。
O	○	当带符号的运算发生上溢时为 “1”，否则为 “0”。

【指令格式】

语法	处理长度	对象			代码长度（字节）
		src	src2	dest	
(1) ADD src, dest	L	#UIMM:4	—	Rd	2
	L	#SIMM:8	—	Rd	3
	L	#SIMM:16	—	Rd	4
	L	#SIMM:24	—	Rd	5
	L	#IMM:32	—	Rd	6
	L	Rs	—	Rd	2
	L	[Rs].memex	—	Rd	2 (memex == UB) 3 (memex != UB)
	L	dsp:8[Rs].memex （注）	—	Rd	3 (memex == UB) 4 (memex != UB)
	L	dsp:16[Rs].memex （注）	—	Rd	4 (memex == UB) 5 (memex != UB)
(2) ADD src, src2, dest	L	#SIMM:8	Rs	Rd	3
	L	#SIMM:16	Rs	Rd	4
	L	#SIMM:24	Rs	Rd	5
	L	#IMM:32	Rs	Rd	6
	L	Rs	Rs2	Rd	3

注 . 本公司的 “RX 族的汇编程序” 的位移量的值 （dsp:8、 dsp:16）必须在长度扩展说明符为 “.W” 或者 “.UW” 时指定为 2 的倍数，在长度扩展说明符为 “.L” 时指定为 4 的倍数。当长度扩展说明符为 “.W” 或者 “.UW” 时，能给 dsp:8 指定 0 ～ 510 （255×2）；当长度扩展说明符为 “.L” 时，能给 dsp:8 指定 0 ～ 1020 （255×4）。当长度扩展说明符为 “.W” 或者 “.UW” 时，能给 dsp:16 指定 0 ～ 131070 （65535×2）；当长度扩展说明符为 “.L” 时，能给 dsp:16 指定 0 ～ 262140 （65535×4）。在指令码里填入位移量除 2 或者除 4 后的值。

【记述例子】

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

AND

逻辑与
AND

AND

算术 / 逻辑运算指令

【语法】
(1) AND src, dest
(2) AND src, src2, dest

【指令码】
记载页：150

【操作】
(1) dest = dest & src;
(2) dest = src & src2;

【功能】
1. 取dest和src的逻辑与，其结果保存到dest。
2. 取src和src2的逻辑与，其结果保存到dest。

【标志的变化】

标志	变化	条件
C	—	
Z	○	当运算后的 dest 是 “0” 时为 “1”，否则为 “0”。
S	○	当运算后的 dest 的 MSB 是 “1” 时为 “1”，否则为 “0”。
O	—	

【指令格式】

语法	处理长度	对象			代码长度（字节）
		src	src2	dest	
(1) AND src, dest	L	#UIMM:4	—	Rd	2
	L	#SIMM:8	—	Rd	3
	L	#SIMM:16	—	Rd	4
	L	#SIMM:24	—	Rd	5
	L	#IMM:32	—	Rd	6
	L	Rs	—	Rd	2
	L	[Rs].memex	—	Rd	2 (memex == UB) 3 (memex != UB)
	L	dsp:8[Rs].memex（注）	—	Rd	3 (memex == UB) 4 (memex != UB)
	L	dsp:16[Rs].memex（注）	—	Rd	4 (memex == UB) 5 (memex !=UB)
(2) AND src, src2, dest	L	Rs	Rs2	Rd	3

注 . 本公司的 “RX 族的汇编程序” 的位移量的值 （dsp:8、 dsp:16）必须在长度扩展说明符为 “.W” 或者 “.UW” 时指定为 2 的倍数，在长度扩展说明符为 “.L” 时指定为 4 的倍数。当长度扩展说明符为 “.W” 或者 “.UW” 时，能给 dsp:8 指定 0 ～ 510 （255×2）；当长度说明符为 “.L” 时，能给 dsp:8 指定 0 ～ 1020 （255×4）；当长度扩展说明符为 “.W” 或者 “.UW” 时，能给 dsp:16 指定 0 ～ 131070 （65535×2）；当长度扩展说明符为 “.L” 时，能给 dsp:16 指定 0 ～ 262140 （65535×4）。在指令码里填入位移量除 2 或者除 4 后的值。

【记述例子】

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

BCLR

位清除
Bit CLeaR

BCLR

位操作指令

【语法】
BCLR src, dest

【指令码】
记载页：152

【操作】

(1) dest 为存储器的情况
unsigned char dest;
dest & = ~(1 << (src & 7));

(2) dest 为寄存器的情况
register unsigned long dest;
dest & = ~(1 << (src & 31));

【功能】

- 将src指定的dest的位置 “0”。
- src的IMM的值为位号。
  IMM:3的范围为0≤IMM:3≤7。
  IMM:5的范围为0≤IMM:5≤31。

【标志的变化】

- 标志不变。

【指令格式】

语法	处理长度	对象		代码长度（字节）
		src	dest	
(1) BCLR src, dest	B	#IMM:3	[Rd].B	2
	B	#IMM:3	dsp:8[Rd].B	3
	B	#IMM:3	dsp:16[Rd].B	4
	B	Rs	[Rd].B	3
	B	Rs	dsp:8[Rd].B	4
	B	Rs	dsp:16[Rd].B	5
(2) BCLR src, dest	L	#IMM:5	Rd	2
	L	Rs	Rd	3

【记述例子】

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

BCnd

相对条件转移
Branch Conditionally

BCnd

转移指令

【语法】
BC*nd*(.length) src

【指令码】
记载页：154

【操作】
if (*Cnd*)
PC = PC + src;

【功能】
- 判断*Cnd*所示条件的真假值，向src所示的转移目标进行相对转移。真时转移，假时不转移。
- BC*nd*有以下的种类：

BC*nd*	条件		式	BC*nd*	条件		式
BGEU, BC	C == 1	大于等于 / C 标志为 “1”	⩽	BLTU, BNC	C == 0	小于 / C 标志为 “0”	⩾
BEQ, BZ	Z == 1	等于 / Z 标志为 “1”	=	BNE, BNZ	Z == 0	不等于 / Z 标志为 “0”	≠
BGTU	C & ~Z == 1	大于	<	BLEU	C & ~Z == 0	小于等于	⩾
BPZ	S == 0	正或者零	0 ⩽	BN	S == 1	负	0 >
BGE	S ^ O == 0	等于或者带符号的 大于	⩽	BLE	(S ^ O) | Z == 1	等于或者带符号的 小于	⩾
BGT	(S ^ O) | Z == 0	带符号的大于	<	BLT	S ^ O == 1	带符号的小于	>
BO	O == 1	O 标志为 “1”		BNO	O == 0	O 标志为 “0”	

【标志的变化】
- 标志不变。

【指令格式】

语法	length	对象		代码长度 （字节）
		src	pcdsp 的范围	
(1) BEQ.S src	S	pcdsp:3	3 ⩽ pcdsp ⩽ 10	1
(2) BNE.S src	S	pcdsp:3	3 ⩽ pcdsp ⩽ 10	1
(3) BC*nd*.B src	B	pcdsp:8	–128 ⩽ pcdsp ⩽ 127	2
(4) BEQ.W src	W	pcdsp:16	–32768 ⩽ pcdsp ⩽ 32767	3
(5) BNE.W src	W	pcdsp:16	–32768 ⩽ pcdsp ⩽ 32767	3

【记述例子】
```
BC   label1
BC.B   label2
```

注 . 本公司的 “RX 族的汇编程序” 的位移量的值 （pcdsp:3、 pcdsp:8、 pcdsp:16）必须指定转移目标的标号或者有效地址。在指令码 （pcdsp）里填入指定的地址减去指令的分配地址后的值。

【记述例子】
```
BC   label
BC   1000h
```

BM*Cnd*

条件位传送
Bit Move Conditional

BM*Cnd*

位操作指令

【语法】
BM*Cnd* src, dest

【指令码】
记载页：155

【操作】

(1) dest 为存储器的情况

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

(2) dest 为寄存器的情况

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

【功能】

- 将*Cnd*所示条件的真假值传送到src指定的dest的位。真时传送 “1”，假时传送 “0”。
- BM*Cnd*有以下的种类。

BM*Cnd*	条件		式	BM*Cnd*	条件		式
BMGEU, BMC	C == 1	大于等于 / C 标志为 “1”	≤	BMLTU, BMNC	C == 0	小于 / C 标志为 “0”	＞
BMEQ, BMZ	Z == 1	等于 / Z 标志为 “1”	＝	BMNE, BMNZ	Z == 0	不等于 / Z 标志为 “0”	≠
BMGTU	C & ~Z == 1	大于	＜	BMLEU	C & ~Z == 0	小于等于	≥
BMPZ	S == 0	正或者零	0 ≤	BMN	S == 1	负	0 ＞
BMGE	S ^ O == 0	等于或者带符号的大于	≤	BMLE	(S ^ O) | Z == 1	等于或者带符号的小于	≥
BMGT	(S ^ O) | Z == 0	带符号的大于	＜	BMLT	S ^ O == 1	带符号的小于	＞
BMO	O == 1	O 标志为 “1”		BMNO	O == 0	O 标志为 “0”	

- src的IMM的值为位号。
  IMM:3的范围为0≤IMM:3≤7。
  IMM:5的范围为0≤IMM:5≤31。

【标志的变化】

- 标志不变。

【指令格式】

语法	处理长度	对象		代码长度（字节）
		src	dest	
(1) BM*Cnd* src, dest	B	#IMM:3	[Rd].B	3
	B	#IMM:3	dsp:8[Rd].B	4
	B	#IMM:3	dsp:16[Rd].B	5
(2) BM*Cnd* src, dest	L	#IMM:5	Rd	3

【记述例子】

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

BNOT

位取反
Bit NOT

BNOT

位操作指令

【语法】
BNOT src, dest

【指令码】
记载页：156

【操作】

(1) dest 为存储器的情况

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

(2) dest 为寄存器的情况

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

【功能】

- 将src指定的dest的位值取反，其结果保存到原来的位。
- src的IMM的值为位号。
  IMM:3的范围为0≤IMM:3≤7。
  IMM:5的范围为0≤IMM:5≤31。

【标志的变化】

- 标志不变。

【指令格式】

语法	处理长度	对象		代码长度（字节）
		src	dest	
(1) BNOT src, dest	B	#IMM:3	[Rd].B	3
	B	#IMM:3	dsp:8[Rd].B	4
	B	#IMM:3	dsp:16[Rd].B	5
	B	Rs	[Rd].B	3
	B	Rs	dsp:8[Rd].B	4
	B	Rs	dsp:16[Rd].B	5
(2) BNOT src, dest	L	#IMM:5	Rd	3
	L	Rs	Rd	3

【记述例子】

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

BRA

相对无条件转移
BRanch Always

BRA

转移指令

【语法】
BRA(.length) src

【指令码】
记载页：157

【操作】
PC = PC + src;

【功能】
- 向src所示的转移目标进行相对转移。

【标志的变化】
- 标志不变。

【指令格式】

语法	length	对象		代码长度（字节）
		src	pcdsp/Rs 的范围	
BRA(.length) src	S	pcdsp:3	3 ≤ pcdsp ≤ 10	1
	B	pcdsp:8	–128 ≤ pcdsp ≤ 127	2
	W	pcdsp:16	–32768 ≤ pcdsp ≤ 32767	3
	A	pcdsp:24	–8388608 ≤ pcdsp ≤ 8388607	4
	L	Rs	–2147483648 ≤ Rs ≤ 2147483647	2

【记述例子】
```
BRA   label1
BRA.A   label2
BRA   R1
BRA.L   R2
```

注 . 本公司的 “RX 族的汇编程序” 的位移量的值 （pcdsp:3、 pcdsp:8、 pcdsp:16、 pcdsp:24）必须指定转移目标的标号或者有效地址。在指令码 （pcdsp）里填入指定的地址减去指令的分配地址后的值。

【记述例子】
```
BRA   label
BRA   1000h
```

BRK

无条件陷阱
BReaK

BRK

系统操作指令

【语法】
BRK

【指令码】
记载页：158

【操作】

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

【功能】

- 产生序号0的无条件陷阱。
- 转移到管理模式， PSW的PM位为 “0”。
- PSW的U位和I位为 “0”。
- 将执行的BRK指令的下一条指令地址压栈。

【标志的变化】

- 标志不变。
- 将执行指令前的PSW压栈。

【指令格式】

语法	代码长度 （字节）
BRK	1

【记述例子】

```
BRK
```

BSET

位置位
Bit SET

BSET

位操作指令

【语法】
BSET src, dest

【指令码】
记载页：158

【操作】

(1) dest 为存储器的情况

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

(2) dest 为寄存器的情况

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

【功能】

- 将src指定的dest的位置 “1”。
- src的IMM的值为位号。
  IMM:3的范围为0≤IMM:3≤7。
  IMM:5的范围为0≤IMM:5≤31。

【标志的变化】

- 标志不变。

【指令格式】

语法	处理长度	对象		代码长度（字节）
		src	dest	
(1) BSET src, dest	B	#IMM:3	[Rd].B	2
	B	#IMM:3	dsp:8[Rd].B	3
	B	#IMM:3	dsp:16[Rd].B	4
	B	Rs	[Rd].B	3
	B	Rs	dsp:8[Rd].B	4
	B	Rs	dsp:16[Rd].B	5
(2) BSET src, dest	L	#IMM:5	Rd	2
	L	Rs	Rd	3

【记述例子】

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

BSR

相对子程序转移
Branch to SubRoutine

BSR

转移指令

【语法】
BSR(.length) src

【指令码】
记载页：160

【操作】
SP = SP - 4;
*SP = (PC + n);（注）
PC = PC + src;

注 . 1. (PC+n) 是 BSR 指令的下一条指令的地址。
2. n 为代码长度。有关代码长度，请参照【指令格式】。

【功能】
- 向src所示的转移目标进行相对转移。

【标志的变化】
- 标志不变。

【指令格式】

语法	length	对象		代码长度（字节）
		src	pcdsp/Rs 的范围	
BSR(.length) src	W	pcdsp:16	–32768 ≤ pcdsp ≤ 32767	3
	A	pcdsp:24	–8388608 ≤ pcdsp ≤ 8388607	4
	L	Rs	–2147483648 ≤ Rs ≤ 2147483647	2

【记述例子】
BSR label1
BSR.A label2
BSR R1
BSR.L R2

注 . 本公司的“RX 族的汇编程序”的位移量的值（pcdsp:16、pcdsp:24）必须指定转移目标的标号或者有效地址。在指令码（pcdsp）里填入指定的地址减去指令的分配地址后的值

【记述例子】
BSR label
BSR 1000h

BTST

位测试
Bit TeST

BTST

位操作指令

【语法】
BTST src, src2

【指令码】
记载页：161

【操作】

(1) src2 为存储器的情况

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

(2) src2 为寄存器的情况

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

【功能】

- 将src指定的src2的位值取反，结果传送到Z标志，并且将src指定的src2的位值传送到C标志。
- src的IMM的值为位号。
  IMM:3的范围为0≤IMM:3≤7。
  IMM:5的范围为0≤IMM:5≤31。

【标志的变化】

标志	变化	条件
C	○	当指定位是 “1” 时为 “1”，否则为 “0”。
Z	○	当指定位是 “0” 时为 “1”，否则为 “0”。
S	—	
O	—	

【指令格式】

语法	处理长度	对象		代码长度（字节）
		src	src2	
(1) BTST src, src2	B	#IMM:3	[Rs2].B	2
	B	#IMM:3	dsp:8[Rs2].B	3
	B	#IMM:3	dsp:16[Rs2].B	4
	B	Rs	[Rs2].B	3
	B	Rs	dsp:8[Rs2].B	4
	B	Rs	dsp:16[Rs2].B	5
(2) BTST src, src2	L	#IMM:5	Rs2	2
	L	Rs	Rs2	3

【记述例子】

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

CLRPSW

PSW 的标志和位的清除
CLeaR flag in PSW

CLRPSW

系统操作指令

【语法】
CLRPSW dest

【指令码】
记载页：162

【操作】
dest = 0;

【功能】
- 将dest指定的O、 S、 Z、 C标志或者U位和I位置 “0”。
- 在用户模式中，忽视写U位和I位；在管理模式中，能写全部标志和位。

【标志的变化】

标志	变化	条件
C	（注）	
Z	（注）	
S	（注）	
O	（注）	

注 . 指定的标志为 “0”。

【指令格式】

语法	对象	代码长度
	dest	（字节）
CLRPSW dest	flag	2

【记述例子】
```
CLRPSW C
CLRPSW Z
```

CMP

比较
CoMPare

CMP

算术 / 逻辑运算指令

【语法】
CMP src, src2

【指令码】
记载页：163

【操作】
src2 - src;

【功能】
- 根据src2减去src的运算结果， PSW的各标志发生变化。

【标志的变化】

标志	变化	条件
C	○	当不带符号的运算不发生上溢时为 “1”，否则为 “0”。
Z	○	当运算结果是 “0” 时为 “1”，否则为 “0”。
S	○	当运算结果的 MSB 是 “1” 时为 “1”，否则为 “0”。
O	○	当带符号的运算发生上溢时为 “1”，否则为 “0”。

【指令格式】

语法	处理长度	对象		代码长度（字节）
		src	src2	
CMP src, src2	L	#UIMM:4	Rs	2
	L	#UIMM:8 （注 1）	Rs	3
	L	#SIMM:8 （注 1）	Rs	3
	L	#SIMM:16	Rs	4
	L	#SIMM:24	Rs	5
	L	#IMM:32	Rs	6
	L	Rs	Rs2	2
	L	[Rs].memex	Rs2	2 (memex == UB) 3 (memex != UB)
	L	dsp:8[Rs].memex （注 2）	Rs2	3 (memex == UB) 4 (memex != UB)
	L	dsp:16[Rs].memex （注 2）	Rs2	4 (memex == UB) 5 (memex != UB)

注 1. 0 ～ 127 的范围总是为零扩展指令码。

注 2. 本公司的 “RX 族的汇编程序” 的位移量的值 （dsp:8、 dsp:16）必须在长度扩展说明符为 “.W” 或者 “.UW” 时指定为 2 的倍数，在长度扩展说明符为 “.L” 时指定为 4 的倍数。当长度扩展说明符为 “.W” 或者 “.UW” 时，能给 dsp:8 指定 0 ～ 510 （255×2）； 当长度扩展说明符为 “.L” 时，能给 dsp:8 指定 0 ～ 1020 （255×4）。当长度扩展说明符为 “.W” 或者 “.UW” 时，能给 dsp:16 指定 0 ～ 131070 （65535×2）； 当长度扩展说明符为 “.L” 时，能给 dsp:16 指定 0 ～ 262140 （65535×4）。在指令码里填入位移量除 2 或者除 4 后的值。

【记述例子】
```
CMP #7, R2
CMP R1, R2
CMP [R1], R2
```

DIV

带符号的除法运算
DIVide

DIV

算术 / 逻辑运算指令

【语法】
DIV src, dest

【指令码】
记载页：164

【操作】
dest = dest / src;

【功能】
- dest除以src （带符号），商保存到dest。将商向零方向舍入。
- 以32位进行运算，并且用32位保存结果。
- 当除数 （src）是 “0”时或者运算结果发生上溢时， dest的值为不定值。

【标志的变化】

标志	变化	条件
C	—	
Z	—	
S	—	
O	○	当除数 （src）是 “0”或者运算是 –2147483648÷(–1) 时为 “1”，否则为 “0”。

【指令格式】

语法	处理长度	对象		代码长度（字节）
		src	dest	
DIV src, dest	L	#SIMM:8	Rd	4
	L	#SIMM:16	Rd	5
	L	#SIMM:24	Rd	6
	L	#IMM:32	Rd	7
	L	Rs	Rd	3
	L	[Rs].memex	Rd	3 (memex == UB) 4 (memex != UB)
	L	dsp:8[Rs].memex （注）	Rd	4 (memex == UB) 5 (memex != UB)
	L	dsp:16[Rs].memex （注）	Rd	5 (memex == UB) 6 (memex != UB)

注 . 本公司的 “RX 族的汇编程序” 的位移量的值 （dsp:8、 dsp:16）必须在长度扩展说明符为 “.W” 或者 “.UW” 时指定为 2 的倍数，在长度扩展说明符为 “.L” 时指定为 4 的倍数。当长度扩展说明符为 “.W” 或者 “.UW” 时，能给 dsp:8 指定 0 ～ 510 （255×2）；当长度扩展说明符为 “.L” 时，能给 dsp:8 指定 0 ～ 1020 （255×4）。当长度扩展说明符为 “.W” 或者 “.UW” 时，能给 dsp:16 指定 0 ～ 131070 （65535×2）；当长度扩展说明符为 “.L” 时，能给 dsp:16 指定 0 ～ 262140 （65535×4）。在指令码里填入位移量除 2 或者除 4 后的值。

【记述例子】
```
DIV  #10, R2
DIV  R1, R2
DIV  [R1], R2
DIV  3[R1].B, R2
```

DIVU

不带符号的除法运算
DIVide Unsigned

DIVU

算术 / 逻辑运算指令

【语法】
DIVU src, dest

【指令码】
记载页：166

【操作】
dest = dest / src;

【功能】
- dest除以src （不带符号），商保存到dest。将商向零方向舍入。
- 以32位进行运算，并且用32位保存结果。
- 当除数 （src）是 “0”时， dest的值为不定值。

【标志的变化】

标志	变化	条件
C	—	
Z	—	
S	—	
O	○	当除数 （src）是 “0”时为 “1”，否则为 “0”。

【指令格式】

语法	处理长度	对象		代码长度（字节）
		src	dest	
DIVU src, dest	L	#SIMM:8	Rd	4
	L	#SIMM:16	Rd	5
	L	#SIMM:24	Rd	6
	L	#IMM:32	Rd	7
	L	Rs	Rd	3
	L	[Rs].memex	Rd	3 (memex == UB) 4 (memex != UB)
	L	dsp:8[Rs].memex （注）	Rd	4 (memex == UB) 5 (memex != UB)
	L	dsp:16[Rs].memex （注）	Rd	5 (memex == UB) 6 (memex != UB)

注 . 本公司的 “RX 族的汇编程序” 的位移量的值 （dsp:8、 dsp:16）必须在长度扩展说明符为 “.W” 或者 “.UW” 时指定为 2 的倍数，在长度扩展说明符为 “.L” 时指定为 4 的倍数。当长度扩展说明符为 “.W” 或者 “.UW” 时，能给 dsp:8 指定 0 ～ 510 （255×2）；当长度扩展说明符为 “.L” 时，能给 dsp:8 指定 0 ～ 1020 （255×4）。当长度扩展说明符为 “.W” 或者 “.UW” 时，能给 dsp:16 指定 0 ～ 131070 （65535×2）；当长度扩展说明符为 “.L” 时，能给 dsp:16 指定 0 ～ 262140 （65535×4）。在指令码里填入位移量除 2 或者除 4 后的值。

【记述例子】
```
DIVU   #10, R2
DIVU   R1, R2
DIVU   [R1], R2
DIVU   3[R1].UB, R2
```

EMUL

带符号的乘法运算
Extended MULtiply, signed

EMUL

算术 / 逻辑运算指令

【语法】
EMUL src, dest

【指令码】
记载页：167

【操作】
dest2:dest = dest * src;

【功能】
- 将dest和src进行带符号的乘法运算。
- src和dest都以32位进行运算，并且用64位将结果保存到寄存器对dest2:dest （R(n+1):Rn）。
- 能给dest指定15种Rn （n：0～14）。

注 . 使用累加器 （ACC），在执行指令后 ACC 的值为不定值。

dest 指定的寄存器	64 位扩展使用的寄存器
R0	R1:R0
R1	R2:R1
R2	R3:R2
R3	R4:R3
R4	R5:R4
R5	R6:R5
R6	R7:R6
R7	R8:R7
R8	R9:R8
R9	R10:R9
R10	R11:R10
R11	R12:R11
R12	R13:R12
R13	R14:R13
R14	R15:R14

【标志的变化】
- 标志不变。

【指令格式】

语法	处理长度	对象		代码长度（字节）
		src	dest	
EMUL src, dest	L	#SIMM:8	Rd(Rd=R0 ～ R14)	4
	L	#SIMM:16	Rd(Rd=R0 ～ R14)	5
	L	#SIMM:24	Rd(Rd=R0 ～ R14)	6
	L	#IMM:32	Rd(Rd=R0 ～ R14)	7
	L	Rs	Rd(Rd=R0 ～ R14)	3
	L	[Rs].memex	Rd(Rd=R0 ～ R14)	3 (memex == UB) 4 (memex != UB)
	L	dsp:8[Rs].memex（注）	Rd(Rd=R0 ～ R14)	4 (memex == UB) 5 (memex != UB)
	L	dsp:16[Rs].memex（注）	Rd(Rd=R0 ～ R14)	5 (memex == UB) 6 (memex != UB)

注 . 本公司的 “RX 族的汇编程序” 的位移量的值 （dsp:8、 dsp:16）必须在长度扩展说明符为 “.W” 或者 “.UW” 时指定为 2 的倍数，在长度扩展说明符为 “.L” 时指定为 4 的倍数。当长度扩展说明符为 “.W” 或者 “.UW” 时，能给 dsp:8 指定 0 ～ 510 （255×2）； 当长度扩展说明符为 “.L” 时，能给 dsp:8 指定 0 ～ 1020 （255×4）。当长度扩展说明符为 “.W” 或者 “.UW” 时，能给 dsp:16 指定 0 ～ 131070 （65535×2）； 当长度扩展说明符为 “.L” 时，能给 dsp:16 指定 0 ～ 262140 （65535×4）。在指令码里填入位移量除 2 或者除 4 后的值。

【记述例子】

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

EMULU

不带符号的乘法运算
Extended MULtiply, Unsigned

EMULU

算术 / 逻辑运算指令

【语法】
EMULU src, dest

【指令码】
记载页：168

【操作】
dest2:dest = dest * src;

【功能】
- 将dest和src进行不带符号的乘法运算。
- src和dest都以32位进行运算，并且用64位将结果保存到寄存器对dest2:dest （R(n+1):Rn）。
- 能给dest指定15种Rn （n：0～14）。

注 . 使用累加器 （ACC），在执行指令后 ACC 的值为不定值。

dest 指定的寄存器	64 位扩展使用的寄存器
R0	R1:R0
R1	R2:R1
R2	R3:R2
R3	R4:R3
R4	R5:R4
R5	R6:R5
R6	R7:R6
R7	R8:R7
R8	R9:R8
R9	R10:R9
R10	R11:R10
R11	R12:R11
R12	R13:R12
R13	R14:R13
R14	R15:R14

【标志的变化】
- 标志不变。

【指令格式】

语法	处理长度	对象		代码长度（字节）
		src	dest	
EMULU src, dest	L	#SIMM:8	Rd(Rd=R0 ～ R14)	4
	L	#SIMM:16	Rd(Rd=R0 ～ R14)	5
	L	#SIMM:24	Rd(Rd=R0 ～ R14)	6
	L	#IMM:32	Rd(Rd=R0 ～ R14)	7
	L	Rs	Rd(Rd=R0 ～ R14)	3
	L	[Rs].memex	Rd(Rd=R0 ～ R14)	3 (memex == UB) 4 (memex != UB)
	L	dsp:8[Rs].memex（注）	Rd(Rd=R0 ～ R14)	4 (memex == UB) 5 (memex != UB)
	L	dsp:16[Rs].memex（注）	Rd(Rd=R0 ～ R14)	5 (memex == UB) 6 (memex != UB)

注 . 本公司的 “RX 族的汇编程序” 的位移量的值 （dsp:8、 dsp:16）必须在长度扩展说明符为 “.W” 或者 “.UW” 时指定为 2 的倍数，在长度扩展说明符为 “.L” 时指定为 4 的倍数。当长度扩展说明符为 “.W” 或者 “.UW” 时，能给 dsp:8 指定 0 ～ 510 （255×2）； 当长度扩展说明符为 “.L” 时，能给 dsp:8 指定 0 ～ 1020 （255×4）。当长度扩展说明符为 “.W” 或者 “.UW” 时，能给 dsp:16 指定 0 ～ 131070 （65535×2）； 当长度扩展说明符为 “.L” 时，能给 dsp:16 指定 0 ～ 262140 （65535×4）。在指令码里填入位移量除 2 或者除 4 后的值。

【记述例子】

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

FADD

浮点加法运算
Floating-point ADD

FADD

浮点运算指令

【语法】
FADD src, dest

【指令码】
记载页：169

【操作】
dest = dest + src;

【功能】
- 将dest保存的单精度浮点数和src保存的单精度浮点数相加，其结果保存到dest。根据FPSW的RM[1:0]位，将结果进行舍入。
- 非规格化数的处理因FPSW的DN位而不同。
- 当持有相反符号的src和dest的和为 “0”时，如果不是向–∞方向舍入的模式，结果就为 “+0”；如果是向–∞方向舍入的模式，结果就为 “–0”。

【标志的变化】

标志	变化	条件
C	—	
Z	○	当运算结果是 “+0”或者 “–0”时为 “1”，否则为 “0”
S	○	当运算结果的符号位 （bit31）是 “1”时为 “1”，是 “0”时为 “0”。
O	—	
CV	○	当发生无效运算时为 “1”，否则为 “0”。
CO	○	当发生上溢时为 “1”，否则为 “0”。
CZ	○	总是为 “0”。
CU	○	当发生下溢时为 “1”，否则为 “0”。
CX	○	当发生精度异常时为 “1”，否则为 “0”。
CE	○	当发生非安装处理异常时为 “1”，否则为 “0”。
FV	○	当发生无效运算时为 “1”，否则不变。
FO	○	当发生上溢时为 “1”，否则不变。
FZ	—	
FU	○	当发生下溢时为 “1”，否则不变。
FX	○	当发生精度异常时为 “1”，否则不变。

注 . 在异常处理允许位 EX、EU、EO、EV 为 “1”时，FX、FU、FO、FV 标志不变。在发生异常处理时，S 标志和 Z 标志不变。

【指令格式】

语法	处理长度	对象		代码长度（字节）
		src	dest	
FADD src, dest	L	#IMM:32	Rd	7
	L	Rs	Rd	3
	L	[Rs].L	Rd	3
	L	dsp:8[Rs].L （注）	Rd	4
	L	dsp:16[Rs].L （注）	Rd	5

注 . 本公司的 “RX 族的汇编程序”的位移量的值 （dsp:8、 dsp:16）必须指定为 4 的倍数。能给 dsp:8 指定 0 ～ 1020 （255×4），并且能给 dsp:16 指定 0 ～ 262140 （65535×4）。在指令码里填入位移量除 4 后的值。

【发生异常】

非安装处理
无效运算
上溢
下溢
精度异常

【记述例子】

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

【操作补充说明】

- 当DN=0和DN=1时， src和dest的值与运算结果的对应如下所示：

DN=0

		src							
		规格化数	+0	–0	+∞	–∞	非规格化数	QNaN	SNaN
dest	规格化数	加法运算							
	+0		+0	（注）		–∞			
	–0		（注）	–0					
	+∞				+∞	无效运算			
	–∞				无效运算	–∞			
	非规格化数						非安装处理		
	QNaN							QNaN	
	SNaN								无效运算

DN=1

		src						
		规格化数	+0、+非规格化数	–0、–非规格化数	+∞	–∞	QNaN	SNaN
dest	规格化数	加法运算	规格化数					
	+0、+非规格化数	规格化数	+0	（注）		–∞		
	–0、–非规格化数		（注）	–0				
	+∞				+∞	无效运算		
	–∞				无效运算	–∞		
	QNaN						QNaN	
	SNaN							无效运算

注 . 在舍入模式为向 –∞ 方向舍入时为 “–0”，否则为 “+0”。

FCMP

浮点比较
Floating-point CoMPare

FCMP

浮点运算指令

【指令码】

记载页：170

【语法】

FCMP src, src2

【操作】

src2 - src;

【功能】

- 将src2保存的单精度浮点数和src保存的单精度浮点数进行比较，标志根据结果而发生变化。
- 非规格化数的处理因FPSW的DN位而不同。

【标志的变化】

标志	变化	条件
C	—	
Z	○	当 src2==src 时为 “1”，否则为 “0”。
S	○	当 src2<src 时为 “1”，否则为 “0”。
O	○	当比较结果不能排序时为 “1”，否则为 “0”。
CV	○	在发生无效运算时为 “1”，否则为 “0”。
CO	○	总是为 “0”。
CZ	○	总是为 “0”。
CU	○	总是为 “0”。
CX	○	总是为 “0”。
CE	○	当发生非安装处理异常时为 “1”，否则为 “0”。
FV	○	当发生无效运算时 “1”，否则不变。
FO	—	
FZ	—	
FU	—	
FX	—	

注 . 在异常处理允许位 EV 为 “1” 时， FV 标志不变。在发生异常处理时， O、 S、 Z 标志不变。

标志 / 条件	O	S	Z
src2>src	“0”	“0”	“0”
src2<src	“0”	“1”	“0”
src2==src	“0”	“0”	“1”
不能排序	“1”	“0”	“0”

【指令格式】

语法	处理长度	对象		代码长度（字节）
		src	src2	
FCMP src, src2	L	#IMM:32	Rs2	7
	L	Rs	Rs2	3
	L	[Rs].L	Rs2	3
	L	dsp:8[Rs].L（注）	Rs2	4
	L	dsp:16[Rs].L（注）	Rs2	5

注 . 本公司的 “RX 族的汇编程序” 的位移量的值 （dsp:8、 dsp:16）必须指定为 4 的倍数。能给 dsp:8 指定 0 ～ 1020（255×4），并且能给 dsp:16 指定 0 ～ 262140 （65535×4）。在指令码里填入位移量除 4 后的值。

【发生异常】

非安装处理

无效运算

【记述例子】

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

【操作补充说明】

- 当DN=0和DN=1时， src和src2的值与运算结果的对应如下所示：
  （>:src2>src， <:src2<src， =:src2==src）

DN=0

		src							
		规格化数	+0	–0	+∞	–∞	非规格化数	QNaN	SNaN
src2	规格化数	比较			<	>			
	+0		=						
	–0								
	+∞	>			=				
	–∞	<				=			
	非规格化数	非安装处理							
	QNaN	不能排序							
	SNaN	无效运算 （不能排序）							

DN=1

		src						
		规格化数	+0、+ 非规格化数	–0、– 非规格化数	+∞	–∞	QNaN	SNaN
src2	规格化数	比较			<	>		
	+0、 + 非规格化数		=					
	–0、 – 非规格化数							
	+∞	>			=			
	–∞	<				=		
	QNaN	不能排序						
	SNaN	无效运算 （不能排序）						

FDIV

浮点除法运算
Floating-point DIVide

FDIV

浮点运算指令

【语法】
FDIV src, dest

【指令码】
记载页：171

【操作】
dest = dest / src;

【功能】
- dest保存的单精度浮点数除以src保存的单精度浮点数，其结果保存到dest。根据FPSW的RM[1:0]位，将结果进行舍入。
- 非规格化数的处理因FPSW的DN位而不同。

【标志的变化】

标志	变化	条件
C	—	
Z	○	当运算结果是 “+0” 或者 “–0” 时为 “1”，否则为 “0”。
S	○	当运算结果的符号位 （bit31）是 “1” 时为 “1”，是 “0” 时为 “0”。
O	—	
CV	○	当发生无效运算时为 “1”，否则为 “0”。
CO	○	当发生上溢时为 “1”，否则为 “0”。
CZ	○	当被零除时为 “1”，否则为 “0”。
CU	○	当发生下溢时为 “1”，否则为 “0”。
CX	○	当发生精度异常时为 “1”，否则为 “0”。
CE	○	当发生非安装处理异常时为 “1”，否则为 “0”。
FV	○	当发生无效运算时为 “1”，否则不变。
FO	○	当发生上溢时为 “1”，否则不变。
FZ	○	当被零除时为 “1”，否则不变。
FU	○	当发生下溢时为 “1”，否则不变。
FX	○	当发生精度异常时为 “1”，否则不变。

注 . 在异常处理允许位 EX、EU、EZ、EO、EV 为 “1” 时，FX、FU、FZ、FO、FV 标志不变。在发生异常处理时，S 标志和 Z 标志不变。

【指令格式】

语法	处理长度	对象		代码长度（字节）
		src	dest	
FDIV src, dest	L	#IMM:32	Rd	7
	L	Rs	Rd	3
	L	[Rs].L	Rd	3
	L	dsp:8[Rs].L （注）	Rd	4
	L	dsp:16[Rs].L （注）	Rd	5

注 . 本公司的 “RX 族的汇编程序” 的位移量的值 （dsp:8、 dsp:16）必须指定为 4 的倍数。能给 dsp:8 指定 0 ～ 1020 （255×4），并且能给 dsp:16 指定 0 ～ 262140 （65535×4）。在指令码里填入位移量除 4 后的值。

【发生异常】

非安装处理
无效运算
上溢
下溢
精度异常
被零除

【记述例子】

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

【操作补充说明】

- 当DN=0和DN=1时， src和dest的值与运算结果的对应如下所示：

DN=0

<table>
<tr><td colspan="2" rowspan="2"></td><td colspan="8">src</td></tr>
<tr><td>规格化数</td><td colspan="2">+0</td><td colspan="2">–0</td><td>+∞</td><td>–∞</td><td>非规格化数</td><td>QNaN</td><td>SNaN</td></tr>
<tr><td rowspan="8">dest</td><td>规格化数</td><td>除法运算</td><td colspan="2">被零除</td><td colspan="2">0</td><td colspan="3" rowspan="6"></td><td rowspan="7"></td></tr>
<tr><td>+0</td><td rowspan="2">0</td><td colspan="2" rowspan="2">无效运算</td><td>+0</td><td>–0</td></tr>
<tr><td>–0</td><td>–0</td><td>+0</td></tr>
<tr><td>+∞</td><td rowspan="2">∞</td><td>+∞</td><td>–∞</td><td colspan="2" rowspan="2">无效运算</td></tr>
<tr><td>–∞</td><td>–∞</td><td>+∞</td></tr>
<tr><td>非规格化数</td><td colspan="7">非安装处理</td><td rowspan="2">QNaN</td></tr>
<tr><td>QNaN</td><td colspan="8">QNaN</td></tr>
<tr><td>SNaN</td><td colspan="9">无效运算</td></tr>
</table>

DN=1

<table>
<tr><td colspan="2" rowspan="2"></td><td colspan="7">src</td></tr>
<tr><td>规格化数</td><td>+0、
+ 非规格化数</td><td>–0、
– 非规格化数</td><td>+∞</td><td>–∞</td><td>QNaN</td><td>SNaN</td></tr>
<tr><td rowspan="7">dest</td><td>规格化数</td><td>除法运算</td><td colspan="2">被零除</td><td colspan="2">0</td><td colspan="2" rowspan="5"></td><td rowspan="6"></td></tr>
<tr><td>+0、 + 非规格化数</td><td rowspan="2">0</td><td colspan="2" rowspan="2">无效运算</td><td>+0</td><td>–0</td></tr>
<tr><td>–0、 – 非规格化数</td><td>–0</td><td>+0</td></tr>
<tr><td>+∞</td><td rowspan="2">∞</td><td>+∞</td><td>–∞</td><td colspan="2" rowspan="2">无效运算</td></tr>
<tr><td>–∞</td><td>–∞</td><td>+∞</td></tr>
<tr><td>QNaN</td><td colspan="6">QNaN</td><td rowspan="2"></td></tr>
<tr><td>SNaN</td><td colspan="7">无效运算</td></tr>
</table>

FMUL

浮点乘法运算
Floating-point MULtiply

FMUL

浮点运算指令

【语法】
FMUL src, dest

【指令码】
记载页：172

【操作】
dest = dest * src;

【功能】
- 将dest保存的单精度浮点数和src保存的单精度浮点数进行乘法运算，其结果保存到dest。根据FPSW的RM[1:0]位，将结果进行舍入。
- 非规格化数的处理因FPSW的DN位而不同。

注 . 使用累加器 （ACC），在执行指令后 ACC 的值为不定值。

【标志的变化】

标志	变化	条件
C	—	
Z	○	当运算结果是 “+0” 或者 “–0” 时为 “1”，否则为 “0”。
S	○	当运算结果的符号位 （bit31）是 “1” 时为 “1”，是 “0” 时为 “0”。
O	—	
CV	○	当发生无效运算时为 “1” 时，否则为 “0”。
CO	○	当发生上溢时为 “1”，否则为 “0”。
CZ	○	总是为 “0”。
CU	○	当发生下溢时为 “1”，否则为 “0”。
CX	○	当发生精度异常时为 “1”，否则为 “0”。
CE	○	当发生非安装处理异常时为 “1”，否则为 “0”。
FV	○	当发生无效运算时为 “1”，否则不变。
FO	○	当发生上溢时为 “1”，否则不变。
FZ	—	
FU	○	当发生下溢时为 “1”，否则不变。
FX	○	当发生精度异常时为 “1”，否则不变。

注 . 在异常处理允许位 EX、EU、EO、EV 为 “1” 时，FX、FU、FO、FV 标志不变。在发生异常处理时，S 标志和 Z 标志不变。

【指令格式】

语法	处理长度	对象		代码长度（字节）
		src	dest	
FMUL src, dest	L	#IMM:32	Rd	7
	L	Rs	Rd	3
	L	[Rs].L	Rd	3
	L	dsp:8[Rs].L （注）	Rd	4
	L	dsp:16[Rs].L （注）	Rd	5

注 . 本公司的 “RX 族的汇编程序” 的位移量的值 （dsp:8、 dsp:16）必须指定为 4 的倍数。能给 dsp:8 指定 0 ～ 1020 （255×4），并且能给 dsp:16 指定 0 ～ 262140 （65535×4）。在指令码里填入位移量除 4 后的值。

【发生异常】

非安装处理
无效运算
上溢
下溢
精度异常

【记述例子】

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

【操作补充说明】

- 当DN=0和DN=1时， src和dest的值与运算结果的对应如下所示：

DN=0

		src							
		规格化数	+0	–0	+∞	–∞	非规格化数	QNaN	SNaN
dest	规格化数	乘法运算			∞				
	+0		+0	–0	无效运算				
	–0		–0	+0					
	+∞	∞	无效运算		+∞	–∞			
	–∞				–∞	+∞			
	非规格化数	非安装处理							
	QNaN	QNaN							
	SNaN	无效运算							

DN=1

		src						
		规格化数	+0、 + 非规格化数	–0、 – 非规格化数	+∞	–∞	QNaN	SNaN
dest	规格化数	乘法运算			∞			
	+0、 + 非规格化数		+0	–0	无效运算			
	–0、 – 非规格化数		–0	+0				
	+∞	∞	无效运算		+∞	–∞		
	–∞				–∞	+∞		
	QNaN	QNaN						
	SNaN	无效运算						

FSUB

浮点减法运算
Floating-point SUBtract

FSUB

浮点运算指令

【语法】
FSUB src, dest

【指令码】
记载页：173

【操作】
dest = dest - src;

【功能】
- 从dest保存的单精度浮点数减去src保存的单精度浮点数，其结果保存到dest。根据FPSW的RM[1:0]位，将结果进行舍入。
- 非规格化数的处理因FPSW的DN位而不同。
- 当持有相同符号的src和dest的差为 “0”时，如果不是向–∞方向舍入的模式，结果就为 “+0”；如果是向–∞方向舍入的模式，结果就为 “–0”。

【标志的变化】

标志	变化	条件
C	—	
Z	○	当运算结果是 “+0”或者 “–0”时为 “1”，否则为 “0”。
S	○	当运算结果的符号位 （bit31）是 “1”时为 “1”，是 “0”时为 “0”。
O	—	
CV	○	当发生无效运算时为 “1”，否则为 “0”。
CO	○	当发生上溢时为 “1”，否则为 “0”。
CZ	○	总是为 “0”。
CU	○	当发生下溢时为 “1”，否则为 “0”。
CX	○	当发生精度异常时为 “1”，否则为 “0”。
CE	○	当发生非安装处理异常时为 “1”，否则为 “0”。
FV	○	当发生无效运算时为 “1”，否则不变。
FO	○	当发生上溢时为 “1”，否则不变。
FZ	—	
FU	○	当发生下溢时为 “1”，否则不变。
FX	○	当发生精度异常时为 “1”，否则不变。

注 . 在异常处理允许位 EX、EU、EO、EV 为 “1”时，FX、FU、FO、FV 标志不变。在发生异常处理时，S 标志和 Z 标志不变。

【指令格式】

语法	处理长度	对象		代码长度（字节）
		src	dest	
FSUB src, dest	L	#IMM:32	Rd	7
	L	Rs	Rd	3
	L	[Rs].L	Rd	3
	L	dsp:8[Rs].L （注）	Rd	4
	L	dsp:16[Rs].L （注）	Rd	5

注 . 本公司的 “RX 族的汇编程序”的位移量的值 （dsp:8、 dsp:16）必须指定为 4 的倍数。能给 dsp:8 指定 0 ～ 1020 （255×4），并且能给 dsp:16 指定 0 ～ 262140 （65535×4）。在指令码里填入位移量除 4 后的值。

【发生异常】

非安装处理
无效运算
上溢
下溢
精度异常

【记述例子】

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

【操作补充说明】

- 当DN=0和DN=1时， src和dest的值与运算结果的对应如下所示：

DN=0

		src							
		规格化数	+0	–0	+∞	–∞	非规格化数	QNaN	SNaN
dest	规格化数	减法运算			–∞	+∞			
	+0		（注）	+0					
	–0		–0	（注）					
	+∞	+∞			无效运算				
	–∞	–∞				无效运算			
	非规格化数	非安装处理							
	QNaN	QNaN							
	SNaN	无效运算							

DN=1

		src						
		规格化数	+0、+ 非规格化数	–0、– 非规格化数	+∞	–∞	QNaN	SNaN
dest	规格化数	减法运算			–∞	+∞		
	+0、 + 非规格化数		（注）	+0				
	–0、 – 非规格化数		–0	（注）				
	+∞	+∞			无效运算			
	–∞	–∞				无效运算		
	QNaN	QNaN						
	SNaN	无效运算						

注 . 在舍入模式为向 –∞ 方向舍入时为 “–0”，否则为 “+0”。

FTOI

浮点数 → 整数的转换
Float TO Integer

FTOI

浮点运算指令
【指令码】
记载页：174

【语法】
FTOI src, dest

【操作】
dest = (signed long) src;

【功能】
- 将src保存的单精度浮点数转换为带符号的长字 （32位）整数，其结果保存到dest。
- 结果与FPSW的RM[1:0]位无关，总是向0方向舍入。

【标志的变化】

标志	变化	条件
C	—	
Z	○	当运算结果是 “0” 时为 “1”，否则为 “0”。
S	○	当运算结果的符号位 （bit31）是 “1” 时为 “1”，是 “0” 时为 “0”。
O	—	
CV	○	当发生无效运算时为 “1”，否则为 “0”。
CO	○	总是为 “0”。
CZ	○	总是为 “0”。
CU	○	总是为 “0”。
CX	○	当发生精度异常时为 “1”，否则为 “0”。
CE	○	当发生非安装处理异常时为 “1”，否则为 “0”。
FV	○	当发生无效运算时为 “1”，否则不变。
FO	—	
FZ	—	
FU	—	
FX	○	当发生精度异常时为 “1”，否则不变。

注 . 在异常处理允许位 EX、 EV 为 “1” 时， FX 标志和 FV 标志不变。在发生异常处理时， S 标志和 Z 标志不变。

【指令格式】

语法	处理长度	对象		代码长度（字节）
		src	dest	
FTOI src, dest	L	Rs	Rd	3
	L	[Rs].L	Rd	3
	L	dsp:8[Rs].L （注）	Rd	4
	L	dsp:16[Rs].L （注）	Rd	5

注 . 本公司的 “RX 族的汇编程序” 的位移量的值 （dsp:8、 dsp:16）必须指定为 4 的倍数。能给 dsp:8 指定 0 ～ 1020 （255×4），并且能给 dsp:16 指定 0 ～ 262140 （65535×4）。在指令码里填入位移量除 4 后的值。

【发生异常】

非安装处理
无效运算
精度异常

【记述例子】

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

【操作补充说明】

- 当DN=0和DN=1时， src和dest的值与运算结果的对应如下所示：

DN=0

src 的值 （指数部为无偏差的值）		dest	异常
src ≥ 0	+∞	当 EV 位为 “1” 并且发生无效运算时：不变 上述以外：7FFFFFFFh	无效运算
	127 ≥指数部≥ 31		
	30 ≥指数部≥ –126	00000000h ～ 7FFFFF80h	无 （注 1）
	+ 非规格化数	不变	非安装处理
	+0	00000000h	无
src < 0	–0		
	– 非规格化数	不变	非安装处理
	30 ≥指数部≥ –126	00000000h ～ 80000080h	无 （注 1）
	127 ≥指数部≥ 31	当 EV 位为 “1” 并且发生无效运算时：不变 上述以外：80000000h	无效运算 （注 2）
	–∞		
NaN	QNaN	当 EV 位为 “1” 并且发生无效运算时：不变 上述以外： 符号位 =0：7FFFFFFFh 符号位 =1：80000000h	无效运算
	SNaN		

注 1. 在进行舍入时，发生精度异常。

注 2. 当 src 为 “CF000000h” 时，不发生无效运算。

DN=1

src 的值 （指数部为无偏差的值）		dest	异常
src ≥ 0	+∞	当 EV 位为 “1” 并且发生无效运算时：不变 上述以外：7FFFFFFFh	无效运算
	127 ≥指数部≥ 31		
	30 ≥指数部≥ –126	00000000h ～ 7FFFFF80h	无 （注 1）
	+0、 + 非规格化数	00000000h	无
src < 0	–0、 – 非规格化数		
	30 ≥指数部≥ –126	00000000h ～ 80000080h	无 （注 1）
	127 ≥指数部≥ 31	当 EV 位为 “1” 并且发生无效运算时：不变 上述以外：80000000h	无效运算 （注 2）
	–∞		
NaN	QNaN	当 EV 位为 “1” 并且发生无效运算时：不变 上述以外： 符号位 =0：7FFFFFFFh 符号位 =1：80000000h	无效运算
	SNaN		

注 1. 在进行舍入时，发生精度异常。

注 2. 当 src 为 “CF000000h” 时，不发生无效运算。

INT

软件中断
INTerrupt

INT

系统操作指令

【语法】
INT #src

【指令码】
记载页：174

【操作】

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

【功能】

- 产生由src指定序号的无条件陷阱。
- src的范围为0≤src≤255。
- 转移到管理模式， PSW的PM位为 “0”。
- PSW的U位和I位为 “0”。

【标志的变化】

- 标志不变。
- 将执行指令前的PSW压栈。

【指令格式】

语法	对象	代码长度
	src	（字节）
INT #src	#IMM:8	3

【记述例子】

```
INT #0
```

ITOF

整数 → 浮点数的转换
Integer TO Floating-point

ITOF

浮点运算指令

【语法】
ITOF src, dest

【指令码】
记载页：175

【操作】
dest = (float)src;

【功能】
- 将src保存的带符号的长字 （32位）整数转换为单精度浮点数，其结果保存到dest。根据FPSW的RM[1:0]位，将结果进行舍入。将00000000h作为 “+0”进行处理，与舍入模式无关。

【标志的变化】

标志	变化	条件
C	—	
Z	○	当运算结果是 “+0”时为 “1”，否则为 “0”。
S	○	当运算结果的符号位 （bit31）是 “1”时为 “1”，是 “0”时为 “0”。
O	—	
CV	○	总是为 “0”。
CO	○	总是为 “0”。
CZ	○	总是为 “0”。
CU	○	总是为 “0”。
CX	○	当发生精度异常时为 “1”，否则为 “0”。
CE	○	总是为 “0”。
FV	—	
FO	—	
FZ	—	
FU	—	
FX	○	当发生精度异常时为 “1”，否则不变。

注 . 在异常处理允许位 EX 为 “1”时， FX 标志不变。在发生异常处理时， S 标志和 Z 标志不变。

【指令格式】

语法	处理长度	对象		代码长度（字节）
		src	dest	
ITOF src, dest	L	Rs	Rd	3
	L	[Rs].memex	Rd	3 (memex == UB) 4 (memex != UB)
	L	dsp:8[Rs].memex （注）	Rd	4 (memex == UB) 5 (memex != UB)
	L	dsp:16[Rs].memex （注）	Rd	5 (memex == UB) 6 (memex != UB)

注 . 本公司的 “RX 族的汇编程序”的位移量的值 （dsp:8、 dsp:16）必须在长度扩展说明符为 “.W”或者 “.UW”时指定为 2 的倍数，在长度扩展说明符为 “.L”时指定为 4 的倍数。当长度扩展说明符为 “.W”或者 “.UW”时，能给 dsp:8 指定 0 ～ 510 （255×2）； 当长度扩展说明符为 “.L”时，能给 dsp:8 指定 0 ～ 1020 （255×4）。当长度扩展说明符为 “.W”或者 “.UW”时，能给 dsp:16 指定 0 ～ 131070 （65535×2）； 当长度扩展说明符为 “.L”时，能给 dsp:16 指定 0 ～ 262140 （65535×4）。在指令码里填入位移量除 2 或者除 4 后的值。

【发生异常】

精度异常

【记述例子】

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

JMP

无条件转移
JuMP

JMP

转移指令

【语法】
JMP src

【指令码】
记载页：175

【操作】
PC = src;

【功能】
- 转移到src。

【标志的变化】
- 标志不变。

【指令格式】

语法	对象	代码长度
	src	（字节）
JMP src	Rs	2

【记述例子】
JMP R1

JSR

子程序转移
Jump SubRoutine

JSR

转移指令

【语法】
JSR src

【指令码】
记载页：176

【操作】
SP = SP - 4;
*SP = (PC + 2);（注）
PC = src;

注 . (PC+2) 为 JSR 指令的下一个指令地址。

【功能】
- 转移到src所示的子程序。

【标志的变化】
- 标志不变。

【指令格式】

语法	对象	代码长度
	src	（字节）
JSR src	Rs	2

【记述例子】
JSR R1

MACHI

高 16 位的乘加运算

Multiply-ACcumulate HIgh-order word

MACHI

DSP 功能指令

【指令码】

记载页：176

【语法】

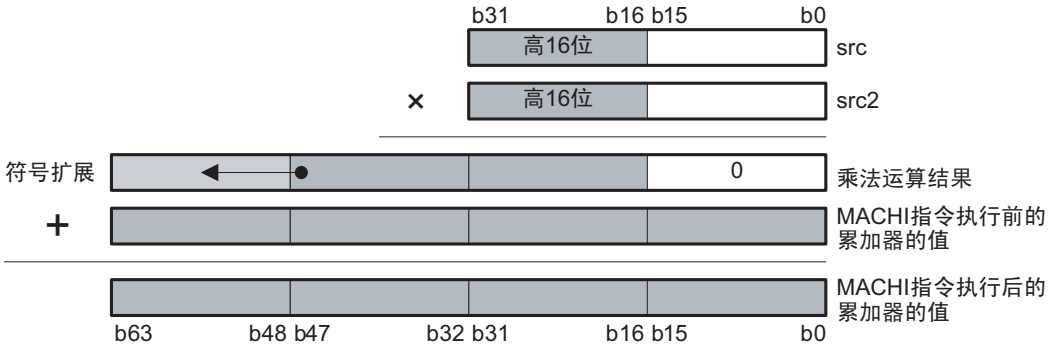
MACHI　src, src2

【操作】

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

【功能】

- 将src的高16位和src2的高16位进行乘法运算，并且将乘法运算结果和累加器相加。但是，在相加时乘法结果的最低位与累加器的b16对齐，加法运算结果保存到累加器。将src的高16位和src2的高16位作为带符号的整数进行处理。



【标志的变化】

- 标志不变。

【指令格式】

语法	对象		代码长度（字节）
	src	src2	
MACHI　src, src2	Rs	Rs2	3

【记述例子】

MACHI　R1, R2

MACLO

低 16 位的乘加运算

Multiply-ACcumulate LOw-order word

MACLO

DSP 功能指令

【语法】

MACLO　src, src2

【指令码】

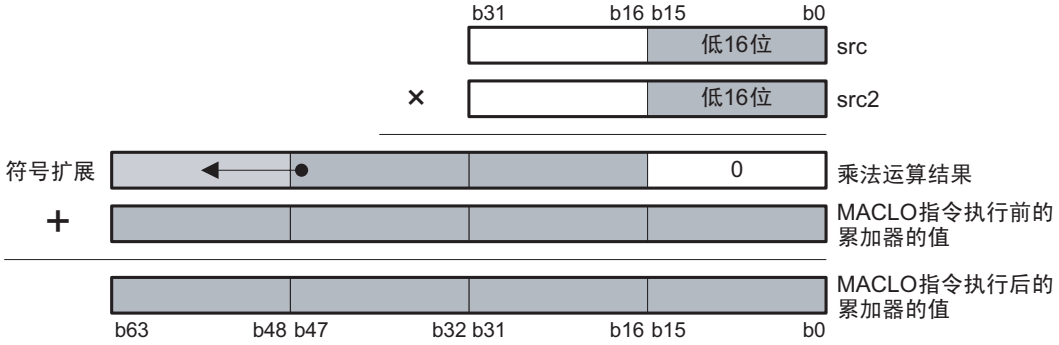
记载页：176

【操作】

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

【功能】

- 将src的低16位和src2的低16位进行乘法运算，并且将乘法运算结果和累加器相加。但是，在相加时乘法结果的最低位与累加器的b16对齐，加法运算结果保存到累加器。将src的低16位和src2的低16位作为带符号的整数进行处理。



【标志的变化】

- 标志不变。

【指令格式】

语法	对象		代码长度（字节）
	src	src2	
MACLO　src, src2	Rs	Rs2	3

【记述例子】

MACLO　R1, R2

MAX

最大值选择
MAXimum value select

MAX

算术 / 逻辑运算指令

【语法】
MAX src, dest

【指令码】
记载页：177

【操作】
```
if ( src > dest )
    dest = src;
```

【功能】
- 将src和dest进行带符号的比较，大的值保存到dest。

【标志的变化】
- 标志不变。

【指令格式】

语法	处理长度	对象		代码长度（字节）
		src	dest	
MAX src, dest	L	#SIMM:8	Rd	4
	L	#SIMM:16	Rd	5
	L	#SIMM:24	Rd	6
	L	#IMM:32	Rd	7
	L	Rs	Rd	3
	L	[Rs].memex	Rd	3 (memex == UB) 4 (memex != UB)
	L	dsp:8[Rs].memex（注）	Rd	4 (memex == UB) 5 (memex != UB)
	L	dsp:16[Rs].memex（注）	Rd	5 (memex == UB) 6 (memex != UB)

注 . 本公司的 “RX 族的汇编程序” 的位移量的值 （dsp:8、 dsp:16）必须在长度扩展说明符为 “.W” 或者 “.UW” 时指定为 2 的倍数，在长度扩展说明符为 “.L” 时指定为 4 的倍数。当长度扩展说明符为 “.W” 或者 “.UW” 时，能给 dsp:8 指定 0 ～ 510 （255×2）；当长度扩展说明符为 “.L” 时，能给 dsp:8 指定 0 ～ 1020 （255×4）。当长度扩展说明符为 “.W” 或者 “.UW” 时，能给 dsp:16 指定 0 ～ 131070 （65535×2）；当长度扩展说明符为 “.L” 时，能给 dsp:16 指定 0 ～ 262140 （65535×4）。在指令码里填入位移量除 2 或者除 4 后的值。

【记述例子】
```
MAX  #10, R2
MAX  R1, R2
MAX  [R1], R2
MAX  3[R1].B, R2
```

MIN

最小值选择
MINimum value select

MIN

算术 / 逻辑运算指令

【语法】
MIN src, dest

【指令码】
记载页：178

【操作】
```
if ( src < dest )
  dest = src;
```

【功能】
- 将src和dest进行带符号的比较，小的值保存到dest。

【标志的变化】
- 标志不变。

【指令格式】

语法	处理长度	对象		代码长度（字节）
		src	dest	
MIN src, dest	L	#SIMM:8	Rd	4
	L	#SIMM:16	Rd	5
	L	#SIMM:24	Rd	6
	L	#IMM:32	Rd	7
	L	Rs	Rd	3
	L	[Rs].memex	Rd	3 (memex == UB) 4 (memex != UB)
	L	dsp:8[Rs].memex（注）	Rd	4 (memex == UB) 5 (memex != UB)
	L	dsp:16[Rs].memex（注）	Rd	5 (memex == UB) 6 (memex != UB)

注 . 本公司的 “RX 族的汇编程序” 的位移量的值 （dsp:8、 dsp:16）必须在长度扩展说明符为 “.W” 或者 “.UW” 时指定为 2 的倍数，在长度扩展说明符为 “.L” 时指定为 4 的倍数。当长度扩展说明符为 “.W” 或者 “.UW” 时，能给 dsp:8 指定 0 ～ 510 （255×2）；当长度扩展说明符为 “.L” 时，能给 dsp:8 指定 0 ～ 1020 （255×4）。当长度扩展说明符为 “.W” 或者 “.UW” 时，能给 dsp:16 指定 0 ～ 131070 （65535×2）；当长度扩展说明符为 “.L” 时，能给 dsp:16 指定 0 ～ 262140 （65535×4）。在指令码里填入位移量除 2 或者除 4 后的值。

【记述例子】
```
MIN  #10, R2
MIN  R1, R2
MIN  [R1], R2
MIN  3[R1].B, R2
```

MOV

传送
MOVe

MOV

传送指令

【语法】
MOV.size src, dest

【指令码】
记载页：179

【操作】
dest = src;

【功能】
- 将src传送到dest，详细内容如下所示。

src	dest	功能
立即数	寄存器	将立即数传送到寄存器。如果指定少于 32 位的立即数，就将 #UIMM 进行零扩展，将 #SIMM 进行符号扩展，然后传送到寄存器。
立即数	存储器	根据指定的长度，将立即数传送到存储器。如果指定位长小于指定长度的立即数，就将 #UIMM 进行零扩展，将 #SIMM 进行符号扩展，然后传送到存储器。
寄存器	寄存器	将寄存器 （src）的数据传送到寄存器 （dest）。当长度说明符为 “.B” 时，将寄存器 （src）的 LSB 侧的字节数据进行符号扩展，在扩展为长字数据后传送到寄存器 （dest）；当长度说明符为 “.W” 时，将寄存器 （src）的 LSB 侧的字数据进行符号扩展，在扩展为长字数据后传送到寄存器 （dest）。
寄存器	存储器	将寄存器的数据传送到存储器。当长度说明符为 “.B” 时，传送寄存器的 LSB 侧的字节数据；当长度说明符为 “.W” 时，传送寄存器的 LSB 侧的字数据。
存储器	寄存器	将存储器的数据传送到寄存器。当长度说明符为 “.B” 或者 “.W” 时，将存储器的数据进行符号扩展，在扩展为长字数据后传送到寄存器。
存储器	存储器	根据指定的长度，将存储器 （src）的数据传送到存储器 （dest）。

【标志的变化】
- 标志不变。

【指令格式】

语法	size	处理长度	对象		代码长度（字节）
			src	dest	
MOV.size src, dest	存储 （短指令）				
	B/W/L	size	Rs (Rs=R0 ～ R7)	dsp:5[Rd]（注 1） (Rd=R0 ～ R7)	2
	加载 （短指令）				
	B/W/L	L	dsp:5[Rs]（注 1） (Rs=R0 ～ R7)	Rd (Rd=R0 ～ R7)	2
	寄存器的立即数设定 （短指令）				
	L	L	#UIMM:4	Rd	2
	存储器的立即数设定 （短指令）				
	B	B	#IMM:8	dsp:5[Rd]（注 1） (Rd=R0 ～ R7)	3
	W/L	size	#UIMM:8	dsp:5[Rd]（注 1） (Rd=R0 ～ R7)	3
	寄存器的立即数设定				
	L	L	#UIMM:8（注 2）	Rd	3
	L	L	#SIMM:8（注 2）	Rd	3
	L	L	#SIMM:16	Rd	4
	L	L	#SIMM:24	Rd	5
	L	L	#IMM:32	Rd	6
	寄存器间的传送 （有符号扩展）				
	B/W	L	Rs	Rd	2
	寄存器间的传送 （无符号扩展）				
	L	L	Rs	Rd	2
	存储器的立即数设定				
	B	B	#IMM:8	[Rd]	3
	B	B	#IMM:8	dsp:8[Rd]（注 1）	4
	B	B	#IMM:8	dsp:16[Rd]（注 1）	5
	W	W	#SIMM:8	[Rd]	3
	W	W	#SIMM:8	dsp:8[Rd]（注 1）	4
	W	W	#SIMM:8	dsp:16[Rd]（注 1）	5
	W	W	#IMM:16	[Rd]	4
	W	W	#IMM:16	dsp:8[Rd]（注 1）	5
	W	W	#IMM:16	dsp:16[Rd]（注 1）	6
	L	L	#SIMM:8	[Rd]	3
	L	L	#SIMM:8	dsp:8[Rd]（注 1）	4
	L	L	#SIMM:8	dsp:16[Rd]（注 1）	5
	L	L	#SIMM:16	[Rd]	4
	L	L	#SIMM:16	dsp:8[Rd]（注 1）	5
	L	L	#SIMM:16	dsp:16[Rd]（注 1）	6
	L	L	#SIMM:24	[Rd]	5
	L	L	#SIMM:24	dsp:8[Rd]（注 1）	6
	L	L	#SIMM:24	dsp:16[Rd]（注 1）	7
	L	L	#IMM:32	[Rd]	6
	L	L	#IMM:32	dsp:8[Rd]（注 1）	7
	L	L	#IMM:32	dsp:16[Rd]（注 1）	8

语法	size	处理长度	对象		代码长度（字节）
			src	dest	
MOV.size src, dest	加载				
	B/W/L	L	[Rs]	Rd	2
	B/W/L	L	dsp:8[Rs]（注 1）	Rd	3
	B/W/L	L	dsp:16[Rs]（注 1）	Rd	4
	B/W/L	L	[Ri, Rb]	Rd	3
	存储				
	B/W/L	size	Rs	[Rd]	2
	B/W/L	size	Rs	dsp:8[Rd]（注 1）	3
	B/W/L	size	Rs	dsp:16[Rd]（注 1）	4
	B/W/L	size	Rs	[Ri, Rb]	3
	存储器间的传送				
	B/W/L	size	[Rs]	[Rd]	2
	B/W/L	size	[Rs]	dsp:8[Rd]（注 1）	3
	B/W/L	size	[Rs]	dsp:16[Rd]（注 1）	4
	B/W/L	size	dsp:8[Rs]（注 1）	[Rd]	3
	B/W/L	size	dsp:8[Rs]（注 1）	dsp:8[Rd]（注 1）	4
	B/W/L	size	dsp:8[Rs]（注 1）	dsp:16[Rd]（注 1）	5
	B/W/L	size	dsp:16[Rs]（注 1）	[Rd]	4
	B/W/L	size	dsp:16[Rs]（注 1）	dsp:8[Rd]（注 1）	5
	B/W/L	size	dsp:16[Rs]（注 1）	dsp:16[Rd]（注 1）	6
	带后增的存储（注 3）				
	B/W/L	size	Rs	[Rd+]	3
	带先减的存储（注 3）				
	B/W/L	size	Rs	[–Rd]	3
	带后增的加载（注 4）				
	B/W/L	L	[Rs+]	Rd	3
	带先减的加载（注 4）				
	B/W/L	L	[–Rs]	Rd	3

注 1. 本公司的 “RX 族的汇编程序” 的位移量的值 （dsp:5、 dsp:8、 dsp:16）必须在长度说明符为 “.W” 时指定为 2 的倍数，在长度说明符为 “.L” 时指定为 4 的倍数。当长度说明符为 “.W” 时，能给 dsp:5 指定 0 ～ 62 （31×2）；当长度说明符为 “.L” 时，能给 dsp:5 指定 0 ～ 124 （31×4）。当长度说明符为 “.W” 时，能给 dsp:8 指定 0 ～ 510 （255×2）；当长度说明符为 “.L” 时，能给 dsp:8 指定 0 ～ 1020 （255×4）。当长度说明符为 “.W” 时，能给 dsp:16 指定 0 ～ 131070 （65535×2）；当长度说明符为 “.L” 时，能给 dsp:16 指定 0 ～ 262140 （65535×4）。在指令码里填入位移量除 2 或者除 4 后的值。

注 2. 0 ～ 127 的范围总是为零扩展指令码。

注 3. 在带后增的存储和带先减的存储的情况下，如果给 Rs 和 Rd 指定相同的寄存器，就将更新地址前的值作为源操作数进行传送。

注 4. 在带后增的加载和带先减的加载的情况下，如果给 Rs 和 Rd 指定相同的寄存器，就将从存储器传送来的数据保存到 Rd。

【记述例子】

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

MOVU

不带符号的数据传送
MOVe Unsigned data

MOVU

传送指令

【语法】
MOVU.size src, dest

【指令码】
记载页：184

【操作】
dest = src;

【功能】
- 将src传送到dest，详细内容如下所示。

src	dest	功能
寄存器	寄存器	将寄存器 （src）的 LSB 侧的字节数据或者字数据进行零扩展，在扩展为长字数据后传送到寄存器 （dest）。
存储器	寄存器	将存储器的字节数据或者字数据进行零扩展，在扩展为长字数据后传送到寄存器。

【标志的变化】
- 标志不变。

【指令格式】

语法	size	处理长度	对象		代码长度
			src	dest	（字节）
MOVU.size src, dest	加载 （短指令）				
	B/W	L	dsp:5[Rs] （注 1） (Rs=R0 ～ R7)	Rd (Rd=R0 ～ R7)	2
	寄存器间的传送 （有零扩展）				
	B/W	L	Rs	Rd	2
	加载				
	B/W	L	[Rs]	Rd	2
	B/W	L	dsp:8[Rs] （注 1）	Rd	3
	B/W	L	dsp:16[Rs] （注 1）	Rd	4
	B/W	L	[Ri, Rb]	Rd	3
	带后增的加载 （注 2）				
	B/W	L	[Rs+]	Rd	3
	带先减的加载 （注 2）				
	B/W	L	[–Rs]	Rd	3

注 1. 本公司的 “RX 族的汇编程序” 的位移量的值 （dsp:5、 dsp:8、 dsp:16）必须在长度说明符为 “.W” 时指定为 2 的倍数。当长度说明符为 “.W” 时，能给 dsp:5 指定 0 ～ 62 （31×2）；当长度说明符为 “.W” 时，能给 dsp:8 指定 0 ～ 510 （255×2）；当长度说明符为 “.W” 时，能给 dsp:16 指定 0 ～ 131070 （65535×2）。在指令码里填入位移量除 2 后的值。

注 2. 在带后增的加载和带先减的加载的情况下，如果给 Rs 和 Rd 指定相同的寄存器，就将从存储器传送来的数据保存到 Rd。

【记述例子】
```
MOVU.W  2[R1], R2
MOVU.W  R1, R2
MOVU.B  [R1+], R2
MOVU.B  [-R1], R2
```

MUL

乘法运算
MULtiply

MUL

算术 / 逻辑运算指令

【语法】
(1) MUL src, dest
(2) MUL src, src2, dest

【指令码】
记载页：185

【操作】
(1) dest = src * dest;
(2) dest = src * src2;

【功能】
1. 将src和dest进行乘法运算，其结果保存到dest。
   - 以32位进行运算，保存结果的低32位。
   - 与是否带符号的乘法运算无关，运算结果相同。
2. 将src和src2进行乘法运算，其结果保存到dest。
   - 以32位进行运算，保存结果的低32位。
   - 与是否带符号的乘法运算无关，运算结果相同。

注 . 使用累加器 （ACC），在执行指令后 ACC 的值为不定值。

【标志的变化】
- 标志不变。

【指令格式】

语法	处理长度	对象			代码长度（字节）
		src	src2	dest	
(1) MUL src, dest	L	#UIMM:4	—	Rd	2
	L	#SIMM:8	—	Rd	3
	L	#SIMM:16	—	Rd	4
	L	#SIMM:24	—	Rd	5
	L	#IMM:32	—	Rd	6
	L	Rs	—	Rd	2
	L	[Rs].memex	—	Rd	2 (memex == UB) 3 (memex != UB)
	L	dsp:8[Rs].memex（注）	—	Rd	3 (memex == UB) 4 (memex != UB)
	L	dsp:16[Rs].memex（注）	—	Rd	4 (memex == UB) 5 (memex != UB)
(2) MUL src, src2, dest	L	Rs	Rs2	Rd	3

注 . 本公司的 “RX 族的汇编程序” 的位移量的值 （dsp:8、 dsp:16）必须在长度扩展说明符为 “.W” 或者 “.UW” 时指定为 2 的倍数，在长度扩展说明符为 “.L” 时指定为 4 的倍数。当长度扩展说明符为 “.W” 或者 “.UW” 时，能给 dsp:8 指定 0 ～ 510 （255×2）；当长度扩展说明符为 “.L” 时，能给 dsp:8 指定 0 ～ 1020 （255×4）。当长度扩展说明符为 “.W” 或者 “.UW” 时，能给 dsp:16 指定 0 ～ 131070 （65535×2）；当长度扩展说明符为 “.L” 时，能给 dsp:16 指定 0 ～ 262140 （65535×4）。在指令码里填入位移量除 2 或者除 4 后的值。

【记述例子】

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

MULHI

高 16 位的乘法运算
MULtiply HIgh-order word

MULHI

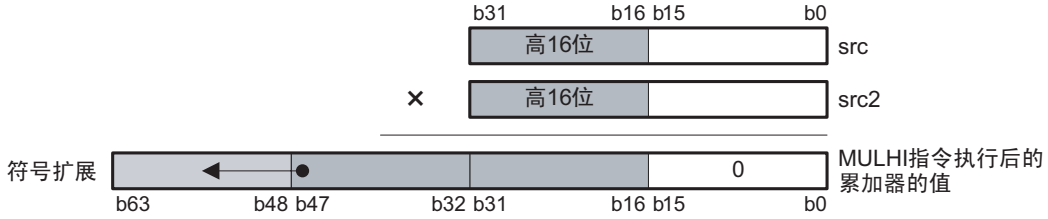
DSP 功能指令

【指令码】
记载页：186

【语法】
MULHI src, src2

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

【功能】
- 将src的高16位和src2的高16位进行乘法运算，其结果保存到累加器。但是，乘法结果的最低位与累加器的b16对齐，与累加器的b63～b48对应的部分被符号扩展。另外，累加器的b15～b0为“0”，并且将src的高16位和src2的高16位作为带符号的整数进行处理。



【标志的变化】
- 标志不变。

【指令格式】

语法	对象		代码长度（字节）
	src	src2	
MULHI src, src2	Rs	Rs2	3

【记述例子】
MULHI R1, R2

MULLO

低 16 位的乘法运算
MULtiply LOw-order word

MULLO

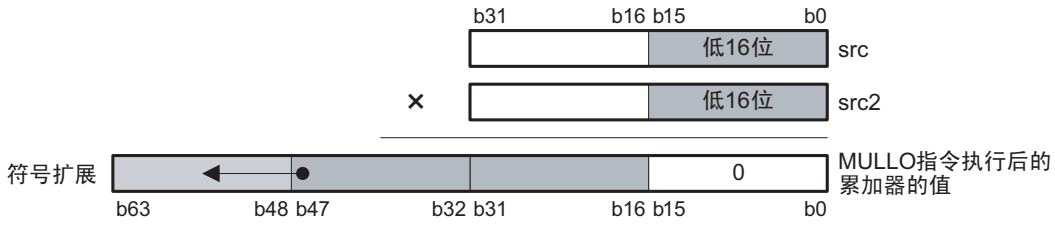
DSP 功能指令

【指令码】
记载页：187

【语法】
MULLO　src, src2

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

【功能】
- 将src的低16位和src2的低16位进行乘法运算，其结果保存到累加器。但是，乘法结果的最低位与累加器的b16对齐，与累加器的b63～b48对应的部分被符号扩展。另外，累加器的b15～b0为“0”，并且将src的低16位和src2的低16位作为带符号的整数进行处理。



【标志的变化】
- 标志不变。

【指令格式】

语法	对象		代码长度（字节）
	src	src2	
MULLO　src, src2	Rs	Rs2	3

【记述例子】
```
MULLO　R1, R2
```

MVFACHI

从累加器高 32 位的传送

MoVe From ACcumulator HIgh-order longword

MVFACHI

DSP 功能指令

【指令码】

记载页：187

【语法】

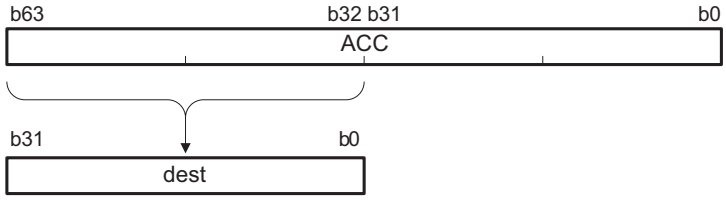
MVFACHI dest

【操作】

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

【功能】

- 将累加器的高32位的内容传送到dest。



【标志的变化】

- 标志不变。

【指令格式】

语法	对象	代码长度
	dest	（字节）
MVFACHI dest	Rd	3

【记述例子】

MVFACHI R1

MVFACMI

从累加器中间 32 位的传送

MoVe From ACcumulator MIddle-order longword

MVFACMI

DSP 功能指令

【指令码】

记载页：187

【语法】

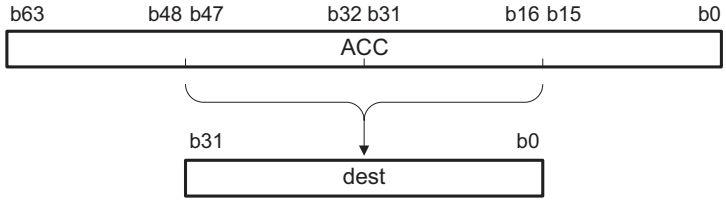
MVFACMI dest

【操作】

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

【功能】

- 将累加器的b47～b16的内容传送到dest。



【标志的变化】

- 标志不变。

【指令格式】

语法	对象	代码长度
	dest	（字节）
MVFACMI dest	Rd	3

【记述例子】

MVFACMI R1

MVFC

从控制寄存器的传送
MoVe From Control register

MVFC

系统操作指令
【指令码】
记载页：188

【语法】
MVFC src, dest

【操作】
dest = src;

【功能】
- 将src传送到dest。
- 如果给src指定PC，就将此指令的地址传送到dest。

【标志的变化】
- 标志不变。

【指令格式】

语法	处理长度	对象		代码长度（字节）
		src（注）	dest	
MVFC src, dest	L	Rx	Rd	3

注 . 可选择的 src：PC、 ISP、 USP、 INTB、 PSW、 BPC、 BPSW、 FINTV、 FPSW

【记述例子】
MVFC USP, R1

MVTACHI

向累加器高 32 位的传送

MoVe To ACcumulator High-order longword

MVTACHI

DSP 功能指令

【语法】

MVTACHI　src

【指令码】

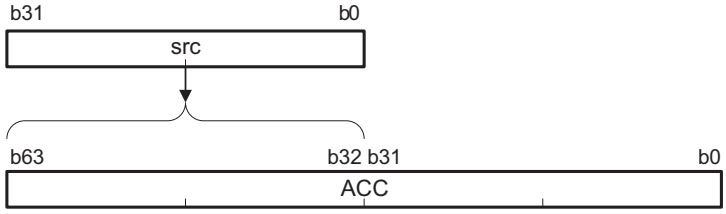
记载页：188

【操作】

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

【功能】

- 将src的内容传送到累加器的高32位 （b63～b32）。



【标志的变化】

- 标志不变。

【指令格式】

语法	对象	代码长度
	src	（字节）
MVTACHI　src	Rs	3

【记述例子】

MVTACHI　R1

MVTACLO

向累加器低 32 位的传送

MoVe To ACcumulator LOw-order longword

MVTACLO

DSP 功能指令

【语法】

MVTACLO　src

【指令码】

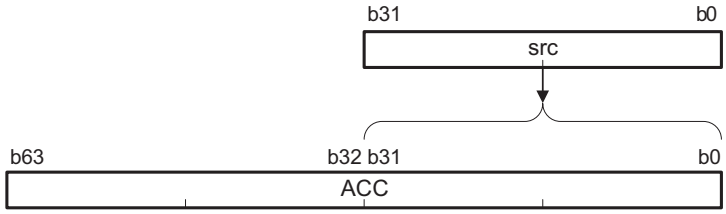
记载页：189

【操作】

ACC = (ACC & FFFFFFFF00000000h) | src;

【功能】

- 将src的内容传送到累加器的低32位 （b31～b0）。



【标志的变化】

- 标志不变。

【指令格式】

语法	对象	代码长度
	src	（字节）
MVTACLO　src	Rs	3

【记述例子】

MVTACLO　R1

MVTC

向控制寄存器的传送
MoVe To Control register

MVTC

系统操作指令

【指令码】
记载页：189

【语法】
MVTC src, dest

【操作】
dest = src;

【功能】
- 将src传送到dest。
- 在用户模式中，忽视写ISP、INTB、BPC、BPSW、FINTV以及PSW的IPL[3:0]位、PM位、U位、I位；在管理模式中，忽视写PSW的PM位。

【标志的变化】

标志	变化	条件
C	（注）	
Z	（注）	
S	（注）	
O	（注）	

注 . 只在 dest 为 PSW 时发生变化。

【指令格式】

语法	处理长度	对象		代码长度（字节）
		src	dest（注）	
MVTC src, dest	L	#SIMM:8	Rx	4
	L	#SIMM:16	Rx	5
	L	#SIMM:24	Rx	6
	L	#IMM:32	Rx	7
	L	Rs	Rx	3

注 . 可选择的 dest：ISP、USP、INTB、PSW、BPC、BPSW、FINTV、FPSW
不能给 dest 指定 PC。

【记述例子】
```
MVTC #0FFFFF000h, INTB
MVTC R1, USP
```

MVTIPL

中断优先级的设定
MoVe To Interrupt Priority Level

MVTIPL

系统操作指令

【语法】
MVTIPL src

【指令码】
记载页：190

【操作】
IPL = src;

【功能】
- 将src传送到PSW的IPL[3:0]位。
- 此指令是特权指令。如果在用户模式中执行，就发生特权指令异常。
- src的值为不带符号的整数， src的范围为0≤src≤15。

【标志的变化】
- 标志不变。

【指令格式】

语法	对象	代码长度
	src	（字节）
MVTIPL src	#IMM:4	3

【记述例子】
MVTIPL #2

注 . 在 RX610 群中，不能使用 MVTIPL 指令。要写处理器状态字 （PSW）的处理器中断优先级 （IPL[2:0]）时，必须使用 MVTC 指令。

NEG

符号取反
NEGate

NEG

算术 / 逻辑运算指令

【指令码】
记载页：191

【语法】
(1) NEG dest
(2) NEG src, dest

【操作】
(1) dest = -dest;
(2) dest = -src;

【功能】
1. 将dest的符号取反（2的补码），其结果保存到dest。
2. 将src的符号取反（2的补码），其结果保存到dest。

【标志的变化】

标志	变化	条件
C	○	当运算后的 dest 是 “0” 时为 “1”，否则为 “0”。
Z	○	当运算后的 dest 是 “0” 时为 “1”，否则为 “0”。
S	○	当运算后的 dest 的 MSB 是 “1” 时为 “1”，否则为 “0”。
O	○	1. 当运算前的 dest 是 “80000000h” 时为 “1”，否则为 “0”。 2. 当运算前的 src 是 “80000000h” 时为 “1”，否则为 “0”。

【指令格式】

语法	处理长度	对象		代码长度（字节）
		src	dest	
(1) NEG dest	L	—	Rd	2
(2) NEG src, dest	L	Rs	Rd	3

【记述例子】
NEG R1
NEG R1, R2

NOP

空操作
No OPeration

NOP

算术 / 逻辑运算指令

【语法】
NOP

【指令码】
记载页：191

【操作】
/* 空操作 */

【功能】
- 不进行任何处理，从下一条指令开始继续执行。

【标志的变化】
- 标志不变。

【指令格式】

语法	代码长度 （字节）
NOP	1

【记述例子】
NOP

NOT

逻辑取反
NOT

NOT

算术 / 逻辑运算指令

【指令码】
记载页：192

【语法】
(1) NOT dest
(2) NOT src, dest

【操作】
(1) dest = ~dest;
(2) dest = ~src;

【功能】
1. 将dest逻辑取反，其结果保存到dest。
2. 将src逻辑取反，其结果保存到dest。

【标志的变化】

标志	变化	条件
C	—	
Z	○	当运算后的 dest 是 “0” 时为 “1”，否则为 “0”。
S	○	当运算后的 dest 的 MSB 是 “1” 时为 “1”，否则为 “0”。
O	—	

【指令格式】

语法	处理长度	对象		代码长度（字节）
		src	dest	
(1) NOT dest	L	—	Rd	2
(2) NOT src, dest	L	Rs	Rd	3

【记述例子】
NOT R1
NOT R1, R2

OR

逻辑或
OR

OR

算术 / 逻辑运算指令

【语法】
(1) OR src, dest
(2) OR src, src2, dest

【指令码】
记载页：193

【操作】
(1) dest = dest | src;
(2) dest = src | src2;

【功能】
1. 取dest和src的逻辑或，其结果保存到dest。
2. 取src和src2的逻辑或，其结果保存到dest。

【标志的变化】

标志	变化	条件
C	—	
Z	○	当运算后的 dest 是 “0” 时为 “1”，否则为 “0”。
S	○	当运算后的 dest 的 MSB 是 “1” 时为 “1”，否则为 “0”。
O	—	

【指令格式】

语法	处理长度	对象			代码长度（字节）
		src	src2	dest	
(1) OR src, dest	L	#UIMM:4	—	Rd	2
	L	#SIMM:8	—	Rd	3
	L	#SIMM:16	—	Rd	4
	L	#SIMM:24	—	Rd	5
	L	#IMM:32	—	Rd	6
	L	Rs	—	Rd	2
	L	[Rs].memex	—	Rd	2 (memex == UB) 3 (memex != UB)
	L	dsp:8[Rs].memex（注）	—	Rd	3 (memex == UB) 4 (memex != UB)
	L	dsp:16[Rs].memex（注）	—	Rd	4 (memex == UB) 5 (memex != UB)
(2) OR src, src2, dest	L	Rs	Rs2	Rd	3

注 . 本公司的 “RX 族的汇编程序” 的位移量的值 （dsp:8、 dsp:16）必须在长度扩展说明符为 “.W” 或者 “.UW” 时指定为 2 的倍数，在长度扩展说明符为 “.L” 时指定为 4 的倍数。当长度扩展说明符为 “.W” 或者 “.UW” 时，能给 dsp:8 指定 0 ～ 510 （255×2）；当长度扩展说明符为 “.L” 时，能给 dsp:8 指定 0 ～ 1020 （255×4）。当长度扩展说明符为 “.W” 或者 “.UW” 时，能给 dsp:16 指定 0 ～ 131070 （65535×2）；当长度扩展说明符为 “.L” 时，能给 dsp:8 指定 0 ～ 262140 （65535×4）。在指令码里填入位移量除 2 或者除 4 后的值。

【记述例子】

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

POP

寄存器的数据退栈
POP data from the stack

POP

传送指令
【指令码】
记载页：194

【语法】
POP dest

【操作】
```
tmp = *SP;
SP = SP + 4;
dest = tmp;
```

【功能】
- 将退栈的数据传送到dest。
- 使用的堆栈指针为PSW的U位所示的堆栈指针。

【标志的变化】
- 标志不变。

【指令格式】

语法	处理长度	对象	代码长度（字节）
		dest	
POP dest	L	Rd	2

【记述例子】
```
POP R1
```

POPC

控制寄存器的退栈
POP Control register

POPC

传送指令

【语法】
POPC dest

【指令码】
记载页：195

【操作】
```
tmp = *SP;
SP = SP + 4;
dest = tmp;
```

【功能】
- 将退栈的数据传送到dest所示的控制寄存器。
- 使用的堆栈指针为PSW的U位所示的堆栈指针。
- 在用户模式中，忽视写ISP、INTB、BPC、BPSW、FINTV以及PSW的IPL[3:0]位、PM位、U位、I位；在管理模式中，忽视写PSW的PM位。

【标志的变化】

标志	变化	条件
C	（注）	
Z	（注）	
S	（注）	
O	（注）	

注 . 只在 dest 为 PSW 时发生变化。

【指令格式】

语法	处理长度	对象	代码长度
		dest（注）	（字节）
POPC dest	L	Rx	2

注 . 可选择的 dest：ISP、USP、INTB、PSW、BPC、BPSW、FINTV、FPSW
不能给 dest 指定 PC。

【记述例子】
```
POPC PSW
```

POPM

多个寄存器的退栈
POP Multiple registers

POPM

传送指令

【语法】
POPM dest-dest2

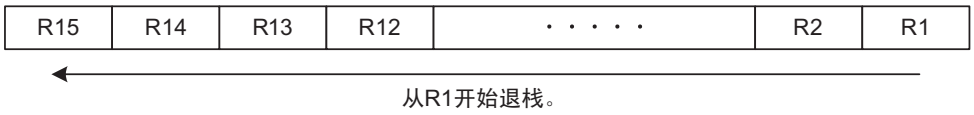
【指令码】
记载页：195

【操作】

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

【功能】

- 将由dest和dest2指定范围的寄存器进行一次性的退栈。
- 用开始寄存器号和最后寄存器号指定范围。但是，必须为 （开始寄存器的寄存器号＜最后寄存器的寄存器号）。
- 不能指定R0。
- 使用的堆栈指针为PSW的U位所示的堆栈指针。
- 退栈的顺序如下：



【标志的变化】

- 标志不变。

【指令格式】

语法	处理长度	对象		代码长度（字节）
		dest	dest2	
POPM dest-dest2	L	Rd (Rd=R1 ～ R14)	Rd2 (Rd2=R2 ～ R15)	2

【记述例子】

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

PUSH

数据的压栈
PUSH data onto the stack

PUSH

传送指令

【语法】
PUSH.size src

【指令码】
记载页：196

【操作】
tmp = src;
SP = SP - 4;（注）
*SP = tmp;

注 . 即使长度说明符 （.size）为 “.B” 或者 “.W”，也从 SP 减去 4。在 “.B” 时高 24 位为不定值；在 “.W” 时高 16 位为不定值。

【功能】
- 将src压栈。
- 当src为寄存器并且长度说明符为 “.B” 或者 “.W” 时，分别将寄存器的LSB侧的字节数据或者字数据压栈。
- 以长字进行堆栈的传送。在长度说明符为 “.B” 时，高24位为不定值；在长度说明符为 “.W” 时，高16位为不定值。
- 使用的堆栈指针为PSW的U位所示的堆栈指针。

【标志的变化】
- 标志不变。

【指令格式】

语法	size	处理长度	对象	代码长度
			src	（字节）
PUSH.size src	B/W/L	L	Rs	2
	B/W/L	L	[Rs]	2
	B/W/L	L	dsp:8[Rs]（注）	3
	B/W/L	L	dsp:16[Rs]（注）	4

注 . 本公司的 “RX 族的汇编程序” 的位移量的值 （dsp:8、 dsp:16）必须在长度说明符为 “.W” 时指定为 2 的倍数，在长度说明符为 “.L” 时指定为 4 的倍数。当长度说明符为 “.W” 时，能给 dsp:8 指定 0 ～ 510 （255×2）；当长度说明符为 “.L” 时，能给 dsp:8 指定 0 ～ 1020 （255×4）。当长度说明符为 “.W” 时，能给 dsp:16 指定 0 ～ 131070 （65535×2）；当长度说明符为 “.L” 时，能给 dsp:16 指定 0 ～ 262140 （65535×4）。在指令码里填入位移量除 2 或者除 4 后的值。

【记述例子】
PUSH.B R1
PUSH.L [R1]

PUSHC

控制寄存器的压栈
PUSH Control register

PUSHC

传送指令

【语法】
PUSH src

【指令码】
记载页：197

【操作】
```
tmp = src;
SP = SP - 4;
*SP = tmp;
```

【功能】
- 将src所示的控制寄存器压栈。
- 使用的堆栈指针为PSW的U位所示的堆栈指针。
- 如果给src指定PC，就将此指令的地址压栈。

【标志的变化】
- 标志不变。

【指令格式】

语法	处理长度	对象	代码长度
		src（注）	（字节）
PUSHC src	L	Rx	2

注 . 可选择的 src：PC、 ISP、 USP、 INTB、 PSW、 BPC、 BPSW、 FINTV、 FPSW

【记述例子】
```
PUSHC PSW
```

PUSHM

多个寄存器的压栈
PUSH Multiple registers

PUSHM

传送指令

【语法】
PUSHM src-src2

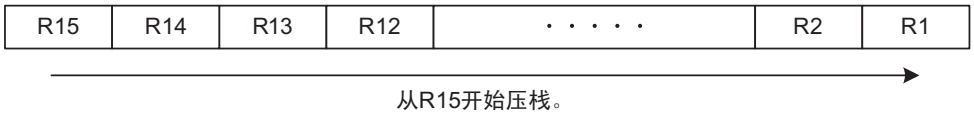
【指令码】
记载页：197

【操作】

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

【功能】

- 将由src和src2指定范围的寄存器进行一次性的压栈。
- 用开始存器号和最后寄存器号指定范围。但是，必须为（开始寄存器的寄存器号＜最后寄存器的寄存器号）。
- 不能指定R0。
- 使用的堆栈指针为PSW的U位所示的堆栈指针。
- 压栈的顺序如下：



【标志的变化】

- 标志不变。

【指令格式】

语法	处理长度	对象		代码长度（字节）
		src	src2	
PUSHM src-src2	L	Rs (Rs=R1 ～ R14)	Rs2 (Rs2=R2 ～ R15)	2

【记述例子】

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

RACW

16 位带符号的累加器舍入处理
Round ACcumulator Word

RACW

DSP 功能指令

【指令码】

记载页：198

【语法】

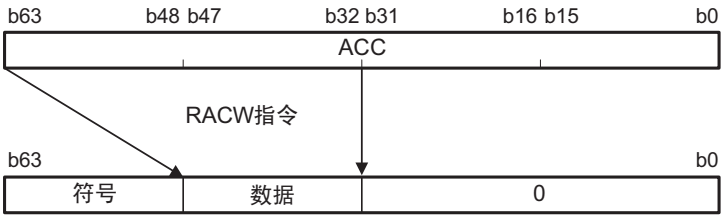
RACW　src

【操作】

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

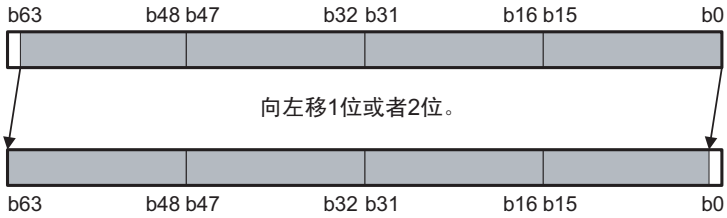
【功能】

- 以字为单位将累加器的值进行舍入，其结果保存到累加器。操作概要图如下所示：

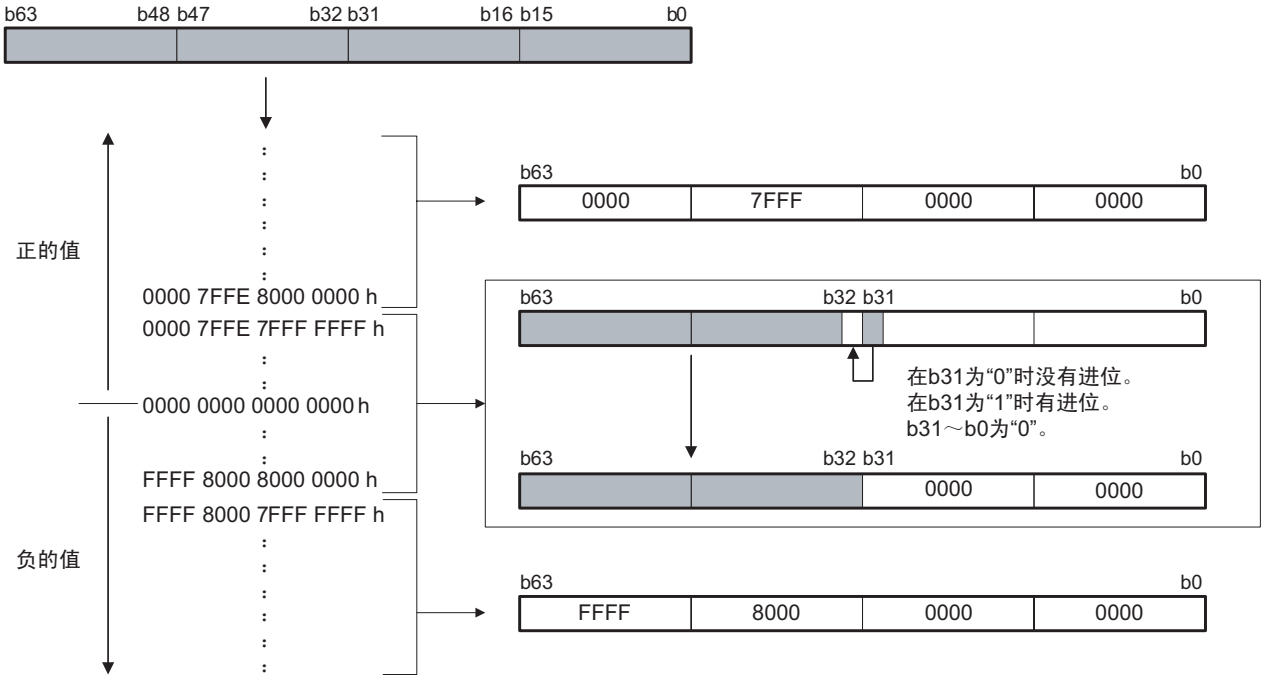


- 按以下步骤执行RACW指令。

处理 1. 将累加器的值向左移 src 指定的位数 （1 位或者 2 位）。



处理 2. 根据向左移 1 位或者 2 位的 64 位的值，累加器的值发生变化。



【标志的变化】

- 标志不变。

【指令格式】

语法	对象	代码长度
	src	（字节）
RACW src	#IMM:1（注） （IMM:1=1 ～ 2）	3

注 . 本公司的 “RX 族的汇编程序” 的立即数 （IMM:1）必须指定 1 ～ 2。在指令码里填入立即数减 1 后的值。

【记述例子】

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

REVL

字节序转换

REVerse Longword data

REVL

传送指令

【语法】

REVL src, dest

【指令码】

记载页：198

【操作】

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

【功能】

- 以字节为单位将src指定的32位数据进行字节序转换，其结果保存到dest。

【标志的变化】

- 标志不变。

【指令格式】

语法	对象		代码长度（字节）
	src	dest	
REVL src, dest	Rs	Rd	3

【记述例子】

REVL R1, R2

REVW

字节序转换

REVerse Word data

REVW

传送指令

【语法】

REVW src, dest

【指令码】

记载页：198

【操作】

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

【功能】

- 以字节为单位分别将src指定的高16位数据和低16位数据进行字节序转换，其结果保存到dest。

【标志的变化】

- 标志不变。

【指令格式】

语法	对象		代码长度（字节）
	src	dest	
REVW src, dest	Rs	Rd	3

【记述例子】

REVW R1, R2

RMPA

乘加运算
Repeated MultiPly and Accumulate

RMPA

算术 / 逻辑运算指令

【语法】
RMPA.size

【指令码】
记载页：199

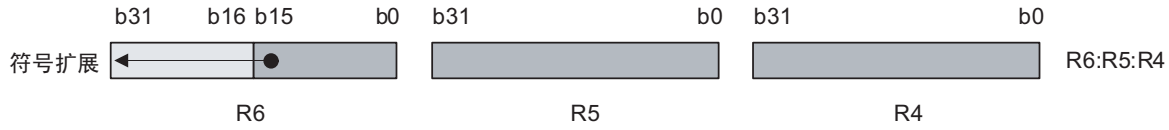
【操作】

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

注 . 1. 如果在给 R3 设定 “0” 后执行，就忽视此指令，寄存器和标志不变。
2. n：在长度说明符 （.size）是 “.B” 时为 “1”；在长度说明符 （.size）是 “.W” 时为 “2”；在长度说明符 （.size）是 “.L” 时为 “4”。

【功能】

- 进行R1为被乘数地址、 R2为乘数地址、 R3为次数的带符号的乘加运算，其结果保存到R6:R5:R4的80位。但是， R6的高16位保存将低16位进行符号扩展的值。
- 能给R3设定的最大值为00010000h。



- 在指令执行结束时， R1和R2的内容为不定值。
- 必须在执行指令前给R6:R5:R4设定初始值。当R5:R4为负时，必须给R6设定 “FFFFFFFFh”； 当R5:R4为正时，必须给R6设定 “00000000h”。
- 如果在指令执行过程中发生中断请求，就在中止运算后接受中断。在从中断子程序返回后，继续执行被中止的处理。在使用此指令并且发生中断时，必须将R1、 R2、 R3、 R4、 R5、 R6和PSW进行压栈和退栈。
- 在执行指令时，可能分别从R1所示的被乘数地址和R2所示的乘数地址中预取数据。但是，预取的数据不超出R3指定的范围。有关预取的数据长度，请参照各产品的硬件手册。

注 . 使用累加器 （ACC），在执行指令后 ACC 的值为不定值。

【标志的变化】

标志	变化	条件
C	—	
Z	—	
S	○	当 R6 的 MSB 是 “1” 时为 “1”，否则为 “0”。
O	○	当 R6:R5:R4 的内容超过 $2^{63}–1$ 或者 $–2^{63}$ 时为 “1”，否则为 “0”。

【指令格式】

语法	size	处理长度	代码长度 （字节）
RMPA.size	B/W/L	size	2

【记述例子】

RMPA.W

ROLC

带进位的左循环
ROtate Left with Carry

ROLC

算术 / 逻辑运算指令

【语法】
ROLC dest

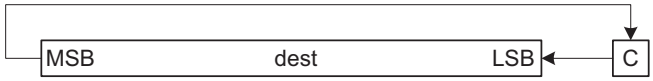
【指令码】
记载页：199

【操作】

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

【功能】

- 包含C标志将dest循环左移1位。



【标志的变化】

标志	变化	条件
C	○	当移出的位是 “1” 时为 “1”，否则为 “0”。
Z	○	当运算后的 dest 是 “0” 时为 “1”，否则为 “0”。
S	○	当运算后的 dest 的 MSB 是 “1” 时为 “1”，否则为 “0”。
O	—	

【指令格式】

语法	处理长度	对象	代码长度（字节）
		dest	
ROLC dest	L	Rd	2

【记述例子】

```
ROLC R1
```

RORC

带进位的右循环
ROtate Right with Carry

RORC

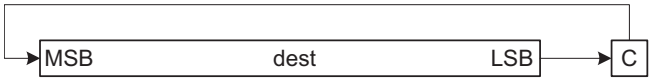
算术 / 逻辑运算指令

【语法】
RORC dest

【指令码】
记载页：199

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

【功能】
- 包含C标志将dest循环右移1位。



【标志的变化】

标志	变化	条件
C	○	当移出的位是 “1” 时为 “1”，否则为 “0”。
Z	○	当运算后的 dest 是 “0” 时为 “1”，否则为 “0”。
S	○	当运算后的 dest 的 MSB 是 “1” 时为 “1”，否则为 “0”。
O	—	

【指令格式】

语法	处理长度	对象	代码长度
		dest	（字节）
RORC dest	L	Rd	2

【记述例子】
```
RORC R1
```

ROTL

左循环
ROTate Left

ROTL

算术 / 逻辑运算指令

【语法】
ROTL src, dest

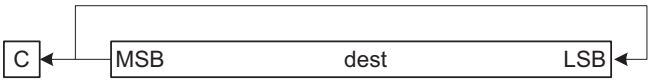
【指令码】
记载页：200

【操作】

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

【功能】

- 将dest进行左循环移位，移动的位数由src指定。从MSB移出的位被传送到LSB和C标志。
- src的值为不带符号的整数， src的范围为0≤src≤31。
- 在src为寄存器时，只有LSB侧的5位有效。



【标志的变化】

标志	变化	条件
C	○	和运算后的 dest 的 LSB 相同，即使在 src 为 “0” 时也和运算后的 dest 的 LSB 相同。
Z	○	当运算后的 dest 是 “0” 时为 “1”，否则为 “0”。
S	○	当运算后的 dest 的 MSB 是 “1” 时为 “1”，否则为 “0”。
O	—	

【指令格式】

语法	处理长度	对象		代码长度（字节）
		src	dest	
ROTL src, dest	L	#IMM:5	Rd	3
	L	Rs	Rd	3

【记述例子】

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

ROTR

右循环
ROTate Right

ROTR

算术 / 逻辑运算指令

【语法】
ROTR src, dest

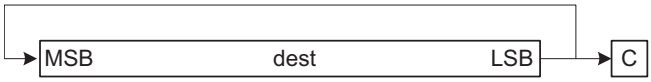
【指令码】
记载页：200

【操作】

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

【功能】

- 将dest进行右循环移位，移动的位数由src指定。从LSB移出的位被传送到MSB和C标志。
- src的值为不带符号的整数， src的范围为0≤src≤31。
- 在src为寄存器时，只有LSB侧的5位有效。



【标志的变化】

标志	变化	条件
C	○	和运算后的 dest 的 MSB 相同，即使在 src 为 “0” 时也和运算后的 dest 的 MSB 相同。
Z	○	当运算后的 dest 是 “0” 时为 “1”，否则为 “0”。
S	○	当运算后的 dest 的 MSB 是 “1” 时为 “1”，否则为 “0”。
O	—	

【指令格式】

语法	处理长度	对象		代码长度（字节）
		src	dest	
ROTR src, dest	L	#IMM:5	Rd	3
	L	Rs	Rd	3

【记述例子】

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

ROUND

浮点数 → 整数的转换
ROUND floating-point to integer

ROUND

浮点运算指令

【语法】
ROUND src, dest

【指令码】
记载页：201

【操作】
dest = (signed long) src;

【功能】
- 将src保存的单度浮点数转换为带符号的长字 （32位）整数，其结果保存到dest。根据FPSW的RM[1:0]位，将结果进行舍入。

RM[1:0] 位的值	舍入模式
00b	向最接近的值舍入
01b	向 0 方向舍入
10b	向 +∞ 方向舍入
11b	向 –∞ 方向舍入

【标志的变化】

标志	变化	条件
C	—	
Z	○	当运算结果是 “0” 时为 “1”，否则为 “0”。
S	○	当运算结果的符号位 （bit31）是 “1” 时为 “1”，是 “0” 时为 “0”。
O	—	
CV	○	当发生无效运算时为 “1”，否则为 “0”。
CO	○	总是为 “0”。
CZ	○	总是为 “0”。
CU	○	总是为 “0”。
CX	○	当发生精度异常时为 “1”，否则为 “0”。
CE	○	当发生非安装处理异常时为 “1”，否则为 “0”。
FV	○	当发生无效运算时为 “1”，否则不变。
FO	—	
FZ	—	
FU	—	
FX	○	当发生精度异常时为 “1”，否则不变。

注 . 在异常处理允许位 EX、 EV 为 “1” 时， FX 标志和 FV 标志不变。在发生异常处理时， S 标志和 Z 标志不变。

【指令格式】

语法	处理长度	对象		代码长度（字节）
		src	dest	
ROUND src, dest	L	Rs	Rd	3
	L	[Rs].L	Rd	3
	L	dsp:8[Rs].L（注）	Rd	4
	L	dsp:16[Rs].L（注）	Rd	5

注 . 本公司的 “RX 族的汇编程序” 的位移量的值 （dsp:8、 dsp:16）必须指定为 4 的倍数。能给 dsp:8 指定 0 ～ 1020 （255×4），并且能给 dsp:16 指定 0 ～ 262140 （65535×4）。在指令码里填入位移量除 4 后的值。

【发生异常】

非安装处理
无效运算
精度异常

【记述例子】

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

【操作补充说明】

- 当DN=0和DN=1时， src和dest的值与运算结果的对应如下所示：

DN=0

src 的值 （指数部为无偏差的值）		dest	异常
src ≥ 0	+∞	当 EV 位为 “1” 并且发生无效运算时：不变 上述以外：7FFFFFFFh	无效运算
	127 ≥指数部≥ 31		
	30 ≥指数部≥ –126	00000000h ～ 7FFFFF80h	无（注 1）
	+ 非规格化数	不变	非安装处理
	+0	00000000h	无
src < 0	–0		
	– 非规格化数	不变	非安装处理
	30 ≥指数部≥ –126	00000000h ～ 80000080h	无（注 1）
	127 ≥指数部≥ 31	当 EV 位为 “1” 并且发生无效运算时：不变 上述以外：80000000h	无效运算（注 2）
	–∞		
NaN	QNaN	当 EV 位为 “1” 并且发生无效运算时：不变 上述以外： 符号位 =0：7FFFFFFFh 符号位 =1：80000000h	无效运算
	SNaN		

注 1. 在进行舍入时，发生精度异常。

注 2. 当 src 为 “CF000000h” 时，不发生无效运算。

DN=1

src 的值 （指数部为无偏差的值）		dest	异常
src ≥ 0	+∞	当 EV 位为 “1”并且发生无效运算时：不变 上述以外：7FFFFFFFh	无效运算
	127 ≥指数部≥ 31		
	30 ≥指数部≥ –126	00000000h ～ 7FFFFF80h	无 （注 1）
	+0、 + 非规格化数	00000000h	无
src <0	–0、 – 非规格化数		
	30 ≥指数部≥ –126	00000000h ～ 80000080h	无 （注 1）
	127 ≥指数部≥ 31	当 EV 位为 “1”并且发生无效运算时：不变 上述以外：80000000h	无效运算 （注 2）
	–∞		
NaN	QNaN	当 EV 位为 “1”并且发生无效运算时：不变 上述以外： 符号位 =0：7FFFFFFFh 符号位 =1：80000000h	无效运算
	SNaN		

注 1. 在进行舍入时，发生精度异常。

注 2. 当 src 为 “CF000000h”时，不发生无效运算。

RTE

异常返回
ReTurn from Exception

RTE

系统操作指令

【语法】
RTE

【指令码】
记载页：201

【操作】

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

【功能】

- 恢复在接受异常时被压栈的PC和PSW，并且从异常处理子程序返回。
- 此指令是特权指令。如果在用户模式中执行，就发生特权指令异常。
- 在转移到用户模式时， PSW的U位为 “1”。

【标志的变化】

标志	变化	条件
C	（注）	
Z	（注）	
S	（注）	
O	（注）	

注 . 为堆栈中的值。

【指令格式】

语法	代码长度 （字节）
RTE	2

【记述例子】

```
RTE
```

RTFI

从高速中断的返回
ReTurn from Fast Interrupt

RTFI

系统操作指令

【语法】
RTFI

【指令码】
记载页：201

【操作】
PSW = BPSW;
PC = BPC;

【功能】
- 分别从BPC和BPSW恢复在接受高速中断请求时被压栈的PC和PSW，并且从高速中断处理程序返回。
- 此指令是特权指令。如果在用户模式中执行，就发生特权指令异常。
- 在转移到用户模式时， PSW的U位为 “1”。
- 在指令执行结束时， BPC和BPSW的值为不定值。

【标志的变化】

标志	变化	条件
C	（注）	
Z	（注）	
S	（注）	
O	（注）	

注 . 为 BPSW 的值。

【指令格式】

语法	代码长度 （字节）
RTFI	2

【记述例子】
RTFI

RTS

从子程序的返回
ReTurn from Subroutine

RTS

转移指令

【语法】
RTS

【指令码】
记载页：202

【操作】
PC = *SP;
SP = SP + 4;

【功能】
- 从子程序返回。

【标志的变化】
- 标志不变。

【指令格式】

语法	代码长度（字节）
RTS	1

【记述例子】
RTS

RTSD

栈帧的释放和从子程序的返回
ReTurn from Subroutine and Deallocate stack frame

RTSD

转移指令

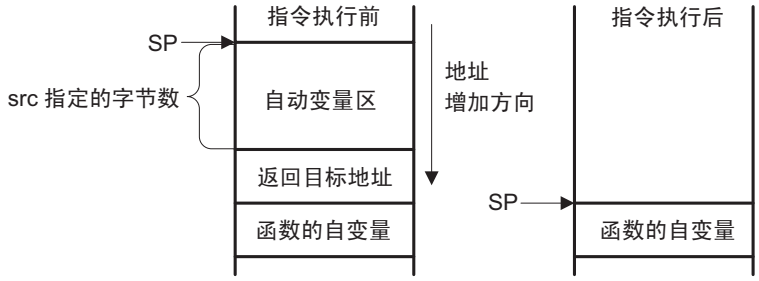
【语法】
(1) RTSD src
(2) RTSD src, dest-dest2

【指令码】
记载页：202

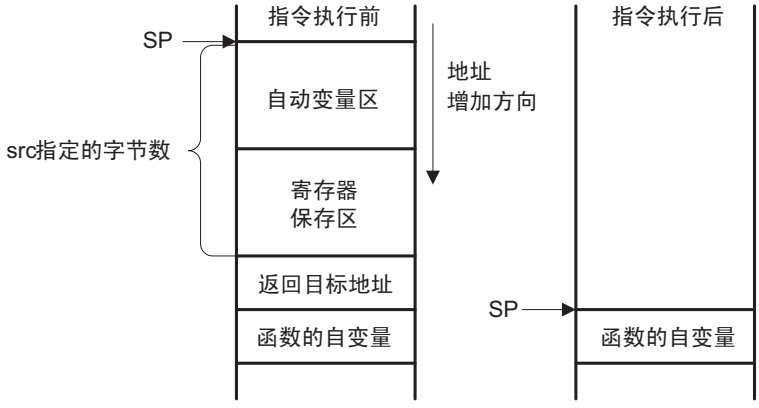
【操作】
(1)
```
SP = SP + src;
PC = *SP;
SP = SP + 4;
```

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

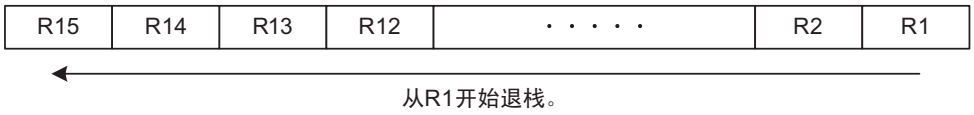
【功能】
1. 在释放栈帧后，从子程序返回。
   - 必须指定src为栈帧（自动变量区）的长度。



2. 在释放栈帧以及恢复寄存器后，从子程序返回。
   - 必须指定src为栈帧（自动变量区和寄存器保存区）的长度。



- 将由dest和dest2指定范围的寄存器进行一次性的退栈。
- 用开始寄存器号和最后寄存器号指定范围。但是，必须为（开始寄存器的寄存器号≤最后寄存器的寄存器号）。
- 不能指定R0。
- 使用的堆栈指针为PSW的U位所示的堆栈指针。
- 退栈的顺序如下：



【标志的变化】

- 标志不变。

【指令格式】

语法	对象			代码长度（字节）
	src	dest	dest2	
(1) RTSD src	#UIMM:8（注）	—	—	2
(2) RTSD src, dest-dest2	#UIMM:8（注）	Rd(Rd=R1 ～ R15)	Rd2(Rd2=R1 ～ R15)	3

注 . 本公司的“RX 族的汇编程序”的立即数必须指定为 4 的倍数。能给 UIMM:8 指定 0 ～ 1020（255×4）。在指令码里填入立即数除 4 后的值。

【记述例子】

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

SAT

32 位带符号的饱和处理
SATurate signed 32-bit data

SAT

算术 / 逻辑运算指令

【语法】
SAT dest

【指令码】
记载页：202

【操作】

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

【功能】

- 进行32位带符号的饱和处理。
- 当O标志是 “1”并且S标志是 “1”时，运算结果为 “7FFFFFFFh”并且保存到dest；当O标志是 “1”并且S标志是 “0”时，运算结果为 “80000000h”并且保存到dest；否则dest不变。

【标志的变化】

- 标志不变。

【指令格式】

语法	处理长度	对象	代码长度
		dest	（字节）
SAT dest	L	Rd	2

【记述例子】

```
SAT R1
```

SATR

RMPA 指令的64 位带符号的饱和处理
SATuRate signed 64-bit data for RMPA

SATR

算术 / 逻辑运算指令

【语法】
SATR

【指令码】
记载页：203

【操作】

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

【功能】

- 进行64位带符号的饱和处理。
- 当O标志是 “1” 并且S标志是 “0” 时，运算结果为 “000000007FFFFFFFFFFFFFFFh” 并且保存到R6:R5:R4；当O标志是 “1” 并且S标志是 “1” 时，运算结果为 “FFFFFFFF8000000000000000h” 并且保存到R6:R5:R4；否则R6:R5:R4不变。

【标志的变化】

- 标志不变。

【指令格式】

语法	代码长度（字节）
SATR	2

【记述例子】

```
SATR
```

SBB

带借位的减法运算
SuBtract with Borrow

SBB

算术 / 逻辑运算指令

【语法】
SBB src, dest

【指令码】
记载页：203

【操作】
dest = dest - src - !C;

【功能】
- 从dest减去src和C标志被取反 （借位）的值，其结果保存到dest。

【标志的变化】

标志	变化	条件
C	○	当不带符号的运算不发生上溢时为 “1”，否则为 “0”。
Z	○	当运算后的 dest 是 “0” 时为 “1”，否则为 “0”。
S	○	当运算后的 dest 的 MSB 是 “1” 时为 “1”，否则为 “0”。
O	○	当带符号的运算发生上溢时为 “1”，否则为 “0”。

【指令格式】

语法	处理长度	对象		代码长度（字节）
		src	dest	
SBB src, dest	L	Rs	Rd	3
	L	[Rs].L	Rd	4
	L	dsp:8[Rs].L （注）	Rd	5
	L	dsp:16[Rs].L （注）	Rd	6

注 . 本公司的 “RX 族的汇编程序” 的位移量的值 （dsp:8、 dsp:16）必须指定为 4 的倍数。能给 dsp:8 指定 0 ～ 1020 （255×4），并且能给 dsp:16 指定 0 ～ 262140 （65535×4）。在指令码里填入位移量除 4 后的值。

【记述例子】
SBB R1, R2
SBB [R1], R2

SCCnd

条件设定
Store Condition Conditionally

SCCnd

传送指令

【语法】
SC*Cnd*.size dest

【指令码】
记载页：204

【操作】
```
if ( Cnd )
  dest = 1;
else
  dest = 0;
```

【功能】
- 给dest设定由*Cnd*所示条件的真假值。真时设定 “1”，假时设定 “0”。
- SC*Cnd*有以下的种类：

SC*Cnd*	条件		式	SC*Cnd*	条件		式
SCGEU, SCC	C == 1	大于等于 / C 标志为 “1”	≤	SCLTU, SCNC	C == 0	小于 / C 标志为 “0”	＞
SCEQ, SCZ	Z = =1	等于 / Z 标志为 “1”	＝	SCNE, SCNZ	Z == 0	不等于 / Z 标志为 “0”	≠
SCGTU	C & ~Z == 1	大于	＜	SCLEU	C & ~Z == 0	小于等于	≥
SCPZ	S == 0	正或者零	0 ≤	SCN	S == 1	负	0 ＞
SCGE	S ^ O == 0	等于或者带符号的大于	≤	SCLE	(S ^ O) | Z == 1	等于或者带符号的小于	≥
SCGT	(S ^ O) | Z == 0	带符号的大于	＜	SCLT	S ^ O == 1	带符号的小于	＞
SCO	O == 1	O 标志为 “1”		SCNO	O == 0	O 标志为 “0”	

【标志的变化】
- 标志不变。

【指令格式】

语法	size	处理长度	对象	代码长度
			dest	（字节）
SC*Cnd*.size dest	L	L	Rd	3
	B/W/L	size	[Rd]	3
	B/W/L	size	dsp:8[Rd] （注）	4
	B/W/L	size	dsp:16[Rd] （注）	5

注 . 本公司的 “RX 族的汇编程序” 的位移量的值 （dsp:8、 dsp:16）必须在长度说明符为 “.W” 时指定为 2 的倍数，在长度说明符为 “.L” 时指定为 4 的倍数。当长度说明符为 “.W” 时，能给 dsp:8 指定 0 ～ 510 （255×2）；当长度说明为 “.L” 时，能给 dsp:8 指定 0 ～ 1020 （255×4）。当长度说明符为 “.W” 时，能给 dsp:16 指定 0 ～ 131070 （65535×2）；当长度说明符为 “.L” 时，能给 dsp:16 指定 0 ～ 262140 （65535×4）。在指令码里填入位移量除 2 或者除 4 后的值。

【记述例子】
```
SCC.L   R2
SCNE.W  [R2]
```

SCMPU

字符串比较
String CoMPare Until not equal

SCMPU

字符串操作指令

【语法】
SCMPU

【指令码】
记载页：204

【操作】

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

注 . 如果在给 R3 设定 “0”后执行，就忽视此指令，寄存器和标志不变。

【功能】

- 以地址增加方向，将R1所示的比较源地址的字符串和R2所示的比较目标地址的字符串进行比较，比较到结果不同或者出现Null字符 ‘\0’（=00h）为止，字节数由R3指定。
- 在执行指令时，可能分别从R1所示的比较源地址和R2所示的比较目标地址中预取数据。但是，预取的数据不超出R3指定的范围。有关预取的数据长度，请参照各产品的硬件手册。
- 在指令执行结束时， R1和R2为不定值。
- 如果在指令执行过程中发生中断请求，就在中止运算后接受中断。在从中断子程序返回后，继续执行被中止的处理。在使用此指令并且发生中断时，必须将R1、 R2、 R3和PSW进行压栈和退栈。

【标志的变化】

标志	变化	条件
C	○	当 (*R1–*R2) 的不带符号的运算结果大于等于 “0”时为 “1”，否则为 “0”。
Z	○	当双方的字符串相同时为 “1”，否则为 “0”。
S	—	
O	—	

【指令格式】

语法	处理长度	代码长度（字节）
SCMPU	B	2

【记述例子】

```
SCMPU
```

SETPSW

PSW 的标志和位的置位
SET flag of PSW

SETPSW

系统操作指令

【语法】
SETPSW dest

【指令码】
记载页：205

【操作】
dest = 1;

【功能】
- 将dest指定的O、 S、 Z、 C标志或者U位、 I位置 “1”。
- 在用户模式中，忽视写PSW的U位和I位；在管理模式中，能写全部标志和位。

【标志的变化】

标志	变化	条件
C	（注）	
Z	（注）	
S	（注）	
O	（注）	

注 . 指定的标志为 “1”。

【指令格式】

语法	对象	代码长度
	dest	（字节）
SETPSW dest	flag	2

【记述例子】
```
SETPSW C
SETPSW Z
```

SHAR

算术右移
SHift Arithmetic Right

SHAR

算术 / 逻辑运算指令

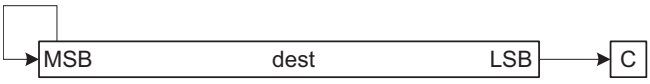
【语法】
(1) SHAR src, dest
(2) SHAR src, src2, dest

【指令码】
记载页：205

【操作】
(1) dest = (signed long) dest >> (src & 31);
(2) dest = (signed long) src2 >> (src & 31);

【功能】
1. 将dest进行算术右移，移动的位数由src指定，其结果保存到dest。
- 从LSB移出的位被传送到C标志。
- src的值为不带符号的整数， src的范围为0≤src≤31。
- 在src为寄存器时，只有LSB侧的5位有效。

2. 在将src2传送到dest后，将dest进行算术右移，移动的位数由src指定，其结果保存到dest。
- 从LSB移出的位被传送到C标志。
- src的值为不带符号的整数， src的范围为0≤src≤31。



【标志的变化】

标志	变化	条件
C	○	当移出的位是 “1” 时为 “1”，否则为 “0”。但是，当 src 是 “0” 时为 “0”。
Z	○	当运算后的 dest 是 “0” 时为 “1”，否则为 “0”。
S	○	当运算后的 dest 的 MSB 是 “1” 时为 “1”，否则为 “0”。
O	○	为 “0”。

【指令格式】

语法	处理长度	对象			代码长度（字节）
		src	src2	dest	
(1) SHAR src, dest	L	#IMM:5	—	Rd	2
	L	Rs	—	Rd	3
(2) SHAR src, src2, dest	L	#IMM:5	Rs	Rd	3

【记述例子】
```
SHAR  #3, R2
SHAR  R1, R2
SHAR  #3, R1, R2
```

SHLL

逻辑 / 算术左移
SHift Logical and arithmetic Left

SHLL

算术 / 逻辑运算指令

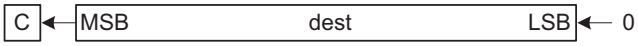
【语法】
(1) SHLL src, dest
(2) SHLL src, src2, dest

【指令码】
记载页：206

【操作】
(1) dest = dest << (src & 31);
(2) dest = src2 << (src & 31);

【功能】
1. 将dest进行逻辑左移，移动的位数由src指定，其结果保存到dest。
- 从MSB移出的位被传送到C标志。
- 在src为寄存器时，只有LSB侧的5位有效。
- src的值为不带符号的整数， src的范围为0≤src≤31。

2. 在将src2传送到dest后，将dest进行逻辑左移，移动的位数由src指定，其结果保存到dest。
- 从MSB溢出的位被传送到C标志。
- src的值为不带符号的整数， src的范围为0≤src≤31。



【标志的变化】

标志	变化	条件
C	○	当移出的位是 “1” 时为 “1”，否则为 “0”。但是，当 src 是 “0 “时为” 0”。
Z	○	当运算后的 dest 是 “0” 时为 “1”，否则为 “0”。
S	○	当运算后的 dest 的 MSB 是 “1” 时为 “1”，否则为 “0”。
O	○	当运算结果的 MSB 的值和移出位的值完全相同时 （移位中的符号不变时）为 “0”，否则为 “1”。但是，当 src 是 “0” 时为 “0”。

【指令格式】

语法	处理长度	对象			代码长度（字节）
		src	src2	dest	
(1) SHLL src, dest	L	#IMM:5	—	Rd	2
	L	Rs	—	Rd	3
(2) SHLL src, src2, dest	L	#IMM:5	Rs	Rd	3

【记述例子】
```
SHLL  #3, R2
SHLL  R1, R2
SHLL  #3, R1, R2
```

SHLR

逻辑右移
SHift Logical Right

SHLR

算术 / 逻辑运算指令

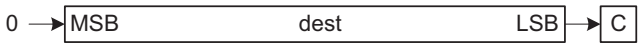
【语法】
(1) SHLR src, dest
(2) SHLR src, src2, dest

【指令码】
记载页：207

【操作】
(1) dest = (unsigned long) dest >> (src & 31);
(2) dest = (unsigned long) src2 >> (src & 31);

【功能】
1. 将dest进行逻辑右移，移动的位数由src指定，其结果保存到dest。
- 从LSB移出的位被传送到C标志。
- src的值为不带符号的整数， src的范围为0≤src≤31。
- 在src为寄存器时，只有LSB侧的5位有效。

2. 在将src2传送到dest后，将dest进行逻辑右移，移动的位数由src指定，其结果保存到dest。
- 从LSB移出的位被传送到C标志。
- src的值为不带符号的整数， src的范围为0≤src≤31。



【标志的变化】

标志	变化	条件
C	○	当移出的位是 “1” 时为 “1”，否则为 “0”。但是，当 src 为 “0 “时为 “0”。
Z	○	当运算后的 dest 是 “0” 时为 “1”，否则为 “0”。
S	○	当运算后的 dest 的 MSB 是 “1” 时为 “1”，否则为 “0”。
O	—	

【指令格式】

语法	处理长度	对象			代码长度（字节）
		src	src2	dest	
(1) SHLR src, dest	L	#IMM:5	—	Rd	2
	L	Rs	—	Rd	3
(2) SHLR src, src2, dest	L	#IMM:5	Rs	Rd	3

【记述例子】
```
SHLR  #3, R2
SHLR  R1, R2
SHLR  #3, R1, R2
```

SMOVB

字符串反向传送
Strings MOVe Backward

SMOVB

字符串操作指令

【语法】
SMOVB

【指令码】
记载页：207

【操作】

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

注 . 如果在给 R3 设定 “0” 后执行，就忽视此指令，寄存器和标志不变。

【功能】

- 以地址减少方向，将R3指定的字节数的字符串，从R2所示的传送源地址传送到R1所示的传送目标地址。
- 在执行指令时，可能从R2所示的传送源地址中预取数据。但是，预取的数据不超出R3指定的范围。有关预取的数据长度，请参照各产品的硬件手册。
- 必须在从R2所示传送源地址中预取的数据范围内不含有R1所示传送目标地址的条件下使用。
- 在指令执行结束时， R1和R2表示最后传送数据的下一个地址。
- 如果在指令执行过程中发生中断请求，就在指令执行中途中止传送后接受中断。在从中断子程序返回后，继续执行被中止的处理。在使用此指令并且发生中断时，必须将R1、 R2、 R3和PSW进行压栈和退栈。

【标志的变化】

- 标志不变。

【指令格式】

语法	处理长度	代码长度（字节）
SMOVB	B	2

【记述例子】

```
SMOVB
```

SMOVF

字符串正向传送
Strings MOVe Forward

SMOVF

字符串操作指令

【语法】
SMOVF

【指令码】
记载页：208

【操作】

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

注 . 如果在给 R3 设定 “0” 后执行，就忽视此指令，寄存器和标志不变。

【功能】

- 以地址增加方向，将R3指定的字节数的字符串，从R2所示的传送源地址传送到R1所示的传送目标地址。
- 在执行指令时，可能从R2所示的传送源地址中预取数据。但是，预取的数据不超出R3指定的范围。有关预取的数据长度，请参照各产品的硬件手册。
- 必须在从R2所示传送源地址中预取的数据范围内不含R1所示传送目标地址的条件下使用。
- 在指令执行结束时， R1和R2表示最后传送数据的下一个地址。
- 如果在指令执行过程中发生中断请求，就在指令执行中途中止传送后接受中断。在从中断子程序返回后，继续执行被中止的处理。在使用此指令并且发生中断时，必须将R1、 R2、 R3和PSW进行压栈和退栈。

【标志的变化】

- 标志不变。

【指令格式】

语法	处理长度	代码长度（字节）
SMOVF	B	2

【记述例子】

```
SMOVF
```

SMOVU

字符串传送
Strings MOVe while Unequal to zero

SMOVU

字符串操作指令

【语法】
SMOVU

【指令码】
记载页：208

【操作】

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

注 . 如果在给 R3 设定 “0”后执行，就忽视此指令，寄存器和标志不变。

【功能】

- 以地址增加方向，将R2所示的传送源地址的字符串传送到R1所示的传送目标地址，传送到出现Null字符 ‘\0’（=00h）为止，字节数由R3指定。在传送Null字符后结束传送。
- 在执行指令时，可能从R2所示的传送源地址中预取数据。但是，预取的数据不超出R3指定的范围。有关预取的数据长度，请参照各产品的硬件手册。
- 必须在从R2所示传送源地址中预取的数据范围内不含R1所示传送目标地址的条件下使用。
- 在指令结束时， R1和R2为不定值。
- 如果在指令执行过程中发生中断请求，就在指令执行中途中止传送后接受中断。在从中断子程序返回后，继续执行被中止的处理。在使用此指令并且发生中断时，必须将R1、 R2、 R3和PSW进行压栈和退栈。

【标志的变化】

- 标志不变。

【指令格式】

语法	处理长度	代码长度（字节）
SMOVU	B	2

【记述例子】

```
SMOVU
```

SSTR

字符串存储
Strings SToRe

SSTR

字符串操作指令

【语法】
SSTR.size

【指令码】
记载页：208

【操作】

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

注 . 1. 如果在给 R3 设定 “0”后执行，就忽视此指令，寄存器和标志不变。
2. R1++：在长度说明符 （.size）是 “.B”时加 1；在长度说明符 （.size）是 “.W”时加 2；在长度说明符 （.size）是 “.L”时加 4。
3. R2：在长度说明符 （.size）是 “.B”时，存储 R2 的 LSB 侧的字节数据；在长度说明符 （.size）是 “.W”时，存储 R2 的 LSB 侧的字数据；在长度说明符 （.size）是 “.L”时，存储 R2 的长字数据。

【功能】

- 以地址增加方向，将R2的内容的字符串存储到R1所示的传送目标地址，存储次数由R3指定。
- 在指令执行结束时， R1表示最后传送数据的下一个地址。
- 如果在指令执行过程中发生中断请求，就在指令执行中途中止传送后接受中断。在从中断子程序返回后，继续执行被中止的处理。在使用此指令并且发生中断时，必须将R1、 R2、 R3和PSW进行压栈和退栈。

【标志的变化】

- 标志不变。

【指令格式】

语法	size	处理长度	代码长度（字节）
SSTR.size	B/W/L	size	2

【记述例子】
SSTR.W

STNZ

带条件的传送
STore on Not Zero

STNZ

传送指令

【语法】
STNZ src, dest

【指令码】
记载页：209

【操作】
```
if ( Z == 0 )
  dest = src;
```

【功能】
- 当Z标志为 “0”时，将src传送到dest；当Z标志为 “1”时， dest不变。

【标志的变化】
- 标志不变。

【指令格式】

语法	处理长度	对象		代码长度（字节）
		src	dest	
STNZ src, dest	L	#SIMM:8	Rd	4
	L	#SIMM:16	Rd	5
	L	#SIMM:24	Rd	6
	L	#IMM:32	Rd	7

【记述例子】
```
STNZ #1, R2
```

STZ

带条件的传送
STore on Zero

STZ

传送命令

【语法】
STZ src, dest

【指令码】
记载页：209

【操作】
```
if ( Z == 1 )
  dest = src;
```

【功能】
- 当Z标志为 “1”时，将src传送到dest；当Z标志为 “0”时， dest不变。

【标志的变化】
- 标志不变。

【指令格式】

语法	处理长度	对象		代码长度（字节）
		src	dest	
STZ src, dest	L	#SIMM:8	Rd	4
	L	#SIMM:16	Rd	5
	L	#SIMM:24	Rd	6
	L	#IMM:32	Rd	7

【记述例子】
```
STZ #1, R2
```

SUB

不带借位的减法运算
SUBtract

SUB

算术 / 逻辑运算指令

【语法】
(1) SUB src, dest
(2) SUB src, src2, dest

【指令码】
记载页：210

【操作】
(1) dest = dest - src;
(2) dest = src2 - src;

【功能】
1. 从dest减去src，其结果保存到dest。
2. 从src2减去src，其结果保存到dest。

【标志的变化】

标志	变化	条件
C	○	当不带符号的运算不发生上溢时为 “1”，否则为 “0”。
Z	○	当运算后的 dest 是 “0” 时为 “1”，否则为 “0”。
S	○	当运算后的 dest 的 MSB 是 “1” 时为 “1”，否则为 “0”。
O	○	当带符号的运算发生上溢时为 “1”，否则为 “0”。

【指令格式】

语法	处理长度	对象			代码长度（字节）
		src	src2	dest	
(1) SUB src, dest	L	#UIMM:4	—	Rd	2
	L	Rs	—	Rd	2
	L	[Rs].memex	—	Rd	2 (memex == UB) 3 (memex != UB)
	L	dsp:8[Rs].memex（注）	—	Rd	3 (memex == UB) 4 (memex != UB)
	L	dsp:16[Rs].memex（注）	—	Rd	4 (memex == UB) 5 (memex != UB)
(2) SUB src, src2, dest	L	Rs	Rs2	Rd	3

注 . 本公司的 “RX 族的汇编程序” 的位移量的值 （dsp:8、 dsp:16）必须在长度扩展说明符为 “.W” 或者 “.UW” 时指定为 2 的倍数，在长度扩展说明符为 “.L” 时指定为 4 的倍数。当长度扩展说明符为 “.W” 或者 “.UW” 时，能给 dsp:8 指定 0 ～ 510 （255×2）；当长度扩展说明符为 “.L” 时，能给 dsp:8 指定 0 ～ 1020 （255×4）。当长度扩展说明符为 “.W” 或者 “.UW” 时，能给 dsp:16 指定 0 ～ 131070 （65535×2）；当长度扩展说明符为 “.L” 时，能给 dsp:16 指定 0 ～ 262140 （65535×4）。在指令码里填入位移量除 2 或者除 4 后的值。

【记述例子】

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

SUNTIL

字符串搜索
Search UNTIL equal string

SUNTIL

字符串操作指令
【指令码】
记载页：211

【语法】
SUNTIL.size

【操作】

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

注 . 1. 如果在给 R3 设定 “0” 后执行，就忽视此指令，寄存器和标志不变。
2. R1++：在长度说明符 （.size）为 “.B” 时加 1；在长度说明符 （.size）为 “.W” 时加 2；在长度说明符 （.size）为 “.L” 时加 4。

【功能】

- 以地址增加方向，从R1所示的比较目标地址检索和R2的内容相同的数据，检索到出现相同的数据为止，比较次数由R3指定。在长度说明符 （.size）为 “.B” 或者 “.W” 时，将存储器的字节数据或者字数据进行零扩展，在扩展为长字数据后和R2的内容进行比较。
- 在执行指令时，可能从R1所示的比较目标地址中预取数据。但是，预取的数据不超出R3指定的范围。有关预取的数据长度，请参照各产品的硬件手册。
- 标志根据 “*R1–R2” 的运算结果而发生变化。
- 在指令执行结束时， R1表示相同数据的下一个地址。如果没有相同的数据，就表示最后传送数据的下一个地址。
- 在指令执行结束后， R3为 “初始值–比较次数”。
- 如果在指令执行过程中发生中断请求，就在指令执行中途中止传送后接受中断。在从中断子程序返回后，继续执行被中止的处理。在使用此指令并且发生中断时，必须将R1、 R2、 R3和PSW进行压栈和退栈。

【标志的变化】

标志	变化	条件
C	○	当不带符号的比较结果大于等于 “0” 时为 “1”，否则为 “0”。
Z	○	当相同时为 “1”，否则为 “0”。
S	—	
O	—	

【指令格式】

语法	size	处理长度	代码长度 （字节）
SUNTIL.size	B/W/L	L	2

【记述例子】

SUNTIL.W

SWHILE

字符串搜索
Search WHILE unequal string

SWHILE

字符串操作指令

【语法】
SWHILE.size

【指令码】
记载页：211

【操作】

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

注 . 1. 如果在给 R3 设定 “0” 后执行，就忽视此指令，寄存器和标志不变。
2. R1++：在长度说明符 （.size）为 “.B” 时加 1；在长度说明符 （.size）为 “.W” 时加 2；在长度说明符 （.size）为 “.L” 时加 4。

【功能】

- 以地址增加方向，从R1所示的比较目标地址检索和R2的内容不同的数据，检索到出现到不同数据为止，比较次数由R3指定。在长度说明符 （.size）为 “.B” 或者 “.W” 时，将存储器的字节数据或者字数据进行零扩展，在扩展为长字数据后和R2的内容进行比较。
- 在执行指令时，可能从R1所示的比较目标地址中预取数据。但是，预取的数据不超出R3指定的范围。有关预取的数据长度，请参照各产品的硬件手册。
- 标志根据 “*R1–R2” 的运算结果而发生变化。
- 在指令执行结束时， R1表示不同数据的下一个地址。如果没有不同的数据，就表示最后传送数据的下一个地址。
- 在指令执行结束后， R3为 “初始值–比较次数”。
- 如果在指令执行过程中发生中断请求，就在指令执行中途中止传送后接受中断。在从中断子程序返回后，继续执行被中止的处理。在使用此指令并且发生中断时，必须将R1、 R2、 R3和PSW进行压栈和退栈。

【标志的变化】

标志	变化	条件
C	○	当不带符号的比较结果大于等于 “0” 时为 “1”，否则为 “0”。
Z	○	当完全相同时为 “1”，否则为 “0”。
S	—	
O	—	

【指令格式】

语法	size	处理长度	代码长度 （字节）
SWHILE.size	B/W/L	L	2

【记述例子】

SWHILE.W

TST

测试
TeST logical

TST

算术 / 逻辑运算指令

【语法】
TST src, src2

【指令码】
记载页：212

【操作】
src2 & src;

【功能】
- 取src2和src的逻辑与， PSW的各标志根据结果而发生变化。

【标志的变化】

标志	变化	条件
C	—	
Z	○	当运算结果是 “0” 时为 “1”，否则为 “0”。
S	○	当运算结果的 MSB 是 “1” 时为 “1”，否则为 “0”。
O	—	

【指令格式】

语法	处理长度	对象		代码长度（字节）
		src	src2	
TST src, src2	L	#SIMM:8	Rs	4
	L	#SIMM:16	Rs	5
	L	#SIMM:24	Rs	6
	L	#IMM:32	Rs	7
	L	Rs	Rs2	3
	L	[Rs].memex	Rs2	3 (memex == UB) 4 (memex != UB)
	L	dsp:8[Rs].memex （注）	Rs2	4 (memex == UB) 5 (memex != UB)
	L	dsp:16[Rs].memex （注）	Rs2	5 (memex == UB) 6 (memex != UB)

注 . 本公司的 “RX 族的汇编程序” 的位移量的值 （dsp:8、 dsp:16）必须在长度扩展说明符为 “.W” 或者 “.UW” 时指定为 2 的倍数，在长度扩展说明符为 “.L” 时指定为 4 的倍数。当长度扩展说明符为 “.W” 或者 “.UW” 时，能给 dsp:8 指定 0 ～ 510 （255×2）；当长度扩展说明符为 “.L” 时，能给 dsp:8 指定 0 ～ 1020 （255×4）。当长度扩展说明符为 “.W” 或者 “.UW” 时，能给 dsp:16 指定 0 ～ 131070 （65535×2）；当长度扩展说明符为 “.L” 时，能给 dsp:16 指定 0 ～ 262140 （65535×4）。在指令码里填入位移量除 2 或者除 4 后的值。

【记述例子】
```
TST   #7, R2
TST   R1, R2
TST   [R1], R2
TST   1[R1].UB, R2
```

WAIT

等待
WAIT

WAIT

系统操作命令
【指令码】
记载页：213

【语法】
WAIT

【操作】

【功能】

- 停止程序的执行。当发生非屏蔽中断、中断或者复位时，开始程序的执行。
- 此指令是特权指令。如果在用户模式中执行，就发生特权指令异常。
- PSW的I位为 “1”。
- 发生中断时被压栈的PC为WAIT指令的下一个地址。

注 . 有关程序停止执行状态下的低功耗状态，请参照各产品的硬件手册。

【标志的变化】

- 标志不变。

【指令格式】

语法	代码长度 （字节）
WAIT	2

【记述例子】
WAIT

XCHG

交换
eXCHanGe

XCHG

传送指令

【语法】
XCHG src, dest

【指令码】
记载页：213

【操作】
tmp = src;
src = dest;
dest = tmp;

【功能】
- 交换src和dest的内容，详细内容如下所示。

src	dest	功能
寄存器	寄存器	将寄存器 （src）的数据和寄存器 （dest）的数据进行交换。
存储器	寄存器	将存储器的数据和寄存器的数据进行交换。当长度扩展说明符为 “.B”或者 “.UB”时，将寄存器的 LSB 侧的字节数据和存储器的数据进行交换；当长度扩展说明符为 “.W”或者 “.UW”时，将寄存器的 LSB 侧的字数据和存储器的数据进行交换；当长度扩展说明符不为 “.L”时，根据指定的扩展方法，将存储器的数据进行扩展，在扩展为长字数据后传送到寄存器。

- 通过安装，此指令也可能用于排他控制。详细内容请参照各产品的硬件手册。

【标志的变化】
- 标志不变。

【指令格式】

语法	处理长度	对象		代码长度（字节）
		src	dest	
XCHG src, dest	L	Rs	Rd	3
	L	[Rs].memex	Rd	3 (memex == UB) 4 (memex != UB)
	L	dsp:8[Rs].memex（注）	Rd	4 (memex == UB) 5 (memex != UB)
	L	dsp:16[Rs].memex（注）	Rd	5 (memex == UB) 6 (memex != UB)

注 . 本公司的 “RX 族的汇编程序”的位移量的值 （dsp:8、 dsp:16）必须在长度扩展说明符为 “.W”或者 “.UW”时指定为 2 的倍数，在长度扩展说明符为 “.L”时指定为 4 的倍数。当长度扩展说明符为 “.W”或者 “.UW”时，能给 dsp:8 指定 0 ～ 510 （255×2）；当长度扩展说明符为 “.L”时，能给 dsp:8 指定 0 ～ 1020 （255×4）。当长度扩展说明符为 “.W”或者 “.UW”时，能给 dsp:16 指定 0 ～ 131070 （65535×2）；当长度扩展说明符为 “.L”时，能给 dsp:16 指定 0 ～ 262140 （65535×4）。在指令码里填入位移量除 2 或者除 4 后的值。

【记述例子】
XCHG R1, R2
XCHG [R1].W, R2

XOR

逻辑异或
eXclusive OR logical

XOR

算术 / 逻辑运算指令

【语法】
XOR src, dest

【指令码】
记载页：214

【操作】
dest = dest ^ src;

【功能】
- 取dest和src的逻辑异或，其结果保存到dest。

【标志的变化】

标志	变化	条件
C	—	
Z	○	当运算后的 dest 是 “0” 时为 “1”，否则为 “0”。
S	○	当运算后的 dest 的 MSB 是 “1” 时为 “1”，否则为 “0”。
O	—	

【指令格式】

语法	处理长度	对象		代码长度（字节）
		src	dest	
XOR src, dest	L	#SIMM:8	Rd	4
	L	#SIMM:16	Rd	5
	L	#SIMM:24	Rd	6
	L	#IMM:32	Rd	7
	L	Rs	Rd	3
	L	[Rs].memex	Rd	3 (memex == UB) 4 (memex != UB)
	L	dsp:8[Rs].memex（注）	Rd	4 (memex == UB) 5 (memex != UB)
	L	dsp:16[Rs].memex（注）	Rd	5 (memex == UB) 6 (memex != UB)

注 . 本公司的 “RX 族的汇编程序” 的位移量的值 （dsp:8、 dsp:16）必须在长度扩展说明符为 “.W” 或者 “.UW” 时指定为 2 的倍数，在长度扩展说明符为 “.L” 时指定为 4 的倍数。当长度扩展说明符为 “.W” 或者 “.UW” 时，能给 dsp:8 指定 0 ～ 510 （255×2）；当长度扩展说明符为 “.L” 时，能给 dsp:8 指定 0 ～ 1020 （255×4）。当长度扩展说明符为 “.W” 或者 “.UW” 时，能给 dsp:16 指定 0 ～ 131070 （65535×2）；当长度扩展说明符为 “.L” 时，能给 dsp:16 指定 0 ～ 262140 （65535×4）。在指令码里填入位移量除 2 或者除 4 后的值。

【记述例子】
```
XOR  #8, R1
XOR  R1, R2
XOR  [R1], R2
XOR  16[R1].L, R2
```

4. 指令码

4.1 本章的阅读方法

在本章中，按各操作码对指令码进行说明。
通过以下的例子说明本章的阅读方法：

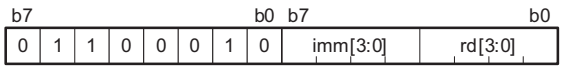
(1) **ADD** **ADD**

【代码长度】

(2)

语法	src	src2	dest	代码长度 （字节）
(1) ADD src, dest	#UIMM:4	—	Rd	2
（为 3 个操作数的指令码。）	#SIMM:8	—	Rd	3
	#SIMM:16	—	Rd	4
	#SIMM:24	—	Rd	5
	#IMM:32	—	Rd	6
(2) ADD src, dest	Rs	—	Rd	2
	[Rs].memex	—	Rd	2(memex == UB) 3(memex != UB)
	dsp:8[Rs].memex	—	Rd	3(memex == UB) 4(memex != UB)
	dsp:16[Rs].memex	—	Rd	4(memex == UB) 5(memex != UB)
(3) ADD src, src2, dest	#SIMM:8	Rs	Rd	3
	#SIMM:16	Rs	Rd	4
	#SIMM:24	Rs	Rd	5
	#IMM:32	Rs	Rd	6
(4) ADD src, src2, dest	Rs	Rs2	Rd	3

(3) (1) ADD src, dest



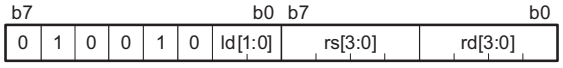
imm[3:0]	src	
0000b ～ 1111b	#UIMM:4	0 ～ 15

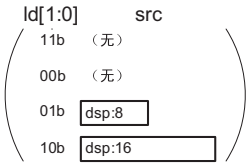
rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(2) ADD src, dest

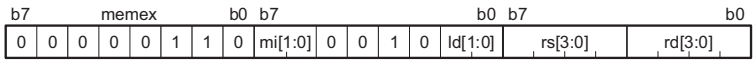
当 memex==UB 或者 src==Rs 时

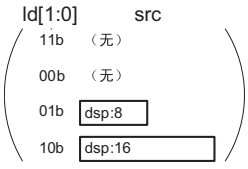
(4)





当 memex !=UB 时





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

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

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

(1) 助记符

表示本页中说明的助记符。

(2) 代码长度表

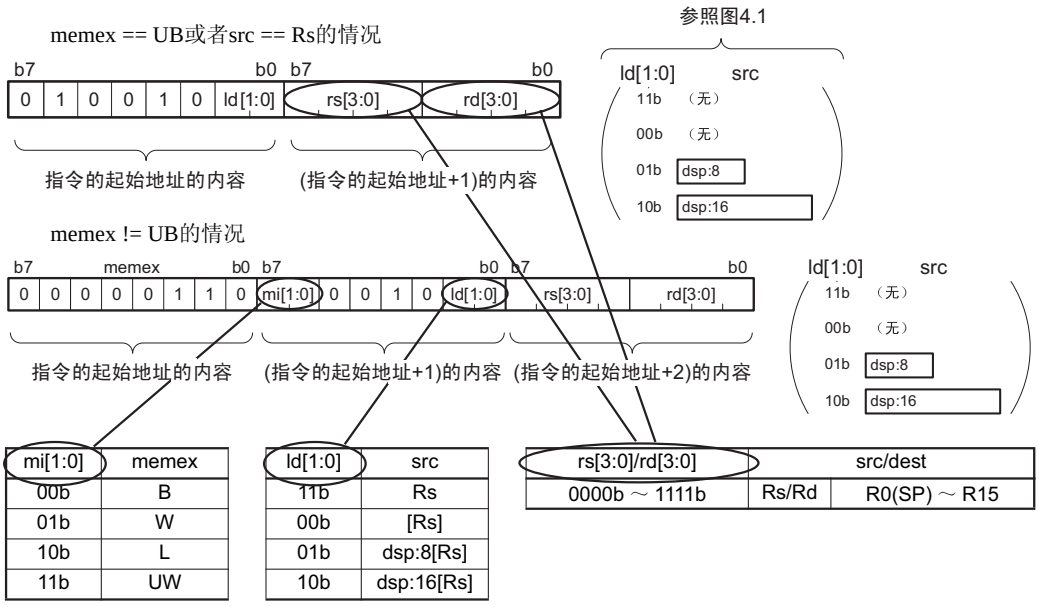
表示指令的字节数。 RX CPU 的指令字节数是 1 字节～ 8 字节。

(3) 语法

用符号表示指令的语法。

(4) 指令码

表示指令码。（ ）内的内容根据所选的 src/dest 而被选择或者省略。



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

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

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

src/dest 的内容 （上一页例子中的 (指令的起始地址 +2) 或者 (指令的起始地址 +3) 以后）的分配如图 4.1 所示。

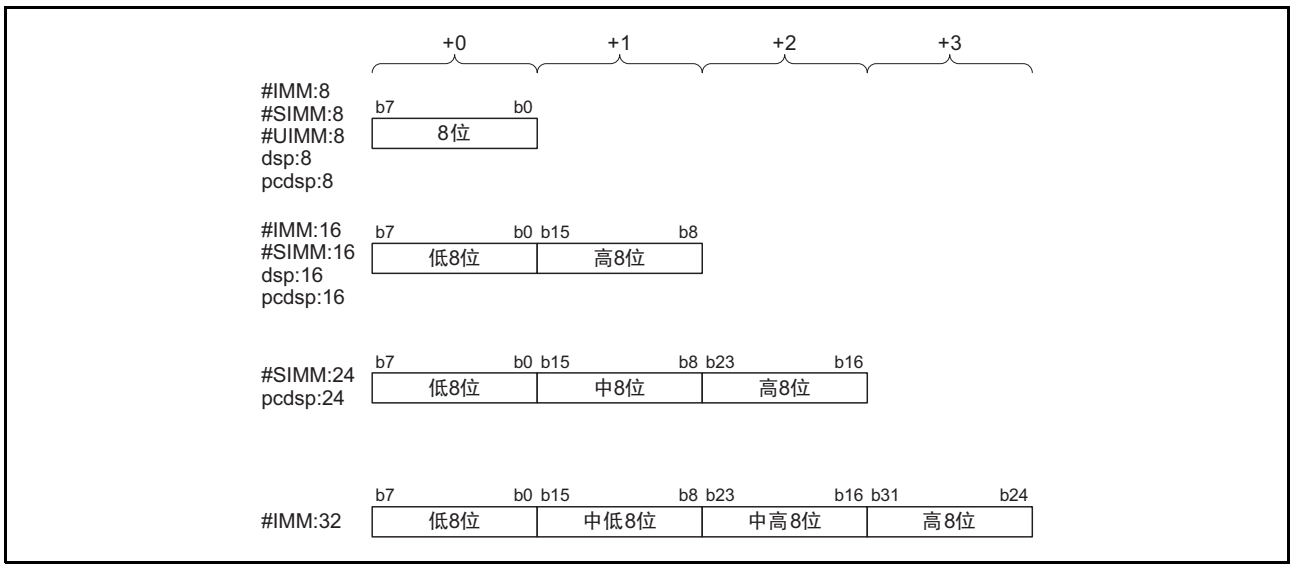


图 4.1 立即数 （IMM）和位移量 （dsp）

rs、 rd、 ld、 mi 等略称的含义如下：

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

4.2 指令码的详细说明

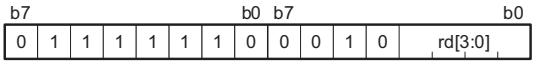
从下一页开始详细说明 RX CPU 的指令码。

ABS ABS

【代码长度】

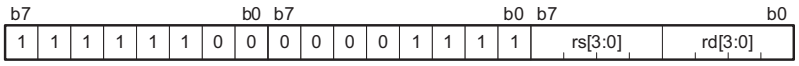
语法	src	dest	代码长度 （字节）
(1) ABS dest	—	Rd	2
(2) ABS src, dest	Rs	Rd	3

(1) ABS dest



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

(2) ABS src, dest



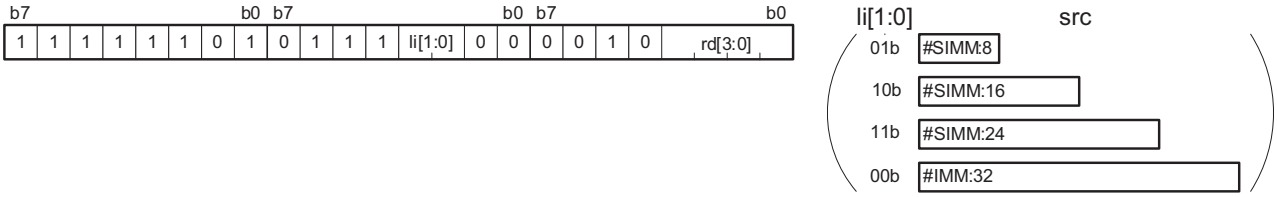
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

ADC ADC

【代码长度】

语法	src	dest	代码长度 （字节）
(1) ADC src, dest	#SIMM:8	Rd	4
	#SIMM:16	Rd	5
	#SIMM:24	Rd	6
	#IMM:32	Rd	7
(2) ADC src, dest	Rs	Rd	3
(3) ADC src, dest	[Rs].L	Rd	4
	dsp:8[Rs].L	Rd	5
	dsp:16[Rs].L	Rd	6

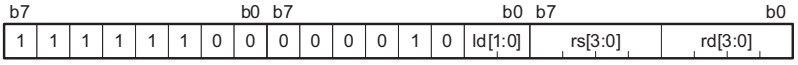
(1) ADC src, dest



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

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

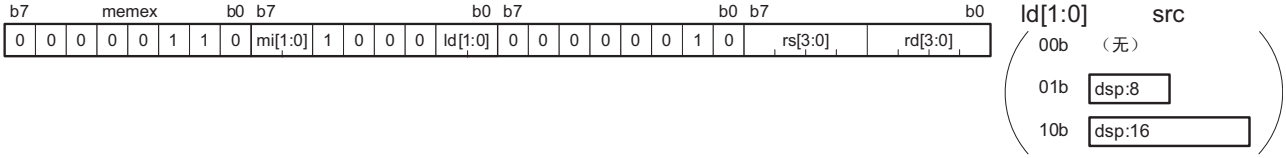
(2) ADC src, dest



ld[1:0]	src
11b	Rs

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

(3) ADC src, dest



mi[1:0]	memex
10b	L

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

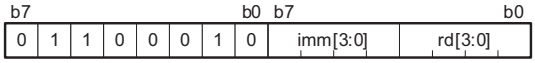
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

ADD ADD

【代码长度】

语法	src	src2	dest	代码长度 （字节）
(1) ADD src, dest	#UIMM:4	—	Rd	2
（为 3 个操作数的指令码。）	#SIMM:8	—	Rd	3
	#SIMM:16	—	Rd	4
	#SIMM:24	—	Rd	5
	#IMM:32	—	Rd	6
(2) ADD src, dest	Rs	—	Rd	2
	[Rs].memex	—	Rd	2 (memex == UB) 3 (memex != UB)
	dsp:8[Rs].memex	—	Rd	3 (memex == UB) 4 (memex != UB)
	dsp:16[Rs].memex	—	Rd	4 (memex == UB) 5 (memex != UB)
(3) ADD src, src2, dest	#SIMM:8	Rs	Rd	3
	#SIMM:16	Rs	Rd	4
	#SIMM:24	Rs	Rd	5
	#IMM:32	Rs	Rd	6
(4) ADD src, src2, dest	Rs	Rs2	Rd	3

(1) ADD src, dest

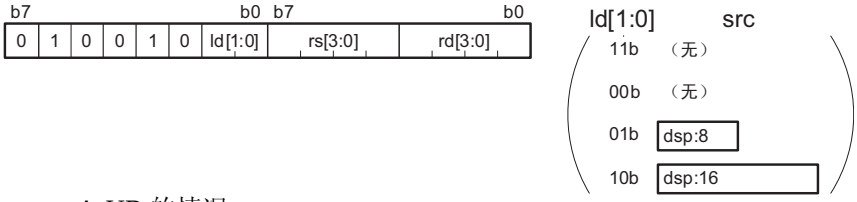


imm[3:0]	src	
0000b ～ 1111b	#UIMM:4	0 ～ 15

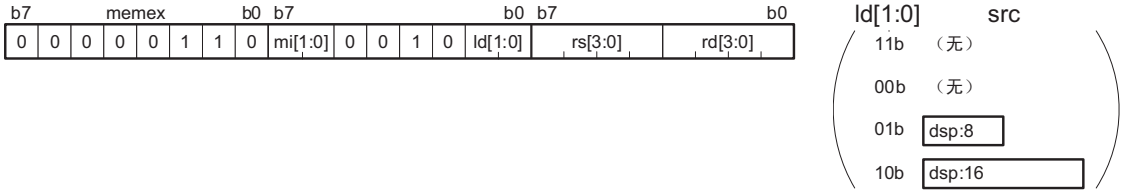
rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(2) ADD src, dest

memex==UB 或者 src==Rs 的情况



memex!=UB 的情况

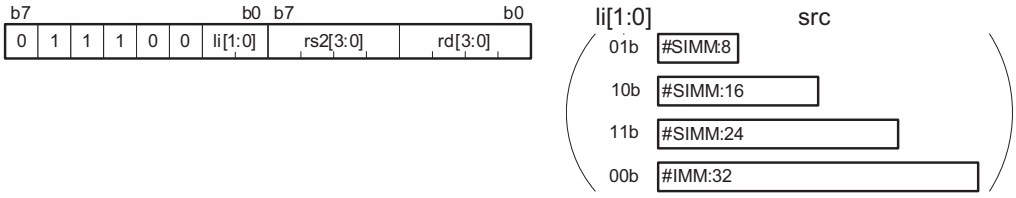


mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

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

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

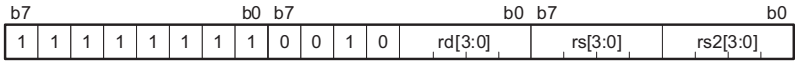
(3) ADD src, src2, dest



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

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

(4) ADD src, src2, dest



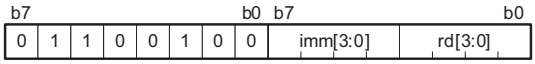
rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b ～ 1111b	Rs/Rs2/Rd	R0(SP) ～ R15

AND AND

【代码长度】

语法	src	src2	dest	代码长度 （字节）
(1) AND src, dest	#UIMM:4	—	Rd	2
(2) AND src, dest	#SIMM:8	—	Rd	3
	#SIMM:16	—	Rd	4
	#SIMM:24	—	Rd	5
	#IMM:32	—	Rd	6
(3) AND src, dest	Rs	—	Rd	2
	[Rs].memex	—	Rd	2 (memex == UB) 3 (memex != UB)
	dsp:8[Rs].memex	—	Rd	3 (memex == UB) 4 (memex != UB)
	dsp:16[Rs].memex	—	Rd	4 (memex == UB) 5 (memex != UB)
(4) AND src, src2, dest	Rs	Rs2	Rd	3

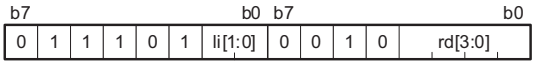
(1) AND src, dest



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

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

(2) AND src, dest

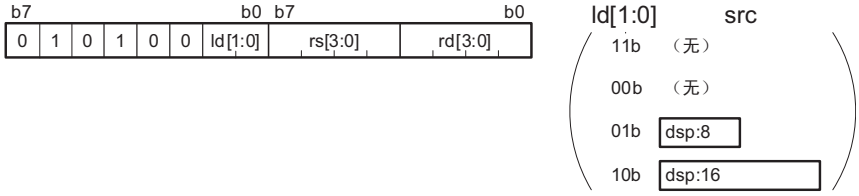


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

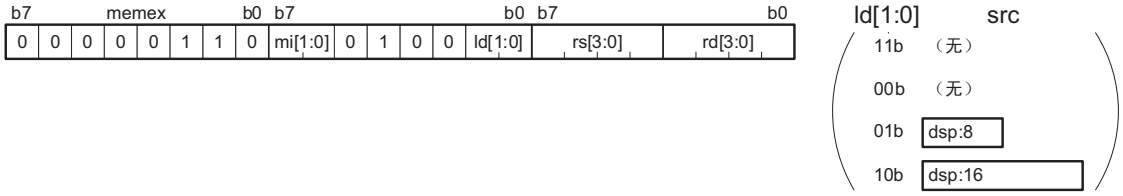
rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(3) AND src, dest

memex==UB 或者 src==Rs 的情况



memex!=UB 的情况

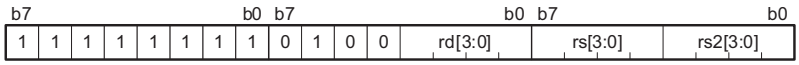


mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

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

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

(4) AND src, src2, dest



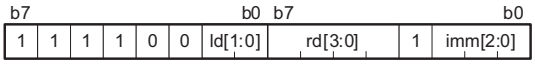
rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b ～ 1111b	Rs/Rs2/Rd	R0(SP) ～ R15

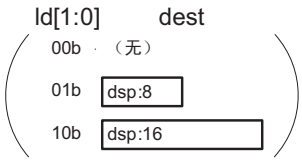
BCLR BCLR

【代码长度】

语法	src	dest	代码长度 （字节）
(1) BCLR src, dest	#IMM:3	[Rd].B	2
	#IMM:3	dsp:8[Rd].B	3
	#IMM:3	dsp:16[Rd].B	4
(2) BCLR src, dest	Rs	[Rd].B	3
	Rs	dsp:8[Rd].B	4
	Rs	dsp:16[Rd].B	5
(3) BCLR src, dest	#IMM:5	Rd	2
(4) BCLR src, dest	Rs	Rd	3

(1) BCLR src, dest



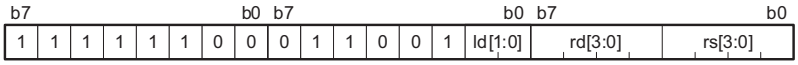


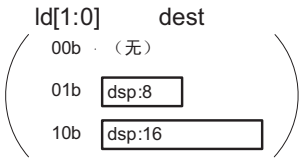
ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

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

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

(2) BCLR src, dest

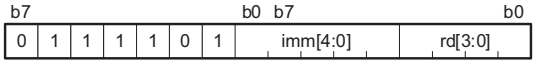




ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

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

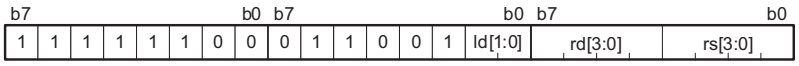
(3) BCLR src, dest



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

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

(4) BCLR src, dest



ld[1:0]	dest
11b	Rd

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

BCnd BCnd

【代码长度】

语法	src	代码长度 （字节）
(1) BCnd.S src	pcdsp:3	1
(2) BCnd.B src	pcdsp:8	2
(3) BCnd.W src	pcdsp:16	3

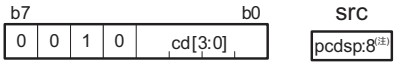
(1) BCnd.S src

注 . 由 dsp[2:0] 指定 pcdsp:3=src。

cd	BCnd
0b	BEQ, BZ
1b	BNE, BNZ

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

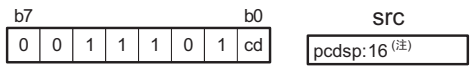
(2) BCnd.B src



注 . pcdsp:8=src 表示的地址 – 指令的起始地址

cd[3:0]	BCnd	cd[3:0]	BCnd
0000b	BEQ, BZ	1000b	BGE
0001b	BNE, BNZ	1001b	BLT
0010b	BGEU, BC	1010b	BGT
0011b	BLTU, BNC	1011b	BLE
0100b	BGTU	1100b	BO
0101b	BLEU	1101b	BNO
0110b	BPZ	1110b	BRA.B
0111b	BN	1111b	（保留）

(3) BCnd.W src



注 . pcdsp:16=src 表示的地址 – 指令的起始地址

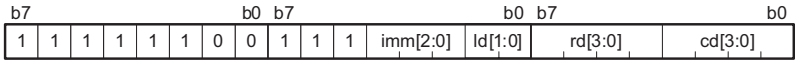
cd	BCnd
0b	BEQ,BZ
1b	BNE,BNZ

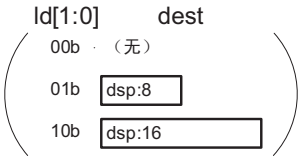
BM*Cnd* BM*Cnd*

【代码长度】

语法	src	dest	代码长度 （字节）
(1) BM*Cnd* src, dest	#IMM:3	[Rd].B	3
	#IMM:3	dsp:8[Rd].B	4
	#IMM:3	dsp:16[Rd].B	5
(2) BM*Cnd* src, dest	#IMM:5	Rd	3

(1) BM*Cnd* src, des





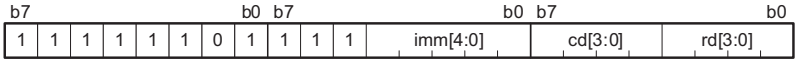
imm[2:0]	src	
000b ～ 111b	#IMM:3	0 ～ 7

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

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

cd[3:0]	BM*Cnd*	cd[3:0]	BM*Cnd*
0000b	BMEQ, BMZ	1000b	BMGE
0001b	BMNE, BMNZ	1001b	BMLT
0010b	BMGEU, BMC	1010b	BMGT
0011b	BMLTU, BMNC	1011b	BMLE
0100b	BMGTU	1100b	BMO
0101b	BMLEU	1101b	BMNO
0110b	BMPZ	1110b	（保留）
0111b	BMN	1111b	（保留）

(2) BM*Cnd* src, dest



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

cd[3:0]	BM*Cnd*	cd[3:0]	BM*Cnd*
0000b	BMEQ, BMZ	1000b	BMGE
0001b	BMNE, BMNZ	1001b	BMLT
0010b	BMGEU, BMC	1010b	BMGT
0011b	BMLTU, BMNC	1011b	BMLE
0100b	BMGTU	1100b	BMO
0101b	BMLEU	1101b	BMNO
0110b	BMPZ	1110b	（保留）
0111b	BMN	1111b	（保留）

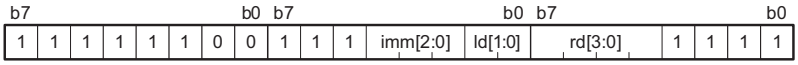
rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

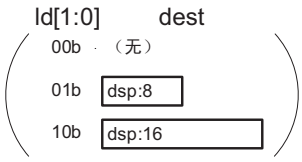
BNOT BNOT

【代码长度】

语法	src	dest	代码长度 （字节）
(1) BNOT src, dest	#IMM:3	[Rd].B	3
	#IMM:3	dsp:8[Rd].B	4
	#IMM:3	dsp:16[Rd].B	5
(2) BNOT src, dest	Rs	[Rd].B	3
	Rs	dsp:8[Rd].B	4
	Rs	dsp:16[Rd].B	5
(3) BNOT src, dest	#IMM:5	Rd	3
(4) BNOT src, dest	Rs	Rd	3

(1) BNOT src, dest



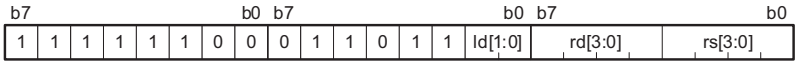


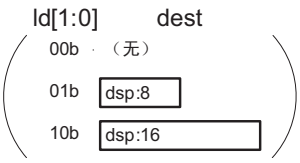
imm[2:0]	src	
000b ～ 111b	#IMM:3	0 ～ 7

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

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

(2) BNOT src, dest

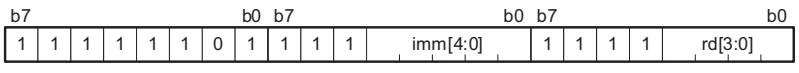




ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

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

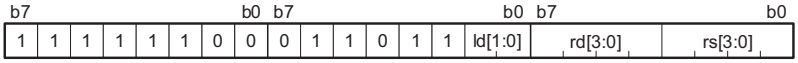
(3) BNOT src, dest



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

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

(4) BNOT src, dest



ld[1:0]	dest
11b	Rd

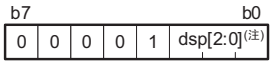
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

BRA BRA

【代码长度】

语法	src	代码长度 （字节）
(1) BRA.S src	pcdsp:3	1
(2) BRA.B src	pcdsp:8	2
(3) BRA.W src	pcdsp:16	3
(4) BRA.A src	pcdsp:24	4
(5) BRA.L src	Rs	2

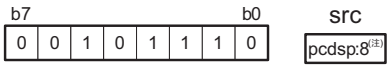
(1) BRA.S src



注 . 由 dsp[2:0] 指定 pcdsp:3=src。

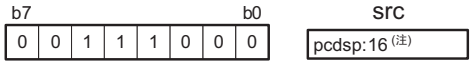
dsp[2:0]	转移距离
011b	3
100b	4
101b	5
110b	6
111b	7
000b	8
001b	9
010b	10

(2) BRA.B src



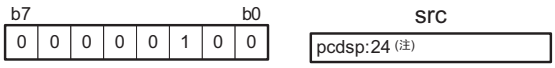
注 . pcdsp:8=src 表示的地址 – 指令的起始地址

(3) BRA.W src



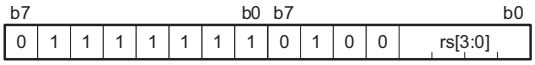
注 . pcdsp:16=src 表示的地址 – 指令的起始地址

(4) BRA.A src



注 . pcdsp:24=src 表示的地址 – 指令的起始地址

(5) BRA.L src



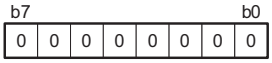
rs[3:0]	src	
0000b ～ 1111b	Rs	R0(SP) ～ R15

BRK BRK

【代码长度】

语法	代码长度 （字节）
(1) BRK	1

(1) BRK

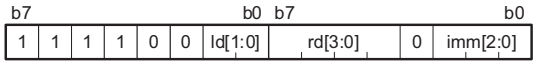


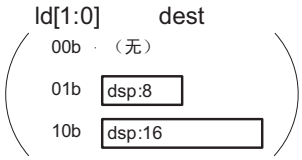
BSET BSET

【代码长度】

语法	src	dest	代码长度 （字节）
(1) BSET src, dest	#IMM:3	[Rd].B	2
	#IMM:3	dsp:8[Rd].B	3
	#IMM:3	dsp:16[Rd].B	4
(2) BSET src, des	Rs	[Rd].B	3
	Rs	dsp:8[Rd].B	4
	Rs	dsp:16[Rd].B	5
(3) BSET src, dest	#IMM:5	Rd	2
(4) BSET src, dest	Rs	Rd	3

(1) BSET src, dest



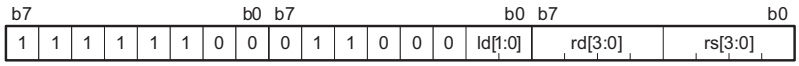


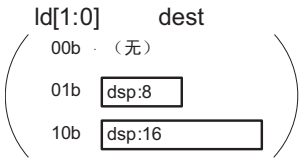
ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

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

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

(2) BSET src, dest

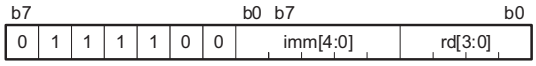




ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

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

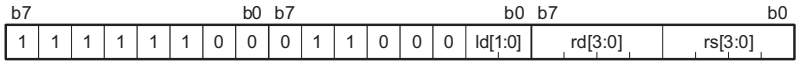
(3) BSET src, dest



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

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

(4) BSET src, dest



ld[1:0]	dest
11b	Rd

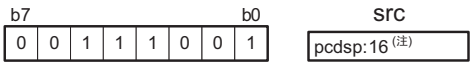
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

BSR BSR

【代码长度】

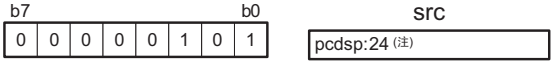
语法	src	代码长度 （字节）
(1) BSR.W src	pcdsp:16	3
(2) BSR.A src	pcdsp:24	4
(3) BSR.L src	Rs	2

(1) BSR.W src



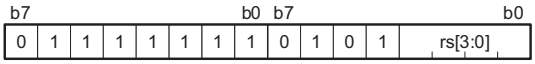
注 . pcdsp:16=src 表示的地址 – 指令的起始地址

(2) BSR.A src



注 . pcdsp:24=src 表示的地址 – 指令的起始地址

(3) BSR.L src



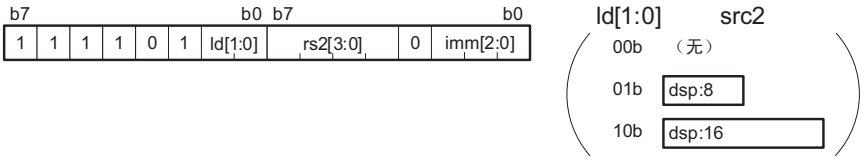
rs[3:0]	src	
0000b ～ 1111b	Rs	R0(SP) ～ R15

BTST BTST

【代码长度】

语法	src	src2	代码长度 （字节）
(1) BTST src, src2	#IMM:3	[Rs2].B	2
	#IMM:3	dsp:8[Rs2].B	3
	#IMM:3	dsp:16[Rs2].B	4
(2) BTST src, src2	Rs	[Rs2].B	3
	Rs	dsp:8[Rs2].B	4
	Rs	dsp:16[Rs2].B	5
(3) BTST src, src2	#IMM:5	Rs2	2
(4) BTST src, src2	Rs	Rs2	3

(1) BTST src, src2

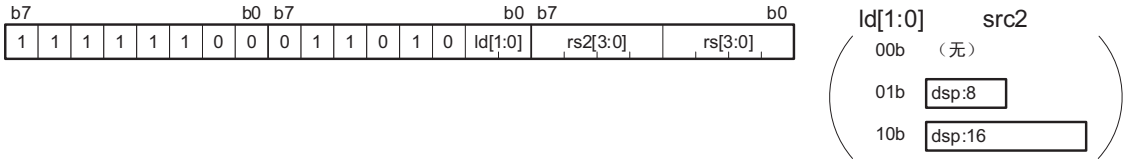


ld[1:0]	src2
00b	[Rs2]
01b	dsp:8[Rs2]
10b	dsp:16[Rs2]

rs2[3:0]	src2	
0000b ～ 1111b	Rs2	R0(SP) ～ R15

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

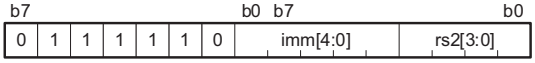
(2) BTST src, src2



ld[1:0]	src2
00b	[Rs2]
01b	dsp:8[Rs2]
10b	dsp:16[Rs2]

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

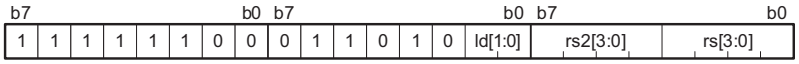
(3) BTST src, src2



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

rs2[3:0]	src2	
0000b ～ 1111b	Rs2	R0(SP) ～ R15

(4) BTST src, src2



ld[1:0]	src2
11b	Rs2

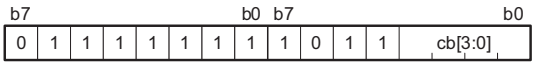
rs[3:0]/rs2[3:0]	src/src2	
0000b ～ 1111b	Rs/Rs2	R0(SP) ～ R15

CLRPSW CLRPSW

【代码长度】

语法	dest	代码长度 （字节）
(1) CLRPSW dest	flag	2

(1) CLRPSW dest



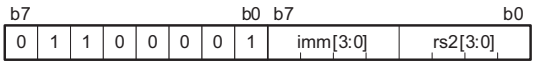
cb[3:0]	dest	
0000b	flag	C
0001b		Z
0010b		S
0011b		O
0100b		（保留）
0101b		（保留）
0110b		（保留）
0111b		（保留）
1000b		I
1001b		U
1010b		（保留）
1011b		（保留）
1100b		（保留）
1101b		（保留）
1110b		（保留）
1111b		（保留）

CMP CMP

【代码长度】

语法	src	src2	代码长度 （字节）
(1) CMP src, src2	#UIMM:4	Rs	2
(2) CMP src, src2	#UIMM:8	Rs	3
(3) CMP src, src2	#SIMM:8	Rs	3
	#SIMM:16	Rs	4
	#SIMM:24	Rs	5
	#IMM:32	Rs	6
(4) CMP src, src2	Rs	Rs2	2
	[Rs].memex	Rs2	2 (memex == UB) 3 (memex != UB)
	dsp:8[Rs].memex	Rs2	3 (memex == UB) 4 (memex != UB)
	dsp:16[Rs].memex	Rs2	4 (memex == UB) 5 (memex != UB)

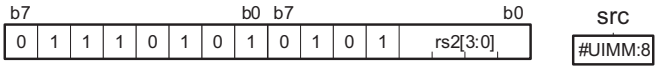
(1) CMP src, src2



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

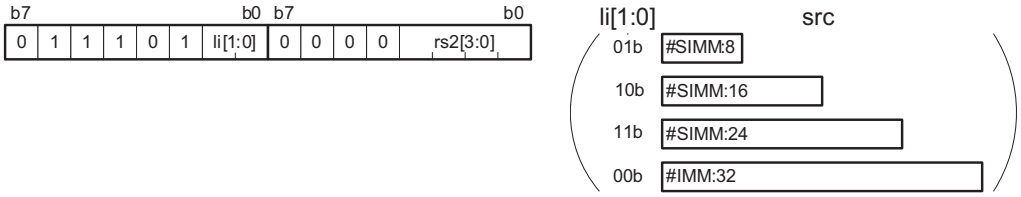
rs2[3:0]	src2	
0000b ～ 1111b	Rs	R0(SP) ～ R15

(2) CMP src, src2



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

(3) CMP src, src2

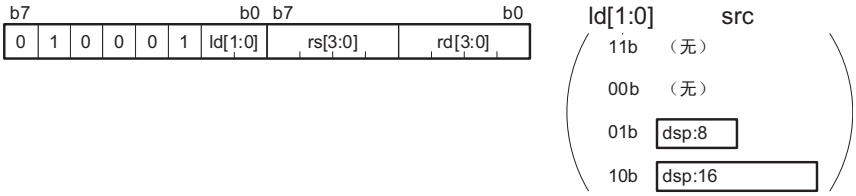


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

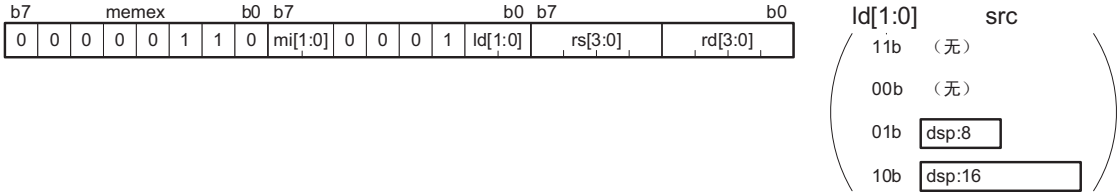
rs2[3:0]	src2	
0000b ～ 1111b	Rs	R0(SP) ～ R15

(4) CMP src, src2

memex==UB 或者 src==Rs 的情况



memex!=UB 的情况



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

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

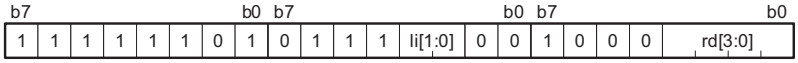
rs[3:0]/rs2[3:0]	src/src2	
0000b ～ 1111b	Rs/Rs2	R0(SP) ～ R15

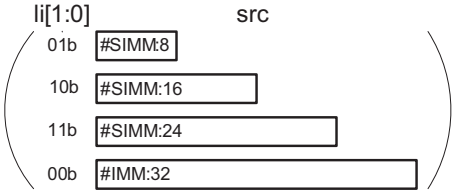
DIV DIV

【代码长度】

语法	src	dest	代码长度 （字节）
(1) DIV src, dest	#SIMM:8	Rd	4
	#SIMM:16	Rd	5
	#SIMM:24	Rd	6
	#IMM:32	Rd	7
(2) DIV src, dest	Rs	Rd	3
	[Rs].memex	Rd	3 (memex == UB) 4 (memex != UB)
	dsp:8[Rs].memex	Rd	4 (memex == UB) 5 (memex != UB)
	dsp:16[Rs].memex	Rd	5 (memex == UB) 6 (memex != UB)

(1) DIV src, dest



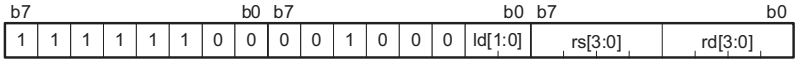


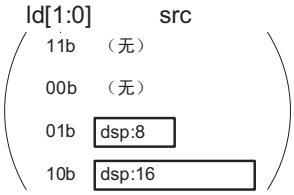
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

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

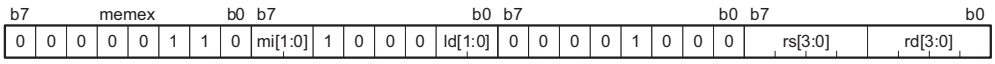
(2) DIV src, dest

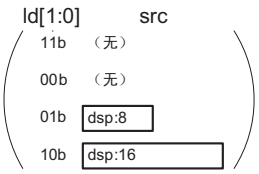
memex==UB 或者 src==Rs 的情况





memex!=UB 的情况





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

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

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

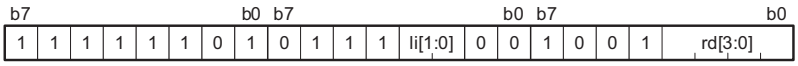
DIVU

DIVU

【代码长度】

语法	src	dest	代码长度 （字节）
(1) DIVU src, dest	#SIMM:8	Rd	4
	#SIMM:16	Rd	5
	#SIMM:24	Rd	6
	#IMM:32	Rd	7
(2) DIVU src, dest	Rs	Rd	3
	[Rs].memex	Rd	3 (memex == UB) 4 (memex != UB)
	dsp:8[Rs].memex	Rd	4 (memex == UB) 5 (memex != UB)
	dsp:16[Rs].memex	Rd	5 (memex == UB) 6 (memex != UB)

(1) DIVU src, dest

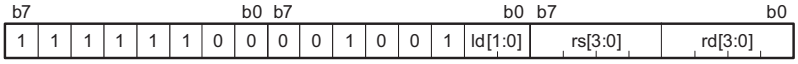


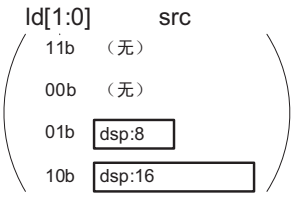
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

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

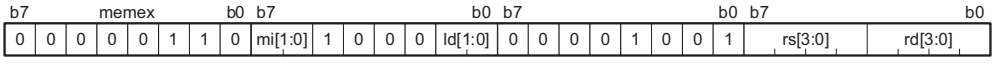
(2) DIVU src, dest

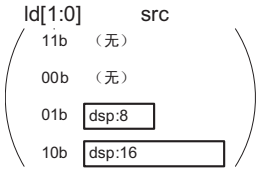
memex==UB 或者 src==Rs 的情况





memex!=UB 的情况





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

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

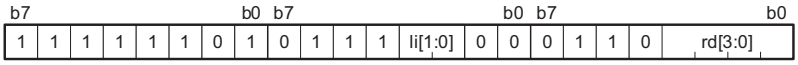
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

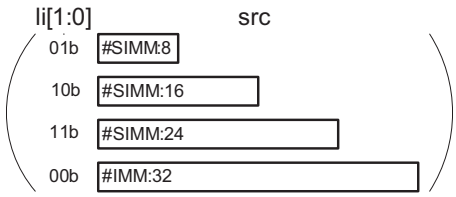
EMUL EMUL

【代码长度】

语法	src	dest	代码长度 （字节）
(1) EMUL src, dest	#SIMM:8	Rd	4
	#SIMM:16	Rd	5
	#SIMM:24	Rd	6
	#IMM:32	Rd	7
(2) EMUL src, dest	Rs	Rd	3
	[Rs].memex	Rd	3 (memex = UB) 4 (memex != UB)
	dsp:8[Rs].memex	Rd	4 (memex == UB) 5 (memex != UB)
	dsp:16[Rs].memex	Rd	5 (memex == UB) 6 (memex != UB)

(1) EMUL src, dest



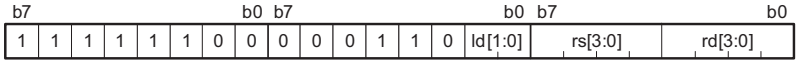


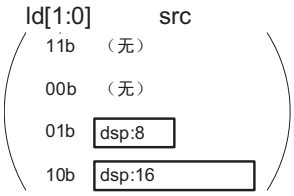
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

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

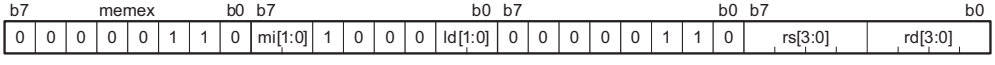
(2) EMUL src, dest

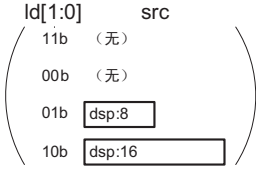
memex==UB 或者 src==Rs 的情况





memex!=UB 的情况





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

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

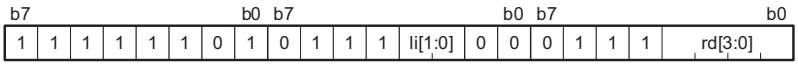
rs[3:0]	src	
0000b ～ 1111b	Rs	R0(SP) ～ R15
rd[3:0]	dest	
0000b ～ 1110b	Rd	R0(SP) ～ R14

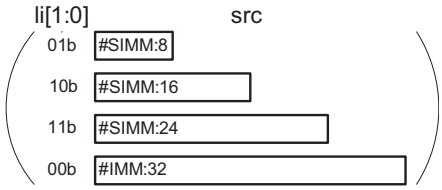
EMULU EMULU

【代码长度】

语法	src	dest	代码长度 （字节）
(1) EMULU src, dest	#SIMM:8	Rd	4
	#SIMM:16	Rd	5
	#SIMM:24	Rd	6
	#IMM:32	Rd	7
(2) EMULU src, dest	Rs	Rd	3
	[Rs].memex	Rd	3 (memex == UB) 4 (memex != UB)
	dsp:8[Rs].memex	Rd	4 (memex == UB) 5 (memex != UB)
	dsp:16[Rs].memex	Rd	5 (memex == UB) 6 (memex != UB)

(1) EMULU src, dest



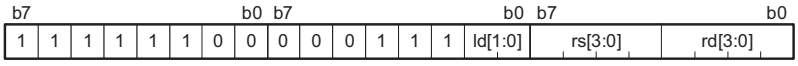


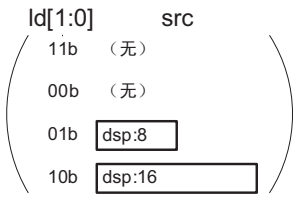
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

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

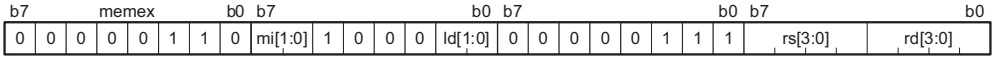
(2) EMULU src, dest

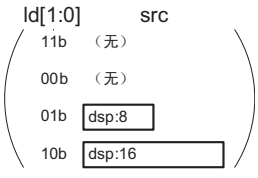
memex==UB 或者 src==Rs 的情况





memex!=UB 的情况





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

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

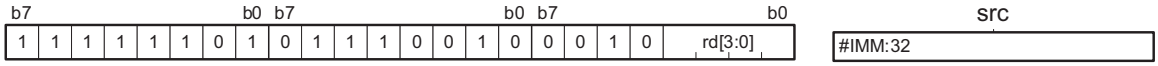
rs[3:0]	src	
0000b ～ 1111b	Rs	R0(SP) ～ R15
rd[3:0]	dest	
0000b ～ 1110b	Rd	R0(SP) ～ R14

FADD FADD

【代码长度】

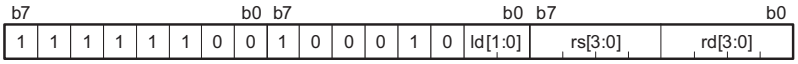
语法	src	dest	代码长度 （字节）
(1) FADD src, dest	#IMM:32	Rd	7
(2) FADD src, dest	Rs	Rd	3
	[Rs].L	Rd	3
	dsp:8[Rs].L	Rd	4
	dsp:16[Rs].L	Rd	5

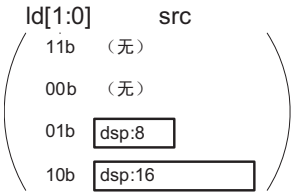
(1) FADD src, dest



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

(2) FADD src, dest





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

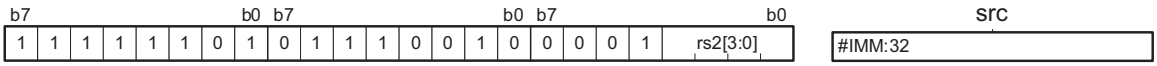
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

FCMP FCMP

【代码长度】

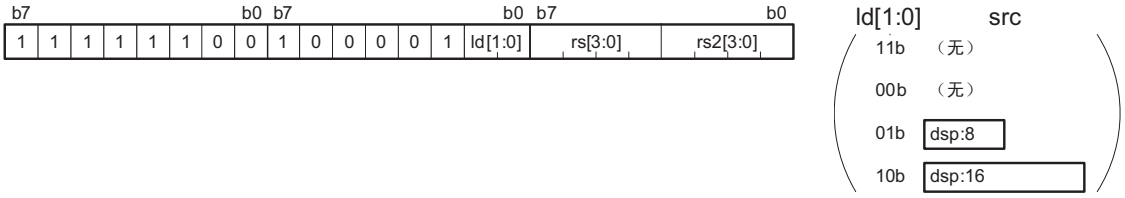
语法	src	src2	代码长度 （字节）
(1) FCMP src, src2	#IMM:32	Rs2	7
(2) FCMP src, src2	Rs	Rs2	3
	[Rs].L	Rs2	3
	dsp:8[Rs].L	Rs2	4
	dsp:16[Rs].L	Rs2	5

(1) FCMP src, src2



rs2[3:0]	src2	
0000b ～ 1111b	Rs2	R0(SP) ～ R15

(2) FCMP src, src2



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

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

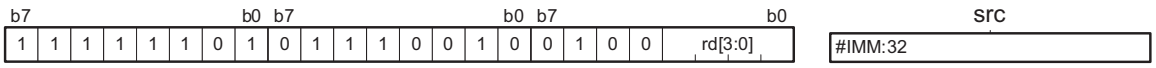
FDIV

FDIV

【代码长度】

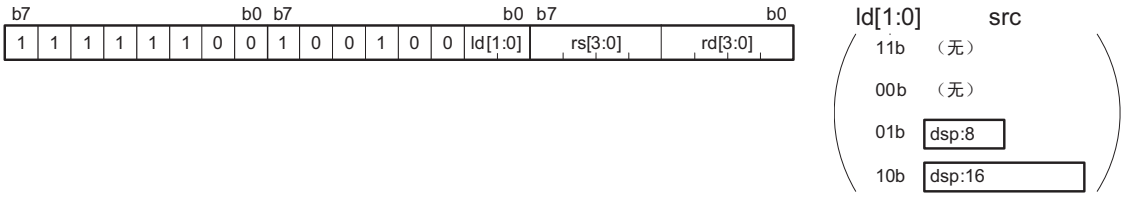
语法	src	dest	代码长度 （字节）
(1) FDIV src, dest	#IMM:32	Rd	7
(2) FDIV src, dest	Rs	Rd	3
	[Rs].L	Rd	3
	dsp:8[Rs].L	Rd	4
	dsp:16[Rs].L	Rd	5

(1) FDIV src, dest



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

(2) FDIV src, dest



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

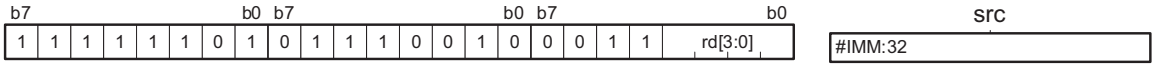
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

FMUL FMUL

【代码长度】

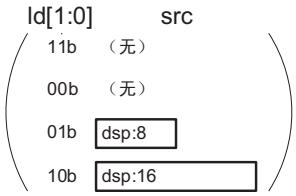
语法	src	dest	代码长度 （字节）
(1) FMUL src, dest	#IMM:32	Rd	7
(2) FMUL src, dest	Rs	Rd	3
	[Rs].L	Rd	3
	dsp:8[Rs].L	Rd	4
	dsp:16[Rs].L	Rd	5

(1) FMUL src, dest



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

(2) FMUL src, dest



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

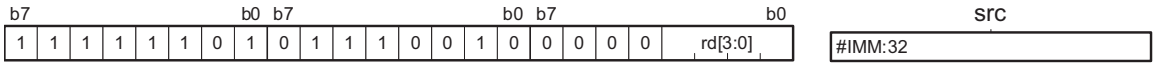
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

FSUB FSUB

【代码长度】

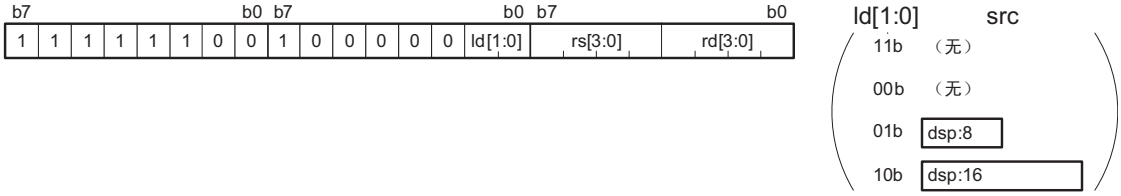
语法	src	dest	代码长度 （字节）
(1) FSUB src, dest	#IMM:32	Rd	7
(2) FSUB src, dest	Rs	Rd	3
	[Rs].L	Rd	3
	dsp:8[Rs].L	Rd	4
	dsp:16[Rs].L	Rd	5

(1) FSUB src, dest



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

(2) FSUB src, dest



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

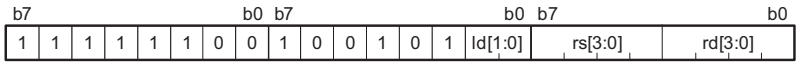
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

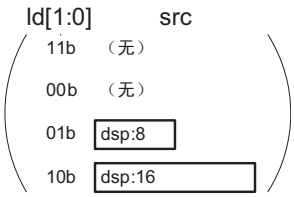
FTOI FTOI

【代码长度】

语法	src	dest	代码长度 （字节）
(1) FTOI src, dest	Rs	Rd	3
	[Rs].L	Rd	3
	dsp:8[Rs].L	Rd	4
	dsp:16[Rs].L	Rd	5

(1) FTOI src, dest





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

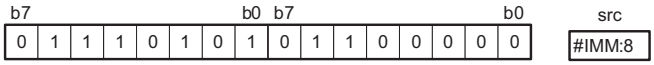
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

INT INT

【代码长度】

语法	src	代码长度 （字节）
(1) INT src	#IMM:8	3

(1) INT src



ITOF

ITOF

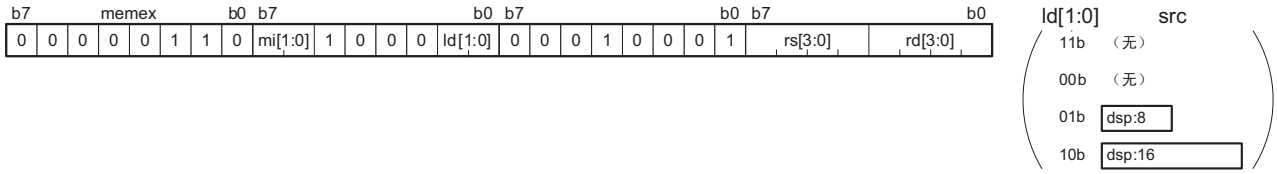
【代码长度】

语法	src	dest	代码长度 （字节）
(1) ITOF src, dest	Rs	Rd	3
	[Rs].memex	Rd	3(memex == UB) 4(memex != UB)
	dsp:8[Rs].memex	Rd	4(memex == UB) 5(memex != UB)
	dsp:16[Rs].memex	Rd	5(memex == UB) 6(memex != UB)

(1) ITOF src, dest

memex==UB 或者 src==Rs 的情况

memex!=UB 的情况



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

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

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

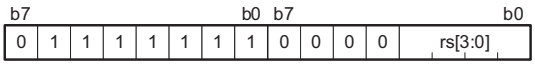
JMP

JMP

【代码长度】

语法	src	代码长度 （字节）
(1) JMP src	Rs	2

(1) JMP src



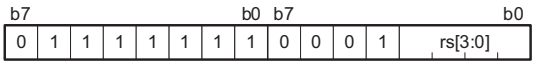
rs[3:0]	src	
0000b ～ 1111b	Rs	R0(SP) ～ R15

JSR JSR

【代码长度】

语法	src	代码长度 （字节）
(1) JSR src	Rs	2

(1) JSR src



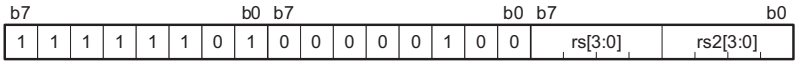
rs[3:0]	src	
0000b ～ 1111b	Rs	R0(SP) ～ R15

MACHI MACHI

【代码长度】

语法	src	src2	代码长度 （字节）
(1) MACHI src, src2	Rs	Rs2	3

(1) MACHI src, src2



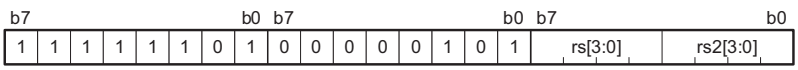
rs[3:0]/rs2[3:0]	src/src2	
0000b ～ 1111b	Rs/Rs2	R0(SP) ～ R15

MACLO MACLO

【代码长度】

语法	src	src2	代码长度 （字节）
(1) MACLO src, src2	Rs	Rs2	3

(1) MACLO src, src2



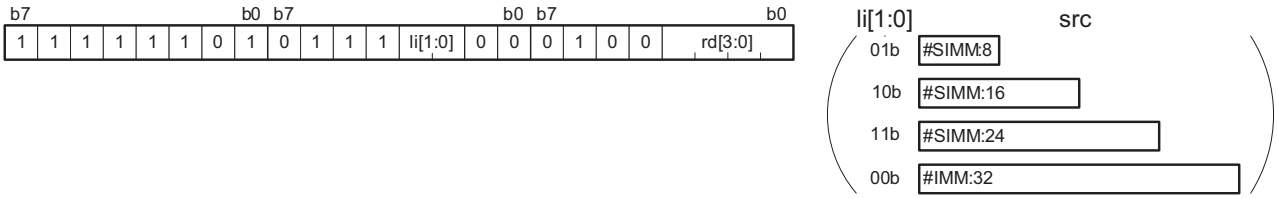
rs[3:0]/rs2[3:0]	src/src2	
0000b ～ 1111b	Rs/Rs2	R0(SP) ～ R15

MAX MAX

【代码长度】

语法	src	dest	代码长度 （字节）
(1) MAX src, dest	#SIMM:8	Rd	4
	#SIMM:16	Rd	5
	#SIMM:24	Rd	6
	#IMM:32	Rd	7
(2) MAX src, dest	Rs	Rd	3
	[Rs].memex	Rd	3 (memex == UB) 4 (memex != UB)
	dsp:8[Rs].memex	Rd	4 (memex == UB) 5 (memex != UB)
	dsp:16[Rs].memex	Rd	5 (memex == UB) 6 (memex != UB)

(1) MAX src, dest

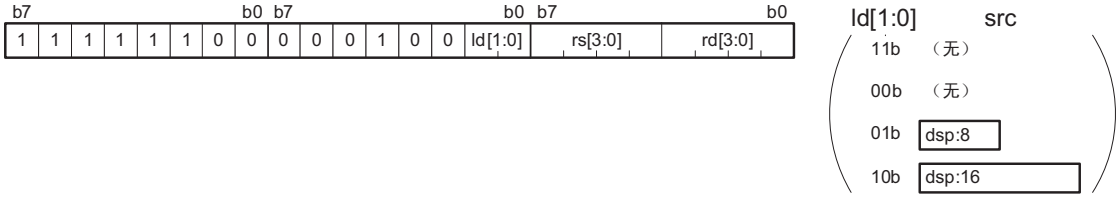


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

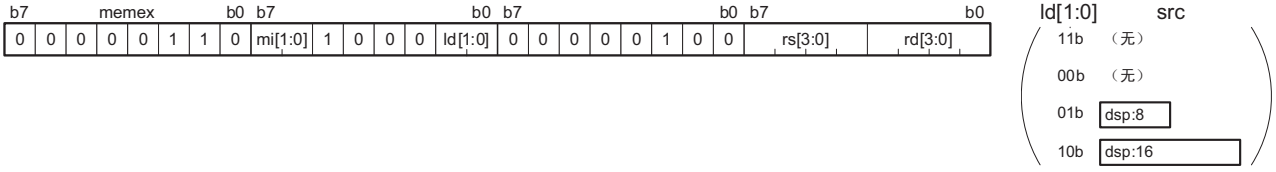
rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(2) MAX src, dest

memex==UB 或者 src==Rs 的情况



memex!=UB 的情况



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

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

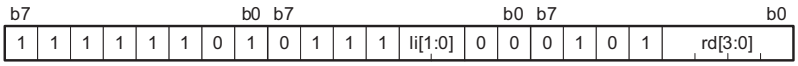
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

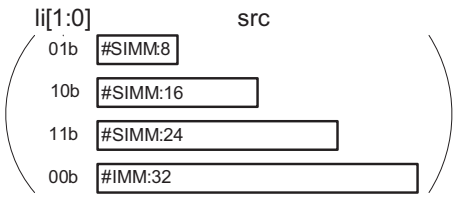
MIN MIN

【代码长度】

语法	src	dest	代码长度 （字节）
(1) MIN src, dest	#SIMM:8	Rd	4
	#SIMM:16	Rd	5
	#SIMM:24	Rd	6
	#IMM:32	Rd	7
(2) MIN src, dest	Rs	Rd	3
	[Rs].memex	Rd	3 (memex == UB) 4 (memex != UB)
	dsp:8[Rs].memex	Rd	4 (memex == UB) 5 (memex != UB)
	dsp:16[Rs].memex	Rd	5 (memex == UB) 6 (memex != UB)

(1) MIN src, dest



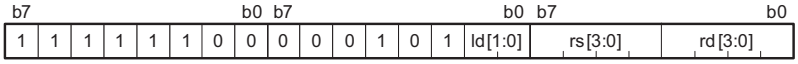


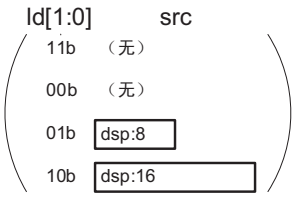
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

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

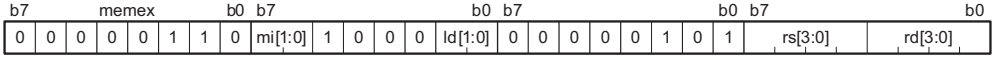
(2) MIN src, dest

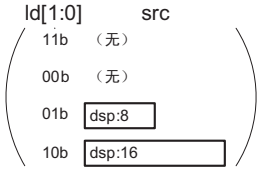
memex==UB 或者 src==Rs 的情况





memex!=UB 的情况





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

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

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

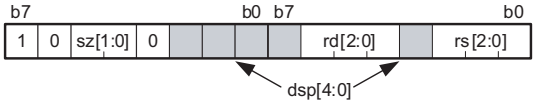
MOV MOV

【代码长度】

语法	size	处理长度	src	dest	代码长度 （字节）
(1) MOV.size src, dest	B/W/L	size	Rs (Rs=R0 ～ R7)	dsp:5[Rd] (Rd=R0 ～ R7)	2
(2) MOV.size src, dest	B/W/L	L	dsp:5[Rs] (Rs=R0 ～ R7)	Rd (Rd=R0 ～ R7)	2
(3) MOV.size src, dest	L	L	#UIMM:4	Rd	2
(4) MOV.size src, dest	B	B	#IMM:8	dsp:5[Rd] (Rd=R0 ～ R7)	3
	W/L	size	#UIMM:8	dsp:5[Rd] (Rd=R0 ～ R7)	3
(5) MOV.size src, dest	L	L	#UIMM:8	Rd	3
(6) MOV.size src, dest	L	L	#SIMM:8	Rd	3
	L	L	#SIMM:16	Rd	4
	L	L	#SIMM:24	Rd	5
	L	L	#IMM:32	Rd	6
(7) MOV.size src, dest	B/W	L	Rs	Rd	2
	L	L	Rs	Rd	2
(8) MOV.size src, dest	B	B	#IMM:8	[Rd]	3
	B	B	#IMM:8	dsp:8[Rd]	4
	B	B	#IMM:8	dsp:16[Rd]	5
	W	W	#SIMM:8	[Rd]	3
	W	W	#SIMM:8	dsp:8[Rd]	4
	W	W	#SIMM:8	dsp:16[Rd]	5
	W	W	#IMM:16	[Rd]	4
	W	W	#IMM:16	dsp:8[Rd]	5
	W	W	#IMM:16	dsp:16[Rd]	6
	L	L	#SIMM:8	[Rd]	3
	L	L	#SIMM:8	dsp:8[Rd]	4
	L	L	#SIMM:8	dsp:16[Rd]	5
	L	L	#SIMM:16	[Rd]	4
	L	L	#SIMM:16	dsp:8[Rd]	5
	L	L	#SIMM:16	dsp:16[Rd]	6
	L	L	#SIMM:24	[Rd]	5
	L	L	#SIMM:24	dsp:8[Rd]	6
	L	L	#SIMM:24	dsp:16[Rd]	7
	L	L	#IMM:32	[Rd]	6
	L	L	#IMM:32	dsp:8[Rd]	7
	L	L	#IMM:32	dsp:16[Rd]	8
(9) MOV.size src, dest	B/W/L	L	[Rs]	Rd	2
	B/W/L	L	dsp:8[Rs]	Rd	3
	B/W/L	L	dsp:16[Rs]	Rd	4
(10) MOV.size src, dest	B/W/L	L	[Ri, Rb]	Rd	3

语法	size	处理长度	src	dest	代码长度 （字节）
(11) MOV.size src, dest	B/W/L	size	Rs	[Rd]	2
	B/W/L	size	Rs	dsp:8[Rd]	3
	B/W/L	size	Rs	dsp:16[Rd]	4
(12) MOV.size src, dest	B/W/L	size	Rs	[Ri, Rb]	3
(13) MOV.size src, dest	B/W/L	size	[Rs]	[Rd]	2
	B/W/L	size	[Rs]	dsp:8[Rd]	3
	B/W/L	size	[Rs]	dsp:16[Rd]	4
	B/W/L	size	dsp:8[Rs]	[Rd]	3
	B/W/L	size	dsp:8[Rs]	dsp:8[Rd]	4
	B/W/L	size	dsp:8[Rs]	dsp:16[Rd]	5
	B/W/L	size	dsp:16[Rs]	[Rd]	4
	B/W/L	size	dsp:16[Rs]	dsp:8[Rd]	5
	B/W/L	size	dsp:16[Rs]	dsp:16[Rd]	6
(14) MOV.size src, dest	B/W/L	size	Rs	[Rd+]	3
	B/W/L	size	Rs	[–Rd]	3
(15) MOV.size src, dest	B/W/L	L	[Rs+]	Rd	3
	B/W/L	L	[–Rs]	Rd	3

(1) MOV.size src, dest

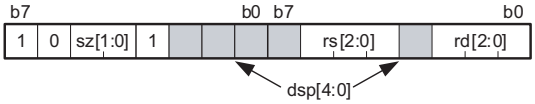


sz[1:0]	size
00b	B
01b	W
10b	L

dsp[4:0]	dsp:5
00000b ～ 11111b	0 ～ 31

rs[2:0]/rd[2:0]	src/dest	
000b ～ 111b	Rs/Rd	R0(SP) ～ R7

(2) MOV.size src, dest

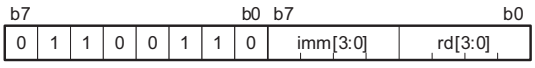


sz[1:0]	size
00b	B
01b	W
10b	L

dsp[4:0]	dsp:5
00000b ～ 11111b	0 ～ 31

rs[2:0]/rd[2:0]	src/dest	
000b ～ 111b	Rs/Rd	R0(SP) ～ R7

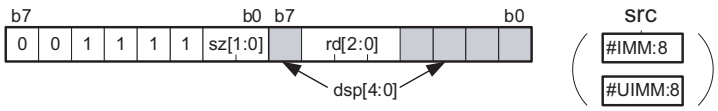
(3) MOV.size src, dest



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

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

(4) MOV.size src, dest

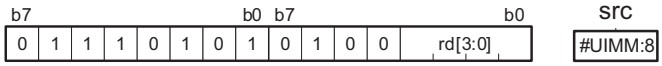


sz[1:0]	size
00b	B
01b	W
10b	L

dsp[4:0]	dsp:5
00000b ～ 11111b	0 ～ 31

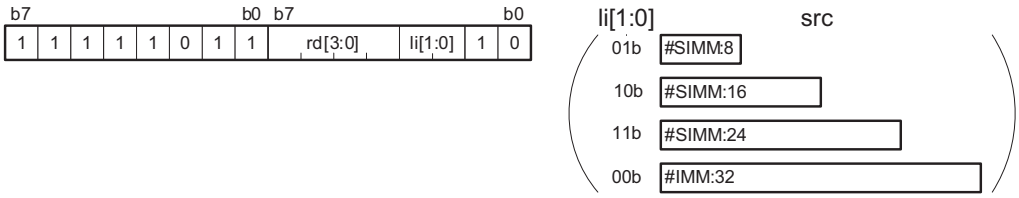
rd[2:0]	dest	
000b ～ 111b	Rd	R0(SP) ～ R7

(5) MOV.size src, dest



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

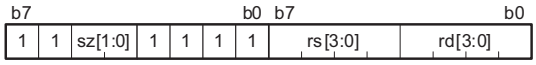
(6) MOV.size src, dest



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

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

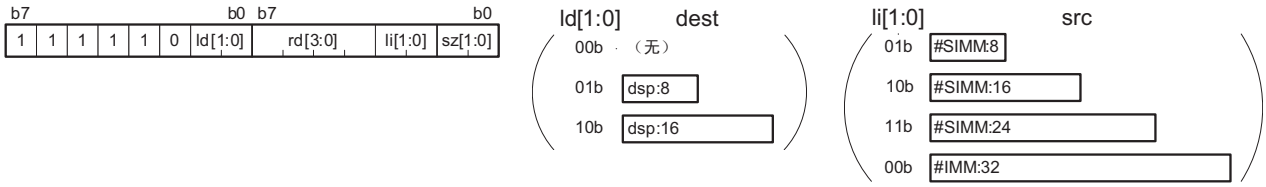
(7) MOV.size src, dest



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

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

(8) MOV.size src, dest



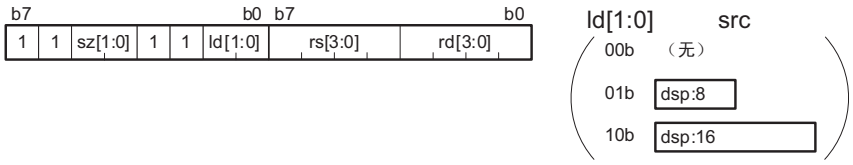
ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

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

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

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

(9) MOV.size src, dest

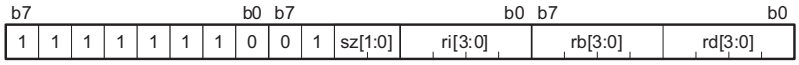


sz[1:0]	size
00b	B
01b	W
10b	L

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

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

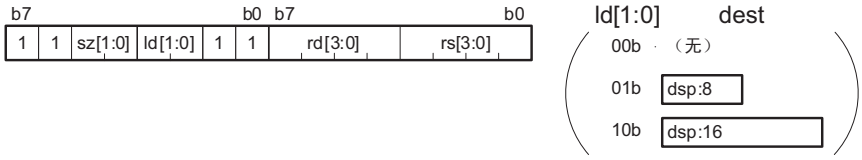
(10) MOV.size src, dest



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

ri[3:0]/rb[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Ri/Rb/Rd	R0(SP) ～ R15

(11) MOV.size src, dest

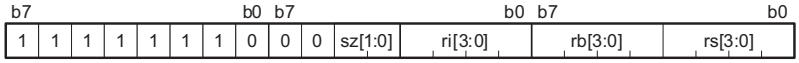


sz[1:0]	size
00b	B
01b	W
10b	L

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

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

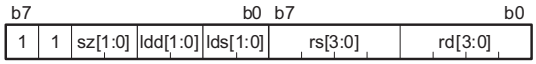
(12) MOV.size src, dest

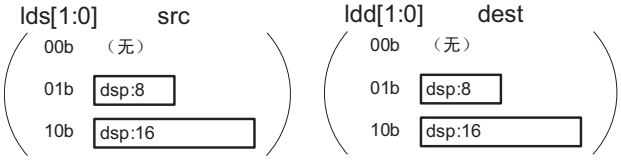


sz[1:0]	size
00b	B
01b	W
10b	L

rs[3:0]/ri[3:0]/rb[3:0]	src/dest	
0000b ～ 1111b	Rs/Ri/Rb	R0(SP) ～ R15

(13) MOV.size src, dest



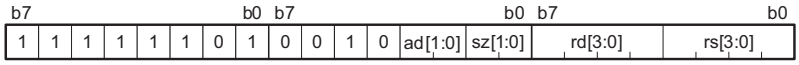


sz[1:0]	size
00b	B
01b	W
10b	L

lds[1:0]/ldd[1:0]	src/dest
00b	[Rs]/[Rd]
01b	dsp:8[Rs]/dsp:8[Rd]
10b	dsp:16[Rs]/dsp:16[Rd]

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

(14) MOV.size src, dest

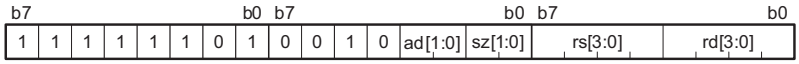


ad[1:0]	addressing
00b	Rs, [Rd+]
01b	Rs, [–Rd]

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

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

(15) MOV.size src, dest



ad[1:0]	addressing
10b	[Rs+], Rd
11b	[–Rs], Rd

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

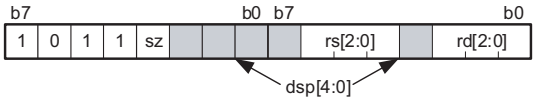
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

MOVU MOVU

【代码长度】

语法	size	处理长度	src	dest	代码长度 （字节）
(1) MOVU.size src, dest	B/W	L	dsp:5[Rs] (Rs=R0 ～ R7)	Rd (Rd=R0 ～ R7)	2
(2) MOVU.size src, dest	B/W	L	Rs	Rd	2
	B/W	L	[Rs]	Rd	2
	B/W	L	dsp:8[Rs]	Rd	3
	B/W	L	dsp:16[Rs]	Rd	4
(3) MOVU.size src, dest	B/W	L	[Ri, Rb]	Rd	3
(4) MOVU.size src, dest	B/W	L	[Rs+]	Rd	3
	B/W	L	[–Rs]	Rd	3

(1) MOVU.size src, dest

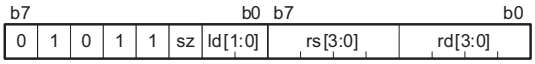


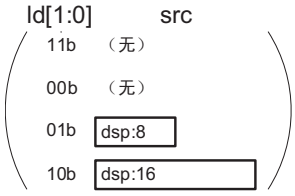
sz	size
0b	B
1b	W

dsp[4:0]	dsp:5
00000b ～ 11111b	0 ～ 31

rs[2:0]/rd[2:0]	src/dest	
000b ～ 111b	Rs/Rd	R0(SP) ～ R7

(2) MOVU.size src, dest



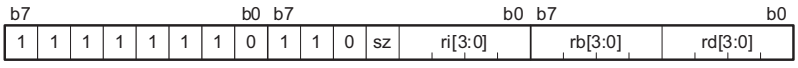


sz	size
0b	B
1b	W

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

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

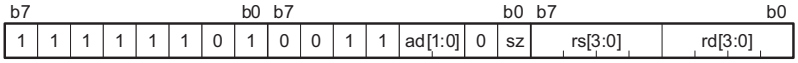
(3) MOVU.size src, dest



sz	size
0b	B
1b	W

ri[3:0]/rb[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Ri/Rb/Rd	R0(SP) ～ R15

(4) MOVU.size src, dest



ad[1:0]	addressing
10b	[Rs+], Rd
11b	[–Rs], Rd

sz	size
0b	B
1b	W

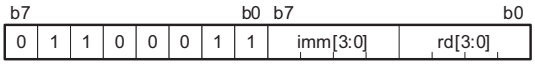
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

MUL MUL

【代码长度】

语法	src	src2	dest	代码长度 （字节）
(1) MUL src, dest	#UIMM:4	—	Rd	2
(2) MUL src, dest	#SIMM:8	—	Rd	3
	#SIMM:16	—	Rd	4
	#SIMM:24	—	Rd	5
	#IMM:32	—	Rd	6
(3) MUL src, dest	Rs	—	Rd	2
	[Rs].memex	—	Rd	2 (memex == UB) 3 (memex != UB)
	dsp:8[Rs].memex	—	Rd	3 (memex == UB) 4 (memex != UB)
	dsp:16[Rs].memex	—	Rd	4 (memex == UB) 5 (memex != UB)
(4) MUL src, src2, dest	Rs	Rs2	Rd	3

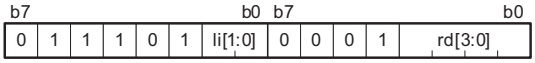
(1) MUL src, dest

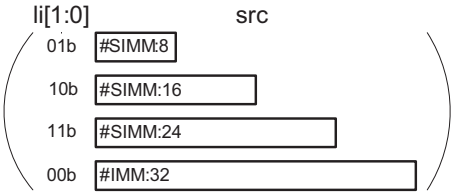


imm[3:0]	src	
0000b ～ 1111b	#UIMM:4	0 ～ 15

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

(2) MUL src, dest



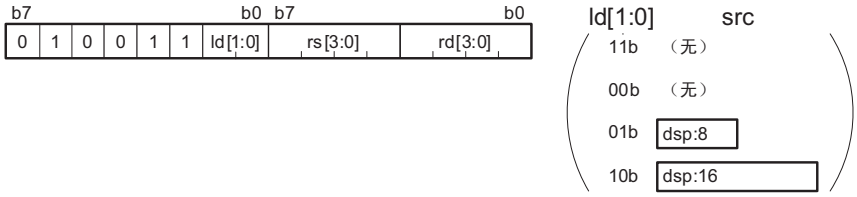


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

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

(3) MUL src, dest

memex==UB 或者 src==Rs 的情况



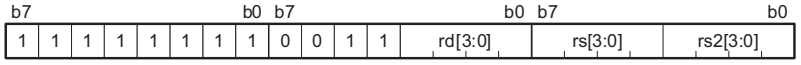
memex!=UB 的情况

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

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

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

(4) MUL src, src2, dest



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

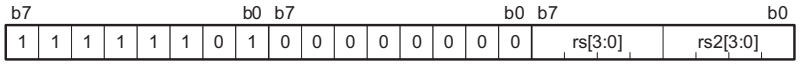
MULHI

MULHI

【代码长度】

语法	src	src2	代码长度 （字节）
(1) MULHI src, src2	Rs	Rs2	3

(1) MULHI src, src2



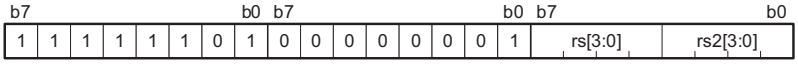
rs[3:0]/rs2[3:0]	src/src2	
0000b ～ 1111b	Rs/Rs2	R0(SP) ～ R15

MULLO MULLO

【代码长度】

语法	src	src2	代码长度 （字节）
(1) MULLO src, src2	Rs	Rs2	3

(1) MULLO src, src2



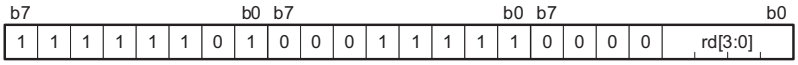
rs[3:0]/rs2[3:0]	src/src2	
0000b ～ 1111b	Rs/Rs2	R0(SP) ～ R15

MVFACHI MVFACHI

【代码长度】

语法	dest	代码长度 （字节）
(1) MVFACHI dest	Rd	3

(1) MVFACHI dest



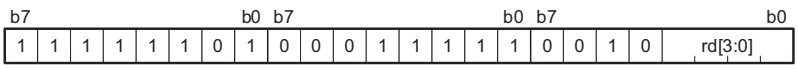
rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

MVFACMI MVFACMI

【代码长度】

语法	dest	代码长度 （字节）
(1) MVFACMI dest	Rd	3

(1) MVFACMI dest



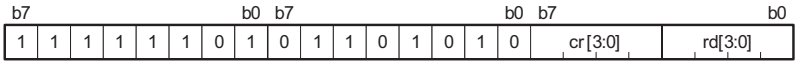
rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

MVFC MVFC

【代码长度】

语法	src	dest	代码长度 （字节）
(1) MVFC src, dest	Rx	Rd	3

(1) MVFC src, dest



cr[3:0]	src	
0000b	Rx	PSW
0001b		PC
0010b		USP
0011b		FPSW
0100b		（保留）
0101b		（保留）
0110b		（保留）
0111b		（保留）
1000b		BPSW
1001b		BPC
1010b		ISP
1011b		FINTV
1100b		INTB
1101b ～ 1111b		（保留）

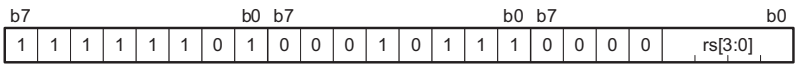
rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

MVTACHI MVTACHI

【代码长度】

语法	src	代码长度 （字节）
(1) MVTACHI src	Rs	3

(1) MVTACHI src



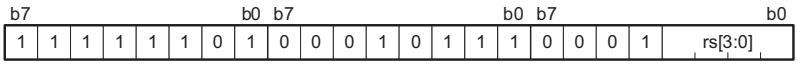
rs[3:0]	src	
0000b ～ 1111b	Rs	R0(SP) ～ R15

MVTACLO MVTACLO

【代码长度】

语法	src	代码长度 （字节）
(1) MVTACLO src	Rs	3

(1) MVTACLO src



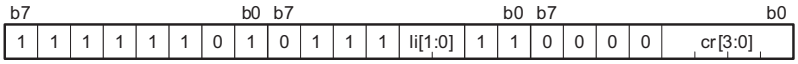
rs[3:0]	src	
0000b ～ 1111b	Rs	R0(SP) ～ R15

MVTC MVTC

【代码长度】

语法	src	dest	代码长度 （字节）
(1) MVTC src, dest	#SIMM:8	Rx	4
	#SIMM:16	Rx	5
	#SIMM:24	Rx	6
	#IMM:32	Rx	7
(2) MVTC src, dest	Rs	Rx	3

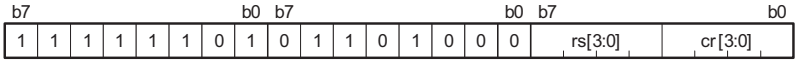
(1) MVTC src, dest



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

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

(2) MVTC src, dest



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

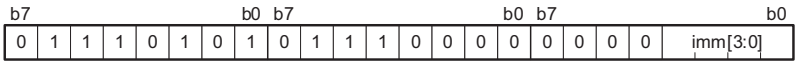
rs[3:0]	src	
0000b ～ 1111b	Rs	R0(SP) ～ R15

MVTIPL MVTIPL

【代码长度】

语法	src	代码长度 （字节）
(1) MVTIPL src	#IMM:4	3

(1) MVTIPL src



imm[3:0]	#IMM:4
0000b ～ 1111b	0 ～ 15

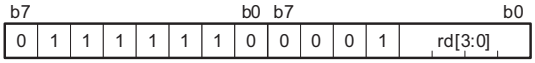
注 . 在 RX610 群中，不能使用 MVTIPL 指令。要写处理器处理字 （PSW）的处理器中断优先级 （IPL[2:0]）时，必须使用 MVTC 指令。

NEG NEG

【代码长度】

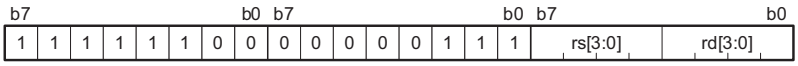
语法	src	dest	代码长度 （字节）
(1) NEG dest	—	Rd	2
(2) NEG src, dest	Rs	Rd	3

(1) NEG dest



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

(2) NEG src, dest



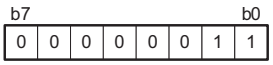
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

NOP NOP

【代码长度】

语法	代码长度 （字节）
(1) NOP	1

(1) NOP

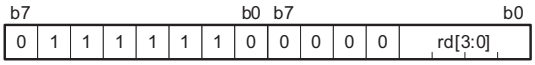


NOT NOT

【代码长度】

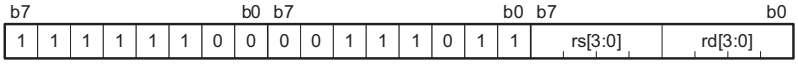
语法	src	dest	代码长度 （字节）
(1) NOT dest	—	Rd	2
(2) NOT src, dest	Rs	Rd	3

(1) NOT dest



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

(2) NOT src, dest



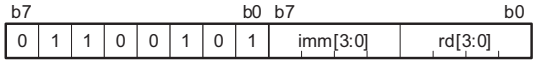
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

OR OR

【代码长度】

语法	src	src2	dest	代码长度 （字节）
(1) OR src, dest	#UIMM:4	—	Rd	2
(2) OR src, dest	#SIMM:8	—	Rd	3
	#SIMM:16	—	Rd	4
	#SIMM:24	—	Rd	5
	#IMM:32	—	Rd	6
(3) OR src, dest	Rs	—	Rd	2
	[Rs].memex	—	Rd	2 (memex == UB) 3 (memex != UB)
	dsp:8[Rs].memex	—	Rd	3 (memex == UB) 4 (memex != UB)
	dsp:16[Rs].memex	—	Rd	4 (memex == UB) 5 (memex != UB)
(4) OR src, src2, dest	Rs	Rs2	Rd	3

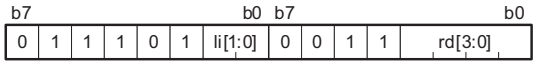
(1) OR src, dest



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

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

(2) OR src, dest

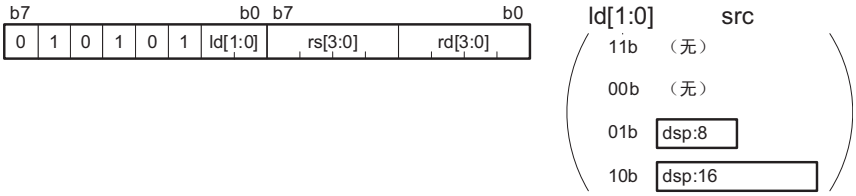


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

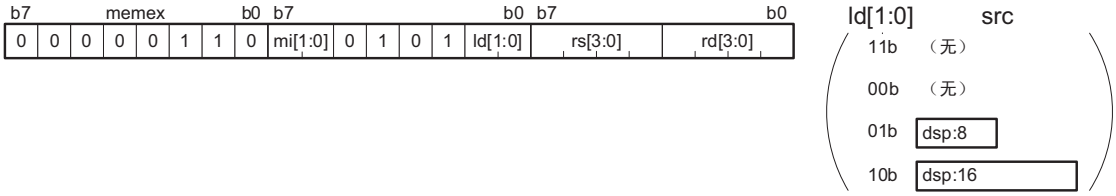
rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(3) OR src, dest

memex==UB 或者 src==Rs 的情况



memex !=UB 的情况

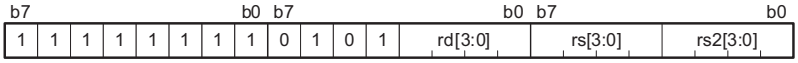


mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

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

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

(4) OR src, src2, dest



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

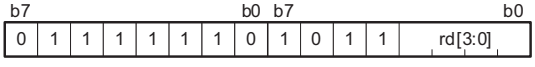
POP

POP

【代码长度】

语法	dest	代码长度 （字节）
(1) POP dest	Rd	2

(1) POP dest



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

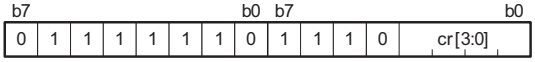
POPC

POPC

【代码长度】

语法	dest	代码长度 （字节）
(1) POPC dest	Rx	2

(1) POPC dest



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

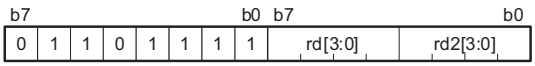
POPM

POPM

【代码长度】

语法	dest	dest2	代码长度 （字节）
(1) POPM dest-dest2	Rd	Rd2	2

(1) POPM dest-dest2



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

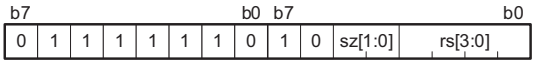
rd2[3:0]	dest2	
0010b ～ 1111b	Rd2	R2 ～ R15

PUSH PUSH

【代码长度】

语法	src	代码长度 （字节）
(1) PUSH.size src	Rs	2
(2) PUSH.size src	[Rs]	2
	dsp:8[Rs]	3
	dsp:16[Rs]	4

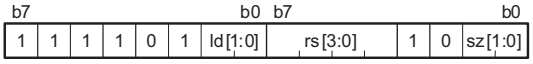
(1) PUSH.size src

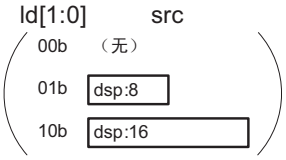


sz[1:0]	size
00b	B
01b	W
10b	L

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

(2) PUSH.size src





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

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

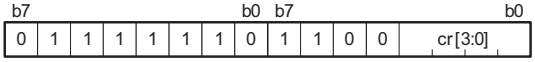
sz[1:0]	size
00b	B
01b	W
10b	L

PUSHC PUSHC

【代码长度】

语法	src	代码长度 （字节）
(1) PUSHC src	Rx	2

(1) PUSHC src



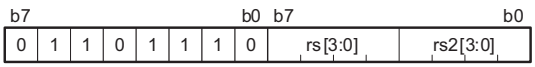
cr[3:0]	src	
0000b	Rx	PSW
0001b		PC
0010b		USP
0011b		FPSW
0100b		（保留）
0101b		（保留）
0110b		（保留）
0111b		（保留）
1000b		BPSW
1001b		BPC
1010b		ISP
1011b		FINTV
1100b		INTB
1101b ～ 1111b		（保留）

PUSHM PUSHM

【代码长度】

语法	src	src2	代码长度 （字节）
(1) PUSHM src-src2	Rs	Rs2	2

(1) PUSHM src-src2



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

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

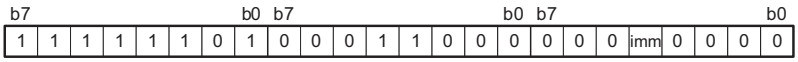
RACW

RACW

【代码长度】

语法	src	代码长度 （字节）
(1) RACW src	#IMM:1	3

(1) RACW src



imm	src	
0b ～ 1b	#IMM:1	1 ～ 2

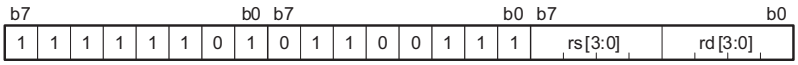
REVL

REVL

【代码长度】

语法	src	dest	代码长度 （字节）
(1) REVL src, dest	Rs	Rd	3

(1) REVL src, dest



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

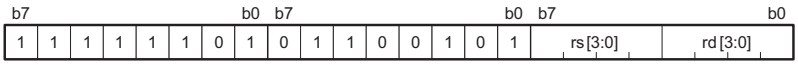
REVW

REVW

【代码长度】

语法	src	dest	代码长度 （字节）
(1) REVW src, dest	Rs	Rd	3

(1) REVW src, dest



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

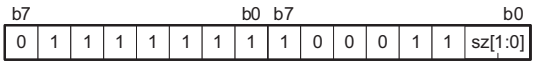
RMPA

RMPA

【代码长度】

语法	size	代码长度 （字节）
(1) RMPA.size	B	2
	W	2
	L	2

(1) RMPA.size



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

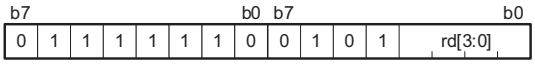
ROLC

ROLC

【代码长度】

语法	dest	代码长度 （字节）
(1) ROLC dest	Rd	2

(1) ROLC dest



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

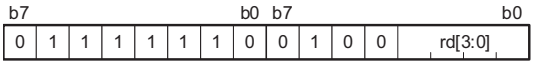
RORC

RORC

【代码长度】

语法	dest	代码长度 （字节）
(1) RORC dest	Rd	2

(1) RORC dest



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

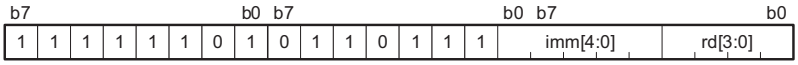
ROTL

ROTL

【代码长度】

语法	src	dest	代码长度 （字节）
(1) ROTL src, dest	#IMM:5	Rd	3
(2) ROTL src, dest	Rs	Rd	3

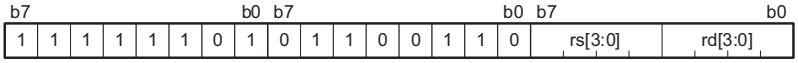
(1) ROTL src, dest



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

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

(2) ROTL src, dest



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

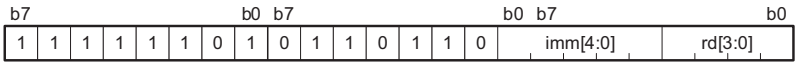
ROTR

ROTR

【代码长度】

语法	src	dest	代码长度 （字节）
(1) ROTR src, dest	#IMM:5	Rd	3
(2) ROTR src, dest	Rs	Rd	3

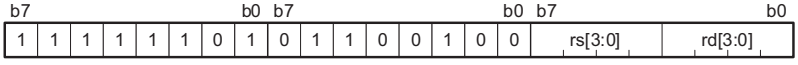
(1) ROTR src, dest



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

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

(2) ROTR src, dest



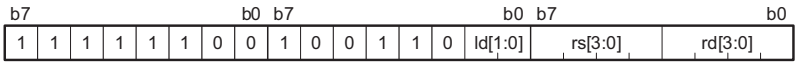
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

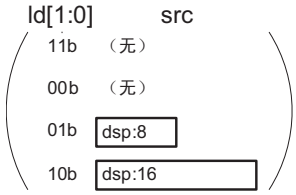
ROUND ROUND

【代码长度】

语法	src	dest	代码长度 （字节）
(1) ROUND src, dest	Rs	Rd	3
	[Rs].L	Rd	3
	dsp:8[Rs].L	Rd	4
	dsp:16[Rs].L	Rd	5

(1) ROUND src, dest





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

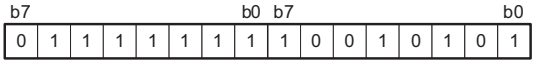
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

RTE RTE

【代码长度】

语法	代码长度 （字节）
(1) RTE	2

(1) RTE

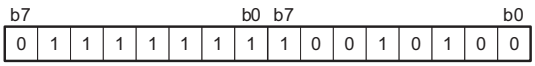


RTFI RTFI

【代码长度】

语法	代码长度 （字节）
(1) RTFI	2

(1) RTFI

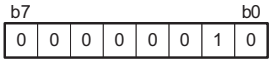


RTS RTS

【代码长度】

语法	代码长度 （字节）
(1) RTS	1

(1) RTS

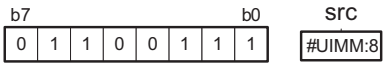


RTSD RTSD

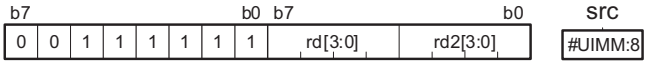
【代码长度】

语法	src	dest	dest2	代码长度 （字节）
(1) RTSD src	#UIMM:8	—	—	2
(2) RTSD src, dest-dest2	#UIMM:8	Rd	Rd2	3

(1) RTSD src



(2) RTSD src, dest-dest2



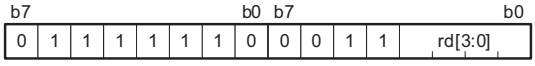
rd[3:0]/rd2[3:0]	dest/dest2	
0001b ～ 1111b	Rd/Rd2	R1 ～ R15

SAT SAT

【代码长度】

语法	dest	代码长度 （字节）
(1) SAT dest	Rd	2

(1) SAT dest



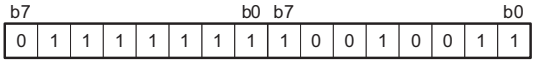
rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

SATR SATR

【代码长度】

语法	代码长度 （字节）
(1) SATR	2

(1) SATR

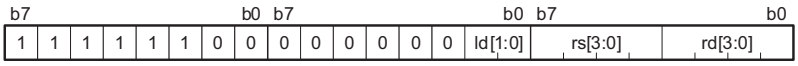


SBB SBB

【代码长度】

语法	src	dest	代码长度 （字节）
(1) SBB src, dest	Rs	Rd	3
(2) SBB src, dest	[Rs].L	Rd	4
	dsp:8[Rs].L	Rd	5
	dsp:16[Rs].L	Rd	6

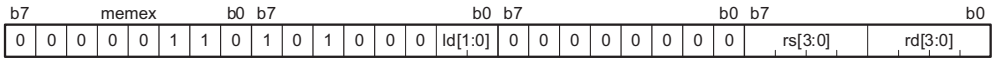
(1) SBB src, dest

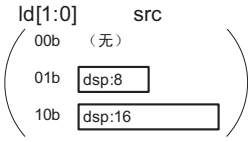


ld[1:0]	src
11b	Rs

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

(2) SBB src, dest





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

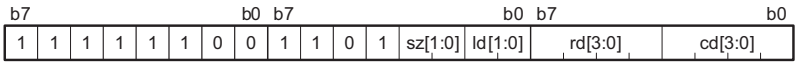
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

SCCnd SCCnd

【代码长度】

语法	size	dest	代码长度 （字节）
(1) SC*Cnd*.size dest	L	Rd	3
	B/W/L	[Rd]	3
	B/W/L	dsp:8[Rd]	4
	B/W/L	dsp:16[Rd]	5

(1) SC*Cnd*.size dest



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

ld[1:0]	dest
11b	Rd
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

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

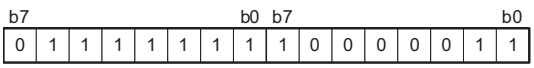
cd[3:0]	SC*Cnd*	cd[3:0]	SC*Cnd*
0000b	SCEQ, SCZ	1000b	SCGE
0001b	SCNE, SCNZ	1001b	SCLT
0010b	SCGEU, SCC	1010b	SCGT
0011b	SCLTU, SCNC	1011b	SCLE
0100b	SCGTU	1100b	SCO
0101b	SCLEU	1101b	SCNO
0110b	SCPZ	1110b	(保留)
0111b	SCN	1111b	(保留)

SCMPU SCMPU

【代码长度】

语法	代码长度 （字节）
(1) SCMPU	2

(1) SCMPU

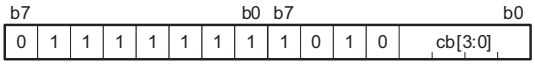


SETPSW SETPSW

【代码长度】

语法	dest	代码长度 （字节）
(1) SETPSW dest	flag	2

(1) SETPSW dest



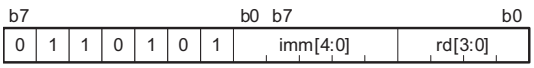
cb[3:0]	dest	
0000b	flag	C
0001b		Z
0010b		S
0011b		O
0100b		（保留）
0101b		（保留）
0110b		（保留）
0111b		（保留）
1000b		I
1001b		U
1010b		（保留）
1011b		（保留）
1100b		（保留）
1101b		（保留）
1110b		（保留）
1111b		（保留）

SHAR SHAR

【代码长度】

语法	src	src2	dest	代码长度 （字节）
(1) SHAR src, dest	#IMM:5	—	Rd	2
(2) SHAR src, dest	Rs	—	Rd	3
(3) SHAR src, src2, dest	#IMM:5	Rs	Rd	3

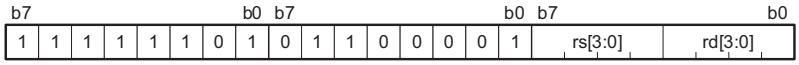
(1) SHAR src, dest



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

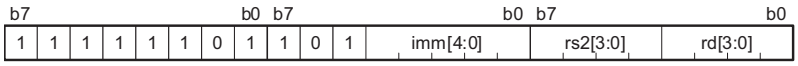
rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(2) SHAR src, dest



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

(3) SHAR src, src2, dest



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

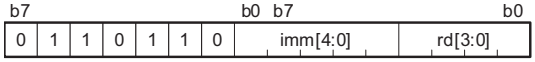
rs2[3:0]/rd[3:0]	src2/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

SHLL SHLL

【代码长度】

语法	src	src2	dest	代码长度（字节）
(1) SHLL src, dest	#IMM:5	—	Rd	2
(2) SHLL src, dest	Rs	—	Rd	3
(3) SHLL src, src2, dest	#IMM:5	Rs	Rd	3

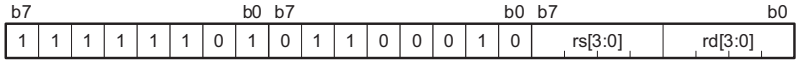
(1) SHLL src, dest



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

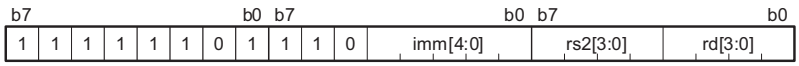
rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(2) SHLL src, dest



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

(3) SHLL src, src2, dest



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

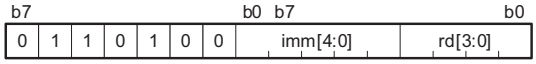
rs2[3:0]/rd[3:0]	src2/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

SHLR SHLR

【代码长度】

语法	src	src2	dest	代码长度 （字节）
(1) SHLR src, dest	#IMM:5	—	Rd	2
(2) SHLR src, dest	Rs	—	Rd	3
(3) SHLR src, src2, dest	#IMM:5	Rs	Rd	3

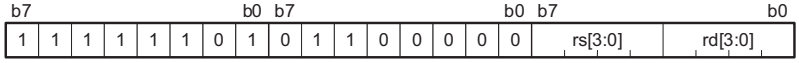
(1) SHLR src, dest



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

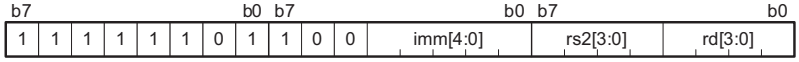
rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(2) SHLR src, dest



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

(3) SHLR src, src2, dest



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

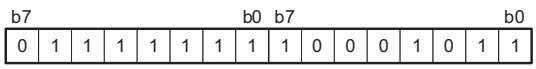
rs2[3:0]/rd[3:0]	src2/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

SMOVB SMOVB

【代码长度】

语法	代码长度 （字节）
(1) SMOVB	2

(1) SMOVB

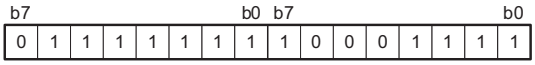


SMOVF SMOVF

【代码长度】

语法	代码长度 （字节）
(1) SMOVF	2

(1) SMOVF

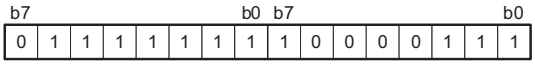


SMOVU SMOVU

【代码长度】

语法	代码长度 （字节）
(1) SMOVU	2

(1) SMOVU

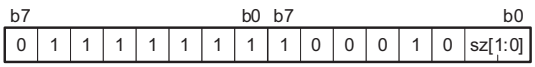


SSTR SSTR

【代码长度】

语法	size	处理长度	代码长度 （字节）
(1) SSTR.size	B	B	2
	W	W	2
	L	L	2

(1) SSTR.size



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

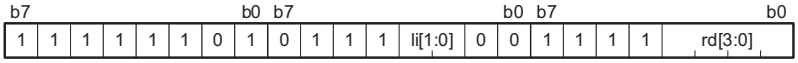
STNZ

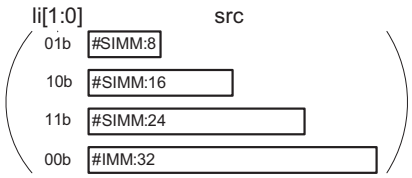
STNZ

【代码长度】

语法	src	dest	代码长度 （字节）
(1) STNZ src, dest	#SIMM:8	Rd	4
	#SIMM:16	Rd	5
	#SIMM:24	Rd	6
	#IMM:32	Rd	7

(1) STNZ src, dest





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

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

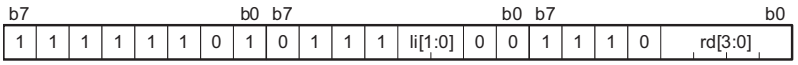
STZ

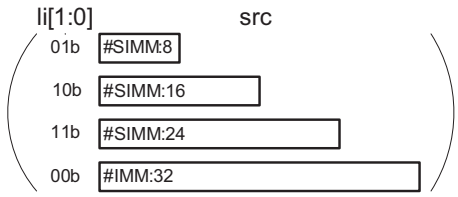
STZ

【代码长度】

语法	src	dest	代码长度 （字节）
(1) STZ src, dest	#SIMM:8	Rd	4
	#SIMM:16	Rd	5
	#SIMM:24	Rd	6
	#IMM:32	Rd	7

(1) STZ src, dest





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

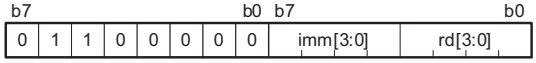
rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

SUB SUB

【代码长度】

语法	src	src2	dest	代码长度 （字节）
(1) SUB src, dest	#UIMM:4	—	Rd	2
(2) SUB src, dest	Rs	—	Rd	2
	[Rs].memex	—	Rd	2 (memex == UB) 3 (memex != UB)
	dsp:8[Rs].memex	—	Rd	3 (memex == UB) 4 (memex != UB)
	dsp:16[Rs].memex	—	Rd	4 (memex == UB) 5 (memex != UB)
(3) SUB src, src2, dest	Rs	Rs2	Rd	3

(1) SUB src, dest

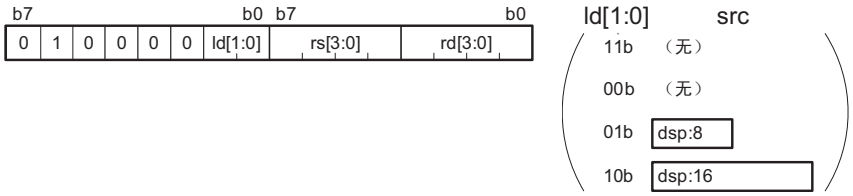


imm[3:0]	src	
0000b ～ 1111b	#UIMM:4	0 ～ 15

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

(2) SUB src, dest

memex==UB 或者 src==Rs 的情况



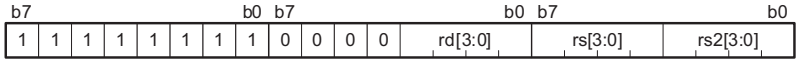
memex!=UB 的情况

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

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

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

(3) SUB src, src2, dest



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

SUNTIL SUNTIL

【代码长度】

语法	size	处理长度	代码长度 （字节）
(1) SUNTIL.size	B	B	2
	W	W	2
	L	L	2

(1) SUNTIL.size

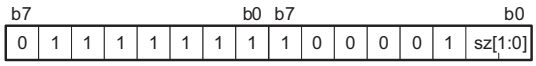
sz[1:0]	size
00b	B
01b	W
10b	L

SWHILE SWHILE

【代码长度】

语法	size	处理长度	代码长度 （字节）
(1) SWHILE.size	B	B	2
	W	W	2
	L	L	2

(1) SWHILE.size



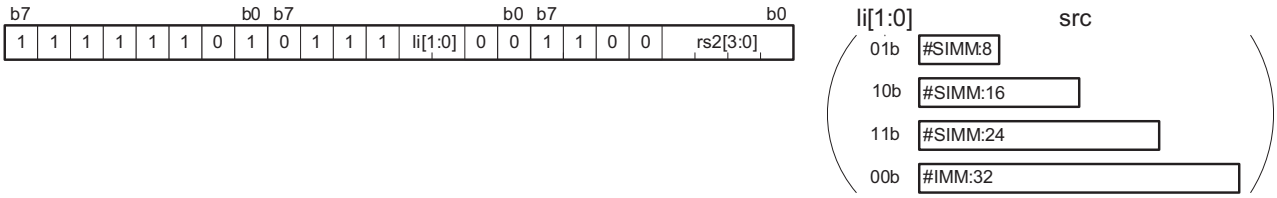
sz[1:0]	size
00b	B
01b	W
10b	L

TST TST

【代码长度】

语法	src	src2	代码长度 （字节）
(1) TST src, src2	#SIMM:8	Rs	4
	#SIMM:16	Rs	5
	#SIMM:24	Rs	6
	#IMM:32	Rs	7
(2) TST src, src2	Rs	Rs2	3
	[Rs].memex	Rs2	3 (memex == UB) 4 (memex != UB)
	dsp:8[Rs].memex	Rs2	4 (memex == UB) 5 (memex != UB)
	dsp:16[Rs].memex	Rs2	5 (memex == UB) 6 (memex != UB)

(1) TST src, src2

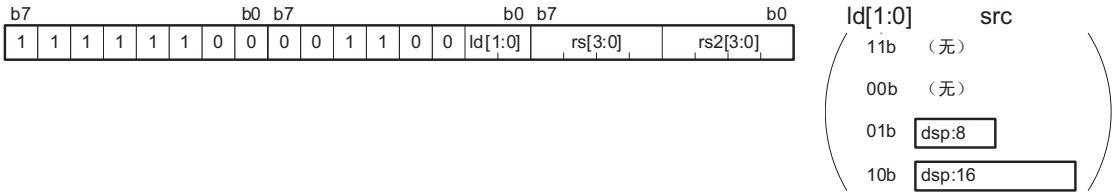


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

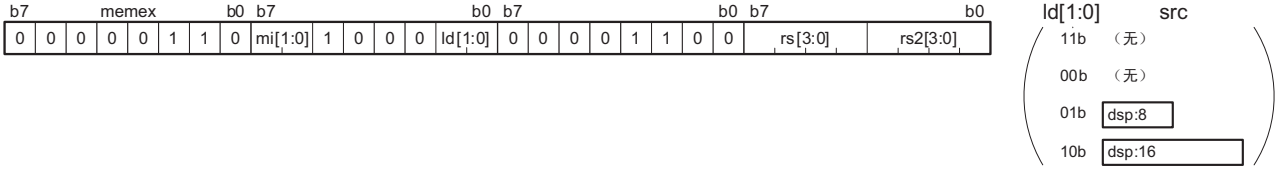
rs2[3:0]	src2	
0000b ～ 1111b	Rs	R0(SP) ～ R15

(2) TST src, src2

memex==UB 或者 src==Rs 的情况



memex!=UB 的情况



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

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

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

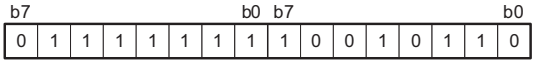
WAIT

WAIT

【代码长度】

语法	代码长度 （字节）
(1) WAIT	2

(1) WAIT



XCHG

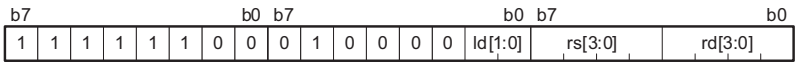
XCHG

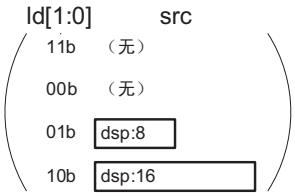
【代码长度】

语法	src	dest	代码长度 （字节）
(1) XCHG src,dest	Rs	Rd	3
	[Rs].memex	Rd	3 (memex == UB) 4 (memex != UB)
	dsp:8[Rs].memex	Rd	4 (memex == UB) 5 (memex != UB)
	dsp:16[Rs].memex	Rd	5 (memex == UB) 6 (memex != UB)

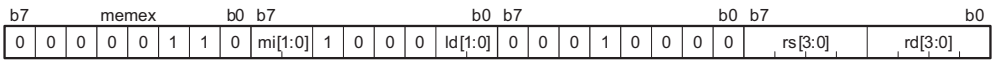
(1) XCHG src, dest

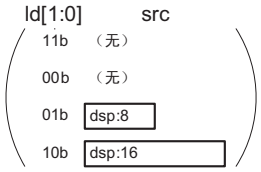
memex==UB 或者 src==Rs 的情况





memex!=UB 的情况





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

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

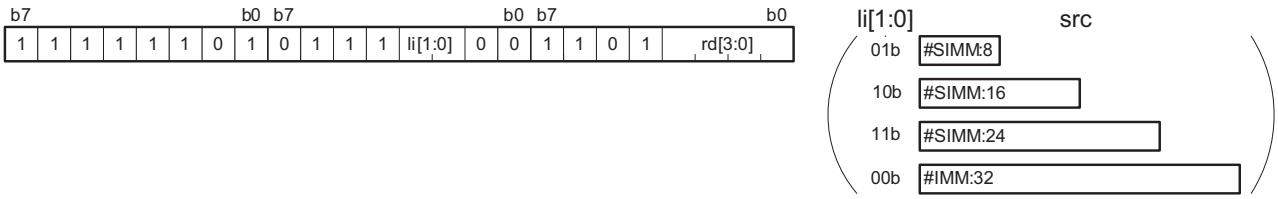
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

XOR XOR

【代码长度】

语法	src	dest	代码长度 （字节）
(1) XOR src, dest	#SIMM:8	Rd	4
	#SIMM:16	Rd	5
	#SIMM:24	Rd	6
	#IMM:32	Rd	7
(2) XOR src, dest	Rs	Rd	3
	[Rs].memex	Rd	3 (memex == UB) 4 (memex != UB)
	dsp:8[Rs].memex	Rd	4 (memex == UB) 5 (memex != UB)
	dsp:16[Rs].memex	Rd	5 (memex == UB) 6 (memex != UB)

(1) XOR src, dest

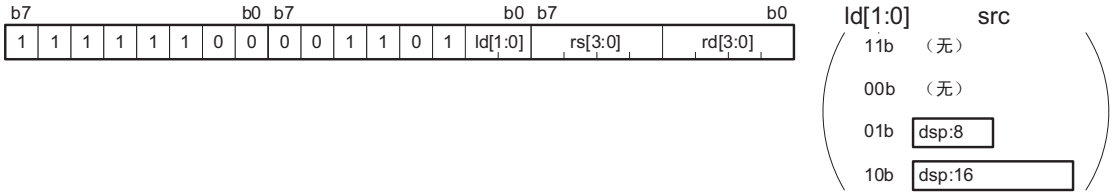


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

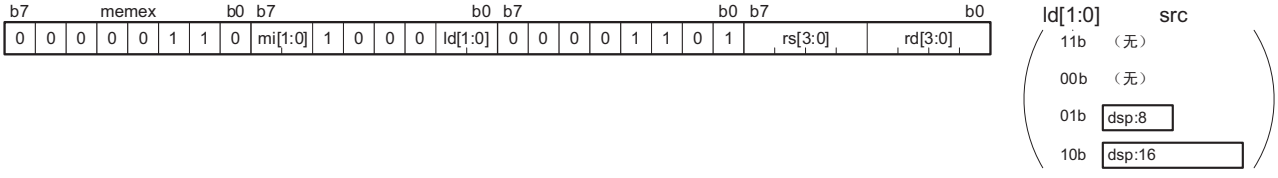
rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(2) XOR src, dest

memex==UB 或者 src==Rs 的情况



memex!=UB 的情况



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

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

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

5. 异常处理

5.1 异常事件

在 CPU 执行一般程序的过程中，有可能因某个事件的发生而中止正在执行的程序并且需要执行其他程序。此类事件统称为异常事件。

RX CPU 对应 8 种异常，异常事件的种类如图 5.1 所示。

如果发生异常，处理器模式就转移到管理模式。

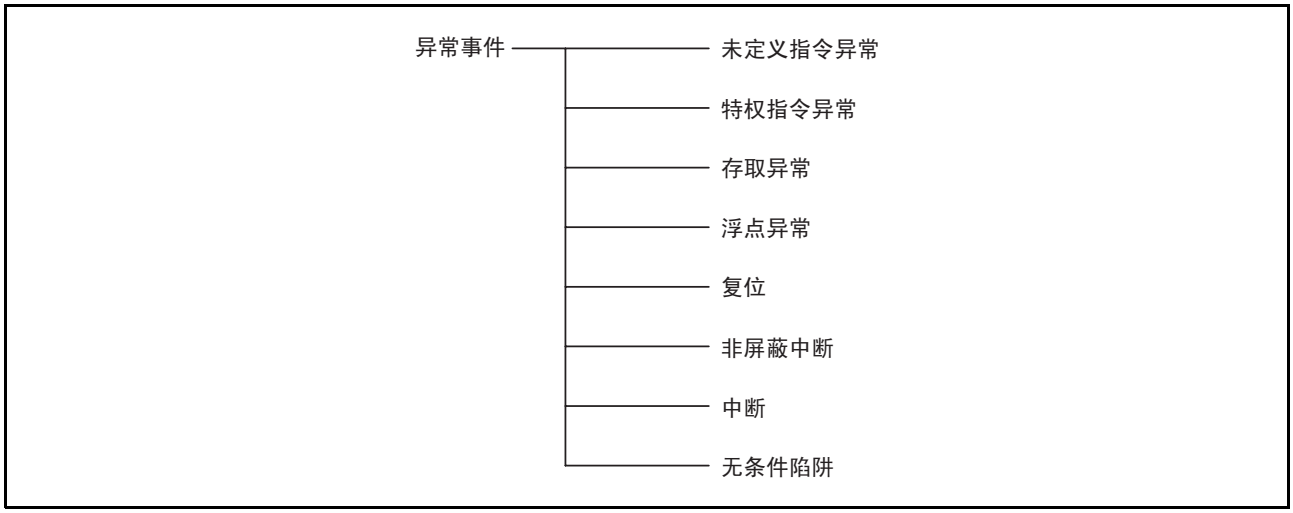


图 5.1 异常事件的种类

5.1.1 未定义指令异常

在检测到执行未定义指令 （未安装的指令）时发生未定义指令异常。

5.1.2 特权指令异常

在用户模式中检测到执行特权指令时发生特权指令异常。只能在管理模式中执行特权指令。

5.1.3 存取异常

在检测到因 CPU 存取存储器引起的错误时发生存取异常。在检测到因存储器保护单元引起的存储器保护错误时发生指令存取异常和操作数存取异常。

5.1.4 浮点异常

在检测到 IEEE754 规格规定的 5 种异常事件 （上溢、下溢、精度异常、被零除、无效运算）以及非安装处理时发生浮点异常。当 FPSW 的 EX 位、 EU 位、 EZ 位、 EO 位和 EV 位为 “0”时，禁止浮点异常处理。

5.1.5 复位

在给 CPU 输入复位信号时产生复位。因为复位的优先级最高，所以随时被接受。

5.1.6 非屏蔽中断

在给 CPU 输入非屏蔽中断信号时发生非屏蔽中断。只在认为是对系统造成致命的故障时使用此中断。使用条件是必须在异常处理程序的处理后不返回到发生异常时正在执行的程序。

5.1.7 中断

在给 CPU 输入中断信号时发生中断。能将中断中的 1 个中断源分配为高速中断，高速中断的硬件预处理和硬件后处理比一般中断快，高速中断的优先级为 15 （最高）（注）。

当 PSW 的 I 位为 “0”时，禁止接受中断。

注 . RX610 群的高速中断的优先级为 7 （最高）。

5.1.8 无条件陷阱

如果执行 INT 指令和 BRK 指令，就产生无条件陷阱。

5.2 异常处理步骤

异常处理包括硬件自动处理的部分以及通过用户记述的程序 （异常处理程序）进行处理的部分。除复位以外，接受异常时的处理步骤如图 5.2 所示。

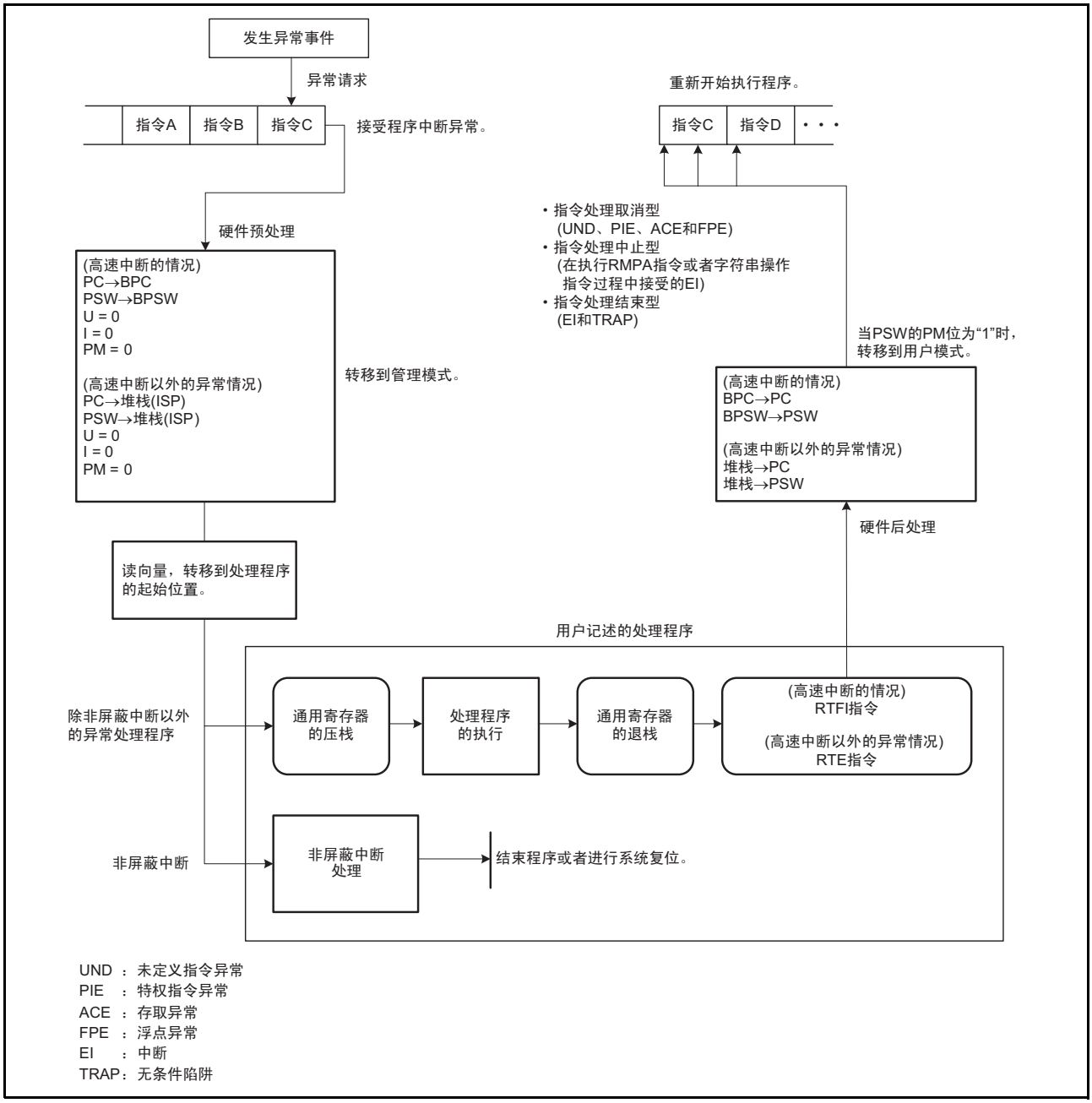


图 5.2 异常处理步骤的概要

一旦接受异常， RX CPU 就在硬件处理后，存取向量并且取得转移目标地址。按各异常给向量分配了向量地址，写异常处理程序的转移目标地址。

有关 RX CPU 的硬件预处理，在高速中断的情况下，将程序计数器 （PC）的内容保存到备用 PC （BPC），处理器状态字 （PSW）的内容保存到备用 PSW （BPSW）；在非高速中断异常的情况下，将 PC 和 PSW 保存到堆栈区。对于异常处理程序中使用的通用寄存器以及 PC 和 PSW 以外的控制寄存器，必须在异常处理程序的起始位置，通过用户程序将这些寄存器压栈。

在异常处理程序处理结束后，通过在恢复被压栈的寄存器后执行 RTE 指令，从异常处理返回到原来的程序。只在高速中断的情况下执行 RTFI 指令。但是，在非屏蔽中断的情况下，不返回到原来的程序而必须结束程序或者进行系统复位。

有关 RX CPU 的硬件后处理，在高速中断的情况下，将 BPC 的值恢复到 PC， BPSW 的值恢复到 PSW。在非高速中断异常的情况下，从堆栈区恢复 PC 和 PSW 的值。

5.3 异常事件的接受

如果发生异常事件，就在中止目前执行的程序后转移到异常处理程序的处理。

5.3.1 接受时序和被保存的 PC 值

各异常事件的接受时序以及被保存的程序计数器 （PC）的值如表 5.1 所示。

表 5.1 接受时序和被保存的 PC 值

异常事件		处理型	接受时序	被保存到 BPC/ 堆栈的 PC 值
未定义指令异常		指令处理取消型	正在执行指令	发生异常的指令的 PC 值
特权指令异常		指令处理取消型	正在执行指令	发生异常的指令的 PC 值
存取异常		指令处理取消型	正在执行指令	发生异常的指令的 PC 值
浮点异常		指令处理取消型	正在执行指令	发生异常的指令的 PC 值
复位		指令处理放弃型	各机器周期	无
非屏蔽中断	正在执行 RMPA、 SCMPU、 SMOVB、 SMOVF、 SMOVU、 SSTR、 SUNTIL、 SWHILE 指令	指令处理中止型	正在执行指令	正在执行的指令的 PC 值
	上述以外的状态	指令处理结束型	在指令和指令之间	下一条指令的 PC 值
中断	正在执行 RMPA、 SCMPU、 SMOVB、 SMOVF、 SMOVU、 SSTR、 SUNTIL、 SWHILE 指令	指令处理中止型	正在执行指令	正在执行的指令的 PC 值
	上述以外的状态	指令处理结束型	在指令和指令之间	下一条指令的 PC 值
无条件陷阱		指令处理结束型	在指令和指令之间	下一条指令的 PC 值

5.3.2 向量和 PC、 PSW 的保存场所

各异常事件的向量、程序计数器 （PC）和处理器状态字 （PSW）的保存场所如表 5.2 所示。

表 5.2 向量和 PC、 PSW 的保存场所

异常事件		向量	PC 和 PSW 的保存场所
未定义指令异常		固定向量表	堆栈
特权指令异常		固定向量表	堆栈
存取异常		固定向量表	堆栈
浮点异常		固定向量表	堆栈
复位		固定向量表	无
非屏蔽中断		固定向量表	堆栈
中断	高速中断	FINTV	BPC、 BPSW
	非高速中断	可变向量表 （INTB）	堆栈
无条件陷阱		可变向量表 （INTB）	堆栈

5.4 接受异常 / 从异常返回时的硬件处理

以下说明接受异常以及从异常返回时的硬件处理 （复位除外）。

(1) 接受异常时的硬件预处理

(a) PSW 的保存

（高速中断的情况）
PSW→BPSW
（非高速中断的异常情况）
PSW→ 堆栈区

注 . 在硬件预处理中不保存 FPSW。如果在异常处理程序内使用浮点运算指令，用户就必须在异常处理程序内将 FPSW 压栈。

(b) PSW 的 PM 位、 U 位和 I 位的更新

I ：置 “0”
U ：置 “0”
PM ：置 “0”

(c) PC 的保存

（高速中断的情况）
PC→BPC
（非高速中断的异常情况）
PC→ 堆栈区

(d) 给 PC 设定异常处理程序的转移目标地址

通过取得对应各异常的向量，转移到异常处理程序的处理。

(2) 执行 RTE 指令和 RTFI 指令时的硬件后处理

(a) PSW 的恢复

（高速中断的情况）
BPSW→PSW
（非高速中断的异常情况）
堆栈区 →PSW

(b) PC 的恢复

（高速中断的情况）
BPC→PC
（非高速中断的异常情况）
堆栈区 →PC

5.5 硬件预处理

以下说明从接受异常请求到执行异常处理程序的硬件预处理。

5.5.1 未定义指令异常

1. 将处理器状态字 （PSW）的内容保存到堆栈区 （ISP）。
2. 将PSW的处理器模式设定位 （PM）、堆栈指针指定位 （U）和中断允许位 （I）置 “0”。
3. 将程序计数器 （PC）的内容保存到堆栈区 （ISP）。
4. 从地址FFFFFFDCh取向量。
5. 将取到的向量设定到PC后转移到异常处理程序。

5.5.2 特权指令异常

1. 将处理器状态字 （PSW）的内容保存到堆栈区 （ISP）。
2. 将PSW的处理器模式设定位 （PM）、堆栈指针指定位 （U）和中断允许位 （I）置 “0”。
3. 将程序计数器 （PC）的内容保存到堆栈区 （ISP）。
4. 从地址FFFFFFD0h取向量。
5. 将取到的向量设定到PC后转移到异常处理程序。

5.5.3 存取异常

1. 将处理器状态字 （PSW）的内容保存到堆栈区 （ISP）。
2. 将PSW的处理器模式设定位 （PM）、堆栈指针指定位 （U）和中断允许位 （I）置 “0”。
3. 将程序计数器 （PC）的内容保存到堆栈区 （ISP）。
4. 从地址FFFFFFD4h取向量。
5. 将取到的向量设定到PC后转移到异常处理程序。

5.5.4 浮点异常

1. 将处理器状态字 （PSW）的内容保存到堆栈区 （ISP）。
2. 将PSW的处理器模式设定位 （PM）、堆栈指针指定位 （U）和中断允许位 （I）置 “0”。
3. 将程序计数器 （PC）的内容保存到堆栈区 （ISP）。
4. 从地址FFFFFFE4h取向量。
5. 将取到的向量设定到PC后转移到异常处理程序。

5.5.5 复位

1. 对控制寄存器进行初始化。
2. 从地址FFFFFFFCh取向量。
3. 将取到的向量设定到程序计数器 （PC）。

5.5.6 非屏蔽中断

1. 将处理器状态字 （PSW）的内容保存到堆栈区 （ISP）。
2. 将PSW的处理器模式设定位 （PM）、堆栈指针指定位 （U）和中断允许位 （I）置 “0”。
3. 当正在执行RMPA、SCMPU、SMOVB、SMOVF、SMOVU、SSTR、SUNTIL、SWHILE指令时，将正在执行的指令的程序计数器 （PC）内容保存到堆栈区 （ISP）；而在其他状态下，将下一条指令的PC内容保存到堆栈区 （ISP）。
4. 将PSW的处理器中断优先级 （IPL[3:0]）置 “Fh”。
5. 从地址FFFFFFF8h取向量。
6. 将取到的向量设定到PC后转移到异常处理程序。

5.5.7 中断

1. 将处理器状态字 （PSW）的内容保存到堆栈区 （ISP）。在高速中断的情况下，保存到备用PSW （BPSW）。
2. 将PSW的处理器模式设定位 （PM）、堆栈指针指定位 （U）和中断允许位 （I）置 “0”。
3. 当正在执行RMPA、SCMPU、SMOVB、SMOVF、SMOVU、SSTR、SUNTIL、SWHILE指令时，将正在执行的指令的程序计数器 （PC）内容保存到堆栈区 （ISP）；而在其他状态下，将下一条指令的PC内容保存到堆栈区 （ISP）。在高速中断的情况下，保存到备用PC （BPC）。
4. 给PSW的处理器中断优先级 （IPL[3:0]）设定已接受中断的中断优先级。
5. 从可变向量表取已接受中断源的向量。在高速中断的情况下，从高速中断向量寄存器 （FINTV）取向量。
6. 将取到的向量设定到PC后转移到异常处理程序。

5.5.8 无条件陷阱

1. 将处理器状态字 （PSW）的内容保存到堆栈区 （ISP）。
2. 将PSW的处理器模式设定位 （PM）、堆栈指针指定位 （U）和中断允许位 （I）置 “0”。
3. 将下一条指令的程序计数器 （PC）内容保存到堆栈区 （ISP）。
4. 在使用INT指令时，从可变向量表取对应INT指令号的向量。
   在使用BRK指令时，从可变向量表的起始地址取向量。
5. 将取到的向量设定到PC后转移到异常处理程序。

5.6 从异常处理程序的返回

如果在异常处理程序的最后执行**表 5.3** 所示的指令，就恢复异常处理顺序前保存到堆栈区或者控制寄存器（BPC 和 BPSW）的程序计数器 （PC）和处理器状态字 （PSW）的内容。

表 5.3 异常处理程序的返回指令

异常事件		返回指令
未定义指令异常		RTE
特权指令异常		RTE
存取异常		RTE
浮点异常		RTE
复位		不能返回
非屏蔽中断		不能返回
中断	高速中断	RTFI
	非高速中断	RTE
无条件陷阱		RTE

5.7 异常事件的优先级

异常事件的优先级如**表 5.4** 所示。如果同时发生多个异常，就先接受优先级高的事件。

表 5.4 中断优先级

优先级		异常事件
高 ↑ 低	1	复位
	2	非屏蔽中断
	3	中断
	4	指令存取异常
	5	未定义指令异常 特权指令异常
	6	无条件陷阱
	7	操作数存取异常
	8	浮点异常

索引

N

O

P

Q

R

S

T

U

W

X

Y

Z

修订记录	RX 族 用户手册 软件篇

Rev.	发行日	修订内容	
		页	修订处
1.00	2011.01.26	—	初版发行

RX 族
用户手册 软件篇

Publication Date: Rev.1.00 Jan 26, 2011

Published by: Renesas Electronics Corporation

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

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

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
7F, No. 363 Fu Shing North Road Taipei, Taiwan, R.O.C.
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
1 harbourFront Avenue, #06-10, keppel Bay Tower, Singapore 098632
Tel: +65-6213-0200, Fax: +65-6278-8001

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 1.0

RX族

RENESAS

瑞萨电子株式会社

R01US0028CJ0100