

Gentech E804 User's Manual

April 24, 2024

Copyright© 2023 Hangzhou C-SKY MicroSystems Co., Ltd. All rights reserved.

This document is the property of Hangzhou C-SKY MicroSystems Co., Ltd. and its affiliates ("C-SKY"). This document may only be distributed to: (i) a C-SKY party having a legitimate business need for the information contained herein, or (ii) a non-C-SKY party having a legitimate business need for the information contained herein. This document may only be distributed to: (i) a C-SKY party having a legitimate business need for the information contained herein, or (ii) a non-C-SKY party having a legitimate business need for the information contained herein. No license, expressed or implied, under any patent, copyright or trade secret right is granted or implied by the conveyance of this document. No part of this document may be reproduced, transmitted, transcribed, stored in a retrieval system, translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise without the prior written consent of the parties. manual, or otherwise without the prior written permission of Hangzhou C-SKY MicroSystems Co.

Trademarks and Permissions

The C-SKY Logo and all other trademarks indicated as such herein (including XuanTie) are trademarks of Hangzhou C-SKY MicroSystems Co., Ltd. All other products or service names are the property of their respective owners. products or service names are the property of their respective owners.

Notice

The purchased products, services and features are stipulated by the contract made between C-SKY and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Hangzhou C-SKY MicroSystems Co., LTD

Address: Room 201, 2/F, Building 5, No.699 Wangshang Road , Hangzhou, Zhejiang, China Website:
www.xrvm.cn

Copyright© 2023 Hangzhou Zhongtian Microsystems Co.

The ownership and intellectual property rights in this document are vested in Hangzhou Zhongtian Microsystems Company Limited and its affiliates ("Zhongtian Microsystems"). This document may be distributed only to (i) employees of ZT Micro who have a legitimate employment relationship and need the information in this document, or (ii) partners who are not part of the ZT Micro organization but who have a legitimate relationship with ZT Micro and need the information in this document. This document may not be used without the express consent of Hangzhou Zhongtian Microsystems Co. No part of this document may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language without the express written permission of ZT Micro.

Trademark Declaration

The ZT Micro logo and all other trademarks (e.g., XuanTie) are owned by Hangzhou ZT Micro Systems Co., Ltd. and its affiliates, and no legal entity is permitted to use any of ZT Micro's trademarks or commercial logos without the written consent of Hangzhou ZT Micro Systems Co.

take note of

Your purchase of products, services or features, etc. shall be subject to the commercial contracts and terms and conditions of Zenith Micro, and all or part of the products, services or features described in this document may be excluded from your purchase or use. Unless otherwise agreed in the contract, ZTE Micro makes no representations or warranties, express or implied, with respect to the contents of this document.

The contents of this document may be updated from time to time due to product version upgrades or other reasons. Unless otherwise agreed, this document is intended only as a guide to use, all statements, information and recommendations in this document does not constitute any express or implied warranty. Hangzhou Zhongtian Microsystems Co., Ltd. shall not be liable to any third party for any loss arising from the use of this document.

Hangzhou C-SKY MicroSystems Co., LTD

Address: Room 201, 2/F, Building 5, No. 699, Netshang

Road, Hangzhou, Zhejiang, China Website: www.xrvm.cn

Edition本 History

block printing	descriptive	dates
01	First official release.	2021.07.31
02	Update the template.	2024.04.24

Gentei E804 User's Manual

Chapt er 1	summarize	1
1.1	Introduction	1
1.2	Features	1
1.3	Configurable options	2
1.4	Design for Testability	3
1.5	Design for Debuggability	3
1.6	Naming rules	3
1.6.1	Symbols	3
1.6.2	Terminology	3
Chapt er II	microarchitecture	6
2.1	Block Diagrams	6
2.2	Introduction to the assembly line	7
2.3	Floating-point processing unit	8
2.4	DSP Expansion Unit	9
2.5	Trusted Protection Technologies	9
2.6	Tightly Coupled IP Architecture	10
Chapt er III	programming model	11
3.1	Operating Mode and Register View	11

3.2	General Purpose Registers	12
3.2.1	Condition Code/Feed Flag Bits	13
3.3	System Control Register	14
3.3.1	Processor Status Register (PSR, CR<0,0>)	14
3.3.2	Update PSR	16
3.3.3	Vector Base Address Register (VBR, CR<1,0>)	16
3.3.4	Exception Reserved Registers (CR<2,0>, CR<4,0>)	17
3.3.5	Product serial number register (CPUIDR, CR<13,0>)	17
3.3.6	Implicit Operation Register (CHR, CR<31,0>)	17
3.3.7	Other control registers	18
3.3.8	Normal User Mode General Purpose Register 14 (R14(User SP), CR<14,1>)	18
3.3.9	Interrupt Pointer Register (R14(Int_SP),CR<15,1>)	18
3.4	Data size end	19
3.5	Data Alignment Access	19
3.6	System address mapping	19
3.7	Memory access order	20
chap	Exception handling	22
4.1	Overview of Exception Handling	22
4.2	Exception type	24
4.2.1	Restart Exception (vector offset 0X0)	24
4.2.2	Unaligned access exception (vector offset 0X4)	24
4.2.3	Access error exception (vector offset 0X8)	25
4.2.4	Divide by zero exception (vector offset 0X0C)	25

4.2.5	Illegal instruction exception (vector offset 0X10)	25
4.2.6	Privilege violation exception (vector offset 0X14)	25
4.2.7	Trace Exception (Vector Offset 0X18)	25
4.2.8	Breakpoint Exception (Vector Offset 0X1C)	26
4.2.9	Unrecoverable error exception (vector offset 0X20)	26
4.2.10	IDLY Exception (Exception Offset 0X24)	26
4.2.11	Trap instruction exception (vector offset 0X40 - 0X4C)	26
4.2.12	TSPEND interrupt (vector cheap 0X58)	26
4.2.13	Floating point exception (vector offset 0X78)	26
4.3	Interrupt Exception	26
4.4	Exception Prioritization	28
4.5	Exception return	29
Chapter V	Tightly Coupled IP	30
5.1	Introduction to Tightly Coupled IP	30
5.2	System timer	31
5.2.1	Introduction	31
5.2.2	Register Definitions	31
5.2.2.1	Control and Status Register (CORET_CSR)	32
5.2.2.2	Backfill Value Register (CORET_RVR)	33
5.2.2.3	Current Value Register (CORET_CVR)	34
5.2.2.4	Calibration register (CORET_CALIB)	34
5.2.3	Procedure	35
5.3	Vector Interrupt Controller	36

5.3.1	Introduction	36
5.3.2	Register Definitions	37
5.3.2.1	Interrupt enable setting register (VIC_ISER)	39
5.3.2.2	Interrupt Low Power Wakeup Setting Register (VIC_IWER)	40
5.3.2.3	Interrupt enable clear register (VIC_ICER)	40
5.3.2.4	Interrupt Low Power Wake-up Clear Register (VIC_IWDR)	41
5.3.2.5	Interrupt Wait Setup Register (VIC_ISPR)	41
5.3.2.6	Interrupt Wait Clear Register (VIC_ICPR)	42
5.3.2.7	Interrupt Response Status Register (VIC_IABR)	42
5.3.2.8	Interrupt Priority Setting Register (VIC_IPR0 - VIC_IPR31)	43
5.3.2.9	Interrupt Status Register (VIC_ISR)	45
5.3.2.10	Interrupt Priority Threshold Register (VIC_IPTR)	45
5.3.2.11	Tspend Interrupt Enable Setting Register (VIC_TSPEND)	46
5.3.2.12	Tspend Interrupt Response Status Register (VIC_TSABR)	47
5.3.2.13	Tspend Interrupt Priority Setting Register (VIC_TSPR)	47
5.3.3	Interrupt Handling Mechanism	48
5.3.3.1	Interrupt status bits	48
5.3.3.2	Interrupt Priority	49
5.3.3.3	Interrupt vector number	49
5.3.3.4	Interrupt Handling Process	50
5.3.3.5	Interrupt Nesting	51
5.3.4	Tspend interrupt	54
5.3.5	Procedure	54
5.3.6	Interface signals	54
5.3.7	Interrupt Setting Example	57
Chapt	er VI	59
6.1	Overview	59

6.2	32-bit instructions	59
6.2.1	32-Bit Instruction Function Classification	59
6.2.1.1	Data Arithmetic Instructions	59
6.2.1.2	Branch jump instruction	63
6.2.1.3	Memory access instructions	64
6.2.1.4	Privileged commands	65
6.2.1.5	Special function commands	66
6.2.1.6	DSP Extended Instructions	66
6.3	16-bit instructions	72
6.3.1	16-Bit Instruction Functional Classification	72
6.3.1.1	Data Arithmetic Instructions	72
6.3.1.2	Branch Jump Instruction	74
6.3.1.3	Memory access instructions	74
6.3.1.4	Multi-register access instructions	75
6.4	Instruction Set List	75
6.5	Instruction execution delay	90
Chapt er VII		100
7.1	Introduction to the Memory Protection Unit	100
7.2	Associated system control registers	100
7.2.1	Cache Configuration Register (CCR, CR<18,0>)	100
7.2.2	Configuration Register for Highly Easeable Access Rights (CAPR, CR<19,0>)	101
7.2.3	Protected Area Control Register (PACR, CR<20,0>)	102

7.2.4	Protected Region Selection Register (PRSR, CR<21,0>)	103
7.3	Memory Access Handling	104
Chapte	On-chip Cache	105
r VIII		
8.1	Introduction to Caching	105
8.2	Associated system control registers	105
8.2.1	Cache Enable Register (CER)	106
8.2.2	Cache Invalidation Register (CIR)	107
8.2.3	Highable Buffer Configuration Register 0~3 (CRCR)	107
8.2.4	Cache Performance Analysis Control Register (CPFCR)	109
8.2.5	Cache Access Count Register (CPFATR)	109
8.2.6	Cache Miss Count Register (CPFMTR)	109
Chapte	Bus Matrix and Bus Interface	111
r 9		
9.1	Introduction	111
9.2	System bus interface	113
9.2.1	Features	113
9.2.2	Protocol content	113
9.2.2.1	Supported transfer types	113
9.2.2.2	Supported response types	113
9.2.3	Behavior with different bus responses	114
9.2.4	Interface signals for the AHB protocol	114
9.2.5	Interface signals for the AHB-Lite protocol	116
9.3	Command Bus Interface	117
9.3.1	Features	117

9.3.2	Protocol content	117
9.3.2.1	Supported transfer types	117
9.3.2.2	Supported response types	117
9.3.3	Behavior with different bus responses	118
9.3.4	Command Bus Interface Signals	119
9.4	Data bus interface	120
9.4.1	Features	120
9.4.2	Protocol content	120
9.4.2.1	Supported transfer types	120
9.4.2.2	Supported response types	121
9.4.3	Behavior with different bus responses	121
9.4.4	Data bus interface signals	122
9.5	Order of access to instructions and data	123
Chapte	debugging interface	124
r 10		
10.1	Overview	124
10.2	External interfaces	125
Chapter	Working modes and conversions	127
XI		
11.1	Normal operating mode	127
11.2	Low Power Mode	128
11.3	Debug Mode	128
11.3.1	Debug Mode	128
11.3.2	Entering Debug Mode	128
11.3.3	Exiting Debug Mode	129

Chapter	Floating Point Processing Unit	130
XII		
12.1	Overview	130
12.1.1	Introduction	130
12.1.2	Features	130
12.2	Microarchitecture	131
12.2.1	Introduction to Floating Point Units	131
12.3	Programming Models	131
12.3.1	Introduction	131
12.3.2	Data format	131
12.3.2.1	Orthopedic data format	131
12.3.2.2	Single-precision data format	132
12.3.3	Floating-Point Register Description	132
12.3.3.1	Transferring Data with General Purpose Registers	132
12.3.3.2	Maintaining Consistent Register Accuracy	132
12.3.4	System Registers	132
12.3.4.1	Methods for reading and writing floating-point system registers	134
12.3.4.2	Floating Point Version Register (FPU ID Register, FID)	134
12.3.4.3	Floating Point Control Register (FPU Control Register, FCR)	135
12.3.4.4	Floating Point Exception Status Register (FESR)	137
12.3.5	Data Size End	138
12.4	Exception Handling	138
12.4.1	Introduction to Floating-Point Exceptions	139
12.4.2	Unspecified Number Input Exception	139

12.4.2.1 Abnormal Enablement	139
12.4.2.2 Abnormal disablement	140
12.4.3 Illegal Operation Exception	140
12.4.3.1 Abnormal Enablement	140
12.4.3.2 Abnormal disablement	141
12.4.4 Dividing by Zero Exception	141
12.4.4.1 Abnormal Enablement	141
12.4.4.2 Abnormal disablement	141
12.4.5 Overflow Exception	141
12.4.5.1 Exception enable	141
12.4.5.2 Abnormal disablement	142
12.4.6 Underflow Exception	142
12.4.6.1 Exception enable	142
12.4.6.2 Abnormal disablement	142
12.4.7 Imprecise exceptions	142
12.4.7.1 Exception enable	143
12.4.7.2 Abnormal disablement	143
12.4.8 Exception Prioritization	143
12.4.9 Exceptions for Special Instructions	143
12.5 Instruction set	143
12.5.1 List of Floating-Point Instructions	144
12.5.1.1 Data Arithmetic Instructions	144

12.5.1.2 Transfer class instructions	145
12.5.1.3 Memory access instructions	145
12.5.2 Floating-Point Instruction Execution Delay	145
Chapter Initialization Reference Code	147
XIII	
13.1 MPU Setup Example	147
13.2 Cache Setting Example	149
13.3 Interrupt Enable Initialization	150
13.4 General Purpose Register Initialization Example	150
13.5 Stack Pointer Initialization Example	151
13.6 Example of Exception and Interrupt Service Program Entry Address Setting	151
13.7 FPU Initialization Example	152
Chapter Appendix A Base Instruction Glossary	154
XIV	
14.1 ABS - Absolute Value Instruction	154
14.2 ADDC - Unsigned with Progressive Addition Instruction	155
14.3 ADDI - Unsigned Immediate Number Addition Instruction	156
14.4 ADDI(SP)-Unsigned (Stack Pointer) Immediate Number Addition Instruction	160
14.5 ADDU - unsigned addition instruction	162
14.6 AND - By Bit with Instructions	164
14.7 ANDI - Immediate Number By Bit with Instruction	165
14.8 ANDN - per-bit non-associate instruction	166
14.9 ANDNI - Immediate Number by Bit Non-Issue Instruction	167
14.10 ASR - arithmetic right shift instruction	168

14.11 ASRC - Immediate Number Arithmetic Right Shift to C-bit Instruction	170
14.12 ASRI - Arithmetic Right Shift Instruction for Immediate Numbers	172
14.13 BCLRI - Immediate Digit Clear Instruction	173
14.14 BEZ - Register Equal to Zero Branch Instruction	175
14.15 BF--C is a 0 branch command	176

14.16	BGENI - Immediate Digit Generation Instruction.....	178
14.17	BGENR - Register Bit Generation Instruction.....	179
14.18	BHSZ - Register Greater Than or Equal to Zero Branching Instruction.....	180
14.19	BHZ - Register Greater Than Zero Branch Instruction	181
14.20	BKPT - Breakpoint Instruction.....	182
14.21	BLSZ - Register Less Than or Equal to Zero Branching Instructions.....	183
14.22	BLZ - Register Less Than Zero Branch Instruction.....	185
14.23	BMASKI - Immediate Digit Mask Generation Instruction.....	186
14.24	BNEZ - Register Not Equal to Zero Branch Instruction.....	188
14.25	BNEZAD - Register Self-Decrement Greater Than Zero Branch Instruction	189
14.26	BR - Unconditional Jump Instructions	191
14.27	BREV - Bit Reverse Order Instruction.....	192
14.28	BSETI - Immediate Number Position Bit Instruction.....	194
14.29	BSR - Jump to Subroutine Instruction.....	195
14.30	BT--C is a 1 branch command.	198.
14.31	BTSTI - Immediate Digit Test Instruction.....	199
14.32	CLRF--C is 0 Clear instruction	200
14.33	CLRT--C is a 1 clear command.....	201
14.34	CMPHS - Unsigned Greater Than or Equal to Compare Instruction.....	202
14.35	CMPHSI - Immediate Number Unsigned Greater Than or Equal Compare Instruction.....	204
14.36	CMPLT - Signed Less Than Compare Instruction.....	207
14.37	CMPLTI - Immediate Number Signed Less Than Compare Instruction	209
14.38	CMPNE - unequal comparison instruction	212
14.39	CMPNEI - Immediate Number of Unequal Comparisons Instruction	214
14.40	DECF--C is a 0 immediate number subtraction instruction.....	216
14.41	DEC GT - Subtract Greater Than Zero Set C Bit Instruction.....	217
14.42	DECLT - Subtract Less Than Zero Set C Bit Instruction.....	218
14.43	DECNE - Subtract Not Equal to Zero Set C Bit Instruction.....	219
14.44	DECT--C is a 1-immediate subtraction instruction.....	220
14.45	DIVS - Signed Division Instruction.....	221
14.46	DIVU - Unsigned Division Instruction	222
14.47	DOZE - Enter Low Power Sleep Mode Instruction.....	223
14.48	FF0 - Fast Find 0 command.....	224
14.49	FF1 - Quick Find 1 command	225
14.50	GRS - symbol generation instructions	226
14.51	IDLY - Interrupt Recognition Disable Instruction	227
14.52	INCF - C is 0 Immediate Number Addition Instruction.....	229
14.53	INCT--C is a 1-immediate number addition instruction.....	230
14.54	INS - Bit Insertion Instruction	231
14.55	IPOP - Interrupt Out of Stack Instruction	233
14.56	IPUSH - Interrupt Stacking Instruction	234
14.57	IXH - Indexed Half-Word Instruction.....	235

14.58 IXW - Index Word Command.....	236
14.59 IXD - Indexed Double Word Directive.....	236
14.60 JMP - Register Jump Instructions.....	237
14.61 JMPI - Indirect Jump Instruction.....	239
14.62 JSR - Register Jump to Subroutine Instruction.....	240
14.63 JSRI - Indirect Jump to Subroutine Instruction	242
14.64 LD.B - Unsigned Extended Byte Load Instruction.....	244
14.65 LD.BS - Signed Extended Byte Load Instruction	247
14.66 LD.H - Unsigned Extended Half-Word Load Instruction	248
14.67 LD.HS - Signed Extended Half-Word Load Instruction.....	251
14.68 LD.W - Word Load Instruction	252
14.69 LDM - Continuous Multi-Word Load Directive.....	255
14.70 LDQ - Consecutive Four-Word Load Instruction.....	257
14.71 LDR.B - Register Shift Addressing Unsigned Extended Byte Load Instruction.....	259
14.72 LDR.BS - Register Shift Addressing Signed Extended Byte Load Instruction	260
14.73 LDR.H - Register Shift Addressing Unsigned Extended Half-Word Load Instruction.....	262
14.74 LDR.HS - Register Shift Addressing Signed Extended Half-Word Load Instruction.....	264
14.75 LDR.W - Register Shift Addressing Word Load Instruction	266
14.76 LRS.B - Byte Symbol Load Instruction.....	268
14.77 LRS.H - Half Word Symbol Loading Instruction.....	270
14.78 LRS.W - Word Symbol Loading Instruction.....	271
14.79 LRW - memory read instruction.....	273
14.80 LSL - Logical Left Shift Instruction.....	275
14.81 LSLC - Immediate Logic Left Shift to C-Bit Instruction.....	276
14.82 LSLI - Immediate Logical Left Shift Instruction.....	278
14.83 LSR - Logical Right Shift Instruction.....	280
14.84 LSRC - Immediate Logic Right Shift to C-Bit Instruction	281
14.85 LSRI - Immediate Logical Right Shift Instruction	283
14.86 MFCR - Control Register Read Transfer Instructions.....	284
14.87 MOV - Data Transfer Instructions	285
14.88 MOVF--C is 0 Data Transfer Instruction	286
14.89 MOVI - Immediate Number Data Transfer Instruction	287
14.90 MOVIH - Immediate Number High Data Transfer Instruction	288
14.91 MOVT--C is a 1 data transfer instruction.....	289
14.92 MTCR - Control Register Write Transfer Instruction.....	290
14.93 MULA.32.I - 32-bit Signed Multiply-Accumulate Fetch Lower 32-bit Instruction	290
14.94 MULALL.S16.S - 16-Bit Signed Low Half-Word Multiply Accumulate Instruction with Saturation Operation 291	291
14.95 MULA.(U/S)32 - 32-bit (None/Have) Signed Multiply Accumulate Instruction.....	293
14.96 MULT - Multiplication Instructions.....	294
14.97 MUL.(U/S)32 - 32-bit (no/have) signed multiply instruction	295
14.98 MVC - C Bit Transfer Instructions.....	296

14.99	MVCV - C-bit inverse transfer	297
14.100	NIE - Interrupt Nested Enable Instruction	298
14.101	NIR - Nested Interrupt Return Instruction	300
14.102	NOR - per-bit or non-instruction	302
14.103	NOT - per-bit non-instruction	303
14.104	OR - By Bit or Command	304
14.105	ORI - Immediate Number by Bit or Instruction	305
14.106	POP - Out of Stack Instruction	307
14.107	PSRCLR - PSR Bit Clear Instruction	311
14.108	PSRSET - PSR position bit instruction	313
14.109	PUSH - Pressure Stack Instruction	315
14.110	REVB - Byte Reversal Instruction	319
14.111	REVH - half-word byte inverted instruction	320
14.112	ROTL - Recurring Left Shift Instruction	322
14.113	ROTLI - Immediate Number Cyclic Left Shift Instruction	323
14.114	RSUB - Reverse Subtraction Instruction	324
14.115	RTS - subroutine return instruction	325
14.116	RTE - Exception and Normal Interrupt Return Instructions	326
14.117	SCE - Conditional Execution Setup Instruction	327
14.118	SEXT - Bit Extraction with Signed Expansion Instruction	330
14.119	SEXTB - Byte Extraction with Signed Expansion Instruction	332
14.120	SEXTH - half-word extraction with signed extension instruction	333
14.121	SRS.B - Byte Symbol Storage Instruction	335
14.122	SRS.H - Half-Word Symbol Storage Instruction	336
14.123	SRS.W - Word Symbol Storage Instruction	338
14.124	ST.B - Byte Storage Instructions	339
14.125	ST.H - Half-Word Storage Instruction	342
14.126	ST.W - Word Storage Instruction	344
14.127	STM - Sequential Multi-Word Storage Instructions	347
14.128	STOP - Enter Low Power Suspend Mode Instruction	349
14.129	STQ - Sequential Four-Word Storage Instruction	350
14.130	STR.B - Register Shift Addressing Byte Store Instructions	352
14.131	STR.H - Register Shift Addressing Half-Word Storage Instructions	353
14.132	STR.W - Register Shift Addressing Word Store Instructions	355
14.133	SUBC - Unsigned with Borrowed Bit Subtraction Instruction	357
14.134	SUBI - Unsigned Immediate Number Subtraction Instruction	359
14.135	SUBI(SP)-Unsigned (Stack Pointer) Immediate Number Subtraction Instruction	362
14.136	SUBU - Unsigned Subtraction Instruction	363
14.137	SYNC - CPU Synchronization Instruction	365
14.138	TRAP - Operating System Trap Instruction	366
14.139	TST - Zero Test Command	367
14.140	TSTNBZ - No Byte Equals Zero Register Test Instruction	369

14.14 ² XOR - per-positional-orthogonal-or instruction	372
14.14 ³ XORI - Immediate Number of Bitwise Orders	373
14.14 ⁴ XSR - Extended right shift instruction	374
14.14 ⁵ XTRB0 - Extract Byte 0 and Unsigned Extension Instruction	376
14.14 ⁶ XTRB1 - Extract Byte 1 and Unsigned Extension Instruction	377
14.14 ⁷ XTRB2 - Extract Byte 2 and Unsigned Extension Instruction	378
14.14 ⁸ XTRB3 - Extract Byte 3 and Unsigned Extension Instruction	379
14.14 ⁹ ZEXT - Bit Extraction and Unsigned Extension Instructions	380
14.15 ⁰ ZEXTB - Byte Extraction with Unsigned Extension Instruction	382
14.15 ¹ ZEXTH - half-word extraction and unsigned extension instructions	383
Chapter XV Appendix B DSP Instruction Set Glossary	385
15.1 PADD.8 - 8-bit Parallel Addition Instructions	386
15.2 PADD.16 - 16-Bit Parallel Addition Instructions	387
15.3 PADD.(U/S)8.S - 8-bit parallel (no/have) signed addition instruction with saturation operation	388
15.4 PADD.(U/S)16.S - 16-bit Parallel (Unsaturated/Having) Signed Addition Instruction with Saturation Operation	390
15.5 ADD.(U/S)32.S - 32-bit (no/have) signed addition instruction with saturation operation	392
15.6 PSUB.8 - 8-bit Parallel Subtraction Instructions	394
15.7 PSUB.16 - 16-Bit Parallel Subtraction Instructions	395
15.8 PSUB.(U/S)8.S - 8-bit parallel (no/have) sign-subtraction instruction with saturation operation	396
15.9 PSUB.(U/S)16.S - 16-bit parallel (no/have) sign-subtraction instruction with saturation operation	398
15.10 SUB.(U/S)32.S - 32-bit (no/have) signed subtraction instructions with saturation operations	400
15.11 PADDH.(U/S)8 - 8-bit parallel (no/have) signed addition averaging instruction	402
15.12 PADDH.(U/S)16 - 16-bit Parallel (Un/Have) Signed Addition to Average Instruction	403

15.13	ADDH.(U/S)32 - 32-bit (Un/Signed) Signed Addition to Average Instruction	404
15.14	PSUBH.(U/S)8 - 8-bit parallel (no/have) signed subtraction averaging instruction	405
15.15	PSUBH.(U/S)16 - 16-bit parallel (no/have) signed subtraction averaging instruction	406
15.16	SUBH.(U/S)32 - 32-bit (no/have) signed subtraction averaging instruction	407
15.17	PASX.16 - 16-bit Parallel Cross Add and Subtract Instructions	407
15.18	PSAX.16 - 16-bit Parallel Cross Subtract and Add Instructions	408
15.19	PASX.(U/S)16.S - 16-bit parallel (no/have) sign cross-add and subtract instructions with saturation operations	409
15.20	PSAX.(U/S)16.S - 16-bit parallel (no/have) sign cross-subtract-add instruction with saturation operation	411
15.21	PASXH.(U/S)16 - 16-bit parallel (no/have) sign cross add/subtract averaging instruction	412
15.22	PSAXH.(U/S)16 - 16-bit parallel (no/have) sign cross-subtraction and averaging instructions	413
15.23	ADD.64 - 64-bit addition instructions	414
15.24	ADD.(U/S)64.S - 64-bit (no/have) signed addition instruction with saturation operation	416
15.25	SUB.64 - 64-bit subtraction instructions	417
15.26	SUB.(U/S)64.S - 64-bit (no/have) signed subtraction instruction with saturation operation	418
15.27	ASRI.S32.R - Immediate Number Arithmetic Right Shift Instruction with Rounding Operation	419
15.28	ASR.S32.R - Arithmetic Right Shift Instructions with Rounding Operations	420
15.29	LSRI.U32.R - Immediate Logical Right Shift Instruction with Rounding Operation	421
15.30	LSR.U32.R - Logical Right Shift Instructions with Rounding Operations	422

15.31 LSLI.(U/S)32.S - Immediate Number (Without/With) Signed Logical Left Shift Instruction with Saturation Operation	423
15.32 LSL.(U/S)32.S - (no/have) signed logical left shift instruction with saturation operation	425
15.33 PASRI.S16 - 16-Bit Parallel Immediate Number Arithmetic Right Shift Instruction.....	426
15.34 PASR.S16 - 16-Bit Parallel Arithmetic Right Shift Instruction	427
15.35 PASRI.S16.R - 16-Bit Parallel Immediate Number Arithmetic Right Shift Instruction with Rounding Operation 428	
15.36 PASR.S16.R - 16-Bit Parallel Arithmetic Right Shift Instruction with Rounding Operation	430
15.37 PLSRI.U16 - 16-Bit Parallel Immediate Logic Right Shift Instruction	431
15.38 PLSR.U16 - 16-bit Parallel Logic Right Shift Instruction.....	432
15.39 PLSRI.U16.R - 16-Bit Parallel Immediate Logical Right Shift Instruction with Rounding Operation	433
15.40 PLSR.U16.R - 16-Bit Parallel Logic Right Shift Instruction with Rounding Operation	434
15.41 PLSLI.16 - 16-Bit Parallel Immediate Logical Left Shift Instruction.....	436
15.42 PLSL.16 - 16-Bit Parallel Logic Left Shift Instruction.....	437
15.43 PLSLI.(U/S)16.S - 16-bit Parallel Immediate Number (Without/With) Signed Logical Left Shift Instruction with Saturation Operation	438
15.44 PLSL.(U/S)16.S - 16-bit parallel (no/have) signed logical left shift instruction with saturation operation	440
15.45 SEL - Bit Selection Instruction.....	441
15.46 PCMPNE.8 - 8 bit parallel compare unequal instructions.....	442
15.47 PCMPNE.16 - 16 bit parallel compare unequal instructions.....	443
15.48 PCMPHS.(U/S)8--8 Bit Parallel (No/Have) Signed Greater Than or Equal Compare Instructions	445
15.49 PCMPHS.(U/S)16 - 16 Bit Parallel (None/Have) Signed Greater Than or Equal Compare Instructions	446
15.50 PCMPLT.(U/S)8--8 Bit Parallel (None/Have) Sign Less Than Compare Selection Instruction ...	447
15.51 PCMPLT.(U/S)16--16 Bit Parallel (No/Have) Sign Less Than Compare Instructions	449
15.52 PMAX.(U/S)8--8 Bit Parallel (No/Have) Sign Fetch Large Value Instruction	450
15.53 PMAX.(U/S)16--16 Bit Parallel (No/Have) Signed Fetch Major Instruction.....	451
15.54 MAX.(U/S)32--32 Bit (no/have) sign fetch large value instruction.....	452
15.55 PMIN.(U/S)8--8 Bit Parallel (No/Have) Sign Fetch Minor Instruction.....	453
15.56 PMIN.(U/S)16--16 Bit Parallel (No/Have) Sign Fetch Minor Instruction.....	454
15.57 MIN.(U/S)32--32 bit (no/have) sign fetch minor instruction.....	455
15.58 DEXTI - Immediate Digital Intercept Instruction	456
15.59 EXT - Word Intercept Command.....	457
15.60 PKG - Immediate Number Shift Packing Instruction.....	458
15.61 PKGLL - low half-word packing instruction	459
15.62 PKGHH - High Half Word Packing Instruction	460
15.63 PEXT.U8.E - 8 bit parallel unsigned extended instructions.....	461
15.64 PEXT.S8.E - 8 bit parallel signed extended instructions	463
15.65 PEXTX.U8.E - 8 bit parallel unsigned cross extended instructions	465
15.66 PEXTX.S8.E - 8 bit parallel signed cross-extension instructions.....	467
15.67 NARL - Low Intercept Splicing Instructions	468
15.68 NARH - High Intercept Splicing Instruction.....	469
15.69 NARLX - Low Cross Intercept Splicing Instructions	470
15.70 NARHX - High Cross Intercept Splicing Instruction	471

15.71 CLIP1.(U/S)32 - Immediate number (no/have) symbol trimming saturation instruction.....	473
15.72 CLIP.(U/S)32 - (No/Have) Symbol Trimming Saturation Instruction	475

15.73 PCLPI.(U/S)16--16 Bit Parallel Immediate Number (Without/With) Symbol Trimming Saturation Instruction	477
15.74 PCLIP.(U/S)16--16 Bit Parallel (Without/With) Symbol Trimming Saturation Instruction	480
15.75 PABS.S8.S - 8-Bit Parallel Absolute Instruction with Saturation Operation	482
15.76 PABS.S16.S - 16-Bit Parallel Absolute Instruction with Saturation Operation	483
15.77 ABS.S32.S - 32-bit Absolute Instruction with Saturation Operation.....	484
15.78 PNEG.S8.S - 8-Bit Parallel Inverse Instruction with Saturation Operation	485
15.79 PNEG.S16.S - 16-Bit Parallel Inversion Instruction with Saturation Operation	486
15.80 NEG.S32.S - Inverse instruction with saturation operation.....	487
15.81 DUP.8 - 8-Bit Operand Copy Instruction	488
15.82 DUP.16 - 16-Bit Operand Copy Instruction.....	489
15.83 MULS.(U/S)32--32 Bit (no/have) signed multiply-accumulate instructions.....	490
15.84 MULA.(U/S)32.S - 32-bit (no/have) signed multiply-accumulate instruction with saturation operation	491
15.85 MULS.(U/S)32.S - 32-bit (no/have) signed multiply-accumulate instruction with saturation operation	493
15.86 MUL.S32.H - 32-bit Signed Multiplication Fetch High 32-bit Instruction.....	494
15.87 MUL.S32.RH - 32-Bit Signed Multiply Fetch High 32-Bit Instruction with Rounding Operation.....	495
15.88 RMUL.S32.H - 32-bit Signed Decimal Multiplication Take High 32-bit Instruction.....	496
15.89 RMUL.S32.RH - 32-Bit Signed Decimal Multiplication with Rounding Operations Fetch High 32-Bit Instruction	497
15.90 MULA.S32.HS - 32-Bit Signed Multiply Accumulate Fetch High 32-Bit Instruction with Saturation Operation	499
15.91 MULS.S32.HS - 32-bit signed multiply-accumulate-take-high 32-bit instruction with saturation operation	500
15.92 MULA.S32.RHS - 32-Bit Signed Multiply Accumulate High 32-Bit Instruction with Rounding and Saturation Operations	502
15.93 MULS.S32.RHS - 32-Bit Signed Multiply Accumulate High 32-Bit Instruction with Rounding and Saturation Operations	504
15.94 MULXL.S32 - 32-bit Signed Low Halfword Unaligned Multiply Instruction.....	505
15.95 MULXL.S32.R - 32-Bit Signed Low Half-Word Unaligned Multiply Instruction with Rounding Operation	506
15.96 MULXH.S32 - 32-bit Signed High Halfword Unaligned Multiply Instruction	507
15.97 MULXH.S32.R - 32-Bit Signed High Half-Word Unaligned Multiply Instruction with Rounding Operation	508
15.98 RMULXL.S32 - 32-Bit Signed Low Half-Word Unaligned Decimal Multiply Instruction.....	509
15.99 RMULXL.S32.R - 32-Bit Signed Low Half-Word Unaligned Decimal Multiply Instruction with Rounding Operation	511
15.100 RMULXH.S32 - 32-bit Signed High Halfword Unaligned Decimal Multiply Instruction	513
15.101 RMULXH.S32.R - 32-Bit Signed High Half-Word Unaligned Decimal Multiply Instruction with Rounding Operation	515
15.102 MULAXL.S32.S - 32-bit Signed Low Halfword Unaligned Multiply Accumulate Instruction with Saturation Operation	516
15.103 MULAXL.S32.RS - 32-Bit Signed Low Half-Word Unaligned Multiply Accumulate Instruction with Rounding and Saturation Operations	518
15.104 MULAXH.S32.S - 32-Bit Signed High Half-Word Unaligned Multiply Accumulate Instruction with Saturation Operation	519

15.105 MULAXH.S32.RS - 32-Bit Signed High Half-Word Unaligned Multiply Accumulate Instruction with Rounding and Saturation Operations	521
15.106 MULLL.S16 - 16-Bit Signed Low Half-Word Multiply Instruction	522
15.107 MULHH.S16 - 16-Bit Signed High Half-Word Multiply Instruction	523
15.108 MULHL.S16 - 16-Bit Signed High-Low Half-Word Multiply Instruction.....	524
15.109 RMULLL.S16 - 16-Bit Signed Low Half-Word Decimal Multiplication Instruction.....	525
15.110 RMULHH.S16 - 16-Bit Signed High Half-Word Decimal Multiplication Instruction.....	526
15.111 RMULHL.S16 - 16-Bit Signed High-Low Half-Word Decimal Multiplication Instruction	528
15.112 MULAHH.S16.S - 16-Bit Signed High Half-Word Multiply Accumulate Instruction with Saturation Operation 529	
15.113 MULAHL.S16.S - 16-Bit Signed High-Low Half-Word Multiply Accumulate Instruction with Saturation Operation	531
15.114 MULALL.S16.E - 16-Bit Signed Low Half-Word Multiply Accumulate Instruction with Extended Operation 532	

15.115 MULAHH.S16.E - 16-Bit Signed High Half-Word Multiply Accumulate Instruction with Extended Operation	533
15.116 MULAHL.S16.E - 16-Bit Signed High-Low Half-Word Multiply Accumulate Instruction with Extended Operation	534
15.117 PMUL.(U/S)16 - 16 Bit Parallel (No/Have) Signed Multiplication Instructions	535
15.118 PMULX.(U/S)16 - 16 bit parallel (no/have) signed cross multiply instructions	537
15.119 PRMUL.S16 - 16-Bit Parallel Signed Decimal Multiplication Instruction	538
15.120 PRMULX.S16 - 16-Bit Parallel Signed Crossed Decimal Multiplication Instruction	541
15.121 PRMUL.S16.H - 16-Bit Parallel Signed Decimal Multiplication Fetch High 16-Bit Instruction	543
15.122 PRMUL.S16.RH - 16-Bit Parallel Signed Decimal Multiplication with Rounding Operations Fetch High 16-Bit Instruction	545
15.123 PRMULX.S16.H - 16-bit Parallel Signed Crossed Decimal Multiplication Fetch High 16-bit Instruction	548
15.124 PRMULX.S16.RH - 16-Bit Parallel Signed Cross-Decimal Multiplication with Rounding Operations Fetch High 16-Bit Instruction	550
15.125 MULCA.S16.S - 16-Bit Signed Multiply-Chain-Add Instruction with Saturation Operation	553
15.126 MULCAX.S16.S - 16-Bit Signed Cross Multiply Chain Add Instruction with Saturation Operation	554
15.127 MULCS.S16 - 16-Bit Signed Multiply Chain Subtract Instructions	555
15.128 MULCSR.S16 - 16-Bit Signed Inverse Multiply Chain Subtract Instruction	556
15.129 MULCSX.S16 - 16-Bit Signed Cross Multiply Chain Subtract Instruction	557
15.130 MULACA.S16.S - 16-Bit Signed Multiply Chain Plus Accumulate Instruction with Saturation Operation	558
15.131 MULACAX.S16.S - 16-Bit Signed Cross Multiply Chain Plus Accumulate Instruction with Saturation Operation	559
15.132 MULACS.S16.S - 16-Bit Signed Multiply Chain Subtract Accumulate Instruction with Saturation Operation	561
15.133 MULACSR.S16.S - 16-Bit Signed Inverse Multiply Chain Decrease Accumulate Instruction with Saturation Operation	562
15.134 MULACSX.S16.S - 16-Bit Signed Cross Multiply Chain Decrease Accumulate Instruction with Saturation Operation	564
15.135 MULSCA.S16.S - 16-Bit Signed Multiply Chain Plus Accumulate and Decrease Instructions with Saturation Operations	566
15.136 MULSCAX.S16.S - 16-Bit Signed Cross Multiply Chain Plus Accumulate and Decrease Instruction with Saturation Operation	567
15.137 MULACA.S16.E - 16-Bit Signed Multiply Chain Plus Accumulate Instruction with Extended Operations	569
15.138 MULACAX.S16.E - 16-Bit Signed Cross Multiply Chain Plus Accumulate Instruction with Extended Operations	570
15.139 MULACS.S16.E - 16-Bit Signed Multiply Chain Subtract Accumulate Instruction with Extended Operations	571
15.140 MULACSR.S16.E - 16-Bit Signed Inverse Multiply Chain Decrease Accumulate Instruction with Extended Operations	572
15.141 MULACSRX.S16.E - 16-Bit Signed Cross Multiply Chain Subtract Accumulate Instruction with Extended Operations	573
15.142 MULSCA.S16.E - 16-Bit Signed Multiply Chain Plus Accumulate and Decrease Instructions with Extended Operations	574

15.143 MULSCAX.S16.E - 16-Bit Signed Cross Multiply Chain Plus Accumulate and Decrease Instruction with Extended Operations.....	575
15.144 PSABSA.U8 - 8-Bit Parallel Unsigned Subtracted Absolute Value Chaining Instruction	576
15.145 PSABSAA.U8 - 8-Bit Parallel Signed or Unsigned Subtract Absolute Chaining Accumulation Instruction	
577	
15.146 DIVSL - Signed Long Division Instruction.....	578
15.147 DIVUL - Unsigned Long Division Instruction.....	579
15.148 MULACA.S8 - 8-Bit Parallel Signed Multiply Chain Plus Accumulate Instruction.....	580
15.149 BLOOP - Loop Body Acceleration Instruction	581
15.150 LDBI.W - Address Self-Incrementing Word Load Instruction	583
15.151 LDBI.H - Unsigned Half-Word Load Instruction with Address Self-Incrementation	584
15.152 LDBI.HS - Address Self-Incrementing Signed Half-Word Load Instruction	584
15.153 LDBI.B - Unsigned Byte Load Instruction with Address Self-Incrementation	585
15.154 LDBI.BS - Address Self-Incrementing Signed Byte Load Instruction.....	586
15.155 PLDBI.D - Double Word Load Instruction with Address Self-Incrementation	587
15.156 STBI.W - Address Self-Incrementing Word Storage Instruction.....	589

15.157	STBI.H - Address Self-Incrementing Half-Word Storage Instruction	589
15.158	STBI.B - Address Self-Incrementing Byte Store Instruction.....	590
15.159	LDBIR.W - Register Address Self-Incrementing Word Load Instruction	591
15.160	LDBIR.H - Unsigned Half-Word Load Instruction for Register Address Self-Incrementation	592
15.161	LDBIR.HS - Signed Half-Word Load Instruction with Register Address Self-Incrementation	593
15.162	LDBIR.B - Register Address Self-Incrementing Unsigned Byte Load Instruction	594
15.163	LDBIR.BS - Register Address Self-Incrementing Signed Byte Load Instruction.....	594
15.164	PLDBIR.D - Register Address Self-Incrementing Double-Word Load Instruction.....	595
15.165	STBIR.W - Register Address Self-Incrementing Word Store Instruction	597
15.166	STBIR.H - Register Address Self-Incrementing Half-Word Storage Instructions.....	597
15.167	STBIR.B - Register Address Self-Incrementing Byte Store Instruction.....	598

Chapter I OVERVIEW

1.1 brief

The E804 is a 32-bit energy-efficient embedded CPU core for control applications, featuring low cost, low power consumption, and high code density.

The E804 utilizes a 16/32-bit mixed-code instruction system and is designed with a lean and efficient 3-stage pipeline.

The E804 provides a variety of configurable features, including hardware floating-point unit, on-chip cache, DSP acceleration unit, trusted guard technology, on-chip tightly coupled IP, etc., which can be configured by users according to application needs. In addition, the E804 provides a multi-bus interface that supports flexible configuration of the system bus, instruction bus, and data bus, and the E804 is optimized for memory copy applications to achieve the ultimate memory copy performance. The E804 is optimized for memory copy applications for the ultimate in memory copy performance. In addition, the E804 is specially accelerated for interrupt response, with an interrupt response delay of only 13 cycles.

1.2 specificities

The main features of the architecture and programming model of the E804 are:

- Reduced Instruction Set Processor Architecture (RISC)
- 32-bit data, 16-bit/32-bit mixed-code instructions;
- 16 32-bit general purpose registers;
- 3-stage assembly line;
- Sequential launch, sequential execution, sequential retirement;
- Configurable multi-bus interface;
- Configurable cache;
- Configurable hardware floating point unit;
- Configurable DSP processing unit;
- Configurable Trusted Protection Technology;
- Configurable memory protection units (0 - 8)
- Configurable hardware multiplier that supports 1-cycle fast multiplication results;
- Configurable tightly coupled IP, including vector interrupt controllers and timers;

User's Manual

- Supports multiple processor clock to system clock ratios;
- Interrupt response latency is only 13 processor cycles.

The main features of the E804 microarchitecture are:

- Static branch prediction;
- Hardware division is supported;
- Supports sequential memory access;
- Support big endian and little endian.

1.3 Configurable Options

The E804 can be configured with the options shown in [Table 1.1](#).

Table 1.1: E804 Configurable Options

Configurable units	Configuration options	in detail
hardware multiplier	No/yes	1 cycle to produce multiplication result if configured, otherwise 1-32 weeks Completion of the period.
Memory Protection Unit	0 to 8 table entries	Can be configured as 0-8 table entries, where 0 means no memory protection is implemented Protect the unit.
cache	None / 2K / 4K / 8K / 16K	Can be configured as 2KB, 4KB, 8KB, 16KB.
floating point unit	No/yes	Single-precision floating-point hardware processing.
DSP Acceleration Unit	No/yes	Hardware acceleration for audio, motor, and other applications. When configuring this technology, the processing The processor is configured with 32 32-bit general purpose registers.
Trusted Protection Technology	No/yes	Configure this technology, combined with ZT Micro's SoC platform technology/systems The system software, which will provide the system's safety and security features.
command bus	Fixed Inclusion	Only AHB-Lite, the direct output(Non-Flop-out) method is supported.
data bus (computer)	Fixed Inclusion	Only AHB-Lite, the direct output(Non-Flop-out) method is supported.

System Manual	AHB Compatible/ AHB-Lite Compatible	It can be configured to be compatible with the AHB protocol or compatible with the AHB-Lite protocol. Only the direct output (Non-Flop-out) method is supported.
Vector Interrupt Controller	None/INT8/INT16 /INT24/INT32	Supports nested processing of hardware interrupts. Supports 8/16/24/32 interrupts Source.
system timer	No/yes	For timing.

1.4 Design for Measurability

The E804 supports Scan Chain Testing (SCAN) and Built-In Self-Test (BIST). The SCAN test is used to test the processor's internal combinatorial and timing logic for manufacturing errors, and the Built-In Self Test is used to test the cache for manufacturing errors.

The number of scan chains for the E804 can be specified by the customer.

1.5 Design for Debuggability

The E804 uses the JTAG standard (2-wire) to design the hardware debugging interface. The E804 supports all common debugging functions, including soft breakpoints, memory breakpoints, register checking and memory checking and modification, instruction single-step and multi-step tracing, program flow tracing, etc. For more details, see [Debugging Interface](#). For details, see [Debugging Interface](#).

1.6 naming convention

1.6.1 notation

The standard symbols and operators used in this document are listed in the following table.

1.6.2 nomenclature

- Logic 1 is the level value that corresponds to the Boolean logic true.
- A logic 0 is a level value that corresponds to a Boolean logic pseudo.
- Setting refers to bringing one or more bits to the level corresponding to logic 1.
- Clear means to bring one or more bits to the level corresponding to a logic 0.
- Reserved bits are reserved for function expansion and have a value of 0 when not specified.
- A signal is an electrical value that conveys information through its state or transitions between states.
- A pin is an indication of an external electrical physical connection, and multiple signals can be connected to the same pin.
- Enable means to put a discrete signal in an active state: a low level active signal switches from high to low; a high level active signal switches from low to high.
- Inhibit means to make a signal in the enable state change its state: the active signal switches from low level to high level;
The active signal switches from high level to low level.
- LSB represents the least significant bit and MSB represents the most significant bit.

符号	功能
+	加
-	减
*	乘
/	除
>	大于
<	小于
=	等于
≥	大于或等于
≤	小于或等于
!=	不等于
.	与
+	或
⊕	异或
NOT	取反
:	连接
⇒	传输
↔	交换
±	误差
0b0011	二进制数
0x0F	十六进制数

Figure 1.1: Symbol List

User's Manual

- The identifier is followed by a number indicating the range, from high to low indicating a set of signals, e.g., addr[4:0] would indicate a set of address buses, with the highest bit being addr[4] and the lowest bit being addr[0].
- A single identifier indicates a single signal, e.g. pad_cpu_reset_b indicates a single signal. Sometimes a number is added to the identifier to indicate a certain meaning, e.g. addr15 for bit 16 of a bus.

Chapter 2 Microarchitecture

2.1 Structural Block Diagram

The block diagram of E804 is shown Figure 2.1.

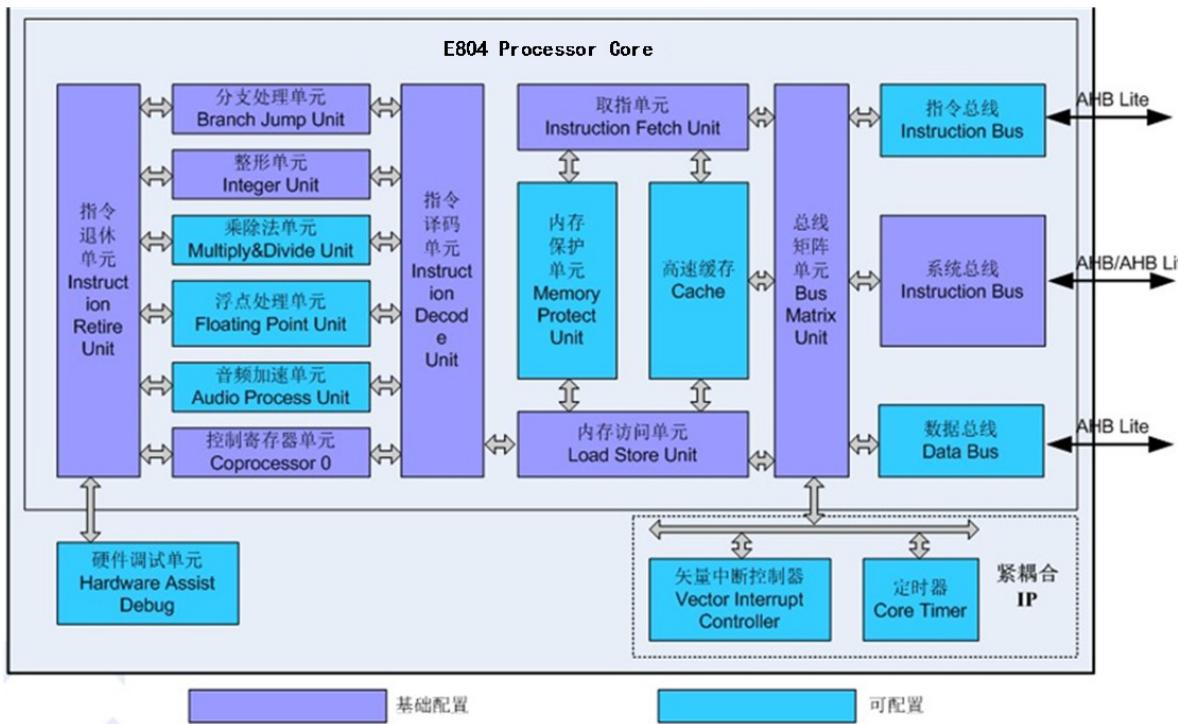


Figure 2.1: Block Diagram of the E804 Architecture

The finger fetch unit is responsible for accessing and fetching instructions, and can fetch 32-bit data per clock cycle, i.e., it can fetch one 32-bit finger per cycle.

order or two 16-bit instructions. In addition, the fetch unit predicts branching instructions and splits complex instructions.

The instruction decoding unit decodes instructions and accesses general-purpose registers as well as completes the firing of instructions.

The branch processing unit checks branch prediction instructions and processes register jump instructions. The shaping unit is responsible for the execution of ALU instructions. The control register unit processes the control register-related instructions (MTCR/MFCR). The execution delay of branch instruction, ALU instruction and control register instruction is 1 cycle.

The multiplication and division unit is responsible for the multiply, multiply-accumulate, and divide instructions. The multiplication implementation is user configurable for both slow and fast multiplication.

User's Manual

The slow multiplication method has a small cost overhead but requires 1-34 cycles to produce the multiplication result. In this case, slow multiplication has a small cost overhead but requires 1-34 cycles to produce the multiplication result, while fast multiplication supports one cycle to produce the multiplication result.

The result is a good one, but the cost is high. Division requires 3-35 cycles to produce results.

The Memory Access Unit is responsible for the sequential execution of Load/Store instructions and supports signed/unsigned byte/halfword/word access. Load/Store instructions can be executed sequentially to optimize performance.

The Retirement Unit collects the execution results of instructions and processes them, completing the write-back of the results and the handling of interrupts and exceptions. In addition, the retirement unit is also responsible for interacting with the hardware debugging unit.

The E804 implements a configurable Memory Protection Unit (MPU) to protect memory access to sensitive data, supporting 1-8 table entries configurable. The user can define the access attributes of each protected zone through the settings of the MPU.

The E804 implements a configurable cache unit, supporting 2KB~32KB configurations. The cache adopts the structure of four-channel group connection and supports two modes of operation: write-back and write-through.

The E804 is designed with a multi-layered bus interface, supporting three bus interfaces instruction bus, data bus and system bus. The instruction/data bus enables high-performance instruction/data access, and users can design local memory subsystems through the instruction/data bus to improve the overall performance of the system. The instruction/data bus supports the AHB-Lite interface, and only supports the direct output (non-flop-out, which requires certain interface timing) method. The system bus also supports only the direct output method, and且 supports both AMBA2 AHB protocol and AHB-Lite protocol configurable. The CPU clock and the system peripheral bus clock must maintain 1:1 relationship.

To facilitate system integration and development, the E804 is designed with tightly coupled IP, including a vector interrupt controller and a system timer, which can be configured by the user according to application needs. The vector interrupt controller supports up to 32 interrupt sources, both level and pulse interrupt modes, and且 supports interrupt nesting in hardware. The system timer provides a 24-bit loop decrement counter, which can be timed according to the CPU clock or external reference clock, and supports interrupt generation by the system timer. See [Tightly Coupled IP](#) for more information on .

The E804 is designed with a configurable floating-point processing unit for floating-point applications. The floating-point processing unit implements only single-precision floating-point operations.

The E804 is designed with a configurable DSP acceleration unit for audio, motor, etc. The DSP acceleration unit implements a subset of DSP acceleration instructions.

The E804 is designed for security protection with configurable trusted protection technology, which works with our SoC platform and system software to provide security protection for the system. For details, please refer to the "Trusted Protection Technology Manual".

2.2 Introduction of assembly line

The E804 is designed with a lean and efficient 3-stage pipeline, namely fetch, decode and execute. Most of the integer instructions (including branch instructions, integer arithmetic class instructions, load-

store instructions, and DSP-accelerated instructions) can be executed in the 3-stage pipeline, while some of the floating-point instructions require an additional execution cycle. For some floating-point instructions, an additional execution cycle is required.

Table 2.1: E804 Pipeline Description

Name of assembly line	abridge	The role of the assembly line
indexing	IF	<ol style="list-style-type: none"> 1. Initiate a command fetch request and process the returned command data; 2. Instruction pre-decoding; 3. Static branch prediction; 4. Complex instruction splitting.
decode	ID	<ol style="list-style-type: none"> 1. Command Decoding; 2. Instruction correlation analysis; 3. Command to fire; 4. Executes the branch jump instruction; 5. Calculates the address of the load store instruction and initiates an access request.
fulfillment	EX	<ol style="list-style-type: none"> 1. Executes and completes integer class instructions; 2. Executes floating-point class instructions; 3. Executes DSP acceleration class instructions; 4. Finish loading the storage instructions; 5. Writeback of instruction execution results; 6. Directed to retire; 7. Exception and interrupt handling.
Implementation 2	EX2	Completes some of the floating point class instructions.

2.3 Floating Point Processing Unit

Designed for floating-point applications, the E804 is a low-cost, power-efficient floating-point processing unit with key features including:

- Fully compatible with ANSI/IEEE Std 754 floating point standard (with system software support)
- Only single-precision floating-point operations are supported;
- Four rounding methods are supported: rounding to zero, rounding to positive infinity, rounding to negative infinity, and rounding to proximity;
- Supports both caught and uncaught handling modes for floating point exceptions;
- Supports precise handling of floating-point exceptions;
- Floating-point hardware division and squaring is supported;
- 16 independent single-precision floating-point registers;
- Single-emitter architecture, processing one floating-point instruction per cycle;

User's Manual

- Supports sequential launch, sequential execution, and sequential write-back of floating-point arithmetic instructions;
- Contains three separate execution pipelines for floating-point ALU, floating-point multiplication, and floating-point division and squaring;
- Optimized execution delay technology, except for floating-point divide-and-square instructions, can be executed in 1-2 clock cycles.

2.4 DSP Expansion Unit

The E804 is designed as a low-cost, power-efficient DSP processing unit for audio, motor, and other applications.

Key design features include:

- Essentially all operations defined in the existing ARM/Andes DSP instruction subset are covered;
- Extreme acceleration technology for short cycles;
- Data prefetching techniques for arrays/matrices. Key features include:
 - The DSP instruction subset manipulates the general purpose registers and expands them to 32;
 - The types of operations supported include multiplication, multiply-accumulate, shift, add, subtract, data wrapping, comparison, and memory access;
 - Supports single-command multi-data operation. The data types supported by single-command multi-data operation are word/half-word, and the operation types supported are multiply, multiply-accumulate, shift, add, and subtract;
 - The maximum operation width is 64 bits and supports up to 4 general-purpose register accesses and 2 general-purpose register write-backs a single instruction;
 - Partial operations support both rounded and unrounded values;
 - Partial operations support both saturated and unsaturated value taking.
 - Partial arithmetic supports both integer and decimal operations;
 - Single-transmit architecture, processing one DSP extended instruction per cycle;
 - Supports sequential launch, sequential execution, and sequential write-back of DSP extended instructions;
 - Optimized execution delay technology, all instructions can be completed in 1 clock cycle.

2.5 Trusted Protection Technology

The E804 is geared towards the security domain and is designed with trusted protection techniques that can be used to secure the system, with key features including:

- Based on the same physical processor core, two worlds are virtualized, the secure world and the non-secure world;

- Supports two-world spatial division of memory and I/O space in software form;
- Support for trusted interruptions;
- Support for trusted debugging;
- Support for credible guidance;

For more information, please refer to the Trusted Protection Technical Manual.

2.6 Tightly Coupled IP Architecture

In order to improve the system integration of the E804 and to facilitate user integration and development, the E804 implements a series of system IPs closely related to the CPU core, which are collectively referred to as Tightly Coupled IP (TCIP).

The main features of the vector interrupt controller include:

- The number of interrupts is hardware configurable, supporting 1/2/4/8/16/32 interrupt sources;
- The interrupt priority is software definable with 4 levels of priority;
- Hardware interrupt nesting is supported;
- Both level and pulse interrupt

source signals are supported. Key

features of the system timer

include:

- 1 24-bit counter;
- Supports selectable input clock, either CPU clock or external input clock;
- Interrupt generation is supported;

See [Tightly Coupled IP](#) for more information.

Chapter 3 Programming Model

3.1 Operating Mode and Register View



Figure 3.1: Programming Model

The E804 defines two processor operating modes: normal user mode and super user mode. The two operating modes correspond to different operating privileges, which are mainly reflected in the following aspects:

1. Access to registers;
2. Use of Privileged Directives;

User's Manual

3. Access to memory space.

Programs access registers according to their privileges. Normal user programs are allowed to access only those registers assigned to the normal user mode; system software working in the superuser mode has access to all registers and uses the control registers for superuser operations. By managing register access privileges, user programs can be prevented from accessing privileged information, and the operating system provides management and services to normal user programs by coordinating their behavior with that of normal user programs.

Most instructions can be executed in both modes, but some privileged instructions that have a significant impact on the system can only work in superuser mode. Privileged instructions include STOP, DOZE, WAIT, MFCR, MTCR, PSRSET, PSRCLR, and RTE.

The E804 is designed with a memory protection unit. The operating system can manage and control access to memory by setting the memory protection unit. Programs access memory spaces according to their privileges, and normal user programs are only allowed to access those memory spaces that are open to normal user mode.

The operating mode of the processor is controlled by the S bit of the Processor Status Register (PSR). When the S bit of the PSR is set, the processor operates in Super User mode, and when the S bit is cleared to zero, the processor operates in Normal User mode. In normal user mode, the processor uses the normal user programming model. During exception handling, the processor switches the mode from Normal User Mode to Super User Mode and stores current PSR value in the Exception-Preserving Processor Status Register (EPSR), then sets the S bit in the PSR to force the processor into Super User Mode. After the exception has been handled, to return to the previous operating mode, the system function executes RTE (Return from Exception), which re-fetches the finger from where the exception occurred and executes it. As a system call instruction, the TRAP#n instruction provides a common user program with a controllable interface to the operating system service program. A normal user program can use TRAP#n to generate a system call exception and force the processor into superuser mode.

The registers that can be operated in normal user mode include 16 32-bit general purpose registers, 32-bit program counter (PC) condition/feed bits

(The C bit is the lowest bit in the PSR and is the only data bit in the PSR that can be accessed in normal user mode. The C bit is the lowest bit in the PSR and is the only data bit in the PSR that can be accessed in normal user mode. In addition to the registers accessible in Normal User Mode, Super User Mode includes the PSR registers containing processor control and status information, a set of Exception Shadow Registers EPSR and EPC hold the PSR and PC in the event of an exception, a register VBR to hold the base address of the interrupt vector table, and other related control registers (as determined by configuration))

3.2 general-purpose register

The E804 base configuration consists of 18 32-bit general-purpose registers: R15~R0, R14 ~R28, and the number of general-purpose registers is expanded to 33 with the configuration of the DSP expansion unit: R31~R0, R14 ~.

Table 3.1: General User Programming Mode Registers

name (of a thing)	functionality
R0	Uncertain, the first argument to a

	function call.
R1	Uncertain, the second argument to the function call.
R2	Uncertain, the third argument to a function call.
R3	Uncertain, the fourth argument to a function call.
R4	Not sure.
R5	Not sure.
R6	Not sure.
R7	Not sure.

Continued on next page

Table 3.1 - continued from previous page

R8	Not sure.
R9	Not sure.
R10	Not sure.
R11	Not sure.
R12	Not sure.
R13	Not sure.
R14 (User SP)	Stack pointer (normal user mode)
R14' (Spv SP)	Stack pointer (superuser mode)
R15	Link registers.
R16	Not sure.
R17	Not sure.
R18	Not sure.
R19	Not sure.
R20	Not sure.
R21	Not sure.
R22	Not sure.
R23	Not sure.
R24	Not sure.
R25	Not sure.
R26	Not sure.
R27	Not sure.
R28	Not sure.
R29	Not sure.
R30	Not sure.
R31	Not sure.

The general-purpose registers are used to store instruction operands and instruction execution results as well as address information. When the DSP extension unit is configured, the base instruction set and the DSP instruction subset can operate the general-purpose registers R31~R0 at the same time.

The E804 is designed with a stack pointer R14 for normal user mode and super user mode respectively. normal user mode can only access R14 (User SP) of the user program; in super user mode, the system software can not only access R14 (Spv SP) of the system program, but also can access R14 (User SP) of the normal user program. The indexing of general-purpose register R14 in Super User mode will use Super User mode register R14 (Spv SP). If the user wants to access R14(User SP) in Super User Mode, this can be done by accessing CR<14,1> via MFCR/MTCR.

3.2.1 Condition Code/Feed Flag Bit

The condition code/progression flag bit represents the result of an operation. The condition code/feed flag bit can be set as the result of a comparison operation instruction or as the result of another high-precision arithmetic or logic instruction.

3.3 System Control Register

The system programmer uses the Super User mode to set system operating functions, I/O controls, and other restricted operations. The system control registers that can be operated in the superuser mode are shown below:

- Processor Status Register (PSR);
- Vector Base Address Register (VBR);
- Exception Hold Program Counter (EPC);
- The Exception-Preserving Processor Status Register (EPSR);
- Product serial number register (CPUID);
- Cache Configuration Register (CCR);
- Can be highly moderated with the Configuration of Access Privileges Register (CAPR)*;
- Protected Area Control Register (PACR)*;
- Protected Region Selection Register (PRSR)*;
- Hidden Operation Register (HINT).

Note: * Valid only for specific configurations.

Table 3.2: Super User Programming Model Control Registers

name (of a thing)	typology	Control register number/address	descriptive
PSR	Read/ Write	CR<0,0>	Processor Status Register.
VBR	Read/ Write	CR<1,0>	Vector base address register.
EPSR	Read/ Write	CR<2,0>	The exception retains the processor status register.
EPC	Read/ Write	CR<4,0>	Exception Retention Program Counter.
CPUID	phrase marked by pause	CR<13,0>	Product serial number register.
CCR	Read/ Write	CR<18,0>	Cache Configuration Register.
CAPR	Read/ Write	CR<19,0>	Highly cacheable and accessible configuration registers.
PACR	Read/ Write	CR<20,0>	Protected Area Control Register.

PSR	Read/ Write	CR<21,0>	Protected Area Selection Register.
CHR	Read/ Write	CR<31,0>	Implicitly manipulate registers.
R14 (User SP)	Read/ Write	CR<14,1>	Stack pointer register.

3.3.1 Processor Status Register (PSR, CR<0,0>)

The Processor Status Register (PSR) stores the current processor status and control information, including the C-bit, interrupt valid bits, and other control bits. The Processor Status Register (PSR) can be accessed by software in Super User mode. The control bits indicate the following states for the processor: trace

User's Manual

mode (TM[1:0] super user mode, or normal user mode (S bit) They also specify whether the interrupt request is valid or not.

31	30	24	23		16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	-			VEC[7:0]	TM[1:0]	0	0	0	MM	EE	IC	IE	0	0	0	0	0	0	0	C	
1	0			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Figure 3.2: Processor Status Registers

S-Superuser mode setting bit:

- S is 0, the processor operates in normal user mode;
- S is 1, the processor operates in superuser mode;

The S bit is set to 1 by hardware when the processor restarts and enters exception handling.

VEC[7:0]-Anomaly vector value:

When an exception occurs, VEC[7:0] is used to calculate the exception service program entry address, and is cleared when reset.

TM[1:0]-Tracking mode bits:

In Instruction Trace mode, the E804 will enter the Trace Exception Service program after each instruction is executed, and in Jump Trace mode, the E804 will enter the Trace Exception Service program when it encounters the execution of an instruction containing a jump, whether it is a jump or not. These bits are cleared by hardware when cleared by reset and when entering the exception service program. The following table shows the TM[1:0] codes and the corresponding operating modes.

Table 3.3: TM[1:0] Codes and Corresponding Operating Modes

(be) wort h	descriptive
00	Normal execution mode
01	Command Trace Mode
10	undefined
11	Jump Tracking Mode

See [Exception Handling](#) for trace mode specifics.

MM-Unaligned Exception Mask Bit:

- When MM is 0, a loaded or stored address misalignment exception will occur normally and will not be masked i.e. the exception will be responded to;
- MM is 1, loaded or stored address misalignment exceptions are masked and data access requests are split into multiple alignment operations; this bit is cleared by reset.

- EE is 0, the exception is invalid, at which point any exception other than a normal interrupt, once it occurs, is considered by the E804 to be an unrecoverable exception;
- EE is 1, the exception is valid and all exceptions respond and use EPSR and EPC normally.

IC-Instruction Interrupt Control Bit:

User's Manual

- IC is 0, interrupts can only be responded to between instructions;
- An IC of 1 indicates that the interrupt can be responded to during long, multi-cycle * instruction execution; it is cleared by reset and is unaffected by other exceptions.

IE-Interrupt Effective Control Bit:

- IE is 0, the interrupt is invalid;
- IE is 1, the interrupt is valid.

This bit is cleared by reset and also on entering the exception service program.

C-Conditional Code/Feed Bit:

This bit is used as a conditional judgment bit for some instructions. It is not determined reset and after being copied to the EPSR.

Note: Some of the multi-cycle instructions including LDM, STM, PUSH, POP, IPUSH, IPOP, LDQ32, and STQ32 can be interrupted without waiting for them to complete, thus reducing the interrupt response latency. Multi-cycle instructions NIE may not respond to interrupts, and NIR responds to interrupts only at the end of instruction execution and cannot be interrupted by the PSR (IC) bit.

3.3.2 Update PSR

The PSR can be updated in several different ways, and changes to the control bits in the PSR can have a variety of effects. The PSR can typically be modified through exception responses, exception handling, and execution of the PSRSET, PSRCLR, RTE, and MTCR instructions. These modifications are implemented in four ways:

1. Exception Response and Exception Handling update PSRs:

Updating the PSR is part of the exception response and will update the S, TM, VEC, IE, and EE bits in the PSR.

2. The RTE command updates the PSR:

Updating the PSR as part of the RTE instruction execution changes all bits in the PSR.

3. The MTCR command updates the PSR:

the target register is CR<0,0>, updating the PSR will be performed as part of the MTCR instruction. This update will potentially change the value of all bits in the PSR, and immediately following instructions, exception events, and interrupt responses will utilize the new PSR value.

4. PSRCLR, PSRSET commands update the PSR:

Updating the PSR As part of the execution of the PSRCLR and PSRSET instructions, immediately following instructions, exception events, and interrupt responses will use the new PSR value.

3.3.3 Vector Base Address Register (VBR, CR<1,0>)

31	10 9	0
VECTOR BASE		Reserved
0		0

Figure 3.3: Base Address Vector Registers

3.3.4 Exception Reserved Register (CR<2,0>, CR<4,0>)

The EPSR, EPC, registers are used to save the current processor state when an exception is encountered. For more detailed information, see [Exception Handling](#).

3.3.5 Product serial number register (CPUIDR, CR<13,0>)

This register is used to record information about the CPU, including the model number of the CPU, its configuration, and so on. The product serial number register is read-only and its reset value is determined by the product itself.

3.3.6 Implicit operation register (CHR, CR<31,0>)

The processor implicit operation registers are a collection of some of the processor's implicit operations, including soft reset operations and the enabling of certain acceleration functions of the processor.

31	16 15 14	0	5 4 3 2 1 0
Reset value[15:0]		0	IAE RPE IPE BE 0
HS_EXP			
0	0	0	0 0 0 0 0

Figure 3.4: Implicit Operation Registers

Reset_value-Soft reset enable field:

Reset_Value is written to 16'hABCD, a processor reset operation is triggered and the processor resets itself and indicates that a software reset of the processor has occurred via the processor pin sysio_pad_srst, which is maintained for one system clock cycle.

HS_EXP-Hardware Stack Exception Indication bit:

When the processor supports the Trusted Execution technique, the hardware stack operation that indicates a trusted world is abnormal.

IAE-Interrupt Response Acceleration Control bit:

- IAE is 0, the interrupt response instruction NIE/IPUSH is executed sequentially;
- IAE is 1, the interrupt response instruction NIE/IPUSH will be executed speculatively to speed up the interrupt response. This bit is cleared by reset.

The RPE-function returns the acceleration control bit:

- RPE is 0, the function return instruction RTS is executed sequentially;

This bit is reset to 1.

IPE-Instruction Prefetch Acceleration Control Bit:

- IPE is 0, the instruction prefetch acceleration function is not enabled;
- When IPE is 1, the instruction prefetch acceleration function is enabled to speed up the instruction prefetch efficiency. This bit is cleared by reset.

BE-Bus Burst Transmit Control Bit:

- BE is 0, only SINGLE is supported for bus transfer type;
- When BE is 1, the system bus supports the Burst transfer type. This bit is cleared by reset.

3.3.7 Other Control Registers

Other control registers for the E804 include:

- Cache Configuration Register (CCR)*;
- Can be highly moderated with the Configuration of Access Privileges Register (CAPR)*;
- Protected Area Control Register (PACR)*;
- Protected Region Selection Register (PRSR)*;

Among them, Cache Configuration Register (CCR), Cacheable and Access Privilege Configuration Register (CAPR)*, Protected Area Control Register (PACR)*, and Protected Area Selection Register (PRSR)* are the control registers related to the memory protection unit settings, and they are valid when the CPU configures the memory protection unit, and the specific definitions of the control registers are described in [Memory Protection](#).

3.3.8 Normal User Mode General Purpose Register 14 (R14(User SP), CR<14,1>)

General Purpose Register 14 of the Normal User Mode is mapped to Control Register CR<14,1>.

3.3.9 Interrupt Pointer Register (R14(Int_SP),CR<15,1>)

The E804 can optionally configure the interrupt pointer (Int_SP) to be used to share the interrupt stack space under different threads to save total stack overhead. If ISE is enabled, the pointer is used only during interrupts (without spending), i.e., when the vector number is greater than or equal to 32, and the original pointer is used for hardware stacking and the rest of the time.

In the non-interrupt superuser state, this register is mapped to control register CR<15,1>, i.e., the superuser can access CR<15,1> by accessing

Accesses the interrupt stack pointer register.

3.4 data-size side

The data size end is proposed relative to the format of the memory data storage, the high address byte stored to the low bit of physical memory is defined as the big end; the high address byte stored to the high bit of physical memory is defined as the little end.

The E804 supports the little-end format and the V1 version big-end (CSKY CPUs are designed with two versions of big-end and little-end, V1 and V2) format.且 does not support non-aligned accesses. In the little-end format, the lowest bit byte data always corresponds to the lowest bit of the address regardless of the access size of the access instruction (byte/halfword/word); in the V1 version big-end format, the data needs to be determined according to the access size (byte/halfword/word) of the access instruction.

Table 3.4: Small End/Big End Storage Access Models

access size	access address	the small end	Version V1本 Big End
word	A	D3D2D1D0	D3D2D1D0
half-word	A	D1D0	D3D2
half-word	A+2	D3D2	D1D0
nibbles	A	D0	D3
nibbles	A+1	D1	D2
nibbles	A+2	D2	D1
nibbles	A+3	D3	D0

3.5 Data Alignment Access

Unaligned accesses are not supported for data, when the CPU detects that an unaligned access has occurred, it enters an unaligned access exception, refer to [Unaligned Access Exception \(Vector Offset 0X4\)](#) for details.

The E804 supports masking of unaligned access exceptions by configuring the MM bit in the Processor Status Register PSR, refer to [Processor Status Register \(PSR, CR<0,0>\)](#) for details.

3.6 system address mapping

To facilitate system integration and development, the E804 assigns address divisions and functions to the 4GB of memory space. The bus matrix unit arbitrates memory access(instruction or data access)addresses and distributes them to different buses.The E804 recommended system address divisions and functions are shown in the table below. To make the SOC design more flexible, chip vendors can also define their own division of the bus address space by configuring pad_bmu_iahbl_base ,pad_bmu_iahbl_mask, pad_bmu_dahbl_base, pad_bmu_dahbl_mask to achieve the purpose. See [Bus Matrix and Bus Interfaces](#) for specific configuration methods .

Table 3.5: Address Map

name (of a thing)	memory address space	functionality
command bus	0x00000000~0x1FFFFFFF	store command (computing)
data bus (computer)	0x20000000~0x3FFFFFFF	Storing data
system bus	0x40000000~0xDFFFFFFF	Functionality is defined by the system developer Can hold commands, data, and system IP
Tightly Coupled IP Bus	0xE0000000~0xFFFFFFFF	Access address space for tightly coupled IPs
system bus	0xF0000000~0xFFFFFFFF	Functionality is defined by the system developer Can hold commands, data, and system IP

The E804's Bus Matrix Unit arbitrates only on the basis of the memory access address and does not care about the type of memory access (instruction access/data access/tightly coupled IP access). Either a fetch instruction request or a fetch data request can access either bus and memory space. The programmer must ensure that the memory access address is correct to ensure successful access to the target memory. For example, in an E804 configured with an instruction bus and a data bus, if the instruction access address falls in [0x20000000,0x40000000], the Bus Matrix Unit will distribute the instruction access request to the data bus and access it from the data bus memory.

The E804 is highly configurable, allowing the user to choose to implement any one or more of the instruction buses, data buses, and tightly coupled IP buses depending on the application, and the way the buses are configured affects memory access as shown in the table below.

Table 3.6: Address Map

name (of a thing)	configurable	Impact of configuration situation on memory access	
command bus	Fixed Inclusion	Memory accesses located at [0x00000000-0x1FFFFFFF] are distributed to the instruction computer bus	
data bus (computer)	Fixed Inclusion	Memory accesses located [0x20000000-0x3FFFFFFF] will be distributed to data computer bus	
system bus	Fixed Inclusion	Accesses to the system bus space are distributed to the system bus	
Tightly Coupled IP Bus	configurable	realization	Memory accesses located [0xE000 0000-0xFFFFFFFF] will be distributed on tightly coupled IP bus
		unrealized	Memory accesses located [0xE0000000-0xFFFFFFFF] will be distributed to the system bus
system bus	Fixed Inclusion	Accesses to the system bus space are distributed to the system bus	

The above accesses will only occur in non-high-speed zones or in the case of cache misses.

3.7 memory access sequence

The E804 is designed with a multi-bus interface that allows the system integrator to connect different memories externally on the bus. Since the access delays of memory devices on different buses are different, to facilitate user development, the E804 is designed in hardware to ensure that the memory access instructions are completed strictly in the order of assembly instructions in sequence, avoiding the need for the user to ensure the order of memory accesses by software.

User's Manual

For example, in the following two instruction sequences, Ins A accesses the system bus memory and Ins B accesses the data bus memory. Assuming that the access latency of the system bus memory is much larger than that of the data bus memory, in order to ensure that the instructions are completed in the order of the program, the hardware ensures that the Ins A instruction is executed before allowing the Ins B instruction to be executed.

```
Ins A: ld r4, (r7)
```

```
Ins B: ld r5, (r14)
```

Chapter IV Exception handling

Exception handling (including instruction exceptions and external interrupts) is an important technique of the processor, whereby the processor stops the execution of the current instruction and switches to the processing of an exception event. These events include external interrupts, instruction execution errors, and system call requests. This chapter describes the types of exceptions, exception priorities, exception vector tables, exception returns, and bus error recovery.

4.1 Exception Handling Overview

Exception handling is the processor's transfer from normal program processing to a specific exception handler in response to an internal or external exception event. External events that cause exceptions include external device interrupt requests, read/write access errors, and hardware reboots; internal events that cause exceptions include illegal instructions, misalignment errors, privileged exceptions, and instruction traces, and the TRAP and BKPT instructions generate exceptions when they are executed normally. The TRAP and BKPT instructions also generate exceptions when executed normally.且 , illegal instructions, unaligned addresses for LD and ST accesses, and privileged instructions executed in user mode all generate exceptions. Exception handling utilizes the exception vector table to jump to the entry point of the exception service program.

The key to exception handling is to preserve the state of the CPU's current instruction run when an exception occurs, and to restore the state before exception handling when exiting exception handling. Exceptions can be recognized at various stages of the instruction pipeline and keep later instructions from changing the state of the CPU. Exceptions are handled at instruction boundaries, i.e., the CPU responds to an interrupt when an instruction retires and saves the address of the next instruction to be executed when it exits exception handling. Even if the exception is recognized during the fetch or decode phase, the exception is not handled until the corresponding instruction retires. The E804 determines which instruction's address is stored in the Exception Address Register based on whether or not the instruction was completed at the time the exception was recognized. For example, if the abnormal event is an external interrupt service request, the interrupted instruction will retire normally and change the state of the CPU, and the address of its next instruction will be stored in the Exception Address Register as the entry for the instruction when the interrupt returns; if the abnormal event is generated by a divide-by-zero instruction, because this instruction cannot be completed, it will retire abnormally but will not change the state of the CPU (i.e., it will not change the value of the register value), the address of this divide instruction will be stored in the Exception Address Register and the CPU will continue to execute this divide instruction when it returns from the interrupt service program.

Exceptions are handled in the following steps:

In the first step, the processor saves the PSR and PC into shadow registers (EPSR and EPC).

In the second step, put the processor into the Super User Mode by setting the Super User Mode Setting Bit S in the PSR to position 1 (regardless of the mode of operation the processor was in when the exception occurred)

In the third step, the exception vector number VEC field in the PSR is updated to the currently occurring exception vector number, identifying the category of the exception as well as the support for shared exception services.

In the fourth step, the exception response is disabled by clearing the exception enable bit EE in the PSR to zero. Any exception that occurs while EE is zero (except for normal interrupts) is treated by the processor as a non-recoverable error exception. The EPSR and EPC are also updated when a non-recoverable error exception occurs.

User's Manual

In the fifth step, the interrupt enable bit IE bit in the PSR is cleared to zero to disable responding to interrupts. The second, third and fourth steps above, occur simultaneously.

In step 6, the processor first calculates the exception entry address from the exception vector number in the PSR and then uses this address to obtain the address of the first instruction of the exception service program. Multiply the anomaly vector by 4 and add the anomaly vector base address (which exists in the vector base address register VBR, and is zero when VBR does not exist) to obtain the anomaly entry address, read a word from memory with the anomaly entry address, and [31:1] of the word to the program counter as the address of the first instruction of the anomaly service program (the lowest bit of the PC is always 0, which has nothing to do with the lowest bit of the PC), and then use this address to obtain the address of the first instruction of the anomaly service program. the lowest bit of the exception entry address value obtained from the exception vector table) For vector interrupts, the exception vector is provided by the external interrupt controller; for other exceptions, the processor determines the exception vector based on internal logic.

In the final step, the processor begins exception handling by starting execution at the first instruction of the exception service program and transferring control of the CPU to the exception service program.

All exception vectors are stored on the address space that the superuser is allowed to access. Only the restart vector is fixed in the processor address mapping. Once the processor has completed initialization, the VBR allows the base address of the exception vector table to be reloaded.

The E804 supports a vector table of up to 1024 bytes, i.e., 256 exception vectors (see Table 4.1 for details) The first 31 vectors are represented by the

The 32nd vector is reserved for vectors recognized within the processor and the remaining 224 vectors are reserved for external devices. The remaining 224 vectors are reserved for external devices. The external device causes the processor to respond to an interrupt service with an 8-bit interrupt vector and an interrupt request. The processor latches this interrupt vector in response to the interrupt request. For those devices that cannot provide an interrupt vector, the processor provides an automatic vector for general interrupts.

Table 4.1: Anomaly Vector Assignments

vector number	Vector offset (hexadecimal)	vector allocation
0	000	Reboot Exception.
1	004	Unaligned access exception.
2	008	Access Error Exception.
3	00C	Divide by zero anomalies.
4	010	Illegal command exception.
5	014	Privilege violation anomalies.
6	018	Tracking anomalies.
7	01C	Breakpoint exception.
8	020	Unrecoverable error exception.
9	024	Idly exception.
10	028	Normal interrupt (automatic vector)
11 - 15	02C - 03C	Reservations.

16 - 19	040 - 04C	Trap instruction exception (TRAP # 0 - 3)
20 - 21	050 - 054	Reservations.
22	058	TSPEND Interrupt
23-29	05C - 074	reservations
30	078	floating point exception
31	07C	Reservations.
32 - 255	080 - 3FC	Reserved for use by the vector interrupt controller.

4.2 Exception type

This section describes external interrupt exceptions and exceptions generated within E804. The following categories of exceptions are handled by the E804:

- Reboot Exception;
- Unaligned access exception;
- Access Error Exception;
- Divide by zero anomalies;
- Illegal command exception;
- Privilege violation anomalies;
- Tracking anomalies;
- Breakpoint exception;
- Unrecoverable error exception;
- Idly exception;
- Ordinary interruptions;
- Trap command exception;
- TSPEND Interrupt.

4.2.1 Restart Exception (vector offset 0X0)

The reboot exception is the highest priority of all exceptions and is used for system initialization and for recovering the system after a major failure. A reboot aborts all operations of the processor and aborted operations are not recoverable.

The restart exception sets PSR(S) high to enable the processor to operate in superuser mode and clears PSR(TM) to disable trace exceptions. The reboot exception also clears PSR(IE) to disable interrupt response. At the same time, the VBR (Vector Base Register) is cleared to zero, and the base address of the exception vector table is 0X00000000. The CPU reads the exception service program entry address from the exception vector table at offset address 0X0 and loads it into the program counter (PC)

4.2.2 Unaligned access exception (vector offset 0X4)

An address unaligned access exception occurs when the processor attempts to perform an access operation on an address boundary that does not match the access size. By setting PSR (MM) this exception can be masked, and the processor ignores the alignment check on the data and accesses smaller than this unaligned address on the address boundary closest to it. the EPC points to the instruction that attempted the unaligned access.

The unaligned access exception occurs only for data accesses.

4.2.3 Access error exception (vector offset 0X8)

An access error exception occurs if a bus access results in an incorrect reply. An access error exception also occurs when an access error occurs when accessing a memory-protected area. the EPC points to the instruction that initiated the erroneous access.

4.2.4 Divide by zero exception (vector offset 0X0C)

When the processor finds that the divisor of a division instruction is zero, the processor performs exception handling instead of executing the division instruction. the EPC points to the division instruction.

4.2.5 Illegal instruction exception (vector offset 0X10)

If the processor finds an illegal or unimplemented instruction when decoding, the instruction will not be executed and an exception is thrown. the EPC points to the illegal instruction.

4.2.6 Privilege violation exception (vector offset 0X14)

In order to protect system security, some instructions are granted privileges and they can only be executed in superuser mode. Attempting to execute any of the following privileged instructions in user mode results in a privilege violation exception: MFCR, MTCR, PSRSET, PSRCLR, RTE, STOP, WAIT, DOZE.

The processor performs exception handling before executing the instruction if it finds a privilege violation exception. the EPC points to the privileged instruction.

4.2.7 Trace Exception (Vector Offset 0X18)

To facilitate program development and debugging, the E804 traces each instruction or change of control flow instruction. In the instruction trace mode, a trace exception is generated for each instruction after it is executed to allow the debugger to monitor program execution. In jump trace mode, a trace exception is generated for each instruction that changes control flow (including POPs). For conditional jump instructions, the trace exception occurs whether the program jumps or not.

The TM bit of the PSR controls the trace mode. The state of TM determines whether a trace exception is generated when the instruction retires. Refer to Chapter 3 for the PSR's

TM Bit Definition.

Trace Exception Handling Begins After the Traced Instruction Retires且 Before the next instruction retires. the EPC points to the next instruction.

The following control-related instructions cannot be traced: RTE, TRAP, STOP, WAIT, DOZE, and BKPT. if EPSR (TP) is valid, the trace exception is handled as part of normal execution of the RTE, regardless of the state of the PSR or the TM bit of the EPSR.

If the traced instruction does not complete because another exception occurred, the trace exception will not be handled. If the processor discovers an interrupt waiting to be handled when the traced instruction retires, the interrupt has a higher priority than the traced exception, TP (Trace Exception Waiting to be Handled) of the PSR's shadow register will be valid, and the processor responds to the

interrupt. The trace exception waiting to be handled is handled when the RTE of the interrupt service program retires. The trace exception waiting to be processed is used to trace the previous instruction, which has been retired in response to an interrupt. In order to avoid the loss of the trace exception after the interrupt is processed, the trace exception needs to be processed immediately after the interrupt is processed, and the trace exception in this case has the highest priority, second only to the restart.

4.2.8 Breakpoint exception (vector offset 0X1C)

The E804 provides the breakpoint instruction BKPT, which generates a breakpoint exception when retired. The EPC points to this instruction when the breakpoint exception occurs.

4.2.9 Unrecoverable error exception (vector offset 0X20)

When PSR(EE) is zero, the exception generates an unrecoverable exception because then the information used for exception recovery (stored in EPC and EPSR) is rewritten due to an unrecoverable error.

Since the software as written excludes the possibility of an abnormal event occurring by default when the PSR (EE) is zero, if an exception occurs on the CPU in this case, such an error generally implies that there is a system error. In a service program for unrecoverable error exceptions, the type of exception that caused the unrecoverable error exception is uncertain.

4.2.10 IDLY Exception (Exception Offset 0X24)

The IDLY exception is used to indicate that a transmission error has occurred during a sequence of IDLY instructions. In this exception service program, the EPC points to the instruction that caused the transmission error. The exception service program should analyze the cause of the transmission error and back up the value of the EPC in order to re-execute the IDLY instruction sequence if necessary.

4.2.11 Trap instruction exception (vector offset 0X40 - 0X4C)

A number of instructions can be used to generate trap exceptions demonstrably. TRAP # n instruction forces the generation of an exception, which is used in a system call to a user program. In an exception-serving program, the EPC points to the TRAP instruction.

4.2.12 TSPEND interrupt (vector cheap 0X58)

When the vector interrupt controller is configured, the TSPEND interrupt is generated by setting the VIC_TSPEND register in software. Refer to [Tightly Coupled IP](#) for details.

4.2.13 Floating Point Exception (Vector Offset 0X78)

Floating-point instructions are subject to floating-point exceptions during arithmetic.

4.3 disruption exception

When an external device needs to request a service from the processor or send information needed by the processor, it can request an interrupt exception from the processor using the interrupt request signal and the corresponding interrupt vector signal.

The E804 provides a tightly coupled vector interrupt controller. Users can choose the E804 CPU plus vector interrupt controller for system integration, in which only the external IP interrupt source signal needs to be connected to the vector interrupt controller. Users can also choose the CPU core and

Gentech **E804**

integrate the interrupt controller in the system by themselves.
User's Manual

XUANTIE 玄铁

User's Manual

The E804 CPU core provides two interrupt request signals, supporting automatic provisioning of interrupt vector numbers and explicit provisioning of interrupt vector numbers by peripherals.

Figure 4.1 shows the interface signals to the processor core associated with interrupts. To support vectorized interrupts when the VIC is not configured, the peripheral provides the interrupt vector number with the 8 bit interrupt vector signals on interrupt requests or sets pad/intc_cpu_avec_b to use the automatic interrupt vector number. If PSR(IE) or PSR(EE) is cleared, the pad/intc_cpu_int_b input signal is masked and the processor does not respond to the interrupt, and the CPU responds to the interrupt when PSR(IE) or PSR(EE) are both valid. In this case, if pad/intc_cpu_avec_b is valid, the processor uses the automatic vector number, which has a vector offset of 0X28, otherwise the processor uses the vector number provided on pad/intc_cpu_vec_b[7:0]. The cpu_intc/pad_int_ack indicates that the CPU has responded to the interrupt. pad/intc_cpu_int_b and pad/intc_cpu_avec_b are both active low and pad/intc_int_ack is active high. When the VIC is configured, the appropriate interrupt signals are generated from the control signals, which are handled and responded to as described above.

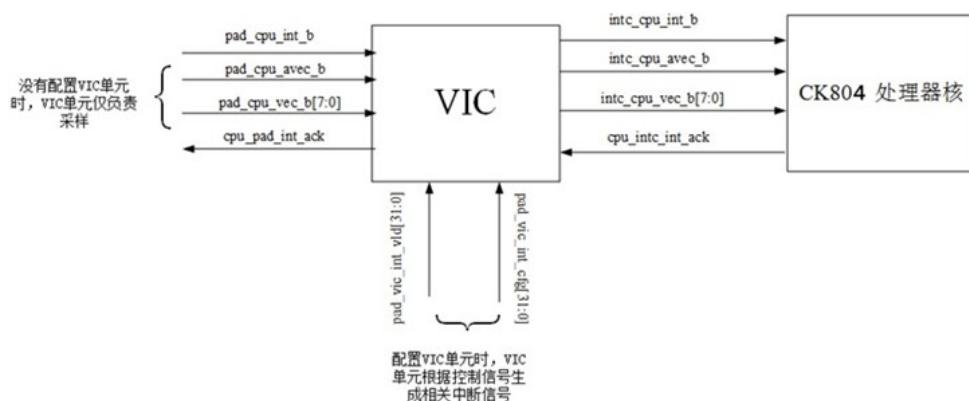


Figure 4.1: Interrupt Interface Signals

In Figure 4.2, the pad_cpu_int_b interrupt signal line is pulled low when the interrupt vector number is ready.

- If no VIC unit is configured, sys_clk samples ($T1=1\text{cyc}$) and then pulls down intc_cpu_int_b to send an interrupt request to the CPU;
- If the VIC is configured then it needs to be sampled and prioritized with cpu_clk over two cycles ($T1=2\text{cyc}$) before it can be sent to the CPU. The CPU sends an interrupt request.

After this signal is sampled by the rising edge of the CPU internal clock cpu_clk, the CPU receives the interrupt internally and obtains the interrupt vector based on the vector signal. Under the condition that all external systems can respond within one cycle, after responding to the interrupt, it takes five cycles ($T2=5\text{cyc}$) for the CPU to issue a fetch command request for the first instruction of the interrupt service program and enter the interrupt service program. In the interrupt service program, the external interrupt source should be cleared by the software, i.e., the interrupt active signal should be pulled up, and this signal should be

The E804 can be configured with interrupt acceleration. After the CPU responds to an interrupt, it starts to execute the NIE and IPUSH instructions speculatively while fetching the exception entry address. If the interrupt service program also executes these two instructions first, six cycles can be saved if all external systems respond within one cycle. The interrupt acceleration function is aborted if a speculative prediction error, access exception, or debug request occurs.

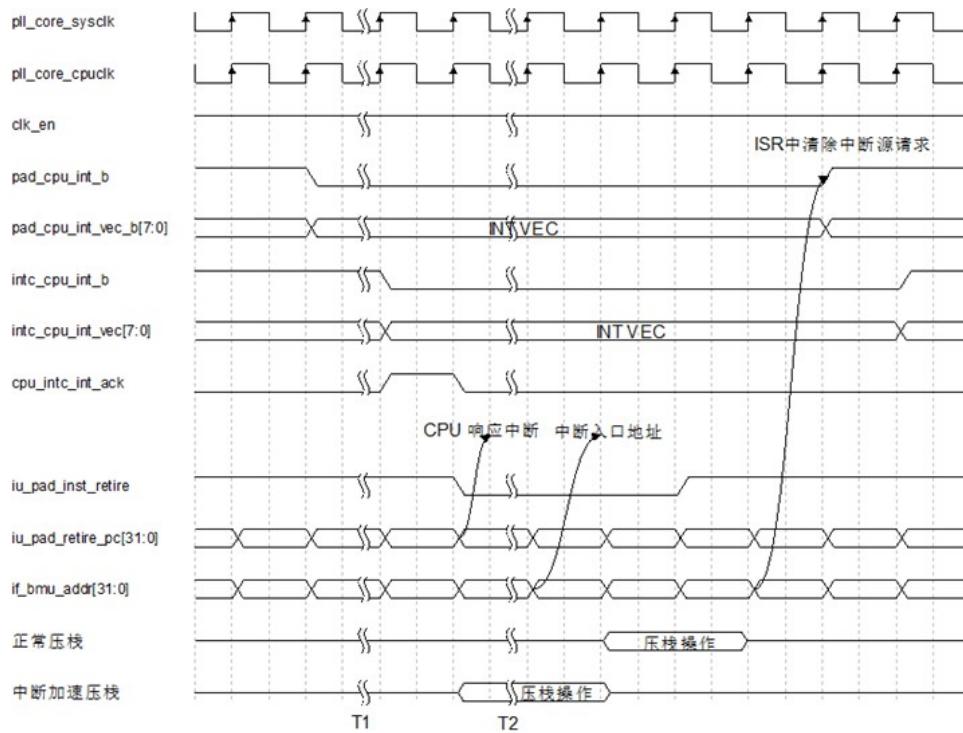


Figure 4.2: Timing Diagram of Processor Interrupt Response

If the corresponding interrupt controller is configured, multiple interrupt sources can be supported, and their corresponding interrupt priorities can be set and interrupt nesting can be realized. More detailed description of the interrupt mechanism and interface signals can be found in the [Tightly Coupled IP](#).

4.4 Exception Priority

As shown Table 4.2, the E804 prioritizes exceptions into 10 levels according to their characteristics and the order in which they are handled. Table 4.2, 1 represents the highest priority and 10 represents the lowest priority. It is worth noting that in group 9, several exceptions share a priority because they are mutually exclusive.

E804, multiple exceptions can occur at the same time. The restart exception is special in that it has the highest priority. All other exceptions are listed in Table

4.2 Prioritization is handled.

If multiple exceptions occur at the same time, the exception with the highest priority is handled first. When the processor re-executes the instruction that generated the exception after the exception returns, the remaining exceptions can be reproduced in order.

Table 4.2: Exception Prioritization

prioritization	The exception and its associated priority	hallmark
1	reboot anomaly (computing)	The processor suspends all program operations and initializes the system.
2	Pending tracking exceptions	If TP=1 for EPSR, after the RTE command retires, the processing The machine handles pending trace exceptions.
3	IDLY error	After the associated instruction retires, the processor saves the context and processes the different Regular.
4	misalignment	After the associated instruction retires, the processor saves the context and processes the different Regular.
6	General Interruption	If IC=0, the interrupt is responded to after the instruction is retired; If IC=1, the processor allows the interrupt to be Response.
7.0 7.1	Unrecoverable error exception access error	After the associated instruction retires, the processor saves the context and processes the different Regular.
8	Illegal Instructions Privileged Exceptions Divide by Zero Trap Instructions Breakpoint Instructions floating point anomaly (math.)	After the associated instruction retires, the processor saves the context and handles the exception.
9	Tracking Exceptions	After the associated instruction retires, the processor saves the context and processes the different Regular.

4.5 Exception return

The processor returns from the exception service program by executing the RTE instruction. The RTE

Gentech **E804**

User's Manual
instruction returns from the exception service program using the context stored in the EPSR and EPC
shadow registers.

XUANTIE 玄铁

Chapter 5 Tightly Coupled IP

5.1 Introduction to Tightly Coupled IP

In order to improve the system integration of the E804 and to facilitate the user's integration and usage, the E804 implements a series of system key IPs that are closely related to the processor, which are collectively referred to as Tightly Coupled IP (TCIP). The E804's Tightly Coupled IPs include the system timer CoreTim, the Vector Interrupt Controller (VIC), the on-chip cache, and the Control Register Unit (CRU). The E804's Tightly Coupled IP includes the system timer CoreTim, the vector interrupt controller VIC, and the on-chip cache unit CRU, which, in combination with the E804 and a small amount of resources, such as memory, make it possible to form a SoC system with minimal functionality, which improves the users ease of use of the E804 and reduces the cost of development and application of the E804.

The main functions of the tightly coupled IP of the E804 are shown in [Table 5.1](#) and the system architecture is shown in [Figure 5.1](#).

Table 5.1: Tightly Coupled IP Main Functions

IP Name	Key Features
system timer	Completes the timing function of the system and can wake up the CPU at low power consumption.
Vector Interrupt Controller	Complete interrupt collection, arbitration, hardware nesting, and interaction with the processor.
On-chip Cache Control Register Unit	Setting up the E804 on-chip cache, such as switching the Cache, configuring the region that can be slowed down, etc.

Unlike traditional IP, tightly-coupled IP is connected to the processor through a dedicated tightly-coupled IP bus interface, eliminating the need for system bus access. The tightly-coupled IP bus interface is directly connected to the Bus Matrix Unit (BMU) of the E804, which supports tightly-coupled IP access transmission in a single CPU clock cycle, which not only improves the access efficiency of tightly-coupled IP, but also improves the system integration efficiency.且 The tightly-coupled IP shares a unified memory address space with other system IPs, and carries out register access and function control through Load and Store instructions. The memory address allocation of the tightly coupled IP is shown in [Table 5.2](#).

Table 5.2: Memory Address Allocation for Tightly Coupled IPs

IP Name	memory address space
system timer	0xE000E010~0xE000E0FF
Vector Interrupt Controller	0xE000E100~0xE000ECFF
On-chip Cache Control Register Unit	0xE000F000~0xE000FFFF

In addition to communicating with the kernel through a dedicated bus interface, the tightly coupled IP interacts functionally with the processor in a directly connected manner. In particular, the vector interrupt controller system transmits arbitrated interrupt information to the processor and receives interrupt response signals from the processor; the on-chip cache is located between the kernel bus interconnect unit (BMU) and the instruction bus interface unit (I-AHBL).

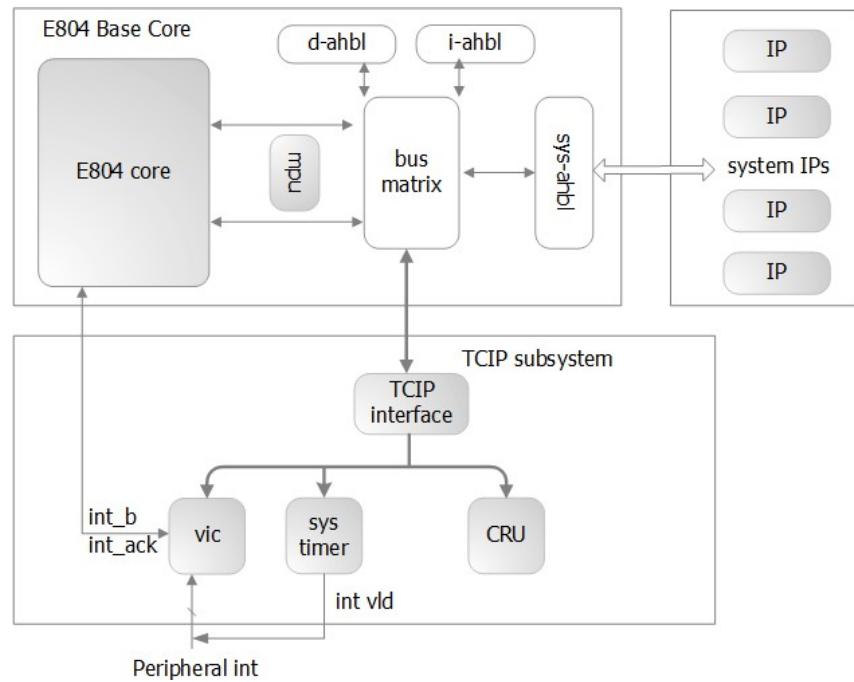


Figure 5.1: System Architecture for Tightly Coupled IPs

All control registers of the tightly coupled IP 32-bit registers, so they can only be accessed and transferred by word, any access by half-word or will cause unexpected errors, and且 tightly coupled IP registers only support small end format. All accesses to the tightly coupled IP registers must be made in super user mode, and accesses in normal user mode will result in an access error exception.

5.2 system timer

5.2.1 brief

The System Timer is an optional module for the E804 that is primarily used for timing. The system timer provides an easy-to-use 24-bit decrementing counter that starts when the system timer is enabled. When the counter decrements to 0, an interrupt request is made to the Vector Interrupt Controller to get a response from the processor and process the system timer transaction. The block diagram of the system timer is shown [Figure 5.2](#).

5.2.2 Register Definition

Each register of the system timer is 32 bits wide, and the register address space is as shown [Table 5.3](#).

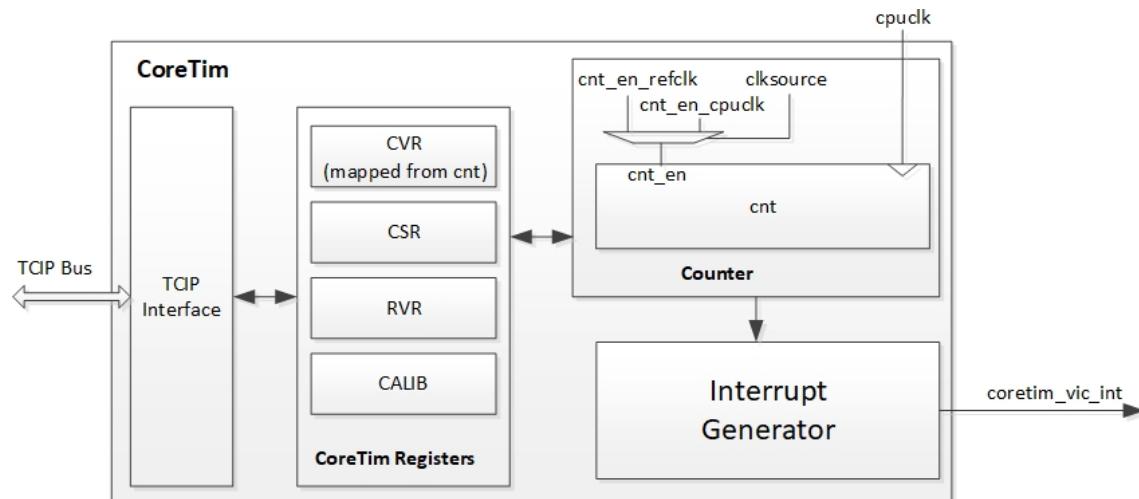


Figure 5.2: System Timer Structure

Table 5.3: System Timer Register Definitions

address	name (of a thing)	typology	starting value	descriptive
0xE000E010	CORET_CSR	Read/Write	0x00000004	Control Status Register.
0xE000E014	CORET_RVR	Read/Write	-	Backfill value register.
0xE000E018	CORET_CVR	Read/Write	-	Current value register.
0xE000E01C	CORET_CALIB	read-only (computing)	-	Calibration Register.
0xE000E020~0xE000E0FF	-	-	-	Reservations.

5.2.2.1 Control and Status Register (CORET_CSR)

CORET_CSR is the control and status register for the system timer as shown in Figure 5.3, and the bit descriptions of the CORET_CSR register are shown in Table 5.4 shown.

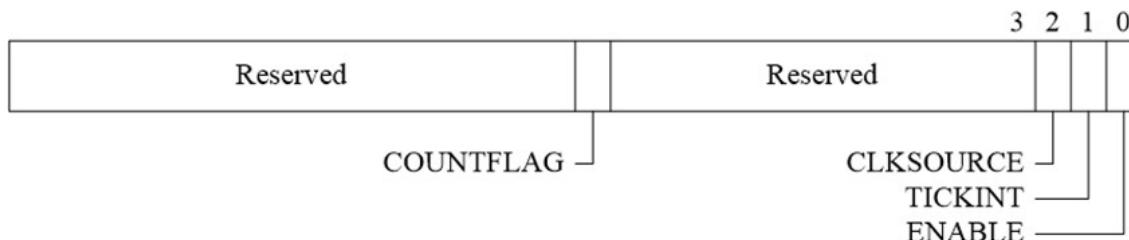


Figure 5.3: System Timer Control and Status Registers

Table 5.4: System Timer Control and Status Register Field Descriptions

classifier for honorific people	typology	name (of a thing)	descriptive
31:17	-	-	Reservations.
16	read-only (computing)	COUNTFLAG	Indicates whether the counter has counted to 0 since the last time this register was read: 0 The counter has not counted to 0 yet; 1 The counter has counted to 0. COUNTFLAG is set when the counter value changes from 1 to 0. A read of the CSR register and any write of the CVR register will cause COUNTFLAG Zeroing.
15:3	-	-	Reservations.
2	Read/Write	CLKSOURCE	Indicates the clock source of the system timer: 0 Use the optional external reference clock as the clock source for the counter; 1 Use the internal clock as the clock source for the counter; If there is no external clock, reading this bit will return a 1. Writing this bit has no effect. The external reference clock frequency must be less than or equal to half the internal clock frequency.
1	Read/Write	TICKINT	Indicates whether the interrupt status bit of the system timer is changed when the count reaches 0: 0 Counting to 0 does not affect the system timer interrupt status bit; 1 Counting to 0 changes the interrupt status bit of the system timer. Writing the CVR register zeroes the counter, but this method does not cause the system to The interrupt status bit of the timer is changed.
0	Read/Write	ENABLE	Indicates the enable status bit of the system timer: 0 The counter is not enabled; 1 Counter Enable.

5.2.2.2 Backfill Value Register (CORET_RVR)

The CORET_RVR register is used to assign a value to the CORET_CVR register at the beginning of each counting cycle. The CORET_RVR register and its bit descriptions are shown Figure 5.4 and Table 5.5.

31	24:23	0
Reserved		RELOAD

Figure 5.4: System Timer Backfill Value Registers

Table 5.5: System Timer Backfill Value Register Field Descriptions

classifier for honorific people	name (of a thing)	descriptive
31:24	-	Reservations.
23:0	RELOAD	The RELOAD value is assigned to the CORET_CVR register when the counter counts to 0. Writing 0 to the CORET_RVR register will stop the counter at the next cycle. Writing a 0 to the CORET_RVR register causes the counter to stop on the next cycle, after which the counter value will remain at 0. After enabling the counter with an external reference clock, you must wait until the counter starts counting normally (i.e., until CORET_CVR changes to a non-zero value) before you can CORET_RVR to 0 to cause the counter to stop on the next cycle. Otherwise, the counter cannot start the first count.

How to calculate the RELOAD value:

The normal range for RELOAD 0x1-0x00FFFFFF. RELOAD can be assigned a value of 0, but it has no effect because the system timer interrupts and the COUNTFLAG bit only function the count value from 1 to 0. To generate a timer with a period of N count clock cycles, for example, to generate a CoreTim interrupt every 100 count clock cycles, the RELOAD value would need to be assigned to N-1. To generate a timer with a period of N count clock cycles, RELOAD needs to be assigned a value of N-1. For example, to generate a CoreTim interrupt every 100 count clock cycles, RELOAD needs to be assigned the value 99.

5.2.2.3 Current Value Register (CORET_CVR)

CORET_CVR contains the current value of the system timer. The CORET_CVR register and its bit descriptions are shown [Figure 5.5](#) and [Table 5.6](#).



Figure 5.5: System Timer Current Value Register

Table 5.6: System Timer Current Value Register Field Descriptions

classifier for honorific people	name (of a thing)	descriptive
31:24	-	Reservations.

23.0 User's Manual	CURRENT	<p>It indicates the value of the counter at the time it was read.</p> <p>A write to the CORET_CVR register clears both this register and the COUNTFLAG status bit, which causes the system timer to take the value in register CORET_RVR and assign it to CORET_CVR at the start of the next clock cycle. Note that a write to CORET_CVR does not cause the system timer's interrupt status bit to change.</p> <p>Reading CORET_CVR returns the value of the counter when the register was accessed.</p>
-----------------------	---------	--

5.2.2.4 Calibration register (CORET_CALIB)

The CORET_CALIB register describes the calibration function of the system timer. Its reset value is implementation-dependent: it is necessary to get the meaning of the CORET_CALIB bit information from the documentation provided by the device provider, as well as the checksum value TENMS in the CORET_CALIB register.

User's Manual

With TENMS as a calibration value, the software can multiply this value by a certain ratio to get other different counting periods, which of course must be within the value range of the counter; the CORET_CALIB register and its bit descriptions are shown Fig. 5.6 and Table 5.7.

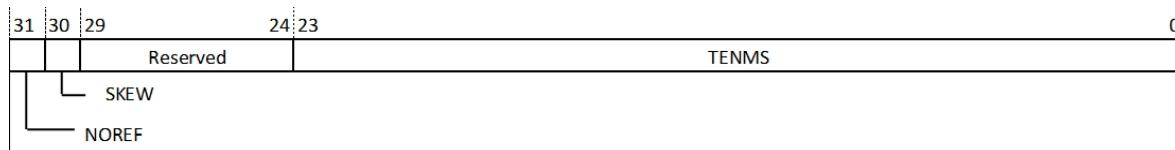


Figure 5.6: System Timer Calibration Registers

Table 5.7: System Timer Calibration Register Field Descriptions

classifier for honorific people	name (of a thing)	descriptive
31	NOREF	Indicates whether the device implements an external reference clock: 0 The device has an external reference clock; 1 The device has no external reference clock. this bit is 1, the CLKSOURCE bit of the CORET_CSR register is fixed to 1 and does not Can be rewritten.
30	SKEW	Indicates whether the 10ms calibration value is accurate: 0 10ms Calibration values are accurate; 1 The 10ms calibration value is in error due to the clock frequency.
29:24	-	Reservations.
23:0	TENMS	Used to indicate the backfill value corresponding to a 10ms time. Depending on the exact value of SKEW, it may represent the exact 10ms value or the value closest to 10ms. If the value of this field is 0, the calibration value is unknown. This may be because the reference clock is an unknown inputs or are dynamically changing.

Note: If CORET_RVR is set to 0, the CoreTim counter will stop working on the next turn, regardless of the state of the counter's enable bit.

The value of the SYST_CVR register is unknown at reset. Before enabling the CoreTim counter, software must first write the desired count value to the CORET_RVR register and then write an arbitrary value to CORET_CVR, the latter operation clearing CORET_CVR value to zero. This way, after enabling the counter, the counter can read the value in the CORET_RVR backfill register and count down from that value, thus avoiding counting from an arbitrary value.

5.2.3 procedure

1. Write the desired backfill value to the CORET_RVR register;
2. Write any value to the CORET_CVR register to clear it;

3. Operate the CORET_CSR register to enable the system timer.

5.3 Vector Interrupt Controller

5.3.1 brief

The Vector Interrupt Controller (VIC) is an IP unit tightly coupled to the E804 for efficient interrupt handling. The VIC supports up to 128 interrupt sources (IRQ[127:0]) each with its own software-programmable interrupt priority. The vector interrupt controller collects interrupt requests from different interrupt sources and arbitrates the interrupt requests according to the interrupt priority. The highest priority interrupt will gain interrupt control and issue an interrupt request to the processor. When the processor responds to the interrupt request, the processor returns an interrupt request response signal to the VIC; when the processor exits the interrupt service program (ISR) the processor returns an interrupt exit signal to the VIC.

The vector interrupt controller supports interrupt nesting. When the processor is processing an interrupt request while a higher priority interrupt request is coming in, the processor will interrupt the current interrupt service program processing and respond to the higher priority interrupt request. At the end of the processing of the higher priority interrupt request, the CPU returns to the interrupted interrupt service program to continue execution. The vector interrupt controller allows higher priority interrupt requests to preempt lower priority interrupt requests, but does not allow interrupts of the same level or lower priority to preempt, ensuring real-time interrupt response.

The system architecture of the Vector Interrupt Controller is shown Figure 5.7.

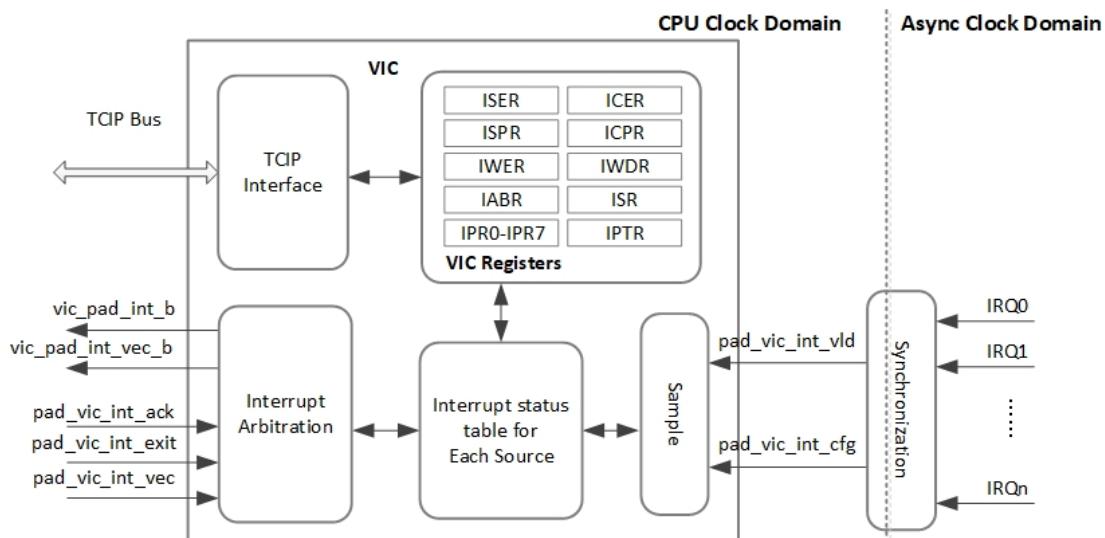


Figure 5.7: Vector Interrupt

Controller System Architecture Diagram The Vector Interrupt

Controller supports the following functions:

- The number of interrupts is hardware configurable and supports 4, 8, 16, 24, 32, 64, 96 and 128;
- The software configures the priority for each interrupt through the 8-bit register, and in E804 only the 8-bit register is used if the number of interrupt sources is less than or equal to 32.

The highest 2 valid bits characterize the priority, the rest of the bits are always 0. If the number interrupt sources is 64, then you can use 8 bits.

If the number of interrupt sources is 96 or 128, then the highest 4 bits of the 8 bits can be used to characterize the 16 priorities. The priorities are listed in descending order, with priority 0 being the highest priority and priority P ($P=4, 8, 16$) being the lowest priority;

User's Manual

- Supports both level and pulse interrupt source signals;
- Interrupts support dynamic adjustment of priority during processing, and inversion of interrupt priority is achieved at a small hardware cost by setting the priority threshold register;
- Supports interrupt nesting, where the CPU allows higher priority interrupt preemption during execution of the interrupt service program.

5.3.2 Register Definition

The VIC provides a set 32-bit registers, and the address space of each register is shown Table 5.8.

Table 5.8: Vector Interrupt Controller Register Definitions

address	name (of a thing)	typology	starting value	descriptive
0xE000E100	VIC_ISER0	Read/Write	0x00000000	Interrupt Enable Setting Register (Interrupts 0-31)
0xE000E104	VIC_ISER1	Read/Write	0x00000000	Interrupt Enable Setting Register (Interrupts 32-63)
0xE000E108	VIC_ISER2	Read/Write	0x00000000	Interrupt Enable Setting Register (Interruptions 64-95)
0xE000E10C	VIC_ISER3	Read/Write	0x00000000	Interrupt Enable Setting Register (Interruptions 96-127)
0xE000E110~0xE000E13F	-	-	-	Reservations.
0xE000E140	VIC_IWERO	Read/Write	0x00000000	Low Power Wakeup Setting Registers (Interrupts 0-31)
0xE000E144	VIC_IWER1	Read/Write	0x00000000	Low Power Wakeup Setting Registers (Interrupts 32-63)
0xE000E148	VIC_IWER2	Read/Write	0x00000000	Low Power Wakeup Setting Registers (Interruptions 64-95)

0xE000E14C User's Manual	VIC_IWER3	Read/W rite	0x00000000	Low Power Wakeup Setting Registers (Interruptions 96-127)
0xE000E150~0xE000E17F	-	-	-	Reservations.

Continued on next page

Table 5.8 - Continued from previous page

0xE000E180	VIC_ICER0	Read/W rite	0x00000000	Interrupt Enable Clear Register (Interrupts 0-31)
0xE000E184	VIC_ICER1	Read/W rite	0x00000000	Interrupt Enable Clear Register (Interrupts 32-63)
0xE000E188	VIC_ICER2	Read/W rite	0x00000000	Interrupt Enable Clear Register (Interruptions 64-95)
0xE000E18C	VIC_ICER3	Read/W rite	0x00000000	Interrupt Enable Clear Register (Interruptions 96-127)
0xE000E190~0xE000E1BF	-	-	-	Reservations.
0xE000E1C0	VIC_IWDR0	Read/W rite	0x00000000	Low Power Wake-Up Clear Registers (Interrupts 0-31)
0xE000E1C4	VIC_IWDR1	Read/W rite	0x00000000	Low Power Wake-Up Clear Registers (Interrupts 32-63)
0xE000E1C8	VIC_IWDR2	Read/W rite	0x00000000	Low Power Wake-Up Clear Registers (Interruptions 64-95)
0xE000E1CC	VIC_IWDR3	Read/W rite	0x00000000	Low Power Wake-Up Clear Registers (Interruptions 96-127)
0xE000E1D0~0xE000E1FF	-	-	-	Reservations.
0xE000E200	VIC_ISPR0	Read/W rite	0x00000000	Interrupt Wait Setting Register (Interrupts 0-31)
0xE000E204	VIC_ISPR1	Read/W rite	0x00000000	Interrupt wait setup register. (Interrupts 32-63)
0xE000E208	VIC_ISPR2	Read/W rite	0x00000000	Interrupt Wait Setting Register (Interruptions 64-95)
0xE000E20C	VIC_ISPR3	Read/W rite	0x00000000	Interrupt Wait Setting Register (Interruptions 96-127)
0xE000E210~0xE000E27F	-	-	-	Reservations.
0xE000E280	VIC_ICPR0	Read/W rite	0x00000000	Interrupt Wait Clear Register (Interrupts 0-31)
0xE000E284	VIC_ICPR1	Read/W rite	0x00000000	Interrupt Wait Clear Register (Interrupts 32-63)
0xE000E288	VIC_ICPR2	Read/W rite	0x00000000	Interrupt Wait Clear Register (Interruptions 64-95)
0xE000E28C	VIC_ICPR3	Read/W rite	0x00000000	Interrupt Wait Clear Register (Interruptions 96-127)
0xE000E290~0xE000E2FF	-	-	-	Reservations.
0xE000E300	VIC_IABR0	Read/W rite	0x00000000	Interrupt Response Status Register (Interrupts 0-31)

Continued on next page

Table 5.8 - Continued from previous page

0xE000E304	VIC_IABR1	Read/W rite	0x00000000	Interrupt Response Status Register (Interrupts 32-63)
0xE000E308	VIC_IABR2	Read/W rite	0x00000000	Interrupt Response Status Register (Interruptions 64-95)
0xE000E30C	VIC_IABR3	Read/W rite	0x00000000	Interrupt Response Status Register (Interruptions 96-127)
0xE000E310~0xE000E3FF	-	-	-	Reservations.
0xE000E400~0xE000E47C	VIC_IPR0- VIC_IPR31	Read/W rite	0x00000000	Interrupt priority setting register.
0xE000E480~0xE000EBFF	-	-	-	Reservations.
0xE000EC00	VIC_ISR	read- only (comput- ing)	0x00000000	Interrupt Status Register.
0xE000EC04	VIC_IPTR	Read/W rite	0x00000000	Interrupt Priority Threshold Register.
0xE000EC08	VIC_TSPEND	Read/W rite	0x00000000	Tspending Enable Setting Register.
0xE000EC0C	VIC_TSABR	Read/W rite	0x00000000	Tspending Response status register.
0xE000EC10	VIC_TSPR	Read/W rite	0x00000000	Tspending Waiting Setting Register.
0xE000EC14-0xE000ECFF	-	-	-	Reservations.

5.3.2.1 Interrupt enable setting register (VIC_ISER)

VIC_ISER is used to enable each interrupt and feedback the enable status of each interrupt. Figure 5.8 depicts the bit distribution of the VIC_ISER.

Figure 5.9 describes the bit definitions for VIC_ISER.



Figure 5.8: Interrupt Enable Setting Registers

Table 5.9: Interrupt Enable Setting Register Field Definitions

classifier for honorific people	name (of a thing)	descriptive

31:0 User's Manual	SETENA	Set use to read the enable state of one or more interrupts. Each bit corresponds to the same numbered interrupt source: Read operation: 0 The corresponding interrupt is not enabled; 1 The corresponding interrupt is enabled. Write operation: 0 Invalid; 1 Enable the corresponding interrupt.
-----------------------	--------	---

If an interrupt in the wait state is enabled, the vector interrupt controller activates the interrupt according to its priority. If an interrupt is not enabled, the vector interrupt controller does not activate the interrupt even if it is in the wait state.

5.3.2.2 Interrupt Low Power Wakeup Setting Register (VIC_IWER)

VIC_IWER is used to enable the low-power wake-up function of each interrupt, and feedback the enable status of low-power wake-up of each interrupt. [Figure 5.9](#) describes the bit distribution of VIC_IWER and [Table 5.10](#) describes the bit definition of VIC_IWER.



Figure 5.9: Interrupt Low Power Wakeup Setting Registers

Table 5.10: Low Power Wakeup Enable Setup Register Field Definitions

classifier for honorific people	name (of a thing)	descriptive
31:0	SETENA	<p>Set use to read the enable state of one or more interrupt low-power wakeups. Each bit corresponds to the same numbered interrupt source:</p> <p>Read operation:</p> <p>0: The low-power wake-up function corresponding to the interrupt is not enabled;</p> <p>1: The low-power wake-up function corresponding to the interrupt is enabled.</p> <p>Write operation:</p> <p>0: Invalid;</p> <p>1: Enable the low-power wake-up function of the corresponding interrupt.</p>

If the Wake-on-Low feature is enabled for an interrupt and the interrupt is in the wait state and the VIC generates a Wake-on-Low request. If the Wake-on-Low feature is not enabled for an interrupt, the VIC does not generate a Wake-on-Low request even if the interrupt is in the wait state.

Note: Interrupt enable and wake-up enable control the interrupt transaction and wake-up function respectively. When both are set, an interrupt in wait state generates both interrupt request and low-power wake-up request; when only one of them is enabled, only the corresponding function is activated; when neither is enabled, even if the interrupt is in wait state, no interrupt request or low-power wake-up request is generated.

5.3.2.3 Interrupt enable clear register (VIC_ICER)

VIC_ICER is used to clear the enable of each interrupt and feedback the enable status of each interrupt. [Figure 5.10](#) describes the bit distribution of VIC_ICER and [Table 5.11](#) describes the bit definition of VIC_ICER.



Figure 5.10: Interrupt Enable Clear Registers

Table 5.11: Interrupt Enable Clear Register Field Descriptions

classifier for honorific people	name (of a thing)	descriptive
31:0	CLRENA	<p>Clear Use to read the enable state of one or more interrupts. Each bit corresponds to the same numbered interrupt source:</p> <p>Read operation:</p> <p>0: The corresponding interrupt is not enabled;</p> <p>1: The corresponding interrupt is enabled.</p> <p>Write operation:</p> <p>0: Invalid;</p> <p>1: Clear the enable of the corresponding interrupt.</p>

5.3.2.4 Interrupt low-power wake-up clear register (VIC_IWDR)

VIC_IWDR is used to clear the low-power wake-up enable for each interrupt and feeds the enable status of low-power wake-up for each interrupt. [Figure 5.11](#).

describes the bit distribution of VIC_IWDR and [Table 5.12](#) describes the bit definitions of VIC_IWDR.



Figure 5.11: Interrupt Low Power Wake-Up Clear Registers

Table 5.12: Interrupt Low Power Wake-Up Clear Register Field Descriptions

classifier for honorific people	name (of a thing)	descriptive
31:0	CLRENA	<p>Clear Use to read the enable state of one or more interrupt low-power wakeups. Each bit corresponds to the same numbered interrupt source:</p> <p>Read operation:</p> <p>0: The low-power wake-up function corresponding to the interrupt is not enabled;</p> <p>1: The low-power wake-up function corresponding to the interrupt is enabled.</p> <p>Write operation:</p> <p>0: Invalid;</p> <p>1: Clear to enable the low-power wake-up function of the corresponding interrupt.</p>

5.3.2.5 Interrupt wait setup register (VIC_ISPR)

User's Manual characterization sets each interrupt to a wait state. Describing the bit distribution of VIC_ISPR,

Figure 5.12 depicts the VIC_ISPR

Table 5.13 describes the bit definitions of VIC_ISPR. Table 5.13 describes the bit definitions for VIC_ISPR.

31	0
SETPEND	

Figure 5.12: Interrupt Wait Setting Registers

Table 5.13: Interrupt Wait Setup Register Field Descriptions

classifier for honorific people	name (of a thing)	descriptive
31:0	SETPEND	<p>Changes one or more interrupts to the wait state. Each bit corresponds to the same numbered interrupt source:</p> <p>Read operation:</p> <p>0: The corresponding interrupt is not in the wait state.</p> <p>1: The corresponding interrupt is in wait state.</p> <p>Write operation:</p> <p>0: Invalid.</p> <p>1: Change the corresponding interrupt to the wait state.</p>

5.3.2.6 Interrupt Wait Clear Register (VIC_ICPR)

VIC_ICPR characterizes the wait state for clearing each interrupt. Figure 5.13 depicts the bit distribution of VIC_ICPR, and Table 5.14 describes the

Bit Definitions for VIC_ICPR .



Figure 5.13: Interrupt Wait Clear Registers

Table 5.14: Interrupt Wait Clear Registers

classifier for honorific people	name (of a thing)	descriptive
31:0	CLRPEND	<p>Clears the wait state of one or more interrupts. Each bit corresponds to an interrupt source of the same number:</p> <p>Read operation:</p> <p>0: The corresponding interrupt is in the not-waiting state;</p> <p>1: The corresponding interrupt is in wait state.</p> <p>Write operation:</p> <p>0: Invalid;</p> <p>1: Clear the wait state of the corresponding interrupt.</p>

5.3.2.7 Interrupt Response Status Register (VIC_IABR)

VIC_IABR is used to indicate the current active state of each interrupt, it is a register for software to query, in addition, the software can clear the active state of all interrupts to 0 when initializing the VIC. Figure 5.14 describes the bit distribution of VIC_IABR, and Table 5.15 describes the bit definitions of VIC_IABR.



Figure 5.14: Interrupt Response Status Registers

Table 5.15: Interrupt Response Status Register Field Descriptions

classifier for honorific people	name (of a thing)	descriptive
31:0	Active	<p>Query bits indicating whether the interrupt source has been responded to by the CPU but has not yet been processed. Each bit corresponds to an interrupt source of the same number.</p> <p>Read operation:</p> <p>0: Not responded to by the CPU;</p> <p>1: Has been responded to by the CPU but has not been processed yet.</p> <p>Write operation:</p> <p>0: Clears the Active state of the interrupt (software must not write a 1 to this register or it will result in an unavailable interrupt).</p> <p>(Expected errors)</p>

5.3.2.8 Interrupt Priority Setting Register (VIC_IPR0 - VIC_IPR31)

Each interrupt priority setting register provides priority settings for 4 interrupt sources. Each interrupt source setting area corresponds to the corresponding interrupt source, as defined by the application. For implementations with hardware support for 128 interrupt sources, the registers range from IPR0 to IPR31 as shown in [Figure 5.15](#), and [Figure 5.16](#) depicts the Interrupt Priority Setting Registers.

The vector interrupt controller selects the interrupt priority based on the priority number, the smaller the priority number, the higher the priority. If the priority numbers are the same, the priority order is determined based on the interrupt source number, and the smaller the number, the higher the priority.

IPR0	31	24	23	16	15	8	7	0
	PRI_3		PRI_2		PRI_1		PRI_0	
IPR1	31	24	23	16	15	8	7	0
	PRO_4n+3		PRO_4n+2		PRO_4n+1		PRO_4n	
IPR2	31	24	23	16	15	8	7	0
	PRO_127		PRO_126		PRO_125		PRO_124	

Figure 5.15: Overall Distribution of Interrupt Priority Setting Registers

31	30	29	24	23	22	21	16	15	14	13	8	7	6	5	0
	-			-			-			-			-		
	PRI_N+3				PRI_N+2				PRI_N+1				PRI_N		

Figure 5.16: Interrupt Priority Setting Registers

[Table 5.16](#) and [Table 5.17](#) describe VIC_IPRn bit definitions. In this table, N = 4n, n is the VIC_IPRn register number. If VIC_IPR2, n is 2 then N is 8.

Table 5.16: Interrupt Priority Setting Register Field Descriptions

Priority order	classifier for honorific people	name (of a thing)	descriptive
2 bits 4 priorities (The number of interrupts is less than or equal to 32)	31:30	PRI_N+3	The interrupt number is the priority of N+3, the smaller the value the higher the priority. The higher.
	29:24	-	Reservations.
	23:22	PRI_N+2	The interrupt number is the priority of N+2, the smaller the value the higher the priority. The higher.
	21:16	-	Reservations.
	15:14	PRI_N+1	The interrupt number is the priority of N+1, the smaller the value the higher the priority. The higher.
	13:8	-	Reservations.
	7:6	PRI_N	The interrupt number is the priority of N. The smaller the value, the higher the priority. High.
3 bits 8 priority levels (Number of interrupts is 64) (a)	5:0	-	Reservations.
	31:29	PRI_N+3	The interrupt number is the priority of N+3, the smaller the value the higher the priority. The higher.
	28:24	-	Reservations.
	23:21	PRI_N+2	The interrupt number is the priority of N+2, the smaller the value the higher the priority. The higher.
	20:16	-	Reservations.
	15:13	PRI_N+1	The interrupt number is the priority of N+1, the smaller the value the higher the priority. The higher.
	12:8	-	Reservations.
	7:5	PRI_N	The interrupt number is the priority of N. The smaller the value, the higher the priority. High.
	4:0	-	Reservations.

Table 5.17: Interrupt Priority Setting Register Field Descriptions

Priority order	classifier for honorific people	name (of a thing)	descriptive
4 bits 16 priority levels (96 or 128 interrupts)	31:28	PRI_N+3	The interrupt number is the priority of N+3, the smaller the value the higher the priority. The higher.
	27:24	-	Reservations.
	23:20	PRI_N+2	The interrupt number is the priority of N+2, the smaller the value the higher the priority. The higher.
	19:16	-	Reservations.
	15:12	PRI_N+1	The interrupt number is the priority of N+1, the smaller the value the higher the priority. The higher.
	11:8	-	Reservations.
	7:4	PRI_N	The interrupt number is the priority of N. The smaller the value, the higher the priority. High.
	3:0	-	Reservations.

5.3.2.9 Interrupt Status Register (VIC_ISR)

VIC_ISR indicates the interrupt vector number currently being processed by the CPU and the highest priority interrupt vector number that is waiting, and is a read-only register for software query. Figure 5.17 depicts the bit distribution of VIC_ISR and Table 5.18 depicts the bit definitions of VIC_ISR.

31	21:20	12	11	9:8	0
Reserved		VECTPENDING	Reserved	VECTACTIVE	

Figure 5.17: Interrupt Status Registers

Table 5.18: Interrupt Status Register Field Descriptions

classifier for honorific people	name (of a thing)	descriptive
31:21	Reserved	Reservations.
20:12	VECTPENDING	Indicates the interrupt vector number with the highest priority that is currently in the wait state.

User's Manual	Reserved	Reservations.
8:0	VECTACTIVE	Indicates the interrupt vector number that the CPU is currently processing.

5.3.2.10 Interrupt Priority Threshold Register (VIC_IPTR)

VIC_IPTR defines the priority threshold at which an interrupt request currently in the wait state can initiate interrupt preemption. The priority of an interrupt request in the waiting state must be higher than the priority threshold defined by VIC_IPTR in order to initiate an interrupt preemption request. Figure 5.18 depicts the bit distribution of the VIC_IPTR and Table 5.19 depicts the bit definitions of the VIC_IPTR.

31 30 EN	17 16 Reserved	16 15 VECTTHRESHOLD	8 7 PRIOTHRESHOLD	0
-------------	-------------------	------------------------	----------------------	---

Figure 5.18: Interrupt Priority Threshold Registers

Table 5.19: Interrupt Priority Threshold Register Field Descriptions

classifier for honorific people	name (of a thing)	descriptive
31	EN	Interrupt priority threshold valid bit: 0: Interrupt preemption does not require a priority above the threshold; 1: Interrupt preemption requires a priority above the threshold.
30:17	Reserved	Reservations.
16:8	VECTTHRESHOLD	Indicates the interrupt vector number corresponding to the priority threshold. When the VIC finds CPU interrupt service program corresponding to VECTTHRESHOLD On exit, hardware automatically clears the interrupt priority threshold valid bit.
7:0	PRIOTHRESHOLD	Indicates the priority threshold for interrupt preemption. Notes. The number of priorities in the E804 based on the number of interrupts configured can be characterized by setting [7:6], [7:5] or [7:4] bits to the priority threshold.

5.3.2.11 Tspend Interrupt enable setting register (VIC_TSPEND)

VIC_TSPEND is used to enable the tspend interrupt and feed back the enable status of the tspend interrupt. Figure 5.19 depicts the VIC_TSPEND

bit distribution, Table 5.20 describes the bit definitions for VIC_TSPEND.

classifier for honorific people	name (of a thing)	descriptive
		Reserved
0:0	SETENA	<p>Set Enable to read the enable state of the tspend interrupt: Read operation: 0: Corresponds to tspend interrupt not enabled; 1: Corresponds to Table 5.20 tspend interrupt enabled. Write operation: 0: Invalid; 1: Enable tspend interrupt.</p>
www.xrvm.cn		56 © Copyright © Hangzhou Zhongtian Microsystems Co.

5.3.2.12 Tspend Interrupt Response Status Register (VIC_TSABR)

VIC_TSABR is used to indicate the current Active state of tspend interrupts, it is a register for software to query, in addition, the software can clear the Active state of all tspend interrupts to 0 when initializing the VIC. [Figure 5.20](#) describes the bit distribution of VIC_TSABR, and [Table 5.21](#) describes the bit definition of VIC_TSABR.



[Figure 5.20: Tspend Interrupt Response](#)

classifier for honorific people	name (of a thing)	Status Register Table 5.21: Tspend Interrupt descriptive
0	Active	<p>Response Status Register Field</p> <p>Query bit indicating whether the tspend interrupt has been responded to by the CPU but has not yet been processed. Only the lowest bit of this register is valid.</p> <p>Descriptions</p> <p>Read operation 0 was not responded to by the CPU;</p> <p>1 The CPU has responded but has not finished processing.</p> <p>Write operation 0 Clears the Active state of the tspend interrupt.</p> <p>(Software must not write 1 to this register or it will cause unanticipated errors)</p>

5.3.2.13 Tspend Interrupt Priority Setting Register (VIC_TSPR)

VIC_TSPR provides the priority setting for the tspend interrupt, the priority of the tspend interrupt needs to be set to the lowest. [Figure 5.21](#) depicts the

VIC_TSPR bit distribution, [Table 5.22](#) describes the bit definitions of VIC_TSPR.

31	8	7	0	
classifier for honorific people	Reserved	name (of a thing)	descriptive	PRI
Figure 5.21: Tspend Interrupt Priority				
2 bits 4 priorities (number of interrupts less than or equal to 32)	31:8	-	Reservations.	
	7:6	PRI	It is recommended that it be set to the lowest priority 2'b11.	
	5:0	-	Reservations.	
3 bits 8 priority levels (64 interrupts)	31:8	PRI	Reservations.	
	7:5	PRI	It is recommended that it be set to lowest priority 3'b111.	
	5:0	-	Reservations.	
4 bits 16 priority levels (96 or 128 interrupts)	31:8	-	Reservations.	
	7:4	PRI	It is recommended that it be set to the lowest priority 4'b1111.	
www.xrvm.cn			© Copyright © Hangzhou Zhongtian Microsystems Co.	
	5:0	-	Reservations.	

5.3.3 interrupt handling mechanism

The vector interrupt controller supports both level and pulse interrupts.

For level interrupts, the vector interrupt controller samples the high level of the interrupt valid signal and sets the corresponding interrupt to enter the wait state, and then requests the CPU to respond. Level interrupts require the interrupt source valid signal of the peripheral to be cleared in the interrupt service program, otherwise the interrupt controller will re-initiate an interrupt request to the CPU when the interrupt exits. The peripheral device can always set the interrupt signal until it no longer needs to be handled by the interrupt handler according to this feature.

For pulsed interrupts, also known as edge interrupts, the vector interrupt controller samples the rising edge of the interrupt's active signal and then sets the corresponding interrupt into a wait state, followed by an interrupt request to the CPU. To ensure that the vector interrupt controller detects the pulsed interrupt, the periphery needs to hold the interrupt signal for at least one CPU clock cycle. Before the CPU responds to the pulse interrupt request, if the pulse interrupt source initiates multiple interrupt requests to the vector interrupt controller, the vector interrupt controller will only record the interrupt request once; after the CPU responds to the pulse interrupt request, if the pulse interrupt source initiates another request to the interrupt controller, the vector interrupt controller will trigger the corresponding interrupt to enter into the wait state again, and the interrupt request in wait state can only be responded to by the CPU again after the last interrupt has exited. The interrupt request in the waiting state can only be responded to by the CPU again after the last interrupt exits.

In addition, the vector interrupt controller supports software interrupts. The software can trigger the interrupt to enter the wait state and send an interrupt request to the CPU by setting the corresponding interrupt wait state bit high in the interrupt setup wait register (VIC_ISPR).

When the processor responds to an interrupt request, the vector interrupt controller automatically clears the wait status bit for the corresponding interrupt. The wait status bit of the corresponding interrupt can also be cleared by setting the interrupt clear wait register (VIC_ICPR). For level interrupts, the wait status bit cannot be cleared by setting the VIC_ICPR register if the interrupt valid signal is continuously high.

5.3.3.1 interrupt status bit

The VIC provides 2 bit status bits for each interrupt source, respectively:

- Pending: characterizes that the interrupt is in waiting state, that is, the interrupt request is waiting for CPU to respond.
 - 0: Characterizes that the interrupt is not yet in the wait state;
 - 1: Characterize that the interrupt is already in the wait state.
- Active: characterizes that the interrupt request has been responded to by the CPU, but has not yet been processed.
 - 0: Indicates that the interrupt request has not been responded to by the CPU;
 - 1: Characterizes that the interrupt request has been responded to by the CPU, but has not yet been processed.

The setting condition of the Pending bit:

High level interrupt source, interrupt source active signal is high, Active is low or the CPU is exiting this interrupt

service program;

2. Pulse interrupt source, rising edge active;

3. The software sets the

ISPR. clear condition of

the Pending bit:

1. The CPU responds to the interrupt request;

2. The software clears the ICPR.

The Active bit is set when the CPU responds to an interrupt request.

Active bit clear condition: CPU exits this interrupt service program.

Note: For level interrupt source, since the interrupt service program is required to pull down the level interrupt source request, it is only necessary to sample the active signal of the interrupt source and set the Pending bit when Active is low or when the interrupt service program is exited; for pulse interrupt source, since the request signal will be pulled down automatically, it is necessary to sample the rising edge of the pulse interrupt signal and set the Pending bit in real time.

5.3.3.2 interrupt priority

The VIC provides priority settings for each interrupt source through the Interrupt Priority Setting Registers (IPR0-IPR31) for interrupts less than or equal to 32.

If the number of interrupts is less than or equal to 64, the hardware can provide 4 priorities; if the number of interrupts is 96, the hardware can provide 8 priorities. or 128, the hardware can provide 16 priority levels. The lower the priority number, the higher the priority, refer to [Interrupt Priority Setting Register](#) for details (*VIC_IPR0 - VIC_IPR31*) . When the Priority register is not set by software, all interrupt source priorities default to the highest priority - the 0.

When multiple interrupts are in Pending state, VIC arbitrates the interrupt request with the highest priority according to the priority of each interrupt and submits it to CPU for processing. For example, if two interrupt requests, IRQ0 and IRQ1, are in Pending state at the same time, and if the priority number of IRQ1 is smaller than IRQ0, i.e., the priority of IRQ1 is higher than that of IRQ0, IRQ0 is submitted to the CPU for processing first.

When multiple Pending interrupts have the same priority number, the order of interrupt submission is determined according to the interrupt number, and the interrupt with a smaller interrupt number is prioritized for CPU processing. For example, two interrupt requests, IRQ0 and IRQ1, have the same interrupt priority, and the interrupt source number of IRQ0 is smaller than that of IRQ1, so IRQ0 is given priority to be submitted to the CPU for processing.

5.3.3.3 interrupt vector number (computing)

The interrupt vector number is the location of the interrupt request in the exception vector table. [Table 5.23](#) shows the exception vector table of the E804. Vectors 0-30 at the beginning are used as the processor's internal identification vectors; vector 31 is reserved; and the vector numbers from 32 are reserved for external interrupt requests.且 Each interrupt source corresponds to an interrupt vector number.

Table 5.23: Interrupt Vector Number Descriptions

vector number	Vector offset (hexadecimal)	vector allocation
0	000	Reboot Exception.
1	004	Unaligned access exception.
2	008	Access Error Exception.
3	00C	Except for the 0 exception.
4	010	Illegal command exception.
5	014	Privilege violation anomalies.
6	018	Tracking anomalies.
7	01C	Breakpoint exception.
8	020	Unrecoverable error exception.
9	024	IDLY exception.
10	028	Normal interrupt (Auto Vector)
11-15	02C - 03C	Reservations.
16 - 19	040 - 04C	Trap instruction exception (TRAP # 0 - 3)
20-21	050-054	Reservations.
22	058	Tspend interrupt.
23-29	05C-074	Reservations.
30	078	Floating point anomaly.
31	07C	Reservations.
32	080	IRQ0.
33	084	IRQ1.
.....
32+n	0x80+4n	IRQn.

5.3.3.4 interrupt processing

The interrupt handling process can be carried out in the following steps:

- Synchronization of interrupt source request: the external device generates an interrupt source request, and the system completes the synchronization operation of the asynchronous interrupt source request to the CPUCLK clock domain by setting high pad_vic_int_vld;
- Sampling of interrupt source request: VIC samples the interrupt source request according to the type of interrupt source; when a valid interrupt request is sampled, the Pending status bit is set, triggering the corresponding interrupt to enter the waiting state;
- Interrupt request initiation: Among all the interrupts in the waiting state, an interrupt request is initiated to the CPU after priority arbitration;
- Interrupt response: CPU responds to the interrupt when the instruction retires, returns the interrupt response signal to VIC, updates PSR and PC to EPSR and EPC at the same time, updates PSR.VEC with the interrupt vector number of the interrupt being responded to, clears PSR.EE, and finally

fetches the exception entry address; VIC clears the Pending status bit of the corresponding interrupt according to the interrupt response signal, and sets its Active status bit;

- Interrupt site saving: first save the interrupt control register site {EPSR, EPC} and turn on PSR.EE and PSR.IE to enable interrupt nesting; then save the general-purpose register site;

User's Manual

- Interrupt Transaction: The CPU starts to process the interrupt transaction. For level interrupts, the interrupt source signal needs to be cleared, otherwise the interrupt will be reentered when the interrupt exits;
- Interrupt site restoration and interrupt exit: first restore the general-purpose register site; then restore the interrupt control register site {EPSR, EPC}, restore EPC and EPSR to PC and PSR, and exit the interrupt service program; VIC receives the interrupt exit signal and clears the Active status bit.

The saving of the interrupt site can be accomplished by executing the NIE and IPUSH instructions at the beginning of the interrupt service program, and the restoration and exit of the interrupt site can be accomplished by executing the IPOP and NIR instructions at the end of the interrupt service program.

The synchronization of the interrupt source request is done by the system and is not implemented internally by the VIC. Figure 5.22 shows an example of the synchronization of the pad_vic_int_vld signal, where the interrupt source request signal irq_n is synchronized to the CPU clock domain through the registers of the two CPUCLK levels. In addition, the interrupt source type configuration signal pad_vic_int_cfg is a fixed value for a given interrupt source, 0 for a level interrupt source and 1 for a pulsed interrupt source, and therefore does not need to be synchronized.

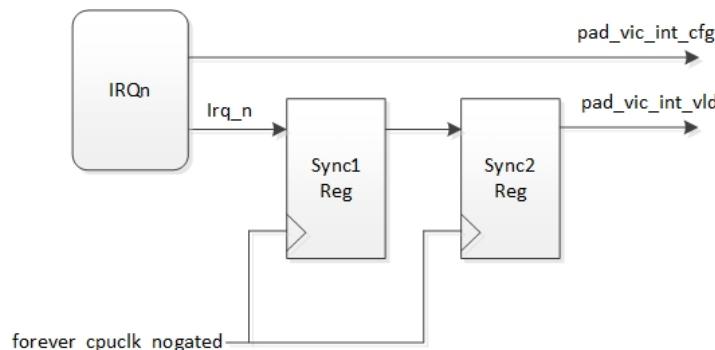


Figure 5.22: Interrupt Source Request Synchronization Circuit Example

5.3.3.5 interrupt nesting

The VIC supports interrupt nesting, which allows higher priority interrupts to be preempted during interrupt processing, thus improving real-time interrupt response. When multi-level interrupt nesting exists, the priority of the preempted interrupt may need to be changed in certain scenarios, such as when the response time of a low-priority interrupt reaches the maximum limit. At this time, the software can set the interrupt priority threshold register to increase the priority conditions for interrupt preemption, so that the preempted low priority interrupt can be responded to in a timely manner.

Interrupt Nested Priority Conditions

The priority conditions for interrupt preemption can be categorized into two types as follows:

- When the interrupt priority threshold is not enabled, the priority of interrupt preemption must be higher than the priority of the interrupt currently being processed by the CPU; the same priority level cannot be preempted;
- When Interrupt Priority Threshold is enabled, the priority of interrupt preemption should not only be higher than the current interrupt priority being processed by the CPU, but should be higher than the

The VIC supports dynamic adjustment of interrupt priority. When the priority of a preempted interrupt needs to be adjusted up or down, the interrupt priority threshold register is set at the same time as the interrupt priority setting register.

User's Manual

Figure 5.23 gives an example of interrupt preemption. The interrupt priority is set to IRQ0<IRQ1<IRQ2<IRQ3, and the order of interrupt source request generation is IRQ0>IRQ1>IRQ2>IRQ3. The CPU first responds to the IRQ0, and during the execution of the IRQ0 interrupt service program, a higher-priority IRQ1 comes in, and IRQ0 is preempted, and the CPU starts to execute the IRQ1 interrupt service program. Therefore, IRQ0 is preempted and the CPU starts to execute the interrupt service program of IRQ1. Similarly, IRQ2 preempts IRQ1 and sets the interrupt priority threshold registers (IPTR.VECTTHRESHOLD=IRQ0, IPTR.PRIOTHRESHOLD=0, IPTR.EN=1) When IRQ3 arrives, although the priority is higher than IRQ2, IRQ3 does not have a priority higher than the interrupt characterized by the priority threshold set in IPTR, so IRQ3 cannot seize IRQ2. IRQ3 waits until the hardware automatically clears IPTR.EN at the end of the interrupt service program execution of IRQ0 to get a response from the CPU.

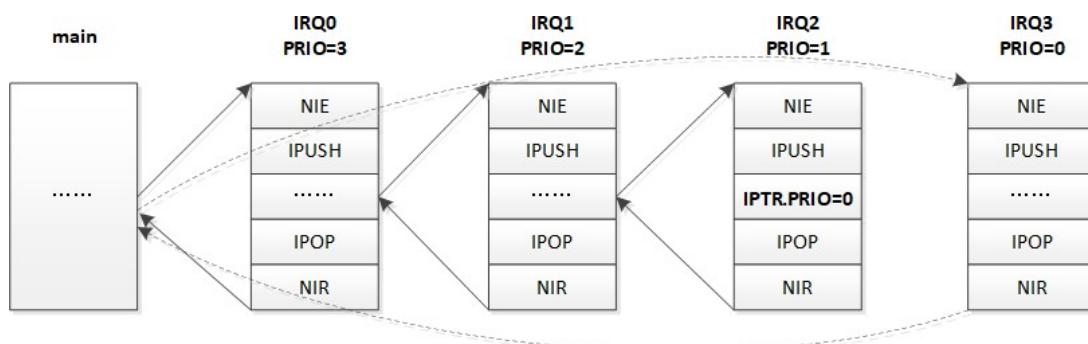


Figure 5.23: Interrupt Nested Priority Example

Interrupt Nested Timing Conditions

Interrupt preemption also needs to determine the current stage of interrupt response under the condition of satisfying the priority. The process of interrupt response refers to [the interrupt handling process](#), and nesting-related are divided into the following main phases:

- (1) EPSR, EPC, PSR updates and exception entry address reads
- (2) NIE Instruction
- (3) IPUSH command
- (4) Interruptions
- (5) IPOPO command
- (6) NIR command.

To ensure that the interrupt nested site is preserved and restored, the CPU cannot be interrupted by interrupts during the following phases:

- The process of updating the EPSR, EPC, PSR, and getting the exception entry address after the interrupt response;
- The NIE instruction execution at includes instruction retirement;
- The PSR.IC bit is turned off and the IPUSH and IPOPO instructions are executed excluding instruction retirement;
- The NIR instruction execution process does not include instruction retirement.

The CPU can safely and reliably respond to new interrupts in the following stages:

During the execution of a normal program, before the interrupt response;

- When the IPUSH and IPOP commands are retired;

User's Manual

- The PSR.IC bit is turned on during the execution of the IPUSH and IPOP instructions;
- When the NIR command is retired;
- Interrupts transaction processing in progress.

When the PSR.IC bit is turned on, the IPUSH and IPOP instructions can respond to an interrupt during execution, so it is necessary to re-execute the instruction when the interrupt returns. For the case that the IPUSH or IPOP instruction responds to an interrupt during retirement, the next instruction of IPUSH/IPOP is executed directly after the interrupt exits, and there is no need to re-execute the IPUSH/IPOP instruction. The NIR instruction cannot be interrupted by an interrupt during execution but can respond to an interrupt when it retires, and if an interrupt hits the NIR instruction, the CPU, when retiring the NIR instruction, directly presses the return address of NIR into the stack. If the interrupt hits the NIR instruction, the CPU will directly press the return address of NIR into the stack and return the target address of NIR when the interrupt exits.

Figure 5.24 gives the IRQ0/IRQ1/IRQ2/IRQ3 interrupt nesting process. After IRQ0 is responded to by the CPU, a higher priority IRQ1 is generated, which responds to IRQ1 when the IPUSH instruction is executed when PSR.IC is turned on. IRQ2 is generated when IRQ1 is processing an interrupt transaction, and therefore can be responded to immediately by the CPU. IRQ3 is generated when IRQ2 is executing the NIR instruction, and it responds to IRQ3 when the NIR instruction retires. When IRQ3 finishes processing and exits the interrupt service program, it returns directly to the point where IRQ1 was interrupted by IRQ2. When IRQ1 returns to IRQ0, the IPUSH instruction needs to be re-executed.

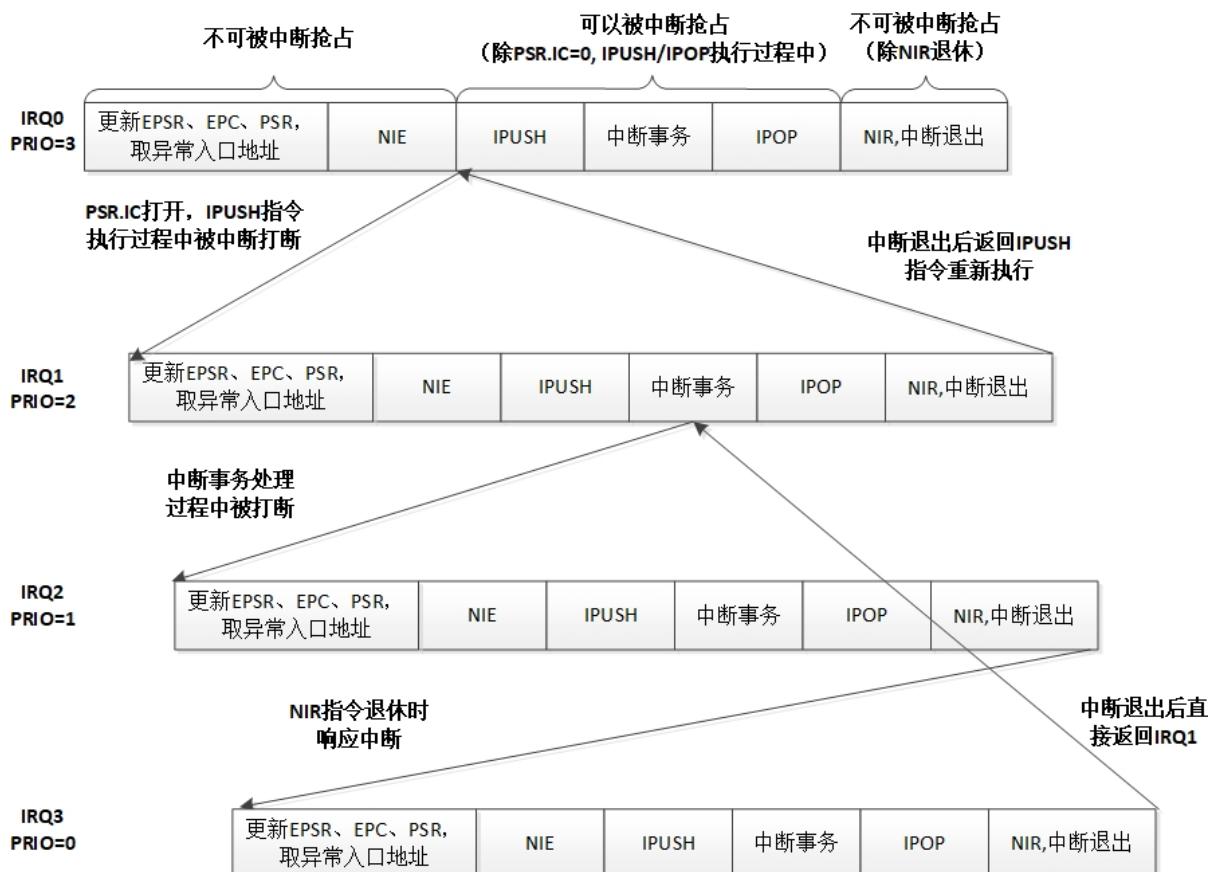


Figure 5.24: Example of Interrupt Nested Timing Conditions

5.3.4 Tspend Interrupt

Tspend interrupt controller provides, an interrupt does not occupy the external interrupt vector number (32-159), Tspend interrupt vector number 22, tspend interrupt function and settings with other interrupts are basically the same, tspend interrupt and other interrupts are different:

1. Tspend interrupt, does not need external device to generate interrupt source request, no pending status bit, enter wait state when software set VIC_TSPEND enable.
2. Tspend When interrupts and other interrupts have the same priority, the interrupt priority is not judged by the size of the interrupt number, and when interrupts have the same priority, Tspend is not judged by the size of the interrupt number.
The interrupt priority is lowest; wait until no other interrupt of the same level is pending to process the tspend interrupt.

5.3.5 procedure

To ensure that the VIC is able to generate interrupt requests with the expected priority, the user needs to set the interrupt priority and enable the response interrupt in advance, as follows:

1. First set VIC_IPR0-31 to configure the appropriate priority for each interrupt;
2. Then set VIC_ISER to enable the corresponding interrupt.

In addition, the VIC supports software setting of VIC_ISPR to generate interrupt requests. Again, the interrupt priority and interrupt enable bits must be configured as described above before VIC_ISPR can be set.

Notes. The CPU needs to enable PSR.IE and PSR.EE before responding to a VIC-triggered interrupt request to turn on the CPU's interrupt response enable, otherwise the CPU cannot respond to the interrupt.

Before the VIC samples an external interrupt source to generate an interrupt low-power wake-up, the VIC_IWER must be set to set the corresponding low-power wake-up enable bit high, otherwise a low-power wake-up request cannot be generated.

Note: For an external interrupt source request, only the low-power wake-up function of the corresponding interrupt is enabled (corresponding to VIC_IWER to generate a low-power wake-up request, independent of the interrupt's own interrupt enable (corresponding to VIC_ISER))

5.3.6 interface signal

The interface signals of the vector interrupt controller can be divided into three groups, which are:

- Vector Interrupt Controller Interfaces with Interrupt Source: the VIC receives and samples the interrupt request signal from the interrupt source;

Interface between Vector Interrupt Controller and TCIP: TCIP provides a set of interfaces to read/write VIC registers to accomplish the reading/writing of VIC related control registers, such as enabling interrupts, setting interrupt priority, and querying the current interrupt being processed;

- The vector interrupt controller interfaces with the processor core E804: the VIC initiates an interrupt request and the corresponding interrupt vector number to the E804, which returns the VIC interrupt response and interrupt return indication signals as well as the interrupt vector number currently being processed.

Table 5.24: Vector Interrupt Controller Interface Signals

signal name	I/O	Reset	define
Tightly coupled IP bus interface signals:			
tcipif_vic_sel	I	0	Check the signal: Indicates that the Vector Interrupt Controller is selected and data is transferred. 1: Check the VIC; 0: VIC is unchecked.
tcipif_vic_addr[15:0]	I	-	Address bus: 16-bit address bus (intercepts the lower 16 bits of the 32-bit address bus), refers to the Show access address.
tcipif_vic_write	I	0	Read and write representation signals: Indicates whether the current TCIP access is reading data or writing data: 1: Write a visit; 0: Read access.
tcipif_vic_wdata[31:0]	I	-	Write data bus: 32-bit write data bus.
vic_tcipif_rdata[31:0]	O	-	Read data bus: 32-bit read data bus.
vic_tcipif_cmplt	O	0	Transmission completion indication signal: When valid, indicates that the current transmission is complete.
The processor interrupts the handshake signal:			
vic_pad_int_b	O	1	Interrupt request signal: A low level indicates a normal interrupt request.
vic_pad_intraw_b	O	1	Interrupt wake-up request signal: A low level indicates a low-power wake-up request.
vic_pad_int_vec_b[7:0]	O	-	Interrupt vector serial number signal: Provides the vector number for core to perform interrupt processing.
pad_vic_int_ack	I	0	Interrupt response signal: Indicates that the CPU has responded to the current interrupt request.
pad_vic_int_exit	I	0	Interrupt service program exit signal: Instructs the CPU to exit the interrupt service program.
pad_vic_int_vec[7:0]	I	-	Indicates the interrupt vector number being processed by the CPU.
pad_vic_ack_vec[7:0]	I	-	Indicates the interrupt vector number to which the CPU responds.

Users Manual

Interrupt source signal:			
pad_vic_int_cfg[127:0]	I	-	Interrupt source type configuration signal: 0: Level interrupt source; 1: Pulse interrupt source.
pad_vic_int_vld[127:0]	I	0	Interrupt valid signal: Active High.

Table 5.25: Vector Interrupt Controller Interface Signals

signal name	I/O	Reset	define
Clock Signal:			
forever_cpuclk	I	-	The operating clock of the VIC: The registers in the vector interrupt controller that are not related to low-power wakeup are all working In this clock, the clock signal can be turned off in low-power mode.
forever_cpuclk_nogated	I	-	VIC interrupts the clock of the sampling circuit: The registers in the vector interrupt controller related to the low-power wakeup are all working on this clock. This clock signal cannot be turned off in low power mode.
Reset signal:			
cpurst_b	I	-	Vector interrupt controller reset signal: Initializes the vector interrupt controller internally when low. Vector Interrupt Controller The brake uses an asynchronous reset method.
Other signals:			
pad_yyy_gate_clk_en_b	I	-	Gated clock enable signal: The gated clock of the VIC's internal module is only valid when this signal is valid. When this signal is not used, you need to connect 1.
pad_yyy_test_mode	I	-	Enter test mode: Enables the VIC to enter test mode, at which time the VIC clock is the test clock (pad_had_jtg_tclk) Only if the VIC is in test mode and the processor input signal pad_yy_scan_enable is active to test through the scan chain. When this signal is not used, you need to connect 0.

When the CPU does not have an integrated vector interrupt controller, the external interrupt controller arbitrates the interrupt source and generates an interrupt request to be sent to the CPU, which synchronizes the externally input interrupt request signals and passes them on to the core for processing. In addition, the CPU also needs to synchronize the low-power wake-up signal generated by the external IP.

Table 5.26: Interface Signals without Vector Interrupt Controller Configuration

signal name	I/O	Reset	define
Tightly coupled IP bus interface signals:			
pad_cpu_int_b	I	1	Interrupt request signal: A low level indicates a normal interrupt request.
pad_cpu_intraw_b	I	1	Low-power wake-up request signal: A low level indicates a low-power wake-up request.
pad_cpu_int_vec_b[7:0]	I	-	Interrupt vector number: Indicates the interrupt vector number corresponding to the current interrupt request.
Clock Signal:			
forever_cpublk_nogated	I	-	VIC Synchronized Circuit Clock: For synchronization of interrupt requests and low-power wake-up signals.
clk_en	I	-	System clock synchronization enable signal.

```
// set ee and ie bits in psr, cpu responds to interrupts
```

5.3.7 Interrupt Setting Example

```
psrset ee, ie
```

```
//Set the interrupt enable bits for interrupt numbers 32-35.
```

```
lw r1, 0x0
bseti r1, 0x0 //set interrupt 32 enable
bit bseti r1, 0x1 //set interrupt 33
enable bit bseti r1, 0x2 //set interrupt
34 enable bit bseti r1, 0x3 //set
interrupt 35 enable bit
lw r2, 0xe000e100 //Address corresponding to ISER, Interrupt Enable Register
st.w r1, (r2, 0x0) //Enable 4 interrupts with interrupt numbers 32-35
```

```
//Other interrupts are enabled as above, by setting different bits in the ISER register to enable the
corresponding interrupts.
```

```
//Set Interrupt Priority Register IPR0 to set the priority of interrupts 32-35.
```

```
lw r1, 0x0
```

```
//Set IPR0[7:6] Set the priority of interrupt #32 to 2'b11, lowest priority
```

```
// Set IPR0[15:14] to set the priority of interrupt #33 to 2'b00, highest priority
```

(continued on next page)

```
// Set IPR0[23:22] to set the priority of interrupt #34 to 2'b10.

// Set IPR0[31:30] to set the priority of interrupt #35 to

    2'b10 bseti r1, 0x6 // Set interrupt priority #32 by the

    low [7:6] two bits
    bseti r1, 0x7 //set interrupt #32 to lowest priority, IPR0[7:6]=2'b11
    bclri r1, 0x14 // set interrupt priority 33 with [15:14] bits
    bclri r1, 0x15 //Set interrupt 33 priority to highest, IPR0[15:14]=2'b00
    bclri r1, 0x22 //Set interrupt 34 priority by [23:22] two bits
    bseti r1, 0x23 //Set interrupt priority 2 for interrupt 34,
    IPR0[23:22]=2'b10 bclri r1, 0x30 //Set interrupt priority 35 with
    [31:30] two bits
    bseti r1, 0x31 //set priority 2 for interrupt ,IPR0[31:30]=2'b10

// Write the set priority to IPR0.

    lrw r2, 0xe000e400 // Address corresponding to interrupt enable register IPR0
    st.w r1, (r2, 0x0) //complete interrupt priority setting for numbers 32-35

//Other interrupt priority settings the same as above, by setting the corresponding interrupt priority registers IPR1-
IPR7.

// Set interrupt priority threshold register VIC_IPTR

    lrw r1, 0x2200 //set interrupt number of interrupt priority threshold
    register, IPTR[16:8]=34 bseti r1, 0x0 //enable interrupt priority
    threshold register

// Set the interrupt priority that can seize interrupt 34 by IPTR[7:6]. 

//Set the priority that can preempt interrupt 34 must be higher than 2'b01.

// i.e. only interrupts with priority 2'b00 (highest) can seize interrupt 34

    bseti r1, 0x6
    bclri r1, 0x7

//Write the priority threshold, enable and interrupt number
```

corresponding to the set interrupt number 34 to IPTR lrw r2,

User's Manual

```
0xe000ec04 //Interrupt Priority Threshold Register address  
st.w r1, (r2, 0x0) //finish setting priority threshold for interrupt 34
```

Chapter 6 instruction set

6.1 summarize

The E804's instruction set has high-level language features and is optimized for a number of frequently executed instructions. The instruction set includes standard arithmetic logic instructions, bit manipulation instructions, byte fetch instructions, data transfer instructions, control flow change instructions, and conditional execution instructions, which help to minimize conditional transfers for short jumps.

The E804 instructions are available in two widths: 16-bit and 32-bit. The instruction code is a mix of the two types of instructions, and there is no additional overhead for switching between the two types of instructions.

6.2 32-bit instruction

This section focuses on the 32-bit instruction set of the E804 implementation of CSKY V2.

6.2.1 32-bit Instruction Function Classification

The 32-bit instruction set of the E804 can be divided according to the functions implemented by the instructions:

- Data arithmetic instructions;
- Branch jump instruction;
- Memory access instructions;
- Privileged Directive;
- Special function commands;
- DSP Extended Instructions.

6.2.1.1 data operation instruction

The data operation class of instructions can be further categorized into:

addition and subtraction commands

Table 6.1: List of 32-Bit Addition and Subtraction Instructions

ADDU32	Unsigned addition instruction.
ADDC32	Unsigned with rounding addition instructions.
ADDI32	Unsigned immediate number addition instruction.
SUBU32	Unsigned subtraction instruction.
SUBC32	Unsigned with debit subtraction instruction.
SUBI32	Unsigned immediate number subtraction instruction.
RSUB32	Reverse subtraction instruction.
IXH32	Index half-word commands.
IXW32	Index word command.
IXD32	Indexed double-word commands.
INCF32	C is a 0 immediate number addition instruction.
INCT32	C is a 1-immediate number addition instruction.
DECF32	C is a 0 immediate number subtraction instruction.
DECT32	C is a 1-immediate number subtraction instruction.
DEC GT32	Subtract Greater Than Zero Set C Bit Instruction.
DECLT32	Subtract Less Than Zero Set C Bit Instruction.
DECNE32	Subtract not equal to zero C-bit instruction.

logical operation instruction

Table 6.2: List of 32-Bit Logic Instructions

AND32	Pressing bits and commands.
ANDI32	Immediate numbers by bit with instructions.

ANDN32	Per-bit non-associative instructions.
ANDNI32	Immediate number by bit non-associative instruction.
OR32	By Bit or Command.
ORI32	Immediate numbers by bit or instruction.
XOR32	Bitwise Alteration Instruction.
XORI32	Immediate number by bitwise dissimilarity instruction.
NOR32	Bitwise or non-bitwise instructions.
NOT32	Press the bit non-command.

shift command (computing)

Table 6.3: List of 32-Bit Shift Instructions

LSL32	Logical left shift instruction.
LSLI32	Immediate number logical left shift instruction.
LSLC32	Immediate number logic left shift to C-bit instruction.
LSR32	Logical right shift instruction.
LSRI32	Immediate number logical right shift instruction.
LSRC32	Immediate number logic is shifted right to the C-bit instruction.
ASR32	Arithmetic right shift instruction.
ASRI32	Immediate number arithmetic right shift instruction.
ASRC32	Immediate number arithmetic right shift to C-bit instruction.
ROTL32	Cyclic left shift instruction.
ROTLI32	Immediate number cyclic left shift instruction.
XSR32	Extended right shift command.

compare instruction

Table 6.4: List of 32-Bit Compare Instructions

CMPNE32	Unequal Comparison Instruction.
CMPNEI32	Immediate number unequal comparison instruction.
CMPHS32	Unsigned greater-than-equal comparison instructions.
CMPHSI32	Immediate number unsigned greater than or equal to compare instruction.
CMPLT32	Signed less than compare instructions.
CMPLTI32	Immediate numbers are signed less than the compare instruction.
TST32	Zero test command.
TSTNBZ32	No byte equals zero register test instruction.

data transmission instruction

Table 6.5: List of 32-bit Data Transfer Instructions

MOV32	Data Transfer Command.
MOVF32	C is 0 Data transfer instruction.
MOVT32	C is a 1 data transfer instruction.
MOVI32	Immediate number data transfer instructions.
MOVIH32	Immediate number of high bit data transfer instructions.
MVCV32	C Bit Fetch and Reverse Transfer Instruction.
MVC32	C Bit Transfer Command.
CLRF32	C is 0 Clear command.
CLRT32	C is a 1 clear instruction.
LRW32	Memory read instruction.
GRS32	Symbols generate instructions.

bit operation instruction

Table 6.6: List of 32-Bit Bit Operation Instructions

BSETI32	Immediate Number Position Bit Instructions.
BTSTI32	Immediate digital test instructions.

Extract Insert command

Table 6.7: List of 32-bit Extract Insert Instructions

ZEXT32	Bit extraction and unsigned extension instructions.
SEXT32	Bit extraction with signed extension instructions.
INS32	Bit Insertion Instruction.
ZEXTB32	Byte extraction and unsigned extension instructions.
ZEXTH32	Half-word extraction and unsigned extension instructions.
SEXTB32	Byte extraction with symbolic extension instructions.

SEXTH32	Half-word extraction with signed extension instructions.
XTRB0.32	Extracts byte 0 with an unsigned spread instruction.
XTRB1.32	Extract byte 1 and unsigned extension instruction.
XTRB2.32	Extract byte 2 and unsigned extension instruction.
XTRB3.32	Extract byte 3 and unsigned extension instruction.
BREV32	Bit Reverse Order Instruction.
REVB32	Byte Reverse Instruction.
REVH32	Byte inversion instruction within a half-word.

Multiplication and division commands

Table 6.8: List of 32-Bit Multiply and Divide Instructions

MULT32	multiplication instruction
MULLL.S16 (MULSH32)*	16-bit signed multiply instruction.
MUL.(U/S)32	32-bit (no/have) signed multiply instructions.
MULA.32.	32-bit signed multiply-accumulate takes the lower 32 bits of the instruction.
MULA.(U/S)32	32-bit (no/have) signed multiply-accumulate instruction.
MULALL.S16.	16-bit signed low halfword multiply-accumulate instruction with saturation operation.
DIVU32	Unsigned division instruction.
DIVS32	Signed division instructions.

Note: * The instruction MULSH32 in the historical version will be replaced by MULLL.S16 with the same function.

miscellaneous arithmetic instructions

Table 6.9: List of 32-bit Miscellaneous Arithmetic Instructions

ABS32	Absolute value command.
FF0. 32	Quick Find 0 command.
FF1. 32	Quick Find 1 command.
BMASKI32	Immediate digit mask generation instructions.
BGENR32	Register bit generation instructions.
BGENI32	Immediate digit generation instructions.

6.2.1.2 branch jump instruction

Branch jump instructions can be further categorized into:

branching instruction

Table 6.10: List of 32-Bit Branching Instructions

BT32	C is a 1-branch command.
BF32	C is a 0 branch command.
BEZ32	Register equals zero branch instruction.

BNEZ32	Register not equal to zero branch instruction.
BNEZAD32	Register self-decrement greater than zero branch instruction.
BHZ32	Registers greater than zero branch instructions.
BHSZ32	Registers greater than or equal to zero branch instructions.
BLZ32	Register less than zero branch instructions.
BLSZ32	Register less than or equal to zero branch instruction.

jump instruction

Table 6.11: List of 32-bit Jump Instructions

BR32	Unconditional jump instruction.
BSR32	Jump to subroutine instruction.
JMPI32	Indirect jump instructions.
JSRI32	Indirect jumps to subroutine instructions.
JMP32	Register jump instructions.
JSR32	Register jumps to subroutine instructions.
RTS32	Link register jump instructions.

6.2.1.3 memory access instruction

Memory access instructions can be further categorized into:

Immediate Number Offset Access Instruction

Table 6.12: List of 32-Bit Immediate Number Offset Access Instructions

LD32.	Unsigned extended byte load instruction.
LD32.BS	Signed Extended Byte Load instruction.
LD32.H	Unsigned extended half-word load instruction.
LD32.HS	Signed extended half-word load instructions.
LD32.	Word Load command.
ST32.	Byte storage instructions.
ST32.H	Half-word storage instructions.
ST32.	Word storage instructions.

Vector Register Offset Access Instructions

Table 6.13: List of 32-Bit Vector Register Offset Access Instructions

LDR32.	Register shift addressing unsigned extended byte load instructions.
LDR32.BS	Register shift addressing signed extended byte load instructions.

LDR32.H	Register Shift Addressing Unsigned Extended Half-Word Load Instruction.
LDR32.HS	Register shift addressing signed extended half-word load instructions.
LDR32.	Register Shift Addressing Word Load Instruction.
STR32.	Register shift addressing byte store instructions.
STR32.H	Register shift addressing half-word store instructions.
STR32.	Register shift addressing word store instructions.

multi-register access instruction

Table 6.14: List of 32-Bit Multi-Register Access Instructions

LDQ32	Consecutive four-word loading instructions.
LDM32	Continuous multi-word load instructions.
STQ32	Consecutive four-word storage instructions.
STM32	Continuous multi-word store instructions.
PUSH32	Stacking instructions.
POP32	Out-of-stack instructions.

symbolic access instruction

6.2.1.4 privileged instruction

Privileged directives can be further categorized into:

Control Register

Operation Instructions

Table 6.15: List of 32-Bit Symbolic Access Instructions

LRS32.	Byte symbol loading instructions.
LRS32.	Half-word symbol loading instructions.
LRS32.	Character symbol loading instructions.
SRS32.	Byte symbol storage instructions.
SRS32.H	Half-word symbol storage instructions.
SRS32.	Word Symbol Storage Instruction.

Table 6.16: 32-Bit Control Register Operation Instruction List

MFCR32	Control register read transfer instructions.
MTCR32	Control register write transfer instruction.
PSRSET32	PSR Position Bit Command.

PSRCLR32	PSR bit clear instruction.
----------	----------------------------

Low-power commands

Table 6.17: List of 32-bit Low Power Instructions

WAIT32	Enter low-power wait mode command.
DOZE32	Enter low-power sleep mode command.
STOP32	Enter low-power suspend mode command.

Exception return instruction

Table 6.18: List of 32-Bit Exception Return Instructions

RTE32	Exception and normal interrupt return instructions.
-------	---

6.2.1.5 Special Function Instructions

The special function commands are shown [Table 6.19](#).

Table 6.19: List of 32-Bit Special Function Instructions

SYNC32	CPU synchronization instructions.
BKPT32	Breakpoint command.
SCE32	The condition executes the setup command.
IDLY32	Interrupt recognition disable command.
TRAP32	Unconditional OS trap command.

6.2.1.6 DSP Extended Instructions

The E804 supports DSP extension instructions when the DSP extension unit is configured. The following describes the subset of DSP instructions in alphabetical order.

The mnemonic for each DSP instruction consists of three parts, each separated by a "._." The first part identifies the basic operation of the instruction, the second part identifies the data type of the operand, and the third part identifies the processing of the result of the operation. Take " 16 bit parallel signed addition instruction with saturation operation - PADD.S16.S" as an example, PADD means that this is a parallel addition instruction, S16 means that the operand is a signed half-word, and S means that the saturation operation is applied to the result of the operation.

Table 6.20: Common Letters and Their Meanings in Command Mnemonics

mnemonic sign	English presentation	significance
Part I	P(Parallel)	Parallel computation, denoting single instruction multiple data operations.
Part I	R(fraction)	Fractional operations.
Part I	X (Unalign)	The operand bit widths are not aligned, or the operand positions are not aligned.
Part I	H (Half)	Take the average.
Part II	(S/U)/(8/16/32)	(signed/unsigned)/(byte/half word/word) If it is not indicated by a sign bit, it means that the operation of the instruction is not concerned with the sign bit.

User's Manual		Heart.
Part III	S(Saturate)	Saturation operation.
Part III	R (Round)	rounding operation.
Part III	E (Extension)	Expansion operation, that is, the operand into the expansion operation after the operation to participate in the operation to produce the result.

The specific commands are shown Table 6.21.

Table 6.21: DSP Extended Instruction List

command name	command description
addition and subtraction commands	
PADD.8	8-bit parallel addition instructions.
PADD.16	16-bit parallel addition instructions.
PADD.(U/S)8.	8-bit parallel (no/have) signed addition instruction with saturation operation.
PADD.(U/S)16.	16-bit parallel (no/have) signed addition instruction with saturation operation.
ADD.(U/S)32.	32-bit (no/have) signed addition instruction with saturation operation.
PSUB.8	8-bit parallel subtraction instructions.
PSUB.16	16-bit parallel subtraction instructions.
PSUB.(U/S)8.	8-bit parallel (no/have) signed subtraction instruction with saturation operation.
PSUB.(U/S)16.	16-bit parallel (no/have) signed subtraction instruction with saturation operation.
SUB(U/S)32.	32-bit (no/have) signed subtraction instruction with saturation operation.
PADDH.(U/S)8	8-bit parallel (no/have) signed addition averaging instruction.
PADDH.(U/S)16	16-bit parallel (no/have) signed addition averaging instruction.
ADDH.(U/S)32	32-bit (no/have) signed addition averaging instruction.
PSUBH.(U/S)8	8-bit parallel (no/have) signed subtraction averaging instruction.
PSUBH.(U/S)16	16-bit parallel (no/have) signed subtraction averaging instruction.
SUBH.(U/S)32	32-bit (no/have) signed subtraction averaging instruction.
PASX.16	16-bit parallel cross-add and subtract instructions.
PSAX.16	16-bit parallel cross-subtract and add instructions.
PASX.(U/S)16.	16-bit parallel (no/have) sign cross-add and subtract instructions with saturation operations.
PSAX.(U/S)16.	16-bit parallel (no/have) sign cross-subtract-add instruction with saturation operation.
PASXH.(U/S)16	16-bit parallel (no/have) sign cross-add and subtract averaging instructions.
PSAXH.(U/S)16	16-bit parallel (no/have) sign cross-subtraction and averaging instructions.
ADD.64	64-bit addition instructions.
ADD.(U/S)64.	64-bit (no/have) symbolic addition instruction with saturation operation.
SUB.64	64-bit subtraction instruction.
SUB.(U/S)64.	64-bit (no/have) signed subtraction instruction with saturation operation.

Shift command (computing)

Shift command (computing)	
ASRI.S32.R	Immediate number arithmetic right shift instruction with rounding operation.
ASR.S32.R	Arithmetic right shift instruction with rounding operation.
LSRI.U32.R	Immediate number logical right shift instruction with rounding operation.
LSR.U32.	Logical right shift instruction with rounding operation.
LSLI.(U/S)32.	Immediate number (no/have) signed logical left instruction with saturation operation.
LSL.(U/S)32.	(No/Have) symbolic logical left shift instruction with saturation operation.
PASRI.S16	16-bit parallel immediate number arithmetic right shift instruction.
PASR.S16	16-bit parallel arithmetic right-shift instruction.
PASRI.S16.R	16-bit parallel immediate number arithmetic right-shift instruction with rounding operation.

Continued on next page

Table 6.21 - Continued from previous page

command name	command description
PASR.S16.R	16-bit parallel arithmetic right-shift instructions with rounding operations.
PLSRI.U16	16-bit Parallel Immediate Logical Right Shift instruction.
PLSR.U16	16-bit parallel logic right-shift instructions.
PLSRI.U16.R	6-bit Parallel Immediate Logical Right Shift instruction with rounding operation.
PLSR.U16.R	16-bit parallel logical right-shift instructions with rounding operations.
PLSLI.16	16-bit Parallel Immediate Logical Left Shift instruction.
PLSL.16	16-bit parallel logic left-shift instruction.
PLSLI.(U/S)16.	16-bit parallel immediate number (no/have) signed logic left shift instruction with saturation operation.
PLSL.(U/S)16.	16-bit parallel (no/have) symbolic logic left shift instruction with saturation operation.
Control flow data volume instructions (or compare/select instructions)	
SEL	Bit select instructions.
PCMPNE.8	8-bit parallel compare unequal instructions.
PCMPNE.16	16-bit parallel compare unequal instructions.
PCMPSHS.(U/S)8	8-bit parallel (no/have) sign greater than or equal to compare instructions.
PCMPSHS.(U/S)16	16-bit parallel (no/have) sign greater than or equal to compare instructions.
PCMPLT.(U/S)8	8-bit parallel (no/have) signed less-than-comparison select instructions.
PCMPLT.(U/S)16	16-bit parallel (no/have) signed less-than-comparison select instructions.
PMAX.(U/S)8	8-bit parallel (no/have) symbolic big-endian instructions.
PMAX.(U/S)16	16-bit parallel (no/have) symbolic big-endian instructions.
MAX.(U/S)32	32-bit (no/have) symbolic take-large instructions.
PMIN.(U/S)8	8-bit parallel (no/have) signed minus instruction.
PMIN.(U/S)16	16-bit parallel (no/have) signed minus instruction.
MIN.(U/S)32	32-bit (no/have) signed minus instruction.
Commands for data extraction/unpacking/saturation, etc.	
DEXTI	Immediate digital intercept instruction.
DEXT	Word Intercept Command. Continued on next page
PKG	Immediate number shift packing instruction.
PKGLL	Low half-word packing instructions.
PKGHH	High half-word packing instructions.
PEXT.	8-bit parallel unsigned extended instructions.
PEXT.	8-bit parallel signed extended instructions.
PEXTX.U8.	8-bit parallel unsigned cross-extended instructions.
PEXTX. www.xrvm.cn	75 8-bit parallel signed cross-extension instructions. © Copyright © Hangzhou Zhongfian
NARL	Low interception splicing instructions.
NARH	High Intercept Splicing Command.
NARLX	Low cross intercept splicing instructions.

Table 6.21 - continued from previous page

command name	command description
CLIP.(U/S)32	Immediate number (no/yes) Symbolic clipping saturation instruction.
CLIP.(U/S)32	(None/Yes) Symbol crop saturation command.
PCLIP.(U/S)16	16-bit Parallel Immediate Number (no/have) Symbol Trimming Saturation Instruction.
PCLIP.(U/S)16	16-bit parallel (no/have) symbol clipping saturation instruction.
PABS.S8.	8-bit parallel absolute instruction with saturation operation.
PABS.S16.	16-bit parallel absolute instruction with saturation operation.
ABS.S32.	32-bit absolute instruction with saturation operation.
PNEG.	8-bit parallel inverse instruction with saturation operation.
PNEGS16.	16-bit parallel inverse instruction with saturation operation.
NEG.S32.	32-bit inverse instruction with saturation operation.
DUP.8	8-bit operand copy instruction.
DUP.16	16-bit operand copy instruction.

Multiply/Multiply Accumulate/Multiply Decrease Instructions**32×32 takes 64 bits**

MULS.(U/S)32	32-bit (no/have) signed multiply-accumulate instruction.
MULA.(U/S)32.	32-bit (no/have) signed multiply-accumulate instruction with saturation operation.
MULS.(U/S)32.	32-bit (no/have) signed multiply-accumulate instruction with saturation operation.

32×32 Taking the height of the word

MUL.S32.H	32-bit signed multiplication takes the high 32-bit instruction.
MUL.S32.RH	A 32-bit signed multiplication with rounding takes the upper 32 of the instruction.
RMUL.	32-bit signed decimal multiplication takes the upper 32 of the instruction.
RMUL.S32.RH	The 32-bit signed decimal multiplication with rounding takes the upper 32 of the instruction.
MULA.S32.HS	A 32-bit signed multiply-accumulate fetch-high 32-bit instruction with saturation operation.
MULS.S32.HS	32-bit signed multiply-accumulate-take-higher 32-bit instruction with saturation operation.
MULA.S32.RHS	32-bit signed multiply-accumulate fetch-high 32-bit next page instructions with rounding and saturation operations.
MULS.S32.RHS	32-bit signed multiply-accumulate-take-higher 32-bit instructions with rounding and saturation operations.

32×16 Higher characters

MULXL.S32	32-bit signed low halfword unaligned multiply instruction.
MULXL.S32.R	32-bit signed low halfword unaligned multiply instruction with rounding operation.

~~wMULXLH.S32~~ 76 32-bit signed high halfword unaligned multiply instruction.

MULXH.S32.R	32-bit signed high halfword unaligned multiply instruction with rounding operation.
-------------	---

~~RMULXL.S32~~ 22-bit signed low halfword unaligned decimal multiply instruction.

Table 6.21 - continued from previous page

command name	command description
MULAXH.S32.	32-bit signed high halfword unaligned multiply-accumulate instruction with saturation operation.
MULAXH.S32.RS	32-bit signed high halfword unaligned multiply-accumulate instruction with rounding and saturation operations.
16×16, Non-SIMD	
MULLL.S16	16-bit signed low half-word multiply instruction.
MULHH.S16	16-bit signed high half-word multiply instruction.
MULHL.S16	16-bit signed high-low half-word multiply instruction.
RMULLL.S16	16-bit signed low half-word decimal multiply instruction.
RMULHH.S16	16-bit signed high half-word decimal multiply instruction.
RMULHL.S16	16-bit signed high-low half-word decimal multiply instruction.
MULAHH.S16.S	16-bit signed high half-word multiply-accumulate instruction with saturation operation.
MULAHL.S16.	16-bit signed high-low half-word multiply-accumulate instruction with saturation operation.
MULALL.S16.E	16-bit signed low halfword multiply-accumulate instruction with extended operation.
MULAHH.S16.E	16-bit signed high half-word multiply-accumulate instruction with extended operation.
MULAHL.S16.E	16-bit signed high-low half-word multiply-accumulate instruction with extended operation.
16 x 16, SIMD	
PMUL.(U/S)16	16-bit parallel (no/have) signed multiply instructions.
PMULX.(U/S)16	16-bit parallel (no/have) signed cross-multiply instruction.
PRMUL.S16	16-bit parallel signed decimal multiply instructions.
PRMULX.(U/S)16	16-bit parallel (no/have) signed cross-decimal multiply instructions.
PRMUL.S16.H	16-bit parallel signed decimal multiplication takes the high 16-bit instruction.
PRMUL.S16.RH	The 16-bit parallel signed decimal multiplication with rounding operation takes the high 16 bits of the instruction.
PRMULX.S16.H	16-bit parallel signed cross-decimal multiplication takes the high 16-bit instruction.
PRMULX.S16.RH	The 16-bit parallel signed cross-decimal multiplication with rounding operation takes the high 16 bits of the instruction.
16 x 16, chain addition and subtraction	
MULCA.S16.	16-bit signed multiply-chain-add instruction with saturation operation.
MULCAX.S16.	16-bit signed cross-multiply chain add instruction with saturation operation.
MULCS.S16	16-bit signed multiply-chain-subtract instructions.
MULCSR.S16	16-bit signed inverse multiply-chain-subtract instructions.
MULCSX.S16	16-bit signed cross-multiply chain subtract instructions.
16 x 16, chain accumulation and subtraction	
MULACA.S16.S	16-bit signed multiply chain plus accumulate instruction with saturation operation.

Table 6.21 - continued from previous page

command name	command description
MULACA.S16.E	16-bit signed multiply-chain-add-accumulate instruction with extended operations.
MULACAX.S16.E	16-bit signed cross-multiply chain plus accumulate instruction with extended operations.
MULACS.S16.E	16-bit signed multiply-chain-subtract-accumulate instruction with extended operations.
MULACSR.S16.E	16-bit signed inverse multiply-chain-subtract-accumulate instruction with extended operations.
MULACSRX.S16.E	16-bit signed cross-multiply-chain-subtract-accumulate instruction with extended operations.
MULSCA.S16.E	16-bit signed multiply-chain-add-accumulate instruction with extended operations.
MULSCAX.S16.	16-bit signed cross-multiply chain plus accumulate-decrease instruction with extended operations.
miscellaneous	
PSABSA.U8	8-bit parallel unsigned subtractive absolute value chaining instructions.
PSABSAA.U8	8-bit parallel unsigned subtractive absolute value chaining accumulation instruction.
DIVSL	Signed long division instructions.
DIVUL	Unsigned long division instruction.
MULACA.S8	8-bit parallel signed multiply chain plus accumulate instructions.
Featured Technology 1: Cyclic Body Acceleration	
BLOOP	Loop body acceleration instructions.
Featured Technology 2: LD/ST	
LDBI.W	Address self-incrementing word load instructions.
LDBI.	Address self-incrementing unsigned half-word load instructions.
LDBI.HS	Address self-incrementing signed half-word load instructions.
LDBI.	Address self-incrementing unsigned byte load instruction.
LDBI.BS	Address self-incrementing signed byte load instruction.
PLDBI.	Address self-incrementing double-word load instructions.
STBI.W	Address self-incrementing word store instructions.
STBI.	Address self-incrementing half-word store instructions.
STBI.	Address self-incrementing byte store instructions.
LDBIR.	Register address self-incrementing word load instruction.
LDBIR.	Unsigned half-word load instructions with self-incrementing register addresses.
LDBIR.HS	Signed half-word load instructions with self-incrementing register addresses.
LDBIR.	Register address self-incrementing unsigned byte load instruction.
www.xrvm.cn	78 instruction. © Copyright © Hangzhou Zhongtian
LDBIR.BS	Register address self-incrementing signed byte load instruction.
PLDBIR.	A double-word load instruction with self-incrementing register

6.3 16-bit instructions

This section describes the 16-bit instruction set of the E804.

6.3.1 16-bit Instruction Function Classification

The 16-bit instruction set of the E804 can be categorized according to the functions implemented by the instructions:

- Data arithmetic instructions;
- Branch jump instruction;
- Memory access instructions;
- Multi-register access instructions.

6.3.1.1 data operation instruction

The data operation class of instructions can be further categorized into:

addition and subtraction commands

Table 6.22: List of 16-Bit Addition and Subtraction Instructions

ADDU16	Unsigned addition instruction.
ADDC16	Unsigned with rounding addition instructions.
ADDI16	Unsigned immediate number addition instruction.
SUBU16	Unsigned subtraction instruction.
SUBC16	Unsigned with debit subtraction instruction.
SUBI16	Unsigned immediate number subtraction instruction.

logical operation instruction

Table 6.23: List of 16-Bit Logic Instructions

AND16	Pressing bits and commands.
ANDN16	Per-bit non-associative instructions.
OR16	By Bit or

	Command.
XOR16	Bitwise Alteration Instruction.
NOR16	Bitwise or non-bitwise instructions.
NOT16	Press the bit non-command.

shift command (computing)

Table 6.24: List of 16-Bit Shift Instructions

LSL16	Logical left shift instruction.
LSLI16	Immediate number logical left shift instruction.
LSR16	Logical right shift instruction.
LSRI16	Immediate number logical right shift instruction.
ASR16	Arithmetic right shift instruction.
ASRI16	Immediate number arithmetic right shift instruction.
ROTL16	Cyclic left shift instruction.

compare instruction

Table 6.25: List of 16-Bit Compare Instructions

CMPNE16	Unequal Comparison Instruction.
CMPNEI16	Immediate number unequal comparison instruction.
CMPHS16	Unsigned greater-than-equal comparison instructions.
CMPHSI16	Immediate number unsigned greater than or equal to compare instruction.
CMPLT16	Signed less than compare instructions.
CMPLTI16	Immediate numbers are signed less than the compare instruction.
TST16	Zero test command.
TSTNBZ16	No byte equals zero register test instruction.

data transmission instruction

Table 6.26: List of 16-bit Data Transfer Instructions

MOV16	Data Transfer Command.
MOVI16	Immediate number

	data transfer instructions.
MVCV16	C Bit Transfer Command.
LRW16	Memory read instruction.

bit operation instruction

Table 6.27: List of 16-Bit Bit Operation Instructions

BCLRI16	Immediate digital clear instruction.
BSETI16	Immediate Number Position Bit Instructions.
BTSTI16	Immediate digital test instructions.

Extract Insert command

Table 6.28: List of 16-Bit Extract Insert Instructions

ZEXTB16	Byte extraction and unsigned extension instructions.
ZEXTH16	Half-word extraction and unsigned extension instructions.
SEXTB16	Byte extraction with symbolic extension instructions.
SEXTH16	Half-word extraction with signed extension instructions.
REVB16	Byte Reverse Instruction.
REVH16	Byte inversion instruction within a half-word.

Multiplication and division commands

Table 6.29: List of 16-Bit Multiplication Instructions

MULT16	Multiply command.
--------	-------------------

6.3.1.2 branch jump instruction

Branch jump instructions can be further categorized into:

branching instruction

Table 6.30: List of 16-Bit Branching Instructions

BT16	C is a 1-branch command.
BF16	C is a 0 branch command.

jump instruction

Table 6.31: List of 16-bit Jump Instructions

BR16	Unconditional jump instruction.
JMP16	Register jump instructions.
JSR16	Register jumps to subroutine instructions.
RTS16	Link register jump instructions.

Memory access instructions can be further categorized into:

Immediate Number Offset Access Instruction

Table 6.32: List of 16-Bit Immediate Number Offset Access Instructions

LD16.	Unsigned extended byte load instruction.
LD16.H	Unsigned extended half-word load instruction.
LD16.W	Word Load command.
ST16.	Byte storage instructions.
ST16.H	Half-word storage instructions.
ST16.	Word storage instructions.

6.3.1.4 multi-register access instruction

Table 6.33: List of 16-Bit Multi-Register Access Instructions

POP16	Out-of-stack instructions.
PUSH16	Stacking instructions.
IPOP16	Interrupts the out-of-stack instruction.
IPUSH16	Interrupt Stack Instruction.
NIE16	Interrupt nested enable instruction.
NIR16	Interrupt nested return instructions.

Note: NIE16 and NIR16 need to be executed in superuser mode.

6.4 Instruction Set List

The E804's instruction set has high-level language features and is optimized for a number of frequently executed instructions. The instruction set includes standard arithmetic logic instructions, bit manipulation instructions, byte fetch instructions, data transfer instructions, control flow change instructions, and conditional execution instructions, which help to minimize conditional transfers for short jumps.

Table 6.34 lists all 16-bit and 32-bit instructions in the E804 instruction set.

Table 6.34: Instruction Set of the E804

User's Manual	Assembly instruction	32-bit	16-bit	compilation format	command description
ABS	O	×		ABS32 RZ, RX	Absolute value command.
ADDC	O	O		ADDC32 RZ,RX, RY ADDC16 RZ, RX	Unsigned with rounding addition instructions.
ADDI	O	O		ADDI32 RZ, RX, OIMM12 ADDI16 RZ, OIMM8	Unsigned immediate number addition instruction.

Continued on next page

Table 6.34 - continued from previous page

assembly instruction	32-bit	16-bit	compilation format	command description
ADDU	O	O	ADDU32 RZ, RX, RY ADDU16 RZ, RX	Unsigned addition instruction.
AND	O	O	AND32 RZ, RX, RY AND16 RZ, RX	Pressing bits and commands.
ANDI	O	×	ANDI32 RZ, RX, IMM12	Immediate numbers by bit with instructions.
ANDN	O	O	andn32 rz, rx, rx ANDN16 RZ, RX	Per-bit non-associative instructions.
ANDNI	O	×	ANDNI32 RZ, RX, IMM12	Immediate number by bit non-associative instruction.
ASR	o	o	ASR32 RZ, RX, RY ASR16 RZ, RX	Arithmetic right shift instruction.
ASRC	o	×	asrc32 rz, rx, oimm5	Immediate number arithmetic right shift to C-bit instruction.
ASRI	o	o	ASRI32 RZ, RX, IMM5 ASRI16 RZ, RX, IMM5	Immediate number arithmetic right shift instruction.
BCLRI	O	O	BCLRI32 RZ, RX, IMM5 BCLRI16 RZ, IMM5	Immediate digital clear instruction.
BEZ	O	×	BEZ32 RX, LABEL	Register equals zero branch instruction.
BF	O	O	BF32 LABEL BF16 LABEL	C is a 0 branch command.
BGENI	O	×	BGENI32 RZ, IMM5	Immediate digit generation instructions.
BGENR	O	×	BGENR32 RZ, RX	Register bit generation instructions.
BHSZ	O	×	BHSZ32 RX, LABEL	Registers greater than or equal to zero branch instructions.
BHZ	o	×	BHZ32 RX, LABEL	Registers greater than zero branch instructions.
BKPT	×	o	BKPT16	Breakpoint command.
BLSZ	o	×	BLSZ32 RX, LABEL	Register less than or equal to zero branch instruction.
BLZ	o	×	BLZ32 RX, LABEL	Register less than zero branch instructions.
BMASKI	o	×	BMASKI32 RZ, OIMM5	Continued on next page Immediate digit mask generation instructions.
BNEZ	o	×	BNEZ32 RX, LABEL	Stores are not equal to the zero branch instruction.
BR	o	o	BR32 LABEL BR16 LABEL	Unconditional jump instruction.
BREV	o	×	BREV32 RZ, RX	Bit Reverse Order Instruction.
BSSETI	O	O	BSSETI32 RZ, RX, IMM5 BSSETI16 RZ, MM5	Copyright © HuaWei HiSilicon Microsystems Co. Immediate Number Position Bit Instructions.
BSR	O	×	BSR32 LABEL	Jump to subroutine instruction.
BT	O	O	BT32 LABEL	

Table 6.34 - continued from previous page

assembly instruction	32-bit	16-bit	compilation format	command description
CLRT	o	×	CLRT32 RZ	C is a 1 clear instruction.
CMPHS	o	O	CMPHS32 RX, RY CMPHS16 RX, RY	Unsigned greater-than-equal comparison instructions.
CMPHSI	O	O	CMPHSI32 RX, OIMM16 CMPHSI16 RX, IMMS	Immediate number unsigned is greater than or equal to the comparison finger Order.
CMPLT	O	o	CMPLT32 RX, RY CMPLT16 RX, RY	Signed less than compare instructions.
CMPLTI	o	o	CMPLTI32 RX, OIMM16 CMPLTI16 RX, OIMM5	Immediate numbers are signed less than the compare instruction.
CMPNE	o	o	CMPNE32 RX, RY CMPNE16 RX, RY	Unequal Comparison Instruction.
CMPNEI	o	o	CMPNEI32 RX, IMM16 CMPNEI16 RX, IMM5	Immediate number unequal comparison instruction.
DECFS	o	×	DECFS32 RZ, RX, IMM5	C is a 0 immediate number subtraction instruction.
DECFT	o	×	DECFT32 RZ, RX, IMM5	Subtract Greater Than Zero Set C Bit Instruction.
DECLT	o	×	DECLT32 RZ, RX, IMM5	Subtract Less Than Zero Set C Bit Instruction.
DECNE	o	×	DECNE32 RZ, RX, IMM5	Subtract not equal to zero C-bit instruction.
DECT	o	×	DECT32 RZ, RX, IMM5	C is a 1-immediate number subtraction instruction.
DIVS	O	×	divs32 rz, rx, ry	Signed division instructions.
DIVU	o	×	divu32 rz, rx, ry	Unsigned division instruction.
DOZE	o	×	DOZE32	Enter low-power sleep mode command.
FF0	o	×	FF0.32 RZ, RX	Quick Find 0 command.
FF1	O	×	FF1.32 RZ, RX	Quick Find 1 command.
GRS	O	×	GRS32 RZ, LABEL GRS32 RZ, IMM32	Symbols generate instructions.
IDLY	O	×	IDLY32 N	Interrupt recognition disable command. Continued on next page
INCF	O	×	INCF32 RZ, RX, IMM5	C is a 0 immediate number addition instruction.
INCT	O	×	INCT32 RZ, RX, IMM5	C is a 1-immediate number addition instruction.
INS	O	×	INS32 RZ, RX, MSB, LSB	Bit Insertion Instruction.
IPOP	×	O	IPOP16	Interrupts the out-of-stack instruction.
IPUSH	×	O	IPUSH16 88	Interrupt Stack Instruction.
IXD	O	×	IXD32 RZ, RX, RY	Indexed double-word commands.
IXH	O	×	IXH32 RZ, RX, RY	Index half-word commands.
IXW	O	×	IXW32 RZ, RX, RY	Index word commands.

Table 6.34 - continued from previous page

assembly instruction	32-bit	16-bit	compilation format	command description
JSR	o	o	JSR32 RX JSR16 RX	Register jumps to subroutine instructions.
JSRI	o	×	JSRI32 LABEL	Indirect jumps to subroutine instructions.
LD.	o	o	LD32.B RZ, (RX, DISP) LD16.B RZ, (RX, DISP)	Unsigned extended byte load instruction.
LD.BS	O	×	LD32.BS RZ, (RX, DISP)	Signed Extended Byte Load instruction.
LD.H	o	o	LD32.H RZ, (RX, DISP) LD16.H RZ, (RX, DISP)	Unsigned extended half-word load instruction.
LD.HS	o	×	LD32.HS RZ, (RX, DISP)	Signed extended half-word load instructions.
LD.W	o	o	LD32.W RZ, (RX, DISP) LD16.W RZ, (RX, DISP)	Word Load command.
LDM	o	×	LDM32 RY-RZ, (RX)	Continuous multi-word load instructions.
LDQ	O	×	LDQ32 R4-R7, (RX)	Consecutive four-word loading instructions.
LDR.	o	×	LDR32.B RZ, (RX, RY<< 0) LDR32.B RZ, (RX, RY<< 1) LDR32.B RZ, (RX, RY<< 2) LDR32.B RZ, (RX, RY<< 3)	Register Shift Addressing Unsigned Extended Byte Load command.
LDR.BS	o	×	LDR32.BS RZ, (RX, RY<< 0) LDR32.BS RZ, (RX, RY<< 1) LDR32.BS RZ, (RX, RY<< 2) LDR32.BS RZ, (RX, << 3)	Register Shift Addressing Signed Extended Byte Load command.
LDR.	o	×	LDR32.H RZ, (RX, RY<< 0) LDR32.H RZ, (RX, RY<< 1) LDR32.H RZ, (RX, RY<< 2) LDR32.H RZ, (RX, RY<< 3)	Register Shift Addressing Unsigned Extended Half-Word Load command. Continued on next page
LDR.HS	o	×	LDR32.HS RZ, (RX, RY<< 0) LDR32.HS RZ, (RX, RY<< 1) LDR32.HS RZ, (RX, RY<< 2) LDR32.HS RZ, (RX, RY) << 3)	Register Shift Addressing Signed Extended Half-Word Load command.

Table 6.34 - continued from previous page

assembly instruction	32-bit	16-bit	compilation format	command description
LDR.	o	×	LDR32.W RZ, (RX, RY<< 0) LDR32.W RZ, (RX, RY<< 1) LDR32.W RZ, (RX, RY<< 2) LDR32.W RZ, (RX, << 3)	Register Shift Addressing Word Load Instruction.
LRS.	o	×	lrs32.b rz, [label]	Byte symbol loading instructions.
LRS.	O	×	LRS32.H RZ, [LABEL]	Half-word symbol loading instructions.
LRS.	O	×	LRS32.W RZ, [LABEL]	Character symbol loading instructions.
LRW	O	O	lrw16 label lrw16 imm32 lrw32 label LRW32 IMM32	Memory read instruction.
LSL	O	O	LSL32 RZ, RX, RY LSL16 RZ, RY	Logical left shift instruction.
LSLC	O	×	LSLC32 RZ, RX, OIMM5	Immediate number logic left shift to C-bit instruction.
LSLI	O	O	LSLI32 RZ, RX, IMM5 LSLI16 RZ, RX, IMM5	Immediate number logical left shift instruction.
LSR	O	O	LSR32 RZ, RX, RY LSR16 RZ, RY	Logical right shift instruction.
LSRC	O	×	lsrc32 rz, rx, oimm5	Immediate number logic is shifted right to the C-bit instruction.
LSRI	O	O	LSRI32 RZ, RX, IMM5 LSRI16 RZ, RX, IMM5	Immediate number logical right shift instruction.
MFCR	O	×	mfcr32 rz, cr <x, sel>	Control register read transfer instructions.
MOV	O	O	MOV16 RZ, RX	Data Transfer Command.
MOVF	O	×	MOVF32 RZ, RX	C is 0 Data transfer instruction.
MOVI	O	O	MOVI32 RZ, IMM16 MOVI16 RZ, IMM8	Immediate contended data transfer instructions.
MOVIH	O	×	MOVIH32 RZ, IMM16	Immediate number of high bit data transfer instructions.
MOVT	O	×	MOVT32 RZ, RX	C is a 1 data transfer instruction.
MTCR	o	×	mtcrr32 rx, cr<z, sel>	Control register write transfer instruction.
MULA.32. www.xrvm.cn	o	×	MULA.32.L RZ, RX, RY	32-bit signed multiply-accumulate takes the lower 32-bit reference. © Copyright Hangzhou Zhongtian Microsystems Co.
SWP A,B,C,D			SWP A,B,C,D	Order.

Table 6.34 - continued from previous page

assembly instruction	32-bit	16-bit	compilation format	command description
MULT	o	o	MULT32 RZ, RX, RY MULT16 RZ, RX	Multiply command.
MUL.(U/S)32	o	×	MUL.S32 RZ, RX, RY	32-bit (no/have) signed multiply instructions.
MVC	O	×	MVC32 RZ	C Bit Transfer Command.
MVCV	O	O	MVCV32 RZ MVCV16 RZ	C Bit Fetch and Reverse Transfer Instruction.
NIE	×	O	NIE	Interrupt nested enable instruction.
NIR	×	O	NIR	Interrupt nested return instructions.
NOR	O	O	NOR32 RZ, RX, RY NOR16 RZ, RX	Bitwise or non-bitwise instructions.
NOT	O	O	NOT32 RZ, RX NOT16 RZ	Press the bit non-command.
OR	O	O	OR32 RZ, RX, RY OR16 RZ, RX	By Bit or Command.
ORI	o	×	ORI32 RZ, RX, IMM16	Immediate number by bit or instruction.
POP	o	o	POP32 REGLIST POP16 REGLIST	Out-of-stack instructions.
PSRCLR	o	×	PSRCLR32 EE, IE, FE, AF	PSR bit clear instruction.
PSRSET	O	×	PSRSET32 EE, IE, FE, AF	PSR Position Bit Command.
PUSH	O	O	PUSH32 REGLIST PUSH16 REGLIST	Stacking instructions.
REVB	O	O	REVB32 RZ, RX REVB16 RZ, RX	Byte Reverse Instruction.
REVH	O	O	REVH32 RZ, RX REVH16 RZ, RX	Byte inversion instruction within a half-word.
ROTL	O	O	ROTL32 RZ, RX, RY ROTL16 RZ, RY	Cyclic left shift instruction.
ROTLI	O	×	ROTLI32 RZ, RX, IMM5	Immediate number cyclic left shift instruction.
RSUB	O	×	RSUB32 RZ, RX, RY	Reverse subtraction instruction.
RTE	O	×	RTE32	Exception and more than interrupt return instructions.
RTS	O	O	RTS16	Link register jump instructions.
SCE	O	×	SCE32 COND	The condition executes the setup command.
SEXT	O	×	SEXT32 RZ, RX, MSB, LSB	Bit extraction with signed extension instructions.
SEXTB www.xrvm.cn	o	o	SEXTB32 RZ, RX SEXTB16 RZ, RX	Byte extraction with symbolic extension instructions. © Copyright © Hangzhou Zhongtian Microsystem Co., Ltd.
SETHX	o	o	SETHX32 RZ, RX SETHX16 RZ, RX	Half-word extraction with signed extension instructions.

Table 6.34 - continued from previous page

assembly instruction	32-bit	16-bit	compilation format	command description
SRS.	o	×	SRS32.H RZ, [LABEL]	Half-word symbol storage instructions.
SRS.W	o	×	SRS32.W RZ, [LABEL]	Word Symbol Storage Instruction.
ST.	o	o	ST32.B RZ, (RX, DISP) ST16.B RZ,(RX, DISP)	Byte storage instructions.
ST.H	o	O	ST32.H RZ, (RX, DISP) ST16.H RZ,(RX, DISP)	Half-word storage instructions.
ST.W	o	o	ST32.W RZ, (RX, DISP) ST16.W RZ,(RX, DISP)	Word storage instructions.
STM	o	×	STM32 RY-RZ, (RX)	Continuous multi-word store instructions.
STOP	o	×	STOP32	Enter low-power suspend mode command.
STQ	o	×	STQ32 R4-R7, (RX)	Consecutive four-word storage instructions.
STR.	o	×	STR32.B RZ, (RX, RY<< 0) STR32.B RZ, (RX, RY<< 1) STR32.B RZ, (RX, RY<< 2) STR32.B RZ, (RX, RY<< 3)	Register shift addressing byte store instructions.
STR.	o	×	STR32.H RZ, (RX, RY<< 0) STR32.H RZ, (RX, RY<< 1) STR32.H RZ, (RX, RY<< 2) STR32.H RZ, (RX, RY<< 3)	Register shift addressing half-word store instructions.
STR.	o	×	STR32.W RZ, (RX, RY<< 0) STR32.W RZ, (RX, RY<< 1) STR32.W RZ, (RX, RY<< 2) STR32.W RZ, (RX, << 3)	Register shift addressing word store instructions.
SUBC	o	o	SUBC32 RZ, RX, RY SUBC16 RZ, RY	Unsigned with debit subtraction instruction.
SUBI	O	O	SUBI32 RZ, RX, OIMM12 SUBI16 RZ, OIMM8	Unsigned immediate number subtraction instruction. <small>Continued on next page</small>
SUBU	o	o	SUBU32 RZ, RX, RY SUBU16 RZ, RY	Unsigned subtraction instruction.
SYNC	o	×	SYNC32 IMM5	CPU synchronization instructions.
TRAP	o	×	TRAP32 0 TRAP32 1 TRAP32 2	Unconditional Operating System Trap command.
www.xrvm.cn			TRAP32 3 92	© Copyright © Hangzhou Zhongtian Microsystems Co.

Table 6.34 - continued from previous page

assembly instruction	32-bit	16-bit	compilation format	command description
TST	O	O	TST32 RX, RY TST16 RX, RY	Zero test command.
TSTNBZ	O	O	TSTNBZ32 RX TSTNBZ16 RX	No Byte Equals Zero Register Test Instructions.
WAIT	O	×	WAIT32	Enter low-power wait mode command.
XOR	O	O	XOR32 RZ, RX, RY XOR16 RZ, RX	Bitwise Alteration Instruction.
XORI	o	×	XORI32 RZ, RX, IMM12	Immediate number by bitwise dissimilarity instruction.
XSR	o	×	xsr32 rz, rx, oimm5	Extended right shift command.
XTRB0	o	×	XTRB0.32 RZ, RX	Extract byte 0 and unsigned extension instruction.
XTRB1	O	×	XTRB1.32 RZ, RX	Extract byte 1 and unsigned extension instruction.
XTRB2	o	×	XTRB2.32 RZ, RX	Extract byte 2 and unsigned extension instruction.
XTRB3	o	×	XTRB3.32 RZ, RX	Extract byte 3 and unsigned extension instruction.
ZEXT	o	×	ZEXT32 RZ, RX, MSB, LSB	Bit extraction and unsigned extension instructions.
ZEXTB	o	o	ZEXTB32 RZ, RX ZEXTB16 RZ, RX	Byte extraction and unsigned extension instructions.
ZEXTH	o	o	ZEXTH32 RZ, RX ZEXTH16 RZ, RX	Half-word extraction and unsigned extension instructions.

DSP instruction

PADD.8	o	×	PADD.8 RZ, RX, RY	8-bit parallel addition instructions.
PADD.16	o	×	PADD.16 RZ, RX, RY	16-bit parallel addition instructions.
PADD.(U/S)8.	o	×	PADD.(U/S)8.S RZ, RX, RY	8-bit parallel with saturation operation (without/with) Symbolic addition instruction.
PADD.(U/S)16.	o	×	PADD.(U/S)16.S RZ, RX, RY	16-bit parallel with saturation operation (without/with) Symbolic addition instruction. <small>Continued on next page</small>
ADD.(U/S)32.	O	×	ADD.(U/S)32.S RZ, RX, RY	32-bit (no/have) character with saturation operation No. Addition instruction.
PSUB.8	O	×	PSUB.8 RZ, RX, RY	8-bit parallel subtraction instructions.
PSUB.16	O	×	PSUB.16 RZ, RX, RY	16-bit parallel subtraction instructions.
PSUB.(U/S)8.	O	×	PSUB.(U/S)8.S RZ, RX, RY	8-bit parallel with saturation operation (without/with) Symbolic subtraction instructions. <small>Continued on next page</small>

Table 6.34 - continued from previous page

assembly instruction	32-bit	16-bit	compilation format	command description
PADDH.(U/S)16	o	×	PADDH.(U/S)16 RZ, RX, RY	16-bit parallel(unsigned/symbolized) addition leveling Mean value command.
ADDH.(U/S)32	O	×	ADDH.(U/S)32 RZ, RX, RY	32 unsigned/symbolized) addition averaging Instructions.
PSUBH.(U/S)8	o	×	PSUBH.(U/S)8 RZ, RX, RY	8-bit parallel (no/have) signed subtraction leveling Mean value command.
PSUBH.(U/S)16	o	×	PSUBH.(U/S)16 RZ, RX, RY	16-bit parallel (no/have) signed subtraction leveling Mean value command.
SUBH.(U/S)32	o	×	SUBH.(U/S)32 RZ, RX, RY	32 (no/have) signed subtraction averaging Instructions.
PASX.16	O	×	PASX.16 RZ, RX, RY	16-bit parallel cross-add and subtract instructions.
PSAX.16	O	×	PSAX.16 RZ, RX, RY	16-bit parallel cross-subtract and add instructions.
PASX.(U/S)16.	O	×	PASX.(U/S)16.S RZ, RX, RY	16-bit parallel with saturation operation (without/with) Symbolic cross-add and subtract instructions.
PSAX.(U/S)16.	O	×	PSAX.(U/S)16.S RZ, RX, RY	16-bit parallel with saturation operation (without/with) Symbolic cross-subtraction and addition instructions.
PASXH.(U/S)16	O	×	PASXH.(U/S)16 RZ, RX, RY	16-bit parallel (no/have) sign cross-add and subtract Average command.
PSAXH.(U/S)16	O	×	PSAXH.(U/S)16 RZ, RX, RY	16-bit parallel (no/have) signed cross-subtract and add Average command.
ADD.64	O	×	ADD.64 RZ, RX, RY	64-bit addition instructions.
ADD.(U/S)64.	O	×	ADD.(U/S)64.S RZ, RX, RY	64-bit (no/have) character with saturation operation <small>Continued on next page</small> No. Addition instruction.
SUB.64	O	×	SUB.64 RZ, RX, RY	64-bit subtraction instruction.
SUB.(U/S)64.	O	×	SUB.(U/S)64.S RZ, RX, RY	64-bit (no/have) character with saturation operation No. Subtract command.
ASRI.S32.R	O	×	asri.s32.r rz,rx,oimm5	Immediate number arithmetic right shift finger with rounding
www.xrvm.cn			94	operation © Copyright © Hangzhou Zhongtian Order Microsystems Co.
ASR.S32.R	o	×	ASR.S32.R RZ, RX, RY	Arithmetic right shift instruction with rounding operation.

Table 6.34 - continued from previous page

assembly instruction	32-bit	16-bit	compilation format	command description
PASRI.S16.R	o	×	PASRI.S16.R RZ,RX,OIMM4	16-bit Parallel Immediate Number with Rounding Operations Arithmetic right shift instruction.
PASR.S16.R	O	×	PASR.S16.R RZ, RX, RY	16-bit parallel arithmetic right with rounding operations shift command (computing)
PLSRI.U16	o	×	plsri.u16 rz,rx,oimm4	16-bit Parallel Immediate Logical Right Shift instruction.
PLSR.U16	o	×	PLSR.U16 RZ, RX, RY	16-bit parallel logic right-shift instructions.
PLSRI.U16.R	o	×	PLSRI.U16.R RZ,RX,OIMM4	16-bit Parallel Immediate Number with Rounding Operations Logical right shift instruction.
PLSR.U16.R	O	×	PLSR.U16.R RZ, RX, RY	16-bit Parallel Logic Right with Rounding Operations Move command.
PLSLI.16	O	×	PLSLI.16 RZ,RX,OIMM4	16-bit Parallel Immediate Logical Left Shift instruction.
PLSL.16	O	×	PLSL.16 RZ, RX, RY	16-bit parallel logic left-shift instruction.
PLSLI.(U/S)16.	O	×	PLSLI.(U/S)16. RZ,RX,OIMM4	16-Bit Parallel Immediate Number with Saturation Operation (No/Have) symbolic logical left shift instruction.
PLSL.(U/S)16.	O	×	PLSL.(U/S)16.S RZ, RX, RY	16-bit parallel with saturation operation (without/with) Symbolic logic left shift instruction.
SEL	o	×	sel rz, rx, ry,rs	Bit select instructions.
PCMPNE.8	o	×	PCMPNE.8 RZ, RX, RY	8-bit parallel compare unequal instructions.
PCMPNE.16	o	×	PCMPNE.16 RZ, RX, RY	16-bit parallel compare unequal instructions.
PCMPHS.(U/S)8	o	×	PCMPHS.(U/S)8 RZ, RX.	8-bit Parallel (none/have) Signed greater than or equal to Compare instructions. <small>Continued on next page</small>
PCMPHS.(U/S)16	O	×	PCMPHS.(U/S)16 RZ, RX.	16-bit parallel (no/have) sign greater than or equal to Compare instructions.
PCMPLT.(U/S)8	o	×	PCMPLT.(U/S)8 RZ, RX.	8-bit parallel (no/have) sign less than compare Select the command.
PCMPLT.(U/S)16 www.xrvm.cn	o	×	pcmplt.(u/s)16 rz, rx. RY	16-bit parallel (no/have) sign less than compare <small>© Copyright © Hangzhou Zhongtian Microsystems Co.</small> Select the command.
PMAX.(U/S)8	o	×	PMAX.(U/S)8 RZ, RX, RY	8-bit Parallel (None/Have)

Table 6.34 - continued from previous page

assembly instruction	32-bit	16-bit	compilation format	command description
DEXT	o	×	dext rz, rx, ry,rs	Word Intercept Command.
PKG	o	×	PKG RZ, RX. imm4a, ry, oimm4b	Immediate number shift packing instruction.
PKGLL	O	×	PKGLL RZ, RX, RY	Low half-word packing instructions.
PKGHH	o	×	PKGHH RZ, RX, RY	High half-word packing instructions.
PEXT.	o	×	PEXT.U8.E RZ,RX	8-bit parallel unsigned extended instructions.
PEXT.	o	×	PEXT.S8.E RZ,RX	8-bit parallel signed extended instructions.
PEXTX.U8.	o	×	PEXTX.U8.E RZ,RX	8-bit parallel unsigned cross-extended instructions.
PEXTX.	O	×	PEXTX.S8.E RZ,RX	8-bit parallel signed cross-extension instructions.
NARL	O	×	NARL RZ, RX, RY	Low interception splicing instructions.
NARH	o	×	NARH RZ, RX, RY	High Intercept Splicing Command.
NARLX	o	×	NARLX RZ, RX, RY	Low cross intercept splicing instructions.
NARHX	o	×	NARHX RZ, RX, RY	High level cross intercept splicing instructions.
CLIP.(U/S)32	O	×	CLIP.(U/S)32 RZ. RX,IMM5	Immediate number (no/have) Symbol clipping saturation finger Order.
CLIP.(U/S)32	o	×	CLIP.(U/S)32 RZ, RX, RY	(None/Yes) Symbol crop saturation command.
PCLIP.(U/S)16	o	×	PCLIP.(U/S)16 RZ, RX, IMM4	16-bit Parallel Immediate (No/Have) Symbol Cutter Shear saturation command.
PCLIP.(U/S)16	o	×	PCLIP.(U/S)16 RZ, RX, RY	16-bit parallel (without/with) symbol clipping saturation Instructions.
PABS.S8.	O	×	PABS.S8.S RZ,RX	8-bit parallel absolute value pointer with saturation operation Order. Continued on next page
PABS.S16.	o	×	pabs.s16.srz,rx	16-bit parallel absolute with saturation operation Instructions.
ABS.S32.	o	×	ABS.S32.S RZ,RX	32-bit absolute instruction with saturation operation.
PNEG.	o	×	pneg.s8.s rz,rx	8-bit Parallel Inverted Finger with Saturation Operation
www.xrvm.cn			96	© Copyright © Hangzhou Zhongtian Order Microsystems Co.
PNEGS16.	O	×	pnegs.s16.s rz,rx	16-bit Parallel Inverted Finger with Saturation Operation

Table 6.34 - continued from previous page

assembly instruction	32-bit	16-bit	compilation format	command description
MUL.S32.RH	o	×	MUL.S32.RH RZ, RX, RY	32-Bit Signed Multiplication with Rounding Fetch high 32-bit instructions.
RMUL.	o	×	RMUL.S32.H RZ, RX, RY	32-bit Signed Decimal Multiplication Takes Higher 32 Bits Instructions.
RMUL.S32.RH	o	×	RMUL.S32.RH RZ, RX, RY	32-bit signed decimal with rounding operation Multiply to fetch the high 32-bit instruction.
MULA.S32.HS	O	×	MULA.S32.HS RZ, RX, RY	32-Bit Signed Multiply with Saturation Operation Add the high 32-bit instructions.
MULS.S32.HS	O	×	MULS.S32.HS RZ, RX, RY	32-Bit Signed Multiply with Saturation Operation Minus the high 32-bit instructions.
MULA.S32.RHS	O	×	MULA.S32.RHS RZ, RX, RY	32-bit hash with rounding and saturation operations The No. multiply-accumulate-take-high 32-bit instruction.
MULS.S32.RHS	O	×	MULS.S32.RHS RZ, RX, RY	32-bit hash with rounding and saturation operations The No. multiply-accumulate-take-higher 32-bit instruction.
MULXL.S32	O	×	MULXL.S32 RZ, RX, RY	32-bit Signed Low Halfword Unaligned Multiplication Finger Order.
MULXL.S32.R	O	×	MULXL.S32.R RZ, RX, RY	32-bit Signed Low Half with Rounding Operation Word unaligned multiply command.
MULXH.S32	O	×	MULXH.S32 RZ, RX, RY	32-bit Signed High Halfword Unaligned Multiplication Finger Order. Continued on next page
MULXH.S32.R	O	×	MULXH.S32.R RZ, RX, RY	32-bit Signed High Half with Rounding Operations Word unaligned multiply command.
RMULXL.S32	O	×	RMULXL.S32 RZ, RX, RY	32-bit Signed Low Halfword Unaligned Decimal Multiplication Instructions.
www.xrvm.cn			97	Copyright © Hangzhou Zhongtian Systems Co.
RMULXL.S32.R	o	×	RMULXL.S32.R RZ, RX, RY	32-bit Signed Low Half with Rounding Operation

Table 6.34 - continued from previous page

assembly instruction	32-bit	16-bit	compilation format	command description
MULLL.S16	O	×	MULLL.S16 RZ, RX, RY	16-bit signed low half-word multiply instruction.
MULHH.S16	O	×	MULHH.S16 RZ, RX, RY	16-bit signed high half-word multiply instruction.
MULHL.S16	O	×	MULHL.S16 RZ, RX, RY	16-bit signed high-low half-word multiply instruction.
RMULLL.S16	O	×	RMULLL.S16 RZ, RX, RY	16-Bit Signed Low Half-Word Decimal Multiplication Finger Order.
RMULHH.S16	O	×	RMULHH.S16 RZ, RX, RY	16-Bit Signed High Half-Word Decimal Multiplication Finger Order.
RMULHL.S16	o	×	RMULHL.S16 RZ, RX, RY	16-bit Signed High/Low Half Decimal Multiplication Finger Order.
MULAHH.S16.S	o	×	MULAHH.S16.S RZ, RX, RY	16-bit Signed High Half with Saturation Operation Word Multiply Accumulate instruction.
MULAHL.S16.	o	×	MULAHL.S16.S RZ, RX, RY	16-bit signed high/low with saturation operation Half-word multiply-accumulate instruction.
MULALL.S16.E	O	×	MULALL.S16.E RZ, RX, RY	16-bit Signed Lower Half with Extended Operation Word Multiply Accumulate instruction.
MULAHH.S16.E	o	×	MULAHH.S16.E RZ, RX, RY	16-bit Signed High Half with Extended Operation Word Multiply Accumulate instruction.
MULAHL.S16.E	o	×	MULAHL.S16.E RZ, RX, RY	16-bit Signed High-Low with Extended Operation Half-word multiply-accumulate instruction.
PMUL.(U/S)16	o	×	PMUL.(S/U)16 RZ, RX, RY	16-bit Parallel (Without/With) Signed Multiplication Finger Order. <small>Continued on next page</small>
PMULX.(U/S)16	O	×	PMULX.(S/U)16 RZ, RX, RY	16-bit parallel (no/have) signed cross-multipliers Order.
PRMUL.S16	O	×	PRMUL.S16 RZ, RX, RY	16-bit parallel signed decimal multiply instructions.
wPRMULX.(U/S)16	O	×	PRMULX.S16 RZ, RX, RY	16-bit Parallel Hangzhou Zhongtian (Unsigned/Halfed) Signed Crossed Decimals Multiply command

Table 6.34 - continued from previous page

assembly instruction	32-bit	16-bit	compilation format	command description
MULCS.S16	o	×	MULCS.S16 RZ, RX, RY	16-bit signed multiply-chain-subtract instructions.
MULCSR.S16	o	×	MULCSR.S16 RZ, RX, RY	16-bit signed inverse multiply-chain-subtract instruction.
MULCSX.S16	O	×	MULCSX.S16 RZ, RX, RY	16-bit signed cross-multiply-chain-subtract instructions.
MULACA.S16.S	O	×	MULACA.S16.S RZ, RX, RY	16-bit Signed Multiplication Chain with Saturation Operation Add the accumulate command.
MULACAX.S16.	O	×	MULACAX.S16.S RZ, RX, RY	16-bit signed cross with saturation operation Multiply chain plus accumulate instruction.
MULACS.S16.	O	×	MULACS.S16.S RZ, RX, RY	16-bit Signed Multiplication Chain with Saturation Operation Minus Accumulation instruction.
MULACSR.S16.	O	×	MULACSR.S16.S RZ, RX, RY	16-bit signed inverse with saturation operation Multiply Chain Subtract Accumulate instruction.
MULACSX.S16.	o	×	MULACSX.S16.S RZ, RX, RY	16-bit signed cross with saturation operation Multiply Chain Subtract Accumulate instruction.
MULSCA.S16.	o	×	MULSCA.S16.S RZ, RX, RY	16-bit Signed Multiplication Chain with Saturation Operation Add and Decrement commands.
MULSCAX.S16.	o	×	MULSCAX.S16.S RZ, RX, RY	16-bit signed cross with saturation operation Multiply chain plus accumulate and decrease instructions.
MULACA.S16.E	O	×	MULACA.S16.E RZ, RX, RY	16-Bit Signed Multiplication Chain with Extended Operations Add the accumulate command.
MULACAX.S16.E	o	×	MULACAX.S16.E RZ, RX, RY	16-Bit Signed Cross with Extended Operation Continued on next page Multiply chain plus accumulate instruction.
MULACS.S16.E	o	×	MULACS.S16.E RZ, RX, RY	16-Bit Signed Multiplication Chain with Extended Operations Minus Accumulation instruction.
MULACSR.S16.E	o	×	MULACSR.S16.E RZ, RX, RY	16-Bit Signed Inverse with Extended Operations
www.xrvm.cn			99	Multiply Chain Subtract Microsystems Co. Accumulate Instruction.
MULACSRX.S16.E	O	×	MULACSRX.S16.E RZ, RX, RY	16-Bit Signed Cross with Extended Operation

Table 6.34 - continued from previous page

assembly instruction	32-bit	16-bit	compilation format	command description
BLOOP	O	×	bloop rx, label1, la-BEL2	Loop body acceleration instructions.
LDBI.W	O	×	LDBI.W RZ, (RX)	Address self-incrementing word load instructions.
LDBI.	O	×	LDBI.H RZ, (RX)	Unsigned half-word loaded finger with address self-incrementation Order.
LDBI.HS	O	×	LDBI.HS RZ, (RX)	Address self-incrementing signed half-word loaded finger Order.
LDBI.	O	×	LDBI.B RZ, (RX)	Address self-incrementing unsigned byte loaded finger Order.
LDBI.BS	O	×	LDBI.BS RZ, (RX)	Address self-incrementing signed byte load refers to the Order.
PLDBI.	O	×	PLDBI.D RZ, (RX)	Address self-incrementing double-word load instructions.
STBI.W	O	×	STBI.W RZ, (RX)	Address self-incrementing word store instructions.
STBI.	O	×	STBI.H RZ, (RX)	Address self-incrementing half-word store instructions.
STBI.	O	×	STBI.B RZ, (RX)	Address self-incrementing byte store instructions.
LDBIR.	O	×	LDBIR.W RZ, (RX), RY	Register address self-incrementing word load instruction
LDBIR.	O	×	LDBIR.H RZ, (RX), RY	Register address self-incrementing unsigned halfword plus Load instructions.
LDBIR.HS	O	×	LDBIR.HS RZ, (RX), RY	Register address self-incremented signed halfword plus Load instructions.
LDBIR.	O	×	LDBIR.B RZ, (RX), RY	Register address self-incrementing unsigned byte plus Load instructions.
LDBIR.BS	O	×	LDBIR.BS RZ, (RX), RY	Register address self-incremented signed byte plus Load instructions.

PLDBIR User's Manual	O	×	PLDBIR.D RZ, (RX) , RY	Register address self-incrementing double-word load finger Order.
STBIR.W	O	×	STBIR.W RZ, (RX) , RY	Register address self-incrementing word store instructions.
STBIR.	O	×	STBIR.H RZ, (RX) , RY	Register Address Self-Incrementing Half-Word Memory Finger Order.
STBIR.	O	×	STBIR.B RZ, (RX) , RY	Register address self-incrementing byte storage refers to the Order.

Note: O means the instruction exists in the corresponding instruction set, × means the instruction does not exist in the corresponding instruction set. * denotes a DSP extension instruction.

6.5 Command execution delay

Table 6.35: Instruction Execution Delay Table

Instruction type	implementation period	note
Base command		
addition and subtraction commands		
ADDU32/16	1	-
ADDC32/16	1	-
ADDI32/16	1	-
SUBU32/16	1	-
SUBC32/16	1	-
SUBI32/16	1	-
RSUB32	1	-
IXH32	1	-
IXW32	1	-
IXD32	1	-
INCF32	1	-
INCT32	1	-
DECFF32	1	-
DECT32	1	-
DEC GT32	1	-
DECLT32	1	-
DECNE32	1	-
logical operation instruction		
AND32/16	1	-
ANDI32/16	1	-
ANDN32	1	-
ANDNI32	1	-
OR32/16	1	-
ORI32	1	-
XOR32	1	-
XORI32	1	-
NOR32/16	1	-
NOT32/16	1	-
shift command (computing)		
LSL32/16	1	-
LSLI32/16	1	-
LSLC32	1	-
LSR32/16	1	-

Continued on next page

Table 6.35 - continued from previous page

Instruction type	implementation period	note
LSRI32/16	1	-
LSRC32	1	-
ASR32/16	1	-
ASRI32/16	1	-
ASRC32	1	-
ROTL32/16	1	-
ROTLI32	1	-
XSR32	1	-
compare instruction		
CMPNE32/16	1	-
CMPNEI32/16	1	-
CMPHS32/16	1	-
CMPHSI32/16	1	-
CMPLT32/16	1	-
CMPLTI32/16	1	-
TST32/16	1	-
TSTNBZ32/16	1	-
data transmission instruction		
MOV32/16	1	-
MOVF32	1	-
MOVT32	1	-
MOVI32/16	1	-
MOVIH32	1	-
MVCV32/16	1	-
MVC32	1	-
CLRF32	1	-
CLRT32	1	-
LRW32/16*		-
GRS32	1	-
bit operation instruction		
BCLRI32/16	1	-
BSETI32/16	1	-
BTSTI32/16	1	-
Extract Insert command		
ZEXT32/16	1	-
SEXT32/16	1	-
INS32	1	-

Continued on next page

Table 6.35 - continued from previous page

Instruction type	implementation period	note
ZEXTB32	1	-
ZEXTH32	1	-
SEXTB32/16	1	-
SEXTH32/16	1	-
XTRB0.32	1	-
XTRB1.32	1	-
XTRB2.32	1	-
XTRB3.32	1	-
BREV32	1	-
REVB32/16	1	-
REVH32/16	1	-
Multiplication and division commands		
MULT32/16	1/3	1 cycle when DSP or floating point is configured; 3 cycles in base configuration
MULLLL.S16 (MULSH)	1	-
MUL.(U/S)32	1/2/5	1 cycle with DSP configuration; 2 with floating point configuration; base configuration 5 cycles as follows
MULA.32.	1/2/4	1 cycle with DSP configuration; 2 with floating point configuration; base configuration 4 cycles as follows
MULA.(U/S)32	1/2/5	1 cycle for DSP configuration; 2 for floating point configuration; base configuration 4 cycles as follows
MULALL.S16.	1-4	1 cycle for DSP configuration; 2 for floating point configuration; base configuration 2 cycles as follows
DIVU32	5-36	-
DIVS32	5-36	-
miscellaneous arithmetic instructions		
ABS32	1	-
FF0.32	1	-
FF1.32	1	-
BMASKI32	1	-
BGENR32	1	-
BGENI32	1	-
branching instruction		
BT32/16	1	-
BF32/16	1	-
BEZ32	1	-
BNEZ32 www.xrvm.cn	1	-
BHZ32	1	-
BLSZ32	1	-

Continued on next page

Table 6.35 - continued from previous page

Instruction type	implementation period	note
BLZ32	1	-
BHSZ32	1	-
jump instruction		
BR32/16	1	-
BSR32	1	-
JMPI32*	3	Split into LRW and JMP instructions
JSRI32*	3	Split into LRW and JSR instructions
JMP32/16	1	-
JSR32/16	1	-
RTS32/16	1	-
Immediate Number Offset Access Instruction		
LD32/16.	1	-
LD32.BS*	1	-
LD32/16.H*	1	-
LD32.HS*	1	-
LD32/16.W*	1	-
ST32/16.	1	-
ST32/16.H*	1	-
ST32/16.W*	1	-
register offset access instruction		
LDR32.	3	-
LDR32.BS*	3	-
LDR32.H*	3	-
LDR32.HS*	3	-
LDR32.W*	3	-
STR32.	3	-
STR32.H*	3	-
STR32.W*	3	-
multi-register access instruction		
LDQ32*	4	Split into 4 LD.W instructions.
LDM32*	N	Split into N LD.W instructions.
STQ32*	4	Split into 4 ST.W instructions.
STM32*	N	Split into N ST.W instructions. Continued on next page
PUSH32/16*	1+N	Split into SUB and N ST.W instructions.
POP32/16*	2+N	W, ADD and RTS instructions.
IPOP16*	7	Split into 7 atomic instructions
IPUSH16*	7	Split into 7 atomic instructions

Table 6.35 - continued from previous page

Instruction type	implementation period	note
NIE16*	5	Split into 5 atomic instructions
NIR16*	5	Split into 5 atomic instructions
symbolic access instruction		
LRS32.	1	-
LRS32.H*	1	-
LRS32.W*	1	-
SRS32.	1	-
SRS32.H*	1	-
SRS32.W*	1	-
Control Register Operation Instructions		
MFCR32	1	-
MTCR32	1	-
PSRSET32	1	-
PSRCLR32	1	-
Low-power commands		
WAIT32	1	-
DOZE32	1	-
STOP32	1	-
Exception return instruction		
RTE32	1	-
Special Function Instructions		
SYNC32	1	-
BKPT32	1	-
SCE32	1	-
IDLY32	1	-
TRAP32	1	-
DSP instruction		
addition and subtraction commands		
PADD.8	1	-
PADD.16	1	-
PADD.(U/S)8.	1	-
PADD.(U/S)16.	1	-
ADD.(U/S)32.	1	-
PSUB.8	1	-
PSUB.16	1	-
PSUB.(U/S)8.	1	-
PSUB.(U/S)16.	1	-

Continued on next page

Table 6.35 - continued from previous page

Instruction type	implementation period	note
SUB(U/S)32.	1	-
PADDH.(U/S)8	1	-
PADDH.(U/S)16	1	-
ADDH.(U/S)32	1	-
PSUBH.(U/S)8	1	-
PSUBH.(U/S)16	1	-
SUBH.(U/S)32	1	-
PASX.16	1	-
PSAX.16	1	-
PASX.(U/S)16.	1	-
PSAX.(U/S)16.	1	-
PASXH.(U/S)16	1	-
PSAXH.(U/S)16	1	-
ADD.64	1	-
ADD.(U/S)64.	1	-
SUB.64	1	-
SUB.(U/S)64.	1	-
PABS.S8.	1	-
PABS.S16.	1	-
ABS.S32.	1	-
PNEG.	1	-
PNEGS16.	1	-
NEG.S32.	1	-
PSABSA.U8	1	-
PSABSAA.U8	1	-
shift command (computing)		
ASRI.S32.R	1	-
ASR.S32.R	1	-
LSRI.U32.R	1	-
LSR.U32.	1	-
LSLI.(U/S)32.	1	-
LSL.(U/S)32.	1	-
PASRI.S16	1	-
PASR.S16	1	-
PASRI.S16.R	1	-
PASR.S16.R	1	-
PLSRI.U16	1	-

Continued on next page

Table 6.35 - continued from previous page

Instruction type	implementation period	note
PLSR.U16	1	-
PLSRI.U16.R	1	-
PLSR.U16.R	1	-
PLSLI.16	1	-
PLSL.16	1	-
PLSLI.(U/S)16.	1	-
PLSL.(U/S)16.	1	-
compare instruction		
PCMPNE.8	1	-
PCMPNE.16	1	-
PCMPHS.(U/S)8	1	-
PCMPHS.(U/S)16	1	-
PCMPLT.(U/S)8	1	-
PCMPLT.(U/S)16	1	-
PMAX.(U/S)8	1	-
PMAX.(U/S)16	1	-
MAX.(U/S)32	1	-
PMIN.(U/S)8	1	-
PMIN.(U/S)16	1	-
MIN.(U/S)32	1	-
Special Function Instructions		
DEXTI	1	-
DEXT	1	-
PKG	1	-
PKGLL	1	-
PKGHH	1	-
PEXT.	1	-
PEXT.	1	-
PEXTX.U8.	1	-
PEXTX.	1	-
NARL	1	-
NARH	1	-
NARLX	1	-
NARHX	1	-
CLIP.(U/S)32	1	-
CLIP.(U/S)32	1	-
PCLIP.(U/S)16	1	-

Continued on next page

Table 6.35 - continued from previous page

Instruction type	implementation period	note
PCLIP.(U/S)16	1	-
DUP.8	1	-
DUP.16	1	-
BLOOP*	1	-
SEL	1	-
multiplication instruction		
MULS.(U/S)32	2	-
MULA.(U/S)32.	2	-
MULS.(U/S)32.	2	-
MUL.S32.H	2	-
MUL.S32.RH	2	-
RMUL.	2	-
RMUL.S32.RH	2	-
MULA.S32.HS	2	-
MULS.S32.HS	2	-
MULA.S32.RHS	2	-
MULS.S32.RHS	2	-
MULXL.S32	2	-
MULXL.S32.R	2	-
MULXH.S32	2	-
MULXH.S32.R	2	-
RMULXL.S32	2	-
RMULXL.S32.R	2	-
RMULXH.S32	2	-
RMULXH.S32.R	2	-
MULAXL.S32.	2	-
MULAXL.S32.RS	2	-
MULAXH.S32.	2	-
MULAXH.S32.RS	2	-
MULLL.S16	2	-
MULHH.S16	2	-
MULHL.S16	2	-
RMULLL.S16	2	-
RMULHH.S16	2	-
RMULHL.S16	2	-
MULAHH.S16.S	2	-
MULAHL.S16.	2	-

Continued on next page

Table 6.35 - continued from previous page

Instruction type	implementation period	note
MULALL.S16.E	2	-
MULAHH.S16.E	2	-
MULAHL.S16.E	2	-
PMUL.(U/S)16	2	-
PMULX.(U/S)16	2	-
PRMUL.S16	2	-
PRMULX.(U/S)16	2	-
PRMUL.S16.H	2	-
PRMUL.S16.RH	2	-
PRMULX.S16.H	2	-
PRMULX.S16.RH	2	-
MULCA.S16.	2	-
MULCAX.S16.	2	-
MULCS.S16	2	-
MULCSR.S16	2	-
MULCSX.S16	2	-
MULACA.S16.S	2	-
MULACAX.S16.	2	-
MULACS.S16.	2	-
MULACSR.S16.	2	-
MULACSX.S16.	2	-
MULSCA.S16.	2	-
MULSCAX.S16.	2	-
MULACA.S16.E	2	-
MULACAX.S16.E	2	-
MULACS.S16.E	2	-
MULACSR.S16.E	2	-
MULACSRX.S16.E	2	-
MULSCA.S16.E	2	-
MULSCAX.S16.	2	-
DIVUL	5-68	-
DIVSL	5-68	-
MULACA.S8	2	-
memory access instruction		Continued on next page
LDBI.W*	1	-
LDBI.H*	1	-
LDBI.HS*	1	-

Table 6.35 - continued from previous page

Instruction type	implementation period	note
LDBI.B*	1	-
LDBI.BS*	1	-
PLDBI.D*	1	-
STBI.W*	1	-
STBI.H*	1	-
STBI.B*	1	-
LDBIR.W*	1	-
LDBIR.H*	1	-
LDBIR.HS*	1	-
LDBIR.	1	-
LDBIR.BS*	1	-
PLDBIR.D*	2	-
STBIR.W*	1	-
STBIR.H*	1	-
STBIR.B*	1	-

Note: * denotes memory access related instructions, the instruction completion cycle depends on the bus delay or cache hit, the values in the table are the fastest cases.

Chapter 7 Memory Protection

7.1 Introduction to the Memory Protection Unit

In a protected system, there are two main types of resource accesses that need to be monitored: memory systems and peripheral devices. The memory protection unit is responsible for checking the legality of accesses to the memory system (including peripheral devices), and its main functions include:

1. Determines whether the CPU has read/write access to memory addresses in the current operating mode.
2. Get additional attributes for this access address, including security attributes, etc.

The Memory Protection Unit supports N table entries (N is hardware configurable from 1 to 8) which allow you to set access rights and attributes for N areas. Each table entry is identified and indexed by a number from 0 to 7. The table entries of the Memory Protection Unit are shown [Figure 7.1](#).

31 30	11 10	6	5	4 3	2	1	0
EN	Base Address	SIZE	S	NX	AP	-	

Figure 7.1: Memory Protection Unit Table Entry

Among them:

EN: Indicates whether the region is in effect;

Base Address: the starting address of the

region; **Size:** the size of the region;

S: denotes the security

attributes of the region; **NX:**

denotes the executability of the

region's fetch finger; **AP:** denotes

the access rights of the region.

Refer to the relevant system control registers for specific attributes of each field.

7.2 Related System Control Registers

7.2.1 Cache Configuration Register (CCR, CR<18,0>)

The Cache Configuration Register is used to configure the memory protection enable, the size end mode, and the system and processor clock ratios.

	31	17	16	15	11	10	8	7	6	5	4	3	2	1	0
					0	0		SCK	BE	0	0	0	0	0	MP
Reset					0				0	0	0	0	0	0	0

Figure 7.2: Cache Configuration Registers

SCK-System and processor clock ratio:

This bit is used to indicate the clock ratio between the system and the processor, which is calculated as follows:

Clock Ratio = SCK + 1, with corresponding pins pinned out on the CPU.

SCK is configured power on reset and cannot be changed afterwards.

The domain currently has no functionality and is for software queries only.

BE-Endian mode:

- BE is 0, the small end;
- BE is 1, the big end.

BE is configured power on reset and cannot be changed later, there are corresponding pins pinned out on the CPU.

MP-Memory Protection Setting Bit:

MP is used to set whether the MPU is valid or not, as shown in [Table 7.1](#).

Table 7.1: E804 Memory Protection Settings

MP	functionalit y
00	MPU is invalid.
01	MPU Effective.
10	MPU is invalid.
11	MPU Effective.

7.2.2 Configuration Register for Elevatable Access Privileges (CAPR, CR<19,0>)

CAPRs are shown [Figure 7.3](#).

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Reset	S7	S6	S5	S4	S3	S2	S1	S0	AP7		AP6	AP5	AP4	AP3	AP2	AP1	AP0	NX7	NX6	NX5	NX4	NX3	NX2	NX1	NX0								
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 7.3: Elevated Ease of Access Configuration Registers

NX0~NX7-Non-executable attribute setting bits:

User's Manual

When NX is 0, the area is executable;
when NX is 1, the area is non-executable.

Note: An access error exception occurs when the processor fetch finger accesses a non-executable region.

S0~S7-Security attribute setting bits:

- S is 0, the zone is non-secure;
- S is 1, the zone is safe.

AP0~AP7-Access permission setting bits:

Table 7.2: Access Rights Settings

AP	superuser rights	General user rights
00	inaccessible	inaccessible
01	fill out or in (information on a form)	inaccessible
10	fill out or in (information on a form)	read-only (computing)
11	fill out or in (information on a form)	fill out or in (information on a form)

7.2.3 Protected Area Control Register (PACR, CR<20,0>)

PACRs are shown [Figure 7.4](#).

31	12	11	6	5	1	0
Base Address	0	SIZE	E			
Reset	0	-	0			

Figure 7.4: Protected Area Control Registers

Base Address-The base address of the protected area address:

- This register indicates the base address of the protected area address, but the base address written

must be aligned with the page size set, for example, if the page size is set to 8M, CR<20,0>[22:12]

must be 0. The specific requirements for each page are shown in [Table 7.3](#).

- The size of the protected area ranges from 4KB to 4GB, which can be calculated by the formula:
protected area size = $2^{(\text{Size}+1)}$. size can be set in the range of 01011 to 11111, some other values can cause unpredictable results.

Table 7.3: Protected Area Size Configurations and Their Base Site Requirements

Size	Size of protected areas	Requirements for the base address
00000-01010	reservations	-
01011	4KB	Not required
01100	8KB	CR<20,0>.bit[12]=0
01101	16KB	CR<20,0>.bit[13:12]=0
01110	32KB	CR<20,0>.bit[14:12]=0
01111	64KB	CR<20,0>.bit[15:12]=0
10000	128KB	CR<20,0>.bit[16:12]=0
10001	256KB	CR<20,0>.bit[17:12]=0
10010	512KB	CR<20,0>.bit[18:12]=0
10011	1MB	CR<20,0>.bit[19:12]=0
10100	2MB	CR<20,0>.bit[20:12]=0
10101	4MB	CR<20,0>.bit[21:12]=0
10110	8MB	CR<20,0>.bit[22:12]=0
10111	16MB	CR<20,0>.bit[23:12]=0
11000	32MB	CR<20,0>.bit[24:12]=0
11001	64MB	CR<20,0>.bit[25:12]=0
11010	128MB	CR<20,0>.bit[26:12]=0
11011	256MB	CR<20,0>.bit[27:12]=0
11100	512MB	CR<20,0>.bit[28:12]=0
11101	1GB	CR<20,0>.bit[29:12]=0
11110	2GB	CR<20,0>.bit[30:12]=0
11111	4GB	CR<20,0>.bit[31:12]=0

E-Protected areas are effectively set up:

- E is 0, the protected area is invalid;
- E is 1, the protected area is valid.

7.2.4 Protected Region Selection Register (PRSR, CR<21,0>)

PRSR is used to select the protection zone for the current operation, as shown Figure 7.5:

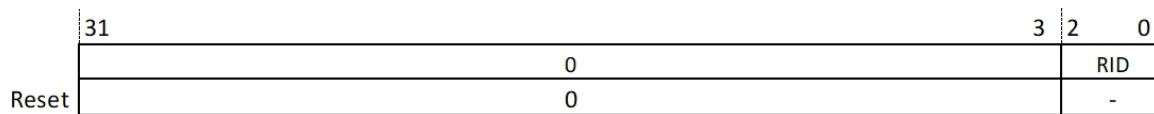


Figure 7.5: Protected Area Selection Registers

RID-Reserve Index Value:

- RID indicates the corresponding protected area selected, e.g. 000 for protected area 0.

7.3 memory access processing

The process flow and results of CPU memory accesses by the memory protection unit are shown Table 7.4.

Table 7.4: Memory Access Processing Flow and Results

Memory Protection Unit Enablement	Memory Protection Unit hit condition	Outcome of the process
disable	not care	All accesses to memory are considered: Super User Mode/Normal User Mode are both readable and writable; The access attribute is unbufferable.
enable	miss	The processor will generate an access error exception.
enable	hits	If the access hits a single protected area, the access rights and attributes of that protected area will prevail; if the access address hits multiple protected areas, the access rights and attributes of the protected area with the highest index number will prevail. The access privileges for this address are combined with the current processor operating mode to determine whether the access is legal. If the access complies with the access rights of the region, the memory protection unit allows this memory operation and provides the access attributes of the access address; if the access request is not allowed, the memory protection unit will generate an access error exception and interrupt processor execution.
<hr/> Note: The access address may hit more than one protected area, in this case it is considered to hit the protected area with the highest index number.		

Chapter 8 On-Chip Cache

8.1 Introduction to Cache

The E804 provides a hardware configurable cache with the following key features:

- Cache size is hardware configurable and supports 2KB/4KB/8KB/16KB;
- 4-way groups are connected, and the cache line size is 16 bytes;
- The maximum width of each access is 4 bytes and supports byte/halfword/word accesses;
- Physical address index, physical address tag;
- The write policy supports both write straight mode and write back mode;
- The cache uses a first-in-first-out (FIFO) replacement strategy;
- Supports invalidate and clear operations for the entire cache and invalidate and clear operations for a single cache line.

When the memory accesses the non-high-slow region, it bypasses the on-chip cache directly and accesses the off-chip memory through the bus to accelerate the access speed of the non-high-slow region. When memory accesses the cacheable region, the on-chip cache is accessed first; the off-chip memory is accessed again in case of cache miss (write accesses in write-straight mode also need to write back to the off-chip memory in case of a cache hit)

8.2 Related System Control Registers

The Cache provides a set of 32-bit registers, including Cache Enable, Cache Line Invalidation, and Cache Area Configuration. The address space of each register is shown [Table 8.1](#).

Table 8.1: Cache Control Register Definitions

address	name (of a thing)	typology	startin g value	descriptive
0xE000F000	CER	Read/ Write	0	Cache Enable Register.
0xE000F004	CIR	Read/ Write	0	Cache Invalidation Register.
0xE000F008	CRCR0	Read/ Write	0	No. 0 can be a high-mitigation zone configuration register.
0xE000F00C	CRCR1	Read/ Write	0	No. 1 Highable Buffer Configuration Register.
0xE000F010	CRCR2	Read/ Write	0	No. 2 Highly cacheable configuration register.
0xE000F014	CRCR3	Read/ Write	0	No. 3 Highly retardable zone configuration register.
0xE000F018~ 0xE000FFF3	-	-	-	Reservations.
0xE000FFF4	CPFCR	Read/ Write	0	Cache performance analysis control register.
0xE000FFF8	CPFATR	Read/ Write	0	Cache access count register.
0xE000FFFC	CPFMTR	Read/ Write	0	Cache miss count register.

8.2.1 Cache Enable Register (CER)

The Cache Enable Register is used to configure the cache enable and the operating mode of the cache.

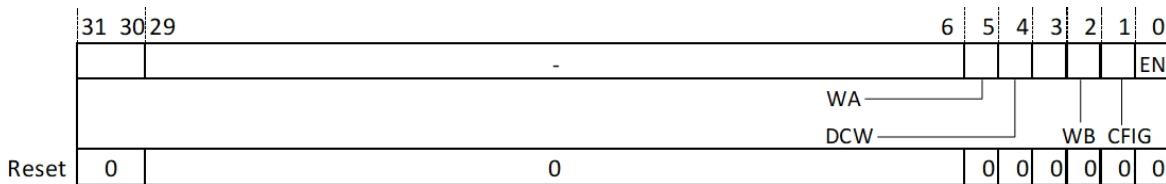


Figure 8.1: Cache Enable Registers

WA Cache Write Allocation Valid Setting Bits:

- 0: The cache is in write non-allocate mode;
- 1: The cache is in write allocate mode;
- The write allocate mode can only be used in write-back mode, otherwise the results are unpredictable.

DCW-Cache Writable Attribute Configuration Bits:

- 0: The cache is not writable;
- 1: Cache writable.

- 0: The cache is in write-straight mode;
- 1: The cache is in write-back mode. CFIG-Cache attribute configuration bit:
- 0: Instruction and data cache;

User's Manual

- 1: Instruction

Cache. EN-Cache

enable bit:

- EN is 0, the cache is off;
- EN is 1, the cache is in operation.

8.2.2 Cache Invalidation Register (CIR)

The Cache Invalidation Register is used to control cache clearing and invalidation, and can support both operations on specific cache lines and operations on the entire cache.

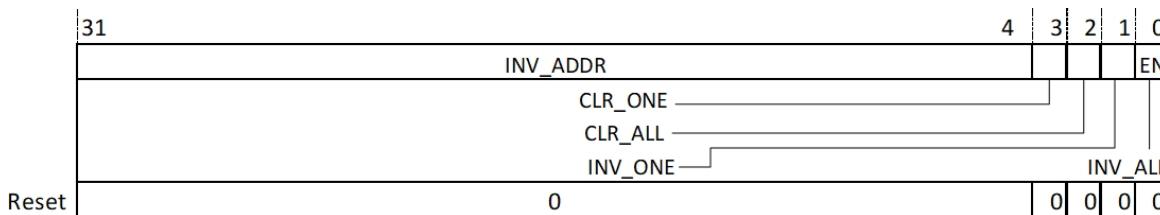


Figure 8.2: Cache Invalidation Registers

INV_ADDR-Cache line address:

- Characterizes the address of the cache line that needs to be invalidated.

CLR_ONE-Single cache line clear set bit:

- CLR_ONE is 1, the selected cache line will be written back to off-chip memory if it is marked as dirty.

CLR_ALL-Entire cache clear set bit:

- CLR_ALL is 1, all cache lines marked as dirty are written back to off-chip memory.

INV_ONE-Single cache line invalidation set bit:

- INV_ONE is 1, invalidates the selected cache line.

INV_ALL-Entire cache invalidation set bit:

- INV_ALL is 1, all cached lines in the invalid cache are invalidated.

Notes. No line-by-line operation can be performed at the same time as any full cache operation. For example, to write 32'b1111 CIR, INV_ONE and

The CLR_ONE operation will be masked.

8.2.3 Highable Buffer Configuration Register 0~3 (CRCR)

The cacheable region configuration registers are used to configure the address space range of cacheable (Cacheable). Four cacheable region configuration registers are provided to configure four different cacheable regions.

31	10	9	6	5	1	0
Base Address		Reserved		Size		EN

Figure 8.3: Elevated Buffer Configuration Registers

Base Address-The base address of the cacheable area:

- The size of the buffer zone is configurable from 4KB to 4GB, this field indicates the high bit of the address of the buffer zone, for example, if you set the page size to 8M, the CRCR[22:12] must be 0. For the specific requirements of each page of the zone of different sizes, please refer to [Table 8.2.](#) size-Size of the buffer zone:
- The size of the buffer can be calculated by the formula: Buffer Size = $2^{(\text{Size}+1)}$. Therefore, the range of Size can be set to 01011. to 11111, some other values can cause unpredictable results.

Table 8.2: Configuration of the size of the buffer zones and their requirements for the base site.

Size	Cache size	Requirements for the base address
00000-01010	reservations	-
01011	4KB	Not required
01100	8KB	CRCR.bit[12]=0
01101	16KB	CRCR.bit[13:12]=0
01110	32KB	CRCR.bit[14:12]=0
01111	64KB	CRCR.bit[15:12]=0
10000	128KB	CRCR.bit[16:12]=0
10001	256KB	CRCR.bit[17:12]=0
10010	512KB	CRCR.bit[18:12]=0
10011	1MB	CRCR.bit[19:12]=0
10100	2MB	CRCR.bit[20:12]=0
10101	4MB	CRCR.bit[21:12]=0
10110	8MB	CRCR.bit[22:12]=0
10111	16MB	CRCR.bit[23:12]=0
11000	32MB	CRCR.bit[24:12]=0
11001	64MB	CRCR.bit[25:12]=0
11010	128MB	CRCR.bit[26:12]=0
11011	256MB	CRCR.bit[27:12]=0
11100	512MB	CRCR.bit[28:12]=0
11101	1GB	CRCR.bit[29:12]=0
11110	2GB	CRCR.bit[30:12]=0
11111	4GB	CRCR.bit[31:12]=0

EN-able high retardation zone effective bit:

- EN is 0, the high-mooring zone is invalid;

Gentech **E804**

User's Manual the high retarding zone is valid.

XUANTIE 玄铁

8.2.4 Cache Performance Analysis Control Register (CPFCR)

The Cache Profiling Control Register is used to enable the Cache Profiling function and reset the performance profiling related counters.

31	-	2	1	0
		PFRST		
Reset	-	PFEN		
		0	0	

Figure 8.4: Cache Performance Analysis Control Registers

PFRST-Cache profiling reset bit:

- Resets CPFATR/CPFMTR to 0 when PERST is 1. PFEN-

Cache profiling enable bit:

- PFEN is 0, the cache performance analysis function is off;
- PFEN is 1, the cache performance analysis function is in operation.

8.2.5 Cache Access Count Register (CPFATR)

The Cache Access Count Register is used to record the number of times the cache has been accessed.

31	0
Cache Access Times	

Figure 8.5: Cache Access Count Register

Cache Access Times - Cache access times:

- Records the number of times the cache has been accessed, and every (256) cache accesses will cause the register value to be increased by one if Cache Performance Analysis is enabled.

8.2.6 Cache Miss Count Register (CPFMTR)

The cache miss count register is used to record the number of misses in the access cache.

31	0
Cache Access Times	

Figure 8.6: Cache Miss Count Register

Cache Miss Times - Access cache miss times:

User's Manual

- Records the number of misses in the access cache, and with Cache Performance Analysis enabled, each (256) cache miss will cause the register value to be incremented by one.

Note: It is recommended that all cache configuration operations be performed with the cache turned off.

Chapter 9 Bus Matrix and Bus Interfaces

9.1 Introduction

The E804 implements a multi-bus interface that includes a system bus, an instruction bus, and a data bus, respectively.

The bus matrix provides interconnections for internal processor requests to access external bus interfaces. The bus matrix is connected to the CPU internal requests and bus interfaces as shown Figure 9.1. The bus matrix distributes internal processor accesses to the system bus, instruction bus, or data bus, depending on the address of the memory access to arbitrate the type of bus interface.

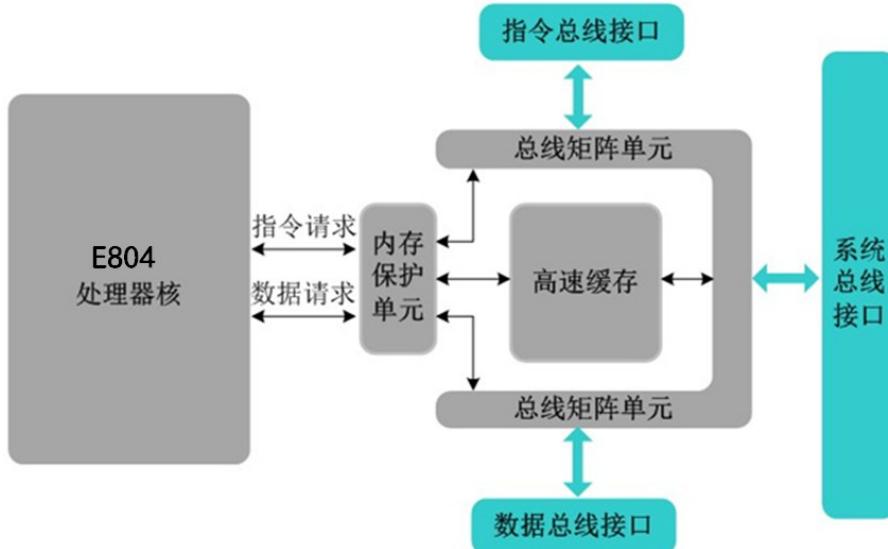


Figure 9.1: E804 Bus Matrix

Finger fetch accesses and data accesses within the processor have the same bus access rights to all bus interfaces. In order to solve the problem of finger access and data access competing for the same bus interface in the same clock cycle, the bus matrix is also responsible for prioritizing the requests. When a finger request and a data request compete for the same bus interface, the data request has a higher priority.

The basic information and configurability of the E804 Multibus Interface is shown Table 9.1.

Table 9.1: Basic information and configurability of the multi-bus interface

bus interface	configurable	bus protocol	chronological method
system bus	Fixed included, protocols can be matched	AHB	direct output
system bus	Fixed included, protocols can be matched	AHB-Lite	direct output
command bus	Fixed Inclusion	AHB-Lite	direct output
data bus (computer)	Fixed Inclusion	AHB-Lite	direct output

In addition, the E804 supports hardware configurability of the base address and space size for the base address and space size of the instruction bus and data bus by providing a set of interface signals (`pad_bmu_xahbl_base` and `pad_bmu_xahbl_mask`). Among them, `pad_bmu_xahbl_base` specifies the base address of the instruction bus/data bus. `pad_bmu_xahbl_base` is a signal with a bit width of 20, which represents the true base address that needs to be extended by the low bit 0, i.e. `{pad_bmu_xahbl_base[19:0],12'b0}`. `pad_bmu_xahbl_mask` specifies the need for address alignment under different address spaces. The address space for the instruction bus or data bus 4KB to 4GB is configurable, e.g. to set the address space size to 8M, `pad_bmu_xahbl_base[10:0]` must be `11'b0`, `pad_bmu_xahbl_mask[19:0]` must be `20'b1111 1111 1000 0000 0000`, see Table 9.2 for specific requirements for different sizes of address space.

Table 9.2: Base Address and Address Alignment Requirements for the Instruction and Data Buses

Address space size	Request for <code>pad_bmu_xahbl_base</code>	Request for <code>pad_bmu_xahbl_mask</code>
4KB	Not required	<code>bit[19:0]=20'b1111 1111 1111 1111 1111 1111</code>
8KB	<code>bit[0]=1'b0</code>	<code>bit[19:0]=20'b1111 1111 1111 1111 1111 1110</code>
16KB	<code>bit[1:0]=2'b0</code>	<code>bit[19:0]=20'b1111 1111 1111 1111 1111 1100</code>
32KB	<code>bit[2:0]=3'b0</code>	<code>bit[19:0]=20'b1111 1111 1111 1111 1111 1000</code>
64KB	<code>bit[3:0]=4'b0</code>	<code>bit[19:0]=20'b1111 1111 1111 1111 1111 0000</code>
128KB	<code>bit[4:0]=5'b0</code>	<code>bit[19:0]=20'b1111 1111 1111 1110 0000</code>
256KB	<code>bit[5:0]=6'b0</code>	<code>bit[19:0]=20'b1111 1111 1111 1100 0000</code>
512KB	<code>bit[6:0]=7'b0</code>	<code>bit[19:0]=20'b1111 1111 1111 1000 0000</code>
1MB	<code>bit[7:0]=8'b0</code>	<code>bit[19:0]=20'b1111 1111 1111 0000 0000</code>
2MB	<code>bit[8:0]=9'b0</code>	<code>bit[19:0]=20'b1111 1111 1110 0000 0000</code>
4MB	<code>bit[9:0]=10'b0</code>	<code>bit[19:0]=20'b1111 1111 1100 0000 0000</code>
8MB	<code>bit[10:0]=11'b0</code>	<code>bit[19:0]=20'b1111 1111 1000 0000 0000</code>
16MB	<code>bit[11:0]=12'b0</code>	<code>bit[19:0]=20'b1111 1111 0000 0000 0000</code>

32MB User's Manual	bit[12:0]=13'b0	bit[19:0]=20'b1111 1110 0000 0000 0000 0000
64MB	bit[13:0]=14'b0	bit[19:0]=20'b1111 1100 0000 0000 0000
128MB	bit[14:0]=15'b0	bit[19:0]=20'b1111 1000 0000 0000 0000
256MB	bit[15:0]=16'b0	bit[19:0]=20'b1111 0000 0000 0000 0000
512MB	bit[16:0]=17'b0	bit[19:0]=20'b1110 0000 0000 0000 0000
1GB	bit[17:0]=18'b0	bit[19:0]=20'b1100 0000 0000 0000 0000
2GB	bit[18:0]=19'b0	bit[19:0]=20'b1000 0000 0000 0000 0000
4GB	bit[19:0]=20'b0	bit[19:0]=20'b000000 0000 0000 0000 0000

9.2 system bus interface

9.2.1 specificities

The E804's system bus interface can be configured to support the AMBA2.0 AHB protocol (in this case refer to the AMBA 2.0 specification - AMBA™ Specification Rev 2.0) or the AMBA3.0 AHB-Lite protocol (in this case refer to the AMBA 3.0 specification - AMBA3 AHB-Lite Protocol Specification Rev 1.0) -AMBA3 AHB-Lite Protocol Specification Rev 1.0)

The basic features of the system bus interface include:

- Supports AMBA2.0 AHB or AMBA3.0 AHB-Lite bus protocols, configurable by the user;
- Only direct output (non-flop-out) is supported;
- The processor and system clock frequency ratio must be 1:1;

9.2.2 Content of the agreement

9.2.2.1 Supported Transfer Types

Considering the application area and the cost of the E804, the system bus interface implements only parts of the AHB/AHB-Lite protocol. Under the AHB-Lite protocol, the transmission types supported by the bus interface as a master device are:

- HBURST supports SINGLE and WRAP4 transfers, all other burst types are not supported;
- HTRANS supports IDLE and NONSEQ and SEQ, all other transport types are not supported;
- HSIZE supports word, byte, and half-word transfers; other transfer sizes are not supported;
- HWRITE supports both read and write operations.

Under the AHB protocol, as a master device, the transmission types supported by the bus interface are:

- HBURST supports SINGLE and WRAP4 transfers, all other burst types are not supported;
- HTRANS supports IDLE, NONSEQ, and SEQ; all other transport types are not supported;
- HSIZE supports word, byte, and half-word transfers; other transfer sizes are not supported;
- HWRITE supports both read and write operations.

9.2.2.2 Supported Response Types

Under the AHB and AHB-Lite protocols, the bus interface accepts the response type of the slave device as:

- HREADY supports Ready and Not Ready;
- HRESP supports OKAY and ERROR, other response types are not supported.

9.2.3 Behavior with different bus responses

Table 9.3 lists CPU behavior when different bus responses occur on the bus.

Table 9.3: Bus Exception Handling

HREADY	HRESP	in the end
not care	ERROR	access error, the bus ends the transmission and handles the access error.
High	OKEY	End of transmission.
Low	OKEY	Insert the wait state.

9.2.4 Interface signals for AHB protocol

Table 9.4: AHB Protocol Interface Signals

signal name	I/O	Reset	define
biu_pad_haddr[31:0]	O	-	Address bus: 32-bit address bus.
biu_pad_hwdata[31:0]	O	-	Write data bus: 32-bit write data bus.
biu_pad_hburst[2:0]	O	-	Burst transmission indication signal: 000: SINGLE; 001: INCR; 010: WRAP4. The E804 supports SINGLE and WRAP4 burst transmissions.
biu_pad_hsize[2:0]	O	-	Transmission width indication signal: 000: byte; 001: halfword; 010: word.
biu_pad_htrans[1:0]	O	00	The transmission type indicates the signal: 00: IDLE; 01: BUSY; 10: NONSEQ; 11: SEQ. The E804 supports IDLE, NONSEQ, and SEQ transmission types.
biu_pad_hwrite	O	0	Read and write representation signals: 1: The instruction is to write the bus transfer; 0: Indicates that it is a read bus transfer.

Continued on next page

Table 9.4 - continued from previous page

signal name	I/O	Reset	define
biu_pad_hprot[3:0]	O	-	Protection control signal: ***0: Fetch command; ***1: Data access; **0*: User access; **1*: Superuser access; *0**: Not bufferable; *1**: bufferable; 0***: Not cacheable; 1***: cacheable.
biu_pad_hbusreq	O	0	Bus request signal: Indicates that the CPU is requesting bus access.
biu_pad_hlock	O	0	Locks the bus signals: When lock is asserted, the CPU has exclusive control of the bus and no other busmaster can occupy the bus.
pad_biu_hrdata[31:0]	I	-	Read data bus: 32-bit read data bus.

Table 9.5: AHB Protocol Interface Signals

signal name	I/O	Reset	define
pad_biu_hready	I	-	Transmission completion indication signal: When valid, it indicates that the current transfer is complete, and the CPU places the bus in the Standby status.
pad_biu_hgrant	I	-	Bus occupancy indication signal: When valid, indicates that the CPU is currently occupying the bus.
pad_biu_hresp[1:0]	I	-	Transmits an answer signal: 00: OKAY; 01: ERROR; 10: RETRY; 11: SPLIT. The E804 supports only OKAY and ERROR response types.

9.2.5 Interface signals for the AHB-Lite protocol

Table 9.6: AHB-Lite Protocol Interface Signals

signal name	I/O	Reset	define
biu_pad_haddr[31:0]	O	-	Address bus: 32-bit address bus.
biu_pad_hwdata[31:0]	O	-	Write data bus: 32-bit write data bus.
biu_pad_hburst[2:0]	O	-	Burst transmission indication signal: 000: SINGLE; 001: INCR; 010: WRAP4. The E804 supports SINGLE and WRAP4 burst transmissions.
biu_pad_hsize[2:0]	O	-	Transmission width indication signal: 000: byte; 001: halfword; 010: word.
biu_pad_htrans[1:0]	O	00	The transmission type indicates the signal: 00: IDLE; 01: BUSY; 10: NONSEQ; 11: SEQ. The E804 supports IDLE, NONSEQ, and SEQ transmission types.
biu_pad_hwrite	O	0	Read and write representation signals: 1: The instruction is to write the bus transfer; 0: Indicates that it is a read bus transfer.
biu_pad_hprot[3:0]	O	-	Protection control signal: ***0: Fetch command; ***1: Data access; **0*: User access; **1*: Superuser access; *0**: Not bufferable; *1**: bufferable; 0***: Not cacheable; 1***: cacheable.
pad_biu_hrdata[31:0]	I	-	Read data bus: 32-bit read data bus.

Table 9.7: AHB-Lite Protocol Interface Signals

signal name	I/O	Reset	define
pad_biu_hready	I	-	Transmission completion indication signal: When valid, it indicates that the current transfer is complete, and the CPU places the bus in the Standby status.
pad_biu_hresp[1:0]	I	-	Transmits an answer signal: 00: OKAY; 01: ERROR; 10: RETRY; 11: SPLIT. The E804 supports only OKAY and ERROR response types.

9.3 command bus interface

9.3.1 specificities

The E804's command bus only supports the AMBA3.0 AHB-Lite protocol, which can be found in the AMBA 3.0 Specification - AMBA3 AHB-Lite Protocol Specification Rev 1.0.

The basic features of the command bus interface are:

- Compatible with AMBA 3.0 AHB-Lite bus protocol;
- Only the direct output (Non-Flop-out) method is supported;

9.3.2 Content of the agreement

9.3.2.1 Supported Transfer Types

Considering the application area and the cost of the E804, the command bus interface only partially implements the AHB-Lite protocol. Under the AHB-Lite protocol, as a master device, the bus interface supports the following transmission types:

- HBURST only supports SINGLE transfers, no other burst types are supported;
- HTRANS only supports IDLE and NONSEQ, no other transport types are supported;
- HSIZE supports word, byte, and half-word transfers; other transfer sizes are not supported;
- HWRITE supports both read and write operations.

9.3.2.2 Supported Response Types

Under the AHB-Lite protocol, the bus interface accepts the response type of the slave device as:

User's Manual

- HREADY supports Ready and Not Ready;
- HRESP supports OKAY and ERROR, other response types are not supported.

9.3.3 Behavior with different bus responses

Table 9.8 lists CPU behavior when different bus responses occur on the bus.

Table 9.8: Bus Exception Handling

HREADY	HRESP	in the end
not care	ERROR	access error, the bus ends the transmission and handles the access error.
High	OKEY	End of transmission.
Low	OKEY	Insert the wait state.

9.3.4 Command Bus Interface Signal

Table 9.9: Command Bus Interface Signals

signal name	I/O	Reset	define
iahbl_pad_haddr[31:0]	O	-	Address bus: 32-bit address bus.
iahbl_pad_hburst[2:0]	O	-	Burst transmission indication signal: 000: SINGLE; 001: INCR; 010: WRAP4. The E804 supports only SINGLE burst transmission.
iahbl_pad_hprot[3:0]	O	-	Protection control signal: ***0: Fetch command; ***1: Data access; **0*: User access; **1*: Superuser access; *0**: Not bufferable; *1**: bufferable; 0***: Not cacheable; 1***: cacheable.
iahbl_pad_hsize[2:0]	O	-	Transmission width indication signal: 000: byte; 001: halfword; 010: word.
iahbl_pad_htrans[1:0]	O	00	The transmission type indicates the signal: 00: IDLE; 01: BUSY; 10: NONSEQ; 11: SEQ. The E804 supports only the IDLE and NONSEQ transport classes. Type.
iahbl_pad_hwdata[31:0]	O	-	Write data bus: 32-bit write data bus.
iahbl_pad_hwwrite	O	0	Read and write representation signals: 1: The instruction is to write the bus transfer; 0: Indicates that it is a read bus transfer.
pad_iahbl_hrdata[31:0]	I	-	Read data bus: 32-bit read data bus.

Table 9.10: Command Bus Interface Signals

signal name	I/O	Reset	define
pad_iahbl_hready	I	-	Transmission completion indication signal: When valid, it indicates that the current transfer is complete and the CPU will total Line placed on standby.
pad_iahbl_hresp	I	-	Transmits an answer signal: 0: OKAY; 1: ERROR.
pad_bmu_iahbl_base[19:0]	I	-	IAHB-Lite Base Address Control Signal, after power-on reset need to be fixed
pad_bmu_iahbl_mask[19:0]	I	-	IAHB-Lite Address Alignment Control Signal, Power-On Reset Afterwards, it needs to be fixed.

9.4 data bus interface

9.4.1 specificities

The E804's data bus only supports the AMBA3.0 AHB-Lite protocol, refer to the AMBA 3.0 Specification - AMBA3 AHB-Lite Protocol Specification Rev 1.0.

The basic features of the data bus interface are:

- Compatible with AMBA 3.0 AHB-Lite bus protocol;
- Only the direct output (Non-Flop-out) method is supported.

9.4.2 Content of the agreement

9.4.2.1 Supported Transfer Types

Considering the application area and the cost of the E804, the data bus interface only partially implements the AHB-Lite protocol. Under the AHB-Lite protocol, the transmission types supported by the bus interface as a master device are:

- HBURST supports SINGLE transmission, no other burst types are supported;
- HTRANS only supports IDLE and NONSEQ, no other transport types are supported;
- HSIZE supports word, byte, and half-word transfers; other transfer sizes are not supported;
- HWRITE supports both read and write operations;

9.4.2.2 Supported Response Types

Under the AHB-Lite protocol, the bus interface accepts the response type of the slave device as:

- HREADY supports Ready and Not Ready;
- HRESP supports OKAY and ERROR, other response types are not supported.

9.4.3 Behavior with different bus responses

Table 9.11 lists CPU behavior when different bus responses occur on the bus:

Table 9.11: Bus Exception Handling

HREADY	HRESP	in the end
not care	ERROR	access error, the bus ends the transmission and handles the access error.
High	OKEY	End of transmission.
Low	OKEY	Insert the wait state.

9.4.4 data bus interface signal

Table 9.12: Data Bus Interface Signals

signal name	I/O	Reset	define
dahbl_pad_haddr[31:0]	O	-	Address bus: 32-bit address bus.
dahbl_pad_hburst[2:0]	O	-	Burst transmission indication signal: 000: SINGLE; 001: INCR; 010: WRAP4. The E804 supports only SINGLE burst transmission.
dahbl_pad_hprot[3:0]	O	-	Protection control signal: ***0: Fetch command; ***1: Data access; **0*: User access; **1*: Superuser access; *0**: Not bufferable; *1**: bufferable; 0***: Not cacheable; 1***: cacheable.
dahbl_pad_hsize[2:0]	O	-	Transmission width indication signal: 000: byte; 001: halfword; 010: word.
dahbl_pad_htrans[1:0]	O	00	The transmission type indicates the signal: 00: IDLE; 01: BUSY; 10: NONSEQ; 11: SEQ. The E804 supports only IDLE and NONSEQ transmission types.
dahbl_pad_hwdata[31:0]	O	-	Write data bus: 32-bit write data bus.
dahbl_pad_hwrite	O	0	Read and write representation signals: 1: The instruction is to write the bus transfer; 0: Indicates that it is a read bus transfer.
pad_dahbl_hrdata[31:0]	I	-	Read data bus: 32-bit read data bus.

Table 9.13: Data Bus Interface Signals

signal name	I/O	Reset	define
pad_dahbl_hready	I	-	Transmission completion indication signal: When valid, it indicates that the current transfer is complete and the CPU will total Line placed on standby.
pad_dahbl_hresp	I	-	Transmits an answer signal: 0: OKAY; 1: ERROR.
pad_bmu_dahbl_base[19:0]	I	-	DAHB-Lite Base Address Control Signal, Power-On Reset post-fixing
pad_bmu_dahbl_mask[19:0]	I	-	DAHB-Lite Address Alignment Control Signal, Power-up Reset Fixed after the position

9.5 Order of access to instructions and data

Once the CPU is configured with a cache, all non-cacheable data bypasses the cache and accesses the corresponding bus directly from the bus matrix, as shown in Channel 1 in Figure 9.2;

The cache is accessed first, and if there is a hit, the data is retrieved directly from the cache; if there is no hit, the cache makes a request to the bus through the bus matrix, as shown in Channel 2 in Figure 9.2.

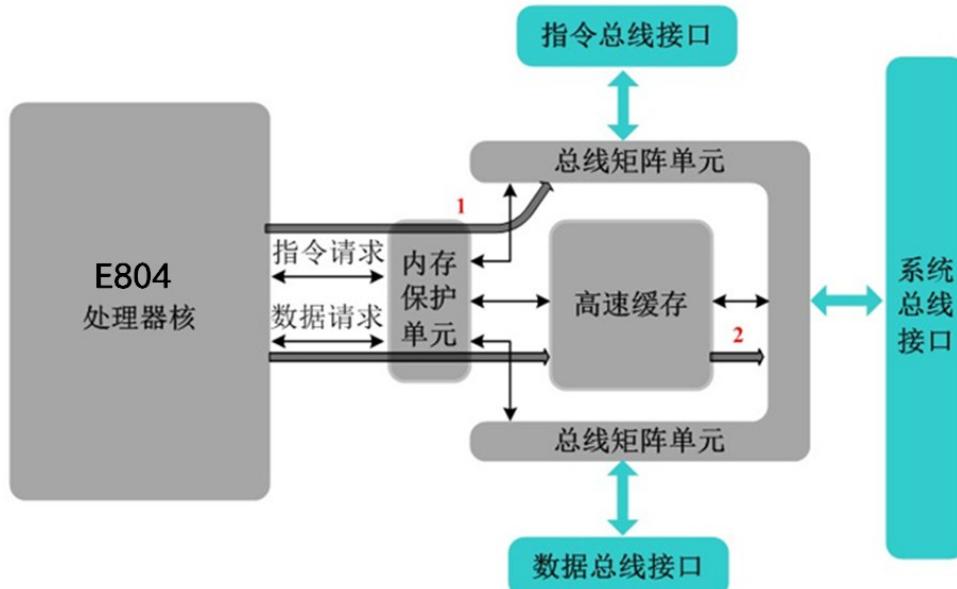


Figure 9.2: Accessing the External Bus Sequence

Chapter 10 Debugging Interface

10.1 summarize

The debug interface is the channel through which the software interacts with the processor. Through the debug interface, the user can obtain information about the CPU's registers and memory contents, including other on-chip device information. In addition, operations such as program download can also be done through the debug interface.

In order to meet the needs of low-cost applications and to save CPU external pins, CSKY defines a set of debugging interfaces, JTAG2 interface. JTAG2 debugging interfaces include JTAG2 communication protocol, JTAG2 interface controller. E804 supports 2-wire JTAG protocol, and the debugging interfaces use the JTAG2 protocol to communicate with the external debugger.

The main features of the debugging interface are as follows:

- Supports 2-wire JTAG interface;
- Get CPU status non-intrusively;
- Supports both synchronous and asynchronous debugging, ensuring that the processor is put into debugging mode under extreme adverse conditions;
- Soft breakpoints are supported;
- Multiple memory breakpoints can be set;
- Checks and sets CPU register values;
- Check and change memory values;
- Single-step or multi-step execution of instructions is possible;
- Quickly download the program;
- Debug mode can be entered after CPU reset or in normal user mode;
- Debug register resources can be accessed using the TCIP interface, as detailed in the debug manual.

The debugging work of C-SKY CPU is accomplished by the debugging software, debugging agent service program, debugger and debugging interface, and the location of debugging interface in the whole CPU debugging environment is shown [Figure 10.1](#). The location of the debugging interface in the whole CPU debugging environment is shown in Figure 10.1. The debugging software and the debugging agent service program are interconnected through the network, the debugging agent service program and the debugger are connected through USB, and the debugger communicates with the debugging interface of the CPU in JTAG mode.

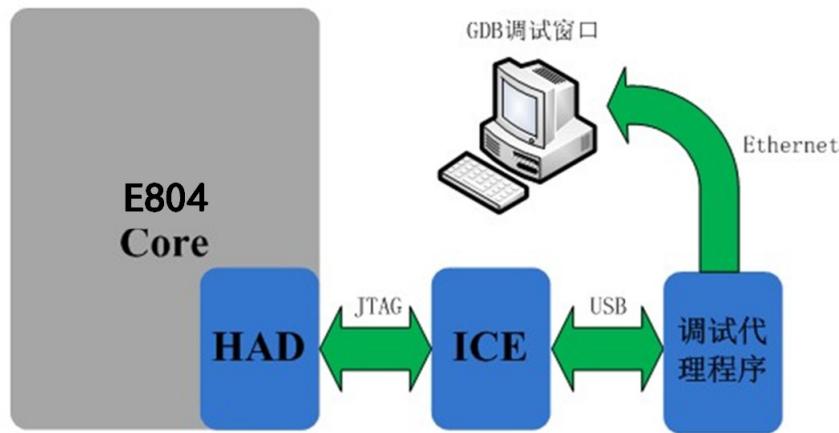


Figure 10.1: Location of the Debug Interface in the Overall CPU Debug Environment

10.2 external interface

The interfaces between the debug module and the outside world are mainly JTAG-related interface signals and debug-related interface signals. [Table 10.1](#) lists the debug-related interface signals.

Table 10.1: External Interface Signals for the Commissioning Module

signal name	orientations
(sysio_pad_dbg_b)	exports
pad_had_jdb_req_b	importation
had_pad_jdb_ack_b	exports
pad_had_jtg_tap_en	importation
had_pad_jtg_tap_on	exports
had_pad_jdb_pm[1:0]	exports
pad_had_jtg_tclk	importation
pad_had_jtg_trst_b	importation
pad_had_jtg_tms_i	importation
had_pad_jtg_tms_o	exports
had_pad_htg_tms_oe	exports

1. **biu_pad_dbg_b (sysio_pad_dbg_b)**

A low level indicates that the CPU is in debug mode.

2. **pad_had_jdb_req_b with had_pad_jdb_ack_b**

The pad_had_jdb_req_b signal is a request signal to allow the CPU to enter the debug state asynchronously. This signal must be held low for at least two JTAG clock cycles in order to ensure that the CPU enters the debug state and that the CPU is able to debug the program. If this signal is held low for less than two JTAG clock cycles, the CPU may enter the debug state but cannot debug the program because the

User's Manual

The signal is also used to enable the TAP state machine in the debug interface.

After the CPU enters the debug state, the had_pad_jdb_ack_b signal is held low for two JTAG clock cycles as an answer.

3. pad_had_jtg_tap_en with had_pad_jtg_tap_on

The pad_had_jtg_tap_en signal is the enable signal for the TAP state machine in the debug interface. Holding this signal high for at least one JTAG clock cycle enables the TAP state machine in the debug interface. If this signal remains invalid after the CPU has been powered up, it may not be possible to debug the program when the CPU enters the debug state using synchronization methods (e.g. setting the DR bit in the debug interface register HCR, breakpoints, etc.).

The had_pad_jtg_tap_on signal will be pulled high after the TAP state machine has started.

4. had_pad_jdb_pm[1:0]

The had_pad_jdb_pm[1:0] signals indicate the current working mode of the CPU, and can be used to determine whether the CPU has entered the debug mode. The details are shown [Table 10.2](#).

Table 10.2: had_pad_jdb_pm Indicates Current CPU Status

had_pad_jdb_pm[1:0]	show
00	Normal mode.
01	Low power consumption modes (STOP, DOZE, WAIT)
10	Debug Mode.
11	Reservations.

5. pad_had_jtg_tclk

JTAG clock signal. This signal is an external input signal, usually the clock signal generated by the debugger, to ensure that the frequency of the clock signal is lower than the CPU clock signal frequency 1/2 to ensure the normal operation between the debug module and the CPU core.

6. pad_had_jtg_trst_b

The pad_had_jtg_trst_b signal is a JTAG reset signal that resets the TAP state machine and other related control signals.

7. JTAG_2 Related Signals

The pad_had_jtg_tms_i signal is a 2-wire JTAG serial data input signal, and the debug interface is on the JTAG clock signal.

It is sampled on the rising edge of TCLK and the external debugger sets the signal on the falling edge of the JTAG clock;

This signal must be held high at idle, and the clock signal should preferably be stopped at idle. This signal can be used to synchronously reset the HAD logic: in debug modules implementing a 2-wire JTAG interface, if the clock signal is active at all times, the user can synchronously reset the debug module by holding this signal high for 80 clock cycles. After resetting the debug

User's Manual
module, the TAP state machine of the debug module returns to the RESET state and the
debug module register HACR is reset to 0x82 (pointing to the ID register)

The had_pad_jtg_tms_o signal is a 2-wire JTAG serial data output signal, and the debug interface is on the
JTAG clock signal.

It is set on the falling edge of TCLK and the external debugger samples it on the rising edge of the JTAG clock;

The had_pad_jtg_tms_oe signal is a valid indication signal for the had_pad_jtg_tms_o signal,
which should be used externally by the CPU to combine the pad_had_jtg_tms_i and
had_pad_jtg_tms_o signals into one bidirectional port signal.

Chapter 11 Working modes and transitions

The E804 has a total of three types of operating modes: normal operation mode, low-power operation mode, and debug mode. The low-power operation modes are divided into three types: STOP mode, DOZE mode, and WAIT mode, which are different from each other as defined by the SOC designer. This chapter describes in detail the operating modes of the CPU and the conversion between modes.

As shown in Figure 11.1, there are three types of operating modes for the E804, i.e., normal operating mode, low-power operating mode, and debugging mode, and which operating mode the CPU is in can be obtained by querying the sysio_pad_jdb_pm[1:0] signals.

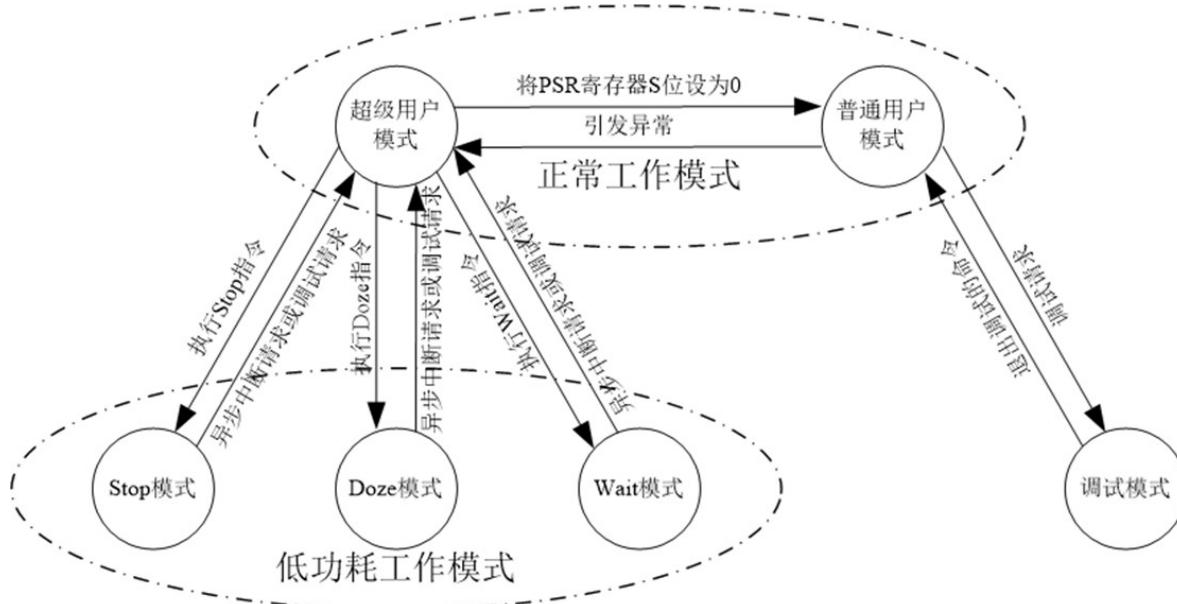


Figure 11.1: Schematic diagram of the various operating states of the CPU

11.1 Normal Operating Mode

The normal operating modes of the CPU can be categorized into two, Super User Mode and Normal User Mode. Which normal operating mode the CPU is in is obtained by querying the S bit in the PSR register.

When the S bit is 1, the CPU operates in Super User mode; the S bit is 0, the CPU operates in Normal User mode.

You can enter normal user mode by setting the S bit to 0 when the CPU is operating in super user mode, and enter super user mode by raising an exception when the CPU is operating in normal user mode.

11.2 Low Power Mode

the CPU executes the low-power instructions (STOP, DOZE, WAIT), the CPU will enter the low-power mode. The process of executing the three instructions is that the CPU will wait for all the previous instructions to be executed after executing the low-power instruction, and then complete the low-power instruction, and at the same time pull up the sysio_pad_lpmd_b[1:0] signals according to the type of the low-power instruction, stop executing the instruction and freeze the pipeline, and turn off the internal clock.

Only asynchronous interrupt requests (pad_sysio_intraw_b and pad_sysio_finraw_b signals) or debug requests can cause the CPU to exit the low-power mode, and then the CPU continues executing subsequent instructions from the low-power instruction that entered the low-power mode. Which low-power mode the current CPU is operating in can be obtained by querying the signal lines sysio_pad_lpmd_b[1:0], and the specific low-power scenarios corresponding to each mode are determined by the SOC designer.

11.3 debug mode

11.3.1 debug mode

When the CPU enters the debugging mode, it stops fetching fingers and executing instructions, and is in a waiting state. In this state, the CPU only executes the instructions inputted from the HAD, and through the instructions inputted from the HAD, it can query and modify the state of the CPU, and then carry out debugging.

11.3.2 Entering Debug Mode

When the CPU receives a debug request, it enters the debug mode, in which there can be
following seven debug request sources:

- The external input signal (pad_had_jdb_req_b) to the HAD allows the processor to enter debug mode asynchronously;
- When the IDRE bit of the E804 HCR is valid, the external input signal to the HAD biu_had_sdb_req_b (pad_biu_dbgreq_b) Requests the processor to synchronize into debug mode after being synchronized by the CPU;
- When the ADR bit of the E804 HCR is active, the processor enters debug mode directly;
- When the DR bit of the E804 HCR is active, the processor enters debug mode after completing the current instruction;
- When the FDB bit of the E804 CSR is valid, the processor enters debug mode while executing the bkpt instruction;
- When the TME bit of the E804 HCR is active, the processor enters debug mode the value of the trace counter is

reduced to zero;

- Store hardware breakpoints in the E804 to enter debug mode.

The processor finishes executing the current instruction, saves the pipeline information, and then enters debug mode. When the processor is in the low-power mode, the processor can exit the low-power mode and enter the debug mode by setting the ADR in the HCR and the debug request of the DR. After entering the debug mode, the CPU stops executing the current instruction and waits for the user to input a valid instruction through the debugging interface to execute or exit the debug mode.

11.3.3 Exit debug mode

If the GO and EX bits of the E804 HACR are set to 1 and R/W is 0 (write operation), and the RS selection is WBBR, PSR, PC, IR, CSR, or Bypass register, the CPU exits the debug mode and enters the normal operation mode when the instruction is executed.

Note: Since PC, CSR, and PSR are variable in debug mode, when exiting debug mode, the values in the above registers must be the same as the values saved when you first entered debug mode.

Chapter 12 Floating Point Processing Unit

12.1 summarize

12.1.1 summary

The Floating Point Unit, a configurable hardware unit of the E804, is designed to enhance the processing power of the E804 for floating point applications. The E804 Floating Point Unit provides a low-cost, high-performance hardware floating point implementation.

The floating-point unit supports single-precision floating-point operations in the IEEE-754 floating-point standard and implements 16 single-precision floating-point registers. With system software support, the E804 supports double-precision floating-point operations.

12.1.2 specificities

The main features of the architecture and programming model of the floating point unit are as follows:

- Fully compatible with ANSI/IEEE Std 754 floating point standard (with system software support);
- Only single-precision floating-point operations are supported;
- Four rounding methods are supported: rounding to zero, rounding to positive infinity, rounding to negative infinity, and rounding to proximity;
- Supports both caught and uncaught handling modes for floating point exceptions;
- Supports precise handling of floating point exceptions;
- Floating-point hardware division and square opening are supported.

The main features of the microarchitecture of the floating point unit are as follows:

- 16 independent single-precision floating-point registers;
- Single-emitter architecture, processing one floating-point arithmetic instruction per cycle;
- Supports sequential launch, sequential execution, and sequential write-back of floating-point arithmetic instructions;
- Contains three separate execution pipelines for floating-point ALU, floating-point multiplication, and floating-point division and squaring;
- Optimized execution delay technology, except for floating-point divide-and-square instructions, can be executed in 1-2 clock cycles;
- Cost optimization techniques based on multiplexing of computing components;

- Power consumption optimization technique based on gated clock and datapath isolation.

12.2 microarchitecture

12.2.1 Introduction to Floating Point Units

The floating-point unit consists of the following three main execution subunits:

- Arithmetic Logic (FALU) Unit
- Multiplication (FMULU) unit
- Divide and Square (FDSU) Unit

The arithmetic logic unit (FALU) unit is responsible for the execution of instructions such as addition, subtraction, comparison, conversion, inversion, absolute value, register transfer, etc., and the results of the operations are completed in 1-2 cycles.

The multiplication unit (FMULU) is responsible for operations such as multiplication and multiplication-inversion, and the multiplication result is completed in 2 cycles.

The FDSU implements division, square, and reciprocal operations, generating 1 bit of result per cycle and taking up to 28 cycles to complete.

12.3 programming model

12.3.1 brief

The E804 floating-point unit implements the single-precision portion of the ANSI/IEEE Std 754 floating-point standard.

- Supports loading of floating-point immediate numbers to improve the efficiency of floating-point immediate number generation;

12.3.2 data format

The floating-point unit supports only single-precision data formats. The value of the floating-point register can be assigned in three ways: the floating-point load instruction, the floating-point constant load instruction, and the floating-point register transfer instruction. Data in a floating-point register has the same format as data in memory. Any transfer or load does not change this format.

Recording the type of data stored in each register is guaranteed by the program itself. As a result, the hardware will not make any judgment about the type of data stored in the registers. For example, even if the data stored in a floating-point register is an integer, a floating-point instruction will only treat the data as single-precision. This means that the hardware's interpretation of the data depends only on the instruction being executed.

12.3.2.1 Plasticity Data Format

The floating-point unit supports signed or unsigned 32-bit integers. The format of the integers in the floating-point registers is the same as in the E804 main pipeline registers or in memory.

12.3.2.2 Single precision data format

Figure 12.1 represents the region splitting of a single-precision floating-point number:



Figure 12.1: Single-Precision Data Format

Single precision data formats include:

- Sign bit (S) bit[31];
- The exponential domain of the bias: The biased exponent $e=E+bias$, here bias=127, bits[30:23];
- Fractional domain: the fraction, bits[22:0].

12.3.3 Floating Point Register Description

The E804 floating-point unit has a separate floating-point register consisting of 16 32-bit single-precision registers.

Figure 12.2 shows the E804 register resource map containing the floating-point unit. Compared to the shaping pipeline, the floating-point unit adds floating-point registers that can be accessed in both normal and privileged modes. Three system registers have been added and are only accessible in privileged mode.

12.3.3.1 Transferring data with general-purpose registers

The transfer of data between general purpose registers and floating point registers can be accomplished through floating point register transfer instructions. Floating point register transfer instructions include:

- FMTVRL Floating-point register write transfer instruction.
- FMFVRL Floating Point Register Read Transfer Instruction.

12.3.3.2 Maintain consistent register accuracy

Floating-point registers can store both single-precision floating-point numbers and shaped data. As an example, the type of data stored in floating-point register FR1, depending on the last write operation, could be either of the two data types.

The floating-point unit does not do any data format detection on the data type in hardware, and the hardware's interpretation of the data format in the floating-point register depends only on the executed floating-point instruction itself, and does not care about the data format used for the last write operation in this register. It is entirely up to the compiler or the program itself to ensure the consistency of the precision of the data in the register.

12.3.4 system register

There are three system registers associated with the floating-point unit: Floating-point version register FID (CR<0,2>), Floating-point control register FCR (CR<1,2>), and Floating-



Figure 12.2: Register Resources in Programming Mode

User's Manual

The Floating Point Control Register is used to configure the exception handling and execution of the floating point. The floating-point control register is used to configure the exception handling and execution of the floating-point, while the floating-point exception status register records information about floating-point exceptions during execution.

Table 12.1 lists the three system registers for the floating-point unit.

Table 12.1: Floating-Point System Register List

processor register	CR	access method	access mode
Floating Point Version Register, FID (FPU ID Register)	CR<0,2>	read-only (computing)	privileged mode
Floating Point Control Register, FCR (FPU Control Register)	CR<1,2>	Read/Write	privileged mode
Floating Point Exception Status Register FESR (Exception Status Register)	CR<2,2>	Read/Write	privileged mode

12.3.4.1 Methods for reading and writing floating-point system registers

The MTCR and MFCR instructions are used to read and write the floating point system registers. Since the floating-point system registers are stored in coprocessor , MTCR and MFCR will read and write to coprocessor with the following register indexes: floating-point version register CR<0,2>, floating-point control register CR<1,2>, and floating-point exception status register CR<2,2>.

The instruction to write a floating-point system register is MTCR Rx, CR<y,2>, which rewrites the value of floating-point system register number y with the value of general-purpose register Rx.

The instruction to read the floating-point system register is MFCR Rx, CR<y,2>, which reads the value of floating-point system register and rewrites general-purpose register Rx.

12.3.4.2 Floating Point Version Register (FPU ID Register, FID)

The floating-point version register is a read-only register that identifies the version information of the floating-point unit. Figure 12.3 shows the bit field of the floating-point version register:

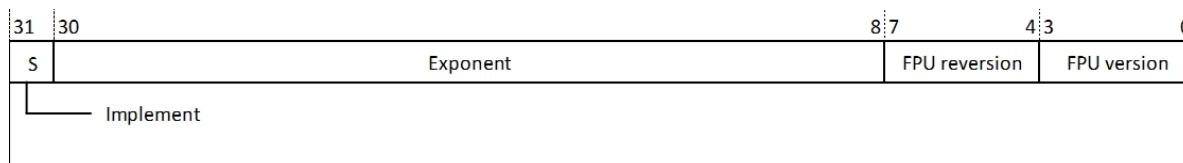


Figure 12.3: Floating-Point Version Registers

The software user can determine whether the current processor is configured with a floating-point unit and the version information by querying the floating-point version

Table 12.2: Floating-Point Version Register Descriptions

classifier for honorific people	functionality	Read/Write	descriptive
[31]	Implement	read-only (computing)	Floating-point unit implementation: 1: Hardware floating point unit implemented; 0: Hardware floating-point unit is not implemented, the processor will simulate it through software. style implementation of floating point functions.
[30:8]	Reserved	read-only (computing)	All 0.
[7:4]	FPU reversion	read-only (computing)	Upgrade information for floating point units.
[3:0]	FPU version	read-only (computing)	Version information of the floating point unit.

12.3.4.3 Floating Point Control Register (FPU Control Register, FCR)

The Floating Point Control Register is read-write and is responsible for configuring the exception care and execution method of the floating point unit. The Floating Point Control Register can only be accessed in Super User mode, and the reserved bits of the FCR are initialized to 0. When it is desired to modify some of these bits, it is recommended that the code read, then modify the value read, and then write it back to the FCR in order to ensure that unwanted bits will not be modified in error. Failure to follow this method is likely to result in unexpected errors in subsequent programs.

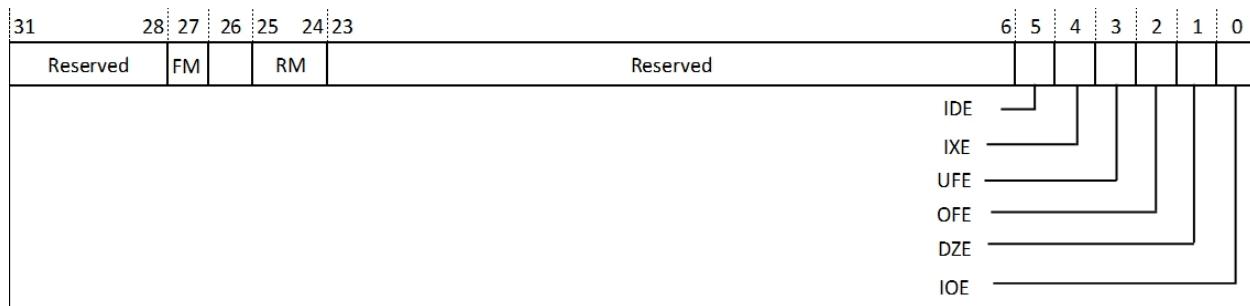


Figure 12.4: Floating Point Control Registers

Reset value: 0x00000000.

The definition of each bit in the FCR register is shown Table 12.3.

Table 12.3: Floating-Point Control Register Descriptions

classifier for honorifi c people	function ality	Read/W rite	descriptive
[31:28]	reservati ons	read-only (computi ng)	All 0

Continued on next page

[27] Gentech E804 User's Manual	FM	Read/Write	This field determines the overflow exception enable bit under Input Unspecified Number Exception Enable Bit (IDE). Result (IDE/OFE) is not turned on for Denormalized Number. Table 12-3, continued from previous page. 1: Brush the unspecified numbers into Signed Minimal Normal Number. 0: Brush the unspecified number to Signed Zero.
[25:24]	RM	Read/Write	This field determines the rounding mode for floating point calculations. 00: Round to the Nearest 01: Round towards 0 10: Round towards positive infinity (Round towards $+\infty$) 11: Round towards negative infinity (Round towards $-\infty$)
[23:6]	reservations	read-only (computing)	All 0
[5]	IDE	Read/Write	Input Denormalized Count Exception Enable bit (IDE, Input Denormalized trap Enable bit) 1: Enable. At this point, if an input off-specification number is encountered, the processor will terminate program execution and throw an off-specification number input exception. 0: Not enabled. In this case, the processor will process the input out-of-specification number and continue arithmetic according to the FM indication bit.
[4]	IXE	Read/Write	Inexact Exception Enable bit (IXE, IneXact trap Enable bit) 1: Enable. In this case, if the result of the operation is inexact, the processor will terminate the program execution and throw an inexact exception without writing back any result. 0: Not enabled. In this case, the processor will round the result of the operation according to the RM indication bit and continue the operation.
[3]	UFE	Read/Write	Underflow Exception Enable bit (UFE, UnderFlow trap Enable bit) 1: Enable. In this case, if the result of the operation overflows, the processor will terminate the program execution and throw an overflow exception without writing back any result. 0: Not enabled. At this point, the processor will process the underflow result and continue the operation according to the FM indication bit.
[2]	OFE	Read/Write	Overflow Exception Enable bit (OFE, OverFlow trap Enable bit) 1: Enable. In this case, if the result of the operation overflows, the processor will terminate program execution and throw an overflow exception without writing back any result. 0: Not enabled. In this case, the processor will process the upper overflow result according to the RM indication bit and continue the operation. Continued on next page
[1]	DZE	Read/Write	Division by Zero Exception Enable bit (DZE, Division by Zero trap Enable bit) 1: Enable. At this point, if a divide-by-zero operation occurs, the processor will terminate program execution and throw a divide-by-zero exception without writing back any results. 0: Not enabled. In this case, the processor will take a positive/negative infinite floating point number as the result of the operation and continue the operation.

Table 12.3 - continued from previous page

[0]	IOE	Read/Write	Illegal Operation Exception Enable bit (IOE, Invalid Operation trap Enable bit) 1: Enable. At this point, if an illegal operand occurs, the processor will terminate program execution and throw an illegal operation exception without writing back any result. 0: Not enabled. In this case, the processor will use QNaN as the result of the operation and continue the operation.
-----	-----	------------	---

12.3.4.4 Floating Point Exception Status Register (FESR)

FESR provides floating-point exception status information. The individual bits indicate the type of floating-point exception that occurred. FESR can only be accessed in superuser mode.

FESR regions are shown [Figure 12.5](#):

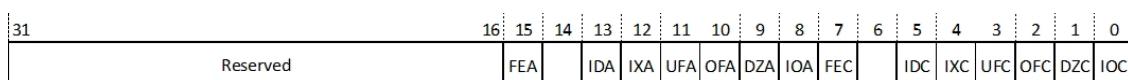


Figure 12.5: Floating Point Exception Status Registers

[Table 12.4](#) shows the bit definitions for the floating-point exception status register.

Table 12.4: Floating Point Exception Status Register Descriptions

classifier for honorific people	functionality	Read/Write	descriptive
[29:16]	reservations	read-only (computing)	All 0.
[15]	FEA	Read/Write	Floating-point anomaly accumulation bit. When any of the floating-point exceptions occurs and the corresponding enable bit of IOE is not turned on, this bit will be set to 1.
[14]	reservations	read-only (computing)	All 0
[13]	IDA	Read/Write	Input Denormalized Accumulative bit (IDA, Input Denormalized Accumulative bit) This bit is set to one when the Input Unspecified Number Exception <input checked="" type="checkbox"/> IDE bit is zero.
[12]	IXA	Read/Write	Inexact Exception Accumulative bit (IXA, IneXact Accumulative bit) This bit is set to one when the non-precision exception <input checked="" type="checkbox"/> IXE bit is zero.

User's Manual	UFA	Read/Write	Underflow Anomaly Accumulative bit (UFA, UnderFlow Accumulative bit) This bit is set to 1 when the underflow exception \boxtimes UFE bit is zero.
[10]	OFA	Read/Write	Overflow Abnormal Accumulative bit (OFA, OverFlow Accumulative bit) This bit is set to 1 when the overflow exception \boxtimes OFE bit is 0.
[9]	DZA	Read/Write	Division by Zero Accumulative bit (DZA, Division by Zero Accumulative bit) This bit is set to 1 when a divide by 0 exception occurs \boxtimes DZE bit is zero.
[8]	IOA	Read/Write	Illegal Operand Accumulative bit (IOA, Invalid Operation Accumulative bit) This bit is set to one the illegal operand exception \boxtimes IOE bit is zero.
[7]	FEC	Read/Write	This bit is set to 1 when an arbitrary floating-point exception occurs on the current floating-point instruction, and is cleared to 0 otherwise.
[6]	reservations	read-only (computing)	0.

Continued on next page

Table 12.4 - continued from previous page

[5]	IDC	Read/Write	Input Denormalized Current bit (IDC, Input Denormalized Current bit) This bit is set to 1 when an off-specification number input exception occurs for the current floating-point instruction, and is cleared to 0 otherwise.
[4]	IXC	Read/Write	Inexact Exception Indication bit (IXC, InExact Current bit) This bit is set to 1 when an inexact exception occurs on the current floating-point instruction, and is cleared to 0 otherwise.
[3]	UFC	Read/Write	Underflow current bit (UFC, UnderFlow Current bit) This bit is set to 1 when an underflow exception occurs for the current floating-point instruction, and is cleared to 0 otherwise.
[2]	OFC	Read/Write	Overflow Exception Indicator bit (OFC, OverFlow Current bit) This bit is set to 1 when an overflow exception occurs for the current floating-point instruction, and is cleared to 0 otherwise.
[1]	DZC	Read/Write	Division by Zero Current bit (DZC, Division by Zero Current bit) This bit is set to 1 when a divide by 0 exception occurs for the current floating-point instruction, and is cleared otherwise.
[0]	IOC	Read/Write	Illegal operation exception indicator bit (IOC, Invalid Operation Current bit) This bit is set to 1 when an illegal operand exception occurs for the current floating-point instruction, and is cleared to 0 otherwise.

12.3.5 data-size side

The data size end is proposed relative to the format of the memory data storage, the high address byte stored to the low bit of physical memory is defined as the big end; the high address byte stored to the high bit of physical memory is defined as the little end.

The memory access unit of the E804 floating-point, like the memory access unit of the shaping pipeline, supports both the little-end format and the V1 version big-end (CSKY CPUs are designed with two versions of big-end and small-end, V1 and V2) format.且 does not support non-aligned accesses. In the small-end format, the lowest bit byte data always corresponds to the lowest bit of the address regardless of the access size of the access instruction (byte/halfword/word); in the V1 version big-end format, the data needs to be determined according to the access size of the access instruction (byte/halfword/word). Since floating-point access instructions are accessed in word units, the memory access pattern for floating-point access instructions is the same in small-end and V1 version big-end mode.

Table 12.5: Small End/Big End Storage Access Modes

access size	access address	the small end	Version V1本 Big End
word	A	D3D2D1D0	D3D2D1D0
half-word	A	D1D0	D3D2
half-word	A+2	D3D2	D1D0
nibbles	A	D0	D3

nibbles	A+1	D1	D2
nibbles	A+2	D2	D1
nibbles	A+3	D3	D0

12.4 Exception handling

This chapter describes exception handling for floating-point operations. It includes the following.

- Introduction to floating point exceptions;

User's Manual

- Input Denormalized Exception;
- Invalid Operation Exception;
- Divide by Zero Exception;
- Underflow Exception;
- Overflow Exception;
- Inexact Exception;
- Exception Prioritization.

12.4.1 Introduction to Floating Point Anomalies

Floating-point instructions generate six floating-point arithmetic-type exceptions, including off-specification number input exceptions in addition to the five floating-point arithmetic exceptions defined by the Floating-Point Protocol standard (ANSI/IEEE Std 754). The exception response process for floating-point instructions is similar to the main pipeline. All six types of floating-point arithmetic exceptions share the same exception vector number 30.

After the processor responds to a floating-point arithmetic exception, the software will determine which exception occurred on the current instruction by querying the Current bit in the Floating-Point Exception Register, and will process it sequentially based on the floating-point exception priority.

Floating point arithmetic exceptions are included:

- Input Denormalized Exception;
- Invalid Operation Exception;
- Divide by Zero Exception;
- Underflow Exception;
- Overflow Exception;
- Inexact Exception.

12.4.2 Unspecified number input abnormality

When a floating-point operand is a Denormalized Number, the IDC flag bit (FESR[5]) is set to 1. In this case, the specific exception handling is divided into two cases: exception enable and exception disable.

12.4.2.1 exception enable (computing)

IDE bit of 1 enables the off-specification input exception. When the input operand is an off-specification number, the processor will terminate normal program execution and throw an exception number 30, while the FEC bit (FESR[7]) and IDC bit (FESR[5]) are set to 1 for querying and processing by the exception service program. The source and destination registers of the exception instruction will not be changed.

12.4.2.2 abnormal disablement

When the IDE bit is 0, the off-specification number input exception is not enabled. The floating-point unit replaces the off-specification number with an approximation based on the FM bit. If the FM bit is 1, the off-specification number is swiped to the smallest floating-point number with the same sign as before; if the FM is 0, the off-specification number is swiped to 0 with the same sign as before. The floating-point unit then uses this approximation to complete the instruction operation and writes the result back to the registers.FEA (FESR[15]) IDA (FESR[13]) FEC (FESR[7]), and (IDCFESR[5]) are set to one.

12.4.3 Illegal operation exception

An illegal operation exception is thrown when one of the operands of the operation is illegal. Table 12.6 lists the operations and corresponding operands that generate illegal operation exceptions. The handling of illegal operation exceptions is divided into two cases: exception enable and exception disable.

Table 12.6: Illegal Operation Exceptions

directives	Illegal operation exception
All floating-point instructions (FABS, FNEG, FMOV except) Outside)	The operand contains sNaN
FADDS	(+INF) + (-INF) or (-INF) + (+INF)
FSUBS	(+INF) - (+INF) or (-INF) - (-INF)
fcmphss fcmplhs fcmpzhhss FCMPZLSS	Any NaN operand
FSTOSI	Arbitrary NaN operands or results with overflow
FSTOUI	Arbitrary NaN operand or result overflow up/down
FMULS FN MULS	(Zero x± INF) or (± INF x Zero) :sup:a
FDIVS	Zero/zero or INF/INF. ^a
FSQRTS	Operands less than 0

Notes. If the FM bit is 0 and the input operand of \pm is a Denormalized Number, the denormalized number will be seen as 0 and may also satisfy the IO exception condition.

12.4.3.1 exception enable (computing)

When the IOE bit (FCR[0]) is 1, the illegal operation exception is enabled. In this case, if an illegal operation exception occurs, the processor will terminate the execution and generate exception No. 30, and set FEC (FESR[7]) and IOC (FESR[0]) to 1 for the exception service program to query and handle. The source and destination registers of the instruction are not changed.

12.4.3.2 abnormal disablement

When the IOE bit is 0, floating-point instructions (except integer conversion operations) generate qNaN as the result of the operation and write it back to the destination register, and FEA (FESR[15]) IOA (FESR[8]) FEC (FESR[7]), and IOC (FESR[0]) are set to 1. For the format conversion instructions, the default result when an illegal operation occurs is shown in Table 4-2. For the format conversion instruction, the default result when illegal operation occurs is shown in Table 4-2.

Table 12.7: Default Results for Illegal Conversion Results

directive s	input value	Default Output
FSTOSI	NaN	0x00000000
	+INF or $x > 2^{31}$	0x7fffffff
	-INF or $x < -2^{31}$	0x80000000
FSTOUI	NaN	0x00000000
	+INF or $x > 2^{32}$	0xffffffff
	-INF or $x < 0$	0x00000000

12.4.4 divide by zero exception (math.)

the divisor is a definite non-zero number and the divisor is zero, a divide by zero exception is generated. A divide-by-zero exception is generated when the divisor is a non-specified number and the FM bit is 0, the divisor will be brushed to 0, again causing a divide by zero exception.

12.4.4.1 exception enable (computing)

If the DZE bit (FCR[1]) is set to 1, the divide by zero exception is enabled. If a divide-by-zero exception occurs, the processor terminates program execution and throws Exception 30. FEC (FESR[7]) and DZC (FESR[1]) are set to 1 for querying and handling by the exception service program. The source and destination registers of the instruction are not changed.

12.4.4.2 abnormal disablement

The Division by Zero exception is not enabled if DZE position 0. If this exception occurs, a signed infinity is written to the target register, and FEA (FESR[15]) DZA (FESR[9]) FEC (FESR[7]), and DZC (FESR[1]) are set to one.

12.4.5 overflow anomaly

The floating-point unit of the E804 will detect an overflow exception after a rounding operation. An overflow exception is thrown when the result exceeds the maximum finite number for the corresponding format. An overflow exception is always generated at the same time as an inexact exception because the result cannot be represented accurately when an overflow exception occurs.

12.4.5.1 exception enable (computing)

If the OFE bit (FCR[2]) is 1, the overflow exception is enabled. If an overflow exception occurs as a result of an operation, the processor terminates program execution and throws exception and sets FEC (FESR[7]) IXC (FESR[4]), and OFC (FESR[2]) to 1. The source of the instruction is the following

Neither the registers nor the destination register will be changed.

12.4.5.2 abnormal disablement

If the OFE bit is 0, the overflow exception is not enabled. If this exception occurs, the floating-point unit takes a signed infinity or a maximum finite number as the result of the operation and writes it to the target register, depending on the rounding mode.FEA (FESR[15]), IXA (FESR[12]), OFA (FESR[10]) FEC (FESR[7]) IXC (FESR[4]), and OFC (FESR[2]) are set to one. FESR[2] will be set to 1.

12.4.6 underflow anomaly (math.)

The E804 floating-point unit will detect if an underflow exception occurs after the result is rounded. An underflow exception is thrown when the result is less than the smallest finite number in the appropriate format. An underflow exception is always generated at the same time as an inexact exception.

12.4.6.1 exception enable (computing)

If the UFE bit (FCR[3]) is 1, the underflow exception is enabled. If an underflow exception occurs as a result of an operation, the processor terminates program execution and throws exception 30, and sets FEC (FESR[7]) IXC (FESR[4]), and UFC (FESR[3]) to 1. Neither the source nor the destination registers of the instruction are changed.

12.4.6.2 abnormal disablement

If the UFE bit is 0, the underflow exception is not enabled. If this exception occurs, the result will be swiped to the smallest finite number of the corresponding format with the same sign when the FM bit is 1. Otherwise, the result will be a 0 with the same sign, and FEA (FESR[15]) IXA (FESR[12]) UFA (FESR[11]) FEC (FESR[7]) IXC (FESR[4]), and UFC (FESR[3]) will be set for software query.

12.4.7 imprecise anomaly

Non-exact exceptions are caused by either the exponent field or the mantissa field of the result of a floating-point operation, as defined in the Floating-Point Protocol Standard (ANSI/IEEE Std 754) All of the following conditions result in inexact exceptions:

- The results before and after rounding are inconsistent;
- An overflow abnormality occurs;
- Underflow abnormality occurs.

Note: Imprecise exceptions occur frequently in floating-point operations. If the IXE bit is enabled, floating-point performance will be greatly degraded. It is recommended to turn off the exact exception bit if the application does not require high precision.

12.4.7.1 exception enable (computing)

If the IXE bit (FCR[4]) is 1, the inexact exception is enabled. If a floating-point operation generates an inexact exception, the processor terminates program execution and throws Exception 30 and sets FEC (FESR[7]) and IXC (FESR[4]) to one for querying by the exception service program. The source and destination registers of the instruction are not changed.

12.4.7.2 abnormal disablement

If the IXE bit is 0, the non-exact exception is not enabled. The floating-point unit will round the floating-point operation result according to the rounding mode and write the result after rounding back to the destination register. FE(FESR[15]) and FA(FESR[12]) and FC(FESR[7]) and XC(FESR[4]) will be set to one.

12.4.8 Exception Priority

Floating-point instructions may generate several exceptions at the same time during arithmetic operations. The exceptions that can be generated at the same time are listed below:

- Unspecified number of input exceptions and other 5 types of exceptions;
- Overflow anomalies and imprecise anomalies;
- Underflow anomalies and inexact anomalies.

Take a floating-point multiply instruction as an example, one operand is an unspecified number, the other operand is the least-specified number, the unspecified number input exception is not enabled (IDE is 0) and FM=1. This instruction will generate three kinds of exceptions, namely, unspecified number input exception, underflow exception, and imprecise exception. The floating-point exception register will record all the exceptions occurred during the processing of floating-point instructions for the software to query and handle.

The priority relationship for floating-point exceptions is as follows (recommended for floating-point exception programs)

1. Unspecified number input exception;
2. Illegal operation exception;
3. Divide by zero anomalies;
4. Overflow/underflow exception;
5. Non-exact anomalies.

12.4.9 Exceptions to special instructions

For multiply-accumulate/multiply-subtract instructions, the floating-point unit is treated as two separate floating-point operations, and the Floating-Point Exception Status Register will record all exceptions occurring during multiply operations and add/subtract operations to maximize the software's ability to record and handle exceptions.

12.5 instruction set

12.5.1 List of floating-point instructions

This section introduces the E804 floating-point instruction set. The floating-point instructions of the E804 are all 32-bit instructions, which can be classified according to the functions realized by the instructions:

- Data arithmetic instructions;
- Transmission class instructions;
- Memory access instructions.

12.5.1.1 data operation instruction

Table 12.8: List of Single-Precision Data Arithmetic Instructions

FSTOSI	Conversion of single-precision floating-point numbers to signed integers
FSTOUI	Converting single precision floating point numbers to unsigned integers
FSITOS	Conversion of signed integers to single precision floating point numbers
FUITOS	Converting unsigned integers to single precision floating point numbers
FCMPZHSS	Single precision floating point greater than or equal to 0
FCMPZLSS	Single precision floating point less than or equal to 0
FCMPZNES	Single precision floating point not equal to 0
FCMPZUOS	Whether a single precision floating point single operand is a noncount
FCMPHSS	Single precision floating point greater than or equal to
FCMPLTS	Single precision floating point less than
FCMPNES	Single precision floating point is not equal to
FCMPUOS	Whether a single-precision floating-point double operand is a noncount
FMOVIS	Single precision floating point immediate number generation
FMOVS	Single-precision floating-point numbers are carried between floating-point registers.

FABSS	Single-precision floating-point numbers take absolute values
FNEGS	Single precision floating point number inversion
FADDS	Single precision floating point addition
FSUBS	Single precision floating point subtraction
FMULS	Single precision floating point multiplication
FNMULS	Single-precision floating-point multiplication and inversion
FMACS	Single-precision floating-point multiply-accumulate
FMSCS	Single-precision floating-point number multiply-accumulate
FNMACS	Single-precision floating-point number multiplication and inverse accumulation
FNMSCS	Single-precision floating-point number multiplication and back-accumulation
FDIVS	Single precision floating point division
FSQRTS	Taking the square root of a single precision floating point number
FRECIPS	Single-precision floating-point numbers take the reciprocal

12.5.1.2 transport class command

Table 12.9: List of Data Transfer Instructions

FMTVRL	Floating Point Register Word Write Transfer Instruction
FMFVRL	Floating-point register word read transfer instruction

12.5.1.3 memory access instruction

Table 12.10: List of Memory Access Instructions

FLDS	Single-precision floating-point load instruction
FLDRS	Register Shift Addressing Single-Precision Floating-Point Load Instructions
FLDMS	Continuous single-precision floating-point load instruction
FLRWS	Single-precision floating-point immediate number memory read instruction
FSTS	Single-precision floating-point memory instruction
FSTRS	Register Shift Addressing Single-Precision Floating-Point Memory Instructions
FSTMS	Sequential single-precision floating-point store instructions

12.5.2 floating-point instruction execution delay

The E804 floating-point unit has been designed with strict instruction execution latency to reduce floating-point instruction execution latency while meeting low-cost and low-power design requirements. Software developers can use the instruction delay information to optimize the software and improve the efficiency of floating-point program execution.

Table 12.11: Floating-Point Instruction Execution Latency List

directives	implementation period	note
data operation instruction		
FSTOSI	1	-
FSTOUI	1	-
FSITOS	1	-

ELUTOS	1	-
FCMPZHSS	1	-
FCMPZLSS	1	-
FCMPZNES	1	-
FCMPZUOS	1	-
FCMPHSS	1	-
FCMPLTS	1	-
FCMPNES	1	-
FCMPUOS	1	-

Continued on next page

Table 12.11 - continued from previous page

FMOVIS	1	-
FMOVS	1	-
FABSS	1	-
FNEGS	1	-
FADDS	2	-
FSUBS	2	-
FMULS	2	-
FNMULS	2	-
FMACS	4	Split into FMULS and FADDS instructions.
FMSCS	4	Split into FMULS and FSUBS instructions.
FNMACS	4	Split into FNMULS and FADDS instructions.
FNMSCS	4	Split into FNMULS and FSUBS instructions.
FDIVS	2-28	-
FSQRTS	2-28	-
FRECIPS	2-28	-
transmission instruction		
FMTVRL	1	-
FMFVRL	1	-
memory access instruction		
FLDS	1	-
FLDRS	3	Split into IXW and FLDS commands.
FLDMS	N	Split into N FLDS instructions.
FLRWS	1	-
FSTS	1	-
FSTRS	3	Split into IXW and FSTS instructions.
FSTMS	N	Split into N FSTS instructions.

Note: Indicates memory access related instructions, the instruction completion cycle depends on the bus delay or cache hit, the values in the table are the fastest case. *The values in the table are the fastest.

Chapter 13 Initialization Reference Code

```
/* Sets the region 0 attribute, which is not executable, not cacheable, and readable and writable by super users and normal users.
```

13.1 MPU Setting Example

```
mfcr r10,cr<19,0>
bclri r10,0 //Set bit 0 of the control register to 0. Area 0 is not executable.
bseti r10,8 //Set control register bits 8 and 9 to 1. Area 0 is readable and writable by both super users and normal users.
bseti r10,9
bclri r10,24 //Set bit 24 of the control register to 0. Area 0 cannot be buffered high.
mtcr r10,cr<19,0>
```

```
/* Set area 1 attributes, executable, cacheable, readable and writable by super users and normal users.
```

```
mfcr r10,cr<19,0>
bseti r10,1 //set control register 1 to 1, area 1 executable
bseti r10,10 //Set bits 10 and 11 of the control register to 1. Area 1 can be read//written by both super-users and normal users.
bseti r10,11
bseti r10,25 //set control register bit 25 to 1, area 1 can be buffered high.
mtcr r10,cr<19,0>
```

```
/* Sets the region 2 attribute to be executable, non-highlightable, read-write for super users and read-only for normal users.
```

```
mfcr r10,cr<19,0>
bseti r10,2 //Set bit 2 of the control register to 1, area 2 is executable.
bclri r10,12 //Set bit 12 to 0,bit 13 to 1, read/write for super user, read only for normal user.
bseti r10,13
bclri r10,26 //Set control register 26 to 0, area 2 cannot be buffered.
mtcr r10,cr<19,0>
```

(Continued on next page)

```
/* Sets the region 3 attribute, which is not executable, cacheable, and readable and writable by both super users and normal users.
```

```

mfcr r10,cr<19,0>
bclri r10,3 //Set bit 3 of the control register to 0, area 3 is not executable.
bseti r10,14 //Set bits 14,15 to 1. Area 3 can be read and written by both super users and normal users.
bclri r10,15
bseti r10,27 //set control register 27 to 1, area 3 can be buffered high.
mtcr r10,cr<19,0>

/* The attribute settings for areas 4 to 7 are the same as those areas 0,1, and 2.

Executable attribute of //Area 0~7, set bit [7:0] of control register CR<19,0>.

Set the access privilege of //Areas 0~7, and set the [23:12] bits of Control Register CR<19,0>.

The cacheable privileges of //Areas 0~7 are set by setting bits [31:24] of control register CR<19,0>.

/* Sets protected area 0, base address and

protected area size.movi r10,0
mtcr r10,cr<21,0> //select protection area 0.
movi r10,0x0 //Set the base address of the
protected area 0x00000000. ori r10,r10,0x3f
//Set the size of the protected area 4G.
mtcr r10,cr<20,0> //Set the base address of the protected area to 0 and the size of the protected area to 4G.

/* Sets protected area 1, base address and

protected area size.movi r10,1
mtcr r10,cr<21,0> //Select protection area 1.
movih r10,0x2800 //Set the base address of the
protected area 0x28000000. ori r10,r10,0x2f //Set
the protected area size 16M.
mtcr r10,cr<20,0> //Set the protection zone address interval to 0x28000000~ 0x29000000.

/* Sets protected area 2, base address and

protected area size.movi r10,2
mtcr r10,cr<21,0> //Select protection area 2.
movih r10,0x2800 //Set the base address of the
protected area 0x28000000. ori r10,r10,0x27 //Set
the protected area size 1M.
mtcr r10,cr<20,0> //Set the protection zone address interval to 0x28000000 ~0x28100000.

```

(Continued on next page)

```
/* Sets protection area 3, base address, and protection area size.

movi r10,3
mtcr r10,cr<21,0> //Select protection area 2.
movih r10,0x28f0 //Set the base address of the
protected area 0x28f00000. ori r10,r10,0x27 //Set
the size of the protected area to 1M.

mtcr r10,cr<20,0> //Set the protection zone address interval to 0x28f00000 ~0x29000000.
```

The base address and protected area size settings for //Protected Area 3~7 are the same as those for Area 0, 1, and 2.

```
//Setting control register CR<21,0> Selects the protection area.

// Write the protected area base address and protected area size, to the control register CR<20,0>

/* Enable MPU

mfcr r7, cr<18,0> //Select the MUP enable control register.
bseti r7, 0 //Set control register CR<18,0> lowest bit to 2'b01, enable MPU.
bclri r7, 1
mtcr r7, cr<18,0> // Write preset value to MPU enable control register to turn on MPU.
```

13.2 Example of Cache Settings

Before turning on the cache, you need to invalidate the entire cache, then configure the CER and enable CACHE

```
// Cache invalidation according to the actual application requirements.
```

```
lrw r1, 1 // Preset bit 0 INV_ALL to 1.lrw
r2, 0xe000f004 // Preset CIR address.
st.w r1, (r2) // Write preset value to CIR, invalidation operation starts.
```

```
// Set cacheable area crcr0.
```

```
lrw r1, 0x00000039 // Cacheable 512M address space with preset start address 0x0.
lrw r2, 0xe000f008 // Preset the address where CRCR0 is located.
st.w r1, (r2) // Write preset value to CIR, invalidation operation starts.
```

```
//Bufferable area crcr1~3 are set as crcr0, and the start address and size of buffer and enable bit are
written to them respectively.
```

(Continued on next page)

```
//crcr1 address 0xe000f00c.

//crcr2 address 0xe000f010

//crcr3 address 0xe000f014

// Enable cache

    lrw r1, 0x0 // Preset value is 0x0000000.
    bseti r1,0 // Set Cache enable on.
    bclri r1,1 //Set, both instruction and data can be
    cached.bseti r1,2 //Set the Cache to write
    back mode.bseti r1,4 //Set the Cache to write
    back mode.
    bclri r1,5 // Set Cache off write allocate.lrw
    r2, 0xe000f000 // Preset the address where the
    CER is located. st.w r1, (r2) // Write preset value
    to CER, Cache enable.

// At this point, the cache is writable and in write-back mode, the quad is connected to the instruction
and data cache, and the write allocate function is turned off.
```

13.3 Interrupt Enable Initialization

After configuring the interrupt controller and interrupt vector table (refer to [Tightly Coupled IP](#) for details), the interrupt enable bit needs to be turned on as follows:

```
psrset ee,ie
```

13.4 General Purpose Register Initialization Example

```
//Initialize general-purpose registers r0~r15 and r28 to 0.
```

```
movi r0, 0 //Initialize general purpose
register 0 to 0.movi r1,0
movi r2, 0
movi r3, 0
movi r4, 0
movi r5, 0
movi r6, 0
movi r7, 0
```

(Continued on next page)

```

movi r8,0
movi r9,0
movi r10,0
movi r11,0
movi r12,0
movi r13,0
movi r14,0
movi r15,0
movi r28,0

```

13.5 Stack Pointer Initialization Example

The setting of the stack pointer is related to the current state of the program as shown below. The stack pointer is initialized to Example 1 in the super user state and Example 2 in the user state.

Example 1:

```

//superuser state, set superuser state and user state stack pointer

//r14 is mapped to the stack pointer register in the superuser state.

//Superuser state, the user state stack pointer register is CR<14,1>.

lrw r14, 0x01000000 // set superuser state
pointer lrw r0, 0x02000000 //
mtcr r0, cr<14,1> // set userland stack pointer

```

Example 2:

```

// In the user state, only the user state pointer can be set:

//User state r14 is mapped to the user state stack pointer register.

lrw r14, 0x02000000 //set user-state pointer

```

13.6 Example of Exception and Interrupt Service Program Entry Address Setting

The entry address of the exception and interrupt service program is set in two steps:

Step 1: Set the exception vector table address register VBR, according to the vector number, each exception vector number corresponds to the exception vector table address as, VBR

+ (Vector number<< 2)

Step 1: Write the entry address of the exception service program to the address of the exception vector table corresponding to the exception vector number in Step1. The example of access error exception with exception vector number 2 is as follows:

```
//Set the entry address of exception vector table, VBR corresponds to control register cr<1,0>.

Lrw r2,0
mtcr r2, cr<1,0> // set the address of the exception vector table to 0

//Exception/Interrupt Service Program Entry Address Setting

lrw r1, ACCERR_ERROR_BEGIN //set exception service
program entry address lrw r2, 0 //VBR address
movi r3, 0x2 //exception
vector number lsli r3,r3,2
addu r2,r2,r3 //calculate the exception vector number corresponding to the address of the exception
vector table
st.w r1,(r2,0x0) //Store the exception service program entry address into the corresponding exception
vector table address
br START

// User-set access error exception

service program label

ACCERR_ERROR_BEGIN

/* User's exception service program */

label ACCERR_ERROR_END
label START
```

13.7 FPU Initialization Example

When initializing the floating-point operation, you need to set the rounding mode of the floating-point operation, the handling of abnormal data, and other cases.

//clear floating-point exception status handling registers

Gemedi E804

User's Manual2,2> //clear the exception flag register.

//Set floating-point control registers

XUANTIE 玄铁

(Continued on next page)

```
lrw r1,0x0
bclri r1, 27 //Set the non-specification number handling method as if it were 0.
bclri r1, 25 // set bits 25 and 24 to 0, rounding mode round to nearest
even.bclri r1, 24 //
bclri r1,5 //Set the off-specification number without triggering an exception.
bclri r1, 4 //Set the result of the imprecise, when not enter the
exception, according to the rounding mode processing. bclri r1, 3
//Set the result of the lower overflow, do not trigger an exception,
according to the rounding mode. bclri r1, 2 //Set the result of
overflow, do not trigger an exception, according to the rounding mode.
bclri r1,1 //Set an exception to be triggered on a divide by 0 operation,
do not write back the result, and jump to a divide by 0 exception. bclri r1,0
//Set if NaN occurs in input and output, do not trigger exception, output
QNaN.mtcr r1,cr<1,2> //Write prefabricated value to floating point control
register FCR.

// At this time, the control state of FPU is such that only except for 0 exception, it will be transferred to
exception handling, and other exceptions will not be triggered, and the control state of FPU is such that
only except for 0 exception, it will be transferred to exception handling.
// Unspecified numbers are treated as 0, and the rounding mode is round to nearest even.
```

Chapter 14 Appendix A Base本 Instruction Glossary

The following are specific descriptions of each E804-implemented Xantei V2 instruction, and each instruction is described in detail below according to the alphabetical order of the instructions.

The bit width of an instruction is indicated by the number "32" or "16" at the end of the mnemonic for each Gentei V2 instruction. For example, "addc32" indicates that the instruction is 32 bits wide.

The "addc16" indicates that the instruction is a 16-bit unsigned carry-add instruction.

If you omit the bit-width of the instruction in the mnemonic (e.g., "addc") the system will automatically compile it into the most optimized instruction. The instructions with # in their names are pseudo instructions.

14.1 ABS - Absolute Value Command

Harmonization Directive	
grammatical	abs rz, rx
manipulate	RZ ← RX
Compilation results	Only 32-bit instructions exist. abs32 rz, rx
clarification	Take the absolute value of the RX value and store the result in RZ. Note that operand 0x80000000 results in 80000000.
Affecting Flags	unaffected
exceptions	not have

32-bit instruction	
manipulate	RZ ← RX
grammatical	abs32 rz, rx
clarification	Take the absolute value of the RX value and store the result in RZ. Note that operand 0x80000000 results in 80000000.
Affecting	unaffected

Flags	
exceptions	not have

Command Format.

	31 30	26 25	21 20	16 15	10 9	5 4	0
1	10001	00000	RX	000000	10000	RZ	

14.2 ADDC - Unsigned with Progressive Addition Instruction

Harmonization Directive		
tell to Budd hist teaching	addc rz, rx	addc rz, rx, ry
drill (practice) writings or works	RZ ← RZ + RX + C. C ← Progressive	RZ ← RX + RY + C. C ← Progressive
C o m p i l a t i o n r e s u l t s	Compile to 16-bit or 32-bit depending on the register range. Bit Commands. if (x<16) and (z<16), then addc16 rz, rx; else addc32 rz, rx, ry.	Compiles to the corresponding 16-bit or 32-bit instructions depending on the register range. if (y==z) and (x<16) and (z<16), then addc16 rz, rx; else addc32 rz, rx, ry.
persuade generic term for a sacrifice the gods	Add RZ/RY, RX and the value of the C bit and store the result in RZ and the rounding in the C bit.	

User's Manual	Bitwise
ar k er s of in fl u e n ce classi fier for hono rific peop le	
discri mina te const ant	not have

16-bit instructions	
manipulate	RZ \leftarrow RZ + RX + C, C \leftarrow Progressive
grammatical	addc16 rz, rx
instructions	Add RZ, RX and the value of the C bit and store the result in RZ and the rounding in the C bit.
Affecting Flags	C \leftarrow Bitwise
limitation	The range of the register is r0-r15.
exceptions	not have

16-bit instruction format:

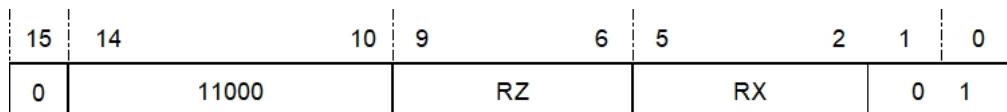


Figure 14.1: ADDC-1

32-bit instruction	
manipulate	RZ \leftarrow RX + RY + C, C \leftarrow Progressions
grammatical	addc32 rz, rx, ry
clarification	Add RX, RY and the value of the C bit and store the result in RZ and the rounding in the C bit.
Affecting Flags	C \leftarrow Bitwise
exceptions	not have

32-bit instruction format:

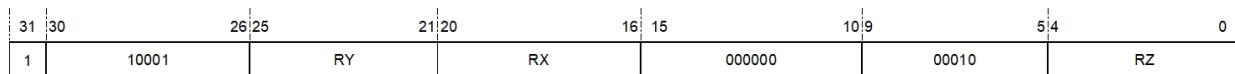


Figure 14.2: ADDC-2

14.3 ADDI - Unsigned Immediate Number Addition Instruction

Harmonization Directive			
tell to Bud dhis t teac hing	addi rz, oimm12	addi rz, rx, oimm12	addi rz, r28, oimm18
drill (practic e) writ ings or wor ks	RZ \leftarrow RZ + zero_extend(OIMM12)	RZ \leftarrow RX + zero_extend(OIMM12)	RZ \leftarrow R28 + zero_extend(OIMM18)

User's Manual	Depending on the range of the immediate numbers and registers compiled into the corresponding 16-bit or 32-bit instructions. if (z<8) and (oimm12<257), and (oimm12<257), and addi16 rz, oimm8; else addi32 rz, rz, oimm12.	Depending on the range of the immediate numbers and registers compiled into the corresponding 16-bit or 32-bit instructions. if (z<8) and (oimm12<257), and (oimm12<257), and addi16 rz, oimm8; else addi32 rz, rx, oimm12.	Compiles to the corresponding 16-bit or 32-bit instructions depending on the range of the immediate numbers and registers. if (oimm12<9) and (z<8) and (x<8), addi16 rz, rx, oimm3. elsif (oimm12<257) and (x==z) and (z<8), addi16 rz, oimm8; else addi32 rz, rx, oimm12.
persuade generic tester for a sacrifice the gods	Expands an immediate zero with a bias of 1 to 32 bits, then adds it to the value of RX/RZ and stores the result in RZ.		
Market sectors of influence classifier for honorific people	unaffected		
set a limit (on)	If the source register is R28, the range of immediate numbers is 0x1-0x40000. If the source register is not R28, the range of immediate numbers is 0x1-0x1000.		

16-bit instruction 1	
manipulate	$RZ \leftarrow RZ + \text{zero_extend}(OIMM8)$
grammatical	addi16 rz, oimm8
instructions	Extend the 8-bit immediate number (OIMM8) zero with a bias of 1 to 32 bits, then add it to the value of RZ and store the result in RZ. Note: The binary operand IMM8 is equal to OIMM8 - 1.
affect aspiration	unaffected
limitation	The range of registers is r0-r7; the range of immediate numbers is 1-256.
exceptions	not have

16-bit instruction format 1:

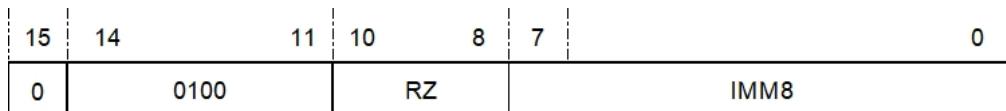


Figure 14.3: ADDI-I

IMM8 domain:

Specifies a value without a bias immediate number.

Note: The value OIMM8 added to the register is biased by 1 compared to the binary operand IMM8.

00000000:

Plus 1

00000001:

Plus 2

.....

11111111:

plus 256

16-bit instructions 2	
manipulate	$RZ \leftarrow RX + \text{zero_extend}(OIMM3)$
grammatical	addi16 rz, rx, oimm3
clarification	Expand the 3-bit immediate number (OIMM3) zero with bias 1 to 32 bits, then add it to the value of RX and store the result in RZ.

User's Manual	Note: The binary operand IMM3 is equal to OIMM3 - 1.
affect aspirati on	unaffected
limitati on	The range of registers is r0-r7; the range of immediate numbers is 1-8.
exception	not have

16-bit instruction format 2:

15	14	11	10	8	7	5	4	2	1	0
0	1011		RX		RZ		IMM3	1	0	

Figure 14.4: ADDI-2

IMM3 domain:

Specifies a value without a bias immediate number.

Note: The value OIMM3 added to the register is biased by 1 compared to the binary operand IMM3.

000:

Plus 1

001:

Plus 2

.....

111:

Plus 8

32-bit instructions 1	
manipulate	$RZ \leftarrow RX + \text{zero_extend}(OIMM12)$
grammatical	addi32 rz, rx, oimm12
clarification	Expand the 12-bit immediate number (OIMM12) zero with bias 1 to 32 bits, then add it to the value of RX and store the result in the RZ. Note: The binary operand IMM12 is equal to OIMM12 - 1.
affect aspiration	unaffected
limitation	The range of immediate numbers is 0x1-0x1000.
exceptions	not have

32-bit instruction format 1:

31	30	26	25	21	20	16	15	12	11	0
1	11001		RZ		RX	0000		IMM12		

Figure 14.5: ADDI-3

Specifies a value without a bias immediate number.

Note: The value OIMM12 added to the register is biased by 1 compared to the binary operand IMM12.

000000000000:

Add 0x1

000000000001:

Add 0x2

.....

111111111111:

add 0x1000

32-bit instructions 2	
manipulate	RZ← R28+ zero_extend(OIMM18)
grammatical	addi32 rz, r28, oimm18
instructions	Expand the 18-bit immediate number (OIMM18) zero with bias 1 to 32 bits, then add it to the value of R28 and store the result in the RZ. Note: The binary operand IMM18 is equal to OIMM18 - 1.
affect aspiration	unaffected
limitation	The range of immediate numbers is 0x1-0x40000.
exceptions	not have

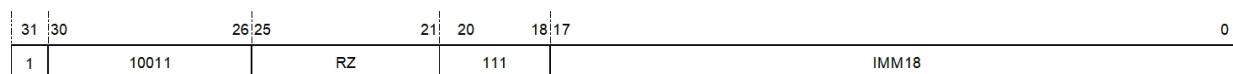
32-bit instruction format 2:

Figure 14.6: ADDI4

IMM18 domain:

Specifies a value without a bias immediate number.

Note: The value OIMM18 added to the register is biased by 1 compared to the binary operand IMM18.

0000000000000000:

Add 0x1

0000000000000001:

Add 0x2

.....

1111111111111111:

14.4 ADDI(SP)-Unsigned (stack pointer) immediate number addition instruction

Harmonization Directive		
grammatical	addi rz, sp, imm	addi sp, sp, imm
manipulate	RZ← SP+ zero_extend(IMM)	SP← SP+ zero_extend(IMM)
Compilation results	Only 16-bit instructions exist. addi rz, sp, imm	Only 16-bit instructions exist. addi sp, sp, imm
clarification	Expand the immediate number (IMM) zeroes to 32 bits and add them to the value of the stack pointer (SP), storing the result in RZ or SP.	
Markers of influence classifier for honorific people	unaffected	
limitation	The range of registers is r0-r7; the range of immediate numbers is 0x0-0x3fc.	
exceptions	not have	

16-bit instruction 1	
manipulate	RZ← SP+ zero_extend(IMM)
grammatical	addi16 rz, sp, imm8
instructions	Expand the Immediate Number (IMM) zeroes to 32 bits, then add them to the value of the Stack Pointer (SP) and store the result in RZ. Note: Immediate number (IMM) is equal to the binary operand IMM8 << 2.
affect aspiration	unaffected
limitation	The range of registers is r0-r7; the range of immediate numbers is (0x0-0xff) << 2.
exceptions	not have

16-bit instruction format 1:

User's Map	15	14	11	10	8	7	0
	0	11	RZ			IMM8	

Figure 14.7: ADDI(SP)-1

IMM8 domain:

Specifies the value without shifting the immediate number.

Note: The value IMM added to the register is shifted 2 bits to the left compared to the binary operand IMM8.

00000000:

Add 0x0

00000001:

Add 0x4

.....

11111111:

Add 0x3fc

16-bit instructions 2	
manipulate	SP \leftarrow SP + zero_extend(IMM)
grammatical	addi16 sp, sp, imm
clarification	Expand the Immediate Number (IMM) zeroes to 32 bits, then add them to the value of the Stack Pointer (SP) and store the result in RZ. Note: Immediate number (IMM) is equal to the binary operand {IMM2, IMM5} << 2.
affect aspiration	unaffected
limitation	The source and destination registers are both stack pointer registers (R14) the immediate number range is (0x0-0x7f) << 2.
exceptions	not have

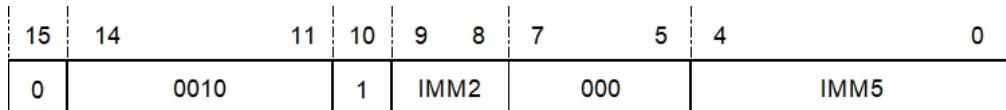
16-bit instruction format 2:

Figure 14.8: ADDI(SP)-2

IMM domain.

Specifies the value without shifting the immediate number.

Note: The value IMM added to the register needs to be shifted 2 bits to the left compared to the binary operands {IMM2, IMM5}.

{00, 00000}

Add 0x0

{00, 00001}

Add 0x4

.....

{11, 11111}

Add 0x1fc

14.5 ADDU - unsigned addition instruction

Harmonization Directive		
tell to Bud dhis t teac hing	addu rz, rx	addu rz, rx, ry
drill (pra ctice) writi ngs or wor ks	RZ← RZ+ RX	RZ← RX+ RY
C o m pi la ti o n re s ul ts	Compile to the corresponding 16 according to the range of the registers. Bit or 32-bit instructions. if (z<16) and (x<16), then addu16 rz, rx; else addu32 rz, rx, ry.	Compiles to the corresponding 16-bit or 32-bit instructions depending on the register range. if (z<8) and (x<8) and (y<8), then addu16 rz, rx, ry; elseif (y==z) and (x<16) and (z<16), then addu16 rz, rx; else addu32 rz, rx, ry.
pers uad e gene ric ter m for a saci fice the gods	Add RZ/RY to the value of RX and store the result in RZ.	

User's Manual ar k er s of in fl u e n ce class ifier for hon orifi c peo ple	Manufactured not affected
disc rimi nate cons tant	not have

16-bit instruction 1	
Operation.	RZ← RZ+ RX
grammatical	addu16 rz, rx
Description.	Add the values of RZ and RX and store the result in RZ.
Impact Flags.	unaffected
limitation	The range of the register is r0-r15.
exceptions	not have

16-bit instruction format 1:

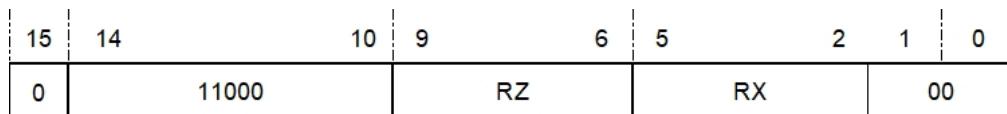


Figure 14.9: ADDU-1

16-bit instructions 2	
manipulate	RZ \leftarrow RX+ RY
grammatical	addu16 rz, rx, ry
clarification	Add the values of RX and RY and store the result in RZ.
Affecting Flags	unaffected
limitation	The range of the register is r0-r7.
exceptions	not have

16-bit instruction format 2:

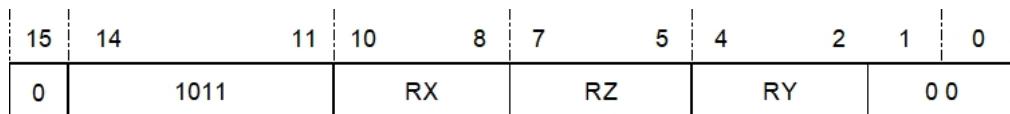


Figure 14.10: ADDU-2

32-bit instruction	
manipulate	RZ \leftarrow RX+ RY
grammatical	addu32 rz, rx, ry
clarification	Add the values of RX and RY and store the result in RZ.
Affecting Flags	unaffected
exceptions	not have

32-bit instruction format:

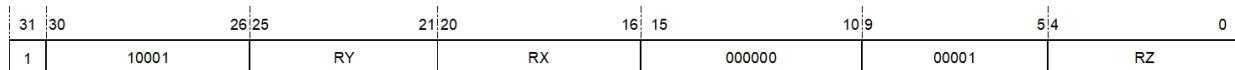


Figure 14.11: ADDU-3

14.6 AND - By Bit and Command

Harmonization Directive		
tell to Budd hist teaching	and rz, rx	and rz, rx, ry
drill (practice) writings or works	RZ← RZ and RX	RZ← RX and RY
C o m p i l a t i o n r e s u l t s	Compile to 16-bit or 32-bit depending on the register range. Bit Commands. if (x<16) and (z<16), then and16 rz, rx; else and32 rz, rx.	Compiles to the corresponding 16-bit or 32-bit instructions depending on the register range. if (y==z) and (x<16) and (z<16), then and16 rz, rx; else and32 rz, rx, ry.
persuade generic term for a sacrifice the gods	The value of RZ/RY and RX are bitwise summed and the result is stored in RZ.	
M ar k er s of in	unaffected	

fl	User's Manual
u	
e	
n	
ce	
classi	
fier	
for	
hono	
rific	
peopl	
e	
discr	not have
mina	
te	
const	
ant	

16-bit instructions	
manipulate	RZ← RZ and RX
grammatical	and16 rz, rx
clarification	Bitwise sums the values of RZ and RX and stores the result in RZ.
Affecting Flags	unaffected
limitation	The range of the register is r0-r15.
exceptions	not have

16-bit instruction format:

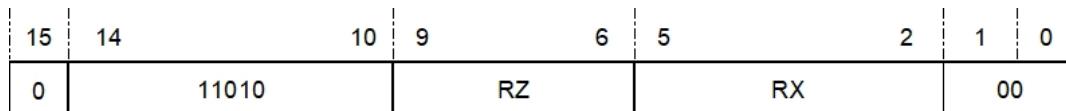


Figure 14.12: AND-1

32-bit instruction	
manipulate	RZ← RX and RY
grammatical	and32 rz, rx, ry
clarification	The values of RX and RY are summed bitwise and the result is stored in RZ.
Affecting Flags	unaffected
exceptions	not have

32-bit instruction format:

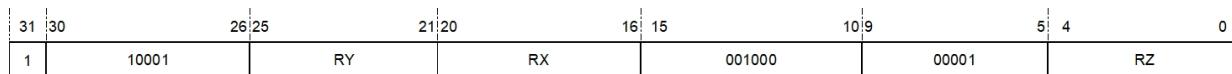


Figure 14.13: AND-2

14.7 ANDI - Immediate Number By Bit with Instruction

Harmonization Directive	
grammatical	andi rz, rx, imm16
manipulate	RZ← RX and zero_extend(IMM12)
Compilation results	32-bit instructions only andi32 rz, rx, imm12
clarification	Expands the 12-bit immediate zero to 32 bits and then performs a bitwise sum operation with the value of RX and stores the result in RZ.
Influence Flag Bit	unaffected
limitation	The range of immediate numbers is 0x0-0xFFFF.
exceptions	not have

32-bit instruction	
manipulate	RZ← RX and zero_extend(IMM12)
grammatical	andi32 rz, rx, imm12
clarification	Expands the 12-bit immediate zero to 32 bits and then performs a bitwise sum operation with the value of RX and stores the result in RZ.
Affecting Flags	unaffected
limitation	The range of immediate numbers is 0x0-0xFFFF.

32-bit instruction format:

31	30	26	25	21	20	16	15	12	11	0
1	11001		RZ		RX	0010			IMM 12	

Figure 14.14: ANDI

14.8 ANDN - per-bit non-associative instruction

Harmonization Directive		
tell to Budd hist teaching	andn rz, rx	andn rz, rx, ry
drill (practice) writings or works	RZ← RZ and (!RX)	RZ← RX and (!RY)
Comiles to the corresponding 16-bit or 32-bit instructions depending on the register range. if (x==z) and (y<16) and (z<16), then andn16 rz, ry; else andn32 rz, rz, rx.	Compile to 16-bit or 32-bit depending on the register range. Bit Commands. if (x<16) and (z<16), then andn16 rz, rx; else andn32 rz, rz, rx.	
pers uade generic term for a sacrifice the gods	For andn rz, rx, sum the value of RZ bitwise with the non-value of RX, and store the result in RZ; for andn rz, rx, ry, sum the value of RX is bitwise summed with the non-value of RY and the result is stored in RZ.	

User's Manual	Unaffected
ar k er s of in fl u e n ce classi fier for hono rific peop le	
discrimina te const ant	not have

16-bit instructions	
manipulate	RZ \leftarrow RZ and (!RX)
grammatical	andn16 rz, rx
clarification	Bitwise sums the value of RZ with the non-value of RX and stores the result in RZ.
Affecting Flags	unaffected
limitation	The range of the register is r0-r15.
exceptions	not have

16-bit instruction format:

15	14	10	9	6	5	2	1	0
0		11010		RZ		RX		01

Figure 14.15: ANDN-1

32-bit instruction	
manipulate	RZ← RX and (!RY)
grammatical	andn32 rz, rx, ry
clarification	The value of RX is bitwise summed with the non-value of RY and the result is stored in RZ.
Influence Flag Bit	unaffected
exceptions	not have

32-bit instruction format:

31	30	26 25	21 20	16 15	10 9	5 4	0
1	10001	RY	RX	001000	00010	RZ	

Figure 14.16: ANDN-2

14.9 ANDNI - Immediate Number by Bit Non-Issue Instruction

Harmonization Directive	
grammatical	andni rz, rx, imm16
manipulate	RZ← RX and ! (zero_extend(IMM12))
Compilation results	32-bit instructions only andni32 rz, rx, imm12
clarification	Extends the 12-bit immediate zero to 32 bits and takes the negation, then performs a bitwise sum operation with the value of RX and stores the result in RZ.
Affecting Flags	unaffected
limitation	The range of immediate numbers is 0x0-0xFFFF.
exceptions	not have

32-bit instruction	
manipulate	RZ← RX and ! (zero_extend(IMM12))

Users Manual 1	andni32 rz, rx, imm12
clarification	Extends the 12-bit immediate zero to 32 bits and takes the negation, then performs a bitwise sum operation with the value of RX and stores the result in RZ.
Affecting Flags	unaffected
limitation	The range of immediate numbers is 0x0-0xFFFF.
exceptions	not have

32-bit instruction format:

31	30	26	25	21 20	16	15	12 11	0
1	11001	RZ		RX		0011	IMM12	

Figure 14.17: ANDNI

14.10 ASR - arithmetic right shift instruction

Harmonization Directive		
tell to Budd hist teaching	asr rz, rx	asr rz, rx, ry
drill (practice) writings or works	RZ← RZ>>> RX[5:0]	RZ← RX>>> RY[5:0]
C o m p i l a t i o n r e s u l t s	Compile to 16-bit or 32-bit depending on the register range. Bit Commands. if (x<16) and (z<16), then asr16 rz, rx; else asr32 rz, rx.	Compiles to the corresponding 16-bit or 32-bit instructions depending on the register range. if (x==z) and (y<16) and (z<16), then asr16 rz, ry; else asr32 rz, rx, ry.
c l a r i f i c a t i o n	For asr rz, rx, the value of RZ is arithmetically right-shifted (the original value is right-shifted, and the left-hand side is shifted into a copy of the original sign bit) and the result is deposited into RZ, with the number of right-shifted bits being determined by the value of the 6 lower bits of RX (RX[5:0]); if the value of RX[5:0] is greater than 30, then the value of RZ (0 or -1) is determined by the sign bit of the original value of RZ; For asr rz, rx, ry, the value of RX is arithmetically right-shifted (the original value is right-shifted, and the left-hand side is shifted into a copy of the original sign bit) and the result is deposited into the RZ, the number of right shifts is determined by the value of the lower 6 bits of RY (RY[5:0]); if the value of RY[5:0] is greater than 30, then the value of RZ (0) is the same as the value of RY[5:0].	

User's Manual (version) is determined by the sign bit of RX.

M ar k er s of in fl u e n ce classi fier for hono rific peopl e	unaffected
discrimina te const ant	not have

16-bit instructions	
manipulate	RZ← RZ>>> RX[5:0]
grammatical	asr16 rz, rx
clarification	The value of RZ is arithmetically right-shifted (the original value is right-shifted, and the left-hand side is shifted into a copy of the original sign bit) and the result is deposited into RZ, with the number of right-shifted bits being determined by the value of the six lower bits of RX (RX[5:0]); if the value of RX[5:0] is greater than 30, then the value of RZ (0 or -1) is determined by the original value of RZ is determined by the sign bit of the
Impact Signs classifier for honorific people	unaffected
limitation	The range of the register is r0-r15.
exceptions	not have

16-bit instruction format:

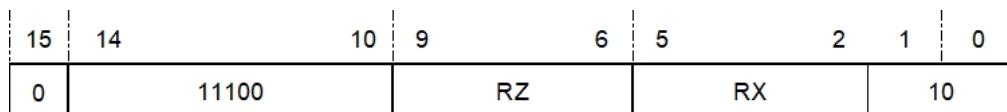


Figure 14.18: ASR-1

32-bit instruction	
manipulate	RZ← RX>>> RY[5:0]
grammatical	asr32 rz, rx, ry
clarification	The value of RX is arithmetically right-shifted (the original value is right-shifted, and the left-hand side is shifted into a copy of the original sign bit), and the result is deposited into RZ, and the number of right-shifted bits is determined by the value of the six lower bits of RY (RY[5:0]); if the value of RY[5:0] is greater than 30, then the value of RZ (0 or -1) is determined

User's Manual	by the value of RX. The sign bit determines.
Impact Signs classifier for honorific people	unaffected
exceptions	not have

32-bit instruction format:



Figure 14.19: ASR-2

14.11 ASRC - Immediate Number Arithmetic Right Shift to C-Bit Instruction

Harmonization Directive	
tell to Bud dhist teac hing	asrc rz, rx, oimm5
drill (pra ctice) writi ngs or wor ks	RZ← RX>>> OIMM5, C← RX[OIMM5 - 1]
c o m pi la ti o n re s ul t resol ute	32-bit instructions only asrc32 rz, rx, oimm5
cl ar ifi ca ti o n	The value of RX is arithmetically shifted right (the original value is shifted right, and the left side is shifted into a copy of the original sign bit), and the last bit of the shifted out bit is stored in the condition bit C, and the result of the shift is stored in RZ, and the number of right shifted bits is determined by the value of the 5-bit immediate number with bias 1 (OIMM5). If the value of OIMM5 is equal to 32, then the condition bit C is the sign bit (highest bit) of RX, and the value of RZ (0 or -1) is determined by the sign bit of RX.

User's Manual	C RX[OIMM5 - 1]
ar k er s of in fl u e n ce class ifier for hon orifi c peop le	
set a limit (on) regu late	Immediate numbers range from 1-32.
discr imin ate cons tant	not have

32-bit instruction	
drill (practic e) writ ings or wor ks	$RZ \leftarrow RX >>> OIMM5, C \leftarrow RX[OIMM5 - 1]$
tell to Bud dhis t teac hing	asrc32 rz, rx, oimm5
cl ar ifi ca ti o n	<p>The value of RX is arithmetically shifted right (the original value is shifted right, and the left side is shifted into a copy of the original sign bit), and the last bit of the shifted bit is stored in the condition bit C, and the result of the shift is stored in RZ, and the number of right shifted bits is determined by the value of the 5-bit immediate number with bias 1 (OIMM5). If the value of OIMM5 is equal to 32, then the condition bit C is the sign bit (highest bit) of RX, and the value of RZ (0 or -1) is determined by the sign bit of RX. Attention:</p> <p>The binary operand IMM5 is equal to OIMM5 - 1.</p>
M ar k er s of in fl u e n ce clas sifie r for hon orifi c peo ple	$C \leftarrow RX[OIMM5 - 1]$

Set & M	Immediate numbers range from 1-32.
limit (on) register update	not have

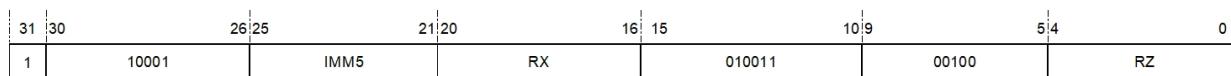
32-bit instruction format:

Figure 14.20: ASRC

IMM5 domain:

Specifies a value without a bias immediate number.

Note: The shifted value OIMM5 is biased by 1 compared to the binary operand IMMS.

00000:

Shift 1 bit

00001:

Move 2 positions

.....

11111:

Shift 32 bits

14.12 ASRI - Arithmetic Right Shift Instruction for Immediate Numbers

Harmonization Directive	
grammatica 1	asri rz, rx, imm5
manipulate	RZ← RX>>> IMM5
Translation results	Compiles to the corresponding 16-bit or 32-bit instructions depending on the register range. if (x<8) and (z<8), then asri16 rz, rx, imm5; else asri32 rz, rx, imm5.
clarification	For asri rz, rx, imm5, the value of RX is arithmetically right-shifted (the original value is right-shifted, and the left-hand side is shifted into a copy of the original sign bit) and the result is deposited into RZ, with the number of bits shifted to the right determined by the value of the 5-bit immediate number (IMM5); if the value of IMM5 is equal to 0, then the value of RZ's The value will be the same as RX.
Impact Signs classifier for honorific people	unaffected
exceptions	not have

16-bit instructions	
manipulate	RZ← RX>>> IMM5
grammatica 1	asri16 rz, rx, imm5
clarification	The value of RX is arithmetically right-shifted (the original value is right-shifted, and the left-hand side is shifted into a copy of the original sign bit), and the result is deposited into RZ, with the number of right-shifted bits shifted from The value of the 5-bit immediate number (IMM5) determines this; if the value of IMM5 is equal to 0, then the value of RZ will be unchanged.

impact	affected
Logo classifi er for honor fic people	
limitat ion	The range of registers is r0-r7; the range of immediate numbers is 0-31.
except ions	not have

16-bit instruction format:

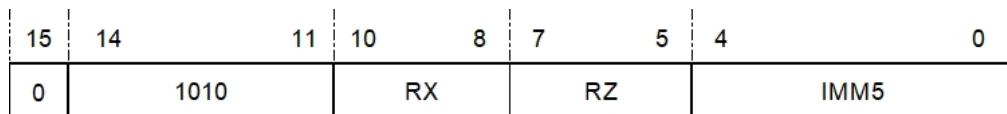


Figure 14.21: ASRI-1

32-bit instruction	
manipulate	RZ← RX>>> IMM5
grammatical	asri32 rz, rx, imm5
clarification	<p>The value of RX is arithmetically right-shifted (the original value is right-shifted, and the left-hand side is shifted into a copy of the original sign bit) and the result is deposited into RZ, with the number of right-shifted bits shifted from 5 to 5.</p> <p>The value of the Immediate Number of Bits (IMM5) determines this; if the value of IMM5 is equal to 0, then the value of RZ will be as RX.</p>
impact	unaffected
Logo classifier for honorific people	
limitation	The range of immediate numbers is 0-31.
exceptions	not have

32-bit instruction format:

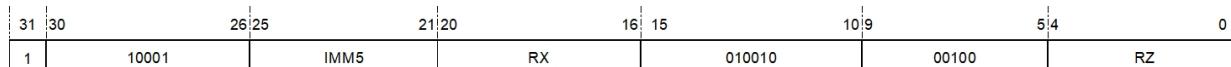


Figure 14.22: ASRI-2

14.13 BCLRI - Immediate Digit Clear Instruction

Harmonization Directive		
tell to Budd hist teaching	bclri rz, imm5	bclri rz, rx, imm5
drill (practice)	RZ ← RZ[IMM5] Zeroing	RZ← RX[IMM5]

Users Manual	
C o m pi la ti o n re s ul ts	<p>Zeroing is compiled to the corresponding 16-bit or 32-bit instructions.</p> <p>if ($z < 8$), then bclri16 rz, imm5; else bclri32 rz, rx, imm5.</p>
persuade gener ic term for a sacrif ice the gods	Clear the bit indicated by the IMM5 field value of the RZ/RX value to zero, leave the rest of the bits unchanged, and store the result of the clearing to RZ.
M ar k er s of in fl u e n ce classi fier for hono rific peopl e	unaffected

Set a limit (on) regulate	The range of immediate numbers is 0-31.
---------------------------	---

16-bit instructions	
manipulate	RZ \leftarrow RZ[IMM5] Zeroing
grammatical	bclri16 rz, imm5
instructions	Clear the bit of the value of RZ indicated by the IMM5 field value to zero, leave the remaining bits unchanged, and store the result of the clearing to RZ.
Impact Marker classifier for honorific people	unaffected
limitation	The range of the register is r0-r7; The range of immediate numbers is 0-31.
exceptions	not have

16-bit instruction format:

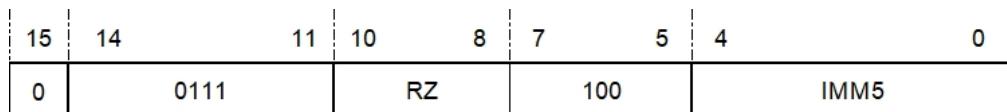


Figure 14.23: BCLRI-1

32-bit instruction	
manipulate	RZ \leftarrow RX[IMM5] Clear to Zero
grammatical	bclri32 rz, rx, imm5
clarification	Clear the bit of the value of RX indicated by the value of the IMM5 field to zero, leave the rest of the bits unchanged, and store the result of the clearing to RZ.
Impact Marker classifier for honorific people	unaffected
limitation	The range of immediate numbers is 0-31.
exceptions	not have

32-bit instruction format:

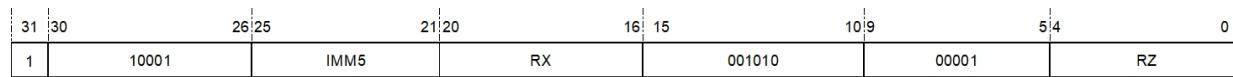


Figure 14.24: BCLRI-2

14.14 BEZ - Register Equals Zero Branch Instruction

Harmonization Directive	
gram matica 1	bez rx, label
manip ulate	Register equal to zero transfers the program if (RX== 0) PC← PC+ sign_extend(offset<< 1) else PC← PC+ 4
com pilin g in the end	32-bit instructions only bez32 rx, label
clarifi cation	If register RX is equal to zero, the program transfers to label; otherwise, the program executes the next instruction, i.e., PC ← PC + 4. Label is the value of the current program PC plus the 16-bit relative offset signed extension to 32 bits shifted by 1 bit to the left. The transfer range for the instruction is ± 64KB of address space.
Mar ker s of infl uen ce classi fier for hono rific peopl e	unaffected
except ions	not have

32-bit instruction	
manip ulate	Register equal to zero transfers the program if(RX== 0) PC← PC+ sign_extend(offset<< 1) else PC← PC+ 4

gram users Manual matica l	be732 rx, label
clarifi cation	If register RX is equal to zero, the program transfers to label; otherwise, the program executes the next instruction, i.e., $PC \leftarrow PC + 4$. Label is the value of the current program PC plus the 16-bit relative offset signed extension to 32 bits shifted by 1 bit to the left. The transfer range for the instruction is $\pm 64KB$ of address space.
Mar ker s of infl uen ce classi fier for hono rific peopl e	unaffected
except ions	not have

32-bit instruction format:

31	30	26	25	21	20	16	15	0
1	11010		01000		RX		Offset	

Figure 14.25: BEZ

14.15 BF--C is a 0 branch command.

Harmonization Directive	
gram matica l	bf label
manip ulate	C equals zero then the program is transferred. if(C==0) PC← PC+ sign_extend(offset<< 1); else PC← next PC;
Translati on resu lts	Compiles to the corresponding 16-bit or 32-bit instruction depending on the range of the jump. if (offset<1KB), then bf16 label; else bf32 label.
clarifi cation	If the condition flag bit C is equal to zero, the program transfers to label; otherwise, the program executes the next instruction. Label is the value of the current program PC plus the relative offset shifted by 1 bit to the left and extended to 32 bits in signed form. The shift range is ± 64KB address space.
Impact Sign s classifi er for honor ific peopl e	unaffected
except ions	not have

16-bit instructions	
manipulate	C equals zero then the program is transferred. if(C==0) PC← PC+ sign_extend(offset<< 1) else PC← PC+ 2
grammatical	bf16 label
clarification	If the condition flag C is equal to 0, the program transfers label; otherwise, the program executes the next instruction, i.e. PC ← PC + 2. Label is the value of the current program PC plus the 10-bit relative offset after a 1-bit leftward signed extension to 32 bits. The transfer range for the instruction is ± 1KB of address space.
Markers of influence classifier for honorific people	unaffected
exceptions	not have

16-bit instruction format:

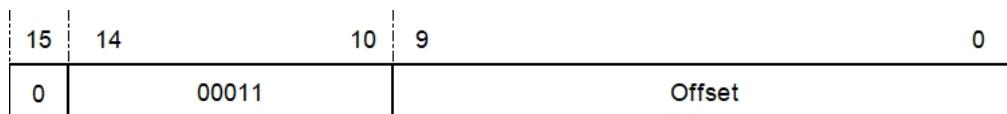


Figure 14.26: BF-1

32-bit instruction

User Manual	<p>if(C== 0) $PC \leftarrow PC + sign_extend(offset \ll 1)$ else $PC \leftarrow PC + 4$</p>
grammatical	bf32 label
instructions	<p>If the condition flag C is equal to zero, the program transfers to label; otherwise, the program executes the next instruction, i.e., $PC \leftarrow PC + 4$. Label is the value of the current program PC plus the 16-bit relative offset signed extension to 32 bits shifted by 1 bit to the left. The transfer range for the order is ± 64KB address space.</p>
Markers of influence classifier for honorific people	unaffected
exceptions	not have

32-bit instruction format:

31	30	26	25	21	20	16	15	0
1	11010		00010		00000		Offset	

Figure 14.27: BF-2

14.16 BGENI - Immediate Digit Generation Instruction

Harmonization Directive	
grammatica 1	bgeni rz, imm5
manipulate	$RZ \leftarrow (2)^{IMM5}$
Translation in the end	Only 32-bit instructions exist. bgeni32 rz, imm5
clarification	Sets the bit of RZ determined by the 5-bit immediate number ($RZ[IMM5]$) and clears the other bits of RZ. Note that if IMM5 is less than 16, this instruction is a pseudo-instruction of $movi\ rz, (2)^{IMM5}$; if IMM5 is greater than or equal to 16, the The instruction is a pseudo-instruction for $movih\ rz, (2)^{IMM5}$.
Impact Signs classifier for honorific people	unaffected
limitation	The range of immediate numbers is 0-31.
exceptions	not have

32-bit instruction	
manipulate	$RZ \leftarrow (2)^{IMM5}$.
grammatica 1	bgeni32 rz, imm5

clears Manual classification	Sets the bit of RZ determined by the 5-bit immediate number (RZ[IMM5]) and clears the other bits of RZ. Note that if IMM5 is less than 16, this instruction is a pseudo-instruction of movi32 rz, $(2)^{IMM5}$; if IMM5 is greater than or equal to 16, the This instruction is a pseudo-instruction for movih32 rz, $(2)^{IMM5}$.
Impact Signs classifier for honorific people	unaffected
limitation	The range of immediate numbers is 0-31.
exceptions	not have

32-bit instruction format:

If IMM5 is less than 16:

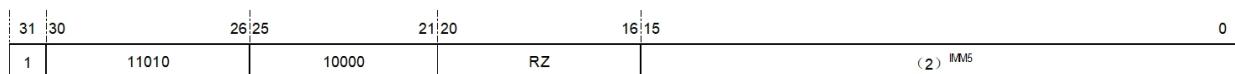


Figure 14.28: BGENI-1

If IMM5 is greater than or equal to 16:

31 30	26 25	21 20	16 15	0
1	11010	10000	RZ	(2) IMM5

Figure 14.29: BGENI-2

14.17 BGENR - Register Bit Generation Instruction

Harmonization Directive	
grammatical	bgenr rz, rx
manipulate	If (RX[5]== 0) , then RZ← 2 ^{RX[4:0]} ; else RZ← 0;
translation resolute	32-bit instructions only bgenr32 rz, rx
clarification	If RX[5] is 0, then set the register bits of RZ determined by the five lower bits of RX (RX[4:0]) and clear RZ of all its bits; otherwise, clear RZ.
impact indicator aspiration	unaffected
exceptions	not have

32-bit instruction	
manipulate	If (RX[5]== 0) , then RZ← 2 ^{RX[4:0]} ; else RZ← 0;
grammatical	bgenr32 rz, rx
clarification	If RX[5] is 0, then set RZ the register bits determined by RX five bits lower (RX[4:0]) and clear RZ of all its bits; otherwise, clear RZ.
impact indicator	unaffected

User's Manual	
aspiratio	n
exceptio	not have

32-bit instruction format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1	10001	00000	RX	010100	00010	RZ

Figure 14.30: BGENR

14.18 BHSZ - register greater than or equal to zero branching instruction

Harmonization Directive	
grammatica 1	bhsz rx, label
manipulate	Register greater than or equal to zero transfers the program if($RX \geq 0$) $PC \leftarrow PC + \text{sign_extend}(\text{offset} \ll 1)$; else $pc \leftarrow pc + 4$;
compiling in the end	32-bit instructions only bhsz32 rx, label
clarification	If register RX is greater than or equal to zero, the program transfers to label; otherwise, the program executes the next instruction, i.e., $PC \leftarrow PC + 4$. Label is the value of the current program PC plus the 16-bit relative offset signed extension to 32 bits shifted 1 bit to the left. The transfer range for the instruction is $\pm 64KB$ of address space.
Markers of influence classifier for honorific people	unaffected
exceptions	not have

32 bit instruction
User's Manual

manipulate	Register greater than or equal to zero transfers the program if($RX \geq 0$) $PC \leftarrow PC + \text{sign_extend}(\text{offset} \ll 1)$ else $PC \leftarrow PC + 4$
grammatical	bhsz32 rx, label
clarification	If register RX is greater than or equal to zero, the program transfers to label; otherwise, the program executes the next instruction, i.e., $PC \leftarrow PC + 4$. Label is the value of the current program PC plus the 16-bit relative offset signed extension to 32 bits shifted 1 bit to the left. The transfer range for the instruction is $\pm 64KB$ of address space.
Markers of influence classifier for honorific people	unaffected
exceptions	not have

32-bit instruction format:

31	30	26	25	21	20	16	15	0
1	11010		01101	RX			Offset	

Figure 14.31: BHSZ

14.19 BHZ - Register Greater Than Zero Branch Instruction

Harmonization Directive	
gram matica l	bhz rx, label
manip ulate	Register greater than zero transfers the program if($RX > 0$) $PC \leftarrow PC + \text{sign_extend}(\text{offset} \ll 1)$ else $PC \leftarrow PC + 4$
com pilin g in the end	32-bit instructions only bhz32 rx, label
clarifi cation	If register RX is greater than zero, the program transfers to label; otherwise, the program executes the next instruction, i.e., $PC \leftarrow PC + 4$. Label is the value of the current program PC plus the 16-bit relative offset after signed extension to 32 bits shifted by 1 bit to the left. The transfer range for the instruction is $\pm 64KB$ of address space.
Mar ker s of infl uen ce classi fier for hono rific peopl e	unaffected
except	not have

32-bit instruction	
manipulate	Register greater than zero transfers the program if($RX > 0$) $PC \leftarrow PC + \text{sign_extend}(\text{offset} \ll 1)$ else $PC \leftarrow PC + 4$
grammatical	bhz32 rx, label
clarification	If register RX is greater than zero, the program transfers to label; otherwise, the program executes the next instruction, i.e., $PC \leftarrow PC + 4$. Label is the value of the current program PC plus the 16-bit relative offset after the signed extension to 32 bits shifted by 1 bit to the left. The transfer range for the instruction is $\pm 64KB$ of address space.
Markers of influence classifier for honoriific people	unaffected
exceptions	not have

32-bit instruction format:

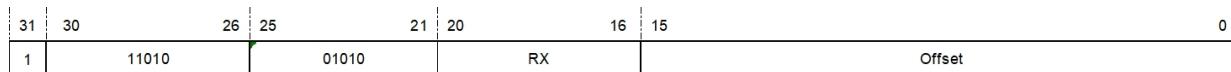


Figure 14.32: BHZ

14.20 BKPT - Breakpoint Instruction

Harmonization Directive	
manipulate	Raise a breakpoint exception or enter debug mode

Compilation results	Always compiles to 16-bit instructions. bkpt16
clarification	breakpoint instruction
Affecting Flags	unaffected
exceptions	breakpoint exception

16-bit instructions	
manipulate	Raise a breakpoint exception or enter debug mode
grammatical	bkpt16
clarification	breakpoint instruction
Influence Flag Bit	unaffected
exceptions	breakpoint exception

16-bit instruction format:

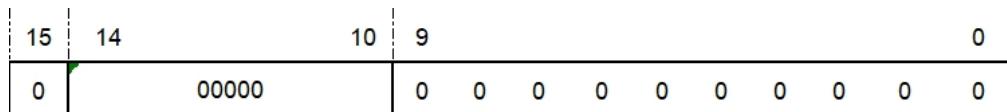


Figure 14.33: BKPT

14.21 BLSZ - Register Less Than or Equal to Zero Branching Instruction

Harmonization Directive	
gram matica l	blsz rx, label
manip ulate	Program transfer if register is less than or equal to zero if(RX<= 0) PC ← PC + sign_extend(offset<< 1) else PC ← PC + 4
com pilin g in the end	32-bit instructions only blsz32 rx, label
clarifi cation	If register RX is less than or equal to zero, the program transfers to label for execution; otherwise the program executes the next instruction, PC PC + 4. Label is the value of the current program PC plus the 16-bit relative offset signed expanded to 32 bits by shifting it 1 bit to the left. The transfer range for the instruction is ± 64KB of address space.
Mar kers of infl uen ce classi fier for hono rific peopl e	unaffected

Gentech E804
except Manual
ions

XUANTIE 玄铁

32-bit instruction	
manipulate	Program transfer if register is less than or equal to zero if($RX \leq 0$) $PC \leftarrow PC + \text{sign_extend}(\text{offset} \ll 1)$ else $PC \leftarrow PC + 4$
grammatical	blsz32 rx, label
clarification	If register RX is less than or equal to zero, the program transfers to label for execution; otherwise the program executes the next instruction, PC + 4. Label is the value of the current program PC plus the 16-bit relative offset signed expanded to 32 bits by shifting it 1 bit to the left. The transfer range for the instruction is $\pm 64KB$ of address space.
Markers of influence classifier for honorific people	unaffected
exceptions	not have

32-bit instruction format:



Figure 14.34: BLSZ

14.22 BLZ - Register Less Than Zero Branch Instruction

Harmonization Directive	
gram matica 1	blz rx, label
manip ulate	Program transfer if register is less than zero if($RX < 0$) $PC \leftarrow PC + \text{sign_extend}(\text{offset} \ll 1)$ else $PC \leftarrow PC + 4$
com pilin g in the end	32-bit instructions only blz32 rx, label
clarifi cation	If register RX is less than zero, the program transfers to label; otherwise, the program executes the next instruction, i.e., $PC \leftarrow PC + 4$. Label is the value of the current program PC plus the 16-bit relative offset signed extension to 32 bits shifted by 1 bit to the left. The transfer range for the instruction is $\pm 64KB$ of address space.
Mar ker s of infl uen ce classi fier for hono rific peopl e	unaffected
except ions	not have

32-bit instruction	
manip ulate	Program transfer if register is less than zero if($RX < 0$) $PC \leftarrow PC + \text{sign_extend}(\text{offset} \ll 1)$ else $PC \leftarrow PC + 4$

gram users Manual matica l	blz32 rx, label
clarifi cation	If register RX is less than zero, the program transfers to label; otherwise, the program executes the next instruction, i.e., $PC \leftarrow PC + 4$. Label is the value of the current program PC plus the 16-bit relative offset signed extension to 32 bits shifted by 1 bit to the left. The transfer range for the instruction is $\pm 64KB$ of address space.
Mar ker s of infl uen ce classi fier for hono rific peopl e	unaffected
except ions	not have

32-bit instruction format:

31	30	26	25	21	20	16	15	0
1	11010		01100		RX		Offset	

Figure 14.35: BLZ

14.23 BMASKI - Immediate Digit Mask Generation Instruction

Harmonization Directive	
tell to	bmaski rz, oimm5
Budd	
hist	
teach	
ing	
drill	$RZ \leftarrow (2)^{OIMM5} - 1$
(prac	
tice)	
writi	
ngs	
or	
work	
s	
c	32-bit instructions only
o	bmaski32 rz, oimm5
m	
pi	
la	
ti	
o	
n	
re	
s	
ul	
t	
resol	
ute	
cl	Generates an immediate number with consecutive low bits of 1 and high bits of 0 and stores the immediate number in RZ. The biased immediate number OIMM5 specifies the number of bits in the consecutive low bits (RX[OIMM5-1:0]) that are set to 1, and the remaining high bits are cleared to zero. When OIMM5 is 0 or 32, all bits of RX are set to 1.
ar	
ifi	
ca	
ti	
o	
n	Note that OIMM5 of 1-16 is executed by the movi instruction.

User's Manual	affected
ar k er s of in fl u e n ce classi fier for hono rific peop le	
set a limit (on) regul ate	Immediate numbers range from 0, 17-32;
discrimina te const ant	not have

32-bit instruction	
drill (pra ctice) writi ngs or wor ks	RZ $\leftarrow (2)^{OIMM5} - 1$
tell to Bud dhist teac hing	bmaski32 rz, oimm5
cl ar ifi ca ti o n	Generates an immediate number with consecutive low bits of 1 and high bits of 0 and stores the immediate number in RZ. The biased immediate number OIMM5 specifies the number of bits in the consecutive low bits (RX[OIMM5-1:0]) that are set to 1, and the remaining high bits are cleared to zero. When OIMM5 is 0 or 32, all bits of RX are set to 1. Note that OIMM5 of 1 - 16 is executed by the movi instruction; the binary operand IMM5 is equal to OIMM5 - 1.
M ar k er s of in fl u e n ce class ifier for hon orifi c peop le	unaffected
set a limit	Immediate numbers range from 0, 17-32;

(on) User's Manual regu late	not have
---------------------------------------	----------

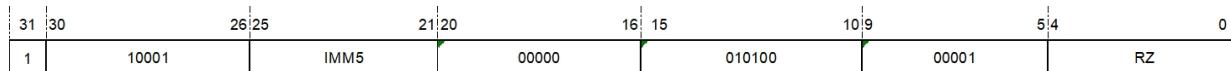
32-bit instruction format:

Figure 14.36: BMASKI

IMM5 domain:

Specifies the highest bit of the consecutive low bits that are set to 1.

Note: The immediate number OIMM5 is biased by 1 compared to the binary operand IMM5.

10,000:

0-16 position bits

10001:

0-17 position bits

.....

11111:

0-31 position bits

14.24 BNEZ - Register Not Equal to Zero Branch Instruction

Harmonization Directive	
grammatical	bnez rx, label
manipulate	Program transfer if register not equal to zero if(RX != 0) PC ← PC + sign_extend(offset<< 1) else PC ← PC + 4
coming in the end	Only 32-bit instructions exist. bnez32 rx, label
clarification	If register RX is not equal to zero, the program transfers to label; otherwise, the program executes the next instruction, i.e., $PC \leftarrow PC + 4$. Label is the value of the current program PC plus the 16-bit relative offset signed extension to 32 bits shifted by 1 bit to the left. The transfer range for the instruction is $\pm 64KB$ of address space.
Markers of influence classifier for honorific people	unaffected
exceptions	not have

32-bit instruction

User	Program	Program transfer if register not equal to zero if($RX \neq 0$) $PC \leftarrow PC + sign_extend(offset \ll 1)$ else $PC \leftarrow PC + 4$
grammatical	bnez32 rx, label	
clarification	If register RX is not equal to zero, the program transfers to label; otherwise, the program executes the next instruction, i.e., $PC \leftarrow PC + 4$. Label is the value of the current program PC plus the 16-bit relative offset signed extension to 32 bits shifted by 1 bit to the left. The transfer range for the instruction is $\pm 64KB$ of address space.	
Markers of influence classifier for honorific people	unaffected	
except ions	not have	

32-bit instruction format:

31	30	26	25	21	20	16	15	0
1		11010		01001		RX		Offset

Figure 14.37: BNEZ

14.25 BNEZAD - Register Self-Decrement Greater Than Zero Branch Instruction

standard directives	
grammatical	bnezad rx, label
manipulate	The register is judged after subtracting 1, if it is greater than zero, the program is transferred, and the result of subtraction unconditionally updates the register. RX ← RX-1 if(RX > 0) PC ← PC + sign_extend(offset<< 1) else PC ← PC + 4
compilation result resolute	Only 32-bit instructions exist. bnez32 rx, label

Explanation :	If the result is greater than zero, the program is transferred to label; otherwise, the program executes the next instruction, i.e., $PC \leftarrow PC + 4$. The result of self-subtraction unconditionally updates register RX. Label is the value of the current program PC plus the 16-bit relative offset signed expanded to 32 bits by 1 bit to the left. The transfer range for the instruction is $\pm 64KB$ of address space.
Marker of influence	unaffected

Bit	User's Manual
discriminate Regulator:	not have

32 in d e x (m at h.) hono rific title	
Ope rate : tell to Law:	<p>Program transfer if register not equal to zero $RX \leftarrow RX - 1$ $\text{if}(RX > 0)$ $PC \leftarrow PC + \text{sign_extend}(\text{offset} \ll 1)$ else $PC \leftarrow PC + 4$</p>
Exp lana tion : Mark er s of in fl u e n ce Bit:	<p>If the result is greater than zero, the program is transferred to label; otherwise, the program executes the next instruction, i.e., $PC \leftarrow PC + 4$. The result of self-subtraction unconditionally updates register RX.</p> <p>Label is the value of the current program PC plus the 16-bit relative offset signed expanded to 32 bits by 1 bit to the left.</p> <p>The transfer range for the instruction is $\pm 64KB$ of address space.</p>
discr imin ate Reg ular:	unaffected
discr imin ate Reg ular:	not have

31	30	26	25	21	20	16	15	0
1	11010		00001		Rx			Offset

Figure 14.38: BNEZAD

14.26 BR - Unconditional Jump Instruction

Harmonization Directive	
grammatical	br label
manipulate	$PC \leftarrow PC + \text{sign_extend}(\text{offset} \ll 1)$
Compilation results	Compiles to 16-bit or 32-bit instructions depending on the jump range. if($\text{offset} < 1\text{KB}$), then br16 label; else br32 label.
instructions	The program jumps unconditionally to label for execution. Label is the value of the current program PC plus a relative offset shifted 1 bit to the left after a signed extension to 32 bits.
Markers of influence classifier for honorific people	unaffected
exceptions	not have

16-bit instructions	
manipulate	$PC \leftarrow PC + \text{sign_extend}(\text{offset} \ll 1)$
grammatical	br16 label
clarification	The program jumps unconditionally to label for execution. Label is the value of the current program PC plus a 10-bit relative offset shifted 1 bit to the left after a signed extension to 32 bits. The jump range for the instruction is $\pm 1\text{KB}$ of address space.
impact	unaffected
Logo classifier for honorific	

User Manual	
except ions	not have

16-bit instruction format:

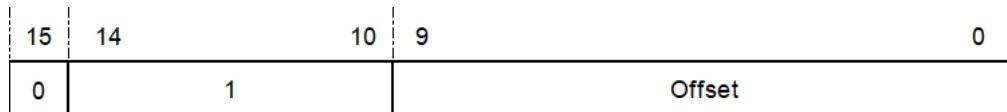


Figure 14.39: BR-1

32-bit instruction	
manipulate	PC ← PC + sign_extend(offset << 1)
grammatical	br32 label
clarification	The program jumps unconditionally to label for execution. Label is the value of the current program PC plus the 16-bit relative offset signed extension to 32 bits shifted by 1 bit. The jump range for the instruction is ± 64KB of address space.
impact	unaffected
Logo classifier for honorific people	
exceptions	not have

32-bit instruction format:

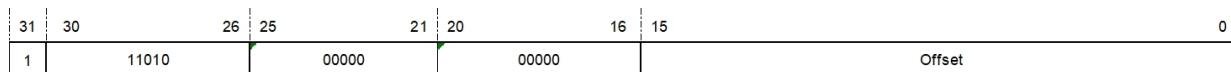


Figure 14.40: BR-2

14.27 BREV - Bit Reverse Order Instruction

Harmonization Directive	
grammatical	brev rz, rx
manipulate	for i=0 to 31 RZ[i] ← RX[31-i].
Translation in the end	Only 32-bit instructions exist. brev32 rz, rx

Users Manual	The value of RX is taken in reverse order by bit and the result is stored in RZ.
cation	If the value of RX is "abc defghijklmnopqrstuvwxyz012345", then the value of RZ becomes "543210zyxwvutsrqponmlkjihgfedcba".
impac t	unaffected
Logo classifier for honorific people	
except ions	not have

32-bit instruction	
manipulate	for i=0 to 31 RZ[i]← RX[31-i].
grammatical	brev32 rz, rx
clarification	The value of RX is taken in reverse order by bit and the result is stored in RZ. If the value of RX is "abc defghijklmnopqrstuvwxyz012345", then the value of RZ becomes " 543210zyxwvutsrqponmlkjihgfedcba".
impact Logo classifier for honorific people	unaffected
exceptions	not have

Command Format:

Figure 14.41: BREV

14.28 BSETI - Immediate Number Position Bit Instruction

Harmonization Directive		
tell to Budd hist teach ing	bseti rz, imm5	bseti rz, rx, imm5
drill (practice) writings or works	RZ ← RZ[IMM5] reset	RZ ← RX[IMM5] is set
C o m p i l a t i o n r e s u l t s	Compile to 16-bit or 32-bit depending on the register range. Bit Commands. if (z<8), then bseti16 rz, imm5; else bseti32 rz, rz, imm5;	Compiles to the corresponding 16-bit or 32-bit instructions depending on the register range. if((x==z) and (z<8)), then bseti16 rz, imm5; else bseti32 rz, rx, imm5;
persuade generic term for a sacrifice the gods	Set the position 1 of the value of RZ/RX indicated by the value of the IMM5 field, leave the rest of the bits unchanged, and store the result of the setting into RZ.	

User's Manual	Unaffected
Impact Marker classifier for honorific people	The range of immediate numbers is 0-31.
discriminate constant	not have

16-bit instructions	
manipulate	RZ \leftarrow RZ[IMM5] reset
grammatical	bseti16 rz, imm5
clarification	Set the position 1 of the value of RZ indicated by the value of the IMM5 field, leave the remaining bits unchanged, and store the result of the setting into RZ.
Impact Marker classifier for honorific people	unaffected
limitation	The range of registers is r0-r7; the range of immediate numbers is 0-31.
exceptions	not have

16-bit instruction format:

15	14	11	10	8	7	5	4	0
0	0111	RZ		101		IMM5		

Figure 14.42: BSETI-1

32-bit instruction	
manipulate	RZ ← RX[IMM5] is set
grammatical	bseti32 rz, rx, imm5
clarification	Set the position 1 of the value of RX indicated by the value of the IMM5 field, leave the remaining bits unchanged, and store the result of the setting to RZ.
Impact Marker classifier for honorific people	unaffected
limitation	The range of immediate numbers is 0-31.
exceptions	not have

32-bit instruction format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1	10001	IMM5	RX	001010	00010	RZ

Figure 14.43: BSETI-2

14.29 BSR - jump to subroutine instruction

collectively known as "harmonization" or "harmonization index" (math.) honorific title	
grammatical	bsr label

User's Manual
manipulate
on results

Description:	The subroutine jumps, the return address of the subroutine (the PC of the next instruction) is stored in link register R15, and the program transfers to the Implemented at the label. Label is the value of the current program PC plus a relative offset shifted 1 bit to the left after a signed extension to 32 bits.
impact Logo Bit:	unaffected
Exception:	not have

16 in d e x (m at h.) hono rific title	
Operate : tell to Law:	Link and jump to the subroutine: R15 \leftarrow PC+ 2 PC \leftarrow PC+ sign_extend(offset<< 1)
Explana tion :	The subroutine jumps, saving the return address of the subroutine (the PC of the next instruction, i.e., the current PC + 2) in the link register R15 in which the program is transferred to label execution. Label is the value of the current program PC plus the 10-bit relative offset after a 1-bit leftward signed extension to 32 bits. The jump range for the instruction is $\pm 1KB$ of address space.
Mark er s of in fl	unaffected

User's Manual e n ce Bit:	
set a limit (on) Syste m:	The jump target of a BSR16 instruction cannot be the BSR16 instruction itself.
discri mina te Regu lar:	not have

Command Format:

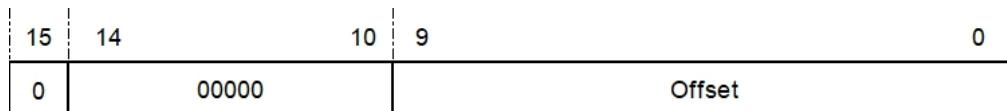


Figure 14.44: BSR-1

Offset field - Specifies the relative offset of the jump.

Note: The relative offset (Offset) of the jump cannot be 0x0.

32	
in d e x (m at h.) hono rific title	
Ope rate : tell to Law:	Link and jump to the subroutine: $R15 \leftarrow PC+4$ $PC \leftarrow PC + sign_extend(offset \ll 1)$
Exp lana tion : M ar k er s of in fl u e n ce Bit:	The subroutine jumps and saves the return address of the subroutine (PC of the next instruction, i.e., current PC+4) in the link register R15 in which the program is transferred to label execution. Label is the value of the current program PC plus a 26-bit relative offset shifted 1 bit to the left after signed expansion to 32 bits. The jump range for the order is $\pm 64MB$ address space.
discri mina te Regu lar:	unaffected
	not have

Command Format:

31	30	-	26	25	0
1	11000			Offset	

Figure 14.45: BSR-2

14.30 BT--C is a 1 branch command.

Harmonization Directive	
gram matica 1	bt label
manip ulate	if(C== 1) PC← PC+ sign_extend(offset<< 1); else PC← next PC;
Translati on resul ts	Compiles to the corresponding 16-bit or 32-bit instruction depending on the range of the jump. if (offset<1KB), then bt16 label; else bt32 label.
clarifi cation	If the condition flag bit C is equal to 1, the program transfers label; otherwise, the program executes the next instruction. Label is the value of the current program PC plus the relative offset shifted by 1 bit to the left after signed extension to 32 bits, and the value of the BT instruction. The shift range is ± 64KB address space.
Impact Sign s classifi er for honor ific peopl e	unaffected
except ions	not have

16-bit instructions	
manip ulate	C Equivalent to a transfer of proceedings if(C== 1) PC← PC+ sign_extend(offset<< 1) else PC← PC+ 2
gram matica 1	bt16 label

Labels	If the condition flag C is equal to 1, the program transfers label; otherwise, the program executes the next instruction, i.e., $PC \leftarrow PC + 2$. Label is the value of the current program PC plus the 10-bit relative offset after a 1-bit left shift to 32 bits of signed extension to BT16. The transfer range for the instruction is $\pm 1KB$ of address space.
Markers of influence classifier for honorific people	unaffected
exceptions	not have

16-bit instruction format:

15	14	10	9	0
0	00010			Offset

Figure 14.46: BT-1

32-bit instruction	
manipulate	C Equivalent to a transfer of proceedings if(C== 1) PC← PC+ sign_extend(offset<< 1) else PC← PC+ 4
grammatical	bt32 label
clarification	If the condition flag C is equal to 1, the program transfers label; otherwise, the program executes the next instruction, i.e., PC ← PC + 4. Label is the value of the current program PC plus the 16-bit relative offset signed extension to 32 bits shifted by 1 bit to the left. The transfer range for the order is ± 64KB address space.
Markers of influence classifier for honorific people	unaffected
exceptions	not have

32-bit instruction format:

31	30	26	25	21	20	16	15	0
1	11010	00011		00000				Offset

Figure 14.47: BT-2

14.31 BTSTI - Immediate Digital Test Instruction

Harmonization Directive	
grammatical	btsti rx, imm5
manipulate	C← RX[IMM5]
Compilation results	32-bit instructions only btsti32 rx, imm5
clarification	The bit of RX determined by IMM5 (RX[IMM5]) is tested and the value of condition bit C is made equal to the value of this bit.
Affecting Flags	C← RX[IMM5]
limitation	The range of immediate numbers is 0-31.
exceptions	not have

32-bit instruction	
manipulate	C← RX[IMM5]
grammatical	btsti32 rx, imm5
clarification	The bit of RX determined by IMMS (RX[IMM5]) is tested and the value of condition bit C is made equal to the value of this bit.
Affecting Flags	C← RX[IMM5]
limitation	The range of immediate numbers is 0-31.
exceptions	not have

32-bit instruction format:

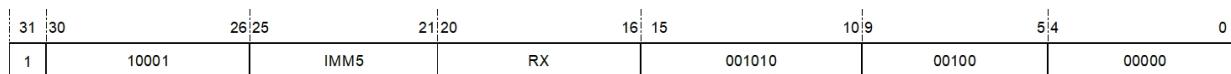


Figure 14.48: BTSTI

14.32 CLRF--C is 0 Clear command.

Harmonization Directive	
grammatical	clrf rz
manipulate	if C==0, then RZ ← 0; else RZ← RZ.
Compilation results	32-bit instructions only clrf32 rz
clarification	If C is 0, register RZ is cleared; otherwise, register RZ remains unchanged.
Affecting Flags	unaffected
exceptions	not have

32-bit instruction	
manipulate	if C==0, then RZ ← 0; else RZ← RZ.
grammatical	clrf32 rz

User's Manual¹

clarification	If C is 0, register RZ is cleared; otherwise, register RZ remains unchanged.
Influence Flag Bit	unaffected
exceptions	not have

32-bit instruction format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1	10001	RZ	00000	001011	00001	00000

Figure 14.49: CLRF

14.33 CLRT--C is a 1 clear command.

Harmonization Directive	
grammatical	clrt rz
manipulate	if C==1, then RZ ← 0; else RZ← RZ.
Compilation results	32-bit instructions only clrt32 rz
clarification	If C is 1, register RZ is cleared; otherwise, register RZ remains unchanged.
Affecting Flags	unaffected
exceptions	not have

32-bit instruction	
manipulate	if C==1, then RZ ← 0; else RZ← RZ.
grammatical	clrt32 rz
clarification	If C is 1, register RZ is cleared; otherwise, register RZ remains unchanged.
Affecting Flags	unaffected
exceptions	not have

32-bit instruction format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1	10001	RZ	00000	001011	00010	00000

Figure 14.50: CLRT

14.34 CMPHS - unsigned greater-than-equal comparison instruction

Harmonization Directive	
gram matica l	cmphs rx, ry
manip ulate	RX compares unsigned with RY. If RX>= RY, then C $\leftarrow 1;$ else $C \leftarrow 0;$
Translati on resu lts	Compiles to the corresponding 16-bit or 32-bit instructions depending on the register range. if ($x < 16$) and ($y < 16$), then cmphs16 rx, ry; else cmphs32 rx, ry.
clarifi cation	The value of RX is subtracted from value of RY, the result compared to 0, and bit C is updated. cmphs performs unsigned comparisons, i.e., the operands are considered unsigned. If RX is greater than or equal to RY, i.e., the result of the subtraction is greater than or equal to 0, the condition bit C is set; no Then, clear condition bit C.
Imp act Sign s classif ier for honor ific peopl e	Setting the condition bit according to the comparison result C
except ions	not have

16-bit instructions	
manip ulate	RX compares unsigned with RY. If RX>= RY, then C $\leftarrow 1;$ else $C \leftarrow 0;$
gram	cmphs16 rx, ry

Technical Users Manual 1	
instructions	The value of RX is subtracted from the value of RY, the result is compared to 0, and bit C is updated. cmphs16 performs an unsigned comparison, i.e., the operands are considered to be unsigned. If RX is greater than or equal to RY, i.e., the result of the subtraction is greater than or equal to 0, the condition bit C is set; no Then, clear condition bit C.
Impact Signs classifier for honorable people	Setting the condition bit according to the comparison result C
limitation	The range of the register is r0-r15.
exceptions	not have

16-bit instruction format:

15	14	10	9	6	5	2	1	0
0		11001		RY		RX		00

Figure 14.51: CMPHS-1

32-bit instruction	
manipulate	RX compares unsigned with RY. If RX>= RY, then C $\leftarrow 1;$ else $C \leftarrow 0;$
grammatical	cmphs32 rx, ry
clarification	The value of RX is subtracted from the value of RY, the result is compared to 0, and bit C is updated. cmphs32 performs unsigned comparisons, i.e., the operands are considered unsigned. If RX is greater than or equal to RY, i.e., the result of the subtraction is greater than or equal to 0, the condition bit C is set; no Then, clear condition bit C.
Impact Signs classifier for honorific people	Setting the condition bit according to the comparison result C
exceptions	nothave

32-bit instruction format:

31	30	26 25	21 20	16 15	10 9	5 4	0
1		10001	RY	RX	000001	00001	00000

Figure 14.52: CMPHS-2

14.35 CMPHSI - Immediate Number Unsigned Greater Than or Equal Compare Instruction

Harmonization Directive	
tell to Bud dhist teac hing	cmphsi rx, oimm16
m a ni p ul at e	RX Unsigned comparison with immediate numbers. If RX >= zero_extend(OIMM16), C ← 1. else C← 0;
C o m pi la ti o n re s ul ts	Compiles to the corresponding 16-bit or 32-bit instructions depending on the range of the immediate numbers and registers. if (oimm16<33) and (x<8),then cmphsi16 rx, oimm5. else cmphsi32 rx, oimm16.
cl ar ifi ca ti o n	Extend the 16-bit immediate number (OIMM16) zero with bias 1 to 32 bits, then subtract that 32-bit value from the value of RX, and the result is the same as the value of 0 is compared and the C bit is updated. cmphsi performs unsigned comparisons, i.e., the operands are considered unsigned. If RX is greater than or equal to OIMM16 after zero expansion, i.e., the result of subtraction is greater than or equal to 0, then condition bit C is set; otherwise, condition bit C is cleared.
M ar k er s of in fl u e	Setting the condition bit according to the comparison result C

User's Manual	
set a limit (on) regulate	The range of immediate numbers is 0x1-0x10000.
discriminate constant	not have

16-bit instructions	
m a ni p ul at e	RX Unsigned comparison with immediate numbers. If RX >= zero_extend(OIMM5), then C $\leftarrow 1;$ else $C \leftarrow 0;$
tell to Bud dhis t teac hing	cmphsi16 rx, oimm5
cl ar ifi ca ti o n	Zero-extends a 5-bit immediate number with bias 1 (OIMM5) to 32 bits, then subtracts that 32-bit value from the value of RX, compares the result to 0, and updates bit C. cmphsi16 performs unsigned comparisons, i.e., the operand is considered an unsigned number. If RX is greater than or equal to OIMM5 after zero expansion, i.e., the result of the subtraction is greater than or equal to 0, then condition bit C is set; otherwise, condition bit C is cleared. Note: The binary operand IMM5 is equal to OIMM5 - 1.
M ar k er s of in fl u e n ce clas sifie r for hon orifi c peo ple	Setting the condition bit according to the comparison result C
set a limi t (on)	The range of registers is r0-r7; the range of immediate numbers is 1-32.

User's Manual	late
disc	not have
rimi	
nate	
cons	
tant	

16-bit instruction format:

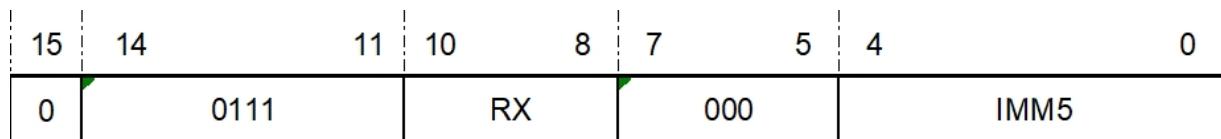


Figure 14.53: CMPHSI-1

IMM5 domain:

Specifies a value without a bias immediate number.

Note: The immediate number OIMM5 involved in the comparison is biased by 1 compared to the binary operand IMM5.

00000:

Comparison with 1

00001:

Comparison with 2

.....

11111:

Comparison with 32

32-bit instruction	
m a ni p ul at e	RX Unsigned comparison with immediate numbers. If RX >= zero_extend(OIMM16), then C $\leftarrow 1;$ else $C \leftarrow 0;$
tell to Bud dhis t teac hing	cmphsi32 rx, oimm16
cl ar ifi ca ti o n	Zero-extends a 16-bit immediate number (OIMM16) with bias 1 to 32 bits, then subtracts that 32-bit value from the value of RX, compares the result to 0, and updates bit C. cmphsi32 performs unsigned comparisons, i.e., the operand is considered an unsigned number. If RX is greater than or equal to the zero-extended OIMM16, i.e., the result of the subtraction is greater than or equal to 0, then condition bit C is set; otherwise, condition bit C is cleared. Note: The binary operand IMM16 is equal to OIMM16 - 1.
M ar k er s of in fl u e n ce clas sifie r for hon orifi c peo ple	Setting the condition bit according to the comparison result C
set a limi t	The range of immediate numbers is 0x1-0x10000.

(on)	Users Manual
regu	
late	
disc	not have
rimi	
nate	
cons	
tant	

32-bit instruction format:

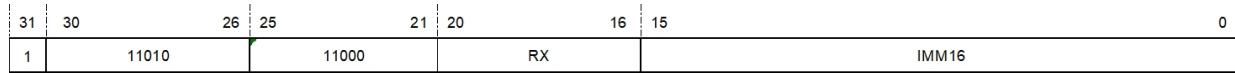


Figure 14.54: CMPHSI-2

IMM16 domain:

Specifies a value without a bias immediate number.

Note: The immediate number OIMM16 involved in the comparison is biased by 1 compared to the binary operand IMM16.

0000000000000000:

Compare to 0x1

0000000000000001:

Compare to 0x2

.....

1111111111111111:

Compare to 0x10000

14.36 CMPLT - Signed Less Than Compare Instruction

Harmonization Directive	
gram matica 1	cmplt rx, ry
manip ulate	RX makes a signed comparison with RY. If RX < RY, then C $\leftarrow 1;$ else $C \leftarrow 0;$
Translati on resu lts	Compiles to the corresponding 16-bit or 32-bit instructions depending on the register range. if ($x < 16$) and ($y < 16$), then cmplt16 rx, ry; else cmplt32 rx, ry.
clarifi cation	The value of RX is subtracted from the value of RY, the result is compared to 0, and bit C is updated. cmplt performs signed comparisons, i.e., the operands are considered to be signed numbers in complement form. If RX is less than RY, i.e., the result of subtraction is less than 0, the condition bit C is set; otherwise, the condition bit C is set. Then, clear condition bit C.
Impact Sign s classifi er for honor ific peopl e	Setting the condition bit according to the comparison result C
except ions	not have

16-bit instructions	
manip ulate	RX makes a signed comparison with RY. If RX < RY, then C $\leftarrow 1;$ else $C \leftarrow 0;$

gram users Manual matica l	cmplt16 rx, ry
clarifi cation	The value of RX is subtracted from value of RY, the result compared to 0, and bit C is updated. cmplt16 performs signed comparisons, i.e., the operands are considered to be signed numbers in complement form. If RX is less than RY, i.e., the result of the subtraction is less than 0, the condition bit C is set; Otherwise, clear condition bit C.
Mar ker s of infl uen ce classif ier for honor ific peopl e	Setting the condition bit according to the comparison result C
limitat ion	The range of the register is r0-r15.
except ions	not have

16-bit instruction format:



Figure 14.55: CMPLT-1

32-bit instruction	
manipulate	RX makes a signed comparison with RY. If RX < RY, then C $\leftarrow 1;$ else $C \leftarrow 0;$
grammatical	cmplt32 rx, ry
clarification	The value of RX is subtracted from value of RY, the result compared to 0, and bit C is updated. cmplt32 performs signed comparisons, i.e., the operands are considered to be signed numbers in complement form. If RX is less than RY, i.e., the result of the subtraction is less than 0, the condition bit C is set; Otherwise, clear condition bit C.
Markers of influence classifier for honorific people	Setting the condition bit according to the comparison result C
exceptions	not have

32-bit instruction format:

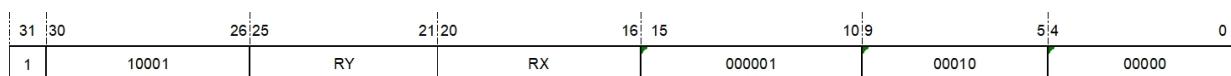


Figure 14.56: CMPLT-2

14.37 CMPLTI - Immediate Number Signed Less Than Compare Instruction

Harmonization Directive	
tell to Bud dhist teac hing	cmplti rx, oimm16
m a ni p ul at e	RX Signed comparison with immediate numbers. If RX < zero_extend(OIMM16), C ← 1; else C ← 0;
C o m pi la ti o n re s ul ts	Compiles to the corresponding 16-bit or 32-bit instructions depending on the range of the immediate numbers and registers. if (x<8) and (oimm16<33), then cmplti16 rx, oimm5; else cmplti32 rx, oimm16.
cl ar ifi ca ti o n	Zero-extends a 16-bit immediate number (OIMM16) with bias 1 to 32 bits, and then subtracts that 32-bit value from the value of RX, comparing result to 0 and updating bit C. cmplti performs signed comparisons, i.e., the value of RX is considered to be a signed number in complement form. If RX is less than the zero-extended OIMM16, i.e., the result of the subtraction is less than 0, the condition bit C is set; otherwise, the condition bit is cleared. C.
M ar k er s of in fl u e	Setting the condition bit according to the comparison result C

User's Manual	
set a limit (on) regulate	The range of immediate numbers is 0x1-0x10000.
discriminate constant	not have

16-bit instructions	
m a ni p ul at e	RX Signed comparison with immediate numbers. If RX < zero_extend(OIMM5), then C $\leftarrow 1;$ else $C \leftarrow 0;$
tell to Bud dhis t teac hing	cmplti16 rx, oimm5
cl ar ifi ca ti o n	Zero-extends a 5-bit immediate number (OIMM5) with bias 1 to 32 bits, and then subtracts that 32-bit value from the value of RX, comparing the result to 0 and updating bit C. cm plti16 performs signed comparisons, i.e., the value of RX is considered to be a signed number in complement form. If RX is less than zero-extended OIMM5, i.e., the result of the subtraction is less than 0, condition bit C is set; otherwise, condition bit C is cleared. Note: The binary operand IMM5 is equal to OIMM5 - 1.
M ar k er s of in fl u e n ce clas sifie r for hon orifi c peo ple	Setting the condition bit according to the comparison result C
set a limi t (on)	The range of registers is r0-r7; the range of immediate numbers is 1-32.

User's Manual	
late	
disc	not have
rimi	
nate	
cons	
tant	

16-bit instruction format:

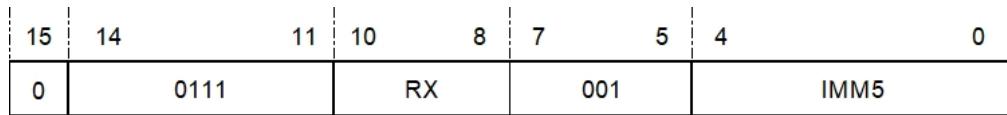


Figure 14.57: CMPLT-1

IMM5 domain:

Specifies a value without a bias immediate number.

Note: The immediate number OIMM5 involved in the comparison is biased by 1 compared to the binary operand IMM5.

00000:

Comparison with 1

00001:

Comparison with 2

.....

11111:

Comparison with 32

32-bit instruction	
m a ni p ul at e	RX Signed comparison with immediate numbers. If RX< zero_extend(OIMM16), then C $\leftarrow 1;$ else $C \leftarrow 0;$
tell to Bud dhis t teac hing	cmplti32 rx, oimm16
cl ar ifi ca ti o n	Zero-extends a 16-bit immediate number (OIMM16) with bias 1 to 32 bits, then subtracts that 32-bit value from the value of RX, compares the result to 0, and updates bit C. cm plti32 performs signed comparisons, i.e., the value of RX is assumed to be a signed number in complement form. If RX is less than the zero-extended OIMM16, i.e., the result of the subtraction is less than 0, condition bit C is set; otherwise, condition bit C is cleared. Note: The binary operand IMM16 is equal to OIMM16 - 1.
Mark ers of in fl u e n ce clas sifie r for hon orifi c peo ple	Setting the condition bit according to the comparison result C

Set a limit (on) register	The range of immediate numbers is 0x1-0x10000.
discriminate constant	not have

32-bit instruction format:

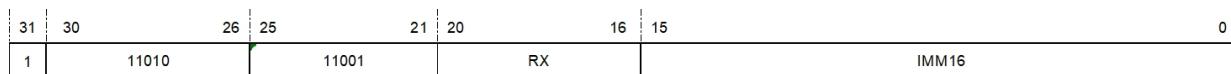


Figure 14.58: CMPLT-2

IMM16 domain:

Specifies a value without a bias immediate number.

Note: The immediate number OIMM16 involved in the comparison is biased by 1 compared to the binary operand IMM16.

0000000000000000:

Compare to 0x1

0000000000000001:

Compare to 0x2

.....

1111111111111111:

Compare to 0x10000

14.38 CMPNE - unequal comparison instruction

Harmonization Directive	
grammatical	cmpne rx, ry
manipulate	RX is compared to RY. If RX != RY, then C ← 1; If RX != else C← 0;
compilation results	Compiles to the corresponding 16-bit or 32-bit instructions depending on the register range. if (x<16) and (y<16), then cmpne16 rx, ry; else cmpne32 rx, ry.
clarification	The value of RX is subtracted from value of RY, the result compared to 0, and the C bit is updated. If RX is not equal to RY, the result is subtracted from 0 and the C bit is updated. If the result of the method is not equal to 0, condition bit C is set; otherwise, condition bit C is cleared.
affect aspiration	Setting the condition bit according to the comparison result C
exceptions	not have

16-bit instructions	
manipulate	RX is compared to RY. If RX != RY, then C ← 1; If RX != else C← 0;
grammatical	cmpne16 rx, ry
clarification	The value of RX is subtracted from value of RY, the result compared to 0, and the C bit is updated. If RX is not equal to RY, the result is subtracted from 0 and the C bit is updated. If the result of the method is not equal to 0, condition bit C is set; otherwise, condition bit C is cleared.

User's Manual	Setting the condition bit according to the comparison result C
limitation	The range of the register is r0-r15.
exceptions	not have

16-bit instruction format:

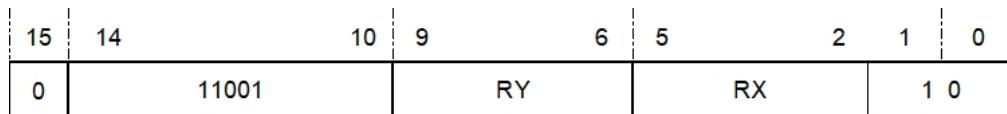


Figure 14.59: CMPNE-1

32-bit instruction	
manipulate	RX is compared to RY. If RX != RY, then C ← 1; If RX != else C← 0;
grammatical	cmpne32 rx, ry
clarification	The value of RX is subtracted from value of RY, the result compared to 0, and the C bit is updated. If RX is not equal to RY, the result is subtracted from 0 and the C bit is updated. If the result of the method is not equal to 0, condition bit C is set; otherwise, condition bit C is cleared.
affect aspiration	Setting the condition bit according to the comparison result C
exceptions	not have

32-bit instruction format:



Figure 14.60: CMPNE-2

14.39 CMPNEI - Immediate Number of Unequal Comparisons Instruction

Harmonization Directive	
grammatica 1	cmpnei rx, imm16
manipulate	RX Compare with the immediate number. If RX != zero_extend(imm16), C ← 1; If RX != else C← 0;
Translation results	Compiles to the corresponding 16-bit or 32-bit instructions depending on the range of the immediate numbers and registers. if (x<7) and (imm16<33), then cmpnei16 rx, imm5; else cmpnei32 rx, imm16.
clarification	The value of RX is subtracted from the value of the zero-extended to 32-bit 16-bit immediate number, the result is compared to 0, and the C bit is updated. If RX is not equal to the zero-extended IMM16, i.e., the result of the subtraction is not equal to 0, the condition bit C is set; otherwise, the condition bit is cleared. C.
Impact Signs classifier for honorific people	Setting the condition bit according to the comparison result C
limitation	The range of immediate numbers is 0x0-0xFFFF.
exceptions	not have

16-bit instructions	
manipulate	RX Compare with the immediate number. If RX != zero_extend(IMM5), then C ← 1; else C← 0;

gram User's Manual matica l	fmpnei16 rx, imm5
clarifi cation	The value of RX is subtracted from the value of a 5-bit immediate number expanded to 32 bits, the result is compared to 0, and the C bit is updated. The result is compared to zero and the C bit is updated. If RX is not equal to IMM5 after zero expansion, i.e., the result of subtraction is not equal to 0, then condition bit C is set; otherwise, condition bit C is cleared.
impa ct Logo classifi er for honori fic people	Setting the condition bit according to the comparison result C
limitat ion	The range of the register is r0-r7; The range of immediate numbers is 0-31.
except ions	not have

16-bit instruction format:

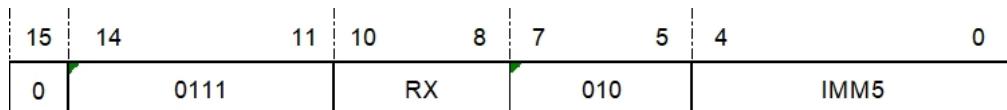


Figure 14.61: CMPNEI-1

32-bit instruction	
manipulate	RX Compare with the immediate number. If RX != zero_extend(imm16), then C $\leftarrow 1;$ else $C \leftarrow 0;$
grammatical	cmpnei rx, imm16
clarification	The value of RX is subtracted from the value of the zero-extended to 32-bit 16-bit immediate number, the result is compared with 0, and the C bit is updated. If RX is not equal to the zero-extended IMM16, i.e., the result of the subtraction is not equal to 0, the condition bit C is set; otherwise, the condition bit is cleared. C.
Impact Signs classifier for honorific people	Setting the condition bit according to the comparison result C
limitation	The range of immediate numbers is 0x0-0xFFFF.
exceptions	not have

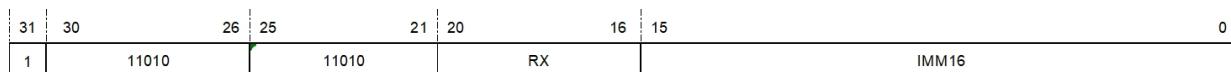
32-bit instruction format:

Figure 14.62: CMPNEI-2

14.40 DECF--C is a 0 immediate number subtraction instruction.

Harmonization Directive	
grammatical	decf rz, rx, imm5
manipulate	if C==0, then RZ ← RX - zero_extend(IMM5); else RZ← RZ.
translation	32-bit instructions only decf32 rz, rx, imm5
instructions	If the condition bit C is 0, expand the 5-bit immediate zero to 32 bits, subtract the 32-bit value from RX value, and store the result in the RZ; otherwise, the values of RZ and RX are unchanged.
impact indicator aspiration	unaffected
limitation	The range of immediate numbers is 0-31.
exceptions	not have

32-bit instruction	
manipulate	if C==0, then RZ ← RX - zero_extend(IMM5); else RZ← RZ.
grammatical	decf32 rz, rx, imm5
clarification	If the condition bit C is 0, expand the 5-bit immediate zero to 32 bits, subtract the 32-bit value from RX value, and store the result in the RZ; otherwise, the values of RZ and RX are unchanged.
impact indicator aspiration	unaffected
limitation	The range of immediate numbers is 0-31.

User's Manual
exceptio ns

32-bit instruction format:

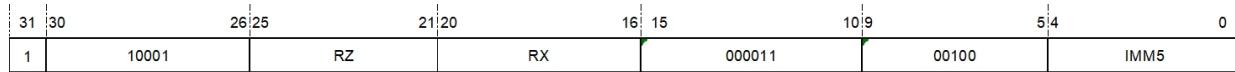


Figure 14.63: DECF

14.41 DECGT - Subtract Greater Than Zero Set C Bit Instruction

Harmonization Directive	
grammatica 1	decgtrz, rx, imm5
manipulate	RZ← RX - zero_extend(IMM5); If RZ > 0, then C← 1; else C← 0;
Translation in the end	32-bit instructions only decgtrz, rx, imm5
clarification	Extends a 5-bit immediate zero to 32 bits, and the result of subtracting that 32-bit value from RX is present in RZ. The result of the subtraction is considered to be complementary. A signed number of the form. If the result is greater than 0, condition bit C is set; otherwise, condition bit C is cleared.
impact Logo classifier for honorific people	If the result of the subtraction is greater than 0, set condition bit C; otherwise, clear condition bit C.
limitation	The range of immediate numbers is 0-31.
exceptions	not have

32-bit instruction	
manipulate	RZ← RX - zero_extend(IMM5); If RZ > 0, then C← 1; else C← 0;
grammatica 1	decgtrz, rx, imm5

clears Manual cation	Extends a 5-bit immediate zero to 32 bits, and the result of subtracting that 32-bit value from RX is present in RZ. The result of the subtraction is considered to be complementary. A signed number of the form. If the result is greater than 0, condition bit C is set; otherwise, condition bit C is cleared.
impac t Logo classifi er for honorif ic people	If the result of the subtraction is greater than 0, set condition bit C; otherwise, clear condition bit C.
limitat ion	The range of immediate numbers is 0-31.
except ions	not have

32-bit instruction format:

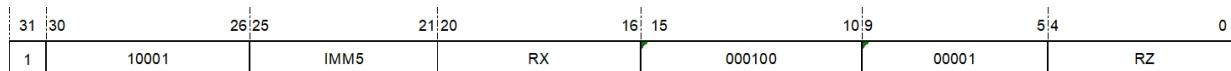


Figure 14.64: DECGT

14.42 DECLT - Subtract Less Than Zero Set C Bit Instruction

Harmonization Directive	
gram matica 1	declt rz, rx, imm5
manip ulate	RZ← RX - zero_extend(IMM5); If RZ < 0, then C← 1; else C← 0;
Trans lation in the end	32-bit instructions only declt32 rz, rx, imm5
clarifi cation	Extends a 5-bit immediate zero to 32 bits, and the result of subtracting that 32-bit value from RX is present in RZ. The result of the subtraction is considered to be complementary. A signed number of the form. If the result is less than 0, condition bit C is set; otherwise, condition bit C is cleared.
impac t Logo classifi er for honorif ic people	If the result of subtraction is less than 0, set condition bit C; otherwise, clear condition bit C.
limitat ion	The range of immediate numbers is 0-31.
except ions	not have

32-bit instruction	
manip ulate	RZ← RX - zero_extend(IMM5); If RZ < 0, then C← 1; else C← 0;
gram matica 1	declt32 rz, rx, imm5

clears Manual cation	Extends a 5-bit immediate zero to 32 bits, and the result of subtracting that 32-bit value from RX is present in RZ. The result of the subtraction is considered to be complementary. A signed number of the form. If the result is less than 0, condition bit C is set; otherwise, condition bit C is cleared.
impact Logo classifier for honorific people	If the result of subtraction is less than 0, set condition bit C; otherwise, clear condition bit C.
limitation	The range of immediate numbers is 0-31.
exceptions	not have

32-bit instruction format:



Figure 14.65: DECLT

14.43 DECNE - Subtract Not Equal to Zero Set C Bit Instruction

Harmonization Directive	
grammatical	decne rz, rx, imm5
manipulate	RZ← RX - zero_extend(IMM5); If RZ != 0, then C← 1; else C← 0;
Translation resolute	32-bit instructions only decne32 rz, rx, imm5
clarification	Extends a 5-bit immediate zero to 32 bits, and the result of subtracting that 32-bit value from RX is stored in RZ. If the result is not equal to 0, set the Set condition bit C; otherwise clear condition bit C.
impact indicator aspiration	If the result of subtraction is not equal to 0, set condition bit C; otherwise, clear condition bit C.
limitation	The range of immediate numbers is 0-31.
exceptions	not have

32-bit instruction	
manipulate	RZ← RX - zero_extend(IMM5); If RZ != 0, then C← 1; else C← 0;
grammatical	decne32 rz, rx, imm5
instructions	Extends a 5-bit immediate zero to 32 bits, and the result of subtracting that 32-bit value from RX is stored in RZ. If the result is not equal to 0, set the Set condition bit C; otherwise clear condition bit C.
impact indicator aspiration	If the result of subtraction is not equal to 0, set condition bit C; otherwise, clear condition bit C.

User's Manual	
limitation	The range of immediate numbers is 0-31.
exceptions	not have

32-bit instruction format:

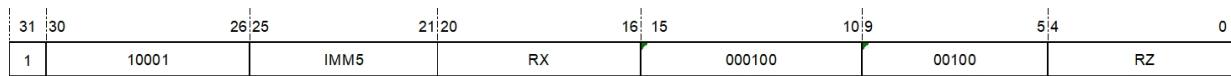


Figure 14.66: DECNE

14.44 DECT--C is a 1-immediate subtraction instruction.

Harmonization Directive	
grammatical	dect rz, rx, imm5
manipulate	if C==1, then RZ ← RX - zero_extend(IMM5); else RZ← RZ.
translation resolute	32-bit instructions only dect32 rz, rx, imm5
instructions	If the condition bit C is 1, expand the 5-bit immediate zero to 32 bits, subtract the 32-bit value from RX value, and store the result in the RZ; otherwise, the values of RZ and RX are unchanged.
impact indicator aspiratio n	unaffected
limitation	The range of immediate numbers is 0-31.
exceptions	not have

32-bit instruction	
manipulate	if C==1, then RZ ← RX - zero_extend(IMM5); else RZ← RZ.
grammatical	dect32 rz, rx, imm5
clarification	If the condition bit C is 1, expand the 5-bit immediate zero to 32 bits, subtract the 32-bit value from RX value, and store the result in the RZ; otherwise, the values of RZ and RX are unchanged.
impact indicator aspiratio n	unaffected
limitation	The range of immediate numbers is 0-31.
exceptions	not have

32-bit instruction format:

31	30	26	25	21	20	16	15	10	9	5	4	0
1		10001		RZ		RX		000011		01000		IMM5

Figure 14.67: DECT

14.45 DIVS - Signed Division Instruction

Harmonization Directive	
grammatical	divs rz, rx, ry
manipulate	signed division RZ= RX / RY
Translation in the end	32-bit instructions only divs32 rz, rx, ry
clarification	The signed register divide instruction divides the value of register RX by value of register RY and the resulting quotient is stored RZ.RX, The values of RY and RZ are considered 32-bit signed numbers. Note that for case 0x80 000000 divided by 0xffffffff, the result is undefined.
Impact Signs classifier for honorific people	not affect
exceptions	zero removal anomaly (math.)

32-bit instruction	
manipulate	signed division RZ= RX / RY
grammatical	divs32 rz, rx, ry
clarification	The signed register divide instruction divides the value of register RX by value of register RY and the resulting quotient is stored RZ.RX, The values of RY and RZ are considered 32-bit signed numbers. Note that for case 0x80000000 divided by 0xffffffff, the result is undefined.
Impact Signs	not affect

Users Manual	
ier for honor ific peopl e	
except ions	zero removal anomaly (math.)

32-bit instruction format:

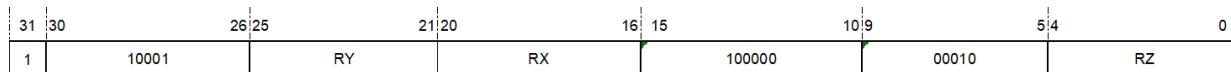


Figure 14.68: DIVS

14.46 DIVU - Unsigned Division Instruction

Harmonization Directive	
grammatical	divu rz, rx, ry
manipulate	unsigned division RZ= RX / RY
clarification	32-bit instructions only divu32 rz, rx, ry
clarification	The unsigned register divide instruction divides the value of register RX by value of register RY and the resulting quotient is stored RZ. rx, The values of RY and RZ are considered 32-bit unsigned numbers.
affect aspiration	not affect
exceptions	zero removal anomaly (math.)

32-bit instruction	
manipulate	unsigned division RZ= RX / RY
grammatical	divu32 rz, rx, ry
clarification	The unsigned register divide instruction divides the value of register RX by value of register RY and the resulting quotient is stored RZ. rx, The values of RY and RZ are considered 32-bit unsigned numbers.
affect aspiration	not affect
exceptions	zero removal anomaly (math.)

32-bit instruction format:



Figure 14.69: DIVU

14.47 DOZE - enter low-power sleep mode command

Harmonization Directive	
grammatical	doze
manipulate	Enter low-power sleep mode
Translation resolute	32-bit instructions only doze32
clarification	This instruction puts the processor into a low-power sleep mode and waits for an interrupt to exit this mode. At this point, the CPU clock stops stops and the corresponding peripheral devices are stopped.
impact indicator aspiration	not affect
exceptions	Privilege violation exceptions

32-bit instruction	
manipulate	Enter low-power sleep mode
grammatical	doze32
Properties:	privileged instruction
clarification	This instruction puts the processor into a low-power sleep mode and waits for an interrupt to exit this mode. At this point, the CPU clock stops stops and the corresponding peripheral devices are stopped.
impact indicator aspiration	not affect
exceptions	Privilege violation exceptions

32-bit instruction format:



Figure 14.70: DOZE

14.48 FF0 - Fast Find 0 command

Harmonization Directive	
gram matica 1	ff0 rz, rx
manip ulate	RZ← find_first_0(RX).
Trans latio n in the end	32-bit instructions only ff0.32 rz, rx
clarifi cation	Finds the first bit of RX that is 0 and returns the result to RZ, scanning from the highest to the lowest bit of RX. If the highest bit of RX (RX[31]) is 0, the value of RZ is returned as 0. If there is no 0 bit in RX, the value of RZ is returned as 0. The value is 32.
Imp act Sign s classif ier for honor ific peopl e	unaffected
except ions	not have

32-bit instruction	
manip ulate	RZ← find_first_0(RX).
gram matica 1	ff0.32 rz, rx
clarifi cation	Finds the first bit of RX that is 0 and returns the result to RZ, scanning from the highest to the lowest bit of RX. If the highest bit of RX (RX[31]) is 0, the value of RZ is returned as 0. If there is no 0 bit in RX, the value of RZ is returned as 0. The value is 32.
Imp act Sign s	unaffected

Users Manual	
except ions	not have

Command Format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1	10001	00000	RX	011111	00001	RZ

Figure 14.71: FF0

14.49 FF1 - Quick Find 1 command

Harmonization Directive	
grammatical	ff1 rz, rx
manipulate	RZ← find_first_1(RX).
Translation in the end	32-bit instructions only ff1.32 rz, rx
clarification	Finds the first bit of RX that is a 1 and returns the result to RZ, scanning from the highest bit of RX to the lowest bit. If the highest bit of RX (RX[31]) is a 1, the value of RZ is returned as 0. If there is no bit of 1 in RX, the value of RZ is returned as 0. The value is 32.
Impact Signs classifier for honorific people	unaffected
exceptions	not have

32-bit instruction	
manipulate	RZ← find_first_1(RX).
grammatical	ff1.32 rz, rx
clarification	Finds the first bit of RX that is a 1 and returns the result to RZ, scanning from the highest bit of RX to the lowest bit. If the highest bit of RX (RX[31]) is a 1, the value of RZ is returned as 0. If there is no bit of 1 in RX, the value of RZ is returned as 0. The value is 32.
Impact Signs	unaffected

Users Manual	
except ions	not have

Command Format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1	10001	00000	RX	011111	00010	RZ

Figure 14.72: FF1

14.50 GRS - symbol generation instructions

Harmonization Directive	
gram matica 1	grs rz, label grs rz, imm32
manip ulate	RZ← PC+ sign_extend(offset<< 1);
Tran slati on resu lts	Only 32-bit instructions exist. grs32 rz, label grs32 rz, imm32
clarifi cation	Produces the value of the symbol, which is determined by the location of the label, or 32-bit immediate number (IMM32). The value of the symbol is determined by the current program PC plus the 18-bit relative offset shifted 1 bit to the left is worth to after the signed extension to 32 bits. The valid range for the value of the sign is ±256KB address space.
Imp act Sign s classif ier for honor ific peopl e	unaffected
except ions	not have

32-bit instruction	
manip ulate	RZ← PC+ sign_extend(offset<< 1);
gram matica 1	grs rz, label grs rz, imm32
clarifi cation	Produces the value of the symbol, which is determined by the location of the label, or 3 2-bit immediate number (IMM32). The value of the symbol is determined by the current program PC plus the 18-bit relative offset shifted 1 bit to the left is worth to after the signed extension to 32 bits. The valid range for the value of the sign is ±256KB address space.

Impacted	affected
act	
Sign	
s	
classifier for	
honorable	
ific	
people	
except	not have
ions	

Command Format:

Figure 14.73: GRS

14.51 IDLY - Interrupt Recognition Lockout Instruction

Harmonization Directive	
tell to Bud dhis t teac hing	idly
drill (prac tic e) writ ings or wor ks	Disable interrupt recognition 4 instructions
c o m pi la ti o n re s ul t reso lute	32-bit instructions only idly32
pers uad e gen eric ter m fora sacr ifice the	The last four instructions of the idly instruction disable interrupt recognition, thus allowing a sequence of non-interruptible instructions to be executed in a multitasking environment.

Users	Manual
M ar k er s of in fl u e n ce clas sifie r for hon orifi c peo ple	Flag bit C is cleared after the execution of the idly instruction. If an exception (including a trace or breakpoint exception) occurs in the instructions following the execution of the idly instruction, bit C is set to one and the interrupt instruction sequence can be observed.
li m it at io n	<p>Instructions following the idly instruction can only be single clock cycle arithmetic, logic, ld, st, or branch instructions. In order to minimize the effect of some potential interrupts, there is no guarantee that an instruction will not be interrupted if it is followed by another instruction. If there is another idly instruction in the sequence of instructions following the idly, it will be ignored. However, if there are rte, rfi, doze, wait, stop, etc. instructions, they will cause the idly sequence to abort.</p> <p>The idly instruction is not allowed in a loop of less than 8 instructions.</p>
disc rimi nate cons tant	not have
:	<p>Interrupts are masked if the idly counter stops in a non-zero state. If a breakpoint exception or a trace exception occurs during an idly instruction sequence, bit C is set to 1 so that the sequence will fail. During exception handling, the interrupt masking is disabled so that the counter is cleared to zero.</p> <p>idly the counter does not change during debugging using the H AD debug port, once the processor is released from debug mode to the positive regular operation, then the counting continues.</p>

32-bit instruction	
drill (practic e) writ ings or wor ks	Disable interrupt recognition 4 instructions disable_int_in_following(4).
tell to Bud dhis t teac hing	idly32
pers uad e gen eric ter m for a sacr ifice the gods	The last four instructions of the idly instruction disable interrupt recognition, thus allowing a sequence of non-interruptible instructions to be executed in a multitasking environment.
M ar k er s of in fl u e n ce clas sifie r for	Flag bit C is cleared after the execution of the idly instruction. If an exception (including a trace or breakpoint exception) occurs in the four instructions following the execution of the idly instruction, bit C is set to one and the interrupt instruction sequence can be observed.

User Manual	
li m it at io n	<p>Instructions following the idly instruction can only be single clock cycle arithmetic, logic, ld, st, or branch instructions. In order to minimize the effect of some potential interrupts, there is no guarantee that an instruction will not be interrupted if it is followed by another instruction. If there is another idly instruction in the sequence of instructions following the idly, it will be ignored. However, if there are rte, rfi, doze, wait, stop, etc. instructions, they will cause the idly sequence to abort.</p> <p>The idly instruction is not allowed in a loop of less than 8 instructions.</p>
disc rimi nate cons tant	not have
:	<p>Interrupts are masked if the idly counter stops in a non-zero state. If a breakpoint exception or a trace exception occurs during an idly instruction sequence, bit C is set to 1 so that the sequence will fail. During exception handling, the interrupt masking is disabled so that the counter is cleared to zero.</p> <p>The counter for idly does not change during debugging using the HAD debug port once the processor is released from debug mode to the positive regular operation, then the counting continues.</p>

32-bit instruction format:

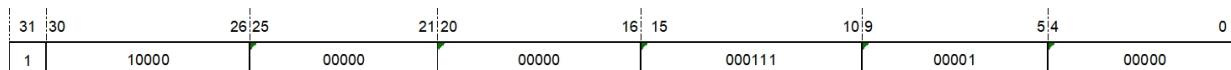


Figure 14.74: IDLY

14.52 INCF--C is 0 Immediate Number Addition Instruction

Harmonization Directive	
grammatical	incf rz, rx, imm5
manipulate	if C==0, then RZ← RX+ zero_extend(IMM5); else RZ← RZ.
Translation resolute	32-bit instructions only incf32 rz, rx, imm5
clarification	If the condition bit C is 0, extend the 5-bit immediate zero to 32 bits, add the value RX to the 32-bit value, and store the result in the RZ; otherwise, the values of RZ and RX are unchanged.
impact indicator aspiratio n	unaffected
limitation	The range of immediate numbers is 0-31.
exceptions	not have

32-bit instruction	
manipulate	if C==0, then RZ← RX+ zero_extend(IMM5); else RZ← RZ.
grammatical	incf32 rz, rx, imm5
clarification	If the condition bit C is 0, extend the 5-bit immediate zero to 32 bits, add the value RX to the 32-bit value, and store the result in the RZ; otherwise, the values of RZ and RX are unchanged.
impact indicator aspiratio n	unaffected
limitation	The range of immediate numbers is 0-31.

User's Manual

exception	not have
ns	

32-bit instruction format:

31	30	26:25	21:20	16:15	10:9	5:4	0
1	10001	RZ	RX	000011	00001	IMM5	

Figure 14.75: INCF

14.53 INCT--C is a 1-immediate number addition instruction.

Harmonization Directive	
grammatical	inct rz, rx, imm5
manipulate	if C==1, then RZ← RX+ zero_extend(IMM5); else RZ← RZ.
Translation resolute	32-bit instructions only inct32 rz, rx, imm5
clarification	If the condition bit C is 1, expand the 5-bit immediate zero to 32 bits, add the value RX to the 32-bit value, and store the result in RZ; otherwise, the values of RZ and RX are unchanged.
impact indicator aspiratio n	unaffected
limitation	The range of immediate numbers is 0-31.
exceptions	not have

32-bit instruction	
manipulate	if C==1, then RZ← RX+ zero_extend(IMM5); else RZ← RZ.
grammatical	inct32 rz, rx, imm5
clarification	If the condition bit C is 1, expand the 5-bit immediate zero to 32 bits, add the value RX to the 32-bit value, and store the result in RZ; otherwise, the values of RZ and RX are unchanged.
impact indicator aspiratio n	unaffected
limitation	The range of immediate numbers is 0-31.

32-bit instruction format:

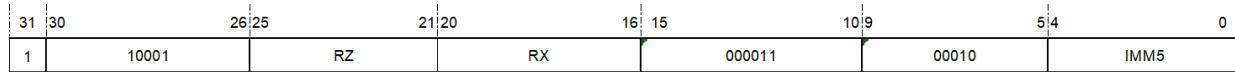


Figure 14.76: INCT

14.54 INS - bit insertion instruction

Harmonization Directive	
tell to Bud dhis t teac hing	ins rz, rx, msb, lsb
drill (prac tic e) writ ings or wor ks	RZ[MSB:LSB]← RX[MSB-LSB:0]
c o m pi la ti o n re s ul t reso lute	32-bit instructions only ins32 rz, rx, msb, lsb
cl ar ifi ca ti o n	A segment of consecutive low bits of RX is inserted into a segment of consecutive bits (RZ[MSB: LSB]) of RZ specified by two 5-bit immediate numbers (MSB,LSB), the other bits of RZ remain unchanged, and the width of the consecutive low bits of RX is specified by the MSB and LSB (i.e., RX[MSB-LSB:0]). If the MSB is equal to 31 and且LSB is equal to 0, the value of RZ is the same as RX. If MSB is equal to LSB, the MSB of RZ is the same as RX. (i.e., LSB) bit is the lowest bit of the RX and the other bits remain unchanged. If the MSB is less than the LSB, the behavior of the instruction is unpredictable.
M ar k er s	unaffected

User's Manual	
set a limit (on) regulate	MSB ranges from 0-31, LSB ranges from 0-31, and MSB should be greater than or equal to LSB.
discriminate constant	not have

32-bit instruction	
drill (practic e) writ ings or wor ks	RZ[MSB:LSB]← RX[MSB-LSB:0]
tell to Bud dhis t teac hing	ins32 rz, rx, msb, lsb
cl ar ifi ca ti o n	A segment of consecutive low bits of RX is inserted into a segment of consecutive bits (RZ[MSB: LSB]) of RZ specified by two 5-bit immediate numbers (MSB,LSB), the other bits of RZ remain unchanged, and the width of the consecutive low bits of RX is specified by the MSB and LSB (i.e., RX[MSB-LSB:0]). If the MSB is equal to 31 and且LSB is equal to 0, the value of RZ is the same as RX. If MSB is equal to LSB, the MSB of RZ is the same as RX. (i.e., LSB) bit is the lowest bit of the RX and the other bits remain unchanged. If the MSB is less than the LSB, the behavior of the instruction is unpredictable.
M ar k er s of in fl u e n ce clas sifie r for hon orifi c peo ple	unaffected

User's Manual	MSB ranges from 0-31, LSB ranges from 0-31, and MSB should be greater than or equal to LSB.
disc rimi nate cons tant	not have

32-bit instruction format:

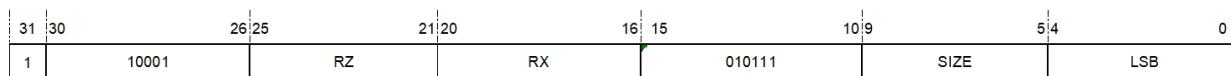


Figure 14.77: INS

SIZE field:

Specifies the width of the insertion bit.

Note: The binary operand SIZE equals MSB-LSB.

00000:

1

00001:

2

.....

11111:

32

LSB Domain:

Specifies the bit at the end of the insertion.

00000:

0 bit

00001:

1 position

.....

11111:

31st

14.55 IPOP - interrupt out of stack instruction

Harmonization Directive	
grammatical	ins rz, rx, msb, lsb
manipulate	RZ[MSB:LSB]← RX[MSB-LSB:0]
Compilation results	32-bit instructions only ins32 rz, rx, msb, lsb

Description:	Load the interrupt's general-purpose register site {R0~R3, R12, R13 } from the stack pointer register, then update the stack pointer register ware to the top of the stack memory. Direct addressing using the stack pointer register.
affect Chi Bit:	unaffected
Exception:	Access Error Exception, Unaligned Exception

16-bit directives	
Operation:	Load the interrupt's general-purpose register site {R0~R3, R12, R13 } from the stack pointer register, then update the stack pointer register to the top of the stack memory; {R0~R3,R12,R13}← MEM[SP]~MEM[SP+20]. SP←SP+24.
Grammar:	ipop16
Description:	Load the interrupt's general-purpose register site {R0~R3, R12, R13 } from the stack pointer register, then update the stack pointer register ware to the top of the stack memory. Direct addressing using the stack pointer register.

Impact	affected
Logo Bit:	
Exception:	Access Error Exception, Unaligned Exception

Command Format:

15	14	10	9	8	7	5	4	0
1		00101	0	0	011		00011	

Figure 14.78: IPOP

14.56 IPUSH - Interrupt Stacking Instruction

unify chemi calizat ion honor ific title	
gram matica 1	ipush
manip ulate	Store the interrupt's general-purpose register site {R0~R3, R12, R13} into the stack memory, then update the stack pointer register to the top of the stack memory; MEM[SP-4]~MEM[SP-24] \leftarrow {R13,R12,R3~R0}. SP \leftrightarrow SP-24.
Transl ation in the end	16-bit instructions only ipush.

Description:	Save the interrupt's general-purpose register site { R0~R3, R12, R13 } to the stack memory, then update the stack pointer registers ware to the top of the stack memory. Direct addressing using the stack pointer register.
affect Chi Bit:	unaffected
Exception:	Access Error Exception, Unaligned Exception

16-bit directi ves	
--------------------------	--

Operati on:	Store the interrupt's general-purpose register site {R0~R3, R12, R13 } into the stack memory, then update the stack pointer register to the top of the stack memory; MEM[SP-4]~MEM[SP-24] \leftarrow {R13,R12,R3~R0}; SP \leftrightarrow SP-24.
Gramm ar:	ipush16
Descrip tion:	Save the interrupt's general-purpose register site { R0~R3, R12, R13 } to the stack memory, then update the stack pointer registers ware to the top of the stack memory. Direct addressing using the stack pointer register.
impac t Logo Bit:	unaffected
Excepti on:	Access Error Exception, Unaligned Exception

Command Format:

15	14	10	9	8	7	5	4	0
0	00101	0	0		011		00010	

Figure 14.79: IPUSH

14.57 IXH - Indexed Half-Word Instruction

Harmonization Directive	
grammatical	ixh rz, rx, ry
manipulate	RZ \leftarrow RX + (RY << 1)
Compilation results	32-bit instructions only ixh32 rz, rx, ry
clarification	Shift the value of RY one bit to the left and add the value of RX, and store the result in RZ.
Affecting Flags	unaffected
exceptions	not have

32-bit instruction	
manipulate	RZ \leftarrow RX + (RY << 1)
grammatical	ixh32 rz, rx, ry
instructions	Shift the value of RY one bit to the left and add the value of RX, and store the result in RZ.
Influence Flag Bit	unaffected
exceptions	not have

32-bit instruction format:

31	30	26 25	21 20	16 15	10 9	5 4	0
1	10001	RY	RX	000010	00001	RZ	

Figure 14.80: IXH

14.58 IXW - index word command

Harmonization Directive	
grammatical	ixw rz, rx, ry
manipulate	$RZ \leftarrow RX + (RY \ll 2)$
Compilation results	32-bit instructions only ixw32 rz, rx, ry
clarification	Shift the value of RY two places to the left and add the value of RX, and store the result in RZ.
Affecting Flags	unaffected
exceptions	not have

32-bit instruction	
manipulate	$RZ \leftarrow RX + (RY \ll 2)$
grammatical	ixw32 rz, rx, ry
clarification	Shift the value of RY two places to the left and add the value of RX, and store the result in RZ.
Affecting Flags	unaffected
exceptions	not have

32-bit instruction format:



Figure 14.81: IXW

14.59 IxD - Indexed Double Word Directive

Harmonization Directive	
grammatical	idx rz, rx, ry
manipulate	$RZ \leftarrow RX + (RY \ll 3)$
Compilation results	32-bit instructions only idx32 rz, rx, ry
clarification	Shift the value of RY three places to the left and add the value of RX, and store the result in RZ.

Influence Flag Bit	unaffected
exceptions	not have

32-bit instruction	
manipulate	RZ \leftarrow RX + (RY << 3)
grammatical	ixd32 rz, rx, ry
clarification	Shift the value of RY three places to the left and add the value of RX, and store the result in RZ.
Affecting Flags	unaffected
exceptions	not have

32-bit instruction format:



Figure 14.82: IXD

14.60 JMP - Register Jump Instructions

Harmonization Directive	
grammatical	jmp rx
manipulate	Jump to the location specified by the register PC \leftarrow RX & 0xffffffff
Compilation results	Compiles to the corresponding 16-bit or 32-bit instructions depending on the register range. if (x<16), then jmp16 rx; else jmp32 rx.
clarification	The program jumps to the location specified by register RX, and the lowest bit of RX is ignored. The jump range of the JMP instruction is all 4GB. Address space.
Markers of influence classifier for honorific people	unaffected
exceptions	not have

16-bit instructions	
manipulate	Jump to the location specified by the register PC← RX & 0xffffffe
grammatical	jmp16 rx
clarification	The program jumps to the location specified by register RX, and the lowest bit of RX is ignored. The jump range of the JMP instruction is all 4GB. Address space.
Markers of influence classifier for honorific people	unaffected
exceptions	not have

16-bit instruction format:

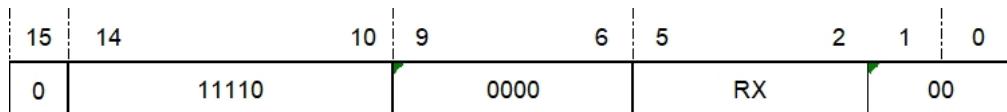


Figure 14.83: JMP-1

32-bit instruction	
manipulate	Jump to the location specified by the register PC← RX & 0xffffffe
grammatical	jmp32 rx
clarification	The program jumps to the location specified by register RX, and the lowest bit of RX is ignored. The jump range of the JMP instruction is all 4GB. Address space.
Markers of influence classifier for honorific people	unaffected
exceptions	not have

User's Manual
32-bit instruction format:

31	30	26	25	21	20	16	15	0
1	11010		00110		RX	0	0	0

Figure 14.84: JMP-2

14.61 JMPI - Indirect Jump Instruction

Harmonization Directive	
gram matica 1	jmpi label
manip ulate	The program jumps to the location specified in the memory $PC \leftarrow MEM[(PC + \text{zero_extend}(offset \ll 2)) \& 0xfffffffffc]$
Trans lation in the end	Only 32-bit instructions exist. jmpi32 label
clarifi cation	The program jumps to the location of the label, which is loaded from memory. The address of the memory is calculated from the PC plus a two-digit left shift of 16 The JMPI instruction's jump range is all of 4GB of address space.
Imp act Sign s classif ier for honor ific peopl e	unaffected
except ions	Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

32-bit instruction	
manip ulate	The program jumps to the location specified in the memory $PC \leftarrow MEM[(PC + \text{zero_extend}(offset \ll 2)) \& 0xfffffffffc]$
gram matica 1	jmpi32 label
instru ctions	The program jumps to the location of the label, which is loaded from memory. The address of the memory is calculated from the PC plus a two-digit left shift of 16 The JMPI instruction's jump range is all of 4GB of address space.
Imp act Sign	unaffected

User's Manual	
classification for honorific people	
exceptions	Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

32-bit instruction format:

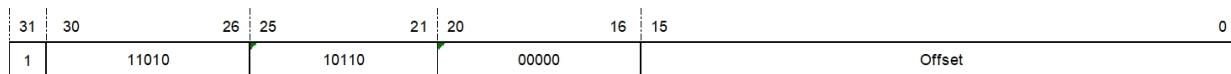


Figure 14.85: JMPI

14.62 JSR - Register Jump to Subroutine Instruction

Harmonization Directive	
gram matica 1	jsr rx
manip ulate	Link and jump to the subroutine location specified in the registers R15← PC+ 4. PC← RX & 0xffffffff
Co mпи лати он реzu лts	Compiles to the corresponding 16-bit or 32-bit instructions depending on the register range. if (x<16), then jsr16 rx; else jsr32 rx.
clarifi cation	Subroutine register jump, the return address of the subroutine (PC of the next instruction, i.e., current PC+4) is stored link register R15, and the program jumps to the subroutine location specified by the contents of register RX for execution, and the lowest bit of RX is ignored. jsr means The jump range of the order is the entire 4GB address space.
Mar ker s of infl uen ce classi fier for hono rific peopl e	unaffected
except ions	not have

16-bit instructions	
manip ulate	Link and jump to the subroutine location specified in the registers R15← PC+ 4, PC← RX & 0xffffffff
gram matica 1	jsr16 rx

OpCode	Subroutine register jump, the return address of the subroutine (PC of the next instruction, i.e., current PC+4) is stored link register R15, and the program jumps to the subroutine location specified by the contents of register RX for execution, and the lowest bit of RX is ignored. jsr means
Marker	unaffected
except	not have

16-bit instruction format:

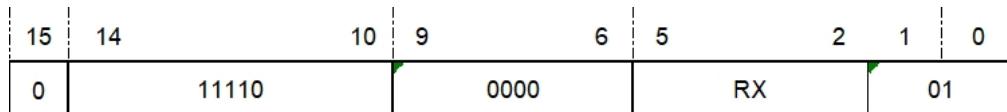


Figure 14.86: JSR-1

32-bit instruction	
manipulate	Link and jump to the subroutine location specified in the registers R15← PC+ 4, PC← RX & 0xffffffff
grammatical	jsr32 rx
clarification	Subroutine register jump, the return address of the subroutine (PC of the next instruction, i.e., current PC+4) is stored link register R15, and the program jumps to the subroutine location specified by the contents of register RX for execution, and the lowest bit of RX is ignored. jsr means The jump range of the order is the entire 4GB address space.
Markers of influence classifier for honoriific people	unaffected
exceptions	not have

32-bit instruction format:

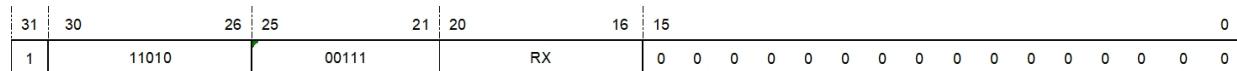


Figure 14.87: JSR-2

14.63 JSRI - Indirect Jump to Subroutine Instruction

Harmonization Directive	
tell to Bud dhis t teac hing	jsri label
m a ni p ul at e	Program jumps to the subroutine location specified in memory R15← next PC. PC← MEM[(PC+ zero_extend(offset<< 2)) & 0xffffffffc]
c o m pi la ti o n re s ul t resol ute	Only 32-bit instructions exist. jsri32 label.
cl ar ifi ca ti o n	Subroutine indirect jump, the return address of the subroutine (PC of the next instruction) is stored in link register R15, and the program jumps to the location where the label is executed, and the label is loaded from the memory. The memory address is obtained by unsigned expansion of the 16-bit relative offset to 32 bits according to the PC plus two left shifts, and then forced clearing by the lowest two bits. 4GB of address space.
M ar k er s of in fl u	unaffected

User's Manual	
discriminate constant	Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

32-bit instruction	
drill (pra ctice) writi ngs or wor ks	Program jumps to the subroutine location specified in memory $R15 \leftarrow PC + 4, PC \leftarrow MEM[(PC + \text{zero_extend}(\text{offset} \ll 2)) \ \& \ 0xffffffff]$
tell to Bud dhis t teac hing	jsri32 label
cl ar ifi ca ti o n	Subroutine indirect jump, the return address of the subroutine (PC of the next instruction, i.e., the current PC+4) is stored in the link register R15, and the program jumps to the location where the label is executed, and the label is loaded from the memory. The program jumps to the location where the label is executed, and the label is loaded from the memory. The memory address is obtained by unsigned expansion of the PC plus the 16-bit relative offset shifted to the left by two bits to 32 bits, and then forced clearing by the lowest two bits. The perimeter is the entire 4GB address space.
M ar k er s of in fl u e n ce clas sifie r for hon orifi c peo ple	unaffected

disc	Macau	Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid
Users	Macau	Exception
rimi		
nate		
cons		
tant		

32-bit instruction format:

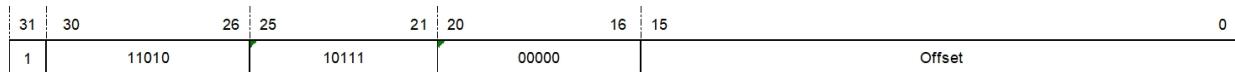


Figure 14.88: JSRI

14.64 LD.B - Unsigned Extended Byte Load Instruction

Harmonization Directive	
tell to Bud dhist teac hing	ld.b rz,(rx, disp)
drill (pra ctice) writi ngs or work s	RZ \leftarrow zero_extend(MEM[RX + zero_extend(offset)])
C o m pi la ti o n re s ul ts	Compiles to the corresponding 16-bit or 32-bit instructions depending on the range of offsets and registers. if (disp<32) and(x<7) and (z<7), then ld16.b rz, (rx, disp); else ld32.b rz, (rx, disp).
in st r u ct io n s	The bytes loaded from memory are zero-extended to 32 bits and stored in register RZ. The register plus immediate offset addressing method is used. The effective address of the memory is determined by the base address register RX plus a 12-bit relative offset unsigned expanded to 32 bits. The LD.B instruction addresses +4KB of address space. Note that the offset DISP is the binary operand Offset.
M ar k er s of in fl	unaffected

User's Manual	
discriminate constant	Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

16-bit instructions	
drill (pra ctice) writi ngs or wor ks	Load bytes from memory to registers, unsigned extension $RZ \leftarrow \text{zero_extend}(\text{MEM}[RX + \text{zero_extend}(\text{offset})])$
tell to Bud dhist teac hing	ld16.b rz, (rx, disp)
cl ar ifi ca ti o n	The bytes loaded from memory are zero-extended to 32 bits and stored in register RZ. The register plus immediate offset addressing method is used. The effective address of the memory is determined by the base address register RX plus a 5-bit relative offset unsigned expanded to 32 bits. The LD16.B instruction addresses the +32B address space. Note that the offset DISP is the binary operand Offset.
M ar k er s of in fl u e n ce class ifier for hono rific peop le	unaffected
set a limit (on)	The range of the register is r0-r7.

User Manual	late
discriminate constant	Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

16-bit instruction format:

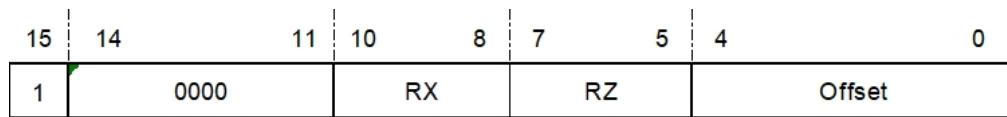


Figure 14.89: LD.

32-bit instruction	
drill (pra ctice) writi ngs or wor ks	Load bytes from memory to registers, unsigned extension $RZ \leftarrow \text{zero_extend}(\text{MEM}[RX + \text{zero_extend}(\text{offset})])$
tell to Bud dhist teac hing	ld32.b rz, (rx, disp)
cl ar ifi ca ti o n	The bytes loaded from memory are zero-extended to 32 bits and stored in register RZ. Addressing is by register plus immediate offset. The effective address of the memory is determined by adding the base address register RX plus a 12-bit relative offset unsigned expanded to 32 bits. The LD32.B instruction addresses +4KB of address space. Note that the offset DISP is the binary operand Offset.
M ar k er s of in fl u e n ce class ifier for hono rific peop le	unaffected
discr imin ate	Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

32-bit instruction format:

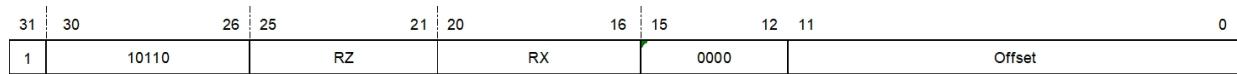


Figure 14.90: LD.B-2

14.65 LD.BS - Signed Extended Byte Load Instruction

Harmonization Directive	
tell to Bud dhist teac hing	ld.bs rz, (rx, disp)
drill (pra ctice) writi ngs or wor ks	RZ \leftarrow sign_extend(MEM[RX + zero_extend(offset)])
c o m pi la ti o n re s ul t resol ute	Only 32-bit instructions exist. ld32.bs rz, (rx, disp)
cl ar ifi ca ti o n	The bytes loaded from memory are stored in register RZ after signed expansion to 32 bits. The register plus immediate offset addressing method is used. The effective address of the memory is determined by adding the base address register RX plus a 12-bit relative offset unsigned expanded to 32 bits. The LD.BS instruction addresses +4KB of address space. Note that the offset DISP is the binary operand Offset.
M ar k er s of in	unaffected

fl User's Manual u e n ce class ifier for hono rific peop le	
discr imin ate cons tant	Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

32-bit instruction	
drill (pra ctice) writi ngs or wor ks	Load bytes from memory to registers, signed extensions $RZ \leftarrow \text{sign_extend}(\text{MEM}[RX + \text{zero_extend}(\text{offset})])$
tell to Bud dhist teac hing	ld32.bs rz, (rx, disp)
cl ar ifi ca ti o n	The bytes loaded from memory are stored in register RZ after signed expansion to 32 bits. Addressing is by register plus immediate offset. The effective address of the memory is determined by adding the base address register RX plus a 12-bit relative offset unsigned expanded to 32 bits. the LD32.BS instruction can address +4KB of address space. Note that the offset DISP is the binary operand Offset.
M ar k er s	unaffected

User's Manual	
discriminate constant	Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

32-bit instruction format:

31	30	26	25	21	20	16	15	12	11	0
1	10110		RZ		RX		0100		Offset	

Figure 14.91: LD.

14.66 LD.H - Unsigned Extended Half-Word Load Instruction

Harmonization Directive	
tell to Bud dhis t teac hing	ld.h rz, (rx, disp)
drill (pra ctice) writi ngs or wor ks	<pre>RZ← zero_extend(MEM[RX+ zero_extend(offset<< 1)])</pre>
Compi la ti o n re s ul ts	<p>Compiles to the corresponding 16-bit or 32-bit instructions depending on the range of offsets and registers.</p> <pre>if (disp<64) and(x<7) and (z<7), then ld16.h rz, (rx, disp); else ld32.h rz, (rx, disp).</pre>
clarifi ca ti	The halfword loaded from memory is expanded to 32 bits by zero and stored in register RZ. The register plus immediate offset addressing method is used. The effective address of the memory is determined by adding the base address register RX to the base address register plus a 12-bit relative offset unsigned expanded by 1 bit to 32 bits. the LD.H instruction addresses +8KB of address space.

User's Manual	Note: The offset DISP is obtained by shifting binary operand Offset by 1 bit.
M ar k er s of in fl u e n ce class ifier for hon orifi c peo ple	unaffected
disc rimi nate cons tant	Unaligned Access Exception, Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

16-bit instructions	
drill (pra ctice) writi ngs or wor ks	Load half-words from memory to registers, unsigned extensions $RZ \leftarrow \text{zero_extend}(\text{MEM}[RX + \text{zero_extend}(\text{offset} \ll 1)])$
tell to Bud dhis t teac hing	ld16.h rz, (rx, disp)
in st r u ct io n s	The halfword loaded from memory is expanded to 32 bits by zero and stored in register RZ. The register plus immediate offset addressing method is used. The effective address of the memory is determined by adding the base address register RX plus a 5-bit relative offset shifted 1 bit to the left, unsigned, to 32 bits, and the value is LD16.H. The LD16.H instruction can address +64B of address space. Note that the offset DISP is obtained by shifting binary operand Offset by 1 bit.
M ar k er s of in fl u e n ce class ifier for hon orifi c peo ple	unaffected

Set a limit (on) regulate	The range of the register is r0-r7.
discriminate constant	Unaligned Access Exception, Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

16-bit instruction format:

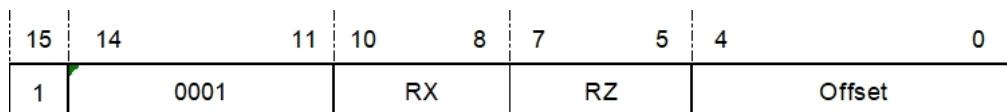


Figure 14.92: LD.H-1

32-bit instruction	
drill (pra ctice) writi ngs or wor ks	Load half-words from memory to registers, unsigned extensions $RZ \leftarrow \text{zero_extend}(\text{MEM}[RX + \text{zero_extend}(\text{offset} \ll 1)])$
tell to Bud dhis t teac hing	ld32.h rz, (rx, disp)
cl ar ifi ca ti o n	<p>The halfword loaded from memory is expanded to 32 bits by zero and stored in register RZ. The register plus immediate offset addressing method is used. The effective address of the memory is determined by adding the base address register RX plus a 12-bit relative offset shifted one bit to the left, unsigned, to 32 bits, and the value is LD32.H. The LD32.H instruction addresses +8KB of address space.</p> <p>Note that the offset DISP is obtained by shifting binary operand Offset by 1 bit.</p>
M ar k er s of in fl u e n ce class ifier for hon orifi c peo ple	unaffected

disc	unaligned Access Exception, Access Error Exception, TLB Unrecoverable Exception, TLB
users	Mismatch Exception, TLB Read Invalid Exception
rimi	
nate	
cons	
tant	

32-bit instruction format:

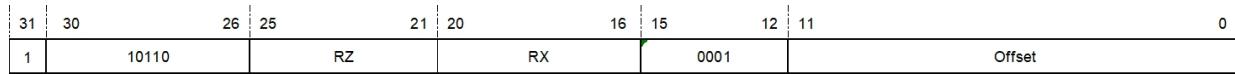


Figure 14.93: LD.H-2

14.67 LD.HS - Signed Extended Half-Word Load Instruction

Harmonization Directive	
tell to Bud dhis t teac hing	ld.hs rz, (rx, disp)
drill (pra ctice) writi ngs or wor ks	RZ← sign_extend(MEM[RX+ zero_extend(offset<< 1)])
c o m pi la ti o n re s ul t reso lute	Only 32-bit instructions exist. ld32.hs rz, (rx, disp)
cl ar ifi ca ti o n	The half-word loaded from memory is signed and expanded to 32 bits and stored in register RZ. The register plus immediate offset addressing method is used. The effective address of the memory is determined by adding the base address register RX to the base address register plus a 12-bit relative offset unsigned expanded to 32 bits by 1 bit to the left. the LD.HS instruction addresses +8KB of address space. Note that the offset DISP is obtained by shifting binary operand Offset by 1 bit.
M ar k er s of	unaffected

in User's Manual fl u e n ce class ifier for hon orifi c peo ple	
disc rimi nate cons tant	Unaligned Access Exception, Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

32-bit instruction	
drill (pra ctice) writi ngs or wor ks	Load half-words from memory to registers with signed extensions $RZ \leftarrow \text{sign_extend}(\text{MEM}[RX + \text{zero_extend}(\text{offset} \ll 1)])$
tell to Bud dhis t teac hing	ld32.hs rz, (rx, disp)
cl ar ifi ca ti o n	The half-word loaded from memory is signed and expanded to 32 bits and stored in register RZ. The register plus immediate offset addressing method is used. The effective address of the memory is determined by adding the base address register RX plus a 12-bit relative offset shifted one bit to the left, unsigned, to 32-bit. Note that the offset DISP is obtained by shifting binary operand Offset by 1 bit.

User's Manual	Manufactured
ar k er s of in fl u e n ce clas sifie r for hon orifi c peo ple	
disc rimi nate cons tant	Unaligned Access Exception, Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

32-bit instruction format:

31	30	26	25	21	20	16	15	12	11	0
1	10110		RZ		RX		0101		Offset	

Figure 14.94: LD.HS

14.68 LD.W - Word Load Instruction

Harmonization Directive	
tell to Bud dhist teac hing	ld.w rz, (rx, disp)
drill (pra ctice) writi ngs or wor ks	RZ← MEM[RX+ zero_extend(offset<< 2)]
C o m pi la ti o n re s ul ts	Compiles to the corresponding 16-bit or 32-bit instructions depending on the range of offsets and registers. if (x=sp) and (z<7) and (disp < 1024), ld16.w rz, (sp, disp). else if (disp<128) and (x<7) and (z<7), ld16.w rz, (rx, disp); else ld32.w rz, (rx, disp).
cl ar ifi ca ti o n	Load word from memory into register RZ. Addressing is by register plus immediate offset. The effective address of the memory is determined by adding the base address register RX plus a 12-bit relative offset two bits to the left, unsigned, to a 32-bit value. The LD.W instruction can address +16KB of address space. Note that the offset DISP is obtained by shifting the binary operand Offset two places to the left.

User's Manual	Manufactured by Xiaomi Technology Co., Ltd.
discriminate constant	Unaligned Access Exception, Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

16-bit instructions	
drill (practic e) writ ings or wor ks	Load word from memory to register $RZ \leftarrow \text{MEM}[RX + \text{sign_extend}(\text{offset} \ll 2)]$
tell to Bud dhis t teac hing	ld16.w rz, (rx, disp) ld16.w rz, (sp, disp)
in st r u ct io n s	Load word from memory into register RZ. The register plus immediate offset addressing method is used. RX is SP, the effective address of the memory is determined by the base address register RX plus an 8-bit relative offset shifted 2 bits to the left and extended unsigned to 32 bits. When rx is other registers, the valid address of memory is the base address register RX plus a 5-bit relative offset shifted 2 bits to the left unsigned to the 32-bit value. the LD16.W instruction can address +1KB address space. Note that the offset DISP is obtained by shifting the binary operand IMM5 two bits to the left. base address register RX is SP, offset DISP is obtained by shifting binary operands {IMM3, IMM5} by two bits.
M ar k er s of in fl u e n ce clas sifie r for hon orifi c peo	unaffected

User's Manual	
set a limit (on) register late	The range of the register is r0-r7.
discriminate constant	Unaligned Access Exception, Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception.

16-bit instruction format:

ld16.w rz, (rx, disp)

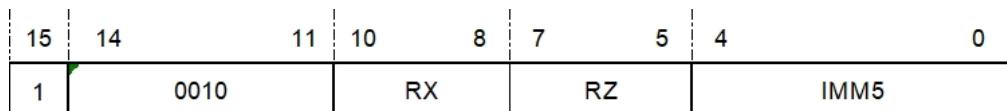


Figure 14.95: LD.W-1

ld16.w rz, (sp, disp)

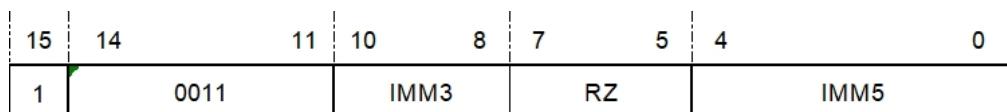


Figure 14.96: LD.W-2

32-bit instruction	
drill (pra ctice) writi ngs or wor ks	Load word from memory to register $RZ \leftarrow \text{MEM}[RX + \text{zero_extend}(\text{offset} \ll 2)]$
tell to Bud dhist teac hing	ld32.w rz, (rx, disp)
cl ar ifi ca ti o n	<p>Load word from memory into register RZ. Addressing is by register plus immediate offset. The effective address of the memory is determined by adding the base address register RX plus a 12-bit relative offset two bits to the left, unsigned, to 32 bits. the LD32.W instruction addresses +16KB of address space.</p> <p>Note that the offset DISP is obtained by shifting the binary operand Offset two places to the left.</p>
M ar k er s of in fl u e n ce class ifier for hono rific peop le	unaffected
discr imin ate	Unaligned Access Exception, Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

32-bit instruction format:

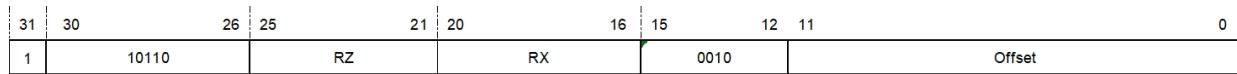


Figure 14.97: LD.W-3

14.69 LDM - Continuous Multi-Word Load Directive

Harmonization Directive	
tell to Budd hist teach ing	ldm ry-rz, (rx)
m a ni p ul at e	<p>Load consecutive multiple words from memory into one contiguous piece of the register stack</p> <p>$dst \leftarrow Y$; $addr \leftarrow RX$.</p> <pre>for (n= 0; n<= (Z-Y); n++){ Rdst← MEM[addr]; dst ← dst + 1; addr← addr+ 4; }</pre>
c o m pi la ti o n re s ul t resol ute	<p>Only 32-bit instructions exist.</p> <p>ldm32 ry-rz, (rx).</p>
cl ar ifi ca ti o n	<p>Sequentially load consecutive words from memory into a stack of consecutive registers starting at register RY, i.e., the first word starting at the specified address in memory is loaded into register RY, the second word loaded into register RY+1, and so on, and the last word is loaded into register RY+1.</p> <p>Load into register RZ. The effective address of the memory is determined by the contents of the base address register RX.</p>
M ar k er s of in fl u	unaffected

User's Manual	
set a limit (on) regulate	RZ should be greater than or equal to RY. The RY-RZ range should not contain the base address register RX, otherwise the results are unpredictable.
discriminate constant	Unaligned Access Exception, Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

32-bit instruction	
m a ni p ul at e	Load consecutive multiple words from memory into one contiguous piece of the register stack dst \leftarrow Y; addr \leftarrow RX. for (n= 0; n \leq IMM5; n++) { Rdst \leftarrow MEM[addr]; dst \leftarrow dst + 1; addr \leftarrow addr+ 4; }
tell to Budd hist teach ing	ldm32 ry-rz, (rx)
cl ar ifi ca ti o n	Sequentially load consecutive words from memory into a stack of consecutive registers starting at register RY, i.e., the first word starting at the specified address in memory is loaded into register RY, the second word loaded into register RY+1, and so on, and the last word is loaded into register RY+1. Load into register RZ. The effective address of the memory is determined by the contents of the base address register RX.
M ar k er s of in fl u e n ce classi fier for hono rific peop le	unaffected
set a limit (on) regul ate	RZ should be greater than or equal to RY. The RY-RZ range should not contain the base address register RX, otherwise the results are unpredictable.

discrepancy	Aligned Access Exception, Access Error Exception, TLB Unrecoverable Exception, TLB
immediate	Mismatch Exception, TLB Read Invalid Exception
constant	

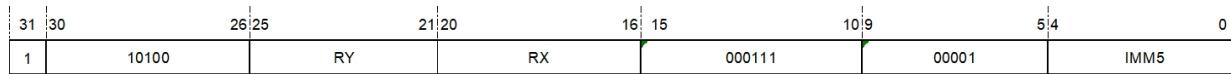
32-bit instruction format:

Figure 14.98: LDM

IMM5 domain:

Specifies the number of target registers, $\text{IMM5} = Z - Y$.

0000:

1 destination register

00001:

2 destination registers

.....

11111:

32 destination registers

14.70 LDQ - Consecutive four-word load command

Harmonization Directive	
tell to Bud dhis t teac hing	ldq r4-r7, (rx)
m a ni p ul at e	<p>Load four consecutive words from memory to registers R4-R7</p> <p>dst\leftarrow 4; addr\leftarrow RX.</p> <pre>for (n = 0; n <= 3; n++) { Rdst ← MEM[addr]. dst ← dst + 1; addr ← addr + 4. }</pre>
c o m pi la ti o n re s ul t reso lute	<p>Only 32-bit instructions exist.</p> <p>ldq32 r4-r7, (rx).</p>
cl ar ifi ca ti o n	<p>consecutive words are loaded sequentially from memory into the register stack [R4, R7] (including boundaries), i.e., the first word starting at the address specified in memory is loaded into register R4, the second word into register R5, the third word into register R6, and the fourth word into register R7. The effective address of the memory is determined by the contents of base address register RX.</p> <p>Note that this instruction is a pseudo-instruction for ldm r4-r7, (rx).</p>
M ar k er s of in fl	unaffected

User's Manual	
set a limit (on) regulate	The R4-R7 range should not contain the base address register RX, otherwise the results are unpredictable.
discriminate constant	Unaligned Access Exception, Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

32-bit instruction	
m a ni p ul at e	Load four consecutive words from memory to registers R4-R7 dst ← 4; addr ← RX. for (n = 0; n <= 3; n++) { Rdst ← MEM[addr]. dst ← dst + 1; addr ← addr + 4. }
tell to Bud dhis t teac hing	ldq32 r4-r7, (rx)
cl ar ifi ca ti o n	consecutive words are loaded sequentially from memory into the register stack [R4, R7] (including boundaries), i.e., the first word starting at the address specified in memory is loaded into register R4, the second word into register R5, the third word into register R6, and the fourth word into register R7. The effective address of the memory is determined by the contents of base address register RX. Note: This instruction is a pseudo-instruction for ldm32 r4-r7, (rx).
M ar k er s of in fl u e n ce clas sifie r for hon orifi c peo ple	unaffected
set a limit (on)	The R4-R7 range should not contain the base address register RX, otherwise the results are unpredictable.

User Manual	late
discriminate constant	Unaligned Access Exception, Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

32-bit instruction format:

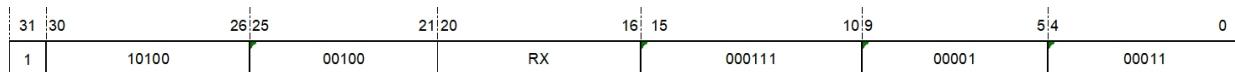


Figure 14.99: LDQ

14.71 LDR.B - Register Shift Addressing Unsigned Extended Byte Load Instruction

Harmonization Directive	
gram matica 1	ldr.b rz, (rx, ry<< 0) ldr.b rz, (rx, ry<< 1) ldr.b rz, (rx, ry<< 2) ldr.b rz, (rx, ry<< 3)
manip ulate	Load bytes from memory to registers, unsigned extension $RZ \leftarrow \text{zero_extend}(\text{MEM}[RX + RY << IMM2])$
Co mpi lati on resu lts	Only 32-bit instructions exist. ldr32.b rz, (rx, ry<< 0) ldr32.b rz, (rx, ry<< 1) ldr32.b rz, (rx, ry<< 2) ldr32.b rz, (rx, ry<< 3)
clarifi cation	The bytes loaded from memory are zero-extended to 32 bits and stored in register RZ. A register plus register shift addressing method is used. The effective address of the memory consists of the base address register RX plus an immediate 2-bit left shift IMM2 of the offset register RY. The default value of 0 for IMM2.
Mar ker s of infl uen ce classif ier for honor ific peopl e	unaffected
except ions	Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

32-bit instruction	
manip ulate	Load bytes from memory to registers, unsigned extension $RZ \leftarrow \text{zero_extend}(\text{MEM}[RX + RY << IMM2])$
gram matica 1	ldr32.b rz, (rx, ry<< 0) ldr32.b rz, (rx, ry<< 1) ldr32.b rz, (rx, ry<< 2) ldr32.b rz, (rx, ry<< 3)

Classification	The bytes loaded from memory are zero-extended to 32 bits and stored in register RZ. A register plus register shift addressing method is used. The effective address of the memory consists of the base address register RX plus an immediate 2-bit left shift IMM2 of the offset register RY. The default value of 0 for IMM2.
Markers of influence classifier for honorific people	unaffected
exceptions	Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

32-bit instruction format:

ldr32.b rz, (rx, ry<< 0)

31 30	26 25	21 20	16 15	10 9	5 4	0
1	10100	RY	RX	000000	00001	RZ

Figure 14.100: LDR.

ldr32.b rz, (rx, ry<< 1)

31 30	26 25	21 20	16 15	10 9	5 4	0
1	10100	RY	RX	000000	00010	RZ

Figure 14.101: LDR.

ldr32.b rz, (rx, ry<< 2)

31 30	26 25	21 20	16 15	10 9	5 4	0
1	10100	RY	RX	000000	00100	RZ

Figure 14.102: LDR.

ldr32.b rz, (rx, ry<< 3)

14.72 LDR.BS - Register Shift Addressing Signed Extended Byte Load Instruction

Harmonization Directive	
gram matica 1	ldr.bs rz, (rx, ry<< 0) ldr.bs rz, (rx, ry<< 1) ldr.bs rz, (rx, ry<< 2) ldr.bs rz, (rx, ry<< 3)
manip ulate	Load bytes from memory to registers, signed extensions $RZ \leftarrow \text{sign_extend}(\text{MEM}[RX + RY \ll IMM2])$
Co mpi lati on resu lts	Only 32-bit instructions exist. ldr32.bs rz, (rx, ry << 0) ldr32.bs rz, (rx, ry << 1) ldr32.bs rz, (rx, ry<< 2) ldr32.bs rz, (rx, ry<< 3)
instr uctio ns	The bytes loaded from memory are stored in register RZ after signed expansion to 32 bits. A register plus register shift addressing method is used. The effective address of the memory is calculated by adding the base address register RX to the offset register RY and shifting it by 2 bits to the left by the immediate number IMM2. The default value of IMM2 is zero.
Mar ker s of infl	unaffected

User's Manual	
except ions	Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

User's Manual

31 30	26 25	21 20	16 15	10 9	5 4	0
1	10100	RY	RX	000000	01000	RZ

Figure 14.103: LDR.

32-bit instruction	
manipulate	Load bytes from memory to registers, signed extensions $RZ \leftarrow \text{sign_extend}(\text{MEM}[RX + RY \ll \text{IMM2}])$
grammatical	ldr32.bs rz, (rx, ry << 0) ldr32.bs rz, (rx, ry << 1) ldr32.bs rz, (rx, ry << 2) ldr32.bs rz, (rx, ry << 3)
clarification	The bytes loaded from memory are stored in register RZ after signed expansion to 32 bits. The register plus register shift addressing method is used. The effective address of the memory is calculated by adding the base address register RX to the offset register RY and shifting it by 2 bits to the left by the immediate number IMM2. The default value of IMM2 is zero.
Markers of influence classifier for honorific people	unaffected
exceptions	Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

32-bit instruction format:

ldr32.bs rz, (rx, ry << 0)

31 30	26 25	21 20	16 15	10 9	5 4	0
1	10100	RY	RX	000100	00001	RZ

Figure 14.104: LDR.BS-1

ldr32.bs rz, (rx, ry << 1)

31 30	26 25	21 20	16 15	10 9	5 4	0
1	10100	RY	RX	000100	00010	RZ

Figure 14.105: LDR.BS-2

31 30	26 25	21 20	16 15	10 9	5 4	0
1 10100	RY	RX		000100	00100	RZ

Figure 14.106: LDR.BS-3

31 30	26 25	21 20	16 15	10 9	5 4	0
1 10100	RY	RX		000100	01000	RZ

Figure 14.107: LDR.BS-4

14.73 LDR.H - Register Shift Addressing Unsigned Extended Half-Word Load Instruction

Harmonization Directive	
grammatical	ldr.h rz, (rx, ry<< 0) ldr.h rz, (rx, ry<< 1) ldr.h rz, (rx, ry<< 2) ldr.h rz, (rx, ry<< 3)
manipulate	Load half-words from memory to registers with unsigned extensions. $RZ \leftarrow \text{zero_extend}(\text{MEM}[RX + RY << \text{IMM2}])$
Compatibility	Only 32-bit instructions exist. ldr32.h rz, (rx, ry<< 0) ldr32.h rz, (rx, ry<< 1) ldr32.h rz, (rx, ry<< 2) ldr32.h rz, (rx, ry<< 3)
clarification	The half-word loaded from memory is expanded to 32 bits by zero and stored in register RZ. A register plus register shift addressing method is used. The effective address of the memory consists of the base address register RX plus an immediate 2-bit left shift IMM2 of the offset register RY. The default value of 0 for IMM2.
Markers of influence classifier for honorific people	unaffected
exceptions	Unaligned Access Exception, Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

32-bit instruction	
manip ulate	Load half-words from memory to registers, unsigned extensions $RZ \leftarrow \text{zero_extend}(\text{MEM}[RX + RY \ll IMM2])$
gram matica 1	ldr32.h rz, (rx, ry<< 0) ldr32.h rz, (rx, ry<< 1) ldr32.h rz, (rx, ry<< 2) ldr32.h rz, (rx, ry<< 3)
clarifi cation	The half-word loaded from memory is expanded to 32 bits by zero and stored in register RZ. A register plus register shift addressing method is used. The effective address of the memory consists of the base address register RX plus an immediate 2-bit left shift IMM2 of the offset register RY. The default value of 0 for IMM2.
Mar ker s of infl uen ce classif ier for honor ific peopl e	unaffected
except ions	Unaligned Access Exception, Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

32-bit instruction format:

ldr32.h rz,(rx, ry << 0)

31	30	26 25	21 20	16 15	10 9	5 4	0
1	10100	RY	RX	000001	00001	RZ	

Figure 14.108: LDR.H-1

ldr32.h rz,(rx, ry << 1)

31	30	26 25	21 20	16 15	10 9	5 4	0
1	10100	RY	RX	000001	00010	RZ	

Figure 14.109: LDR.H-2

ldr32.h rz,(rx, ry << 2)

31	30	26 25	21 20	16 15	10 9	5 4	0
1	10100	RY	RX	000001	00100	RZ	

Figure 14.110: LDR.H-3

ldr32.h rz,(rx, ry << 3)

31 30	26 25	21 20	16 15	10 9	5 4	0
1	10100	RY	RX	000001	01000	RZ

Figure 14.111: LDR.H-4

14.74 LDR.HS - Register Shift Addressing Signed Extended Half-Word Load Instruction

Harmonization Directive	
gram matica 1	ldr.hs rz, (rx, ry<< 0) ldr.hs rz, (rx, ry<< 1) ldr.hs rz, (rx, ry<< 2) ldr.hs rz, (rx, ry<< 3)
manip ulate	Load half-words from memory to registers with signed extensions $RZ \leftarrow \text{sign_extend}(\text{MEM}[RX + RY << IMM2])$
Co mpi lati on resu lts	Only 32-bit instructions exist. ldr32.hs rz, (rx, ry << 0) ldr32.hs rz, (rx, ry << 1) ldr32.hs rz, (rx, ry<< 2) ldr32.hs rz, (rx, ry<< 3)
clarifi cation	The half-word loaded from memory is signed and expanded to 32 bits and stored in register RZ. The register plus register shift addressing method is used. The effective address of the memory is calculated by adding the base address register RX to the offset register RY and shifting it by 2 bits to the left by the immediate number IMM2. The default value of IMM2 is zero.
Mar ker s of infl uen ce classi fier for honor ific peopl e	unaffected
except ions	Unaligned Access Exception, Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

32-bit instruction	
manipulate	Load half-words from memory to registers with signed extensions $RZ \leftarrow \text{sign_extend}(\text{MEM}[RX + RY \ll IMM2])$
grammatical	ldr32.hs rz, (rx, ry << 0) ldr32.hs rz, (rx, ry << 1) ldr32.hs rz, (rx, ry << 2) ldr32.hs rz, (rx, ry << 3)
clarification	The half-word loaded from memory is signed and expanded to 32 bits and stored in register RZ. The register plus register shift addressing method is used. The effective address of the memory is calculated by adding the base address register RX to the offset register RY and shifting it by 2 bits to the left by the immediate number IMM2. The default value of IMM2 is zero.
Markers of influence classifier for honorific people	unaffected
exceptions	Unaligned Access Exception, Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

32-bit instruction format:

ldr32.hs rz, (rx, ry << 0)

31	30	26 25	21 20	16 15	10 9	5 4	0
1	10100	RY	RX	000101	00001	RZ	

Figure 14.112: LDR.HS-1

ldr32.hs rz, (rx, ry << 1)

31	30	26 25	21 20	16 15	10 9	5 4	0
1	10100	RY	RX	000101	00010	RZ	

Figure 14.113: LDR.HS-2

ldr32.hs rz, (rx, ry << 2)

31	30	26 25	21 20	16 15	10 9	5 4	0
1	10100	RY	RX	000101	00100	RZ	

Figure 14.114: LDR.HS-3

ldr32.hs rz, (rx, ry<<3)

31 30	26 25	21 20	16 15	10 9	5 4	0
1	10100	RY	RX	000101	01000	RZ

Figure 14.115: LDR.HS-4

14.75 LDR.W - Register Shift Addressing Word Load Instruction

Harmonization Directive	
gram matica 1	ldr.w rz, (rx, ry<< 0) ldr.w , (rx, ry<< 1) ldr.w , (rx, ry<<2) ldr.w rz, (rx, ry<< 3)
manip ulate	Load word from memory to register RZ← MEM[RX+ RY<< IMM2]
Translati on on resul ts	Only 32-bit instructions exist. ldr32.w rz, (rx, ry<< 0) ldr32.w , (rx, ry<< 1) ldr32.w rz, (rx, ry<< 2) ldr32.w rz, (rx, ry<< 3)
clarifi cation	Load word from memory into register RZ. A register plus register shift addressing method is used. The effective address of the memory is determined by the base The address register RX plus the offset register RY is worth 2 bits immediately IMM2 is shifted to the left. the default value of IMM2 is 0.
impac t Logo classifi er for honori fic people	unaffected
except ions	Unaligned Access Exception, Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

32-bit instruction	
manip ulate	Load word from memory to register RZ← MEM[RX+ RY<< IMM2]
gram matica 1	ldr32.w rz, (rx, ry<< 0) ldr32.w , (rx, ry<< 1) ldr32.w rz, (rx, ry<< 2) ldr32.w rz, (rx, ry<< 3)

Load	Load word from memory into register RZ. A register plus register shift addressing method is used.
impa ct	The effective address of the memory is determined by the base
Logo classifi er for honorific people	The address register RX plus the offset register RY is worth 2 bits immediately IMM2 is shifted to the left. the default value of IMM2 is 0.
except ions	Unaligned Access Exception, Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

32-bit instruction format:

ldr32.w rz, (rx, ry<< 0)

31	30	26 25	21 20	16 15	10 9	5 4	0
1	10100	RY	RX	000010	00001	RZ	

Figure 14.116: LDR.W-1

ldr32.w rz, (rx, ry<< 1)

31	30	26 25	21 20	16 15	10 9	5 4	0
1	10100	RY	RX	000010	00010	RZ	

Figure 14.117: LDR.W-2

ldr32.w rz, (rx, ry<< 2)

31	30	26 25	21 20	16 15	10 9	5 4	0
1	10100	RY	RX	000010	00100	RZ	

Figure 14.118: LDR.W-3

ldr32.w rz, (rx, ry<< 3)

31 30	26 25	21 20	16 15	10 9	5 4	0
1	10100	RY	RX	000010	01000	RZ

Figure 14.119: LDR.W-4

14.76 LRS.B - byte symbol loading instruction

Harmonization Directive	
tell to Bud dhist teac hing	lrs.b rz, [label]
drill (pra ctice) writi ngs or wor ks	Load bytes from memory to registers $RZ \leftarrow \text{zero_extend}(\text{MEM}[R28 + \text{zero_extend}(\text{offset})])$
c o m pi la ti o n re s ul t resol ute	Only 32-bit instructions exist. lrs32.b rz, [label]
cl ar ifi ca ti o n	The byte symbol of the label location is loaded and stored in the destination register RZ after zero expansion to 32 bits. register plus immediate offset addressing is used. The effective address of the memory is determined by adding the base address register R28 to the base address register plus an 18-bit relative offset unsigned expanded to 32 bits. The LRS.B instruction can address +256KB of address space. Note: The offset DISP is the binary operand Offset.

User's Manual	Manufactured ar k er s of in fl u e n ce class ifier for hon orifi c peop le
discriminate constant	Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

32-bit instruction	
drill (pra ctice) writi ngs or wor ks	Load byte symbols from memory to registers, unsigned extension $RZ \leftarrow \text{zero_extend}(\text{MEM}[R28 + \text{zero_extend}(\text{offset})])$
tell to Bud dhist teac hing	lrs32.b rz, [label]
cl ar ifi ca ti o n	The byte symbol of the label location is loaded and stored in the destination register RZ after zero expansion to 32 bits. register plus immediate offset addressing is used. The effective address of the memory is determined by adding the base address register R28 to the base address register plus an 18-bit relative offset unsigned expanded to 32 bits. The LRS.B instruction can address +256KB of address space. Note: The offset DISP is the binary operand Offset.
M ar k er s of in fl u e n ce class ifier for hon orifi c peop le	unaffected
discr imin	Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

32-bit instruction format:



Figure 14.120: LRS.

14.77 LRS.H - half-word symbol loading instruction

Harmonization Directive	
tell to Bud dhis t teac hing	lrs.h rz, [label]
drill (pra ctice) writi ngs or wor ks	RZ← zero_extend(MEM[R28+ zero_extend(offset<< 1)])
c o m pi la ti o n re s ul t reso lute	Only 32-bit instructions exist. lrs32.h rz, [label]
cl ar ifi ca ti o n	The half-word symbol of the loaded label location is zero-extended to 32 bits and stored in the destination register RZ. The register plus immediate offset addressing method is used. The effective address of the memory is determined by the base address register R2 8 plus an 18-bit relative offset shifted one bit to the left, unsigned, to 32 bits, and then stored in the destination register RZ. The LRS.H instruction addresses +512KB of address space. Note that the offset DISP is the binary operand Offset.
M ar k er s of	unaffected

in User's Manual	
fl u e n ce class ifier for hon orifi c peo ple	disc rimi nate cons tant

32-bit instruction	
drill (pra ctice) writi ngs or wor ks	Load half-word symbols from memory to registers, unsigned extension $RZ \leftarrow \text{zero_extend}(\text{MEM}[R28 + \text{zero_extend}(\text{offset} \ll 1)])$
tell to Bud dhis t teac hing	lrs32.h rz, [label]
cl ar ifi ca ti o n	The half-word symbol of the loaded label location is zero-extended to 32 bits and stored in the destination register RZ. The register plus immediate offset addressing method is used. The effective address of the memory is determined by the base address register R2 8 plus an 18-bit relative offset shifted one bit to the left, unsigned, to 32 bits, and then stored in the destination register RZ. The LRS.H instruction addresses +512KB of address space. Note that the offset DISP is the binary operand Offset.

User's Manual	Manufactured ar k er s of in fl u e n ce class ifier for hon orifi c peo ple
disc rimi nate cons tant	Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

32-bit instruction format:

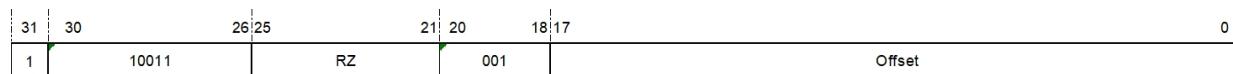


Figure 14.121: LRS.

14.78 LRS.W - Word Symbol Loading Instruction

Harmonization Directive	
tell to Bud dhis t teac hing	lrs.w rz, [label]
drill (pra ctice) writi ngs or wor ks	$RZ \leftarrow \text{zero_extend}(\text{MEM}[R28 + \text{zero_extend}(\text{offset} \ll 2)])$
c o m pi la ti o n re s ul t reso lute	Only 32-bit instructions exist. lrs32.w rz, [label]
cl ar ifi ca	The word symbol of the label location is loaded and stored in the destination register RZ after expanding to 32 bits by zero. register plus immediate offset addressing is used. The effective address of the memory is determined by the base address register R2 8 plus an 18-bit relative offset shifted 2 bits to the left, unsigned, to 32 bits, and then stored in the destination register RZ.

User's Manual	Note that the offset DISP is the binary operand Offset.
M ar k er s of in fl u e n ce class ifier for hon orifi c peo ple	unaffected
disc rimi nate cons tant	Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

32-bit instruction	
drill (pra ctice) writi ngs or wor ks	Load word symbols from memory to registers, unsigned extensions $RZ \leftarrow \text{zero_extend}(\text{MEM}[R28 + \text{zero_extend}(\text{offset} \ll 2)])$
tell to Bud dhis t teac hing	lrs32.w rz, [label]
cl ar ifi ca ti o n	The word symbol of the label location is loaded and stored in the destination register RZ after expanding to 32 bits by zero. register plus immediate offset addressing is used. The effective address of the memory is determined by the base address register R2 8 plus an 18-bit relative offset shifted 2 bits to the left, unsigned, to 32 bits, and then stored in the destination register RZ. Note: The offset DISP is the binary operand Offset.
M ar k er s of in fl u e n ce class ifier for hon orifi c peo ple	unaffected

disc	Access	Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception				
Users						
rimi						
nate						
cons						
tant						

32-bit instruction format:



Figure 14.122: LRS.

14.79 LRW - memory read instruction

Harmonization Directive	
gram matica l	lrw rz, label lrw rz, imm32
manip ulate	Load word from memory to register $RZ \leftarrow \text{zero_extend}(\text{MEM}[(\text{PC} + \text{zero_extend}(\text{offset} \ll 2)) \& 0xffffffff])$
Co mpi lati on resu lts	Compiles to 16-bit or 32-bit instructions depending on the range loaded. if(offset<512B), then lrw16 label; lrw16 imm32; else lrw32 label. lrw32 imm32.
clarifi cation	Load the word where label is located, or a 32-bit immediate number (IMM32) into the destination register RZ. The memory address is obtained by shifting the PC two bits to the left by the relative offset, expanding it unsigned to 32 bits, and then forcing it to zero by the lowest two bits. The load range is the entire 4GB address space.
Mar ker s of infl uen ce classif ier for honor ific peopl e	unaffected
except ions	Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

16-bit instructions	
drill (pra ctice) writi ngs or	Load word from memory to register $RZ \leftarrow \text{zero_extend}(\text{MEM}[(\text{PC} + \text{zero_extend}(\text{offset} \ll 2)) \& 0xffffffff])$

Users Manual	
tell to Bud dhist teac hing	lrw16 rz, label lrw16 rz, imm32
cl ar ifi ca ti o n	The LRW instruction loads the word where label is located, or a 32-bit immediate number (IMM32), into the destination register RZ . The memory address is obtained from the PC plus a 10-bit relative offset shifted two bits to the left, unsigned expanded to 32 bits, and then forced to zero by the lowest two bits. The load range of the LRW instruction is the entire 4GB address space. Note that the relative offset Offset is equal to the binary code {IMM2, IMM5}.
M ar k er s of in fl u e n ce class ifier for hon orifi c peop le	unaffected
discr imin ate cons tant	Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

16-bit instruction format:

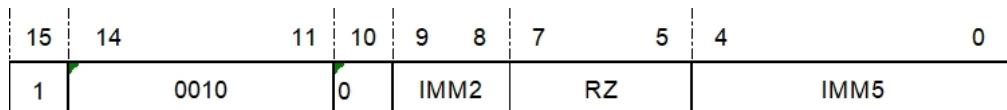


Figure 14.123: LRW-1

32-bit instruction	
manipulate	Load word from memory to register $RZ \leftarrow \text{zero_extend}(\text{MEM}[(\text{PC} + \text{zero_extend}(\text{offset} \ll 2)) \& 0xffffffffc])$
grammatical	lrw32 rz, label lrw32 rz, imm32
clarification	Load the word where label l is located, or a 32-bit immediate number (IMM32) into the destination register RZ. The memory address is obtained from the PC plus a 16-bit relative offset shifted two bits to the left and unsigned expanded to 32 bits, and then forced zeroed by the lowest two bits. The load range of the order is the entire 4GB address space.
Markers of influence classifier for honoriific people	unaffected
exceptions	Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

32-bit instruction format:

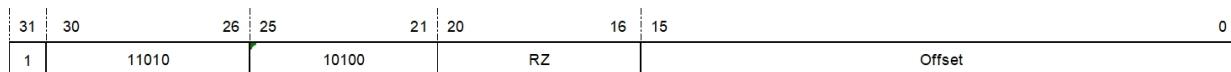


Figure 14.124: LRW-2

14.80 LSL - Logical Left Shift Instruction

Harmonization Directive		
gram matica 1	lsl rz, rx	lsl rz, rx, ry
manip ulate	RZ← RZ<< RX[5:0]	RZ← RX<< RY[5:0]
Co mpi lati on resu lts	<p>Compile to 16-bit or 32-bit depending on the register range.</p> <p>Bit Commands. if (x<16) and (z<16), then lsl16 rz, rx; else lsl32 rz, rz, rx.</p>	<p>Compiles to the corresponding 16-bit or 32-bit instructions depending on the register range.</p> <p>if (x==z) and (y<16) and (z<16), then lsl16 rz, ry else lsl32 rz, rx, ry</p>
clarifi cation	<p>For lsl rz, rx, the value of RZ is logically left-shifted (original value is left-shifted and the right-hand side is shifted into 0), and the result is deposited into RZ, with the number of left-shifted bits determined by the value of the 6 lower bits of RX (RX[5:0]); if the value of RX[5:0] is greater than 31, then RZ is cleared to zero;</p> <p>For lsl rz, rx, ry, the value of RX is logically shifted left (the original value is shifted left, and the right-hand side is shifted into 0) and the result is deposited into RZ, with the number of left-shifted bits</p> <p>Determined by the value of the 6 bits below RY (RY[5:0]); if the value of RY[5:0] is greater than 31, then RZ will be cleared.</p>	
Mar kers of infl uen ce classi fier for hono rific peopl e	unaffected	
except ions	not have	

16-bit instructions	
manip ulate	RZ← RZ<< RX[5:0]
gram	lsl16 rz, rx

Technical Users Manual 1	
clarification	Logical left shift of the value of RZ (original value is shifted left, right-hand side is shifted to 0), the result is stored in RZ, and the number of bits shifted left is 6 bits lower than that of RX. (RX[5:0]) is determined; if the value of RX[5:0] is greater than 31, then RZ is cleared.
impact	unaffected
Logo classifier for honorific people	
limitation	The range of the register is r0-r15.
exceptions	not have

16-bit instruction format:

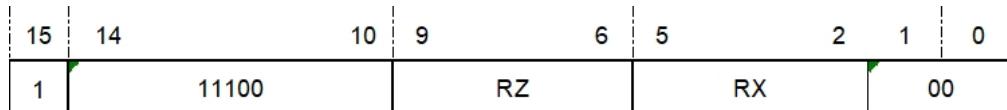


Figure 14.125: LSL-1

32-bit instruction	
manipulate	RZ← RX<< RY[5:0]
grammatical	lsl32 rz, rx, ry
clarification	Logical left shift of the value of RX (original value is shifted left, right side is shifted to 0), the result is stored in RZ, and the number of left shifted bits is 6 bits lower than RY. (RY[5:0]) is determined; if the value of RY[5:0] is greater than 31, then RZ is cleared.
impact	unaffected
Logo classifier for honorific people	
exceptions	not have

32-bit instruction format:

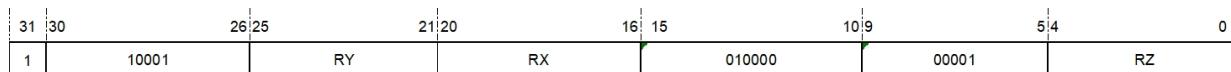


Figure 14.126: LSL-2

14.81 LSLC - Immediate Logic Left Shift to C bit instruction

Harmonization Directive	
grammatical	lslc rz, rx, oimm5
manipulate	RZ← RX<< OIMM5, C← RX[32 - OIMM5]
compiling in the end	Only 32-bit instructions exist. lslc32 rz, rx, oimm5

Users Manual	The value of RX is logically shifted left (the original value is shifted left, and the right side is shifted into 0) and the last bit of the shifted value is stored in the condition bit C, and the result of the shift is stored in RZ, and the number of left shifted bits is determined by the value of the 5-bit immediate number with offset 1 (OIMM5). If the value of OIMM5 is equal to 32, then Condition bit C is the lowest bit of RX and RZ is cleared.
Markers of influence classifier for honorific people	$C \leftarrow RX[32 - OIMM5]$
limitation	Immediate numbers range from 1-32.
exceptions	not have

32-bit instruction	
drill (pra ctice) writi ngs or wor ks	RZ← RX<< OIMM5, C← RX[32 - OIMM5]
tell to Bud dhist teac hing	lslc32 rz, rx, oimm5
cl ar ifi ca ti o n	<p>The value of RX is logically shifted left (the original value is shifted left, and the right side is shifted into 0), and the last bit of the shifted bit is stored in the condition bit C, and the result of the shift is stored in RZ, and the number of left shifted bits is determined by the value of the 5-bit immediate number with bias 1 (OIMM5). If the value of OIMM5 is equal to 32, the condition bit C is the lowest bit of RX and RZ is cleared to zero.</p> <p>Note: The binary operand IMM5 is equal to OIMM5 - 1.</p>
M ar k er s of in fl u e n ce class ifier for hono rific peop le	C← RX[32 - OIMM5]
set a limit (on)	Immediate numbers range from 1-32.

User Manual	late
discriminate constant	not have

32-bit instruction format:



Figure 14.127: LSLC

IMM5 domain:

Specifies a value without a bias immediate number.

Note: The shifted value OIMM5 is biased by 1 compared to the binary operand IMM5.

00000:

Shift 1 bit

00001:

Move 2 positions

.....

11111:

Shift 32 bits

14.82 LSLI - Immediate Logical Left Shift Instruction

Harmonization Directive	
grammatical	lslri rz, rx, imm5
manipulate	RZ ← RX << IMM5
Translation results	<p>Compiles to the corresponding 16-bit or 32-bit instructions depending on the register range.</p> <p>if (x<8) and (z<8), then lslri16 rz, rx, imm5</p> <p>else lslri32 rz, rx, imm5</p>
clarification	<p>The value of RX is logically shifted left (the original value is shifted left and the right side is shifted to 0), and the result is stored in RZ, with the number of left shifted bits consisting of 5 immediate bits.</p> <p>(IMM5) is determined; if the value of IMM5 is equal to 0, then the value of RZ will be unchanged.</p>
impact	unaffected
Logo classifier for honorific people	
limitations	The range of immediate numbers is 0-31.
exceptions	not have

16-bit instructions	
manipulate	RZ ← RX << IMM5
grammatical	lslri16 rz, rx, imm5
clarification	<p>The value of RX is logically shifted left (the original value is shifted left and the right side is shifted to 0), and the result is stored in RZ, with the number of left shifted bits consisting of 5 immediate bits.</p> <p>(IMM5) is determined; if the value of IMM5 is equal to 0, then the value of RZ will be unchanged.</p>
impact	unaffected
Logo classifier for	

User Manual	
ic people	
limitati on	The range of the register is r0-r7; The range of immediate numbers is 0-31.
excepti ons	not have

16-bit instruction format:

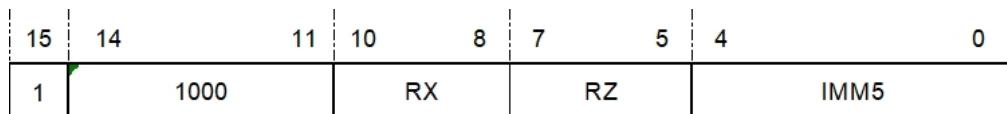


Figure 14.128: LSLI-1

32-bit instruction	
mani pulat e	RZ ← RX << IMM5
gram matic al	lsli32 rz, rx, imm5
clarifi catio n	The value of RX is logically shifted left (the original value is shifted left and the right side is shifted to 0), and the result is stored in RZ, with the number of left shifted bits consisting of 5 immediate bits. (IMM5) is determined; if the value of IMM5 is equal to 0, then the value of RZ will be as RX.
impac t Logo classifi er for honorif ic people	unaffected
limita tion	The range of immediate numbers is 0-31.
excep tions	not have

32-bit instruction format:



Figure 14.129: LSLI-2

14.83 LSR - Logical Right Shift Instruction

Harmonization Directive		
tell to Budd hist teaching	lsr rz, rx	lsr rz, rx, ry
drill (practice) writings or works	RZ← RZ>> RX[5:0]	RZ← RX>> RY[5:0]
C o m p i l a t i o n r e s u l t s	<p>Compile to 16-bit or 32-bit depending on the register range.</p> <p>Bit Commands.</p> <pre>if (z<16) and (x<16), then lsr16 rz, rx ;Â else lsr32 rz, rz, rx.</pre>	<p>Compiles to the corresponding 16-bit or 32-bit instructions depending on the register range.</p> <pre>if (x==z) and (z<16) and (y<16), then lsr16 rz, ry ; else lsr32 rz, rx, ry.</pre>
clarifi ca ti on	<p>For lsr rz, rx, the value of RZ is logically shifted to the right (the original value is shifted right, and the left side is shifted into 0), and the result is deposited into RZ, with the number of bits shifted to the right being determined by the value of the 6 lower bits of RX (RX[5:0]); if the value of RX[5:0] is greater than 31, then RZ is cleared to zero.</p> <p>For lsr rz, rx, ry, the value of RX is logically shifted to the right (the original value is shifted to the right and the left side is shifted into 0) and the result is deposited into RZ, with the number of right-shifted bits</p> <p>Determined by the value of the 6 bits below RY (RY[5:0]); if the value of RY[5:0] is greater than 31, then RZ will be cleared.</p>	
M ar k er s of in fl	unaffected	

User's Manual	
discriminate constant	not have

16-bit instructions	
manipulate	RZ← RZ>> RX[5:0]
grammatical	lsr16 rz, rx
clarification	Logically shift the value of RZ to the right (the original value is shifted to the right, and the left side is shifted to 0), and the result is stored in RZ, with the right shift number 6 bits lower than that of RX. (RX[5:0]) is determined; if the value of RX[5:0] is greater than 31, then RZ is cleared.
impact	unaffected
Logo classifier for honorific people	
limitation	The range of the register is r0-r15.
exceptions	not have

16-bit instruction format:



Figure 14.130: LSR-1

32-bit instruction	
manipulate	$RZ \leftarrow RX >> RY[5:0]$
grammatical	lsr32 rz, rx, ry
clarification	Logically shift the value of RX to the right (the original value is shifted to the right, and the left side is shifted to 0), and the result is deposited into RZ, with the right shift number 6 bits lower than that of RY. (RY[5:0]) is determined; if the value of RY[5:0] is greater than 31, then RZ is cleared.
impact	unaffected
Logo classifier for honorific people	
exceptions	not have

32-bit instruction format:



Figure 14.131: LSR-2

14.84 LSRC - Immediate Logic Right Shift to C-Bit Instruction

Harmonization Directive	
grammatical	lsrc rz, rx, oimm5

User Manual	RZ<= RX>> OIMM5, C← RX[OIMM5 - 1]
com pilin g in the end	Only 32-bit instructions exist. lsrc32 rz, rx, oimm5
clarifi cation	The value of RX is logically shifted to the right (the original value is shifted to the right, and the left side is shifted into 0) and the last bit of the shift is stored in the condition bit C, and the result of the shift is stored in RZ, and the number of bits shifted to the right is determined by the value of the 5-bit immediate number with bias 1 (OIMM5). If the value of OIMM5 is equal to 32, then Condition bit C is the highest bit of RX and RZ is cleared.
Mar ker s of infl uen ce classi fier for hono rific peopl e	C← RX[OIMM5 - 1]
limitat ion	Immediate numbers range from 1-32.
except ions	not have

32-bit instruction	
drill (pra ctice) writi ngs or wor ks	RZ← RX>> OIMM5, C← RX[OIMM5 - 1]
tell to Bud dhist teac hing	lsrc32 rz, rx, oimm5
cl ar ifi ca ti o n	The value of RX is logically shifted to the right (the original value is shifted to the right, and the left side is shifted to 0), and the last bit of the shifted bit is stored in the condition bit C, and the result of the shift is stored in RZ, and the number of bits shifted to the right is determined by the value of the 5-bit immediate number with bias 1 (OIMM5). If the value of OIMM5 is equal to 32, the condition bit C is the highest bit of RX and RZ is cleared to zero. Note: The binary operand IMM5 is equal to OIMM5 - 1.
M ar k er s of in fl u e n ce class ifier for hono rific peop le	C← RX[OIMM5 - 1]
set a limit (on)	Immediate numbers range from 1-32.

User Manual	late
discriminate constant	not have

32-bit instruction format:



Figure 14.132: LSRC

IMM5 domain:

Specifies a value without a bias immediate number.

Note: The shifted value OIMM5 is biased by 1 compared to the binary operand IMM5.

00000:

Shift 1 bit

00001:

Move 2 positions

.....

11111:

Shift 32 bits

14.85 LSRI - Logical Right Shift Instruction for Immediate Numbers

Harmonization Directive	
grammatical	lsri rz, rx, imm5
manipulate	RZ ← RX >> IMM5
Translation result	Compiles to the corresponding 16-bit or 32-bit instructions depending on the register range. if (x<8) and (z<8), then lsri16 rz, rx, imm5 else lsri32 rz, rx, imm5
clarification	The value of RX is logically shifted to the right (the original value is shifted to the right, and the left side is shifted to 0), and the result is stored in RZ, and the number of bits shifted to the right consists of 5 immediate bits. (IMM5); if the value of IMM5 is equal to 0, then the value of RZ is unchanged or will be as RX.
impact Logo classifier for honorific people	unaffected
limitation	The range of immediate numbers is 0-31.
exceptions	not have

16-bit instructions	
manipulate	RZ ← RX >> IMM5
grammatical	lsri16 rz, rx, imm5
clarification	The value of RX is logically shifted to the right (the original value is shifted to the right, and the left side is shifted to 0), and the result is stored in RZ, and the number of bits shifted to the right consists of 5 immediate bits. (IMM5) is determined; if the value of IMM5 is equal to 0, then the value of RZ will be unchanged.
impact Logo	unaffected

Classification User Manual	
range for honorific ic people	
limitation on	The range of registers is r0-r7; the range of immediate numbers is 0-31.
exceptions	not have

16-bit instruction format:

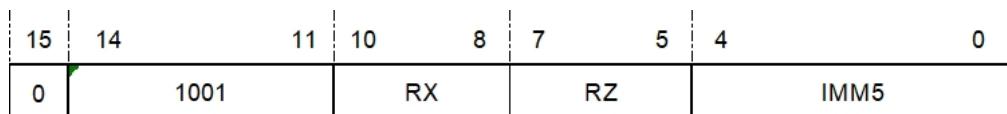


Figure 14.133: LSRI-1

32-bit instruction	
manipulate	RZ ← RX >> IMM5
grammatical	lsri32 rz, rx, imm5
clarification	The value of RX is logically shifted to the right (the original value is shifted to the right, and the left side is shifted to 0), and the result is stored in RZ, and the number of bits shifted to the right consists of 5 immediate bits. (IMM5) is determined; if the value of IMM5 is equal to 0, then the value of RZ will be as RX.
impact	unaffected
Logo classifier for honorific people	
limitation	The range of immediate numbers is 0-31.
exceptions	not have

32-bit instruction format:

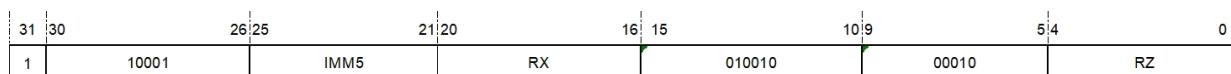


Figure 14.134: LSRI-2

14.86 MFCR - Control Register Read Transfer Instruction

Harmonization Directive	
grammatical	mfcrrz, cr<x, sel>
manipulate	Transferring the contents of a control register to a general purpose register RZ← CR<X, sel>
Compilation results	Only 32-bit instructions exist. mfcrr32 rz, cr<x, sel>
Attributes:	privileged instruction

clarification	Transfer the contents of control register CR<x, sel> to general purpose register RZ.
Affecting Flags	unaffected
exceptions	Privilege violation exceptions

32-bit instruction	
manipulate	Transferring the contents of a control register to a general purpose register RZ← CR<X, sel>
grammatical	mfcr32 rz, cr<x, sel>
Attributes:	privileged instruction
clarification	Transfer the contents of control register CR<x, sel> to general purpose register RZ.
Affecting Flags	unaffected
exceptions	Privilege violation exceptions

32-bit

instruction	26 25	21 20	16 15	10 9	5 4	0
1 format ₄₀₀₀	sel	CRX	011000	00001	RZ	

Figure 14.135: MFCR

14.87 MOV - Data Transfer Instructions

Harmonization Directive	
grammatical	mov rz, rx
manipulate	RZ← RX
Compilation results	Always compiles to 16-bit instructions. mov16 rz, rx
clarification	Copies the value in RX to destination register RZ.
Affecting Flags	unaffected
exceptions	not have

16-bit instructions	
manipulate	RZ← RX
grammatical	mov16 rz, rx
clarification	Copies the value in RX to destination register RZ. Note that the instruction register index range is r0-r31.
Affecting Flags	unaffected
exceptions	not have

16 Format of mosquito instructions:

15	14	10	9	6	5	2	1	0
1	11011	RZ	RX	11				

Figure 14.136: MOV-1

32-bit instruction	
manipulate	RZ← RX
grammatical	mov32 rz, rx

32-bit

instruction		26 25	21 20	16 15	10 9	5 4	0
1	format ₄₀₀₀	sel	CRX	011000	00001	RZ	
clarification	Copies the value in RX to destination register RZ. Note that this instruction is a pseudo-instruction for lsli32 rz, rx, 0x0.						
Affecting Flags	unaffected						
exceptions	not have						

32-bit

31 30	instruction	26 25	21 20	16 15	10 9	5 4	0
1	format000	sel	CRX	011000	00001	RZ	

Figure 14.137: MOV-2

14.88 MOVF--C is 0 Data Transfer Instruction

Harmonization Directive	
grammatical	movf rz, rx
manipulate	if C==0, then RZ ← RX. else RZ← RZ.
Translation resolute	Only 32-bit instructions exist. movf32 rz, rx
clarification	If C is 0, copy the value of RX to destination register RZ; otherwise, the value of RZ remains unchanged. Note that this instruction is a pseudo-instruction for incf rz, rx, 0x0.
impact indicator aspiration	unaffected
exception s	not have

32-bit instruction	
manipulate	if C==0, then RZ ← RX. else RZ← RZ.
grammatical	movf32 rz, rx
clarification	If C is 0, copy the value of RX to destination register RZ; otherwise, the value of RZ remains unchanged. Note that this instruction is a pseudo-instruction for incf32 rz, rx, 0x0.
impact indicator	unaffected

32-bit

instruction	26 25	21 20	16 15	10 9	5 4	0
1 format0000	sel	CRX	011000	00001	RZ	
aspiration						
exceptions	not have					

32-bit instruction format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1	10001	RZ	RX	000011	00001	00000

Figure 14.138: MOVF

14.89 MOVI - Immediate Number Data Transfer Instruction

Harmonization Directive	
grammatical	movi rz, imm
manipulate	RZ← zero_extend(IMM).
compilation results	Compiles to the corresponding 16-bit or 32-bit instructions depending on the range of the immediate numbers and registers. if (imm16<256) and (z<7), then movi16 rz, imm8; else movi32 rz, imm16.
clarification	Extends a 16-bit immediate zero to 32 bits and transfers it to the destination RZ.
affect aspiration	unaffected
limitation	The range of immediate numbers is 0x0-0xFFFF.
exceptions	not have

16-bit instructions	
manipulate	RZ← zero_extend(IMM8).
grammatical	movi16 rz, imm8
clarification	Extends an 8-bit immediate zero to 32 bits and transfers it to the destination RZ.
Affecting Flags	unaffected
limitation	The range of registers is r0-r7; the range of immediate numbers is 0-255.
exceptions	not have

16-bit instruction format:

User's Mat

15	14	11	10	8	7	0
0	0110	RZ		IMM8		

Figure 14.139: MOVI-1

32-bit instruction	
manipulate	RZ← zero_extend(IMM16).
grammatical	movi32 rz, imm16
clarification	Extends a 16-bit immediate zero to 32 bits and transfers it to the destination RZ.
Affecting Flags	unaffected
limitation	The range of immediate numbers is 0x0-0xFFFF.
exceptions	not have

32-bit instruction format:

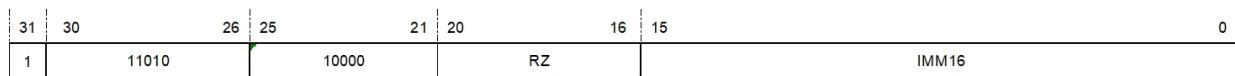


Figure 14.140: MOVI-2

14.90 MOVIH - Immediate Number High Data Transfer Instruction

Harmonization Directive	
grammatical	movih rz, imm16
manipulate	RZ← zero_extend(IMM16)<< 16
compilation result resolute	Only 32-bit instructions exist. movih32 rz, imm16
clarification	Extends a 16-bit immediate number zero to 32 bits, then logically shifts it left by 16 bits to transfer the result to the destination RZ. This instruction can be used in conjunction with the ori rz, rz, imm16 instructions to generate any 32-bit immediate number.
affect aspiration	unaffected
limitation	The range of immediate numbers is 0x0-0xFFFF.
exceptions	not have

User's Manual

32-bit instruction	
manipulate	RZ← zero_extend(Imm16)<< 16
grammatical	movih32 rz, imm16
clarification	<p>Extends a 16-bit immediate number zero to 32 bits, then logically shifts it left by 16 bits to transfer the result to the destination RZ.</p> <p>This instruction can be used in conjunction with the ori32 rz, rz, imm16 instructions to generate any 32-bit immediate number.</p>
affect aspiration	unaffected
limitation	The range of immediate numbers is 0x0-0xFFFF.
exceptions	not have

32-bit instruction format:

31	30	26	25	21	20	16	15	0
1	11010		10001		RZ		IMM16	

Figure 14.141: MOVIH

14.91 MOVT--C is a 1 data transfer command.

Harmonization Directive	
grammatical	movt rz, rx
manipulate	if C==1, then RZ ← RX. else RZ← RZ.
Compilation results	Only 32-bit instructions exist. movt32 rz, rx
Affecting Flags	unaffected
exceptions	not have

32-bit instruction	
manipulate	if C==1, then RZ ← RX. else RZ← RZ.
grammatical	movt32 rz, rx
clarification	If C is 1, copy the value of RX to destination register RZ; otherwise, the value of RZ remains unchanged. Note that this instruction is a pseudo-instruction for inct32 rz, rx, 0x0.
impact indicator aspiration	unaffected
exceptions	not have

32-bit instruction format:

	31 30	26 25	21 20	16 15	10 9	5 4	0
1	10001	RZ	RX	000011	00010	00000	

Figure 14.142: MOVT

14.92 MTCR - Control Register Write Transfer Instruction

Harmonization Directive	
grammatical	mtcr rx, cr<z, sel>
manipulate	Transferring the contents of a general-purpose register to a control register CR<Z, sel><- RX
Compilation results	Only 32-bit instructions exist. mtcr32 rx, cr<z, sel>
Attributes:	privileged instruction
clarification	Transfers the contents of general purpose register RX to control register CR<z, sel>.
Affecting Flags	If the target control register is not a PSR, this instruction does not affect the flag bits.
exceptions	Privilege violation exceptions

32-bit instruction	
manipulate	Transferring the contents of a general-purpose register to a control register CR<Z, sel><- RX
grammatical	mtcr32 rx, cr<z, sel>
Attributes:	privileged instruction
clarification	Transfers the contents of general purpose register RX to control register CR<z, sel>.
Affecting Flags	If the target control register is not a PSR, this instruction does not affect the flag bits.
exceptions	Privilege violation exceptions

32-bit instruction format:



Figure 14.143: MTCR

14.93 MULA.32.I - 32-bit signed multiply-accumulate-take-low 32-bit instructions

tion Directive	
grammatica l	mula.32.1 rz, rx, ry
manipulate	Rz[31:0] <- Rz[31:0]+ {Rx[31:0] X Ry[31:0]}[31:0]
Compilatio n results	Only 32-bit instructions exist. mula.32.1 rz, rx, ry

Description:	Rx is multiplied by Ry to get a 64-bit result, added Rz, and the lower 32 bits of the result are intercepted and stored in Rz.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	not have

32-bit instruction	
Operation:	$Rz[31:0] \leftarrow Rz[31:0] + \{Rx[31:0] \times Ry[31:0]\}[31:0]$
Grammar:	mula.32.1 rz, rx, ry
Description:	Rx is multiplied by Ry to get a 64-bit result, added Rz, and the lower 32 bits of the result are intercepted and stored in Rz.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1	11110	RY	RX	000010	0 0 1 0 0	Rz

Figure 14.144: MULA.32.1

14.94 MULALL.S16.S - 16-bit Signed Low Halfword Multiply Accumulator with Saturation Operation honorific title

Harmonization Directive	
grammatical	mulall.s16.s rz, rx, ry
manipulate	$Rz[31:0] = \text{Saturate}(Rz[31:0] + Rx[15:0] \times Ry[15:0])$
Compilation results	Only 32-bit instructions exist. mulall.s16.s rz, rx, ry

Explana tion: :	The lower half-word of Rx is multiplied by the lower half-word of Ry, the result of the multiplication is added to Rz, and the result of the addition is saturated and stored in Rz. The process of saturation processing is that if the addition result is greater than the upper saturation value 0x7FFF FFFF , the result is the upper saturation value, if the addition result is smaller The result is the lower saturated value 0x8000 0000, in all other cases it is the result of the addition itself.
Mark ers of infl uen ce Bit:	unaffected
set a limit (on) Syste m:	not have
discri mina te Regu lar:	not have

32 index (mat h.) hono rific title	
drill (prac tice) Mak e:	Rz[31:0]= Saturate(Rz[31:0]+ Rx[15:0] X Ry[15:0])
tell to Law:	mulall.s16.s rz, rx, ry
Explana tion: :	The lower half-word of Rx is multiplied by the lower half-word of Ry, the result of the multiplication is added to Rz, and the result of the addition is saturated and stored in Rz. The process of saturation processing is that if the addition result is greater than the upper saturation value 0x7FFF FFFF , the result is the upper saturation value, if the addition result is

User's Manual	The result is the lower saturated value 0x8000 0000, in all other cases it is the result of the addition itself.
Mark ers of infl uen ce Bit:	unaffected
set a limit (on) Syste m:	not have
discri mina te Regu lar:	not have

Command Format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 11110	RY	RX	1 0 0 0 0 0	0 1 1 0 1	Rz	

Figure 14.145: MULALL.S16.

14.95 MULA.(U/S)32 - 32-bit (no/have) symbolic multiply-accumulate instruction

Harmonization Directive	
grammatical	mula.u32 rz, rx, ry mula.s32 rz, rx, ry
manipulation results	$\{Rz+1[31:0], Rz[31:0]\} = \{Rz+1[31:0], Rz[31:0]\} + Rx[31:0] \times Ry[31:0]$
Compilation results	Only 32-bit instructions exist. mula.u32 rz, rx, ry mula.s32 rz, rx, ry

Description:	Rx is multiplied by Ry to get a 64-bit result and added to {Rz+1, Rz}. The upper 32 bits of the result are stored in Rz+1. The lower 32 bits are stored in Rz.
Markers of influence Bit:	unaffected
Limitations:	not have
Exception:	not have

32-bit instruction	
Operation :	$\{Rz+1[31:0], Rz[31:0]\} = \{Rz+1[31:0], Rz[31:0]\} + Rx[31:0] \times Ry[31:0]$
Grammar:	mula.u32 rz, rx, ry mula.s32 rz, rx, ry
Description:	Rx is multiplied by Ry to get a 64-bit result and added to {Rz+1, Rz}. The upper 32 bits of the result are stored in Rz+1. The lower 32 bits are stored in Rz.
Markers of influence Bit:	unaffected
Limitations:	not have

Command Format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 11110	RY	RX	1 0 0 0 0 0	S 0 1 0 0	Rz	

Figure 14.146: MULA

14.96 MULT - Multiplication Instruction

c ol le ct iv el y k n o w n as "h ar m o ni za ti o n" or "h ar m o ni za ti o n in d e x" (m at h.) hono rific title		
---	--	--

User Manual	tell to mult rz, rx	mult rz, rx, ry
Buddhist teaching		
male	Two numbers are multiplied and the lower 32 bits of the result are placed in a general purpose register. $RZ \leftarrow RX \times RZ.$	Two numbers are multiplied and the lower 32 bits of the result are placed in a general purpose register. $rz \leftarrow rx \times ry.$

Description:	The lower 32 bits of the result of multiplying the contents of the two source registers are stored in the destination register, and the upper 32 bits of the result are discarded. No matter whether the source operand is considered a signed or unsigned number results in the same outcome.
affect Chi Bit:	unaffected
Exception:	not have

16 bits directives	
Operation:	Two numbers are multiplied and the lower 32 bits of the result are placed in a general purpose register. $RZ \leftarrow RX \times RZ.$
Grammar:	mult16 rz, rx
Description:	The lower 32 bits of the result obtained by multiplying the contents of general-purpose registers RX and RZ are stored general-purpose register RZ. Higher 32-bit rounding. The result is the same whether the source operand is considered a signed

User's Manual	unsigned number.
impact	unaffected
Logo Bit:	
Limitations:	The range of the register is r0-r15.
Exception:	not have

Command Format:

15	14	10	9	6	5	2	1	0
0		11111		RZ		RX		00

Figure 14.147: MULT

32 bits directives	
Operation:	Two numbers are multiplied and the lower 32 bits of the result are placed in a general purpose register. $rz \leftarrow rx \times ry$.
Grammar:	mult32 rz, rx, ry
Description:	The lower 32 bits of the result of multiplying the contents of general-purpose registers RX and RY are stored in general-purpose register RZ, and the upper 32 bits of the result are stored general-purpose register RZ. Bit rounding. The result is the same whether the source operand is considered a signed or unsigned number.
affect Chi Bit:	unaffected
Exception:	not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1	1 0 0 0 1	RY	RX	1 0 0 0 1	0 0 0 0 1	Rz

Figure 14.148: MULT2

14.97 MUL.(U/S)32 - 32-bit (no/have) signed multiply instruction

Harmonization Directive	
grammatical	mul.u32 rz, rx, ry mul.s32 rz, rx, ry
manipulate	$\{Rz+1[31:0], Rz[31:0]\} = Rx[31:0] \times Ry[31:0]$

Compilation results	Only 32-bit instructions exist. mul.u32 rz, rx, ry mul.s32 rz, rx, ry
---------------------	---

Description:	Rx is multiplied by Ry to get a 64-bit result, where the high 32 bits are stored in Rz+1 and the low 32 bits are stored in Rz.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	not have

32-bit instruction	
Operation:	{Rz+1[31:0],Rz[31:0]}= Rx[31:0] X Ry[31:0]
Grammar:	mul.u32 rz, rx, ry mul.s32 rz, rx, ry
Description:	Rx is multiplied by Ry to get a 64-bit result, where the high 32 bits are stored in Rz+1 and the low 32 bits are stored in Rz.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 1 0	RY	RX	1 0 0 0 0	S 0 0 0 0	0	Rz

Figure 14.149: MUL

14.98 MVC - C Bit Transfer Instructions

Harmonization Directive	
grammatical	mvc rz
manipulate	RZ← C
Compilation results	Only 32-bit instructions exist. mvc32 rz.
clarification	Transmit condition bit C to the lowest bit of RZ and clear the other bits of RZ to zero.
Affecting Flags	unaffected
exceptions	not have

32-bit instruction	
manipulate	RZ← C
grammatical	.mvc32 rz
clarification	Transmit condition bit C to the lowest bit of RZ and clear the other bits of RZ to zero.
Affecting Flags	unaffected

32-bit instruction format:

31	30	26 25	21 20	16 15	10 9	5 4	0
1	10001	00000	00000	000001	01000	RZ	

Figure 14.150: MVC

14.99 MCV - C Bit Fetch and Invert Transmission

Harmonization Directive	
grammatical	mvcv rz
manipulate	RZ← (!C)
Compilation results	Compiles to the corresponding 16-bit or 32-bit instructions depending on the register range. if (z<16), then mvcv16 rz; else mvcv32 rz;
clarification	The condition bit C is inverted and transmitted to the lowest bit of RZ, and the other bits of RZ are cleared to zero.
Impact Marker classifier for honorific people	unaffected
exceptions	not have

16-bit instructions	
manipulate	RZ← (!C)
grammatical	mvcv16 rz
clarification	The condition bit C is inverted and transmitted to the lowest bit of RZ, and the other bits of RZ are cleared to zero.
Affecting Flags	unaffected
limitation	The range of the register is r0-r15.
exceptions	not have

16-bit instruction format:

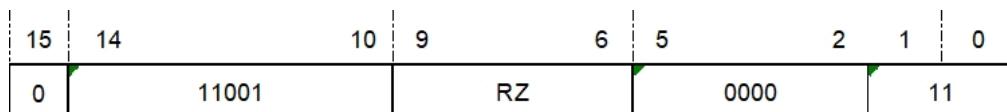


Figure 14.151: MCV-1

32-bit instruction	
manipulate	RZ← (!C)

grammatical	mvcv32 rz
clarification	The condition bit C is inverted and transmitted to the lowest bit of RZ, and the other bits of RZ are cleared to zero.
Affecting Flags	unaffected
exceptions	not have

32-bit instruction format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1	10001	00000	00000	000001	10000	RZ

Figure 14.152: MCV-2

14.100 NIE - Nested Interrupt Enable Instruction

Harmonization measures honorific title	
grammatical 1	nie
manipulate	Store the control register site {EPSR, EPC} for the interrupt into stack memory, then update the stack pointer register to the top of stack memory and turn on PSR.IE and PSR.EE; MEM[SP-4] ← EPC. MEM[SP-8] ← EPSR. SP ← SP-8. PSR({EE,IE}) ← 1;
Translation in the end	16-bit instructions only nie.

Attributes:	privileged instruction
Description:	Save the control register site {EPSR, EPC } for the interrupt to the stack memory, then update the stack pointer register to the top of the stack pointer memory and turn on the interrupt and exception enable bits PS.R.IE and PSR.EE. Use the stack pointer register to directly seek the Address approach.
impact Logo Bit:	unaffected
Exception:	Access Error Exception, Unaligned Exception, Privilege Violation Exception

16 Bit honorific title	
Operation:	Store the control register site {EPSR, EPC} for the interrupt into stack memory, then update the stack pointer register to the top of stack memory and turn on PSR.IE and PSR.EE; MEM[SP-4] ← EPC. MEM[SP-8] ← EPSR. SP ← SP-8. PSR({EE,IE})← 1;
Grammar:	nie16
Properties:	privileged instruction
Description:	Save the control register site {EPSR, EPC } for the interrupt to stack memory, then update the stack pointer register to the top of the stack pointer memory and turn on the interrupt and exception enable bits PSR.IE and PSR.EE. Direct addressing using the stack pointer registers Way.
Impact Signs Bit:	unaffected
Exception:	Unaligned Access Exception, Unaligned Access Exception, Access Error Exception

Command Format:

15	14	10	9	8	7	5	4	0
0	00101	0	0	011	00000			

Figure 14.153: NIE

14.101 NIR - Nested Interrupt Return Instruction

Harmonization means honorific title	
grammatical	nir
manipulate	<p>Load the control register site for the interrupt from stack memory into {EPSR, EPC }, then update the stack pointer register to the top of stack memory; and interrupt return</p> <p>$\text{EPSR} \leftarrow \text{MEM}[\text{SP}]$.</p> <p>$\text{EPC} \leftarrow \text{MEM}[\text{SP}+4]$.</p> <p>$\text{SP} \leftarrow \text{SP}+8$.</p> <p>$\text{PSR} \leftrightarrow \text{EPSR}$.</p> <p>$\text{PC} \leftarrow \text{EPC}$.</p>
Translation in the end	<p>16-bit instructions only</p> <p>nir.</p>

be born in the year of (one of the 12 animals) Sex:	privileged instruction
Explanation :	<p>Load the site of the interrupt from stack memory into {EPSR, EPC }, then update the stack pointer register to the top of stack memory; the PC value is restored to the value in control register EPC, the PSR value is restored to value of EPSR, and the instruction executes from the new PC</p> <p>Start at address. Direct addressing using stack pointer registers.</p>

User's Manual Marker s of infl uen ce Bit:	Manu factured
discri minat e Regul ar:	Access Error Exception, Unaligned Exception, Privilege Violation Exception

16 inde x (mat h.) hono rific title	
Ope rate :	Load the control register site for the interrupt from stack memory into {EPSR, EPC }, then update the stack pointer register to the top of stack memory; and interrupt return EPSR←MEM[SP]. EPC←MEM[SP+4]. SP←SP+8. PSR ↔ EPSR. PC ← EPC.
tell to Law:	nir16
be born in the year of (one of the 12 anim als) Sex:	privileged instruction
Exp lana tion :	Load the site of the interrupt from stack memory into {EPSR, EPC }, then update the stack pointer register to the top of stack memory; the PC value is restored to the value in control register EPC, the PSR value is restored to value of EPSR, and the instruction executes from the new PC Start at address. Direct addressing using stack pointer registers.
Mar ker s of infl uen ce Bit:	unaffected
discri minat e Regul	Unaligned Access Exception, Unaligned Access Exception, Access Error Exception

Command Format:

15	14	10	9	8	7	5	4	0
0	00101	0	0	011		00001		

Figure 14.154: NIR

14.102 NOR--Non-Order by Bit

Harmonization Directive		
tell to Budd hist teach ing	nor rz, rx	or rz, rx, ry
drill (practice) writings or works	RZ← ! (RZ RX)	RZ← ! (RX RY)
C o m p i l a t i o n r e s u l t s	Compile to 16-bit or 32-bit depending on the register range. Bit Commands. if (x<16) and (z<16), then nor16 rz, rx; else nor32 rz, rx, ry.	Compiles to the corresponding 16-bit or 32-bit instructions depending on the register range. if (y==z) and (x<16) and (z<16), then nor16 rz, rx else nor32 rz, rx, ry
persuade generic term for a sacrifice the gods	Bitwise-or the value of RX and RY/RZ, then bitwise-non, and store the result in RZ.	
M ar k er s of in	unaffected	

fl	User's Manual
u	
e	
n	
ce	
classi	
fier	
for	
hono	
rific	
peopl	
e	
discr	not have
mina	
te	
const	
ant	

16-bit instructions	
manipulate	RZ← ! (RZ RX)
grammatical	nor16 rz, rx
clarification	Bitwise-or the values of RZ and RX, then bitwise-non, and store the result in RZ.
Affecting Flags	unaffected
limitation	The range of the register is r0-r15.
exceptions	not have

16-bit instruction format:

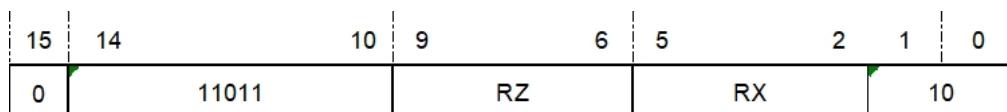


Figure 14.155: NOR-1

32-bit instruction	
manipulate	RZ← ! (RX RY)
grammatical	nor32 rz, rx, ry
clarification	Bitwise-or the values of RX and RY, and then bitwise-fetching the result and presenting it to RZ.
Affecting Flags	unaffected
exceptions	not have

32-bit instruction format:



Figure 14.156: NOR-2

14.103 NOT - per-bit non-instruction

Harmonization Directive		
grammatical	not rz	not rz, rx
manipulate	RZ← ! (RZ)	RZ← ! (RX)
Transliteration results	Compile to 16-bit or 32-bit depending on the register range. Bit Commands. if (z<16), then not16 rz; else not32 rz, rz.	Compiles to the corresponding 16-bit or 32-bit instructions depending on the register range. if (x==z) and (z<16), then not16 rz; else not32 rz, rx.
clarification	Invert the value of RZ/RX by bit and store the result in RZ. Note that this instruction is a pseudo-instruction for nor rz, rz and nor rz, rx, rx.	
Impact Signs classifier for honorific people	unaffected	

User's Manual	
exceptions	not have

16-bit instructions	
manipulate	RZ \leftarrow ! (RZ)
grammatical	not16 rz
clarification	Invert the value of RZ by bit and store the result in RZ. Note that this instruction is a pseudo-instruction for nor16 rz, rz.
Affecting Flags	unaffected
exceptions	not have

16-bit instruction format:

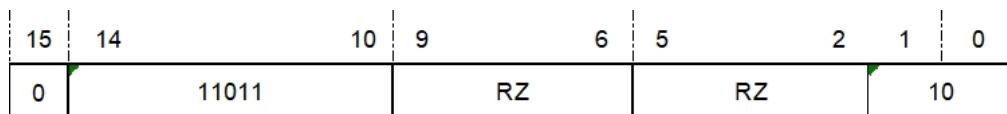


Figure 14.157: NOT-1

32-bit instruction	
manipulate	RZ← ! (RX)
grammatical	not32 rz, rx
instructions	Invert the value of RX by bit and store the result in RZ. Note that this instruction is a pseudo-instruction for nor32 rz, rx, rx.
Influence Flag Bit	unaffected
exceptions	no have

32-bit instruction format:

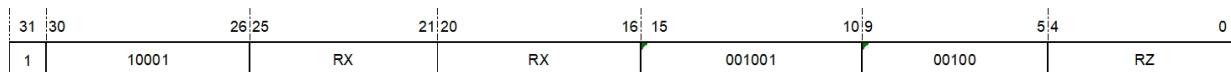


Figure 14.158: NOT-2

14.104 OR - Orders by Bit or

Harmonization Directive		
grammatical	or rz, rx	or rz, rx, ry
manipulate	RZ← RZ RX	RZ← RX RY
compliation results	Compile to 16-bit or 32-bit depending on the register range. Bit Commands. if (x<16) and (z<16), then or16 rz, rx ; if (x<16) and (z<16), then or16 rz, rx ; else or32 rz, rz, rx.	Compiles to the corresponding 16-bit or 32-bit instructions depending on the register range. if (y==z) and (x<16) and (z<16), then or16 rz, rx else or32 rz, rx, ry
clarification	Bitwise-orient RX to the value of RY/RZ and stores the result in RZ.	

Mar User's Ma kers of infl uen ce classi fier for hono rific peopl e	
except ions	not have

16-bit instructions	
manipulate	RZ← RZ RX
grammatical	or16 rz, rx
clarification	Bitwise-orient the values of RZ and RX and stores the result in RZ.
Affecting Flags	unaffected
limitation	The range of the register is r0-r15.
exceptions	not have

16-bit instruction format:

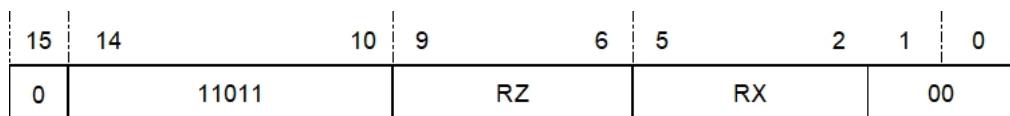


Figure 14.159: OR-I

32-bit instruction	
manipulate	RZ← RX RY
grammatical	or rz, rx, ry
clarification	Bitwise-orient the values of RX and RY and stores the result in RZ.
Affecting Flags	unaffected
exceptions	not have

32-bit instruction format:

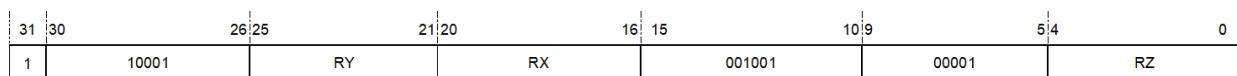


Figure 14.160: OR-2

14.105 ORI - immediate number by bit or instruction

Harmonization Directive	
grammatical	ori rz, rx, imm16
manipulate	RZ← RX zero_extend(IMM16)

User Compilation results	Only 32-bit instructions exist. ori32 rz, rx, imm16
clarification	Expands a 16-bit immediate zero to 32 bits and then performs a bitwise or operation with the value of RX, storing the result in RZ.
Affecting Flags	unaffected
limitation	The range of immediate numbers is 0x0-0xFFFF.
exceptions	not have

32-bit instruction	
manipulate	RZ ← RX zero_extend(IMM16)
Affecting Flags	unaffected
limitation	The range of immediate numbers is 0x0-0xFFFF.
grammatical	ori32 rz, rx, imm16
clarification	Expands a 16-bit immediate zero to 32 bits and then performs a bitwise or operation with the value of RX, storing the result in RZ.
exceptions	not have

32-bit instruction format:



Figure 14.161: ORI

14.106 POP - out of stack instruction

Harmonization Directive	
tell to Budd hist teac hing	pop reglist
m a ni p ul at e	<p>Loads consecutive multiple words from stack memory into a contiguous piece of register stack, then updates the stack registers to the top of stack memory and subroutines back;</p> <pre>dst ← {reglist}; addr ← SP; foreach (reglist){ Rdst← MEM[addr]; dst← next {reglist}; addr ← addr + 4; } sp← addr. PC← R15 & 0xffffffff.</pre>
C o m pi la ti o n re s ul ts	<p>Compiles to 16-bit or 32-bit instructions depending on register range.</p> <pre>if ({reglist}<16), then pop16 reglist; else pop32 reglist.</pre>
cl ar ifi ca ti o n	<p>Loading consecutive multiple words from stack memory into a contiguous piece of register stack, updating the stack pointer register, and then implementing the subroutine return function, i.e., the program jumps to the location specified by the link register R15, with the lowest bit of the link register ignored. Using the stack</p> <p>Register direct addressing mode.</p>
M ar k er s of in	unaffected

	User's Manual
discriminate constant	Unaligned Access Exception, Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Write Invalid Exception

16-bit instructions	
m a ni p ul at e	Loads consecutive multiple words from stack memory into a contiguous piece of register stack, then updates the stack registers to the top of stack memory and subroutines back. dst ← {reglist}; addr ← SP; foreach (reglist){ Rdst← MEM[addr]; dst← next {reglist}; addr ← addr + 4; } sp← addr. PC← R15 & 0xffffffff.
tell to Budd hist teac hing	pop16 reglist
cl ar ifi ca ti o n	Loading consecutive multiple words from stack memory into a contiguous piece of register stack, updating the stack pointer register, and then implementing the subroutine return function, i.e., the program jumps to the location specified by the link register R15, with the lowest bit of the link register ignored. Using the stack Pointer register direct addressing mode.
M ar k er s of in fl u e n ce classi fier for hono rific peop le	unaffected
set a limit	The range of the registers is r4 - r11, r15.

(on) User's Manual regul ate	
discr imin ate const ant	Unaligned Access Exception, Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Write Invalid Exception

16-bit instruction format:

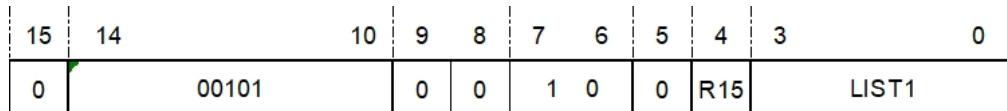


Figure 14.162: POP-1

LIST1 domain:

Specifies whether registers r4-r11 are in the register list.

0000:

r4-r11 not in register list

0001:

r4 in register list

User's Manual

0010:

r4-r5 in register list

0011:

r4-r6 in register list

.....

1000:

r4-r11 in register list

R15 domain:

Specifies whether register r15 is in the register list.

0:

r15 Not in register list

1:

r15 In register list

32-bit instruction	
manipulate	Load consecutive multiple words from stack memory into one contiguous piece of the register heap dst \leftarrow {reglist}; addr \leftarrow SP; foreach (reglist){ Rdst \leftarrow MEM[addr]; dst \leftarrow next {reglist}; addr \leftarrow addr + 4; } sp \leftarrow addr. PC \leftarrow R15 & 0xffffffff.
grammatical	pop32 reglist
clarification	Loading consecutive multiple words from stack memory into a contiguous piece of register stack, updating the stack pointer register, and then implementing the subroutine return function, i.e., the program jumps to the location specified by the link register R15, with the lowest bit of the link register ignored. Using the stack Pointer register direct addressing mode.
Markers of influence classifier for honorific	unaffected

User Manual	
limitation	The registers range from r4 - r11, r15, r16 - r17, r29.
exceptions	Unaligned Access Exception, Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Write Invalid Exception

32-bit instruction format:

LIST1 domain:

31	30	26 25	21 20	16 15	12	11	10	9	8	7	6	5	4	3	0
1		11010		11110		00000		0000	0	0	0	R28	LIST2	R15	LIST1

Figure 14.163: POP-2

Specifies whether registers r4-r11 are in the register list.

0000:

r4-r11 not in register list

0001:

r4 in register list

0010:

r4-r5 in register list

0011:

r4-r6 in register list

.....

1000:

r4-r11 in register list

R15 domain:

Specifies whether register r15 is in the register list.

0:

r15 Not in register list

1:

r15 In register list

LIST2 domain:

Specifies whether registers r16-r17 are in the register list.

000:

r16-r19 not in register list

001:

r16 in register list

010:

r16-r17 In the register list

R28 Domain:

Specifies whether register r28 is in the register list.

0:

r28 Not in register list

1:

r28 In register list

14.107 PSRCLR - PSR Bit Clear Instruction

Harmonization Directive	
gram matica l	psrclr ee, ie, fe, af Or the operands can be any combination of ee, ie, fe, and af.
manip ulate	Clears a bit or bits of the status register. $\text{PSR}(\{\text{EE}, \text{IE}, \text{FE}, \text{AF}\}) \leftarrow 0$
Trans lation in the end	Only 32-bit instructions exist. psrclr32 ee, ie, fe, af
be born in the year of (one of the 12 anima ls) Sex:	privileged instruction
clarifi cation	The selected PSR bit is cleared (1 means selected) The five-bit immediate number IMM5 is used to encode the control bit to be cleared and corresponds to the following: Immediate number IMM5 bits: corresponding PSR control bits Imm5[0]: AF Imm5[1]: FE Imm5[2]: IE Imm5[3]: EE Imm5 [4]: Reservations
Imp act Sign s classif	unaffected

User for honorific people	
exceptions	Privilege violation exceptions

32-bit instruction	
manip ulate	Clears one or more bits of the status register. $\text{PSR}(\{\text{EE}, \text{IE}, \text{FE}, \text{AF}\}) \leftarrow 0$
gram matica l	$\text{psrclr32 ee, ie, fe, af}$ Or the operands can be any combination of ee, ie, fe, and af.
be born in the year of (one of the 12 anima ls) Sex:	privileged instruction
clarifi cation	The selected PSR bit is cleared (1 means selected) The five-bit immediate number IMM5 is used to encode the control bit to be cleared and corresponds to the following: Immediate number IMM5 bits: corresponding PSR control bits Imm5[0]: AF Imm5[1]: FE Imm5[2]: IE Imm5[3]: EE Imm5 [4]: Reservations
Imp act Sign s classif ier for honor ific peopl e	unaffected
except ions	Privilege violation exceptions

32-bit instruction format:

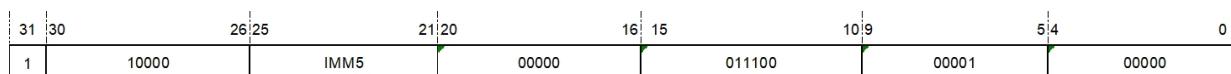


Figure 14.164: PSRCLR

14.108 PSRSET - PSR Position Bit Instruction

Harmonization Directive	
gram matica l	psrset ee, ie, fe, af Or the operands can be any combination of ee, ie, fe, and af.
manip ulate	Setting bits of the status register $\text{PSR}(\{\text{EE}, \text{IE}, \text{FE}, \text{AF}\}) \leftarrow 1$
Trans lation in the end	Only 32-bit instructions exist. <code>psrset32 ee, ie, fe, af</code>
be born in the year of (one of the 12 anima ls) Sex:	privileged instruction
clarifi cation	The selected PSR bit is set (1 for selected) The five-bit immediate number IMM5 is used to encode the control bit to be cleared with the following correspondence: Immediate number IMM5 bits: corresponding PSR control bits Imm5[0]: AF Imm5[1]: FE Imm5[2]: IE Imm5[3]: EE Imm5 [4]: Reservations
Imp act Sign s classif ier for honor ific peopl e	unaffected
except ions	Privilege violation exceptions

32-bit instruction	
manip ulate	Setting bits of the status register $\text{PSR}(\{\text{EE}, \text{IE}, \text{FE}, \text{AF}\}) \leftarrow 1$
gram matica 1	<code>psrset32 ee, ie, fe, af</code> Or the operands can be any combination of ee, ie, fe, and af.
be born in the year of (one of the 12 anima ls) Sex:	privileged instruction
clarifi cation	The selected PSR bit is set (1 for selected) The five-bit immediate number IMM5 is used to encode the control bit to be cleared with the following correspondence: Immediate number PSR control bits corresponding to IMM5 bits Imm5[0]: AF Imm5[1]: FE Imm5[2]: IE Imm5[3]: EE Imm5 [4]: Reservations
Imp act Sign s classif ier for honor ific peopl e	unaffected
except ions	Privilege violation exceptions

32-bit instruction format:



Figure 14.165: PSRSET

14.109 PUSH--Press Stack Instruction

Harmonization Directive	
gram matica 1	push reglist
manip ulate	<p>Stores the word in the register list into stack memory, then updates the stack register to the top of stack memory;</p> <pre>src ← {reglist}; addr ← SP; foreach (reglist){ MEM[addr] ← Rsrc. src ← next {reglist}; addr ← addr - 4. } sp← addr.</pre>
Translati on resul ts	<p>Compiles to 16-bit or 32-bit instructions depending on register range.</p> <pre>if ({reglist}<16), then push16 reglist; else push32 reglist.</pre>
instru ctions	<p>Store the words in the register list into the stack memory and then update the stack registers to the top of the stack memory. Using the stack Register direct addressing mode.</p>
Impact Sign s classifi er for honor ific peopl e	unaffected
except ions	Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, TLB write invalid exception

16-bit instructions	
manipulate	Storing words from the register list into stack memory src \leftarrow {reglist}; addr \leftarrow SP; foreach (reglist){ MEM[addr] \leftarrow Rsrc. src \leftarrow next {reglist}; addr \leftarrow addr - 4. } sp \leftarrow addr
grammatical	push16 reglist
clarification	Store the words in the register list into the stack memory and then update the stack registers to the top of the stack memory. Using the stack Stack register direct addressing mode.
impact Logo classifier for honorific people	unaffected
limitation	The range of the registers is r4 - r11, r15.
exceptions	Unaligned Access Exception, Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Write Invalid Exception

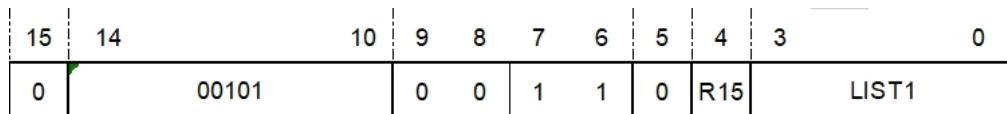
16-bit instruction format:

Figure 14.166: PUSH-1

LIST1 domain:

Specifies whether registers r4-r11 are in the register list.

0000:

r4-r11 not in register list

0001:

r4 in register list

r4-r5 in register list

0011:

r4-r6 in register list

.....

1000:

User's Manual

r4-r11 in register list

R15 domain:

Specifies whether register r15 is in the register list.

1:

r15 In register list

32-bit instruction	
manipulate	Load consecutive multiple words from stack memory into one contiguous piece of the register heap src \leftarrow {reglist}; addr \leftarrow SP; foreach (reglist) { MEM[addr] \leftarrow Rsrc. src \leftarrow next {reglist}; addr \leftarrow addr - 4. } sp \leftarrow addr
grammatical	push32 reglist
clarification	Store the words in the register list into the stack memory and then update the stack registers to the top of the stack memory. Using the stack Register direct addressing mode.
impact Logo classifier for honori fic people	unaffected
limitation	The registers range from r4 - r11, r15, r16 - r17, r29.
exceptions	Unaligned Access Exception, Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Write Invalid Exception

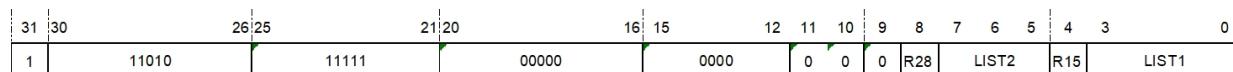
32-bit instruction format:

Figure 14.167: PUSH-2

LIST1 domain:

Specifies whether registers r4-r11 are in the register list.

r4-r11 not in register list

0001:

r4 in register list

0010:

User's Manual

r4-r5 in register list

0011:

r4-r6 in register list

.....

1000:

r4-r11 in register list

R15 domain:

Specifies whether register r15 is in the register list.

0:

r15 Not in register list

1:

r15 In register list

LIST2 domain:

Specifies whether registers r16-r17 are in the register list.

000:

r16-r19 not in register list

001:

r16 in register list

010:

r16-r17 In the register list

R29 Domain:

Specifies whether register r29 is in the register list.

0:

r29 Not in register list

1:

r29 In register list

14.110 REVB - Reverse Byte Order Instruction

Harmonization Directive	
grammatical	revb rz, rx
manipulate	rz[31:24]← rx[7:0]. rz[23:16]← rx[15:8]. rz[15:8]← rx[23:16]. rz[7:0]← rx[31:24].
Compilation results	Compiles to the corresponding 16-bit or 32-bit instructions depending on the register range. if (x<16) and (z<16), then revb16 rz, rx; else revb32 rz, rx.
clarification	The value of RX is inverted byte by byte, the bit order within each byte remains unchanged, and the result is stored in RZ.
Markers of influence classifier for honorific people	unaffected
exceptions	not have

16-bit instructions	
manipulate	rz[31:24]← rx[7:0]. rz[23:16]← rx[15:8]. rz[15:8]← rx[23:16]. rz[7:0]← rx[31:24].
grammatical	revb16 rz, rx
clarification	The value of RX is inverted byte by byte, the bit order within each byte remains unchanged, and the result is stored in RZ.
Impact Marker classifier for honorific people	unaffected
limitation	The range of the register is r0-r15.
exceptions	not have

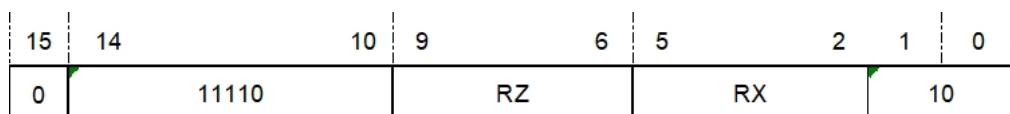


Figure 14.168: REV-B-1

32-bit instruction	
manipulate	rz[31:24]← rx[7:0]. rz[23:16]← rx[15:8]. rz[15:8]← rx[23:16]. rz[7:0]← rx[31:24].
grammatical	revb32 rz, rx
clarification	The value of RX is inverted byte by byte, the bit order within each byte remains unchanged, and the result is stored in RZ.
Impact Marker classifier for honorific people	unaffected
exceptions	not have

32-bit instruction format:

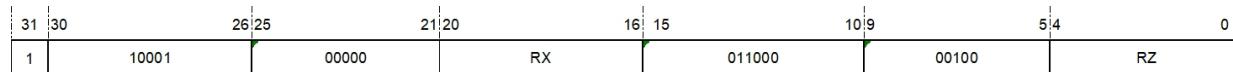


Figure 14.169: REVB-2

14.111 REVH - half-word byte inversion instruction

Harmonization Directive	
grammatical	revh rz, rx
manipulate	rz[31:24]← rx[23:16]. rz[23:16]← rx[31:24]. rz[15:8]← rx[7:0]. rz[7:0]← rx[15:8].
Compilations results	Compiles to the corresponding 16-bit or 32-bit instructions depending on the register range. if (x<16) and (z<16), then revh16 rz, rx; else revh32 rz, rx.
clarification	The value of RX is inverted byte by byte in the halfword, i.e., the two bytes in the high halfword and the two bytes in the low halfword are exchanged, and the value of RX is inverted by byte in the halfword. The order between the two halfwords and the bit order within each byte remains unchanged and the result is deposited into RZ.

User's Manual	
affect	affected
aspiration	not have

16-bit instructions	
manipulate	<code>rz[31:24]← rx[23:16].</code> <code>rz[23:16]← rx[31:24].</code> <code>rz[15:8]← rx[7:0].</code> <code>rz[7:0]← rx[15:8].</code>
grammatical	<code>revh16 rz, rx</code>
clarification	The value of RX is inverted byte by byte in the halfword, i.e., the two bytes in the high halfword and the two bytes in the low halfword are exchanged, and the value of RX is inverted by byte in the halfword. The order between the two halfwords and the bit order within each byte remains unchanged and the result is deposited into RZ.
affect aspiration	unaffected
limitations	The range of the register is r0-r15.
exceptions	not have

16-bit instruction format:

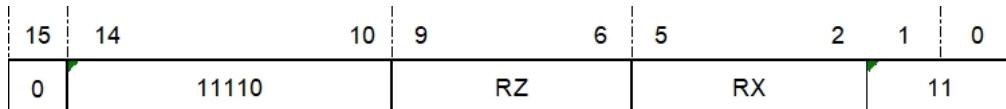


Figure 14.170: REVH-1

32-bit instruction	
manipulate	<code>rz[31:24]← rx[23:16].</code> <code>rz[23:16]← rx[31:24].</code> <code>rz[15:8]← rx[7:0].</code> <code>rz[7:0]← rx[15:8].</code>
grammatical	<code>revh32 rz, rx</code>
clarification	The value of RX is inverted byte by byte in the halfword, i.e., the two bytes in the high halfword and the two bytes in the low halfword are exchanged, and the value of RX is inverted by byte in the halfword. The order between the two halfwords and the bit order within each byte remains unchanged and the result is deposited into RZ.
affect aspiration	unaffected

Executive	not have
User Manual	ons

32-bit instruction format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1	10001	00000	RX	011000	01000	RZ

Figure 14.171: REVH-2

14.112 ROTL - circular left shift instruction

Harmonization Directive		
tell to Budd hist teaching	rotl rz, rx	rotl rz, rx, ry
drill (practice) writings or works	RZ← RZ<<< RX[5:0]	RZ← RX<<< RY[5:0]
C o m p i l a t i o n r e s u l t s	<p>Compile to 16-bit or 32-bit depending on the register range.</p> <p>Bit Commands.</p> <pre>if (x<16) and (z<16), then rotl16 rz, rx ;if (x<16) and (z<16), then rotl16 rz, rx ; else rotl32 rz, rz, rx.</pre>	<p>Compiles to the corresponding 16-bit or 32-bit instructions depending on the register range.</p> <pre>if (x==z) and (y<16) and (z<16), then rotl16 rz, ry else rotl32 rz, rx, ry</pre>
c l a r i f i c a t i o n	<p>For rotl rz, rx, the value of RZ is cyclically shifted left (original value is shifted left, right side is shifted into the bit shifted out from the left side), and the result is deposited into RZ, with the number of bits shifted left being determined by the value of the 6 lower bits of RX (RX[5:0]); if the value of RX[5:0] is greater than 31 then RZ is cleared to zero. For rotl rz, rx, ry, the value of RX is left-shifted cyclically (original value is left-shifted, the right side is shifted into the bit shifted out from the left side), and the result is stored in the RZ, the number of left shifts is determined by the value of the 6 bits below RY (RY[5:0]); if the value of RY[5:0] is greater than 31, then RZ is cleared to zero.</p>	
M a r k e r s o f i n f u e	unaffected	

User's Manual	
discriminate constant	not have

16-bit instructions	
manipulate	RZ ← RZ<<< RX[5:0]
grammatical	rotl16 rz, rx
clarification	The value of RZ is cyclically shifted left (the original value is shifted left, and the right side is shifted into the bit shifted out of the left side) and the result is deposited into RZ, and the number of bits shifted left is given by RX The value of the lower 6 bits (RX[5:0]) determines this; if the value of RX[5:0] is greater than 31, then RZ is cleared.
impact	unaffected
Logo classifier for honorific people	
limitation	The range of the register is r0-r15.
exceptions	not have

16-bit instruction format:

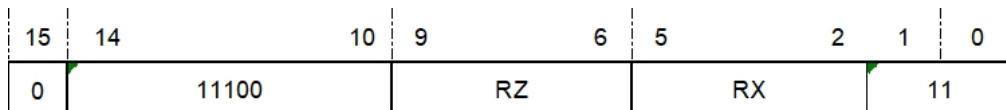


Figure 14.172: ROTL-1

32-bit instruction	
manipulate	RZ← RX<<< RY[5:0]
grammatical	rotl32 rz, rx, ry
clarification	The value of RX is cyclically shifted left (the original value is shifted left, and the right side is shifted into the bit shifted out of the left side) and the result is deposited into RZ, and the number of bits shifted left is determined by the RY The value of the lower 6 bits (RY[5:0]) determines this; if the value of RY[5:0] is greater than 31, then RZ is cleared to zero.
impact Logo classifier for honorific people	unaffected
exceptions	not have

32-bit instruction format:

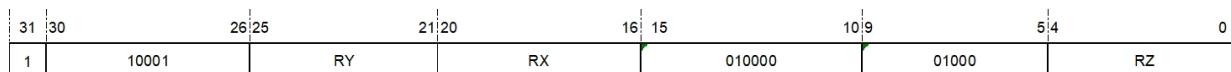


Figure 14.173: ROTL-2

14.113 ROTLI - Immediate Number Cyclic Left Shift Instruction

Harmonization Directive	
grammatical	rotli rz, rx, imm5
manip	RZ← RX<<< IMM5

User Manual	
Translation in the end	rotli32 rz, rx, imm5. The value of RX is shifted left cyclically (the original value is shifted left, and the right side is shifted into the bit shifted out of the left side), and the result is deposited into RZ, with the number of bits shifted left shifted from 5 to 5. The value of the Immediate Number of Bits (IMM5) determines this; if the value of IMM5 is equal to 0, then the value of RZ will be as RX.
Impact Logo classifier for honori fic people	unaffected
Limitation	The range of immediate numbers is 0-31.
Exceptions	not have

32-bit instruction	
manipulate	RZ← RX<<< IMM5
grammatical	rotli32 rz, rx, imm5
clarification	The value of RX is shifted left cyclically (the original value is shifted left, and the right side is shifted into the bit shifted out of the left side), and the result is deposited into RZ, with the number of bits shifted left shifted from 5 to 5. The value of the Immediate Number of Bits (IMM5) determines this; if the value of IMM5 is equal to 0, then the value of RZ will be as RX.
impact	unaffected
Logo classifier for honorific people	
limitation	The range of immediate numbers is 0-31.
exceptions	not have

32-bit instruction format:

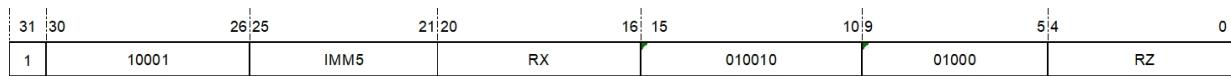


Figure 14.174: ROTLI

14.114 RSUB - Reverse Subtraction Instruction

Harmonization Directive	
grammatical	rsub rz, rx, ry
manipulate	RZ← RY - RX
Compilation results	Only 32-bit instructions exist. rsub32 rz, rx, ry
clarification	Subtract the RX value from the RY value and store the result RZ. Note that this instruction is a pseudo-instruction for subu rz, ry, rx.
Affecting Flags	unaffected

Exceptions	not have
------------	----------

32-bit instruction	
manipulate	RZ← RY - RX
grammatical	rsub32 rz, rx, ry
clarification	<p>Subtract the RX value from the RY value and store the result RZ.</p> <p>Note that this instruction is a pseudo-instruction for subu32 rz, ry, rx.</p>
Impact Marker classifier for honorific people	unaffected
exceptions	not have

32-bit instruction format:

	31 30	26 25	21 20	16 15	10 9	5 4	0
1	10001	RX	RY	000000	00100	RZ	

Figure 14.175: RSUB

14.115 RTS - subroutine return instruction

Harmonization Directive	
gram matica 1	rts
manip ulate	The program jumps to the location specified by the link registers $PC \leftarrow R15 \& 0xffffffff$
Transla tion in the end	Always compiles to 16-bit instructions. rts16
clarification	The program jumps to the location specified by link register R15, and the lowest bit of the link register is ignored. the jump range of the RTS16 instruction is the full 4GB address space. This instruction is used to implement the subroutine return function. Note that this instruction is a pseudo-instruction for <code>jmp r15</code> .
impact Logo classifi er for honori fic people	unaffected
except ions	not have

16-bit instructions	
manip ulate	The program jumps to the location specified by the link registers $PC \leftarrow R15 \& 0xffffffff$
gram matica 1	rts16

classifi cation	The program jumps to the location specified by link register R15, and the lowest bit of the link register is ignored. the jump range of the RTS16 instruction is the full 4GB address space. This instruction is used to implement the subroutine return function. Note that this instruction is a pseudo-instruction for jmp16 r15.
impa ct Logo classifi er for honori fic people	unaffected
except ions	not have

16-bit instruction format:

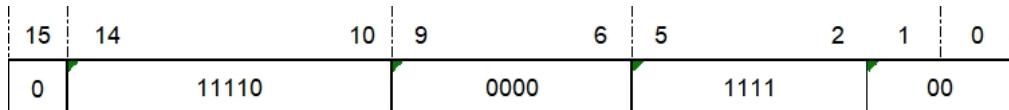


Figure 14.176: RTS-1

32-bit instruction	
manipulate	The program jumps to the location specified by the link registers PC← R15 & 0xffffffff
grammatical	rts32
clarification	The program jumps to the location specified by link register R15, and the lowest bit of the link register is ignored. the jump range of the RTS instruction is the entire 4GB address space. This instruction is used to implement the subroutine return function. Note that this instruction is a pseudo-instruction for jmp32 r15.
impact	unaffected
Logo classifier for honorific people	
exceptions	not have

32-bit instruction format:

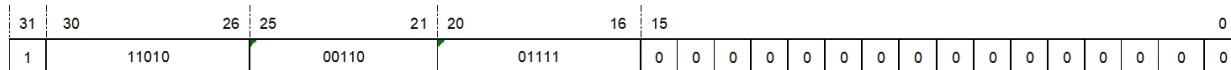


Figure 14.177: RTS-2

14.116 RTE - Exception and Normal Interrupt Return Instructions

Harmonization Directive	
grammatical	rte
manipulate	Exception and normal interrupt return PC ← EPC, PSR ← EPSR

User's Manual	
Compilation results	Only 32-bit instructions exist. rte32
Properties:	privileged instruction
instructions	The PC value is restored to the value stored in control register EPC, the PSR value is restored to value stored in EPSR, and the instruction is executed Start at the new PC address.
Marker of influence . classifier for honorific people	unaffected
exceptions	Privilege violation exceptions

32-bit instruction	
manipula te	Exception and normal interrupt return $PC \leftarrow EPC, PSR \leftarrow EPSR$
grammati cal	rte32
Attribute s:	privileged instruction
clarificati on	The PC value is restored to the value stored in control register EPC, the PSR value is restored to value stored in EPSR, and the instruction is executed Start at the new PC address.
Markers of influence classifier for honorific people	unaffected
exception s	Privilege violation exceptions

32-bit instruction format:

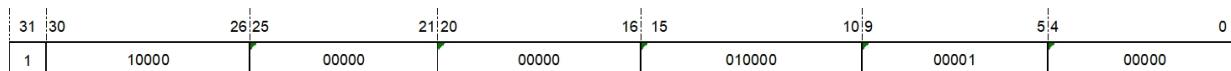


Figure 14.178: RTE

14.117 SCE - Conditional Execution Setup Command

Harmoniza tion Directive	
grammatica l	sce cond
manipulate	Sets the conditional execution bit for the next four instructions.
Compilatio n results	32-bit instructions only sce32 cond

Explanation:	<p>The sce instruction is used to set the conditional execution bits of the next four instructions. The operand COND is a 4-bit binary immediate number. The lowest bit indicates the conditional bit of the first instruction after the sce instruction, the second lowest bit indicates the conditional bit of the second instruction after the sce instruction... and so on. A condition bit of 1 means that C is executed normally at 1, and a condition bit of 0 means that C is executed normally at 0. If the C bit does not satisfy the condition bit, the bar The execution of the instruction does not have any effect. The value of the C bit used for judgment is based on the value at the time of execution of the sce instruction.</p>
Marks of influence Bit:	<p>If an exception or interrupt is generated by following four instructions, the condition execution bit is stored EPSR or FPSR.</p>
Restrictions:	<p>The instructions following the sce instruction can only be arithmetic instructions, multiply and divide instructions, byte, halfword, and word load store instructions in immediate number addressing mode.且 These instructions cannot affect the condition bit C. The operand is a 4-bit binary immediate number.</p>
discriminate Register:	<p>not have</p>
:	<p>For example, the instruction sequence is: sce 0101 mov r1, r0 mov r3, r2 mov r5, r4 mov r7, r6 The condition bit of the sce instruction is 0101, and if the C bit is 0 when the sce instruction is executed, the second and fourth mov instructions are satisfied. The first and third mov instructions do not satisfy the execution conditions and do not write the result to registers 3 and 7. into the destination register.</p>

32 in d e x (m at h.) hon orifi c title	
drill (prac tic e) Ma ke:	Sets the conditional execution bit for the next four instructions. <code>set_condition_execution(COND).</code>
tell to Law :	<code>sce32 cond</code>
Exp lana tion :	The sce instruction is used to set the conditional execution bits of the next four instructions. The operand COND is a 4-bit binary immediate number. The lowest bit indicates the conditional bit of the first instruction after the sce instruction, the second lowest bit indicates the conditional bit of the second instruction after the sce instruction... and so on. A condition bit of 1 means that C is executed normally at 1, and a condition bit of 0 means that C is executed normally at 0. If the C bit does not satisfy the condition bit, the bar The execution of the instruction does not have any effect. The value of the C bit used for judgment is based on the value at the time of execution of the sce instruction.
M ar k er s of in fl u e n ce Bit:	If an exception or interrupt is generated by following four instructions, the condition execution bit is stored EPSR or FPSR.

Best User's Manual	The instructions following the sce instruction can only be arithmetic instructions, multiply and divide instructions, byte, halfword, and word load store instructions in immediate number addressing mode. These instructions cannot affect the condition bit C. The operand is a 4-bit binary immediate number.
discriminate Register:	not have
:	<p>For example, the instruction sequence is: sce32 0101 mov32 r1, r0 mov32 r3, r2 mov32 r5, r4 mov32 r7, r6</p> <p>The condition bit of the sce instruction is 0101, and if the C bit is 0 when the sce instruction is executed, the second and fourth mov instructions are satisfied.</p> <p>The first and third mov instructions do not satisfy the execution conditions and do not write the result to registers 3 and 7.</p> <p>into the destination register.</p>

Command Format:

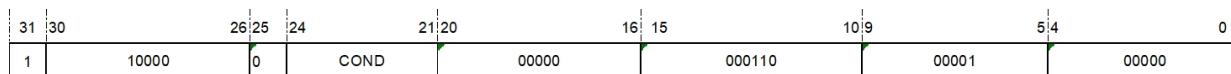


Figure 14.179: SCE

14.118 SEXT - Bit Extraction with Signed Extension Instruction

Harmonization Directive	
tell to Bud dhis t teac hing	sextrz, rx, msb, lsb
drill (pra ctice) writi ngs or wor ks	RZ ← sign_extend(RX[MSB:LSB])
c o m pi la ti o n re s ul t reso lute	Only 32-bit instructions exist. sext32 rz, rx, msb, lsb
cl ar ifi ca ti o n	Extracts a segment of consecutive bits (RX[MSB:LSB]) of RX specified by two 5-bit immediate numbers (MSB,LSB), sign-extends to 32 bits, and stores the result in RZ. If MSB equals to 31 and且 LSB equals to 0, the value of RZ is the same as RX. If MSB is equal to LSB, the value of RZ is the result of sign expansion by one bit of RX[MSB] (i.e., RX[LSB]). If MSB is less than LSB, the value of The behavior of the directive is unpredictable.

User's Manual	Manufactured
ar k er s of in fl u e n ce clas sifie r for hon orifi c peo ple	
set a limit (on) regu late	MSB ranges from 0-31, LSB ranges from 0-31, and <input checked="" type="checkbox"/> MSB should be greater than or equal to LSB.
disc rimi nate cons tant	not have

32-bit instruction	
drill (pra ctice) writi ngs or wor ks	RZ \leftarrow sign_extend(RX[MSB:LSB])
tell to Bud dhis t teac hing	sext32 rz, rx, msb, lsb
cl ar ifi ca ti o n	<p>Extracts a segment of consecutive bits (RX[MSB:LSB]) of RX specified by two 5-bit immediate numbers (MSB,LSB), sign-extends to 32 bits, and stores the result in RZ. If MSB equals to 31 and且 LSB equals to 0, the value of RZ is the same as RX. If MSB is equal to LSB, the value of RZ is the result of sign expansion by one bit of RX[MSB] (i.e., RX[LSB]). If MSB is less than LSB, the value of RZ is unpredictable.</p>
M ar k er s of in fl u e n ce clas sifie r for hon orifi c peo ple	unaffected

User limit (on) regulate	MSB ranges from 0-31, LSB ranges from 0-31, and MSB should be greater than or equal to LSB.
discriminate constant	not have

32-bit instruction format:



Figure 14.180: SEXT

MSB domain

Specifies the bit to be extracted to start with.

LSB domain

Specifies the bit that is to be extracted at the end of the extraction.

00000

0

00000

0 bit

00001

1

00001

1 position

.....

11111

31

11111

31st

14.119 SEXTB - Byte Extraction with Signed Extension Instruction

Harmonization Directive	
grammatical	sextb rz, rx
manipulate	RZ← sign_extend(RX[7:0]);
Compilation results	Compiles to the corresponding 16-bit or 32-bit instructions depending on the register range. if (z<16) and (x<16), then sextb16 rz, rx; else sextb32 rz, rx.
clarification	Extend the sign of the low byte of RX (RX[7:0]) to 32 bits and the result is present RZ.
Markers of influence classifier for honorific people	unaffected
exceptions	not have

16-bit instructions	
manipulate	RZ← sign_extend(RX[7:0]);
grammatical	sextb16 rz, rx
clarification	Extend the sign of the low byte of RX (RX[7:0]) to 32 bits and the result is present RZ.
Affecting Flags	unaffected
limitation	The range of the register is r0-r15.
exceptions	not have

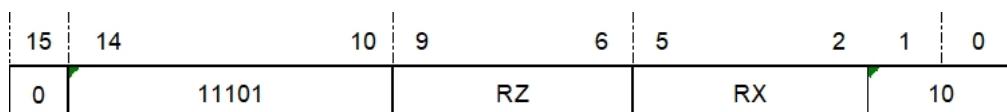


Figure 14.181: SEXTB-1

32-bit instruction	
manipula te	RZ ← sign_extend(RX[7:0]); sextb32 rz, rx
clarificati on	Extend the sign of the low byte of RX (RX[7:0]) to 32 bits and the result is present RZ. Note that this instruction is a pseudo-instruction for sext32 rz, rx, 0x7, 0x0.
Markers of influence classifier for honorific people	unaffected
exception s	not have

32-bit instruction format:

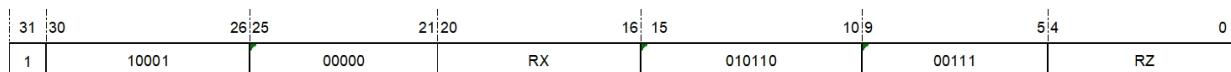


Figure 14.182: SEXTB-2

14.120 SEXTH - half-word extraction with signed extension instruction

Harmonization Directive	
grammati cal	sextb32 rz, rx
manipula te	RZ ← sign_extend(RX[15:0]);
Compilati on results	Compiles to the corresponding 16-bit or 32-bit instructions depending on the register range. if (z<16) and (x<16), then sexth16 rz, rx; else sexth32 rz, rx.
clarificati on	Extend the sign of the lower halfword of RX (RX[15:0]) to 32 bits and the result is stored RZ.
Markers of influence	unaffected

User Manual for honorific people	
exception s	not have

16-bit instructions	
manipulate	RZ← sign_extend(RX[15:0]);
grammatical	sexth16 rz, rx
clarification	Extend the sign of the lower halfword of RX (RX[15:0]) to 32 bits and the result is stored RZ.
Affecting Flags	unaffected
limitation	The range of the register is r0-r15.
exceptions	nohave

16-bit instruction format:

15	14	10	9	6	5	2	1	0
0		11101		RZ		RX		11

Figure 14.183: SEXTH-1

32-bit instruction	
manipula te	RZ← sign_extend(RX[15:0]);
grammat ical	sexth32 rz, rx
clarificati on	Extend the sign of the lower halfword of RX (RX[15:0]) to 32 bits and the result is stored RZ. Note that this instruction is a pseudo-instruction for sext32 rz, rx, 0x15, 0x0.
impact indicat or aspiratio n	unaffected
exceptio ns	not have

32-bit instruction format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1	10001	00000	RX	010110	01111	RZ

Figure 14.184: SEXTH-2

14.121 SRS.B - Byte Symbol Storage Instruction

Harmonization Directive	
tell	srs.b rz, [label]
to	
Bud	
dhist	
teac	
hing	
drill (pra ctice) writi ngs or wor ks	Stores the lowest byte symbol in the register to memory MEM[R28+ zero_extend(offset)] \leftarrow RZ[7:0]
c o m pi la ti o n re s ul t resol ute	Only 32-bit instructions exist. srs32.b rz, [label]
cl ar ifi ca ti o n	Stores the lowest byte symbol in register RZ to the label location. Addressing is by register plus unsigned immediate offset. The effective address of the memory is determined by adding the base address register RX plus an 18-bit relative offset unsigned to 32 bits. The SRS.B instruction can address +256KB of address space. Note that the offset DISP is the binary operand Offset.
M ar k er s of in	unaffected

fl User's Manual u e n ce class ifier for hono rific peop le	
discr imin ate cons tant	Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

32-bit instruction	
drill (pra ctice) writi ngs or wor ks	Stores the lowest byte symbol in the register to memory MEM[R28+ zero_extend(offset)] \leftarrow RZ[7:0]
tell to Bud dhist teac hing	srs32.b rz, [label]
cl ar ifi ca ti o n	Stores the lowest byte symbol in register RZ to the label location. Addressing is by register plus unsigned immediate offset. The effective address of the memory is determined by the base address register RX plus an 18-bit relative offset unsigned expanded to 32 bits. The SRS.B instruction can address +256KB of address space. Note that the offset DISP is the binary operand Offset.
M ar k er s	unaffected

User's Manual	
discriminate constant	Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

32-bit instruction format:

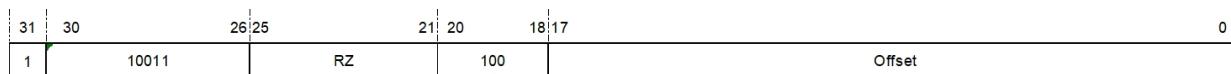


Figure 14.185: SRS.

14.122 SRS.H - Half-Word Symbol Storage Instruction

Harmonization Directive	
tell to Bud dhist teac hing	srs.h rz, [label]
drill (pra ctice) writi ngs or wor ks	Stores the lowest halfword symbol in the register into memory MEM[R28+ zero_extend(offset<< 1)]← RZ[7:0]
c o m pi la ti o n re s ul t resol ute	Only 32-bit instructions exist. srs32.h rz, [label]
cl ar ifi ca ti	Stores the lowest halfword symbol in register RZ to the label location. Addressing is by register plus unsigned immediate offset. The effective address of the memory is determined by adding the base address register RX to the base address register plus an 18-bit relative offset shifted one bit to the left, unsigned, to the 32-bit value. the SRS.H instruction addresses +512KB of address space.

User's Manual	Note: The offset DISP is the binary operand Offset.
M ar k er s of in fl u e n ce class ifier for hon orifi c peop le	unaffected
discr imin ate cons tant	Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

32-bit instruction	
drill (pra ctice) writi ngs or wor ks	Stores the lowest halfword symbol in the register into memory MEM[R28+ zero_extend(offset<< 1)] \leftarrow RZ[7:0]
tell to Bud dhist teac hing	srs32.h rz, [label]
cl ar ifi ca ti o n	Stores the lowest halfword symbol in register RZ to the label location. Addressing is by register plus unsigned immediate offset. The effective address of the memory is determined by adding the base address register R X to the base address register plus an 18-bit relative offset unsigned expanded by 1 bit to 32 bits. The SRS.H instruction addresses +512KB of address space. Note that the offset DISP is the binary operand Offset.
M ar k er s of in fl u e n ce class ifier for hon orifi c peop le	unaffected
discr imin	Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

32-bit instruction format:

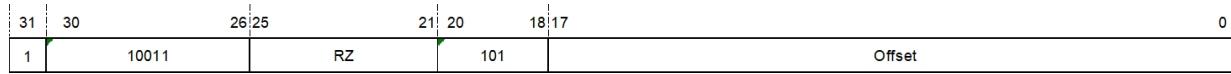


Figure 14.186: SRS.

14.123 SRS.W - Word Symbol Storage Instruction

Harmonization Directive	
tell to Bud dhist teac hing	srs.w rz, [label]
drill (pra ctice) writi ngs or wor ks	Stores the lowest word symbol in the register to memory MEM[R28+ zero_extend(offset<< 2)]← RZ[7:0]
c o m pi la ti o n re s ul t resol ute	Only 32-bit instructions exist. srs32.w rz, [label]
cl ar ifi ca ti o n	Stores the lowest word symbol in register RZ to the label location. Addressing is by register plus unsigned immediate offset. The effective address of the memory is determined by adding the base address register R X to the base address register plus an 18-bit relative offset shifted 2 bits to the left, unsigned, to 32 bits. the SRS.W instruction addresses +1024KB of address space. Note that the offset DISP is the binary operand Offset.
M ar k er s of in	unaffected

fl User's Manual u e n ce class ifier for hon orifi c peop le	
discr imin ate cons tant	Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

32-bit instruction	
drill (pra ctice) writi ngs or wor ks	Stores the lowest word symbol in the register to memory MEM[R28+ zero_extend(offset<< 2)]← RZ[7:0]
tell to Bud dhist teac hing	srs32.w rz, [label]
in st r u ct io n s	Stores the lowest word symbol in register RZ to the label location. Addressing is by register plus unsigned immediate offset. The effective address of the memory is determined by adding the base address register R X to the base address register plus an 18-bit relative offset shifted 2 bits to the left, unsigned, to 32 bits. the SRS.W instruction addresses +1024KB of address space. Note that the offset DISP is the binary operand Offset.

User's Manual	Manufactured ar k er s of in fl u e n ce class ifier for hon orifi c peop le
discriminate constant	Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

32-bit instruction format:

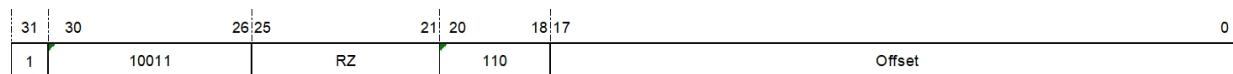


Figure 14.187: SRS.

14.124 ST.B - Byte Storage Instruction

Harmonization Directive	
tell to Budd hist teac hing	st.b rz, (rx, disp)
drill (prac tice) writi ngs or work s	Stores the lowest byte in the register into memory MEM[RX+ zero_extend(offset)]← RZ[7:0]
C o m pi la ti o n re s ul ts	Compiles to the corresponding 16-bit or 32-bit instructions depending on the range of offsets and registers. if (disp<32) and (x<7) and (z<7), then st16.b rz, (rx, disp) ; if (disp<32) and (x<7) and (z<7), then st16.b rz, (rx, disp) ; else st32.b rz, (rx, disp) ;
cl ar ifi ca ti o n	Stores the lowest byte in register RZ into memory. Addressing is by register plus unsigned immediate number offset. The effective address of the memory is determined by adding the base address register RX to the value of the 12-bit relative offset after unsigned expansion to 32 bits. The ST.B instruction can address +4KB of address space. Note that the offset DISP is the binary operand Offset.

User's Manual	
discriminate constant	Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Write Invalid Exception

16-bit instructions	
drill (practice) writings or works	Stores the lowest byte in the register into memory $MEM[RX + \text{zero_extend}(\text{offset})] \leftarrow RZ[7:0]$
tell to Budd hist teaching	st16.b rz, (rx, disp)
clarifi cation ti on	Stores the lowest byte in register RZ into memory. Addressing is by register plus unsigned immediate offset. The effective address of the memory is determined by adding the base address register RX to the value of the 5-bit relative offset unsigned extended to 32 bits. The ST16.B instruction can address +32B of space. Note that the offset DISP is the binary operand Offset.
Mark ers of influence n ce class ifier for honorific people	unaffected
set a limit (on) regular	The range of the register is r0-r7.

User's Manual	
discr imin ate const ant	Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

16-bit instruction format:

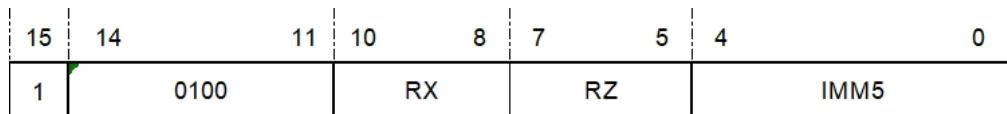


Figure 14.188: ST.B-1

32-bit instruction	
drill (prac tice) writi ngs or work s	Stores the lowest byte in the register into memory $\text{MEM}[\text{RX} + \text{zero_extend}(\text{offset})] \leftarrow \text{RZ}[7:0]$
tell to Budd hist teac hing	st32.b rz, (rx, disp)
cl ar ifi ca ti o n	Stores the lowest byte in register RZ into memory. The register plus unsigned immediate number offset addressing method is used. The effective address of the memory is determined by the base address register RX plus the value of the 12-bit relative offset after unsigned expansion to 32 bits. The ST32.B instruction can address +4KB of address space. Note that the offset DISP is the binary operand Offset.
M ar k er s of in fl u e n ce class ifier for hono rific peop le	unaffected
discr imin ate cons	Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Write Invalid Exception

32-bit instruction format:

31	30	26	25	21	20	16	15	12	11	0
1	10111		RZ		RX	0	0	0	0	Offset

Figure 14.189: ST.B-2

14.125 ST.H - Half-word storage instruction

Harmonization Directive	
gram matica 1	st.h rz, (rx, disp)
manip ulate	Stores the lowest byte in the register into memory MEM[RX+ zero_extend(offset<< 1)]← RZ[15:0]
Co mpi lati on resu lts	Compiles to the corresponding 16-bit or 32-bit instructions depending on the range of offsets and registers. if (disp<64)and(x<7)and(z<7), then st16.h rz, (rx, disp) ; else st32.h rz, (rx, disp) ;
instru ctions	Stores the lower halfword in register RZ into memory. Addressing is by register plus unsigned immediate offset. The effective address of the memory consists of the base address register RX plus a 12-bit relative offset unsigned expanded by 1 bit to 32 bits. The ST.H instruction addresses +8KB of address space.
Mar ker s of infl uen ce classi fier for hono rific peopl e	unaffected
except ions	Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, TLB write invalid exception

16-bit instructions	
drill (pra ctice) writi ngs or wor ks	Stores the low half word in the register into memory MEM[RX+ zero_extend(offset<< 1)]← RZ[15:0]

tell User's Manual to Bud dhist teac hing	st16_hrz, (rx, disp)
cl ar ifi ca ti o n	Stores the lower halfword in register RZ into memory. Addressing is by register plus unsigned immediate offset. The effective address of the memory is determined by adding the base address register RX to the base address register plus a 5-bit relative offset shifted 1 bit to the left, unsigned, to a 32-bit value. The ST16.H instruction addresses +64B of space. Note that the offset DISP is obtained by shifting binary operand Offset by 1 bit.
M ar k er s of in fl u e n ce class ifier for hon orifi c peop le	unaffected
set a limit (on) regu late	The range of the register is r0-r7.
discr imin ate cons tant	Unaligned Access Exception, Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

16-bit instruction format:

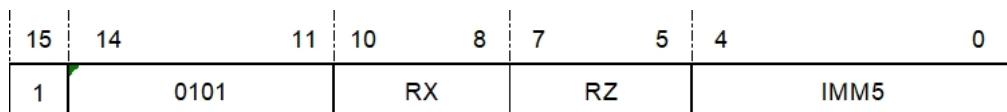


Figure 14.190: ST.H-1

32-bit instruction	
drill (pra ctice) writi ngs or wor ks	Stores the low half word in the register into memory MEM[RX+ zero_extend(offset<< 1)]← RZ[15:0]
tell to Bud dhist teac hing	st32.h rz, (rx, disp)
cl ar ifi ca ti o n	Stores the lower halfword in register RZ into memory. Addressing is by register plus unsigned immediate offset. The effective address of the memory is determined by adding the base address register RX to the base address register plus a 12-bit relative offset shifted one bit to the left, unsigned, to a 32-bit value. The ST32.H instruction addresses +8KB of address space. Note that the offset DISP is obtained by shifting binary operand Offset by 1 bit.
M ar k er s of in fl u e n ce class ifier for hon orifi	unaffected

User's Manual people	
discriminate constant	Unaligned Access Exception, Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Write Invalid Exception

32-bit instruction format:

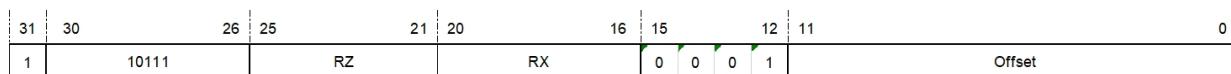


Figure 14.191: ST.H-2

14.126 ST.W - Word Storage Instruction

Harmonization Directive	
gram matica 1	st.w rz, (rx, disp)
manip ulate	Storing words from registers into memory MEM[RX+ zero_extend(offset<< 2)]← RZ[31:0]
Co mpi lati on resu lts	Compiles to the corresponding 16-bit or 32-bit instructions depending on the range of offsets and registers. if (x=sp) and (z<7) and (disp < 1024), st16.w rz, (sp, disp) ; else if (disp<128) and (x<7) and (z<7), st16.w rz, (rx, disp) ;- if (disp<128) and (x<7) and (z<7), st16. else st32.w rz, (rx, disp) ;
clarifi cation	Stores the word in register RZ into memory. Addressing is by register plus unsigned immediate offset. The effective address of the memory is determined by adding the base address register RX plus a 12-bit relative offset shifted two places to the left and extended unsigned to 32 bits. The ST.W instruction addresses +16KB of address space.
Mar ker s of infl uen ce classi fier for hono rific peopl e	unaffected
except ions	Unaligned Access Exception, Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Write Invalid Exception

16-bit instructions	
drill (practic e) writ ings or wor ks	Storing words from registers into memory MEM[RX+ zero_extend(offset<< 2)]← RZ[31:0]
tell to Bud dhis t teac hing	st16.w rz, (rx, disp) st16.w rz, (sp, disp)
cl ar ifi ca ti o n	Stores the word in register RZ into memory. Addressing is by register plus unsigned immediate offset. When rx= sp, the effective address of the memory is determined by the base address register RX plus an 8-bit relative offset shifted two bits to the left and extended unsigned to 32 bits. When rx is other registers, the effective address of memory is the base address register RX plus a 5-bit relative offset shifted two bits to the left, unsigned, to the value of 32 bits. the ST16.W instruction can address +1KB of space. Note that the offset DISP is obtained by shifting the binary operand IMM5 two bits to the left. base address register RX is SP, offset DISP is obtained by shifting binary operands {IMM3, IMM5} by two bits.
M ar k er s of in fl u e n ce clas sifie r for hon orifi c peo	unaffected

User's Manual	
set a limit (on) register	The range of the register is r0-r7.
discriminate constant	Unaligned Access Exception, Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Read Invalid Exception

16-bit instruction format:

st16.w rz, (rx, disp)

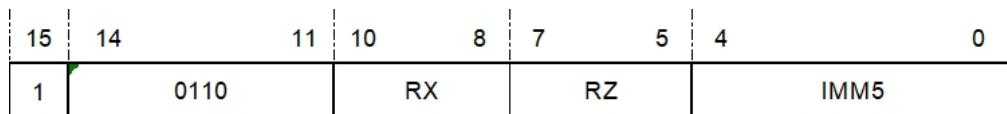


Figure 14.192: ST.W-1

st16.w rz, (sp, disp)

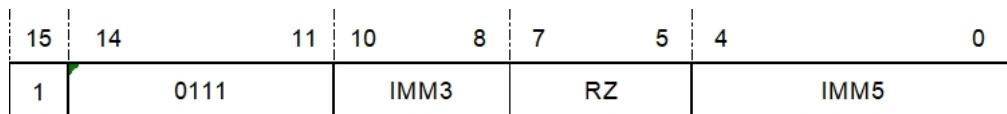


Figure 14.193: ST.W-2

32-bit instruction	
drill (pra ctice) writi ngs or wor ks	Storing words from registers into memory $\text{MEM}[\text{RX} + \text{zero_extend}(\text{offset} \ll 2)] \leftarrow \text{RZ}[31:0]$
tell to Bud dhist teac hing	<code>st32.w rz, (rx, disp)</code>
cl ar ifi ca ti o n	Stores the word in register RZ into memory. Addressing is by register plus unsigned immediate offset. The effective address of the memory is given by the base address register RX plus a 12-bit relative offset shifted two places to the left and unsigned to 32 bits. The ST32.W instruction addresses +16KB of address space. Note that the offset DISP is obtained by shifting the binary operand Offset two places to the left.
M ar k er s of in fl u e n ce class ifier for hon orifi c peop le	unaffected
discr imin	Unaligned Access Exception, Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Write Invalid Exception

32-bit instruction format:

31	30	26	25	21	20	16	15	12	11	0
1	10111		RZ		RX	0	0	1	0	Offset

Figure 14.194: ST.W-3

14.127 STM - Sequential Multi-Word Storage Instruction

Harmonization Directive	
tell	stm ry-rz, (rx)
to	
Bud	
dhis	
t	
teac	
hing	
m	Store the contents of a contiguous piece of the register stack sequentially to a contiguous piece of
a	memory address. src \leftarrow Y; addr \leftarrow RX;
ni	for (n= 0; n <=(Z-Y); n++){
p	MEM[addr] \leftarrow Rsrc; src
ul	\leftarrow src + 1;
at	addr \leftarrow addr+ 4;
e	}
c	Only 32-bit instructions exist.
o	stm32 ry-rz, (rx)
m	
pi	
la	
ti	
o	
n	
re	
s	
ul	
t	
reso	
lute	
cl	Store the contents of a contiguous stack of registers starting from RY to a contiguous piece of
ar	memory address, i.e., store the contents of register RY to the address of the first word at the
ifi	beginning of the specified address of the memory, store the contents of register RY+1 to the
ca	address of the second word at the beginning of the specified address of the memory, and so on,
ti	store the contents of register RZ to the address of the last word at the beginning of the specified
o	address of the memory, and so forth.
n	word on the address. The effective address of the memory is determined by the contents of the base address register RX.

User's Manual	Maintained
ar k er s of in fl u e n ce clas sifie r for hon orifi c peo ple	
set a limi t (on) regu late	RZ should be greater than or equal to RY.
disc rimi nate cons tant	Unaligned Access Exception, Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Write Invalid Exception

32-bit instruction	
m a ni p ul at e	The contents of a contiguous piece of register stack are stored sequentially to a contiguous piece of memory address. src← Y; addr← RX. for (n= 0; n<= IMM5; n++){ MEM[addr] ← Rsrc; src ← src + 1; addr← addr+ 4; }
tell to Bud dhis t teac hing	stm32 ry-rz, (rx)
cl ar ifi ca ti o n	Store the contents of a contiguous stack of registers starting from RY to a contiguous piece of memory address, i.e., store the contents of register RY to the address of the first word at the beginning of the specified address of the memory, store the contents of register RY+1 to the address of the second word at the beginning of the specified address of the memory, and so on, store the contents of register RZ to the address of the last word at the beginning of the specified address of the memory, and so forth. word on the address. The effective address of the memory is determined by the contents of the base address register RX.
M ar k er s of in fl u e n ce clas sifie r for hon orifi c peo ple	unaffected

User's Manual	R should be greater than or equal to RY.
disc rimi nate cons tant	Unaligned Access Exception, Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Write Invalid Exception

32-bit instruction format:

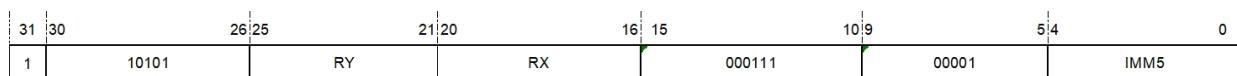


Figure 14.195: STM

IMM5 domain:

Specifies the number of target registers, IMM5 = Z - Y.

0000:

1 destination register

00001:

2 destination registers

.....

1111:

32 destination registers

14.128 STOP - enter low power pause mode command

Harmonization Directive	
grammatical	stop
manipulate	Enter low-power pause mode
Translation resolute	Only 32-bit instructions exist. stop32
clarification	This instruction puts the processor into low-power mode and waits for an interrupt to exit this mode. At this point, the CPU clock stops and the Most of the peripherals were also stopped.
impact indicator aspiration	unaffected
exceptions	Privilege violation exceptions

32-bit instruction	
manipulate	Enter low-power pause mode
grammatical	stop32
Attributes:	privileged instruction
clarification	This instruction puts the processor into low-power mode and waits for an interrupt to exit this mode. At this point, the CPU clock stops and the Most of the peripherals were also stopped.
impact indicator aspiration	unaffected
exceptions	Privilege violation exceptions

32-bit instruction format:

	31 30	26 25	21 20	16 15	10 9	5 4	0
1	10000	00000	00000	010010	00001	00000	

Figure 14.196: STOP

14.129 STQ - sequential four-word store instruction

Harmonization Directive	
tell to Bud dhis t teac hing	stq r4-r7, (rx)
m a ni p ul at e	<p>Store the words in registers R 4-R7 sequentially to a contiguous piece of memory address.</p> <p>src← 4; addr← RX.</p> <pre>for (n = 0; n <= 3; n++) { MEM[addr] ← Rsrc. src← src+ 1; addr← addr+ 4; }</pre>
c o m pi la ti o n re s ul t reso lute	<p>Only 32-bit instructions exist.</p> <p>stq32 r4-r7, (rx).</p>
cl ar ifi ca ti o n	<p>The words in the register stack [R4,R7] (including boundaries) are stored sequentially to a contiguous memory address, i.e., the contents of register R4 are stored to the address of the first word at the beginning of the memory specified address, the contents of register R5 are stored to the address of the second word at the beginning of the memory specified address, the contents of register R6 are stored to the address of the third word at the beginning of the memory specified address, and the contents of register R7 are stored to the address of the fourth word at the beginning of the memory specified address. The contents of register R7 are stored at the address of the fourth word of the beginning of the specified address of the memory. The effective address of the memory is determined by the contents of base address register RX.</p> <p>Note that this instruction is a pseudo-instruction for stm r4-r7, (rx).</p>
M ar k er s	unaffected

User's Manual	
discriminate constant	Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, TLB write invalid exception

32-bit instruction	
m a ni p ul at e	Store the words in registers R 4-R7 sequentially to a contiguous piece of memory address. src← 4; addr← RX. for (n = 0; n <= 3; n++){ MEM[addr] ← Rsrc. src← src+ 1; addr← addr+ 4; }
tell to Bud dhis t teac hing	stq32 r4-r7, (rx)
cl ar ifi ca ti o n	The words in the register stack [R4,R7] (including the boundaries) are stored sequentially to a contiguous memory address, i.e., the content of register R4 is stored to the address of the first word at the beginning of the specified address of the memory, the content of register R5 is stored to the address of the second word at the beginning of the specified address of the memory, the content of register R6 is stored to the address of the third word at the beginning of the specified address of the memory, and the content of register R7 is stored to the address of the fourth word at the beginning of the specified address of the memory. The contents of register R7 are stored to the address of the fourth word of the beginning of the specified address of the memory. The effective address of the memory is determined by the contents of base address register RX. Note that this instruction is a pseudo-instruction for stm r4-r7, (rx).
M ar k er s of in fl u e n ce clas sifie r for hon orifi c peo ple	unaffected

disc	Memory Management	Unaligned Access Exception, Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch
rimi		Exception, TLB Write Invalid Exception
nate		
cons		
tant		

32-bit instruction format:

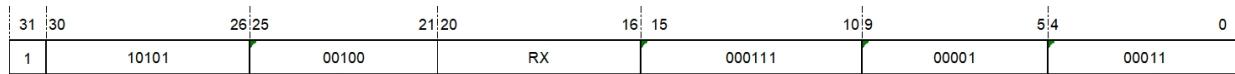


Figure 14.197: STQ

14.130 STR.B - Register Shift Addressing Byte Store Instruction

Harmonization Directive	
gram matica 1	str.b rz, (rx, ry << 0) str.b rz, (rx, ry << 1) str.b rz, (rx, ry<< 2) str.b rz, (rx, ry<< 3)
manip ulate	Stores the lowest byte in the register into memory MEM[RX+ RY<< IMM2]← RZ[7:0]
Translati on resul ts	Only 32-bit instructions exist. str32.b rz, (rx, ry<< 0) str32.b rz, (rx, ry<< 1) str32.b rz, (rx, ry<< 2) str32.b rz, (rx, ry<< 3)
clarifi cation	Stores the lowest byte in register RZ into memory. The register plus register shift addressing method is used. The effective address of the memory is determined by adding the base address register RX to the offset register RY and shifting the value of IMM2 by 2 immediate bits to left, and the value IMM2 to the absence of IMM2. The default value is 0.
Impact Sign s classif ier for honor ific peopl e	unaffected
except ions	Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Write Invalid Exception

32-bit instruction	
manip ulate	Stores the lowest byte in the register into memory MEM[RX+ RY<< IMM2]← RZ[7:0]
gram matica 1	str32.b rz, (rx, ry<< 0) str32.b rz, (rx, ry<< 1) str32.b rz, (rx, ry<< 2) str32.b rz, (rx, ry<< 3)

User Manual classification	Stores the lowest byte in register RZ into memory. The register plus register shift addressing method is used. The effective address of the memory is determined by adding the base address register RX to the offset register RY and shifting the value of IMM2 by 2 immediate bits to left, and the value IMM2 to the absence of IMM2. The default value is 0.
Impact Signs classifier for honorific people	unaffected
exceptions	Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Write Invalid Exception

32-bit instruction format:

str32.b rz, (rx, ry<< 0)

31 30	26 25	21 20	16 15	10 9	5 4	0
1	10101	RY	RX	000000	00001	RZ

Figure 14.198: STR.

str32.b rz, (rx, ry<< 1)

31 30	26 25	21 20	16 15	10 9	5 4	0
1	10101	RY	RX	000000	00010	RZ

Figure 14.199: STR.

str32.b rz, (rx, ry<< 2)

31 30	26 25	21 20	16 15	10 9	5 4	0
1	10101	RY	RX	000000	00100	RZ

Figure 14.200: STR.

str32.b rz, (rx, ry<< 3)

14.131 STR.H--Register Shift Addressing Half-Word Storage Instructions

Harmonization Directive	
gram matica 1	str.h rz, (rx, ry << 0) str.h rz, (rx, ry << 1) str.h rz, (rx, ry<< 2) str.h rz, (rx, ry<< 3)
manip ulate	Stores the low half word in the register into memory MEM[RX+ RY<< IMM2]← RZ[15:0]
Translati on resul ts	Only 32-bit instructions exist. str32.h rz, (rx, ry<< 0) str32.h rz, (rx, ry<< 1) str32.h rz, (rx, ry<< 2) str32.h rz, (rx, ry<< 3)
clarifi cation	Stores the low halfword in register RZ into memory. The register plus register shift addressing method is used. The effective address of the memory is determined by adding the base address register RX to the offset register RY and shifting the value of IMM2 by 2 bits. The value is 0.
Imp act Sign s classif	unaffected

User Manual for honorific people	
except ions	Unaligned Access Exception, Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Write Invalid Exception

31 30	26 25	21 20	16 15	10 9	5 4	0
1	10101	RY	RX	000000	01000	RZ

Figure 14.201: STR.B-4

32-bit instruction	
manipulate	Stores the low half word in the register into memory MEM[RX+ RY<< IMM2]← RZ[15:0]
grammatical	str32.h rz, (rx, ry<< 0) str32.h rz, (rx, ry<< 1) str32.h rz, (rx, ry<< 2) str32.h rz, (rx, ry<< 3)
clarification	Stores the low halfword in register RZ into memory. The register plus register shift addressing method is used. The effective address of the memory is determined by adding the base address register RX to the offset register RY and shifting the value of IMM2 by 2 bits. The value is 0.
Impact Signs classifier for honorific people	unaffected
exceptions	Unaligned Access Exception, Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Write Invalid Exception

32-bit instruction format:

str32.h rz, (rx, ry<< 0)

31 30	26 25	21 20	16 15	10 9	5 4	0
1	10101	RY	RX	000001	00001	RZ

Figure 14.202: STR.H-1

str32.h rz, (rx, ry<< 1)

31 30	26 25	21 20	16 15	10 9	5 4	0
1	10101	RY	RX	000001	00010	RZ

Figure 14.203: STR.H-2

str32.h rz, (rx, ry<< 2)

str32.h rz, (rx, ry<< 3)

31 30	26 25	21 20	16 15	10 9	5 4	0
1	10101	RY	RX	000001	00100	RZ

Figure 14.204: STR.H-3

31 30	26 25	21 20	16 15	10 9	5 4	0
1	10101	RY	RX	000001	01000	RZ

Figure 14.205: STR.H-4

14.132 STR.W - Register Shift Addressing Word Store Instruction

Harmonization Directive	
gram matica 1	str.w rz, (rx, ry<< 0) str.w , (rx, ry<< 1) str.w , (rx, ry<< 2) str.w rz, (rx, ry<< 3)
manip ulate	Storing words from registers into memory MEM[RX+ RY<< IMM2]← RZ[31:0]
Tran slati on resul ts	Only 32-bit instructions exist. str32.w rz, (rx, ry<< 0) str32.w , (rx, ry<< 1) str32.w rz, (rx, ry<< 2) str32.w rz, (rx, ry<< 3)
instr uctions	Stores the word in register RZ into memory. The register plus register shift addressing method is used. The effective address of the memory is determined by adding the base address register RX to offset register RY and shifting it by 2 bits to the left by the immediate number IMM2. The default value of IMM2 is 0.
Imp act Sign s classifi er for honori fic people	unaffected
except ions	Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, TLB write invalid exception

32-bit instruction	
manipulate	Storing words from registers into memory MEM[RX+ RY<< IMM2]← RZ[31:0]
grammatical	str32.w rz, (rx, ry<< 0) str32.w , (rx, ry<< 1) str32.w rz, (rx, ry<< 2) str32.w rz, (rx, ry<< 3)
clarification	Stores the word in register RZ into memory. The register plus register shift addressing method is used. The effective address of the memory is determined by adding the base address register RX to offset register RY and shifting it by 2 bits to the left by the immediate number IMM2. The default value of IMM2 is 0.
Impact Signs classifier for honorific people	unaffected
exceptions	Unaligned Access Exception, Access Error Exception, TLB Unrecoverable Exception, TLB Mismatch Exception, TLB Write Invalid Exception

32-bit instruction format:

str32.w rz, (rx, ry<< 0)

31	30	26 25	21 20	16 15	10 9	5 4	0
1	10101	RY	RX	000010	00001	RZ	

Figure 14.206: STR.W-1

str32.w rz, (rx, ry<< 1)

31	30	26 25	21 20	16 15	10 9	5 4	0
1	10101	RY	RX	000010	00010	RZ	

Figure 14.207: STR.W-2

str32.w rz, (rx, ry<< 2)

31	30	26 25	21 20	16 15	10 9	5 4	0
1	10101	RY	RX	000010	00100	RZ	

Figure 14.208: STR.W-3

str32.w rz, (rx, ry<< 3)

31 30	26 25	21 20	16 15	10 9	5 4	0
1	10101	RY	RX	000010	01000	RZ

Figure 14.209: STR.W-4

14.133 SUBC - unsigned with debit subtraction instruction

Harmonization Directive		
tell to Budd hist teaching	subc rz, rx	subc rz, rx, ry
drill (practice) writings or works	RZ← RZ - RX - (!C) C ← Secondment	RZ← RX - RY - (!C) C ← Secondment
commands	Compile to 16-bit or 32-bit depending on the register range. Bit Commands. if (x<16) and (z<16), then subc16 rz, rx; else subc32 rz, rx, ry.	Compiles to the corresponding 16-bit or 32-bit instructions depending on the register range. if (x==z) and (y<16) and (z<16), then subc16 rz, ry; else subc32 rz, rx, ry.
clarification	For subc rz, rx, subtract the value of RZ from the value of register RX and the non-value of bit C. For subcrz, rx, ry, subtract the value of RX from value of register RY and the non-value of bit C. The result is stored in RZ and the borrowed bit is stored in the C bit. For this subtraction instruction If a borrow occurs, the C bit will be cleared, and vice versa, the C bit will be set.	
Markers	C ← Secondment	

in	User's Manual
f	
u	
e	
n	
ce	
classi	
fier	
for	
hono	
rific	
peop	
le	
discr	not have
mina	
te	
const	
ant	

16-bit instructions	
manipu late	$RZ \leftarrow RZ - RX - (!C)$ C \leftarrow Borrowed Bit
gramm atical	subc16 rz, rx
clarific ation	The value of RZ is subtracted from the value of register RX and the nonvalue of bit C, and the result is present in RZ and the borrowed bit is present in bit C. The value of RZ is subtracted from the value of register RX and the nonvalue of bit C. For this subtraction If a borrow instruction occurs, the C bit is cleared and the C bit is set.
affect aspirati on	C \leftarrow Secondment
limitati on	The range of the register is r0-r15.
excepti ons	not have

16-bit instruction format:

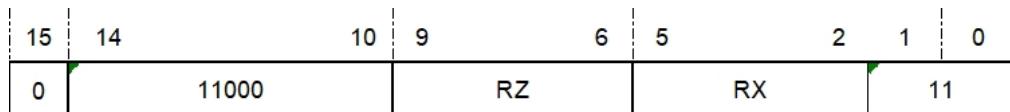


Figure 14.210: SUBC-1

32-bit instruction	
manipulate	RZ \leftarrow RX - RY - (!C) C \leftarrow Borrowed Bit
grammatical	subc32 rz, rx, ry
clarification	The value of RX is subtracted from the value of register RY and the nonvalue of bit C, and the result is stored in RZ and the borrowed bit is stored in C. For this subtraction If a borrow instruction occurs, the C bit is cleared and the C bit is set.
affect aspiration	C \leftarrow Secondment
exceptions	not have

32-bit instruction format:



Figure 14.211: SUBC-2

14.134 SUBI - Unsigned Immediate Number Subtraction Instruction

Harmonization Directive		
tell to Bud dhis t teac hing	ubi rz, oimm12	subi rz, rx, oimm12
S drill (pra ctic e) writ ings or wor ks	RZ← RZ - zero_extend(OIMM12)	RZ← RX - zero_extend(OIMM12)
C o m pi la ti o n re s ul ts	Compile to the corresponding 16 according to the range of the registers. Bit or 32-bit instructions. if (oimm12<257) and (z<8), then subi16 rz, oimm8. else subi32 rz, rz, oimm12.	Compiles to the corresponding 16-bit or 32-bit instructions depending on the register range. if (oimm12<8) and (z<8) and (x<8), then subi16 rz, rx, oimm3; elseif (x==z) and (z<8) and (oimm12<257), then subi16 rz, oimm8; else subi32 rz, rx, oimm12.
pers uad e gen eric ter m for a sacr ifice	Extend the 12-bit immediate number with bias 1 (OIMM12) zero to 32 bits, then subtract that 32-bit number from the value of RZ/RX to put the The result is deposited in the RZ.	

User's Manual god	
M ar k er s of in fl u e n ce clas sifie r for hon orifi c peo ple	unaffected
set a limi t (on) regu late	The range of immediate numbers is 0x1-0x1000.
disc rimi nate cons tant	not have

16-bit instruction 1	
manipu late	RZ← RZ - zero_extend(OIMM8)
gramm atical	subi16 rz, oimm8
clarific ation	Extends an 8-bit immediate number (OIMM8) zero with a bias of 1 to 32 bits, then subtracts that 32-bit number from the value of RZ and stores the result in RZ. Note: The binary operand IMM8 is equal to OIMM8 - 1.
affect aspirati on	unaffected
limitati	The range of registers is r0-r7; the range of immediate numbers is 1-256.

User's Manual	
excepti ons	not have

16-bit instruction format 1:

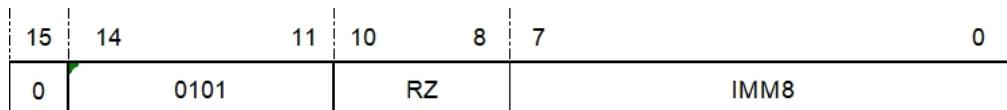


Figure 14.212: SUBI-1

IMM8 domain:

Specifies a value without a bias immediate number.

Note: The register subtraction value OIMM8 is biased by 1 compared to the binary operand IMM8.

00000000:

Less 1

00000001:

Less 2

.....

11111111:

minus 256

16-bit instructions 2	
manipulate	RZ \leftarrow RX - zero_extend(OIMM3)
grammatical	subi16 rz, rx, oimm3
clarification	Extends a 3-bit immediate number (OIMM3) zero with a bias of 1 to 32 bits, then subtracts that 32-bit number from the value of RX and stores the result in RZ. Note: The binary operand IMM3 is equal to OIMM3 - 1.
affect aspiration	unaffected
limitation	The range of registers is r0-r7; the range of immediate numbers is 1-8.
exceptions	not have

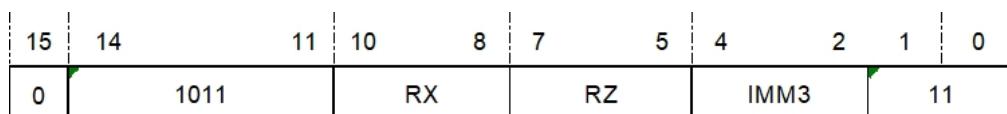
16-bit instruction format 2:

Figure 14.213: SUBI-2

IMM3 domain

Specifies a value without a bias immediate number.

000

Less 1

001

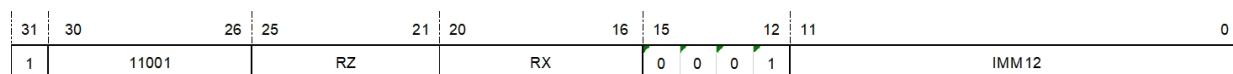
Less 2

.....

111

Less 8

32-bit instruction	
manipulate	RZ← RX - zero_extend(OIMM12)
grammatical	subi32 rz, rx, oimm12
clarification	Extends a 12-bit immediate number (OIMM12) zero with a bias of 1 to 32 bits, then subtracts that 32-bit number from the value of RX and stores the result in RZ. Note: The binary operand IMM12 is equal to OIMM12 - 1.
affect aspiration	unaffected
limitation	The range of immediate numbers is 0x1-0x1000.
exceptions	not have

32-bit instruction format:**IMM12 domain**

Specifies a value without a bias immediate number.

Note: The register subtracted value OIMM12 needs to be biased by 1 compared to the binary operand IMM12.

000000000000

Less 0x1

000000000001

Less 0x2

.....

111111111111

minus 0x1000

14.135 SUBI(SP) - unsigned (stack pointer) immediate number subtraction instruction

Harmonization Directive	
grammatical	subi sp, sp, imm
manipulate	$SP \leftarrow SP - \text{zero_extend(IMM)}$
Compilation results	Only 16-bit instructions exist. subi sp, sp, imm
clarification	Extend the immediate number (IMM) zero to 32 bits and shift it left by 2 bits, then subtract it from the value of the stack pointer (SP) and store the result in the Stack Pointer (SP).
Markers of influence classifier for honorific people	unaffected
limitation	The range of immediate numbers is 0x0-0x1fc.
exceptions	not have

16-bit instructions	
manipulate	$SP \leftarrow SP - \text{zero_extend(IMM)}$
grammatical	subi sp, sp, imm
clarification	Extend the Immediate Number (IMM) zero to 32 bits and shift it 2 bits to the left, then subtract it from the value of the Stack Pointer (SP) and store the result into the Stack Pointer. Note: Immediate number (IMM) is equal to the binary operand {IMM2, IMM5} << 2.
impact Logo classifier for honorific people	unaffected
limitation	The source and destination registers are both stack instruction registers (R14) the range of

Users Manual	immediate numbers is (0x0-0x7f) << 2.
except ions	not have

16-bit instruction format:

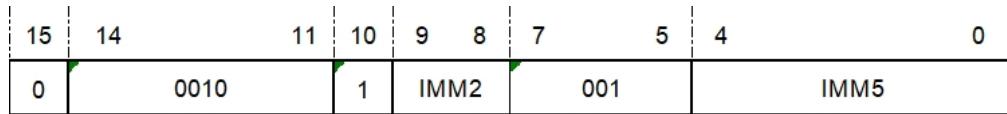


Figure 14.214: SUBI(SP)

IMM domain:

Specifies the value of an immediate number without a shift.

Note: The value IMM added to the register needs to be shifted 2 bits to the left compared to the binary operands {IMM2, IMM5}.

{00, 0000}

Minus 0x0

{00, 00001}

Less 0x4

.....

{11, 11111}

Less 0xfc

14.136 SUBU - unsigned subtraction instruction

Harmonization Directive		
tell to Bud dhis t teac hing	subu rz, rx sub rz, rx	subu rz, rx, ry
drill (pra ctice) writi ngs or wor ks	RZ← RZ - RX	RZ← RX - RY
C o m pi la ti o n re s ul ts	Compile to the corresponding 16 according to the range of the registers. Bit or 32-bit instructions. if (z<16) and (x<16), then subu16 rz, rx; else subu32 rz, rx, ry.	Compiles to the corresponding 16-bit or 32-bit instructions depending on the register range. if (z<8) and (x<8) and (y<8), then subu16 rz, rx, ry; elseif (x==z) and (z<16) and (y<16), then subu16 rz, ry; else subu32 rz, rx, ry.
pers uad e gene ric ter	For subu rz, rx, subtract the RX from the RZ value and store the result RZ. For subu rz, rx, ry, subtract the RY from the RX value and store the result RZ.	

User's Manual for a sacrifice the gods	
M ar k er s of in fl u e n ce class ifier for hon orifi c peo ple	unaffected
disc rimi nate cons tant	not have

16-bit instruction 1	
manipulate	RZ← RZ - RX
grammatical	subu16 rz, rx sub16 rz, rx
clarification	Subtracts the RX value from the RZ value and stores the result in RZ.
Affecting Flags	unaffected
limitation	The range of the register is r0-r15.
exceptions	not have

16-bit instruction format 1:

15	14	10	9	6	5	2	1	0
0	11000		RZ		RX		10	

Figure 14.215: SUBU-1

16-bit instructions 2	
manipulate	$RZ \leftarrow RX - RY$
grammatical	subu16 rz, rx, ry
clarification	subu16 rz, rx, ry
Affecting Flags	Subtract the RY value from the RX value and store the result RZ.
limitation	unaffected
exceptions	The range of the register is r0-r7.

16-bit instruction format 2:

15	14	11	10	8	7	5	4	2	1	0
0	1011		RX		RZ		RY		01	

Figure 14.216: SUBU-2

32-bit instruction	
manipulate	$RZ \leftarrow RX - RY$
grammatical	subu32 rz, rx, ry
clarification	subu32 rz, rx, ry
Affecting Flags	Subtract the RY value from the RX value and store the result RZ.
exceptions	unaffected

32-bit instruction format:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001		RY		RX		00000		00100		RZ	

Figure 14.217: SUBU-3

14.137 SYNC - CPU synchronization instruction

Harmonization Directive	
grammatical	sync.is sync.i sync.s sync
manipulate	This directive guarantees that all preceding instructions will retire earlier than this directive and all following instructions will retire later than this directive.
Compilation results	Only 32-bit instructions exist. sync32.is sync32.i sync32.s sync32
clarification	S characterizes whether to broadcast outside the CPU core," 1" is valid. I Characterizes whether or not the instruction is synchronized with the value when retiring to perform the pipeline clearing operation," 1" is valid.
Markers of influence classifier for honorific people	unaffected
exception	not have

32-bit instruction	
manipulate	synchronize CPUs
grammatical	sync32
clarification	S characterizes whether to broadcast outside the CPU core," 1" is valid. I Characterizes whether or not the instruction is synchronized with the value when retiring to perform the pipeline clearing operation," 1" is valid.
Markers of influence classifier for honorific people	unaffected

exception	not have
UserManuals	

32-bit instruction format:

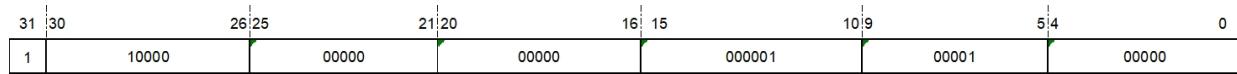


Figure 14.218: SYNC

14.138 TRAP - Operating System Trap Instruction

Harmonization Directive	
grammatical	trap 0. trap 1 Trap 2. trap 3
manipulate	Causes trap anomalies to occur
Description of compilation results	Only 32-bit instructions exist. trap32 0, trap32 1 trap32 2. trap32 3 A trap exception operation occurs when the processor encounters a trap instruction.
Affecting Flags	unaffected
exceptions	booby-trap anomaly

32-bit instruction	
manipulate	Causes trap anomalies to occur
grammatical	trap32 0, trap32 1. trap32 2. trap32 3
clarification	A trap exception operation occurs when the processor encounters a trap instruction.
Affecting Flags	unaffected
exceptions	booby-trap anomaly

32-bit instruction format:

trap32 0

31	30	26 25	21 20	16 15	10 9	5 4	0
1	10000	00000	00000	001000	00001	00000	

Figure 14.219: TRAP-1

trap32 1

31	30	26 25	21 20	16 15	10 9	5 4	0
1	10000	00000	00000	001001	00001	00000	

Figure 14.220: TRAP-2

Gentech **E804**

User's Manual
trap52²

XUANTIE 玄铁

31 30	26 25	21 20	16 15	10 9	5 4	0
1	10000	00000	00000	001010	00001	00000

Figure 14.221: TRAP-3

trap32 3

31 30	26 25	21 20	16 15	10 9	5 4	0
1	10000	00000	00000	001011	00001	00000

Figure 14.222: TRAP-4

14.139 TST - zero test command

Harmonization Directive	
grammatical	tst rx, ry
manipulate	If (RX & RY) != 0, then C ← 1; if (RX & RY) != 0 else C← 0;
Compilation results	Compiles to the corresponding 16-bit or 32-bit instructions depending on the register range. if (x<16) and (y<16), then tst16 rx, ry; else tst32 rx, ry.
clarification	Tests the result of a bitwise sum of the values of RX and RY. If the result is not equal to 0, condition bit C is set; otherwise, condition bit C is cleared.
Markers of influence classifier for honorific people	Setting the condition bit according to the press bit and the result C
exceptions	not have

16-bit instructions	
manipulat e	If (RX & RY) != 0, then C ← 1; if (RX & RY) ! else C← 0;
grammatic al	tst16 rx, ry
clarificatio n	Tests the result of a bitwise sum of the values of RX and RY. If the result is not equal to 0, condition bit C is set; otherwise, condition bit C is cleared.
Markers of influence classifier for honorific people	Setting the condition bit according to the press bit and the result C
limitation	The range of the register is r0-r15.
exceptions	not have

16-bit instruction format:



Figure 14.223: TST-1

32-bit instruction	
manipulat e	If (RX & RY) != 0, then C ← 1; if (RX & RY) ! else C← 0;
grammatic al	tst32 rx, ry
clarificatio n	Tests the result of a bitwise sum of the values of RX and RY. If the result is not equal to 0, condition bit C is set; otherwise, condition bit C is cleared.
Markers of influence classifier for	Setting the condition bit according to the press bit and the result C

User's Manual	
people	
exceptions	not have

32-bit instruction format:

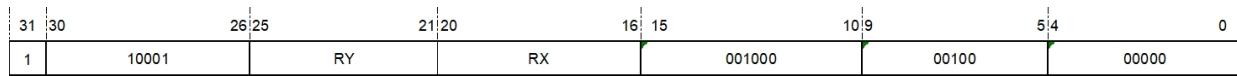


Figure 14.224: TST-2

14.140 TSTNBZ - No Byte Equals Zero Register Test Instruction

Harmonization Directive	
grammatical	tstnbz16 rx
manipulate	If ((RX[31:24] != 0) &(RX[23:16] != 0) &(RX[15: 8] != 0) &(RX[7 : 0] != 0)), then C← 1; else C← 0;
Compilation results	Compiles to the corresponding 16-bit or 32-bit instructions depending on the register range. if (x<16), then tstnbz16 rx; else tstnbz32 rx.
clarification	Tests if there are no bytes equal to zero in RX. If there are no bytes equal to zero in RX, set condition bit C; otherwise, clear the Condition Bit C.
Markers of influence classifier for honorific people	Setting the condition bit according to the press bit and the result C
exception	not have

16-bit instructions	
manipulate	If ((RX[31:24] != 0) &(RX[23:16] != 0) &(RX[15: 8] != 0) &(RX[7 : 0] != 0)), then C← 1; else C← 0;
grammatical	tstnbz16 rx

User Manual on classifier	Tests if there are no bytes equal to zero in RX. If there are no bytes equal to zero in RX, set condition bit C; otherwise, clear the Condition Bit C.
Markers of influenc e classifier for honorific people	Setting the condition bit according to the press bit and the result C
limitation	The range of the register is r0-r15.
exception	not have

16-bit instruction format:

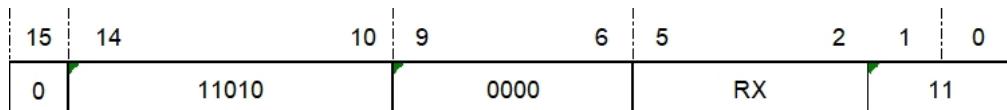


Figure 14.225: TSTNBZ-1

32-bit instruction	
manipulate	If ((RX[31:24] != 0) &(RX[23:16] != 0) &(RX[15: 8] != 0) &(RX[7 : 0] != 0)), then C← 1; else C← 0;
grammatical	tstnbz32 rx
clarification	Tests if there are no bytes equal to zero in RX. If there are no bytes equal to zero in RX, set condition bit C; otherwise, clear the Condition Bit C.
Markers of influence classifier for honorific people	Setting the condition bit according to the press bit and the result C
exceptions	not have

32-bit instruction format:

Figure 14.226: TSTNBZ-2

14.141 WAIT - enter low power wait mode command

Harmonization Directive	
grammatical	wait
manipulate	Enter low-power wait mode
compilation result resolute	Only 32-bit instructions exist. wait32
Attributes:	privileged instruction
clarification	This instruction stops the current instruction execution and waits for an interrupt, at which point the CPU clock stops. All peripherals continue running, and may generate an interrupt that causes the CPU to exit from wait mode.
affect aspiration	unaffected
exceptions	Privileged violation of directives

32-bit instruction	
manipulate	Enter low-power wait mode
grammatical	wait32
Attributes:	privileged instruction
clarification	This instruction stops the current instruction execution and waits for an interrupt, at which point the CPU clock stops. All peripherals continue running, and may generate an interrupt that causes the CPU to exit from wait mode.
affect aspiration	unaffected
exceptions	Privileged violation of directives

32-bit instruction format:

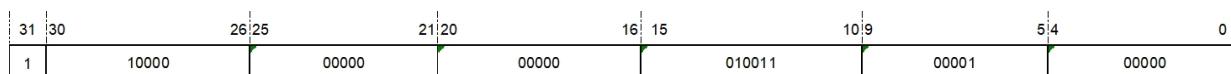


Figure 14.227: WAIT

14.142 XOR - per-bitwise-orthogonal-or instruction

Harmonization Directive		
tell to Budd hist teaching	xor rz, rx	xor rz, rx, ry
drill (practice) writings or works	RZ← RZ^ RX	RZ← RX^ RY
C o m p i l a t i o n r e s u l t s	Compile to 16-bit or 32-bit depending on the register range. Bit Commands. if (x<16) and (z<16), then xor16 rz, rx; else xor32 rz, rx.	Compiles to the corresponding 16-bit or 32-bit instructions depending on the register range. if (y==z) and (z<16) and (x<16), then xor16 rz, rx; else xor32 rz, rx, ry.
persuade generic term for a sacrifice the gods	Bitwise isochronize RX with the value of RZ/RY and store the result in RZ.	
M ark er s of in	unaffected	

fl User's Manual u e n ce classi fier for hono rific peopl e	
discrimina te const ant	not have

16-bit instructions	
manipulate	RZ \leftarrow RZ \wedge RX
grammatical	xor16 rz, rx
clarification	The value of RZ and RX are bitwise-orthogonal and the result is stored in RZ.
Affecting Flags	unaffected
limitation	The range of the register is r0-r15.
exceptions	not have

16-bit instruction format:



Figure 14.228: XOR-1

32-bit instruction	
manipulate	RZ \leftarrow RX \wedge RY
grammatical	xor32 rz, rx, ry
clarification	The values of RX and RY are bitwise-orthogonal and the result is stored in RZ.
Affecting Flags	unaffected
exceptions	not have

32-bit instruction format:



Figure 14.229: XOR-2

14.143 XORI - Immediate Number by Bit Orders

Harmonization Directive	
grammatical	xori rz, rx, imm16
manipulate	RZ \leftarrow RX \wedge zero_extend(IMM12)
Compilation results	Only 32-bit instructions exist. xori32 rz, rx, imm12
clarification	Expands the 12-bit immediate zero to 32 bits, then performs a bitwise different-or operation with the value of RX and stores the result in RZ.
Affecting Flags	unaffected
limitation	The range of immediate numbers is 0x0-0xFFFF.
exceptions	not have

32-bit instruction	
manipulate	RZ \leftarrow RX \wedge zero_extend(IMM12)
grammatical	xori32 rz, rx, imm12
clarification	Expands the 12-bit immediate zero to 32 bits, then performs a bitwise different-or operation with the value of RX and stores the result in RZ.
Affecting Flags	unaffected
limitation	The range of immediate numbers is 0x0-0xFFFF.

32-bit instruction format:

31	30	26	25	21	20	16	15	12	11	0
1	11001		RZ		RX		0100		IMM 12	

Figure 14.230: XORI

14.144 XSR - Extended right shift instruction

Harmonization Directive	
tell to Bud dhist teac hing	xsr rz, rx, oimm5
drill (prac tice) writi ngs or work s	{RZ,C} \leftarrow {RX,C} >>> OIMM5
c o m pi la ti o n re s ul t resol ute	Only 32-bit instructions exist. xsr32 rz, rx, oimm5
cl ar ifi ca ti o n	The value of RX with condition bit C ({RX,C}) is right-shifted cyclically (the original value is right-shifted, and the left side is shifted into the right side of the shifted bit), and the lowest bit ([0]) of the shift result is stored in the condition bit C, and the high bit ([32:1]) is stored in RZ, and the number of right-shifted bits consists of the 5-bit immediate number with a bias of one. (The value of OIMM5 is determined. If the value of OIMM5 is equal to 32, then condition bit C is the highest bit of RX.)

User's Manual	C←RX[OIMM5 - 1]
ar k er s of in fl u e n ce class ifier for hono rific peop le	
set a limit (on) regul ate	Immediate numbers range from 1-32.
discr imin ate cons tant	not have

32-bit instruction	
drill (pra ctice) writ ings or wor ks	{RZ,C} ← {RX,C} >>> OIMM5
tell to Bud dhis t teac hing	xsr32 rz, rx, oimm5
cl ar ifi ca ti o n	<p>The value of RX with condition bit C ({RX,C}) is right-shifted cyclically (the original value is right-shifted, and the left side is shifted into the right side of the shifted bit), and the lowest bit ([0]) of the shift result is stored in the condition bit C, and the high bit ([32:1]) is stored in RZ, and the number of right-shifted bits consists of the 5-bit immediate number with a bias of one.</p> <p>(The value of OIMM5 is determined. If the value of OIMM5 is equal to 32, then condition bit C is the highest bit of RX.)</p> <p>Note: The binary operand IMM5 is equal to OIMM5 - 1.</p>
M ar k er s of in fl u e n ce clas sifie r for hon orifi c peo ple	C ← RX[OIMM5 - 1]

Set & limit (on) register constant	Users Manual Immediate numbers range from 1-32.
discriminate constant	not have

32-bit instruction format:



Figure 14.231: XSR

IMM5 domain

Specifies a value without a bias immediate number.

Note: The shifted value OIMM5 is biased by 1 compared to the binary operand IMM5.

00000

Shift 1 bit

00001

Move 2 positions

.....

11111

Shift 32 bits

14.145 XTRB0 - Extract byte 0 and unsigned extension instruction

Harmonization Directive	
grammatical	xtrb0 rz, rx
manipulate	RZ \leftarrow zero_extend(RX[31:24]); if (RX[31:24] == 0), then C \leftarrow 0; else C \leftarrow 1;
Translation resolute	Only 32-bit instructions exist. xtrb0.32 rz, rx
clarification	Extracts byte 0 of RX (RX[31:24]) to the low bits of RZ (RZ[7:0]) with zero expansion. If the result is equal to 0, the C bit is cleared, and vice versa, the C bit is set.
impact indicator aspiration	If the result is equal to 0, the C bit is cleared, and vice versa, the C bit is set.
exceptions	not have

32-bit instruction	
manipulate	RZ \leftarrow zero_extend(RX[31:24]); if (RX[31:24] == 0), then C \leftarrow 0; else C \leftarrow 1;
grammatical	xtrb0.32 rz, rx
clarification	Extracts byte 0 of RX (RX[31:24]) to the low bits of RZ (RZ[7:0]) with zero expansion. If the result is equal to 0, the C bit is cleared, and vice versa, the C bit is set.
impact indicator aspiration	If the result is equal to 0, the C bit is cleared, and vice versa, the C bit is set.
exceptions	not have

32-bit instruction format:

31	30	26	25	21	20	16	15	10	9	5	4	0
1		10001		00000		RX		011100		00001		RZ

Figure 14.232: XTRB0

14.146 XTRB1 - Extract Byte 1 and unsigned extension instruction

Harmonization Directive	
grammatical	xtrb1 rz, rx
manipulate	RZ \leftarrow zero_extend(RX[23:16]); if (RX[23:16] == 0), then C \leftarrow 0; else C \leftarrow 1;
Translation resolute	Only 32-bit instructions exist. xtrb1.32 rz, rx
clarification	Extracts byte 1 of RX (RX[23:16]) to the low bits of RZ (RZ[7:0]) with zero expansion. If the result is equal to 0, the C bit is cleared, and vice versa, the C bit is set.
impact indicator aspiration	If the result is equal to 0, the C bit is cleared, and vice versa, the C bit is set.
exceptions	not have

32-bit instruction	
manipulate	RZ \leftarrow zero_extend(RX[23:16]); if (RX[23:16] == 0), then C \leftarrow 0; else C \leftarrow 1;
grammatical	xtrb1.32 rz, rx
clarification	Extracts byte 1 of RX (RX[23:16]) to the low bits of RZ (RZ[7:0]) with zero expansion. If the result is equal to 0, the C bit is cleared, and vice versa, the C bit is set.
impact indicator aspiration	If the result is equal to 0, the C bit is cleared, and vice versa, the C bit is set.
exceptions	not have

32-bit instruction format:

31	30	26	25	21	20	16	15	10	9	5	4	0
1		10001		00000		RX		011100		00010		RZ

Figure 14.233: XTRB1

14.147 XTRB2 - Extract Byte 2 and unsigned extension instruction

Harmonization Directive	
grammatical	xtrb2 rz, rx
manipulate	RZ \leftarrow zero_extend(RX[15:8]); if (RX[15:8] == 0), then C \leftarrow 0; else C \leftarrow 1;
Translation resolute	Only 32-bit instructions exist. xtrb2.32 rz, rx
clarification	Extracts byte 2 of RX (RX[15:8]) to the low bits of RZ (RZ[7:0]) with zero expansion. If the result is equal to 0, the Clears the C bit and conversely sets the C bit.
impact indicator aspiration	If the result is equal to 0, the C bit is cleared, and vice versa, the C bit is set.
exceptions	not have

32-bit instruction	
manipulate	RZ \leftarrow zero_extend(RX[15:8]); if (RX[15:8] == 0), then C \leftarrow 0; else C \leftarrow 1;
grammatical	xtrb2.32 rz, rx
clarification	Extracts byte 2 of RX (RX[15:8]) to the low bits of RZ (RZ[7:0]) with zero expansion. If the result is equal to 0, the Clears the C bit and conversely sets the C bit.
impact indicator aspiration	If the result is equal to 0, the C bit is cleared, and vice versa, the C bit is set.
exceptions	not have

32-bit instruction format:

31	30	26	25	21	20	16	15	10	9	5	4	0
1		10001		00000		RX		011100		00100		RZ

Figure 14.234: XTRB2

14.148 XTRB3 - Extract Byte 3 and Unsigned Extension Instruction

Harmonization Directive	
grammatical	xtrb3 rz, rx
manipulate	RZ \leftarrow zero_extend(RX[7:0]); if (RX[7:0] == 0), then C \leftarrow 0; else C \leftarrow 1;
Translation resolute	Only 32-bit instructions exist. xtrb3.32 rz, rx
clarification	Extracts byte 3 of RX (RX[7:0]) to the low bits of RZ (RZ[7:0])with zero expansion. If the result is equal to 0, the Clears the C bit and conversely sets the C bit.
impact indicator aspiration	If the result is equal to 0, the C bit is cleared, and vice versa, the C bit is set.
exceptions	not have

32-bit instruction	
manipulate	RZ \leftarrow zero_extend(RX[7:0]); if (RX[7:0] == 0), then C \leftarrow 0; else C \leftarrow 1;
grammatical	xtrb3.32 rz, rx
clarification	Extracts byte 3 of RX (RX[7:0]) to the low bits of RZ (RZ[7:0])with zero expansion. If the result is equal to 0, the Clears the C bit and conversely sets the C bit.
impact indicator aspiration	If the result is equal to 0, the C bit is cleared, and vice versa, the C bit is set.
exceptions	not have

32-bit instruction format:

31	30	26	25	21	20	16	15	10	9	5	4	0
1		10001		00000		RX		011100		01000		RZ

Figure 14.235: XTRB3

14.149 ZEXT - Bit Extraction and Unsigned Extension Instructions

Harmonization Directive	
tell	zext rz, rx, msb.
to	
Bud	
dhis	
t	
teac	
hing	
m	RZ← zero_extend(RX[MSB:LSB])
a	
ni	
p	
ul	
at	
e	
lsb	
c	Only 32-bit instructions exist.
o	zext32 rz, rx, msb, lsb
m	
pi	
la	
ti	
o	
n	
re	
s	
ul	
t	
reso	
lute	
cl	Extracts a contiguous segment of RX (RX[MSB:LSB]) specified by two 5-bit immediate numbers (MSB,LSB), zero-extends it to 32 bits, and stores the result in RZ. If MSB equals 31, and且 LSB equals 0, then the value of RZ is the as RX. If MSB is equal to LSB, the value of RZ is the result of zero expansion of one bit of RX[MSB] (i.e., RX[LSB]). If the MSB is less than the LSB, the Command behavior is unpredictable.
ar	
ifi	
ca	
ti	
o	
n	
M	unaffected
ar	
k	
er	
s	
of	

in User's fl u e n ce clas sifie r for hon orifi c peo ple	Manual
set a limit (on) regu late	MSB ranges from 0-31, LSB ranges from 0-31, and MSB should be greater than or equal to LSB.
disc rimi nate cons tant	not have

32-bit instruction	
drill (pra ctice) writi ngs or wor ks	RZ \leftarrow zero_extend(RX[MSB:LSB])
tell to Bud dhis t teac hing	zext32 rz, rx, msb, lsb
cl ar ifi ca ti o n	Extracts a contiguous segment of RX (RX[MSB:LSB]) specified by two 5-bit immediate numbers (MSB,LSB), zero-extends it to 32 bits, and stores the result in RZ. If MSB equals 31, and且 LSB equals 0, then the value of RZ is the as RX. If MSB is equal to LSB, the value of RZ is the result of zero expansion of one bit of RX[MSB] (i.e., RX[LSB]). If the MSB is less than the LSB, the Command behavior is unpredictable.
M ar k er s of in fl u e n ce clas sifie r for hon orifi c peo ple	unaffected

User's Manual	MSB ranges from 0-31, LSB ranges from 0-31, and MSB should be greater than or equal to LSB.
Set a limit (on) regulate	not have

32-bit instruction format:

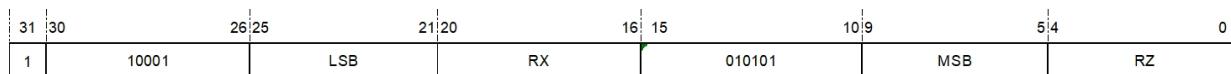


Figure 14.236: ZEXT

MSB domain

Specifies the bit to be extracted to start with.

LSB domain

Specifies the bit that is to be extracted at the end of the extraction.

00000

0

00000

0 bit

00001

1

00001

1 position

.....
11111

31

11111

31 bits

14.150 ZEXTB - Byte Extraction with Unsigned Expansion Instruction

Harmonization Directive	
grammatical	zextb rz, rx
manipulate	RZ← zero_extend(RX[7:0]);
Compilation results	Compiles to the corresponding 16-bit or 32-bit instructions depending on the register range. if (x<16) and (z<16), then zextb16 rz, rx; else zextb32 rz, rx
clarification	Zero-extend the low byte of RX (RX[7:0]) to 32 bits and the result is present RZ.
Markers of influence classifier for honorific people	unaffected
exceptions	not have

16-bit instructions	
manipulate	RZ← zero_extend(RX[7:0]);
grammatical	zextb16 rz, rx
clarification	Zero-extend the low byte of RX (RX[7:0]) to 32 bits and the result is present RZ.
Affecting Flags	unaffected
limitation	The range of the register is r0-r15.

16-bit instruction format:

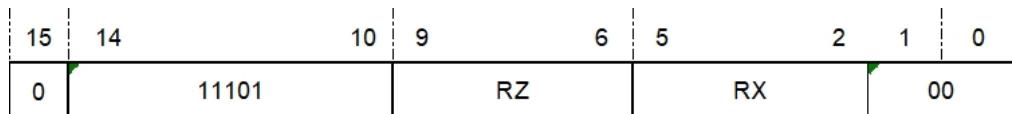


Figure 14.237: ZEXTB-1

32-bit instruction	
manipula te	RZ← zero_extend(RX[7:0]);
grammati cal	zextb32 rz, rx
clarificati on	Zero-extend the low byte of RX (RX[7:0]) to 32 bits and the result is present RZ. Note that this instruction is a pseudo-instruction for zext32 rz, rx, 0x7, 0x0.
Markers of influence classifier for honorific people	unaffected
exception s	not have

32-bit instruction format:



Figure 14.238: ZEXTB-2

14.151 ZEXTH - half-word extraction and unsigned extension instruction

Harmonization Directive	
grammati cal	zexth rz, rx
manipula te	RZ← zero_extend(RX[15:0]);
Compilati on results	Compiles to the corresponding 16-bit or 32-bit instructions depending on the register range. if (x<16) and (z<16), then zexth16 rz, rx; else zexth32 rz, rx
clarificati on	Extend the low halfword of RX (RX[15:0]) by zero to 32 bits and the result is stored RZ.
Markers of	unaffected

influence Users Manual classifier for honorific people	
exception s	not have

16-bit instructions	
manipulate	RZ← zero_extend(RX[15:0]);
grammatical	zexth16 rz, rx
clarification	Extend the low halfword of RX (RX[15:0]) by zero to 32 bits and the result is stored RZ.
Affecting Flags	unaffected
limitation	The range of the register is r0-r15.
exceptions	not have

16-bit instruction format:

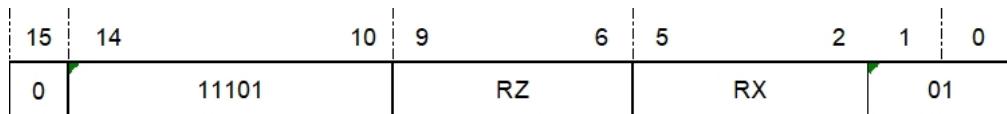


Figure 14.239: ZEXTH-1

32-bit instruction	
manipula te	RZ← zero_extend(RX[15:0]);
grammati cal	zexth32 rz, rx
clarificati on	Extend the low halfword of RX (RX[15:0]) by zero to 32 bits and the result is stored RZ. Note that this instruction is a pseudo-instruction for zext32 rz, rx, 0x15, 0x0.
Markers of influence classifier for honorific people	unaffected
exception s	not have

32-bit instruction format:



Figure 14.240: ZEXTH-2

Chapter 15 Appendix B DSP Instruction Set

Glossary

The following describes the DSP instruction subset in alphabetical order.

The mnemonic for each DSP instruction contains three parts, each separated by . The first part identifies the basic operation of the instruction, the second part identifies the data type of the operand, and the third part identifies the processing of the result of the operation. Taking "16-bit parallel signed addition instruction with saturation operation-PADD.S16.S" as an example, PADD means that this is a parallel addition instruction, S16 means that the operand is a signed half-word, and S means that the result of the operation is saturated.

The common letters in the command mnemonics and their meanings are listed below:

mnemonic sign	English presentation	significance
Part I ingredient	P(Parallel)	Parallel computation, denoting single instruction multiple data operations.
Part I ingredient	R(fRaction)	Fractional operations.
Part I ingredient	X (Unalign)	The operand bit widths are not aligned, or the operand positions are not aligned.
Part I ingredient	H (Half)	Take the average.
Part II ingredient	(S/U)/(8/16/32)	(signed/unsigned)/(byte/half-word/word) If it is not indicated with a sign bit, it means that the operation of the instruction is not concerned with the sign bit.
Part III ingredient	S(Saturate)	Saturation operation.
Part III ingredient	R (Round)	rounding operation.
Part III ingredient	E (Extension)	Expansion operation, that is, the operand into the expansion operation after the operation to participate in the operation to produce a result.

15.1 PADD.8 - 8 bit parallel addition instructions

Unification directives	
grammatical	padd.8 rz, rx, ry
manipulate	Rz[31:24]= Rx[31:24]+ Ry[31:24] Rz[23:16]= Rx[23:16]+ Ry[23:16] Rz[15:8]= Rx[15:8]+ Ry[15:8] Rz[7:0]= Rx[7:0]+ Ry[7:0]
Translation resolute	Only 32-bit instructions exist. padd.8 rz, rx, ry

Description:	The 4 bytes Rx are added to the 4 bytes Ry in bytes, and the result of the addition is stored in Rz.
impact indicator	unaffected
Chi Bit:	
Exception:	not have
32-bit finger honorific title	
Operation:	Rz[31:24]= Rx[31:24]+ Ry[31:24] Rz[23:16]= Rx[23:16]+ Ry[23:16] Rz[15:8]= Rx[15:8]+ Ry[15:8] Rz[7:0]= Rx[7:0]+ Ry[7:0]
Grammar:	padd.8 rz, rx, ry
Description:	The 4 bytes Rx are added to the 4 bytes Ry in bytes, and the result of the addition is stored in Rz.
impact indicator	unaffected
Chi Bit:	
Exception:	not have

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0 RY	RX		1 1 0 0 0 0 0 X 1 0 RZ			

Figure 15.1: PADD.8

15.2 PADD.16 - 16-bit Parallel Addition Instruction

Harmonization Directive	
grammatical	padd.16 rz, rx,ry
manipulate	$Rz[31:16] = Rx[31:16] + Ry[31:16]$ $Rz[15:0] = Rx[15:0] + Ry[15:0]$
Compilation results	Only 32-bit instructions exist. padd.16 rz, rx,ry

Description:	The high and low half words of Rx are added to the high and low half words of Ry respectively, and the result of the addition is stored in Rz.
Impact flag bits:	unaffected
Exception:	not have
32-bit instruction	
Operation:	$Rz[31:16] = Rx[31:16] + Ry[31:16]$ $Rz[15:0] = Rx[15:0] + Ry[15:0]$
Grammar:	padd.16 rz, rx,ry
Description:	The high and low half words of Rx are added to the high and low half words of Ry respectively, and the result of the addition is stored in Rz.
Impact flag bits:	unaffected
Exception:	not have

Command Format:

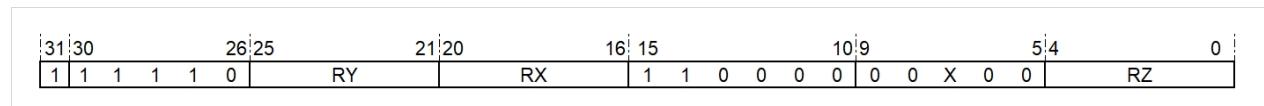


Figure 15.2: PADD.16

15.3 PADD.(U/S)8.S - 8-bit parallel (no/have) signed addition with saturation operation

directives

unify chemistry calibration honorable title	
grammatical	padd.u8.s rz, rx, ry padd.s8.s rz, rx, ry
manipulate	Rz[31:24] = Saturate(Rx[31:24] + Ry[31:24]) Rz[23:16] = Saturate(Rx[23:16] + Ry[23:16]) Rz[15:8] = Saturate(Rx[15:8] + Ry[15:8]) Rz[7:0] = Saturate(Rx[7:0] + Ry[7:0])
translators on results	Only 32-bit instructions exist. padd.u8.s rz, rx, ry padd.s8.s rz, rx, ry

Explana tion: :	For unsigned numbers, the saturation process is as follows: if the 8-bit addition result is greater than the saturation value 0xFF, the result is taken as 0xFF, otherwise the result is taken as the addition result itself. For signed numbers, the saturation process is as follows: if the 8-bit addition result is greater than the upper saturation value of 0x7F, the result is taken as 0x7F, and if the 8-bit addition result is greater than the upper saturation value of 0x7F, the result is taken as 0x7F. If the result of the method is less than the lower saturation value 0x80, the result is taken as 0x80, otherwise the addition result itself is taken.
Mark ers of in fl u e n ce Bit:	unaffected
disc rimi nate Reg ular :	not have
32 in d e x (m at h.) hon orifi c title	
Operate rate: :	Rz[31:24] = Saturate(Rx[31:24] + Ry[31:24]) Rz[23:16] = Saturate(Rx[23:16] + Ry[23:16]) Rz[15:8] = Saturate(Rx[15:8] + Ry[15:8]) Rz[7:0]= Saturate(Rx[7:0]+ Ry[7:0])

tell User's Manual to Law :	padd.u8.s rz, rx,ry padd.s8.s rz, rx,ry
Explana-tion :	<p>For unsigned numbers, the saturation process is as follows: if the 8-bit addition result is greater than the saturation value 0xFF, the result is taken as 0xFF, otherwise the result is taken as the addition result itself.</p> <p>For signed numbers, the saturation process is as follows: if the 8-bit addition result is greater than the upper saturation value of 0x7F, the result is taken as 0x7F, and if the 8-bit addition result is greater than the upper saturation value of 0x7F, the result is taken as 0x7F.</p> <p>If the result of the method is less than the lower saturation value 0x80, the result is taken as 0x80, otherwise the addition result itself is taken.</p>
Mark ers of infl uen ce Bit:	unaffected
disc rimi nate Reg ular :	not have

Command Format:

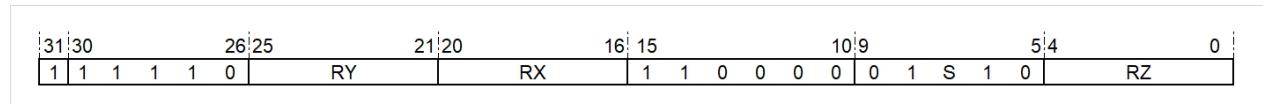


Figure 15.3: PADD.(US)8.

15.4 PADD.(U/S)16.S - 16-bit parallel (without/with) sign add with saturation operation

French instructions

Harmonization Directive	
grammatical	padd.u16.s rz, rx,ry padd.s16.s rz, rx,ry
manipulate	Rz[31:16]= Saturate(Rx[31:16]+ Ry[31:16]) Rz[15:0]= Saturate(Rx[15:0]+ Ry[15:0])
Compilation results	Only 32-bit instructions exist. padd.u16.s rz, rx,ry padd.s16.s rz, rx,ry

Exp lana tion :	The high and low half words of Rx and Ry are added separately, and the result is saturated and stored in Rz. For unsigned numbers, the saturation process is as follows: if the 16-bit addition result is greater than the saturation value of 0xFFFF, the result is taken as 0xFFFF, and in other cases, the result is taken as the result of the addition itself. For signed numbers, the saturation process is as follows: if the 16-bit addition result is greater than the upper saturation value of 0x 7FFF, the result is taken as 0x7FFF, if If the 16-bit addition result is less than the lower saturation value of 0x8000, the result is taken as 0x8000, otherwise the addition result itself is taken.
M ar k er s of in fl u e n ce Bit:	unaffected
disc rimi nate Reg ular :	not have
32 in d e x (m at h.) hon orifi c title	
drill (pra ctic e) Ma	Rz[31:16]= Saturate(Rx[31:16]+ Ry[31:16]) Rz[15:0]= Saturate(Rx[15:0]+ Ry[15:0])

User's Manual	
tell to Law :	padd.u16.s rz, rx,ry padd.s16.s rz, rx,ry
Explanation :	The high and low half words of Rx and Ry are added separately, and the result is saturated and stored in Rz. For unsigned numbers, the saturation process is as follows: if the 16-bit addition result is greater than the saturation value of 0xFFFF, the result is taken as 0xFFFF, and in other cases, the result is taken as the result of the addition itself. For signed numbers, the saturation process is as follows: if the 16-bit addition result is greater than the upper saturation value of 0x 7FFF, the result is taken as 0x7FFF, if If the 16-bit addition result is less than the lower saturation value of 0x8000, the result is taken as 0x8000, otherwise the result itself is taken as the addition result.
Mark ers of infl u e n ce Bit:	unaffected
disc rimi nate Reg ular :	not have
or d er fo r m Styl e:	

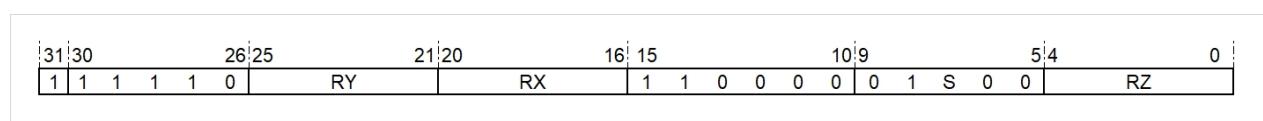


Figure 15.4: PADD.(US)16.

15.5 ADD.(U/S)32.S - 32-bit (no/have) signed addition instruction with saturation operation

Harmonization Directive		
grammatical	manipulate	Compilation results
add.u32.s rz, rx,ry add.s32.s rz, rx,ry	Rz[31:0]= Saturate(Rx[31:0] + Ry[31:0])	Only 32-bit instructions exist. add.u32.s rz, rx,ry add.s32.s rz, rx,ry

Exp lana tion :	Rx is added to Ry and the result of the addition is saturated and stored in Rz. For unsigned numbers, saturation is handled as follows: if the 32-bit addition result is greater than the upper saturation value of 0xFFFF FFFF, the result is taken as 0xFFFF FFFF, otherwise the addition result itself is taken. For signed numbers, the saturation process is as follows: if the 32-bit addition result is greater than the upper saturation value of 0x7FFF FFFF, the result is taken as 0x7FFF FFFF, if the 32-bit addition result is less than the lower saturation value of 0x8000 0000, the result is taken as 0x8000 0000, and in other cases, the result is taken as the result of addition itself.
M ar k er s of in fl u e n ce Bit:	unaffected
disc rimi nate Reg ular :	not have
32 in d e x (m at h.) hon orifi c title	
drill (pra ctic e)	Rz[31:0]= Saturate(Rx[31:0]+ Ry[31:0])

User's Manual	
tell to Law :	add.u32.s rz, rx,ry add.s32.s rz, rx,ry
Explanation :	Rx is added to Ry and the result of the addition is saturated and stored in Rz. For unsigned numbers, saturation is handled as follows: if the 32-bit addition result is greater than the upper saturation value of 0xFFFF FFFF, the result is taken as 0xFFFF FFFF, otherwise the addition result itself is taken. For signed numbers, the saturation process is as follows: if the 32-bit addition result is greater than the upper saturation value of 0x7FFF FFFF, the result is taken as 0x7FFF FFFF, if the 32-bit addition result is less than the lower saturation value of 0x8000 0000, the result is taken as 0x8000 0000, and in other cases, the result is taken as the result of addition itself.
Mark ers of infl uen ce Bit:	unaffected
disc rimi nate Reg ular :	not have

Command Format:

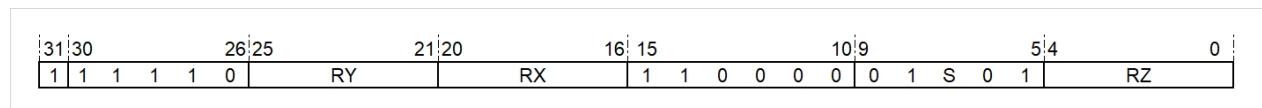


Figure 15.5: ADD.(US)32.

15.6 PSUB.8 - 8-bit Parallel Subtraction Instruction

Unification directives	
grammatical	psub.8 rz, rx, ry
manipulate	$Rz[31:24] = Rx[31:24] - Ry[31:24]$ $Rz[23:16] = Rx[23:16] - Ry[23:16]$ $Rz[15:8] = Rx[15:8] - Ry[15:8]$ $Rz[7:0] = Rx[7:0] - Ry[7:0]$
Translation resolute	Only 32-bit instructions exist. psub.8 rz, rx, ry

Description:	The 4 bytes Rx are subtracted from the 4 bytes Ry in bytes, and the result of the subtraction is stored in Rz.
impact indicator Chi Bit:	unaffected
Exception:	not have
32-bit finger honorific title	
Operation:	$Rz[31:24] = Rx[31:24] - Ry[31:24]$ $Rz[23:16] = Rx[23:16] - Ry[23:16]$ $Rz[15:8] = Rx[15:8] - Ry[15:8]$ $Rz[7:0] = Rx[7:0] - Ry[7:0]$
Grammar:	psub.8 rz, rx, ry
Description:	The 4 bytes Rx are subtracted from the 4 bytes Ry in bytes, and the result of the subtraction is stored in Rz.
impact indicator Chi Bit:	unaffected
Exception:	not have

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	RY	RX	1 1 0 0 0 1	0 0 X 1 0	RZ	

Figure 15.6: PSUB.8

15.7 PSUB.16 - 16-Bit Parallel Subtraction Instruction

Harmonization Directive	
grammatical	psub.16 rz, rx, ry
manipulate	$Rz[31:16] = Rx[31:16] - Ry[31:16]$ $Rz[15:0] = Rx[15:0] - Ry[15:0]$
Compilation results	Only 32-bit instructions exist. psub.16 rz, rx, ry

Description:	Subtract the high and low half-words of Rx from the high and low half-words of Ry, respectively, and the result of the subtraction is stored in Rz.
Impact flag bits:	unaffected
Exception:	not have
32-bit instruction	
Operation:	$Rz[31:16] = Rx[31:16] - Ry[31:16]$ $Rz[15:0] = Rx[15:0] - Ry[15:0]$
Grammar:	psub.16 rz, rx, ry
Description:	Subtract the high and low half-words of Rx from the high and low half-words of Ry, respectively, and the result of the subtraction is stored in Rz.
Impact flag bits:	unaffected
Exception:	not have

Command Format:

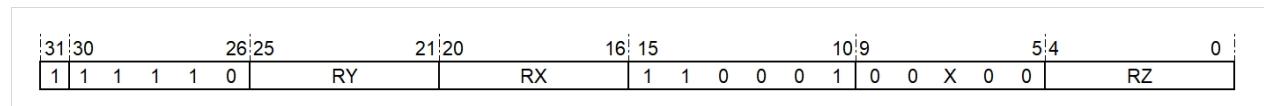


Figure 15.7: PSUB.16

15.8 PSUB.(U/S)8.S - 8-bit parallel (no/have) signed subtraction finger with saturation operation

honorific title

unify chemistry calibration honorific title	
grammatical	psub.u8.s rz, rx, ry psub.s8.s rz, rx, ry
manipulate	Rz[31:24]= Saturate(Rx[31:24] - Ry[31:24]) Rz[23:16]= Saturate(Rx[23:16] - Ry[23:16]) Rz[15:8]= Saturate(Rx[15:8] - Ry[15:8]) Rz[7:0]= Saturate(Rx[7:0] - Ry[7:0])
Translation results	Only 32-bit instructions exist. psub.u8.s rz, rx, ry psub.s8.s rz, rx, ry

Explana tion: :	For unsigned numbers, the saturation process is as follows: if the 8-bit subtraction result is less than the saturation value of 0x0, the result is taken as 0x0, otherwise, the result is taken as the subtraction result itself. For signed numbers, the saturation process is as follows: if the 8-bit subtraction result is greater than the upper saturation value of 0x7F, the result is taken as 0x7F, if the 8-bit subtraction result of the method is less than the lower saturation value 0x80, the result is taken as 0x80, otherwise the subtraction result itself is taken.
Mark ers of in fl u e n ce Bit: :	unaffected
disc rimi nate Reg ular :	not have
32 in d e x (m at h.) hon orifi c title	
Operate rate: :	Rz[31:24]= Saturate(Rx[31:24] - Ry[31:24]) Rz[23:16]= Saturate(Rx[23:16] - Ry[23:16]) Rz[15:8]= Saturate(Rx[15:8] - Ry[15:8]) Rz[7:0]= Saturate(Rx[7:0] - Ry[7:0])

tell User's Manual to Law :	psub.u8.s rz, rx,ry psub.s8.s rz, rx,ry
Explana tion : : Bit:	For unsigned numbers, the saturation process is as follows: if the 8-bit subtraction result is less than the saturation value of 0x0, the result is taken as 0x0, otherwise, the result is taken as the subtraction result itself. For signed numbers, the saturation process is as follows: if the 8-bit subtraction result is greater than the upper saturation value of 0x7F, the result is taken as 0x7F, if the 8-bit subtraction If the result of the method is less than the lower saturation value 0x80, the result is taken as 0x80, otherwise the subtraction result itself is taken.
I m p ac t m ar k er s	unaffected
disc rimi nate Reg ular : :	not have

Command Format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 1 1 0	RY	RX	1 1 0 0 0 1	0 1 S 1 0	RZ	

Figure 15.8: PSUB.(US)8.

15.9 PSUB.(U/S)16.S - 16-bit parallel (no/have) signed subtraction with saturation operation

directives

Harmonization Directive	
grammatical	psub.u16.s rz, rx, ry psub.s16.s rz, rx, ry
manipulate	Rz[31:16]= Saturate(Rx[31:16] - Ry[31:16]) Rz[15:0]= Saturate(Rx[15:0] - Ry[15:0])
Compilation results	Only 32-bit instructions exist. psub.u16.s rz, rx, ry psub.s16.s rz, rx, ry

Explana tion: :	The high and low half words of Rx and Ry are subtracted respectively, and the result of the subtraction is saturated and stored in Rz. For unsigned numbers, the saturation process is as follows: if the result of the 16-bit subtraction is less than the saturated value of 0x0, the result is taken as 0x0, and the result of the subtraction is taken as the result of the subtraction itself in other cases. For signed numbers, the saturation process is as follows: if the 16-bit subtraction result is greater than the upper saturation value of 0x 7FFF, the result is taken as 0x7FFF, if If the 16-bit subtraction result is less than the lower saturation value 0x8000, the result is taken as 0x8000, otherwise the subtraction result itself is taken.
Mark ers of in fl u e n ce Bit:	unaffected
disc rimi nate Reg ular :	not have
32 in d e x (m at h.) hon orifi c title	
drill (prac tic e)	Rz[31:16]= Saturate(Rx[31:16] - Ry[31:16]) Rz[15:0]= Saturate(Rx[15:0] - Ry[15:0])

User's Manual ke:	
tell to Law :	psub.u16.s rz, rx,ry psub.s16.s rz, rx,ry
Exp lana tion : :	The high and low half words of Rx and Ry are subtracted respectively, and the result of the subtraction is saturated and stored in Rz. For unsigned numbers, the saturation process is as follows: if the result of the 16-bit subtraction is less than the saturated value of 0x0, the result is taken as 0x0, and the result of the subtraction is taken as the result of the subtraction itself in other cases. For signed numbers, the saturation process is as follows: if the 16-bit subtraction result is greater than the upper saturation value of 0x 7FFF, the result is taken as 0x7FFF, if If the 16-bit subtraction result is less than the lower saturation value 0x8000, the result is taken as 0x8000, otherwise the subtraction result itself is taken.
M ar k er s of in fl u e n ce Bit:	unaffected
disc rimi nate Reg ular :	not have

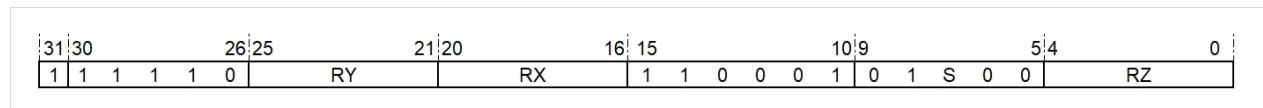
Command Format:

Figure 15.9: PSUB.(US)16.

15.10 SUB.(U/S)32.S - 32-bit (no/have) signed subtraction instruction with saturation operation

Harmonization Directive	
grammatical	sub.u32.s rz, rx,ry sub.s32.s rz, rx,ry
manipulate	Rz[31:0]= Saturate(Rx[31:0] - Ry[31:0])
Compilation results	Only 32-bit instructions exist. sub.u32.s rz, rx,ry sub.s32.s rz, rx,ry

Explanation:	Rx is subtracted from Ry and the result of the subtraction is saturated and stored in Rz. For unsigned numbers, saturation is handled as follows: if the 32-bit subtraction result is less than the saturation value 0x0, the result is taken as 0x0, otherwise the subtraction result itself is taken. For signed numbers, the saturation process is as follows: if the 32-bit subtraction result is greater than the upper saturation value of 0x7FFF FFFF, the result is taken as 0x7FFF FFFF, if the 32-bit subtraction result is less than the lower saturation value of 0x8000 0000, the result is taken as 0x8000 0000, and in other cases, the result is taken as the result of subtraction itself.
Markers of influence Bit:	unaffected
discriminate Register:	not have

32 in d e x (m at h.) hon orifi c title	
drill (prac tic e) Ma ke:	Rz[31:0]= Saturate(Rx[31:0] - Ry[31:0])
tell to Law :	sub.u32.s rz, rx,ry sub.s32.s rz, rx,ry
Exp lana tion : :	Rx is subtracted from Ry and the result of the subtraction is saturated and stored in Rz. For unsigned numbers, saturation is handled as follows: if the 32-bit subtraction result is less than the saturation value 0x0, the result is taken as 0x0, otherwise the subtraction result itself is taken. For signed numbers, the saturation process is as follows: if the 32-bit subtraction result is greater than the upper saturation value of 0x7FFF FFFF, the result is taken as 0x7FFF FFFF, if the 32-bit subtraction result is less than the lower saturation value of 0x8000 0000, the result is taken as 0x8000 0000, and in other cases, the result is taken as the result of subtraction itself.
M ar k er s of in fl u e n ce Bit:	unaffected

disc	Method
rimi	
nate	
Reg	
ular	
:	

Command Format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 1 1 1 0	RY	RX	1 1 0 0 0 1	0 1 S 0 1	RZ	

Figure 15.10: SUB.(US)32.

15.11 PADDH.(U/S)8 - 8-bit parallel (no/have) signed addition averaging instruction

standardization directives	
grammatical	paddh.u8 rz, rx,ry paddh.s8 rz, rx,ry
manipulate	Rz[31:24]=(Rx[31:24]+ Ry[31:24])/2 Rz[23:16]=(Rx[23:16]+ Ry[23:16])/2 Rz[15:8]=(Rx[15:8]+ Ry[15:8])/2 Rz[7:0]=(Rx[7:0]+ Ry[7:0])/2
Compilation results	Only 32-bit instructions exist. paddh.u8 rz, rx,ry paddh.s8 rz, rx,ry

Description :	Add 4 bytes of Rx to the 4 bytes Ry in bytes, and divide the result by 2 and store it in Rz.
Impact Marker Bit:	unaffected
Exception:	not have

32-bit finger honorific title	
Operation:	Rz[31:24]=(Rx[31:24]+ Ry[31:24])/2 Rz[23:16]=(Rx[23:16]+ Ry[23:16])/2 Rz[15:8]=(Rx[15:8]+ Ry[15:8])/2 Rz[7:0]=(Rx[7:0]+ Ry[7:0])/2
Grammar:	paddh.u8 rz, rx,ry paddh.s8 rz, rx,ry
Description:	The 4 bytes Rx are added to the 4 bytes Ry in bytes, and the result is divided by 2 and stored in Rz.
affect Chi Bit:	unaffected
Exception	not have

Command Format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 1 1 0	RY	RX	1 1 0 0 0	1 0 S 1 0	RZ	

Figure 15.11: PADDH.(US)8

15.12 PADDH.(U/S)16 - 16-bit Parallel (Un/Signed) Signed Addition for Averaging Finger

honorific title

Harmonization Directive	
grammatical	paddh.u16 rz, rx, ry paddh.s16 rz, rx, ry
manipulate	Rz[31:16]=(Rx[31:16]+Ry[31:16])/2 Rz[15:0]=(Rx[15:0]+Ry[15:0])/2
Compilation results	Only 32-bit instructions exist. paddh.u16 rz, rx, ry paddh.s16 rz, rx, ry

Description:	Add the upper and lower halves of Rx to the upper and lower halves of Ry, and divide the result by 2 and store it in Rz.
Impact flag bits:	unaffected
Exception:	not have

32-bit instruction	
Operation:	Rz[31:16]=(Rx[31:16]+Ry[31:16])/2 Rz[15:0]=(Rx[15:0]+Ry[15:0])/2
Grammar:	paddh.u16 rz, rx, ry paddh.s16 rz, rx, ry
Description:	Add the upper and lower halves of Rx to the upper and lower halves of Ry, and divide the result by 2 and store it in Rz.
Impact flag bits:	unaffected
Exception:	not have

Command Format:

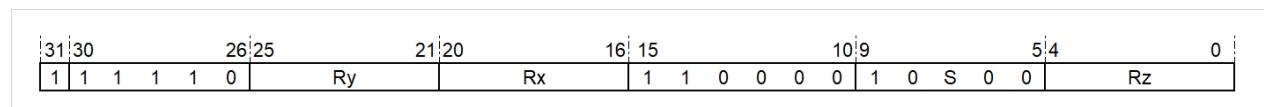


Figure 15.12: PADDH.(US)16

15.13 ADDH.(U/S)32 - 32-bit (no/have) signed addition averaging instruction

Harmonization Directive	
grammatical	addh.u32 rz, rx, ry addh.s32 rz, rx, ry
manipulate	Rz[31:0] = (Rx[31:0] + Ry[31:0])/2
Compilation results	Only 32-bit instructions exist. addh.u32 rz, rx, ry addh.s32 rz, rx, ry

Description:	Rx and Ry are added together and the result is divided by 2 and stored in Rz.
Impact flag bits:	unaffected
Exception:	not have

32-bit instruction	
Operation:	Rz[31:0] = (Rx[31:0] + Ry[31:0])/2
Grammar:	addh.u32 rz, rx, ry addh.s32 rz, rx, ry
Description:	Rx and Ry are added together and the result is divided by 2 and stored in Rz.
Impact flag bits:	unaffected
Exception:	not have

Command Format:

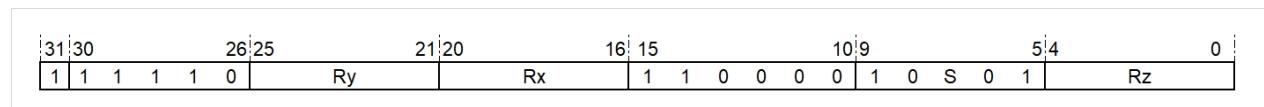


Figure 15.13: ADDH.(US)32

15.14 PSUBH.(U/S)8 - 8-bit parallel (no/have) signed subtraction averaging instruction

standardization directives	
grammatical	psubh.u8 rz, rx, ry psubh.s8 rz, rx, ry
manipulate	Rz[31:24]=(Rx[31:24] - Ry[31:24])/2 Rz[23:16]=(Rx[23:16] - Ry[23:16])/2 Rz[15:8]=(Rx[15:8] - Ry[15:8])/2 Rz[7:0]=(Rx[7:0] - Ry[7:0])/2
Compilation results	Only 32-bit instructions exist. psubh.u8 rz, rx, ry psubh.s8 rz, rx, ry

Description :	Subtract 4 bytes of Rx from the 4 bytes Ry in byte units, divide the result by 2 and store it in Rz.
Impact Marker Bit:	unaffected
Exception:	not have

32-bit finger honorific title	
Operation:	Rz[31:24]=(Rx[31:24] - Ry[31:24])/2 Rz[23:16]=(Rx[23:16] - Ry[23:16])/2 Rz[15:8]=(Rx[15:8] - Ry[15:8])/2 Rz[7:0]=(Rx[7:0] - Ry[7:0])/2
Grammar:	psubh.u8 rz, rx, ry psubh.s8 rz, rx, ry
Description:	Subtract 4 bytes of Rx from the 4 bytes Ry in bytes, and divide the result by 2 and store it in Rz.
impact indicator	unaffected

User's Manual	
Chi Bit:	
Exception:	not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 1 0 0 0	1 1 0 S 1 0	Rz	

Figure 15.14: PSUBH.(US)8

15.15 PSUBH.(U/S)16 - 16-bit parallel (no/have) signed subtraction averaging instruction

Harmonization Directive	
grammatical	psubh.u16 rz, rx, ry psubh.s16 rz, rx, ry
manipulate	Rz[31:16]=(Rx[31:16] - Ry[31:16])/2 Rz[15:0]=(Rx[15:0] - Ry[15:0])/2
Compilation results	Only 32-bit instructions exist. psubh.u16 rz, rx, ry psubh.s16 rz, rx, ry

Description:	Subtract the upper and lower halves of Rx from the upper and lower halves of Ry, and divide the result by 2 and store it in Rz.
Impact flag bits:	unaffected
Exception:	not have

32-bit instruction	
Operation:	Rz[31:16]=(Rx[31:16] - Ry[31:16])/2 Rz[15:0]=(Rx[15:0] - Ry[15:0])/2
Grammar:	psubh.u16 rz, rx, ry psubh.s16 rz, rx, ry
Description:	Subtract the upper and lower halves of Rx from the upper and lower halves of Ry, and divide the result by 2 and store it in Rz.
Impact flag bits:	unaffected
Exception:	not have

Command Format:

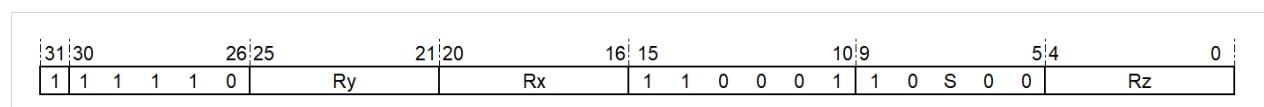


Figure 15.15: PSUBH.(US)16

15.16 SUBH.(U/S)32 - 32-bit (no/have) signed subtraction averaging instruction

Harmonization Directive	
grammatical	subh.u32 rz, rx, ry subh.s32 rz, rx, ry
manipulate	$Rz[31:0] = (Rx[31:0] - Ry[31:0])/2$
Compilation results	Only 32-bit instructions exist. subh.u32 rz, rx, ry subh.s32 rz, rx, ry

Description:	Rx is subtracted from Ry and the result is divided by 2 and stored in Rz.
Impact flag bits:	unaffected
Exception:	not have

32-bit instruction	
Operation:	$Rz[31:0] = (Rx[31:0] - Ry[31:0])/2$
Grammar:	subh.u32 rz, rx, ry subh.s32 rz, rx, ry
Description:	Rx is subtracted from Ry and the result is divided by 2 and stored in Rz.
Impact flag bits:	unaffected
Exception:	not have

Command Format:

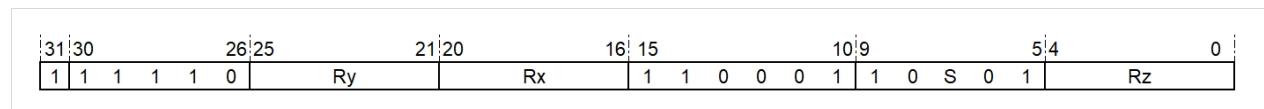


Figure 15.16: SUBH.(US)32

15.17 PASX.16--16 Bit Parallel Cross Add and Subtract Instructions

User's Manual

Harmonization Directive	
grammatical	pasx.16 rz, rx, ry
manipulate	$Rz[31:16] = Rx[31:16] + Ry[15:0]$ $Rz[15:0] = Rx[15:0] - Ry[31:16]$
Compilation results	Only 32-bit instructions exist. pasx.16 rz, rx, ry

Description:	The high 16 bits of Rx are added to the low 16 bits of Ry and the result is stored in the high 16 bits of Rz; The lower 16 bits of Rx are subtracted from the upper 16 bits of Ry and the result is stored in the lower 16 bits of Rz.
impact indicator	unaffected
Chi Bit:	
Exception:	not have

32-bit finger honorific title	
Operation:	Rz[31:16]= Rx[31:16]+ Ry[15:0] Rz[15:0]= Rx[15:0] - Ry[31:16]
Grammar:	pasx.16 rz, rx, ry
Description:	The high 16 bits of Rx are added to the low 16 bits of Ry and the result is stored in the high 16 bits of Rz; The lower 16 bits of Rx are subtracted from the upper 16 bits of Ry and the result is stored in the lower 16 bits of Rz.
impact indicator	unaffected
Chi Bit:	
Exception:	not have

Command Format:

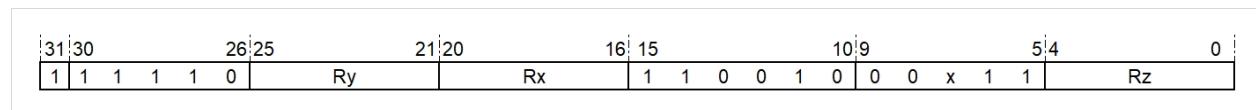


Figure 15.17: PASX.16

15.18 PSAX.16 - 16 bit parallel cross-subtract-add instructions

User's Manual	Harmonization Directive	
	grammatical	psax.16 rz, rx, ry
	manipulate	Rz[31:16]= Rx[31:16] - Ry[15:0] Rz[15:0]= Rx[15:0]+ Ry[31:16]
	Compilation results	Only 32-bit instructions exist. psax.16 rz, rx, ry

Description:	The high 16 bits of Rx are subtracted from the low 16 bits of Ry and the result is stored in the high 16 of Rz; The lower 16 bits of Rx are added to the upper 16 bits of Ry and the result is stored in the lower 16 bits of Rz.
impact indicator	unaffected
Chi Bit:	
Exception:	not have

32-bit finger honorific title	
Operation:	$Rz[31:16] = Rx[31:16] - Ry[15:0]$ $Rz[15:0] = Rx[15:0] + Ry[31:16]$
Grammar:	psax.16 rz, rx, ry
Description:	The high 16 bits of Rx are subtracted from the low 16 bits of Ry and the result is stored in the high 16 of Rz; The lower 16 bits of Rx are added to the upper 16 bits of Ry and the result is stored in the lower 16 bits of Rz.
Impact indicator Chi Bit:	unaffected
Exception:	not have

Command Format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 1 1 0	Ry	Rx	1 1 0 0 1 1 0 0 x 1 1	Rz		

Figure 15.18: PSAX.16

15.19 PASX.(U/S)16.S - 16-bit parallel (no/have) symbol intersection with saturation operation

Fork add and subtract commands

Harmonization Directive	
grammatical	pasx.u16.s rz, rx, ry pasx.s16.s rz, rx, ry
manipulate	$Rz[31:16] = \text{saturate}(Rx[31:16] + Ry[15:0])$ $Rz[15:0] = \text{saturate}(Rx[15:0] - Ry[31:16])$

User compilation results	Only 32-bit instructions exist. pasx.u16.s rz, rx,ry pasx.s16.s rz, rx,ry
--------------------------	---

Exp lana tion :	<p>The high 16 bits of Rx are added to the low 16 bits of Ry and the result is saturated and stored in the high 16 of Rz;</p> <p>The lower 16 bits of Rx are subtracted from the upper 16 bits of Ry and the result is saturated and stored in the lower 16 of Rz.</p> <p>For unsigned numbers, saturation is handled as follows: if the 16-bit addition/subtraction result is greater than the upper saturation value of 0xFFFF, the result is taken as 0xFFFF, if the 16-bit addition/subtraction result is less than the lower saturation value of 0x0, the result is taken as 0x0, and in all other cases, the result is taken by itself.</p> <p>For signed numbers, saturation is handled as follows: if the 16-bit addition/subtraction result is greater than the upper saturation value of 0x7FFF, the result is taken as 0x7FFF, the</p> <p>If the 16-bit addition/subtraction result is less than the lower saturation value of 0x8000, the result is taken as 0x8000, otherwise the result itself is taken.</p>
M ar k er s of in fl u e n ce Bit:	unaffected
disc rimi nate Reg ular :	not have

32 in d e x (m at h.) hon orifi c	
---	--

User's Manual	
drill (practic e) Ma ke:	Rz[31:16]= saturate(Rx[31:16]+ Ry[15:0]) Rz[15:0]= saturate(Rx[15:0] - Ry[31:16])
tell to Law :	pasx.u16.s rz, rx,ry pasx.s16.s rz, rx,ry
Explana tion :	The high 16 bits of Rx are added to the low 16 bits of Ry and the result is saturated and stored in the high 16 of Rz; The lower 16 bits of Rx are subtracted from the upper 16 bits of Ry and the result is saturated and stored in the lower 16 of Rz. For unsigned numbers, saturation is handled as follows: if the 16-bit addition/subtraction result is greater than the upper saturation value of 0xFFFF, the result is taken as 0xFFFF, if the 16-bit addition/subtraction result is less than the lower saturation value of 0x0, the result is taken as 0x0, and in all other cases, the result is taken by itself. For signed numbers, saturation is handled as follows: if the 16-bit addition/subtraction result is greater than the upper saturation value of 0x7FFF, the result is taken as 0x7FFF, the If the 16-bit addition/subtraction result is less than the lower saturation value of 0x8000, the result is taken as 0x8000, otherwise the result itself is taken.
Mark ers of infl uen ce Bit:	unaffected
disc rimi nate Reg ular :	not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 1 0 0 1 0 0 1 S 1 1	Rz		

Figure 15.19: PASX.(US)16.

15.20 PSAX.(U/S)16.S - 16-bit parallel (no/have) symbol intersection with saturation operation

Fork minus plus command

Harmonization Directive	
grammatical	psax.u16.s rz, rx,ry psax.s16.s rz, rx,ry
manipulate	Rz[31:16]= saturate(Rx[31:16] - Ry[15:0]) Rz[15:0]= saturate(Rx[15:0]+ Ry[31:16])
Compilation results	Only 32-bit instructions exist. psax.u16.s rz, rx,ry psax.s16.s rz, rx,ry

Explanation:	The high 16 bits of Rx are subtracted from the low 16 bits of Ry and the result is saturated and stored in the high 16 of Rz; The lower 16 bits of Rx are added to the upper 16 bits of Ry and the result is saturated and stored in the lower 16 bits of Rz. For unsigned numbers, saturation is handled as follows: if the 16-bit addition/subtraction result is greater than the upper saturation value of 0xFFFF, the result is taken as 0xFFFF, if the 16-bit addition/subtraction result is less than the lower saturation value of 0x0, the result is taken as 0x0, and in all other cases, the result is taken by itself. For signed numbers, saturation is handled as follows: if the 16-bit addition/subtraction result is greater than the upper saturation value of 0x7FFF, the result is taken as 0x7FFF, the If the 16-bit addition/subtraction result is less than the lower saturation value of 0x8000, the result is taken as 0x8000, otherwise the result itself is taken.
Markers of influence	unaffected

User's ce Bit:	Manual
disc rimi nate Reg ular :	not have

32 in d e x (m at h.) hon orifi c title	
drill (prac tic e) Ma ke:	Rz[31:16]= saturate(Rx[31:16] - Ry[15:0]) Rz[15:0]= saturate(Rx[15:0]+ Ry[31:16])
tell to Law :	psax.ul6.s rz, rx, ry psax.s16.s rz, rx, ry
Exp lana tion : :	For unsigned numbers, saturation is handled as follows: if the 16-bit addition/subtraction result is greater than the upper saturation value of 0xFFFF, the result is taken as 0xFFFF, if the 16-bit addition/subtraction result is less than the lower saturation value of 0x0, the result is taken as 0x0, and in all other cases, the result is taken by itself. For signed numbers, saturation is handled as follows: if the 16-bit addition/subtraction result is greater than the upper saturation value of 0x7FFF, the result is taken as 0x7FFF, the If the 16-bit addition/subtraction result is less than the lower saturation value of 0x8000, the result is taken as 0x8000, otherwise the result itself is taken.
M ar k er s of in fl u e n ce Bit:	unaffected

disc	Method	have
rimi		
nate		
Reg		
ular		
:		

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 1 0 0 1 1 0 1 S 1 1	Rz		

Figure 15.20: PSAX.(US)16.

15.21 PASXH.(U/S)16 - 16-bit parallel (no/have) signed cross-add and subtract averaging value instruction

Harmonization Directive	
grammatical	pasxh.u16 rz, rx, ry pasxh.s16 rz, rx, ry
manipulate	Rz[31:16] = (Rx[31:16] + Ry[15:0])/2 Rz[15:0] = (Rx[15:0] - Ry[31:16])/2
Compilation results	Only 32-bit instructions exist. pasxh.u16 rz, rx, ry pasxh.s16 rz, rx, ry

Description:	The high 16 bits of Rx are added to the low 16 bits of Ry and the result is divided by 2 and stored in the high 16 of Rz; The lower 16 bits of Rx minus the upper 16 bits of Ry are subtracted and the result is divided by 2 and stored in the lower 16 of Rz.
impact indicator Chi Bit:	unaffected
Exception:	not have

32-bit finger honorific title	
Operation:	$Rz[31:16] = (Rx[31:16] + Ry[15:0])/2$ $Rz[15:0] = (Rx[15:0] - Ry[31:16])/2$
Grammar:	pasxh.u16 rz, rx, ry pasxh.s16 rz, rx, ry
Description:	The high 16 bits of Rx are added to the low 16 bits of Ry and the result is divided by 2 and stored in the high 16 of Rz; The lower 16 bits of Rx minus the upper 16 bits of Ry are subtracted and the result is divided by 2 and stored in the lower 16 of Rz.
impact indicator Chi Bit:	unaffected
Exception:	not have

Command Format:

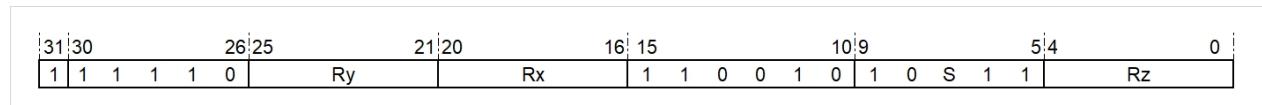


Figure 15.21: PASXH.(US)16

15.22 PSAXH.(U/S)16 - 16-bit parallel (no/have) signed cross-subtraction averaging

value instruction

Harmonization Directive	
grammatical	psaxh.u16 rz, rx, ry psaxh.s16 rz, rx, ry
manipulate	Rz[31:16] = (Rx[31:16] - Ry[15:0])/2 Rz[15:0] = (Rx[15:0] + Ry[31:16])/2
Compilation results	Only 32-bit instructions exist. psaxh.u16 rz, rx, ry psaxh.s16 rz, rx, ry

Description:	The high 16 bits of Rx are subtracted from the low 16 bits of Ry and the result is divided by 2 and stored in the high 16 of Rz; The lower 16 bits of Rx minus the upper 16 bits of Ry are added together and the result is divided by 2 and stored in the lower 16 of Rz.
impact indicator Chi Bit:	unaffected
Exception:	not have

32-bit finger honorific title	
Operation:	Rz[31:16]=(Rx[31:16] - Ry[15:0])/2 Rz[15:0]=(Rx[15:0]+ Ry[31:16])/2
Grammar:	psaxh.u16 rz, rx, ry psaxh.s16 rz, rx, ry
Description:	The high 16 bits of Rx are subtracted from the low 16 bits of Ry and the result is divided by 2 and stored in the high 16 of Rz; The lower 16 bits of Rx minus the upper 16 bits of Ry are added together and the result is divided by 2 and stored in the lower 16 of Rz.
impact indicator Chi Bit:	unaffected
Exception:	not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 1 0 0 1 1 0 S 1 1	Rz		

Figure 15.22: PSAXH.(US)16

15.23 ADD.64 - 64-bit addition instructions

Harmoniza

tion Directive	
grammatical	add.64 rz, rx,ry
manipulate	{Rz+1[31:0], Rz[31:0]}= {Rx+1[31:0], Rx[31:0]}+ {Ry+1[31:0], Ry[31:0]}
Compilation results	Only 32-bit instructions exist. add.64 rz, rx,ry

User's Manual

Explana tion: :	Rx+1 and Rx form a 64-bit number (where Rx+1 is the high 32 bits and Rx is the low 32 bits of data), Ry+1 and Ry Compose a 64-bit number (where Rx+1 is the high 32-bit and Rx is the low 32-bit data)and perform a 64-bit addition. The high 32 bits of the Rz+1 and the low 32 bits of the Rz are stored in the Rz+1 and Rz+1 respectively.
Marker s of infl uen ce Bit:	unaffected
descri minat e Regul ar:	not have

32 inde x (mat h.) honor ific title	
drill (pract ice) Make :	{Rz+1[31:0], Rz[31:0]}= {Rx+1[31:0], Rx[31:0]}+ {Ry+1[31:0], Ry[31:0]}
tell to Law:	add.64 rz, rx,ry
Explana tion: :	Rx+1 and Rx form a 64-bit number (where Rx+1 is the high 32 bits and Rx is the low 32 bits of data), Ry+1 and Ry Compose a 64-bit number (where Rx+1 is the high 32-bit and Rx is the low 32-bit data)and perform a 64-bit addition. The high 32 bits of the Rz+1 and the low 32 bits of the Rz are stored in the Rz+1 and Rz+1 respectively.
Marker s of	unaffected

inf	User's Manual
ue	
ce	
Bit:	
discri	not have
minat	
e	
Regul	
ar:	

Command Format:

:31:30	26:25	21:20	16:15	10:9	5:4	0
[1 1 1 1 0]	Ry	Rx	[1 1 0 0 0 0 0 0]	x	[1 1]	Rz

Figure 15.23: ADD.64

15.24 ADD.(U/S)64.S - 64-bit (no/have) signed addition instruction with saturation operation

Harmonization Directive	
grammatical	add.u64.s rz, rx,ry add.s64.s rz, rx,ry
manipulate	{Rz+1[31:0], Rz[31:0]}= Saturate({Rx+1[31:0], Rx[31:0]}+ {Ry+1[31:0], Ry[31:0]})
Compilation results	Only 32-bit instructions exist. add.u64.s rz, rx,ry add.s64.s rz, rx,ry

32 in d e x (m at h.) hon orific title	
drill (practic e) Ma ke:	{Rz+1[31:0], Rz[31:0]}= Saturate({Rx+1[31:0], Rx[31:0]}+ {Ry+1[31:0], Ry[31:0]})
tell to Law :	add.u64.s rz, rx,ry add.s64.s rz, rx,ry

Exp User Manual lana tion : Bit:	<p>Rx+1 and Rx form a 64-bit number (where Rx+1 is the high 32-bit and Rx is the low data) and Ry+1 and Ry form a 64-bit number (where Rx+1 is the high 32-bit and Rx is the low 32-bit data), and perform a 64-bit addition, and then the result of the saturation process, the high 32-bit is deposited into Rz+1 and the low 32-bit is deposited into Rz.</p> <p>For unsigned numbers, saturation is handled by taking the result 0xFFFF FFFF FFFF FFFF if the result of the addition is greater than the saturation value 0xFFFF FFFF FFFF FFFF, and taking the result itself in all other cases.</p> <p>For signed numbers, the saturation process is as follows: if the addition result is greater than the upper saturation value 0x7FFF FFFF FFFF FFFF, the result is taken as 0x7FFF FFFF FFFF FFFF, and if the addition result is less than the lower saturation value 0x8000 0000 0000 0000, the result is taken as 0x8000 0000 0000 0000, otherwise take the result itself.</p>
M ar k er s of in fl u e n ce Bit:	unaffected
disc rimi nate Reg ular : Bit:	not have

Command Format:

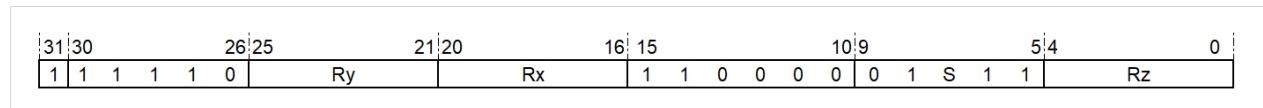


Figure 15.24: ADD.(US)64.

15.25 SUB.64 - 64-bit Subtract Instruction

Harmonization Directive	
grammatical	sub.64 rz, rx, ry
manipulate	$\{Rz+1[31:0], Rz[31:0]\} = \{Rx+1[31:0], Rx[31:0]\} - \{Ry+1[31:0], Ry[31:0]\}$
Compilation results	Only 32-bit instructions exist. sub.64 rz, rx, ry

Explanation :	Rx+1 and Rx form a 64-bit number (where Rx+1 is the high 32 bits and Rx is the low 32 bits of data), Ry+1 and Ry Compose a 64-bit number (where Rx+1 is the high 32-bit and Rx is the low 32-bit data) perform a 64-bit subtraction, and the result is The high 32 bits of the Rz+1 and the low 32 bits of the Rz are stored in the Rz+1 and Rz+1 respectively.
Markers of influence Bit:	unaffected
discriminative Regular:	not have

32 index (matrix.) honorific title	
drill (practice) Make	$\{Rz+1[31:0], Rz[31:0]\} = \{Rx+1[31:0], Rx[31:0]\} - \{Ry+1[31:0], Ry[31:0]\}$

User's Manual	
tell to Law:	sub.64 rz, rx,ry
Explanation :	Rx+1 and Rx form a 64-bit number (where Rx+1 is the high 32 bits and Rx is the low 32 bits of data), Ry+1 and Ry Compose a 64-bit number (where Rx+1 is the high 32-bit and Rx is the low 32-bit data) perform a 64-bit subtraction, and the result is The high 32 bits of the Rz+1 and the low 32 bits of the Rz are stored in the Rz+1 and Rz+1 respectively.
Marker s of infl uence Bit:	unaffected
discri minat e Regular:	not have

Command Format:

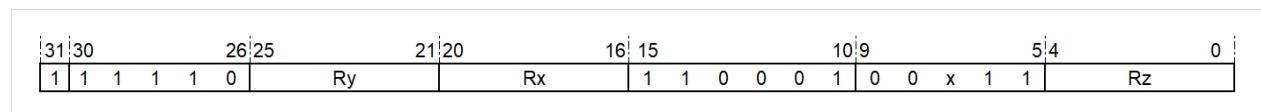


Figure 15.25: SUB.64

15.26 SUB.(U/S)64.S - 64-bit (no/have) signed subtraction instruction with saturation operation

Harmonization Directive	
grammatical	sub.u64.s rz, rx,ry sub.s64.s rz, rx,ry
manipulate	{Rz+1[31:0], Rz[31:0]} = Saturate({Rx+1[31:0], Rx[31:0]} - {Ry+1[31:0], Ry[31:0]})
Compilation results	Only 32-bit instructions exist. sub.u64.s rz, rx,ry sub.s64.s rz, rx,ry

32 in d e x (m at h.) hon orifi c title	
drill (practic e) Ma ke:	{Rz+1[31:0], Rz[31:0]} = Saturate({Rx+1[31:0], Rx[31:0]} - {Ry+1[31:0], Ry[31:0]})
tell to Law :	sub.u64.s rz, rx,ry sub.s64.s rz, rx,ry

Exp User Manual lana tion :	<p>Rx+1 and Rx form a 64-bit number (where Rx+1 is the high 32-bit and Rx is the low data) and Ry+1 and Ry form a 64-bit number (where Rx+1 is the high 32-bit and Rx is the low 32-bit data), and perform a 64-bit subtraction, and the result of the subtraction is saturated so that the high 32-bit is deposited into Rz+1 and the low 32-bit is deposited into Rz.</p> <p>For unsigned numbers, saturation is handled by taking 0x0 for the result if the subtraction result is less than the saturation value of 0x0, and taking the result itself in all other cases.</p> <p>For signed numbers, the saturation process is as follows: if the result of subtraction is greater than the upper saturation value 0x7FFF FFFF FFFF FFFF, the result is taken as 0x7FFF FFFF FFFF, and if the result of subtraction is less than the lower saturation value 0x8000 0000 0000 0000, the result is taken as 0x8000 0000 0000, otherwise take the result itself.</p>
M ar k er s of in fl u e n ce Bit:	unaffected
disc rimi nate Reg ular :	not have

Command Format:

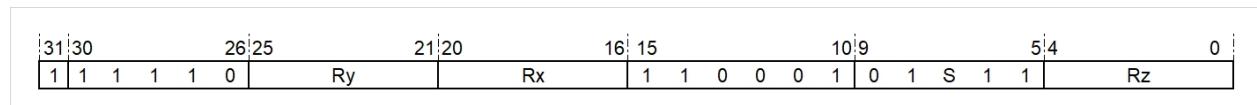


Figure 15.26: SUB.(US)64.

15.27 ASRI.S32.R - Immediate number arithmetic right shift instruction with rounding operation

Harmonization Directive	
grammatical	asri.s32.r rz, rx, oimm5
manipulate	Rz[31:0] <- Round(Rx[31:0] >>>oimm[4:0])
Compilation results	Only 32-bit instructions exist. asri.s32.r rz, rx, oimm5

Explanation:	The value of Rx is arithmetically shifted to the right (the original value is shifted to the right, and the left side is shifted into a copy of the original sign bit) and a rounding operation is done. The process of rounding operation is that if the highest bit of the shifted data is 1, the shift result is added 1, otherwise the shift result remains unchanged. The number of right shifted bits is determined by the number of bits with bias 1. The value of the 5-bit immediate number (OIMM5) is determined.
Impact Sign Bit:	unaffected
set a limit (on) System:	Immediate numbers range from 1-32.
discretionary Regular:	not have

32 Bit honorific title	
drill (practice)	Rz[31:0] <- Round(Rx[31:0] >>>oimm[4:0])

Make	Manual
:	
tell to	asri.s32.r rz, rx, oimm5
Law:	
Exp lana tion :	The value of Rx is shifted arithmetically right (the original value is shifted right, and the left side is shifted into a copy of the original sign bit) and a rounding operation is done. The process of rounding operation is that if the highest bit of the shifted data is 1, the shift result is added 1, otherwise the shift result remains unchanged. The number of right shifted bits is determined by the number of bits with bias 1. The value of the 5-bit immediate number (OIMM5) is determined.
Imp act Sign s Bit:	unaffected
set a limit (on) Syste m:	Immediate numbers in the range 1-32
discri minat e Regul ar:	not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Oimm5	Rx	1 1 0 1 0 0	0 1 1 0 1	Rz	

Figure 15.27: ASR.S32.

15.28 ASR.S32.R - Arithmetic right shift instruction with rounding operation

Harmonization Directive	
grammatical	asr.s32.r rz, rx, ry
manipulate	Rz[31:0] <- Round(Rx[31:0]>>> Ry[5:0])
Compilation results	Only 32-bit instructions exist. asr.s32.r rz, rx, ry

Description:	The value of Rx is arithmetically shifted to the right (the original value is shifted to the right, and the left side is shifted into a copy of the original sign bit), and a rounding operation is done. The process of rounding operation is that if the highest bit of the shifted data is 1, the result of shifting is added 1, otherwise the result of shifting remains unchanged. The number of right shifted bits is determined by Ry[5:0]. Definition.
impact Logo Bit:	unaffected
Exception:	not have

32-bit directives	
Operation:	Rz[31:0] <- Round(Rx[31:0]>>> Ry[5:0])
Grammar:	asr.s32.r rz, rx, ry
Description:	The value of Rx is shifted arithmetically right (the original value is shifted right, and the left side is shifted into a copy of the original sign bit), and a rounding operation is done. The process of rounding operation is that if the highest bit of the shifted data is 1, the result of shifting is added 1, otherwise the result of shifting remains unchanged. The number of right shifted bits is determined by Ry[5:0].

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Oimm5	Rx	1 1 0 1 0 0	0 1 1 0 1	Rz	

	Definition.
impact Logo Bit:	unaffected
Except ion:	not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Oimm5	Rx	1 1 0 1 0 0	0 1 1 0 1	Rz	

Figure 15.28: ASR.S32.

15.29 LSRI.U32.R - Immediate Logical Right Shift Instruction with Rounding Operation

Harmonization Directive	
grammatical	lsri.u32.r rz, rx, oimm5
manipulate	Rz[31:0] <- Round(Rx[31:0]>> oimm[4:0])
Compilation results	Only 32-bit instructions exist. lsri.u32.r rz, rx, oimm5

Description:	The value of Rx is logically shifted to the right (the original value is shifted to the right and the left side is complemented by 0), and a rounding operation is performed. The process of rounding operation is that if the highest bit of the shifted data is 1, the result of shifting is added 1, otherwise the result of shifting remains unchanged. The number of bits shifted to the right consists of a 5-bit immediate number with a bias of 1. (The value of (OIMMS) is determined.)
Impact Signs Bit:	unaffected
Limitations:	Immediate numbers range from 1-32.
Exception:	not have

32-bit directives	
Operation:	Rz[31:0] <- Round(Rx[31:0] >>oimm[4:0])
Grammatical:	lsri.u32.r rz, rx, oimm5

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Oimm5	Rx	1 1 0 1 0 0	0 1 1 0 1	Rz	

Description:	The value of Rx is logically shifted to the right (the original value is shifted to the right and the left side is complemented by 0), and a rounding operation is performed. The process of rounding operation is that if the highest bit of the shifted data is 1, the result of the shift is added 1, otherwise the result of the shift remains unchanged. The number of bits shifted to the right consists of a 5-bit immediate number with a bias of 1. (The value of (OIMM5) is determined.
Impact Sign s Bit:	unaffected
Limitations:	Immediate numbers in the range 1-32
Exception:	not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Oimm5	Rx	1 1 0 1 0 0	0 1 1 0 1	Rz	

Figure 15.29: LSRI.U32.

15.30 LSR.U32.R - Logical right shift instruction with rounding operation

Harmonization Directive	
grammatical	lsr.u32.r rz, rx, ry
manipulate	Rz[31:0] <- Round(Rx[31:0]>> Ry[5:0])
Compilation results	Only 32-bit instructions exist. lsr.u32.r rz, rx, ry

Description:	The value of Rx is logically shifted to the right (the original value is shifted to the right and the left hand side is complemented by 0) and a rounding operation is done. The process of rounding is as follows: if the number shifted out is If the highest bit of the data is 1, the shift result is increased by 1, otherwise the shift result remains unchanged. The right shift number is determined by Ry[5:0].
impact	unaffected
Logo Bit:	
Exception:	not have

32-bit directives	
Operation:	Rz[31:0] <- Round(Rx[31:0]>> Ry[5:0])
Grammatical:	lsr.u32.r rz, rx, ry
Description:	The value of Rx is logically shifted to the right (the original value is shifted to the right and the left hand side is complemented by 0) and a rounding operation is done. The process of rounding is as follows: if the number shifted out is If the highest bit of the data is 1, the shift result is increased by 1, otherwise the shift result remains unchanged. The right shift number is determined by Ry[5:0].

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Oimm5	Rx	1 1 0 1 0 0	0 1 1 0 1	Rz	

impact	unaffected
Logo	
Bit:	
Exception:	not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 1 0 1 0 0	1 1 0 1 1	Rz	

Figure 15.30: LSR.U32.

15.31 LSLI.(U/S)32.S - Immediate number with saturation operation (without/with) signed logic left

shift command (computing)

Harmonization Directive	
grammatical	lsli.u32.s rz, rx, oimm5 lsli.s32.s rz, rx, oimm5
manipulate	Rz[31:0] <- Saturate(Rx[31:0]<< oimm[4:0])
Compilation results	Only 32-bit instructions exist. lsli.u32.s rz, rx, oimm5 lsli.s32.s rz, rx, oimm5

Explanation:	The value of Rx is logically shifted left (original value left, right complement 0) and a saturation operation is done. The number of left shift bits is determined by the value of the 5-bit immediate number with bias 1 (OIMM5). For unsigned numbers, saturation is handled by taking the result 0xFFFF FFFF if the shift result is greater than the saturation value 0xFFFF FFFF, and taking the result itself in all other cases. For signed numbers, the saturation processing is as follows, if the shift result is greater than the upper saturation value 0x7FFF FFFF, the result is taken as 0x7FFF FFFF, if the shift result is less than the lower saturation value 0x8000 0000, the result is taken as 0x8000 0000, otherwise the result itself is taken.
Marker Bit:	unaffected
set a limit (on) System:	Immediate numbers range from 1-32.

disc	Method
rimi	
nate	
Reg	
ular	
:	

32 in d e x (m at h.) hon orifi c title	
drill (prac tic e) Ma ke:	Rz[31:0] <- Saturate(Rx[31:0]<< oimm[4:0])
tell to Law :	lsli.u32.s rz, rx, oimm5 lsli.s32.s rz, rx, oimm5
Exp lana tion : :	The value of Rx is logically shifted left (original value left, right complement 0) and a saturation operation is done. The number of left shift bits is determined by the value of the 5-bit immediate number with bias 1 (OIMM5). For unsigned numbers, saturation is handled by taking the result 0xFFFF FFFF if the shift result is greater than the saturation value 0xFFFF FFFF, and taking the result itself in all other cases. For signed numbers, the saturation processing is as follows, if the shift result is greater than the upper saturation value 0x7FFF FFFF, the result is taken as 0x7FFF FFFF, if the shift result is less than the lower saturation value 0x8000 0000, the result is taken as 0x8000 0000, otherwise the result itself is taken.
M ar k er s of in fl u e n ce Bit:	unaffected

User	Set a limit (on) System:	Immediate numbers in the range 1-32
disc rimi nate Reg ular :	not have	

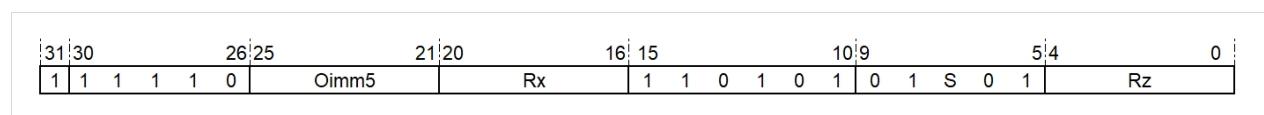
Command Format:

Figure 15.31: LSLI.(US)32.

15.32 LSL.(U/S)32.S - (no/have) signed logical left shift instruction with saturation operation

Harmonization Directive	
grammatical	lsl.u32.s rz, rx, ry lsl.s32.s rz, rx, ry
manipulate	Rz[31:0] <- Saturate(Rx[31:0]<< Ry[5:0])
Compilation results	Only 32-bit instructions exist. lsl.u32.s rz, rx, ry lsl.s32.s rz, rx, ry

Explanation:	The value of Rx is logically shifted left (the original value is shifted left and the right hand side is complemented with 0) and a saturation operation is done. The number of left shift bits is determined by the value of Ry[5:0]. For unsigned numbers, saturation is performed as follows: if the result of the shift is greater than the saturation value 0xFFFF FFFF, the result is taken as 0xFFFF FFFF, otherwise the result is taken as the result itself. For signed numbers, the saturation processing is as follows, if the shift result is greater than the upper saturation value 0x7FFF FFFF, the result is taken as 0x7FFF FFFF, if the shift result is less than the lower saturation value 0x8000 0000, the result is taken as 0x8000 0000, otherwise the result itself is taken.
Markers of influence:	unaffected
discriminate Regular:	not have

32 in d e x (m at h.) hon orifi c title	
drill (prac tic e) Ma ke:	Rz[31:0] <- Saturate(Rx[31:0]<< Ry[5:0])
tell to Law :	lsl.u32.s rz, rx, ry lsl.s32.s rz, rx, ry
Exp lana tion : :	The value of Rx is logically shifted left (the original value is shifted left and the right hand side is complemented with 0) and a saturation operation is done. The number of left shift bits is determined by the value of Ry[5:0]. For unsigned numbers, saturation is performed as follows: if the result of the shift is greater than the saturation value 0xFFFF FFFF, the result is taken as 0xFFFF FFFF, otherwise the result is taken as the result itself. For signed numbers, the saturation processing is as follows, if the shift result is greater than the upper saturation value 0x7FFF FFFF, the result is taken as 0x7FFF FFFF, if the shift result is less than the lower saturation value 0x8000 0000, the result is taken as 0x8000 0000, otherwise the result itself is taken.
M ar k er s of in fl u e n ce Bit:	unaffected

disc	Method	have
rimi		
nate		
Reg		
ular		
:		

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 1 0 1 0 1 0 1 S 1 1	Rz		

Figure 15.32: LSL.(US)32.

15.33 PASRI.S16 - 16 bit parallel immediate number arithmetic right shift instruction

Harmonization Directive	
grammatical	pasri.s16 rz, rx, oimm4
manipulate	Rz[31:16] <- Rx[31:16]>>> oimm[3:0] Rz[15:0] <- Rx[15:0]>>> oimm[3:0]
Compilation results	Only 32-bit instructions exist. pasri.s16 rz, rx, oimm4

Description:	The high and low half words of Rx are arithmetically shifted to the right (the original value is shifted to the right, and the left side is shifted into a copy of the original sign bit), and the results of the shifting are stored in the The high and low half words of Rz. The right shift number is determined by the value of the 4-bit immediate number with bias 1 (OIMM4).
impact Logo Bit:	unaffected
Limitations:	Immediate numbers in the range 1-16
Exception:	not have

32 bits directives	
Operation:	Rz[31:16] <- Rx[31:16]>>> oimm[3:0] Rz[15:0] <- Rx[15:0]>>> oimm[3:0]
Grammar:	pasri.s16 rz, rx, oimm4
Description:	The high and low half words of Rx are arithmetically shifted to the right (the original value is shifted to the right, and the left side is shifted into a copy of the original sign bit), and the results of the shifting are stored in the The high and low half words of Rz. The right shift number is determined by the value of the 4-bit immediate number with bias 1 (OIMM4).
impact Logo Bit:	unaffected
Limitations:	Immediate numbers in the range 1-16
Exception:	not have

Command Format:

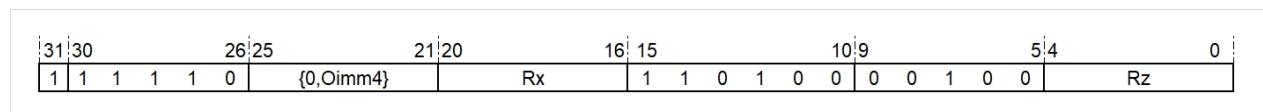


Figure 15.33: PASRI.S16

15.34 PASR.S16 - 16 bit parallel arithmetic right shift instructions

Harmonization Directive	
grammatical	pasr.s16 rz, rx, ry
manipulate	Rz[31:16] <- Rx[31:16]>>> Ry[4:0] Rz[15:0] <- Rx[15:0]>>> Ry[4:0]
Compilation results	Only 32-bit instructions exist. pasr.s16 rz, rx, ry

Description:	The high and low half words of Rx are arithmetically shifted to the right (the original value is shifted to the right, and the left side is shifted into a copy of the original sign bit), and the results of the shifting are stored in the The high and low half words of Rz. The number of right shift bits is determined by the value of Ry 5 bits lower (Ry[4:0]).
affect Chi Bit:	unaffected
Limitations:	not have
Exception:	not have

32-bit 32-bit (computing) directives	
Operation:	Rz[31:16] <- Rx[31:16]>>> Ry[4:0] Rz[15:0] <- Rx[15:0]>>> Ry[4:0]
Grammar:	pasr.s16 rz, rx, ry
Description:	The high and low half words of Rx are arithmetically shifted to the right (the original value is shifted to the right, and the left side is shifted into a copy of the original sign bit), and the results of the shifting are stored in the The high and low half words of Rz. The number of right shift bits is determined by the value of Ry 5 bits lower (Ry[4:0]).
affect Chi Bit:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

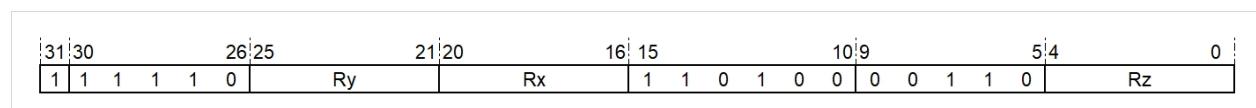


Figure 15.34: PASR.S16

Shift Instruction with Rounding Operation

Harmonization means honorific title	
grammatical	pasri.s16.r rz, rx, oimm4
manipulate	Rz[31:16] <- Round(Rx[31:16]>>> oimm[3:0]) Rz[15:0] <- Round(Rx[15:0]>>> oimm[3:0])
Compilation results	Only 32-bit instructions exist. pasri.s16.r rz, rx, oimm4

Exp lana tion :	The high and low half-words of Rx are shifted arithmetically to the right (the original value is shifted to the right, and the left side is shifted into a copy of the original sign bit), and the shifted result is rounded up and deposited into the high and low half-words of Rz respectively. The rounding operation is as follows: if the highest bit of the shifted data is 1, the shifted result will be added 1, otherwise, the shifted result will be added 1, and the shifted result will be added 1. The bit result remains unchanged. The number of right shift bits is determined by the value of the 4-bit immediate number with bias 1 (OIMM4).
M ar k er s of in fl u e n ce Bit:	unaffected
set a limit (on) Syste m:	Immediate numbers in the range 1-16
discri mina te Regu lar:	not have

32 in d e x (m at h.) hono rific title	
--	--

drill Users Manual (practice) Make:	Rz[31:16] <- Round(Rx[31:16]>>> oimm[3:0]) Rz[15:0] <- Round(Rx[15:0]>>> oimm[3:0])
tell to Law:	pasri.s16.r rz, rx, oimm4
Exp lana tion :	The high and low half-words of Rx are shifted arithmetically to the right (the original value is shifted to the right, and the left side is shifted into a copy of the original sign bit), and the shifted result is rounded up and deposited into the high and low half-words of Rz respectively. The rounding operation is as follows: if the highest bit of the shifted data is 1, the shifted result will be added 1, otherwise, the shifted result will be added 1, and the shifted result will be added 1. The bit result remains unchanged. The number of right shift bits is determined by the value of the 4-bit immediate number with bias 1 (OIMM4).
M ark er s of in fl u e n ce Bit:	unaffected
set a limit (on) System:	Immediate numbers in the range 1-16
discri mina te Regu lar:	not have

Command Format:

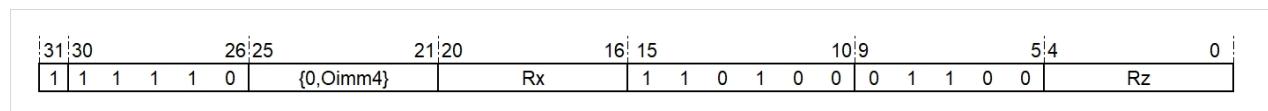


Figure 15.35: PASRI.S16.R

15.36 PASR.S16.R - 16-bit parallel arithmetic right shift instruction with rounding operation

Harmonization Directive	
grammatical	pasr.s16.r Rz, Rx, Ry
manipulate	Rz[31:16] <- Round(Rx[31:16]>>> Ry[4:0]) Rz[15:0] <- Round(Rx[15:0]>>> Ry[4:0])
Compilation results	Only 32-bit instructions exist. pasr.s16.r rz, rx, ry

Explanation:	The high and low half-words of Rx are shifted arithmetically to the right (the original value is shifted to the right, and the left side is shifted into a copy of the original sign bit), and the shifted result is rounded up and deposited into the high and low half-words of Rz respectively. The rounding operation is as follows: if the highest bit of the shifted data is 1, the shifted result will be added 1, otherwise, the shifted result will be added 1, and the shifted result will be added 1. The bit result remains unchanged. The number of right shift bits is determined by the value of Ry 5 bits lower (Ry[4:0]).
Markers of influence Bit:	unaffected
set a limit (on) System:	not have
discriminate Regular:	not have

32 independent matrix (mat)	
-----------------------------	--

User's Manual	
honorific title	
drill (practice)	Rz[31:16] <- Round(Rx[31:16]>>> Ry[4:0]) Rz[15:0] <- Round(Rx[15:0]>>> Ry[4:0])
Make:	
tell to Law:	pasr.s16.r rz, rx, ry
Explana ^{tion} :	The high and low half-words of Rx are shifted arithmetically to the right (the original value is shifted to the right, and the left side is shifted into a copy of the original sign bit), and the shifted result is rounded up and deposited into the high and low half-words of Rz respectively. The rounding operation is as follows: if the highest bit of the shifted data is 1, the shifted result will be added 1, otherwise, the shifted result will be added 1, and the shifted result will be added 1. The bit result remains unchanged. The number of right shift bits is determined by the value of Ry 5 bits lower (Ry[4:0]).
Markers of influence Bit:	unaffected
set a limit (on) System:	not have
discriminate Regular:	not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 1 0 1 0 0	0 1 1 1 0	Rz	

Figure 15.36: PASR.S16.

15.37 PLSRI.U16 - 16-bit Parallel Immediate Logic Right Shift Instructions

Harmonization Directive	
grammatical	plsri.u16 rz, rx, oimm4
manipulate	Rz[31:16] <- Rx[31:16]>> oimm[3:0] Rz[15:0] <- Rx[15:0]>> oimm[3:0]
Compilation results	Only 32-bit instructions exist. plsri.u16 rz, rx, oimm4

Description:	The high and low half words of Rx are logically shifted to the right (the original value is shifted to the right and the left side is complemented by 0), and the result of the shift is deposited into the high and low half words of Rz respectively. The right shift number is determined by the value of the 4-bit immediate number with bias 1 (OIMM4).
affect Chi Bit:	unaffected
Limitations:	Immediate numbers range from 1 to 16.
Exceptions:	not have

32-bit 32-bit (computing) directives	
Operation:	Rz[31:16] <- Rx[31:16]>> oimm[3:0] Rz[15:0] <- Rx[15:0]>> oimm[3:0]
Grammar:	plsri.u16 rz, rx, oimm4

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 1 0 1 0 0	0 1 1 1 0	Rz	

Description:	The high and low half words of Rx are logically shifted to the right (the original value is shifted to the right and the left side is complemented by 0), and the result of the shift is deposited into the high and low half words of Rz respectively. The right shift number is determined by the value of the 4-bit immediate number with bias 1 (OIMM4).
affect Chi Bit:	unaffected
Limitations:	Immediate numbers in the range 1-16
Exception:	not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 1 0 1 0 0	0 1 1 1 0	Rz	

Figure 15.37: PLSR.U16

15.38 PLSR.U16 - 16 bit parallel logic right shift instructions

Harmonization Directive	
grammatical	plsr.u16 rz, rx, ry
manipulate	Rz[31:16] <- Rx[31:16]>> Ry[4:0] Rz[15:0] <- Rx[15:0]>> Ry[4:0]
Compilation results	Only 32-bit instructions exist. plsr.u16 rz, rx, ry

Description:	The high and low half words of Rx are logically shifted to the right (the original value is shifted to the right and the left side is complemented by 0) and the result of the shift is deposited into the high and low half words of Rz respectively. The number of right shift bits is determined by the value of RY 5 bits lower (RY[4:0]).
impact indicator	unaffected
Chi Bit:	
Limitations:	not have
Exception:	not have

32 index (matrix) honorable title	
drill (practice)	Rz[31:16] <- Round(Rx[31:16]>> Ry[4:0]) Rz[15:0] <- Round(Rx[15:0]>> Ry[4:0])

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 1 0 1 0 0	0 1 1 1 0	Rz	

Make:	
tell to Law:	plsr.u16.r rz, rx, ry
Explanation:	<p>The high and low half words of Rx are logically shifted to the right (the original value is shifted to the right and the left side is complemented by 0) and the shifted result is rounded up and deposited into the high and low half words of Rz respectively. The rounding process is that if the highest bit of the shifted data is 1, the shifted result is added 1, otherwise the shifted result remains unchanged.</p> <p>The number of right shift bits is determined by the value of Ry 5 bits lower (Ry[4:0]).</p>
Markers of influence Bit:	unaffected
set a limit (on) System:	not have
discriminate Regular:	not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 1 0 1 0 0 1 0 0 1 0	Rz		

Figure 15.38: PLSR.U16

15.39 PLSRI.U16.R - 16-Bit Parallel Immediate Logical Right Shift Instruction with Rounding Operation

Harmonization Directive	
grammatical	plsri.u16.r rz, rx, oimm4
manipulate	Rz[31:16] <- Round(Rx[31:16]>> oimm[3:0]) Rz[15:0] <- Round(Rx[15:0]>> oimm[3:0])
Compilation results	Only 32-bit instructions exist. plsri.u16.r rz, rx, oimm4

Explanation:	The high and low half words of Rx are logically shifted to the right (the original value is shifted to the right and the left side is complemented by 0) and the shifted result is rounded up and deposited into the high and low half words of Rz respectively. The rounding process is that if the highest bit of the shifted data is 1, the shifted result is added 1, otherwise the shifted result remains unchanged. The right shift number is determined by the value of the 4-bit immediate number with bias 1 (OIMM4).
Markers of influence Bit:	unaffected
set a limit (on) System:	Immediate numbers range from 1 to 16.
discriminate	not have

32 index (mat h.) hono rific title	
drill (prac tice) Mak e:	Rz[31:16] <- Round(Rx[31:16]>> oimm[3:0]) Rz[15:0] <- Round(Rx[15:0]>> oimm[3:0])
tell to Law:	plsri.u16.r rz, rx, oimm4
Exp lana tion : :	The high and low half words of Rx are logically shifted to the right (the original value is shifted to the right and the left side is complemented by 0) and the shifted result is rounded up and deposited into the high and low half words of Rz respectively. The rounding process is that if the highest bit of the shifted data is 1, the shifted result is added 1, otherwise the shifted result remains unchanged. The right shift number is determined by the value of the 4-bit immediate number with bias 1 (OIMM4).
Mar kers of infl uen ce Bit:	unaffected
set a limit (on) Syste m:	Immediate numbers in the range 1-16
discri mina te Regu lar:	not have

Command Format:

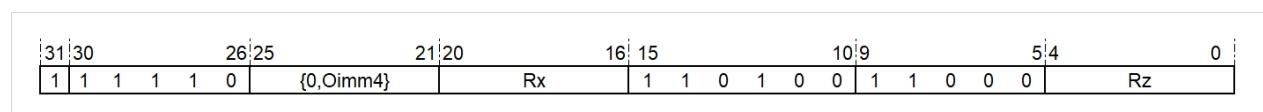


Figure 15.39: PLSR.U16.

15.40 PLSR.U16.R - 16-bit Parallel Logic Right Shift Instruction with Rounding Operation

Harmonization Directive	
grammatical	plsr.u16.r rz, rx, ry
manipulate	Rz[31:16] <- Round(Rx[31:16]>> Ry[4:0]) Rz[15:0] <- Round(Rx[15:0]>> Ry[4:0])
Compilation results	Only 32-bit instructions exist. plsr.u16.r rz, rx, ry

Exp lana tion :	The high and low half words of Rx are logically shifted to the right (the original value is shifted to the right and the left side is complemented by 0) and the shifted result is rounded up and deposited into the high and low half words of Rz respectively. The rounding process is that if the highest bit of the shifted data is 1, the shifted result is added 1, otherwise the shifted result remains unchanged. The number of right shift bits is determined by the value of Ry 5 bits lower (Ry[4:0]).
Marker s of infl uen ce Bit:	unaffected
set a limit (on) Syste m:	not have
discriminat e Regul ar:	not have

32 inde x (mat h.) honor ific title	
drill (pract ice) Make :	Rz[31:16] <- Round(Rx[31:16]>> Ry[4:0]) Rz[15:0] <- Round(Rx[15:0]>> Ry[4:0])
tell to Law:	plsr.u16.r rz, rx, ry
Exp lana tion :	The high and low half words of Rx are logically shifted to the right (the original value is shifted to the right and the left side is complemented by 0) and the shifted result is rounded up and deposited into the high and low half words of Rz respectively. The rounding process is that if the highest bit of the shifted data is 1, the shifted result is added 1, otherwise the shifted result remains unchanged.

The number of right shift bits is determined by the value of Ry 5 bits lower (Ry[4:0]).

Marker s of infl uen ce Bit:	unaffected
set a limit (on) Syste m:	not have
discri minat e Regul ar:	not have

Command Format:

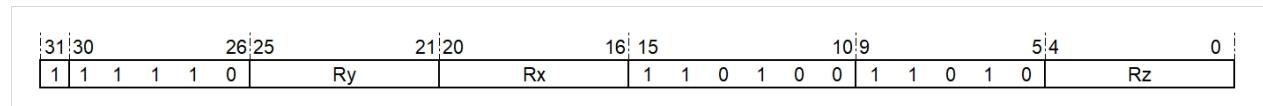


Figure 15.40: PLSR.U16.

15.41 PLSLI.16 - 16-bit Parallel Immediate Logical Left Shift Instruction

Harmonization Directive	
grammatical	plsl.16 rz, rx, oimm4
manipulate	Rz[31:16] <- Rx[31:16]<< oimm[3:0] Rz[15:0] <- Rx[15:0]<< oimm[3:0]
Compilation results	Only 32-bit instructions exist. plsl.16 rz, rx, oimm4

Description:	The high and low half-words of Rx are logically shifted left (original value is shifted left, and the right-hand side is added 0), and the results of the shift are stored in the high and low half-words of Rz. The right shift number is determined by the value of the 4-bit immediate number with bias 1 (OIMM4).
affect Chi Bit:	unaffected
Limitations:	not have
Exceptions:	not have

32-bit 32-bit (computing) directives	
Operation:	Rz[31:16] <- Rx[31:16]<< oimm[3:0] Rz[15:0] <- Rx[15:0]<< oimm[3:0]
Grammar:	plsl.16 rz, rx, oimm4
Description:	The high and low half-words of Rx are logically shifted left (original value is shifted left, and the right-hand side is added 0), and the results of the shift are stored in the high and low half-words of Rz. The right shift number is determined by the value of the 4-bit immediate number with bias 1 (OIMM4).

Affect	affected
User's Manual Chi Bit:	
Limitations:	not have
Exceptions:	not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	{0,0imm4}	Rx	1 1 0 1 0 1 0 0 0 0	0	Rz	

Figure 15.41: PLSLI.16

15.42 PSL1.16 - 16 bit parallel logic left shift instructions

Harmonization Directive	
grammatical	plsl.16 rz, rx, ry
manipulate	Rz[31:16] <- Rx[31:16]<< Ry[4:0] Rz[15:0] <- Rx[15:0]<< Ry[4:0]
Compilation results	Only 32-bit instructions exist. plsl.16 rz, rx, ry

Description:	The high and low half-words of Rx are logically shifted left (original value is shifted left, and the right-hand side is added 0) and the results of the shift are stored in the high and low half-words of Rz. The number of right shift bits is determined by the value of Ry 5 bits lower (Ry[4:0]).
impact indicator	unaffected
Chi Bit:	
Limitations:	not have
Exception:	not have

32-bit finger honorific title	
Operation:	Rz[31:16] <- Rx[31:16]<< Ry[4:0] Rz[15:0] <- Rx[15:0]<< Ry[4:0]
Grammar:	plsl.16 rz, rx, ry
Description:	The high and low half-words of Rx are logically shifted left (original value is shifted left, and the right-hand side is added 0) and the results of the shift are stored in the high and low half-words of Rz. The number of right shift bits is determined by the value of Ry 5 bits lower (Ry[4:0]).
impact indicator	unaffected
Chi Bit:	
Limitations:	not have

Users Manual
Exception:

Command Format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 1 1 1 0 Ry		Rx	1 1 0 1 0 1 0 0 0 1 0 Rz			

Figure 15.42: PSL.16

15.43 PLSLI.(U/S)16.S - 16-bit Parallel Immediate Number with Saturation Operation (Without/With)

Symbolic Logical Left Shift Instruction

Harmonization means honorific title	
grammatical	plsl.s16.s rz, rx, oimm4 plsl.u16.s rz, rx, oimm4
manipulate	Rz[31:16] <- Saturate(Rx[31:16]<< oimm[3:0]) Rz[15:0] <- Saturate(Rx[15:0]<< oimm[3:0])
Compilation results	Only 32-bit instructions exist. plsl.s16.s rz, rx, oimm4 plsl.u16.s rz, rx, oimm4

Explanation:	The high and low half words of Rx are logically left-shifted (original value left-shifted, right-hand side complemented by 0) and saturation operation is performed. The number of left-shifted bits is determined by the number of bits with bias 1 is determined by the value of the 5-bit immediate number (OIMM5). For unsigned numbers, saturation is handled by taking the result to be 0xFFFF if the shift result is greater than the saturation value 0xFFFF, and taking the result itself in all other cases. For signed numbers, the saturation process is as follows: if the shift result is greater than the upper saturation value 0x7FFF, the result is taken as 0x7FFF, and if the shift result is less than the lower saturation value 0x8000, the result is taken as 0x8000, otherwise the result itself is taken.
Mark er s of in fl u e n ce Bit:	unaffected
set a limi t	not have

(en) User's Manual System:	
discriminate Regular :	not have

32 in d e x (m at h.) hon orifi c title	
drill (prac tic e) Ma ke:	Rz[31:16] <- Saturate(Rx[31:16]<< oimm[3:0]) Rz[15:0] <- Saturate(Rx[15:0]<< oimm[3:0])
tell to Law :	plqli.s16.s rz, rx, oimm4 plqli.u16.s rz, rx, oimm4
Explana tion :	The high and low half words of Rx are logically left-shifted (original value left-shifted, right-hand side complemented by 0) and saturation operation is performed. The number of left-shifted bits is determined by the number of bits with bias 1 is determined by the value of the 5-bit immediate number (OIMM5). For unsigned numbers, saturation is handled by taking the result to be 0xFFFF if the shift result is greater than the saturation value 0xFFFF, and taking the result itself in all other cases. For signed numbers, the saturation process is as follows: if the shift result is greater than the upper saturation value 0x7FFF, the result is taken as 0x7FFF, and if the shift result is less than the lower saturation value 0x8000, the result is taken as 0x8000, otherwise the result itself is taken.
Mark er s of in fl u e n ce	unaffected

Bit	Manual
set a limi t (on) Syst em:	Immediate numbers in the range 1-16
disc rimi nate Reg ular :	not have

Command Format:

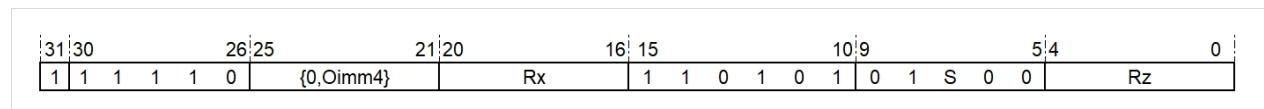


Figure 15.43: PLSLI.(US)16.

15.44 PSL.(U/S)16.S - 16-bit Parallel (Without/With) Symbolic Logic with Saturation Operation

Series Left Shift Command

Harmonization Directive	
grammatical	plsl.s16.s rz, rx, ry plsl.u16.s rz, rx, ry
manipulate	Rz[31:16] <- Saturate(Rx[31:16]<< Ry[4:0]) Rz[15:0] <- Saturate(Rx[15:0]<< Ry[4:0])
Compilation results	Only 32-bit instructions exist. plsl.s16.s rz, rx, ry plsl.u16.s rz, rx, ry

Explanation :	The high and low half words of Rx are logically shifted left (original value is shifted left, and the right hand side is added 0) and saturation operation is performed. The number of bits shifted left is 5 bits below Ry. The value of the bits (Ry[4:0]) is determined. For unsigned numbers, saturation is handled by taking the result to be 0xFFFF if the shift result is greater than the saturation value 0xFFFF, and taking the result itself in all other cases. For signed numbers, the saturation process is as follows: if the shift result is greater than the upper saturation value 0x7FFF, the result is taken as 0x7FFF, and if the shift result is less than the lower saturation value 0x8000, the result is taken as 0x8000, otherwise the result itself is taken.
Marks of influence Bit:	unaffected
set a limit (on)	not have

System:	Manual
discriminate Regular: :	not have

32 in d e x (m at h.) hon orifi c title	
drill (prac tic e) Ma ke:	Rz[31:16] <- Saturate(Rx[31:16]<< Ry[4:0]) Rz[15:0] <- Saturate(Rx[15:0]<< Ry[4:0])
tell to Law :	plsl.s16.s rz, rx, ry plsl.u16.s rz, rx, ry
Explana tion :	The high and low half words of Rx are logically shifted left (original value is shifted left, and the right hand side is added 0) and saturation operation is performed. The number of bits shifted left is 5 bits below Ry. The value of the bits (Ry[4:0]) is determined. For unsigned numbers, saturation is handled by taking the result to be 0xFFFF if the shift result is greater than the saturation value 0xFFFF, and taking the result itself in all other cases. For signed numbers, the saturation process is as follows: if the shift result is greater than the upper saturation value 0x7FFF, the result is taken as 0x7FFF, and if the shift result is less than the lower saturation value 0x8000, the result is taken as 0x8000, otherwise the result itself is taken.
Mar k er s of in fl u e n ce	unaffected

Bit	Manual
set a limit (on) System:	not have
discriminate Regular :	not have

Command Format:

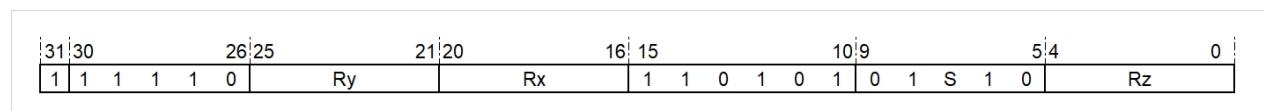


Figure 15.44: PSL.(US)16.

15.45 SEL - bit selection instruction

Harmonization Directive	
grammatical	sel rz, rx, ry, rs
manipulate	for(i=0,i=31;i++) Rz[i]=Rs[i] ? Rx[i] : Ry[i]
Compilation results	Only 32-bit instructions exist. sel rz, rx, ry, rs

Description:	Depending on the value of each bit of the bit selection register Rs, the corresponding bit in Rx and Ry is selected and stored in Rz. If Rs[i]==1 is selected, the bit in Rx and Ry is stored in Rz. Rx[i] is stored in Rz[i], otherwise Ry[i] is selected and stored in Rz[i].
impact indicator	unaffected
Chi Bit:	
Limitations:	not have
Exception:	not have

32-bit finger honorific title	
Operation:	for(i=0;i=31;i++) Rz[i]= Rs[i] ? Rx[i] : Ry[i]
Grammar:	sel rz, rx, ry, rs
Description:	Depending on the value of each bit of the bit selection register Rs, the corresponding bit in Rx and Ry is selected and stored in Rz. If Rs[i]==1 is selected, the bit in Rx and Ry is stored in Rz. Rx[i] is stored in Rz[i], otherwise Ry[i] is selected and stored in Rz[i].
impact indicator	unaffected
Chi Bit:	
Limitations:	not have
Exception:	not have

Command Format:

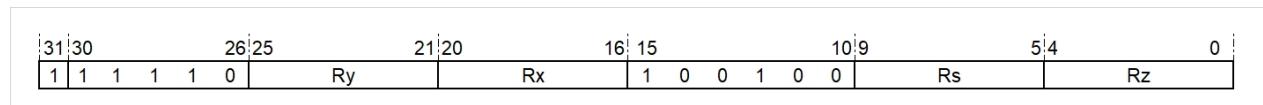


Figure 15.45: SEL

15.46 PCMPNE.8 - 8-bit parallel compare unequal instruction

Harmonization means honorific title	
grammatical	pcmpne.8 rz, rx, ry
manipulate	Rz[31:24]=(Rx[31:24] != Ry[31:24]) ? 8'hFF:8'h00 Rz[23:16]=(Rx[23:16] != Ry[23:16]) ? 8'hFF:8'h00 Rz[15:8]=(Rx[15:8] != Ry[15:8]) ? 8'hFF:8'h00 Rz[7:0]=(Rx[7:0] != Ry[7:0]) ? 8'hFF:8'h00
Translation in the end	Only 32-bit instructions exist. pcmpne.8 rz, rx, ry

Description:	Determine whether 4 bytes of Rx and the 4 bytes Ry are equal or not in terms of bytes. If they are equal, the word corresponding to Rz The result of the section is 0xFF, otherwise the result of the corresponding byte of Rz is 0x0.
affect Chi Bit:	unaffected
Limitations:	not have
Exception:	not have

32-bit directives	
Operation:	Rz[31:24]=(Rx[31:24] != Ry[31:24]) ? 8'hFF:8'h00 Rz[23:16]=(Rx[23:16] != Ry[23:16]) ? 8'hFF:8'h00 Rz[15:8]=(Rx[15:8] != Ry[15:8]) ? 8'hFF:8'h00 Rz[7:0]=(Rx[7:0] != Ry[7:0]) ? 8'hFF:8'h00
Grammar:	pcmpne.8 rz, rx, ry
Description:	Determine whether 4 bytes of Rx and the 4 bytes Ry are equal or not in terms of bytes. If they are equal, the corresponding bytes of Rz The result is 0xFF, otherwise the result is 0x0 for the corresponding byte of Rz.
Impact Signs Bit:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

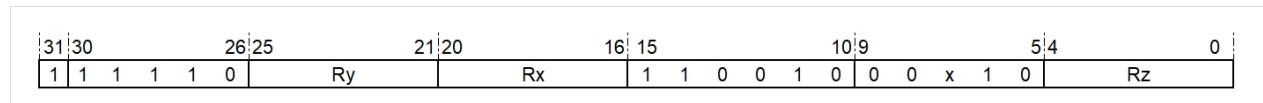


Figure 15.46: PCMPNE.8

15.47 PCMPNE.16 - 16-bit Parallel Compare Unequal

Instruction

Unification directive s	
grammatical	pcmpne.16 rz, rx, ry
manipulate	Rz[31:16]=(Rx[31:16] != Ry[31:16]) ? 16'hFFFF:16'b0000 Rz[15:0]=(Rx[15:0] != Ry[15:0]) ? 16'hFFFF:16'b0000
Translation resolute	Only 32-bit instructions exist. pcmpne.16 rz, rx, ry

Description:	Determine whether the high and low half-words Rx and Ry are equal or not. If they are equal, the result of the corresponding half word of Rz is 0xFFFF, otherwise the result of the corresponding byte of Rz is 0x0.
impact indicator	unaffected
Chi Bit:	
Limitations:	not have
Exception:	not have

32-bit finger honorific title	
Operation:	Rz[31:16]=(Rx[31:16] != Ry[31:16]) ? 16'hFFFF:16'b0000 Rz[15:0]=(Rx[15:0] != Ry[15:0]) ? 16'hFFFF:16'b0000
Grammar:	pcmpne.16 rz, rx, ry
Description:	Determine whether the high and low half-words Rx and Ry are equal or not. If they are equal, the result of the corresponding half word of Rz is 0xFFFF, otherwise the result of the corresponding byte of Rz is 0x0.
impact indicator	unaffected
Chi Bit:	
Limitations:	not have
Exception:	not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 1 0 0 1 0 0 x 0 0	Rz		

Figure 15.47: PCMPNE.16

15.48 PCMPHS.(U/S)8 - 8-bit parallel (no/have) sign greater than or equal to compare instruction

coll ecti vel y kno wn as "ha rm oni zati on" or "ha rm oni zati on ind ex" (ma th.) honor ific title	
gram matica l	pcmphs.u8 rz, rx, ry pcmphs.s8 rz, rx, ry
manip ulate	Rz[31:24]=(Rx[31:24]>= Ry[31:24]) ? 8'hFF:8'h00 Rz[23:16]=(Rx[23:16]>= Ry[23:16]) ? 8'hFF:8'h00 Rz[15:8]=(Rx[15:8]>= Ry[15:8]) ? 8'hFF:8'h00 Rz[7:0]=(Rx[7:0]>= Ry[7:0]) ? 8'hFF:8'h00
Co mpi lati on resu lts	Only 32-bit instructions exist. pcmphs.u8 rz, rx, ry pcmphs.s8 rz, rx, ry

Description:	Determines whether 4 bytes of Rx are greater than or equal to the 4 bytes Ry in terms of bytes. If the bytes of Rx are greater than
--------------	---

User's Manual	If the byte equal to Ry, the result of the byte corresponding to Rz is 0xFF, otherwise the result of the byte corresponding to Rz is 0x0.
Impact	unaffected
Logo Bit:	
Limitations:	not have
Exception:	not have

32 Bit honorific title	
Operation:	Rz[31:24]=(Rx[31:24]>=Ry[31:24])?8'hFF:8'h00 Rz[23:16]=(Rx[23:16]>=Ry[23:16])?8'hFF:8'h00 Rz[15:8]=(Rx[15:8]>=Ry[15:8])?8'hFF:8'h00 Rz[7:0]=(Rx[7:0]>=Ry[7:0])?8'hFF:8'h00
Grammar:	pcmphs.u8 rz, rx, ry pcmphs.s8 rz, rx, ry
Description:	Determines whether 4 bytes of Rx are greater than or equal to the 4 bytes Ry in terms of bytes. If the bytes of Rx are greater than or equal to The result of the byte corresponding to Rz is 0xFF for the byte of Ry, otherwise the result of the byte corresponding to Rz is 0x0.
Impact Signs Bit:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 1 0 0 1 0 0 1 S 1 0	Rz		

Figure 15.48: PCMPHS.(US)8

15.49 PCMPHS.(U/S)16 - 16-bit parallel (no/have) sign greater than or equal to comparison directives

Unification directives	
grammatical	pcmphs.u16 rz, rx, ry pcmphs.s16 rz, rx, ry
manipulate	Rz[31:16]=(Rx[31:16]>= Ry[31:16]) ? 16'hFFFF:16'b0000 Rz[15:0]=(Rx[15:0]>= Ry[15:0]) ? 16'hFFFF:16'b0000
Translation results	Only 32-bit instructions exist. pcmphs.u16 rz, rx, ry pcmphs.s16 rz, rx, ry

Description:	Determine whether the halfword of Rx is greater than or equal to the halfword of Ry. If the half word of Rx is greater than or equal to the half word of Ry, then The result of Rz corresponding to a halfword is 0xFFFF, otherwise the result of Rz corresponding to a byte is 0x0.
affect Chi Bit:	unaffected
Limitations:	not have
Exception:	not have

32 bits directiv es	
Operatio n:	Rz[31:16]=(Rx[31:16]>= Ry[31:16]) ? 16'hFFFF:16'b0000 Rz[15:0]=(Rx[15:0]>= Ry[15:0]) ? 16'hFFFF:16'b0000
Gramma r:	pcmphs.u16 rz, rx, ry pcmphs.s16 rz, rx, ry
Descript ion:	Determine whether the halfword of Rx is greater than or equal to the halfword of Ry. If the half word of Rx is greater than or equal to the half word of Ry, then The result of Rz corresponding to a halfword is 0xFFFF, otherwise the result of Rz corresponding to a byte is 0x0.
affect Chi Bit:	unaffected
Limitati ons:	not have
Exceptio n:	not have

Command Format:

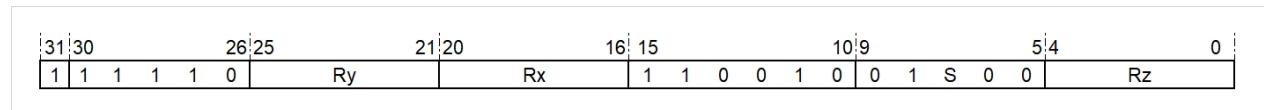


Figure 15.49: PCMPHS.(US)16

15.50 PCMPLT.(U/S)8 - 8-bit parallel (no/have) symbol less than compare select instruction

Har mon izati on mea ns honor ific title	
gram matica	pcmplt.u8 rz, rx, ry pcmplt.s8 rz, rx, ry

User's Manual	
manipulate	Rz[31:24]=(Rx[31:24]< Ry[31:24]) ? 8'hFF:8'h00 Rz[23:16]=(Rx[23:16]< Ry[23:16]) ? 8'hFF:8'h00 Rz[15:8]=(Rx[15:8]< Ry[15:8]) ? 8'hFF:8'h00 Rz[7:0]=(Rx[7:0]< Ry[7:0]) ? 8'hFF:8'h00
Translation results	Only 32-bit instructions exist. pcmplt.u8 rz, rx, ry pcmplt.s8 rz, rx, ry

Description:	Determines whether 4 bytes of Rx are less than 4 bytes of Ry in bytes. If the bytes of Rx are less than the bytes of Ry byte, the result of the corresponding byte of Rz is 0xFF, otherwise the result of the corresponding byte of Rz is 0x0.
affect Chi Bit:	unaffected
Limitations:	not have
Exception:	not have

32-bit directives	
Operation:	Rz[31:24]=(Rx[31:24]< Ry[31:24]) ? 8'hFF:8'h00 Rz[23:16]=(Rx[23:16]< Ry[23:16]) ? 8'hFF:8'h00 Rz[15:8]=(Rx[15:8]< Ry[15:8]) ? 8'hFF:8'h00 Rz[7:0]=(Rx[7:0]< Ry[7:0]) ? 8'hFF:8'h00
Grammar:	pcmplt.u8 rz, rx, ry pcmplt.s8 rz, rx, ry
Description:	Determines whether 4 bytes of Rx are less than 4 bytes of Ry in bytes. If the bytes of Rx are less than the bytes of Ry byte, the result of the corresponding byte of Rz is 0xFF, otherwise the result of the corresponding byte of Rz is 0x0.
Impact Signs Bit:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

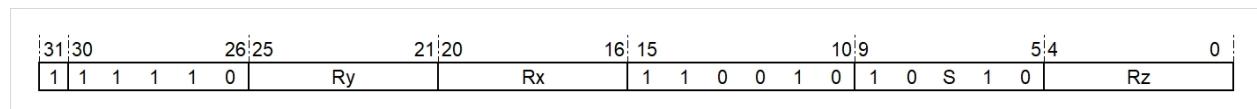


Figure 15.50: PCMPLT.(US)8

15.51 PCMPLT.(U/S)16 - 16-bit parallel (no/have) sign less than compare instruction

Unification directives	
grammatical	pcmplt.u16 rz, rx, Ry pcmplt.s16 rz, rx, ry
manipulate	Rz[31:16]=(Rx[31:16]< Ry[31:16]) ? 16'hFFFF:16'b0000 Rz[15:0]=(Rx[15:0]< Ry[15:0]) ? 16'hFFFF:16'b0000
Translation results	Only 32-bit instructions exist. pcmplt.u16 rz, rx, Ry pcmplt.s16 rz, rx, ry

Description:	Determine whether the halfword of Rx is smaller than the halfword of Ry. If the halfword of Rx is smaller than the halfword of Ry, Rz corresponds to The result of the halfword is 0xFFFF, otherwise the result of the corresponding byte of Rz is 0x0.
affect Chi Bit:	unaffected
Limitations:	not have
Exception:	not have

32-bit 32-bit (computing) directives	
Operation:	Rz[31:16]=(Rx[31:16]< Ry[31:16]) ? 16'hFFFF:16'b0000 Rz[15:0]=(Rx[15:0]< Ry[15:0]) ? 16'hFFFF:16'b0000
Grammar:	pcmplt.u16 rz, rx, Ry pcmplt.s16 rz, rx, ry
Description:	Determine whether the halfword of Rx is smaller than the halfword of Ry. If the halfword of Rx is smaller than the halfword of Ry, Rz corresponds to The result of the halfword is 0xFFFF, otherwise the result of the corresponding byte of Rz is 0x0.

Affect	affected
User's Manual Chi Bit:	
Limitations:	not have
Exception:	not have

Command Format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 1 1 0	Ry	Rx	1 1 0 0 1 0 1 0 S 0 0	Rz		

Figure 15.51: PCMPLT.(US)16

15.52 PMAX.(U/S)8--8-bit parallel (no/have) sign fetch large value instruction

standardization directives	
grammatical	pmax.u8 rz, rx, ry pmax.s8 rz, rx, ry
manipulate	Rz[31:24]= Max(Rx[31:24], Ry[31:24]) Rz[23:16]= Max(Rx[23:16], Ry[23:16]) Rz[15:8]= Max(Rx[15:8], Ry[15:8]) Rz[7:0]= Max(Rx[7:0], Ry[7:0])
Compilation results	Only 32-bit instructions exist. pmax.u8 rz, rx, ry pmax.s8 rz, rx, ry

Description:	The larger of the bytes of Rx and Ry is selected in bytes, and the result is stored in the corresponding byte of Rz.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	not have

32-bit finger honorific title	
Operation:	Rz[31:24]= Max(Rx[31:24], Ry[31:24]) Rz[23:16]= Max(Rx[23:16], Ry[23:16]) Rz[15:8]= Max(Rx[15:8], Ry[15:8]) Rz[7:0]= Max(Rx[7:0], Ry[7:0])
Grammar:	pmax.u8 rz, rx, ry pmax.s8 rz, rx, ry
Description:	The larger of the bytes of Rx and Ry is selected in bytes, and the result is stored in the corresponding byte of Rz.
affect Chi Bit:	unaffected
Limitati	not have

Users Manual
Exception: not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 1 0 0 1 1	0 0 S 1 0	Rz	

Figure 15.52: PMAX.(US)8

15.53 PMAX.(U/S)16 - 16-bit parallel (no/have) sign fetch large value instruction

Harmonization Directive	
grammatical	pmax.u16 rz, rx, ry pmax.s16 rz, rx, ry
manipulate	Rz[31:16]= Max(Rx[31:16], Ry[31:16]) Rz[15:0]= Max(Rx[15:0], Ry[15:0])
Compilation results	Only 32-bit instructions exist. pmax.u16 rz, rx, ry pmax.s16 rz, rx, ry

Description:	The larger of Rx and Ry selected in half-word units, and the result is stored in the corresponding half-word of Rz.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	not have

32-bit instruction	
Operation:	Rz[31:16]= Max(Rx[31:16], Ry[31:16]) Rz[15:0]= Max(Rx[15:0], Ry[15:0])
Grammar:	pmax.u16 rz, rx, ry pmax.s16 rz, rx, ry
Description:	The larger of Rx and Ry selected in half-word units, and the result is stored in the corresponding half-word of Rz.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

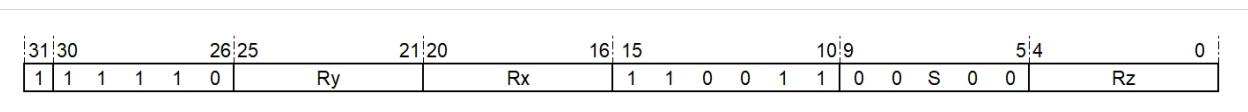


Figure 15.53: PMAX.(US)16

15.54 MAX.(U/S)32 - 32-bit (no/have) sign take large value instruction

Harmonization Directive	
grammatical	max.u32 rz, rx, ry max.s32 rz, rx, ry
manipulate	Rz[31:0]= Max(Rx[31:0], Ry[31:0])
Compilation results	Only 32-bit instructions exist. max.u32 rz, rx, ry max.s32 rz, rx, ry

Description:	Take the larger of Rx and Ry and the result is stored in Rz.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	not have

32-bit instruction	
Operation:	Rz[31:0]= Max(Rx[31:0], Ry[31:0])
Grammar:	max.u32 rz, rx, ry max.s32 rz, rx, ry
Description:	Take the larger of Rx and Ry and the result is stored in Rz.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

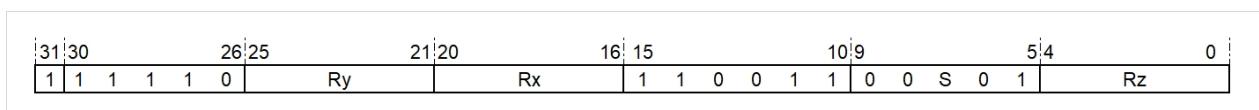


Figure 15.54: MAX.(US)32

15.55 PMIN.(U/S)8--8-bit parallel (no/have) sign fetch minor instruction

standardization directives	
grammatical	pmin.u8 rz, rx, ry pmin.s8 rz, rx, ry
manipulate	Rz[31:24]= Min(Rx[31:24], Ry[31:24]) Rz[23:16]= Min(Rx[23:16], Ry[23:16]) Rz[15:8]= Min(Rx[15:8], Ry[15:8]) Rz[7:0]= Min(Rx[7:0], Ry[7:0])
Compilation results	Only 32-bit instructions exist. pmin.u8 rz, rx, ry pmin.s8 rz, rx, ry

Description:	The smaller of the bytes of Rx and Ry selected in bytes, and the result is stored in the corresponding byte of Rz.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	not have

32-bit finger honorific title	
Operation:	Rz[31:24]= Min(Rx[31:24], Ry[31:24]) Rz[23:16]= Min(Rx[23:16], Ry[23:16]) Rz[15:8]= Min(Rx[15:8], Ry[15:8]) Rz[7:0]= Min(Rx[7:0], Ry[7:0])
Grammar:	pmin.u8 rz, rx, ry pmin.s8 rz, rx, ry
Description:	The smaller of the bytes of Rx and Ry selected in bytes, and the result is stored in the corresponding byte of Rz.
affect Chi Bit:	unaffected
Limitati	not have

Users Manual
Exception:

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 1 0 0 1 1 0 1 S 1 0	Rz		

Figure 15.55: PMIN.(US)8

15.56 PMIN.(U/S)16 - 16-bit Parallel (no/have) Sign Fetch Minor Instruction

Harmonization Directive	
grammatical	pmin.u16 rz, rx, ry pmin.s16 rz, rx, ry
manipulate	Rz[31:16]= Min(Rx[31:16], Ry[31:16]) Rz[15:0]= Min(Rx[15:0], Ry[15:0])
Compilation results	Only 32-bit instructions exist. pmin.u16 rz, rx, ry pmin.s16 rz, rx, ry

Description:	The smaller of Rx and Ry selected in half-word units, and the result is stored in the corresponding half-word of Rz.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	not have

32-bit instruction	
Operation:	Rz[31:16]= Min(Rx[31:16], Ry[31:16]) Rz[15:0]= Min(Rx[15:0], Ry[15:0])
Grammar:	pmin.u16 rz, rx, ry pmin.s16 rz, rx, ry
Description:	The smaller of Rx and Ry selected in half-word units, and the result is stored in the corresponding half-word of Rz.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

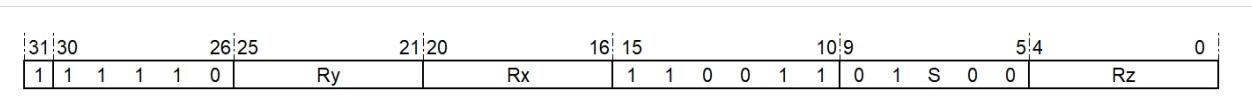


Figure 15.56: PMIN.(US)16

15.57 MIN.(U/S)32 - 32-bit (no/have) sign minima instruction

Harmonization Directive	
grammatical	min.u32 rz, rx, ry min.s32 rz, rx, ry
manipulate	Rz[31:0]= Min(Rx[31:0], Ry[31:0])
Compilation results	Only 32-bit instructions exist. min.u32 rz, rx, ry min.s32 rz, rx, ry

Description:	Take the smaller of Rx and Ry and the result is deposited in Rz.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	not have

32-bit instruction	
Operation:	Rz[31:0]= Min(Rx[31:0], Ry[31:0])
Grammar:	min.u32 Rz, Rx, Ry min.s32 Rz, Rx, Ry
Description:	Take the smaller of Rx and Ry and the result is deposited in Rz.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

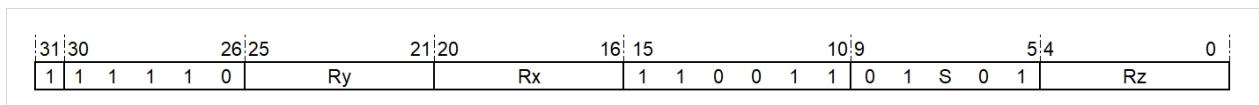


Figure 15.57: MIN.(US)32

15.58 DEXTI - Immediate Digital Intercept Instruction

Harmonization Directive	
grammatical	dexti rz, rx, ry, imm5
manipulate	Rz[31:0]= {Ry[31:0], Rx[31:0}][imm[4:0]+31:imm[4:0]]
Compilation results	Only 32-bit instructions exist. dexti rz, rx, ry, imm5

Description:	Ry and Rx and form a 64-bit number {Ry[31:0], Rx[31:0]}, which is logically shifted to the right, with the low word of the result deposited into the The number of right shifts is determined by the value of the 5-bit immediate number (IMM5).
affect Chi Bit:	unaffected
Limitations:	Immediate numbers range from 0-31
Exception:	not have

32-bit finger honorific title	
Operation:	Rz[31:0]= {Ry[31:0], Rx[31:0}][imm[4:0]+31:imm[4:0]]
Grammar:	dexti rz, rx, ry, imm5
Description:	Ry and Rx and form a 64-bit number {Ry[31:0], Rx[31:0]}, which is logically shifted to the right, with the low word of the result deposited into the The number of right shifts is determined by the value of the 5-bit immediate number (IMM5).
affect Chi Bit:	unaffected
Limitations:	Immediate numbers range from 0-31
Exception:	not have

Command Format:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1	1	1	1	0	Ry	Rx	1	0	0	1	1

Figure 15.58: DEXTI

15.59 EXT - Word Intercept Command

Harmonization Directive	
grammatical	dext rz, rx, ry, rs
manipulate	Rz[31:0]= {Ry[31:0], Rx[31:0}][Rs[5:0]+31:Rs[5:0]]
Compilation results	Only 32-bit instructions exist. dext rz, rx, ry, rs

Description:	Ry and Rx and form a 64-bit number {Ry[31:0], Rx[31:0]}, which is logically shifted to the right, with the low word of the result deposited into the RZ. the number of right shifts is determined by the value of Rs [5:0] Rs [5:0] is greater than 32, the number of right shifts is 32.
Impact Logo Bit:	unaffected
Limitations:	Ry[5:0] is in the range of 0-32.
Exception:	not have

32 bits directives	
Operation:	Rz[31:0]= {Ry[31:0], Rx[31:0}][Rs[5:0]+31:Rs[5:0]]
Grammatical:	dext rz, rx, ry, rs
Description:	Ry and Rx and form a 64-bit number {Ry[31:0], Rx[31:0]}, which is logically shifted to the right, with the low word of the result deposited into the RZ. the number of right shifts is determined by the value of Rs [5:0] Rs [5:0] is greater than 32, the number of right shifts is 32.
Impact Logo Bit:	unaffected
Limitations:	Ry[5:0] is in the range of 0-32.

Exception not have
User's Manual
n:

Command Format:

:31 30	26 25	21 20	16 15	10 9	5 4	0
[1 1 1 1 1 0]	Ry	Rx	[1 0 0 1 1]	Rs	Rz	

Figure 15.59: EXT

15.60 PKG - Immediate Number Shift Packing Instruction

Harmonization Directive	
grammatical	pkg rz, rx, imm4a, ry, oimm4b
manipulate	Rz[31:0]= {(Ry[31:0]>> oimm4b[3:0])[15:0], (Rx[31:0]>> imm4a[3:0])[15:0]}
Compilation results	Only 32-bit instructions exist. pkg rz, rx, imm4a, ry, oimm4b

Explanation:	For Rx logical right shift, the lower half word of the intercepted shift result is deposited into the lower half word of RZ, and the number of right shifts is determined by the value of the immediate number (imm4a[3:0]); for RY logical right shift, the lower half word of the intercepted shift result is deposited into the upper half word of RZ, and the number of right shifts is determined by the number of bits in a cubic meter with a bias of one. That is, the value of the number (oimm4b[3:0]) is determined.
Impact Signs Bit:	unaffected
set a limit (on) System:	imm4a ranges from 0 to 15 The range of oimm4b is 1 to 16.
discretionary Regular:	not have

32 Bit honorific title	
drill (practice)	Rz[31:0]= {(Ry[31:0]>> oimm4b[3:0])[15:0], (Rx[31:0]>> imm4a[3:0])[15:0]}

Make User's Manual	
tell to Law:	pkg rz, rx, imm4a, ry, oimm4b
Exp lana tion : Bit:	For Rx logical right shift, the lower half word of the intercepted shift result is deposited into the lower half word of RZ, and the number of right shifts is determined by the value of the immediate number (imm4a[3:0]); for RY logical right shift, the lower half word of the intercepted shift result is deposited into the upper half word of RZ, and the number of right shifts is determined by the number of bits in a cubic meter with a bias of one. That is, the value of the number (oimm4b[3:0]) is determined.
Imp act Sign s Bit:	unaffected
set a limit (on) Syste m:	imm4a ranges from 0 to 15 The range of oimm4b is 1 to 16.
discri minat e Regul ar:	not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 1	oimm4b	oimm4a	Rz

Figure 15.60: PKG

15.61 PKGLL - low half-word packing instruction

Harmonization Directive	
grammatical	pkgll rz, rx, ry
manipulate	Rz[31:0]= {Ry[15:0], Rx[15:0]}
Compilation results	Only 32-bit instructions exist. pkgll rz, rx, ry

Description:	The low halfword of Ry is stored in the high halfword of Rz, and the low halfword of Rx is stored in the low halfword of Rz.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	not have

32-bit instruction	
Operation:	Rz[31:0]= {Ry[15:0], Rx[15:0]}
Grammar:	pkgll rz, rx, ry
Description:	The low halfword of Ry is stored in the high halfword of Rz, and the low halfword of Rx is stored in the low halfword of Rz.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 1 0 1 1 0 0 0 x 1 0		Rz	

Figure 15.61: PKGLL

15.62 PKGHH - high half-word packing instruction

Harmonization Directive	
grammatical	pkghh rz, rx, ry
manipulate	Rz[31:0]= {Ry[31:16], Rx[31:16]}
Compilation results	Only 32-bit instructions exist. pkghh rz, rx, ry

Description:	The high halfword of Ry is stored in the high halfword of Rz, and the high halfword of Rx is stored in the low halfword of Rz.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	not have

32-bit instruction	
Operation:	Rz[31:0]= {Ry[31:16], Rx[31:16]}
Grammar:	pkghh rz, rx, ry
Description:	The high halfword of Ry is stored in the high halfword of Rz, and the high halfword of Rx is stored in the low halfword of Rz.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

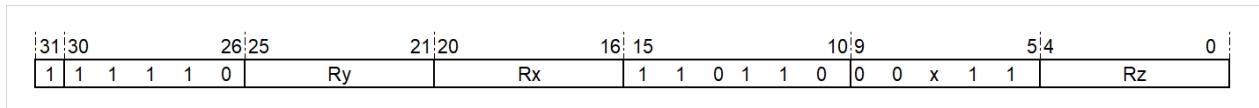


Figure 15.62: PKGHH

15.63 PEXT.U8.E - 8 bit parallel unsigned extended instructions

standardization directives	
grammatical	pext.u8.e rz, rx
manipulate	$Rz+1[31:16] = \text{zero_extend}(Rx[31:24])$ $Rz+1[15:0] = \text{zero_extend}(Rx[23:16])$ $Rz[31:16] = \text{zero_extend}(Rx[15:8])$ $Rz[15:0] = \text{zero_extend}(Rx[7:0])$
compilation result resolute	Only 32-bit instructions exist. pext.u8.e rz, rx

Exp lanation :	Define Rx[31:24], Rx[23:16], Rx[15:8], Rx[7:0] as byte 3, byte 2, byte 1, and byte 0. Expand 3, byte 2, byte 1, and byte 0 unsigned, and then store the result in the unsigned half-word format. High halfword of Rz+1, low halfword of Rz+1, high halfword of Rz, low halfword Rz.
Mark er s of in fl u e n ce Bit:	unaffected
set a limit (on) Syste m:	not have
discriminate	not have

Begin	Manual
User	lar:

32 in d e x (m at h.) hono rific title	
Ope rate : tell to Law:	Rz+1[31:16] = zero_extend(Rx[31:24]) Rz+1[15:0]= zero_extend(Rx[23:16]) Rz[31:16] = zero_extend(Rx[15:8]) Rz[15:0]= zero_extend(Rx[7:0]) pext.u8.e rz, rx
Exp lana tion : M ar k er s of in fl u e n ce Bit:	Define Rx[31:24], Rx[23:16], Rx[15:8], Rx[7:0] as byte 3, byte 2, byte 1, and byte 0. Expand 3, byte 2, byte 1, and byte 0 unsigned, and then store the result in the unsigned half-word format. High halfword of Rz+1, low halfword of Rz+1, high halfword of Rz, low halfword Rz. unaffected
set a limit (on) Syste m:	not have
discrimina te Regula lar:	not have

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	5'b0	Rx	1 1 0 1 1 0	0 1 S 0 0	Rz	

Figure 15.63: PEXT.

15.64 PEXT.S8.E - 8 bit parallel signed extended instructions

standardization directives	
grammatical	pext.s8.e rz, rx
manipulate	Rz+1[31:16] = sign_extend(Rx[31:24]) Rz+1[15:0]= sign_extend(Rx[23:16]) Rz[31:16] = sign_extend(Rx[15:8]) Rz[15:0]= sign_extend(Rx[7:0])
compilation result resolute	Only 32-bit instructions exist. pext.s8.e rz, rx

Expansion: : Bit:	Define Rx[31:24], Rx[23:16], Rx[15:8], Rx[7:0] as byte 3, byte 2, byte 1, and byte 0. Signed expansion is performed on byte 3, byte 2, byte 1, and byte 0 in order, and the expanded data format is a signed half-word, and the result is deposited into High halfword of Rz+1, low halfword of Rz+1, high halfword of Rz, low halfword Rz.
Marker: Bit:	unaffected
set a limit (on) System:	not have
discriminate Regu	not have

32 in d e x (m at h.) hono rific title	
Operate rate : tell to Law: Explanation : Mark er s of in fl u e n ce Bit:	Rz+1[31:16] = sign_extend(Rx[31:24]) Rz+1[15:0]= sign_extend(Rx[23:16]) Rz[31:16] = sign_extend(Rx[15:8]) Rz[15:0]= sign_extend(Rx[7:0]) pext.s8.e rz, rx Define Rx[31:24], Rx[23:16], Rx[15:8], Rx[7:0] as byte 3, byte 2, byte 1, and byte 0. Signed expansion is performed on byte 3, byte 2, byte 1, and byte 0 in order, and the expanded data format is a signed half-word, and the result is deposited into High halfword of Rz+1, low halfword of Rz+1, high halfword of Rz, low halfword Rz. unaffected
set a limit (on) System:	not have
discriminate Regular:	not have

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	5'b0	Rx	1 1 0 1 1 0	0 1 S 0 0	Rz	

Figure 15.64: PEXT.

15.65 PEXTX.U8.E - 8 bit parallel unsigned cross-extended instructions

standardization directives	
grammatical	pextx.u8.e rz, rx
manipulate	$Rz+1[31:16] = \text{zero_extend}(Rx[31:24])$ $Rz+1[15:0] = \text{zero_extend}(Rx[15:8])$ $Rz[31:16] = \text{zero_extend}(Rx[23:16])$ $Rz[15:0] = \text{zero_extend}(Rx[7:0])$
compilation result resolute	Only 32-bit instructions exist. pextx.u8.e rz, rx

Explanation:	Define Rx[31:24], Rx[23:16], Rx[15:8], Rx[7:0] as byte 3, byte 2, byte 1, and byte 0. Expand 3, byte 1, byte 2, and byte 0 unsigned, and then store the result in the unsigned half-word format. High halfword of Rz+1, low halfword of Rz+1, high halfword of Rz, low halfword Rz.
Marker	unaffected
set a limit (on) System:	not have
discriminate	not have

Begin	Manual
User	lar:

32 in d e x (m at h.) hono rific title	
Ope rate : :	Rz+1[31:16] = zero_extend(Rx[31:24]) Rz+1[15:0] = zero_extend(Rx[15:8]) Rz[31:16] = zero_extend(Rx[23:16]) Rz[15:0]= zero_extend(Rx[7:0])
tell to Law:	pextx.u8.e rz, rx
Exp lana tion : :	Define Rx[31:24], Rx[23:16], Rx[15:8], Rx[7:0] as byte 3, byte 2, byte 1, and byte 0. Expand 3, byte 1, byte 2, and byte 0 unsigned, and then store the result in the unsigned half-word format. High halfword of Rz+1, low halfword of Rz+1, high halfword of Rz, low halfword Rz.
M ar k er s of in fl u e n ce Bit:	unaffected
set a limit (on) Syste m:	not have
discri mina te Regu lar:	not have

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	5'b0	Rx	1 1 0 1 1 0	0 1 S 0 1	Rz	

Figure 15.65: PEXTX.

15.66 PEXTX.S8.E - 8 bit parallel signed cross-extension instructions

standardization directives	
grammatical	pextx.s8.e rz, rx
manipulate	$Rz+1[31:16] = \text{sign_extend}(Rx[31:24])$ $Rz+1[15:0] = \text{sign_extend}(Rx[15:8])$ $Rz[31:16] = \text{sign_extend}(Rx[23:16])$ $Rz[15:0] = \text{sign_extend}(Rx[7:0])$
compilation result resolute	Only 32-bit instructions exist. pextx.s8.e rz, rx

Explanation :	Define Rx[31:24], Rx[23:16], Rx[15:8], Rx[7:0] as byte 3, byte 2, byte 1, and byte 0. Signed expansion is performed on byte 3, byte 1, byte 2, and byte 0 in order, and the expanded data format is a signed half-word, and the result is deposited into High halfword of Rz+1, low halfword of Rz+1, high halfword of Rz, low halfword Rz.
Markers of influence Bit:	unaffected
set a limit (on) System:	not have
discriminate	not have

User Manual
lar:

32 in d e x (m at h.) hono rific title	
Ope rate : tell to Law:	Rz+1[31:16] = sign_extend(Rx[31:24]) Rz+1[15:0] = sign_extend(Rx[15:8]) Rz[31:16] = sign_extend(Rx[23:16]) Rz[15:0]= sign_extend(Rx[7:0]) pextx.s8.e rz, rx
Exp lana tion : M ar k er s of in fl u e n ce Bit:	Define Rx[31:24], Rx[23:16], Rx[15:8], Rx[7:0] as byte 3, byte 2, byte 1, and byte 0. Signed expansion is performed on byte 3, byte 1, byte 2, and byte 0 in order, and the expanded data format is a signed half-word, and the result is deposited into High halfword of Rz+1, low halfword of Rz+1, high halfword of Rz, low halfword Rz. unaffected
set a limit (on) Syste m:	not have
discri mina te Regu lar:	not have

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	5'b0	Rx	1 1 0 1 1 0	0 1 S 0 1	Rz	

Figure 15.66: PEXTX.

15.67 NARL——低位截取拼装指令

Harmonization Directive	
grammatical	narl rz, rx, ry
manipulate	Rz[31:0]= {Ry[23:16], Ry[7:0], Rx[23:16], Rx[7:0]}
Compilation results	Only 32-bit instructions exist. narl rz, rx, ry

Description:	Define the high and low halves of Ry, and the high and low halves of Rx as halves 3, 2, 1, and 0. Intercept halves 3, 2, and 1, The lower 8 bits of 0 are stored in Rz[31:24], Rz[23:16], Rz[15:8], and Rz[7:0] respectively.
affect Chi Bit:	unaffected
Limitations:	not have
Exception:	not have

32 bits directives	
Operation:	Rz[31:0]= { Ry[23:16], Ry[7:0], Rx[23:16], Rx[7:0] }
Grammar:	narl rz, rx, ry
Description:	Define the high and low halves of Ry, and the high and low halves of Rx as halves 3, 2, 1, and 0. Intercept halves 3, 2, and 1, The lower 8 bits of 0 are stored in Rz[31:24], Rz[23:16], Rz[15:8], and Rz[7:0] respectively.
affect Chi Bit:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

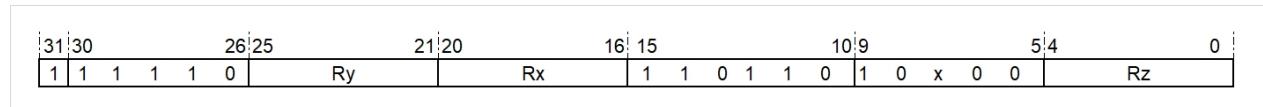


Figure 15.67: NARL

15.68 NARH - High Intercept Splicing Directive

Harmonization Directive	
grammatical	narh rz, rx, ry

1	
manipulate	Rz[31:0]= {Ry[31:24], Ry[15:8], Rx[31:24], Rx[15:8]}
Compilation results	Only 32-bit instructions exist. narh rz, rx, ry

Description:	Define the high and low halves of Ry, and the high and low halves of Rx as halves 3, 2, 1, and 0. Intercept halves 3, 2, and 1, The upper 8 bits of 0 are stored in Rz[31:24], Rz[23:16], Rz[15:8], and Rz[7:0] respectively.
affect Chi Bit:	unaffected
Limitations:	not have
Exception:	not have

32 bits directives	
Operation:	Rz[31:0]= {Ry[31:24], Ry[15:8], Rx[31:24], Rx[15:8]}
Grammar:	narh rz, rx, ry
Description:	Define the high and low halves of Ry, and the high and low halves of Rx as halves 3, 2, 1, and 0. Intercept halves 3, 2, and 1, The upper 8 bits of 0 are stored in Rz[31:24], Rz[23:16], Rz[15:8], and Rz[7:0] respectively.
affect Chi Bit:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

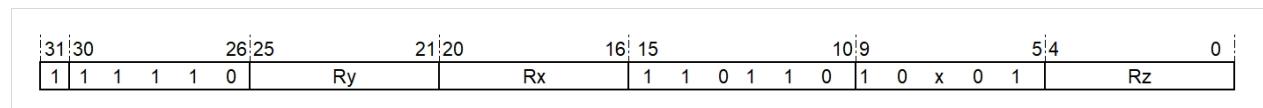


Figure 15.68: NARH

15.69 NARLX - low cross intercept splice command

Harmonization Directive	
grammatical	narlx rz, rx, ry

l	
manipulate	Rz[31:0]= {Ry[23:16], Rx[23:16], Ry[7:0], Rx[7:0]}
Compilation results	Only 32-bit instructions exist. narlx rz, rx, ry

Description:	Define the high and low halves of Ry, and the high and low halves of Rx as halves 3, 2, 1, and 0. Intercept halves 3, 1, and 2, The lower 8 bits of 0 are stored in Rz[31:24], Rz[23:16], Rz[15:8], Rz[7:0], respectively.
affect Chi Bit:	unaffected
Limitations:	not have
Exception:	not have

32 bits directives	
Operation:	Rz[31:0]= {Ry[23:16], Rx[23:16], Ry[7:0], Rx[7:0]}
Grammar:	narlx rz, rx, ry
Description:	Define the high and low halves of Ry, and the high and low halves of Rx as halves 3, 2, 1, and 0. Intercept halves 3, 1, and 2, The lower 8 bits of 0 are stored in Rz[31:24], Rz[23:16], Rz[15:8], Rz[7:0], respectively.
affect Chi Bit:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

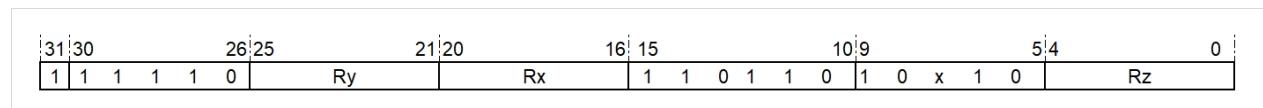


Figure 15.69: NARLX

15.70 NARHX - High level cross intercept splice command

Harmonization Directive	
grammatical	narhx rz, rx, ry

1	
manipulate	Rz[31:0]= {Ry[31:24], Rx[31:24], Ry[15:8], Rx[15:8]}
Compilation results	Only 32-bit instructions exist. narhx rz, rx, ry

Description:	Define the high and low halves of Ry, and the high and low halves of Rx as halves 3, 2, 1, and 0. Intercept halves 3, 1, and 2, The upper 8 bits of 0 are stored in Rz[31:24], Rz[23:16], Rz[15:8], Rz[7:0], respectively.
affect Chi Bit:	unaffected
Limitations:	not have
Exception:	not have

32 bits directives	
Operation:	Rz[31:0]= {Ry[31:24], Rx[31:24], Ry[15:8], Rx[15:8]}
Grammar:	narhx rz, rx, ry
Description:	Define the high and low halves of Ry, and the high and low halves of Rx as halves 3, 2, 1, and 0. Intercept halves 3, 1, and 2, The upper 8 bits of 0 are stored in Rz[31:24], Rz[23:16], Rz[15:8], Rz[7:0], respectively.
affect Chi Bit:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

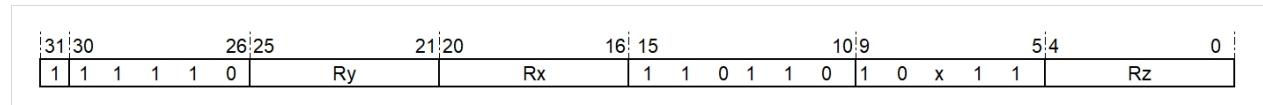


Figure 15.70: NARHX

15.71 CLIP.(U/S)32 - Immediate number (no/have) symbol trimming saturation instruction

unify chemicalization honori fic title	
gram matica 1	clipi.u32 rz, rx, imm5 clipi.s32 rz, rx, oimm5
manip ulate	Max= 2(imm5)-1, Min= 0 //unsigned Max= 2(oimm5-1)-1, Min= -2(oimm5-1) //signed if(Rx[31:0] > Max) Rz[31:0]= Max else if(Rx[31:0] <Min) Rz[31:0]= Min else Rz[31:0]= Rx[31:0]
Translati on resul ts	Only 32-bit instructions exist. clipi.u32 rz, rx, imm5 clipi.s32 rz, rx, oimm5

Explana tion :	Saves the Rx scaling bit to the target data specified by IMM5/OIMM5, where IMM5/OIMM5 characterizes the bit width of the target data. If the value of Rx exceeds the range of values that can be represented by the target data, saturation is performed. For the unsigned CLIP instruction, the upper saturation value is $2^{IMM5}-1$ and the lower saturation value is 0x0. If Rx is greater than the upper saturation value, the result is $2^{IMM5}-1$, the other cases result in Rx. For the signed CLIP instruction, OIMM5 is immediate number with a bias of 1. The upper saturation value is $2^{(OIMM5-1)}-1$ and the lower saturation value is $-2^{(OIMM5-1)}$. If Rx is greater than the upper saturation value, the result is $2^{(OIMM5-1)}-1$, and if Rx is less than the lower saturation value, the result is $-2^{(OIMM5-1)}$, and the other cases result in Rx.
Mark er s of	unaffected

in User's fl u e n ce Bit:	Manual
set a limi t (on) Syst em:	The range of IMM5 is 0~31 The range of OIMM5 is 1 to 32.
disc rimi nate Reg ular :	not have

32 in d e x (m at h.) hon orifi c title	
Ope rate : tell to Law : Exp lana tion : M ar k er s of in fl	<p>If(unsigned)Max = 2(oimm5)-1, Min = 0 // If(signed)Max= 2(oimm5-1)-1, Min= -2(oimm5-1) // if(Rx[31:0] > Max) Rz[31:0]= Max else if(Rx[31:0] <Min) Rz[31:0]= Min else Rz[31:0]= Rx[31:0]</p> <p>clipi.u32 rz, rx, imm5 clipi.s32 rz, rx, oimm5</p> <p>Saves the Rx scaling bit to the target data specified by IMM5/OIMM5, where IMM5/OIMM5 characterizes the bit width of the target data. If the value of Rx exceeds the range of values that can be represented by the target data, saturation is performed. For the unsigned CLIP instruction, the upper saturation value is $2^{IMM5}-1$ and the lower saturation value is 0x0. If Rx is greater than the upper saturation value, the result is $2^{IMM5}-1$, the other cases result in Rx. For the signed CLIP instruction, OIMM5 is immediate number with a bias of 1. The upper saturation value is $2^{(OIMM5-1)}-1$ and the lower saturation value is $-2^{(OIMM5-1)}$. If Rx is greater than the upper saturation value, the result is $2^{(OIMM5-1)}-1$, and if Rx is less than the lower saturation value, the result is $-2^{(OIMM5-1)}$, and the other cases result in Rx.</p> <p>unaffected</p>

User's Manual	
Bit: set a limi t (on) Syst em:	The range of IMM5 is 0~31 The range of OIMM5 is 1 to 32.
disc rimi nate Reg ular :	not have

Command Format:

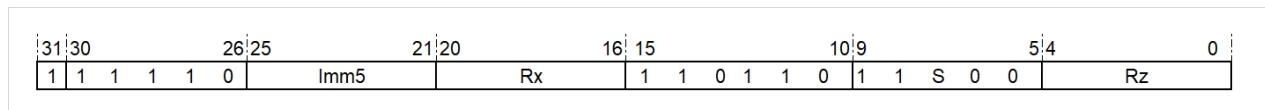


Figure 15.71: CLIP.(US)32

15.72 CLIP.(U/S)32 - (No/Have) Symbol Trimming Saturation Instruction

unify chemicalization honori fic title	
gram matica 1	clip.u32 rz, rx, ry clip.s32 rz, rx, ry
manip ulate	Max= 2(Ry[4:0])-1 Min= 0 //unsigned Max= 2(Ry[4:0]-1)-1 Min= -2(Ry[4:0]-1) //signed if(Rx[31:0] > Max) Rz[31:0]= Max else if(Rx[31:0] <Min) Rz[31:0]= Min else Rz[31:0]= Rx[31:0]
Translati on resul ts	Only 32-bit instructions exist. clip.u32 rz, rx, ry clip.s32 rz, rx, ry

Explana tion :	Saves the Rx scaling bit to the target data specified by Ry[4:0], where Ry[4:0] characterizes the bit width of the target data. If the value of Rx exceeds the range of values that can be represented by the target data, saturation is performed. For the unsigned CLIP instruction, the upper saturation value is $2^{Ry[4:0]}-1$ and the lower saturation value is 0x0. If Rx is greater than the upper saturation value, the result is $2^{Ry[4:0]}-1$, otherwise the result is Rx. For signed CLIP instructions, the upper saturation value is $2^{(Ry[4:0]-1)}-1$ and the lower saturation value is $-2^{(Ry[4:0]-1)}$. If Rx is greater than the upper saturation sum, the result is $2^{(Ry[4:0]-1)}-1$, if Rx is less than the lower saturation value, the result is $-2^{(Ry[4:0]-1)}$, and the result is Rx in all other cases.
Mark er s of in	unaffected

User's Manual	
Bit: set a limi t (on) Syst em:	not have
disc rimi nate Reg ular :	not have

32 in d e x (m at h.) hon orifi c title	
Ope rate : tell to Law : Exp lana tion : M ar k er s of in fl u	<pre> Max= 2(Ry[4:0])-1 Min= 0 //unsigned Max= 2(Ry[4:0]-1)-1 Min= -2(Ry[4:0]-1) //signed if(Rx[31:0] > Max) Rz[31:0]= Max else if(Rx[31:0] < Min) Rz[31:0]= Min else Rz[31:0]= Rx[31:0] clip.u32 rz, rx, ry clip.s32 rz, rx, ry Saves the Rx scaling bit to the target data specified by Ry[4:0], where Ry[4:0] characterizes the bit width of the target data. If the value of Rx exceeds the range of values that can be represented by the target data, saturation is performed. For the unsigned CLIP instruction, the upper saturation value is $2^{Ry[4:0]}-1$ and the lower saturation value is 0x0. If Rx is greater than the upper saturation value, the result is $2^{Ry[4:0]}-1$, otherwise the result is Rx. For signed CLIP instructions, the upper saturation value is $2^{(Ry[4:0]-1)}-1$ and the lower saturation value is $-2^{(Ry[4:0]-1)}$. If Rx is greater than the upper saturation sum, the result is $2^{(Ry[4:0]-1)}-1$, if Rx is less than the lower saturation value, the result is $-2^{(Ry[4:0]-1)}$, and the result is Rx in all other cases. unaffected </pre>

User's n ce Bit:	Manual
set a limi t (on) Syst em:	not have
disc rimi nate Reg ular :	not have

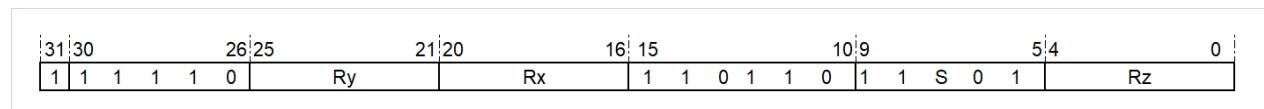
Command Format:

Figure 15.72: CLIP.(US)32

15.73 PCLIP1.(U/S)16 - 16-bit Parallel Immediate Number (without/with) Symbol Trimming Saturation directives

c	
ol	
le	
ct	
iv	
el	
y	
k	
n	
o	
w	
n	
as	
"h	
ar	
m	
o	
ni	
za	
ti	
o	
n"	
or	
"h	
ar	
m	
o	
ni	
za	
ti	
o	
n	
in	
d	
e	
x"	
(
m	
at	
h.	
)	
hono	

title	Users Manual
tell to Budd hist teaching	pclip.u16 rz, rx, imm4 pclip.s16 rz, rx, oimm4
m a ni pi ul at e	Max= 2(imm4)-1, Min= 0 //unsigned Max = 2(oimm4-1)-1, Min = -2(oimm4-1) //signed if(Rx[31:16] Rx[15:0] > Max) Rz[31:16] Rz[15:0]= Max else if(Rx[31:16] Rx[15:0] <Min) Rz[31:16] Rz[15:0]= Min else Rz[31:16] Rz[15:0]= Rx[31:16] Rx[15:0]
c o m pi la ti o n re s ul t resol ute	Only 32-bit instructions exist. pclip.u16 rx, rx, imm4 pclip.s16 rx, rx, oimm4

Explana tion: :	The high and low half words of Rx are used as the raw data, and are saved separately scaled to the target data specified by IMM4/OIMM4, where IMM4/OIMM4 characterizes the bit width of the target data. If the value of the raw data exceeds the range of values that can be represented by the target data, saturation is performed. For the unsigned CLIP instruction, the upper saturation value is $2^{\text{IMM4}-1}$ and the lower saturation value is 0x0. If the original data is greater than the upper saturation value, the result is $2^{\text{IMM4}-1}$, otherwise the result is the original data itself. For the signed CLIP instruction, OIM M4 is immediate number with a bias of 1. The upper saturation value is $2^{(\text{OIMM4}-1)-1}$ and the lower saturation value is $-2^{(\text{OIMM4}-1)}$. If the original is greater than the upper saturation value, the result is $2^{(\text{OIMM4}-1)-1}$, if the original is less than the lower saturation value, the result is $-2^{(\text{OIMM4}-1)}$. The result is $-2^{(\text{OIMM4}-1)}$, in other cases the result is the original data itself.
Mark er s of in fl u e n ce Bit:	unaffected
set a limi t (on) Syst em:	Imm4 ranges from 0 to 15 The range of Oimm4 is 1 to 16.
disc rimi nate Reg ular :	not have

32 in d e x (m at h.) hon orifi c title	
Ope rate : tell to Law :	<pre>Max= 2(imm4)-1, Min= 0 //unsigned Max = 2(oimm4-1)-1, Min = -2(oimm4-1) //signed if(Rx[31:16] Rx[15:0] > Max) Rz[31:16] Rz[15:0]= Max else if(Rx[31:16] Rx[15:0] <Min) Rz[31:16] Rz[15:0]= Min else Rz[31:16] Rz[15:0]= Rx[31:16] Rx[15:0]</pre> <p>pclip.u16 rx, rx, imm4 pclip.s16 rx, rx, oimm4</p>
Exp lana tion : M ar k er s of in fl	<p>The high and low half words of Rx are used as the raw data, and are saved separately scaled to the target data specified by IMM4/OIMM4, where IMM4/OIMM4 characterizes the bit width of the target data. If the value of the raw data exceeds the range of values that can be represented by the target data, saturation is performed.</p> <p>For the unsigned CLIP instruction, the upper saturation value is $2^{\text{IMM4}-1}$ and the lower saturation value is 0x0. If the original data is greater than the upper saturation value, the result is $2^{\text{IMM4}-1}$, otherwise the result is the original data itself.</p> <p>For the signed CLIP instruction, OIMM4 is immediate number with a bias of 1. The upper saturation value is $2^{(\text{OIMM4}-1)}-1$ and the lower saturation value is $-2^{(\text{OIMM4}-1)}$. If the original is greater than the upper saturation value, the result is $2^{(\text{OIMM4}-1)}-1$, if the original is less than the lower saturation value, the result is $-2^{(\text{OIMM4}-1)}$.</p> <p>The result is $-2^{(\text{OIMM4}-1)}$, in other cases the result is the original data itself.</p> <p>unaffected</p>

User's Manual	
Bit: set a limit (on) System:	Imm5 ranges from 0 to 15 Oimm5 ranges from 1 to 16.
discriminate Regular: :	not have

Command Format:

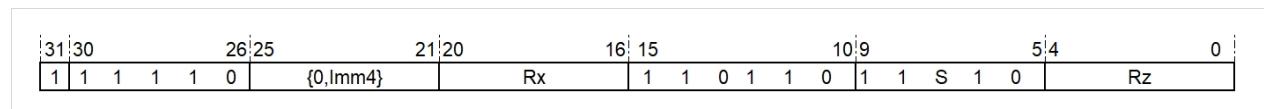


Figure 15.73: PCLIPI.(US)16

15.74 PCLIP.(U/S)16 - 16-bit Parallel (Without/With) Symbol Trimming Saturation Instruction

c ol le ct iv el y k n o w n as "h ar m o ni za ti o n" or "h ar m o ni za ti o n in d e x" (m at h.) hono rific title	
---	--

tel to User Manual Budd hist teach ing	pclip.u16 rz, rx, ry pclip.s16 rz, rx, ry
m a ni p ul at e	Max= 2(Ry[3:0])-1, Min= 0 //unsigned Max = 2(Ry[3:0]-1)-1, Min = -2(Ry[3:0]-1) //signed if(Rx[31:16] Rx[15:0] > Max) Rz[31:16] Rz[15:0]= Max else if(Rx[31:16] Rx[15:0] <Min) Rz[31:16] Rz[15:0]= Min else Rz[31:16] Rz[15:0]= Rx[31:16] Rx[15:0]
c o m pi la ti o n re s ul t resol ute	Only 32-bit instructions exist. pclip.u16 rx, ry pclip.s16 rx, , ry

Exp lana tion :	The high and low half-words of Rx are used as raw data, and are saved bit-scaled into the target data specified by Ry[3:0] respectively, where Ry[3 :0]4 characterizes the bit width of the target data. If the value of the original data exceeds the range of values that can be represented by the target data, saturation is performed. For the unsigned CLIP instruction, the upper saturation value is $2^{\text{Ry}[3:0]}-1$, and the lower saturation value is 0x0. If the original data is larger than the upper saturation value, the result is $2^{\text{Ry}[3:0]}-1$, and in other cases the result is the original data itself. For the signed CLIP instruction, the upper saturation value is $2^{(\text{Ry}[3:0]-1)}-1$ and the lower saturation value is $-2^{(\text{Ry}[3:0]-1)}$. If the original data is greater than the upper saturation value, the result is $2^{(\text{Ry}[3:0]-1)}-1$, and if the original data is less than the lower saturation value, the result is $-2^{(\text{Ry}[3:0]-1)}$, in all other cases The result is the raw data itself.
M ar k er s of in fl	unaffected

User's Manual	
set a limit (on) System:	not have
discriminate Regular :	not have

32 in d e x (m at h.) hon orifi c title	
Ope rate : tell to Law :	<pre>Max= 2(Ry[3:0])-1, Min= 0 //unsigned Max = 2(Ry[3:0]-1)-1, Min = -2(Ry[3:0]-1) //signed if(Rx[31:16] Rx[15:0] > Max) Rz[31:16] Rz[15:0]= Max else if(Rx[31:16] Rx[15:0] <Min) Rz[31:16] Rz[15:0]= Min else Rz[31:16] Rz[15:0]= Rx[31:16] Rx[15:0]</pre>
Exp lana tion : M ar k er s of in fl u	<p>pclip.u16 rx, ry pclip.s16 rx, ry</p> <p>The high and low half-words of Rx are used as raw data, and are saved bit-scaled into the target data specified by Ry[3:0] respectively, where Ry[3 :0]4 characterizes the bit width of the target data. If the value of the original data exceeds the range of values that can be represented by the target data, saturation is performed. For the unsigned CLIP instruction, the upper saturation value is $2^{Ry[3:0]}-1$, and the lower saturation value is 0x0. If the original data is larger than the upper saturation value, the result is $2^{Ry[3:0]}-1$, and in other cases the result is the original data itself. For the signed CLIP instruction, the upper saturation value is $2^{(Ry[3:0]-1)}-1$ and the lower saturation value is $-2^{(Ry[3:0]-1)}$. If the original data is greater than the upper saturation value, the result is $2^{(Ry[3:0]-1)}-1$, and if the original data is less than the lower saturation value, the result is $-2^{(Ry[3:0]-1)}$, in all other cases</p> <p>The result is the raw data itself.</p>
	unaffected

User's n ce Bit:	Manual
set a limi t (on) Syst em:	not have
disc rimi nate Reg ular :	not have

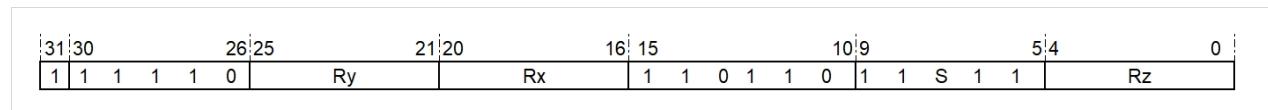
Command Format:

Figure 15.74: PCLIP.(US)16

15.75 PABS.S8.S - 8-bit Parallel Absolute Instruction with Saturation Operation

collectively known as "harmonization" or "harmonization index" (math.) honorific title	
grammatical	manipulate
pabs.s8.s rz, rx	$Rz[31:24] = \text{Saturate}(\text{abs}(Rx[31:24]))$ $Rz[23:16] = \text{Saturate}(\text{abs}(Rx[23:16]))$ $Rz[15:8]=\text{Saturate}(\text{abs}(Rx[15:8]))$ $Rz[7:0]=\text{Saturate}(\text{abs}(Rx[7:0]))$
Compilation results	Only 32-bit instructions exist. pabs.s8.s rz, rx

Description:	The bytes of Rx are saturated by taking the absolute value of the bytes in turn, in units of bytes. the byte is 0x80, the absolute value is saturated The result of the processing is 0x7F, and the rest of the cases are the same as the absolute result.
Impact indicator	unaffected
Chi Bit:	
Limitations:	not have
Exception:	not have

32-bit 32-bit (computing)	
---------------------------------	--

directive User's Manual	
Operatio n:	Rz[31:24] = Saturate(abs(Rx[31:24])) Rz[23:16] = Saturate(abs(Rx[23:16])) Rz[15:8]= Saturate(abs(Rx[15:8])) Rz[7:0]= Saturate(abs(Rx[7:0]))
Gramma r:	pabs.s8.s rz, rx
Description:	The bytes of Rx are saturated by taking the absolute value of the bytes in turn in byte units. the byte is 0x8000, the absolute value is saturated The result of the processing is 0xFFFF, the rest of the cases are the same as the absolute result.
affect Chi Bit:	unaffected
Limitati ons:	not have
Exceptio n:	not have

Command Format:

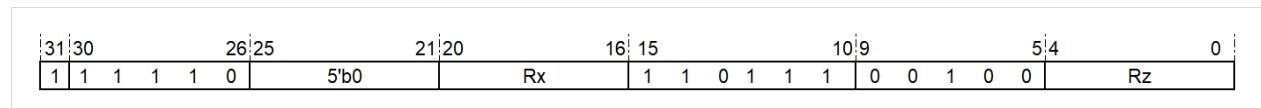


Figure 15.75: PABS.

15.76 PABS.S16.S - 16-bit Parallel Absolute Instruction with Saturation Operation

Harmonization Directive	
grammatical	pabs.s16.s rz, rx
manipulate	Rz[31:16]= Saturate(abs(Rx[31:16])) Rz[15:0]= Saturate(abs(Rx[15:0]))
Compilation results	Only 32-bit instructions exist. pabs.s16.s rz, rx

Description:	The high and low half words of Rx are taken as absolute values in turn and saturated. the half-word is 0x8000, the absolute value saturation processing is done. The result is 0x7FFF, the rest of the cases are the same as the absolute value result.
impact indicator Chi Bit:	unaffected
Limitations:	not have
Exception:	not have

32-bit finger honorific title	
Operation:	Rz[31:16]= Saturate(abs(Rx[31:16])) Rz[15:0]= Saturate(abs(Rx[15:0]))
Grammar:	pabs.s16.s rz, rx
Description:	The high and low half words of Rx are taken as absolute values in turn and saturated. the half-word is 0x8000, the result of absolute value saturation process is 0x7FFF, the rest of the cases are the same as the absolute value result.
impact indicator Chi Bit:	unaffected
Limitations:	not have

Exceptional User's Manual
n:

Command Format:

:31:30	26:25	21:20	16:15	10:9	5:4	0:
[1 1 1 1 0]	5'b0	Rx	[1 1 0 1 1]	[0 0 1 0 1]	Rz	

Figure 15.76: PABS.S16.

15.77 ABS.S32.S - 32-bit absolute instruction with saturation operation

Harmonization Directive	
grammatical	abs.s32.s rz, rx
manipulate	Rz[31:0]= Saturate(abs(Rx[31:0]))
Compilation results	Only 32-bit instructions exist. abs.s32.s rz, rx

Description:	Take the absolute value of Rx and saturate it. Rx is 0x8000 0000, the result of the absolute value saturation process is 0x7FFF FFFF, and the rest of the cases are the same as the absolute results.
impact indicator	unaffected
Chi Bit:	
Limitations:	not have
Exception:	not have

32-bit finger honorific title	
Operation:	Rz[31:0]= Saturate(abs(Rx[31:0]))
Grammar:	abs.s32.s rz, rx
Description:	Take the absolute value of Rx and saturate it. Rx is 0x8000 0000, the result of the absolute value saturation process is 0x7FFF FFFF, and the rest of the cases are the same as the absolute results.
impact indicator	unaffected
Chi Bit:	
Limitations:	not have
Exception:	not have

Command Format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 1 1 0	5'b0	Rx	1 1 0 1 1	0 0 1 1 0	Rz	

Figure 15.77: ABS.S32.

15.78 PNEG.S8.S - 8-bit parallel inverse instruction with saturation operation

standardization directives	
grammatical	pneg.s8.s rz, rx
manipulate	$Rz[31:24] = \text{Saturate}(\text{neg}(Rx[31:24]))$ $Rz[23:16] = \text{Saturate}(\text{neg}(Rx[23:16]))$ $Rz[15:8] = \text{Saturate}(\text{neg}(Rx[15:8]))$ $Rz[7:0] = \text{Saturate}(\text{neg}(Rx[7:0]))$
compilation result resolute	Only 32-bit instructions exist. pneg.s8.s rz, rx

32-bit finger honorific title	
Operation:	$Rz[31:24] = \text{Saturate}(\text{neg}(Rx[31:24]))$ $Rz[23:16] = \text{Saturate}(\text{neg}(Rx[23:16]))$ $Rz[15:8] = \text{Saturate}(\text{neg}(Rx[15:8]))$ $Rz[7:0] = \text{Saturate}(\text{neg}(Rx[7:0]))$
Grammar:	pneg.s8.s rz, rx
Description:	The bytes of Rx are inverted and saturated in byte order. the byte is 0x80, the inverse saturation process of The result is 0x7F, and the rest of the cases are the same as the inverse result.
affect Chi Bit:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1	1	1	1	0	5'b0	Rx	1	1	0	1	Rz

Figure 15.78: PNEG.

15.79 PNEG.S16.S - 16 bit parallel inverse instruction with saturation operation

Harmonization Directive	
grammatical	pneg.s16.s rz, rx
manipulate	Rz[31:16]= Saturate(neg(Rx[31:16])) Rz[15:0]= Saturate(neg(Rx[15:0]))
Compilation results	Only 32-bit instructions exist. pneg.s16.s rz, rx

Description:	The high and low half words of Rx are inverted in turn and saturated. the half-word is 0x8000, the result of the inverse saturation process is 0x7FFF, the rest of the cases are the same as the inverse result.
impact indicator Chi Bit:	unaffected
Limitations:	not have
Exception:	not have

32-bit finger honorific title	
Operation:	Rz[31:16]= Saturate(neg(Rx[31:16])) Rz[15:0]= Saturate(neg(Rx[15:0]))
Grammar:	pneg.s16.s rz, rx
Description:	The high and low half words of Rx are inverted in turn and saturated. the half-word is 0x8000, the result of the inverse saturation process is 0x7FFF, the rest of the cases are the same as the inverse result.
impact indicator Chi Bit:	unaffected
Limitations:	not have

Exceptional User's Manual
n:

Command Format:

:31:30	26:25	21:20	16:15	10:9	5:4	0:
[1 1 1 1 0]	5'b0	Rx	[1 1 0 1 1]	[0 1 1 0 1]	Rz	

Figure 15.79: PNEG.S16.

15.80 NEG.S32.S - Inverse instruction with saturation operation

Harmonization Directive	
grammatical	neg.s32.s rz, rx
manipulate	Rz[31:0]= Saturate(neg(Rx[31:0]))
Compilation results	Only 32-bit instructions exist. neg.s32.s rz, rx

Description:	Invert Rx and saturate. Rx is 0x8000 0000, the result of the inverse saturation process is 0x7FFF FFFF. The rest of the cases are the same as the absolute value results.
Markers of influence Bit:	unaffected
Limitations:	not have
Exception:	not have

32 Bitmap honorific title	
Operation:	Rz[31:0]= Saturate(neg(Rx[31:0]))
Grammatical:	neg.s32.s rz, rx
Description:	Invert Rx and saturate. Rx is 0x8000 0000, the result of the inverse saturation process is 0x7FFF FFFF. The rest of the cases are the same as the absolute value results.
Markers of influence Bit:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1	1	1	0	5'b0	Rx	1	1	0	1	1	0

Figure 15.80: NEG.S32.

15.81 DUP.8--8 Bit Operand Copy Instruction

unify chemicalization honori fic title	
gram matica l	dup.8 rz, rx, index
manip ulate	Element = Rx[7:0] //index= 0 Element = Rx[15:8] //index= 1 Element = Rx[23:16] //index= 2 Element= Rx[31:24] //index= 3 Rz[31:0]= {Element, Element, Element, Element }
Translati on in the end	Only 32-bit instructions exist. dup.8 rz, rx, index

Description:	Define Rx[31:24], Rx[23:16], Rx[15 :8], Rx[7:0] as byte 3, byte 2, byte 1, and byte 0, respectively, as defined by index. Select the appropriate byte and copy it into 4 bytes of the Rx.
affect Chi Bit:	unaffected
Limitations:	Index ranges from 0 to 3
Exception:	not have

32-bit directives	
Operation:	Element = Rx[7:0] //index= 0 Element = Rx[15:8] //index= 1 Element = Rx[23:16] //index= 2 Element= Rx[31:24] //index= 3 Rz[31:0]= { Element, Element, Element, Element }
Gramm ar:	dup.8 rz, rx, index

Gentech E804

Description:	Define Rx[31:24], Rx[23:16], Rx[15 :8], Rx[7:0] as byte 3, byte 2, byte 1, and byte 0, respectively, as defined by index. Select the appropriate byte and copy it into 4 bytes of the Rx.
impact Logo Bit:	unaffected
Limitations:	Index ranges from 0 to 3
Exception:	not have

Command Format:

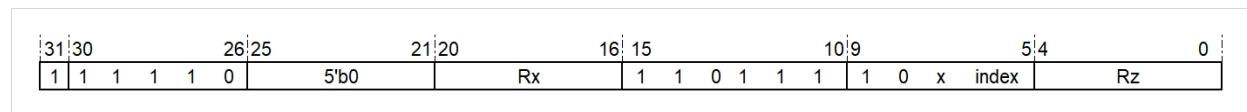


Figure 15.81: DUP.8

15.82 DUP.16--16 Bit Operand Copy Instructions

collectively known as "harmonization" or "harmonization index" (math.) honorific title	
grammatical	dup.16 rz, rx, index
manipulate	Element = Rx[15:0] //index= 0 Element= Rx[31:16] //index= 1 Rz[31:0]= { Element, Element }
Compilation results	Only 32-bit instructions exist. dup.16 rz, rx, index

Description:	Define Rx[31:16] and Rx[15:0] as halfword 1 and halfword 0. Select the corresponding halfword from index and copy it to Rx. in 2 half-words.
Markers of influence Bit:	unaffected
Limitations:	Index ranges from 0 to 1.
Exception:	not have

32-bit instruction	
--------------------	--

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	5'b0	Rx	1 1 0 1 1 1	1 0 x	index	Rz

n	
Operation:	Element = Rx[15:0] //index= 0 Element= Rx[31:16] //index= 1 Rz[31:0]= { Element, Element }
Grammar:	dup.16 rz, rx, index
Description:	Define Rx[31:16] and Rx[15:0] as halfword 1 and halfword 0. Select the corresponding halfword from index and copy it to Rx. in 2 half-words.
Markers of influence Bit:	unaffected
Limitations:	Index ranges from 0 to 1.
Exception:	not have

Command Format:

:31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	5'b0	Rx	1 1 0 1 1 1	1 0 x index	Rz	

Figure 15.82: DUP.16

15.83 MULS.(U/S)32 - 32-bit (without/with) signed multiply-accumulate instruction

Harmonization Directive	
grammatical	muls.u32 rz, rx, ry muls.s32 rz, rx, ry
manipulate	{Rz+1[31:0],Rz[31:0]} = {Rz+1[31:0],Rz[31:0]} - Rx[31:0] X Ry[31:0]
Compilation results	Only 32-bit instructions exist. muls.u32 rz, rx, ry muls.s32 rz, rx, ry

Description:	Rx is multiplied by Ry to get a 64-bit result and subtracted from {Rz+1, Rz}. The upper 32 bits of the result are stored in Rz+1. The lower 32 bits are stored in Rz.
Markers of influence Bit:	unaffected
Limitations:	not have
Exception:	not have

32-bit instruction	
Operation :	{Rz+1[31:0],Rz[31:0]} = {Rz+1[31:0],Rz[31:0]} - Rx[31:0] X Ry[31:0]
Grammar:	muls.u32 rz, rx, ry muls.s32 rz, rx, ry
Description:	Rx is multiplied by Ry to get a 64-bit result and subtracted from {Rz+1, Rz}. The upper 32 bits of the result are stored in Rz+1. The lower 32 bits are stored in Rz.

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	5'b0	Rx	1 1 0 1 1 1	1 0 x	index	Rz

Markers of influenc e Bit:	unaffected
Limitation s:	not have
Exception:	not have

Command Format:

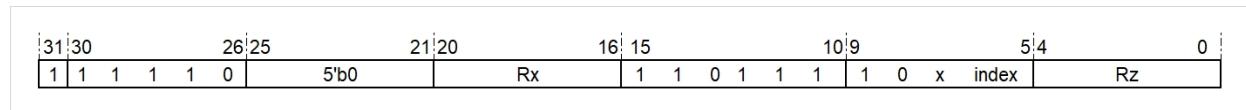


Figure 15.83: MULS.(US)32

15.84 MULA.(U/S)32.S - 32-bit (no/have) signed multiply-accumulate with saturation operation directives

Harmonization Directive	
grammatical	mula.u32.s rz, rx, ry mula.s32.s rz, rx, ry
manipulate	{Rz+1[31:0],Rz[31:0]}= Saturate({Rz+1[31:0],Rz[31:0]}+ Rx[31:0] X Ry[31:0])
Compilation results	Only 32-bit instructions exist. mula.u32.s rz, rx, ry mula.s32.s rz, rx, ry

Explanation:	Rx is multiplied by Ry to get a 64-bit result and added to {Rz+1, Rz}. The result is saturated and the high 32 bits are stored in the Rz+1, the lower 32 bits are deposited into Rz. For unsigned operations, saturation is handled as follows: if the result of the addition is greater than the saturation value 0xFFFF FFFF FFFF FFFF, the result is the saturation value, otherwise it is the result of the addition itself. For signed arithmetic, the saturation processing is as follows: if the result of addition is greater than the upper saturation value 0x7FFF FFFF FFFF FFFF, the result is the upper saturation value, if the result of addition is less than the lower saturation value 0x8000 0000 0000 0000, the result is the lower saturation value, and in other cases, the result is the plus The results of the law itself.
Markers of influence	unaffected

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	5'b0	Rx	1 1 0 1 1 1	1 0 x index		Rz

Bit:	
set a limit (on) System:	not have
discriminate Regular :	not have

32 in d e x (m at h.) hon orifi c title	
drill (prac tic e) Ma ke:	$\{Rz+1[31:0], Rz[31:0]\} = \text{Saturate}(\{Rz+1[31:0], Rz[31:0]\} + Rx[31:0] \times Ry[31:0])$
tell to Law :	mula.u32.s rz, rx, ry mula.s32.s rz, rx, ry
Explana tion :	Rx is multiplied by Ry to get a 64-bit result and added to {Rz+1, Rz}. The result is saturated and the high 32 bits are stored in the Rz+1, the lower 32 bits are deposited into Rz. For unsigned operations, saturation is handled as follows: if the result of the addition is greater than the saturation value 0xFFFF FFFF FFFF FFFF, the result is the saturation value, otherwise it is the result of the addition itself. For signed arithmetic, the saturation processing is as follows: if the result of addition is greater than the upper saturation value 0x7FFF FFFF FFFF FFFF, the result is the upper saturation value, if the result of addition is less than the lower saturation value 0x8000 0000 0000 0000, the result is the lower saturation value, and in other cases, the result is the plus The results of the law itself.
M ar k er s of in fl u e n	unaffected

User's Manual Bit:	
set a limit (on) System:	not have
discriminate Regular :	not have

Command Format:

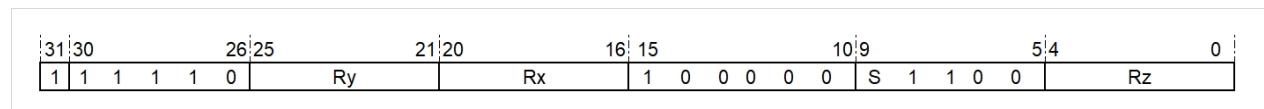


Figure 15.84: MULA.(US)32.

15.85 MULS.(U/S)32.S - 32-bit (without/with) signed multiply-accumulate with saturation operation directives

Harmonization Directive	
grammatical	muls.u32.s rz, rx, ry muls.s32.s rz, rx, ry
manipulate	{Rz+1[31:0],Rz[31:0]} = Saturate({Rz+1[31:0],Rz[31:0]} - Rx[31:0] X Ry[31:0])
Compilation results	Only 32-bit instructions exist. muls.u32.s rz, rx, ry muls.s32.s rz, rx, ry

Explanation :	Rx is multiplied by Ry to get a 64-bit result and subtracted from {Rz+1, Rz}. The result of the subtraction is saturated and the upper 32 bits are stored in the Rz+1, the lower 32 bits are deposited into Rz. For unsigned operations, saturation is handled as follows: if the result of the subtraction is less than the saturation value 0x0, the result is 0x0, otherwise it is the result of the subtraction itself. For signed arithmetic, the saturation processing is as follows: if the result of subtraction is greater than the upper saturation value 0x7FFF FFFF FFFF FFFF FFFF , the result is the upper saturation value, if the result of subtraction is less than the lower saturation value 0x8000 0000 0000 0000, the result is the lower saturation value, and in all other cases, the result is the subtraction of the The results of the law itself.
Mark ers of in fl u e n ce Bit:	unaffected
set a limi t (on) Syst	not have

User's Manual
discriminate regularly:

32 in d e x (m at h.) hon orifi c title	
drill (prac tic e) Ma ke:	$\{Rz+1[31:0], Rz[31:0]\} = \text{Saturate}(\{Rz+1[31:0], Rz[31:0]\} - Rx[31:0] \times Ry[31:0])$
tell to Law :	muls.u32.s rz, rx, ry muls.s32.s rz, rx, ry
Exp lana tion : :	Rx is multiplied by Ry to get a 64-bit result and subtracted from {Rz+1, Rz}. The result of the subtraction is saturated and the upper 32 bits are stored in the Rz+1, the lower 32 bits are deposited into Rz. For unsigned operations, saturation is handled as follows: if the result of the subtraction is less than the saturation value 0x0, the result is 0x0, otherwise it is the result of the subtraction itself. For signed arithmetic, the saturation processing is as follows: if the result of subtraction is greater than the upper saturation value 0x7FFF FFFF FFFF FFFF , the result is the upper saturation value, if the result of subtraction is less than the lower saturation value 0x8000 0000 0000 0000, the result is the lower saturation value, and in all other cases, the result is the subtraction of the The results of the law itself.
M ar k er s of in fl u e n	unaffected

User's Manual Bit:	
set a limit (on) System:	not have
discriminate Regular :	not have

Command Format:

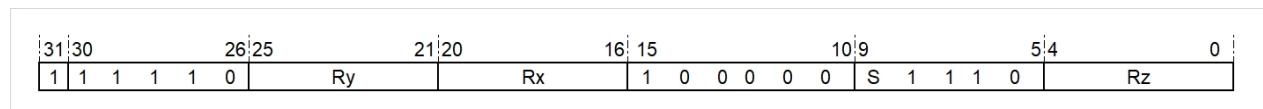


Figure 15.85: MULS.(US)32.

15.86 MUL.S32.H - 32-bit signed multiplication fetch-high 32-bit instruction

Harmonization Directive	
grammatical	mul.s32.h rz, rx, ry
manipulate	Rz[31:0]= {Rx[31:0] X Ry[31:0]}[63:32]
Compilation results	Only 32-bit instructions exist. mul.s32.h rz, rx, ry

Description:	Rx is multiplied by Ry to get a 64-bit result, where the upper 32 bits are stored in Rz.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	not have

32-bit instruction	
Operation:	$Rz[31:0] = \{Rx[31:0] \times Ry[31:0]\}[63:32]$
Grammar:	mul.s32.h rz, rx, ry
Description:	Rx is multiplied by Ry to get a 64-bit result, where the upper 32 bits are stored in Rz.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

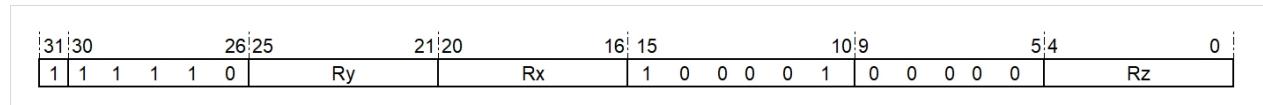


Figure 15.86: MUL.S32.

15.87 MUL.S32.RH - 32-bit signed multiplication with rounding takes the upper 32 bits of the finger.

honorific title

Harmonization Directive	
grammatical	mul.s32.rh rz, rx, ry
manipulate	$Rz[31:0] = \{Rx[31:0] \times Ry[31:0] + 32'h8000 0000\}[63:32]$
Compilation results	Only 32-bit instructions exist. mul.s32.rh rz, rx, ry

Description:	The 64-bit result obtained by multiplying Rx and Ry is added to 0x8000 0000, and the upper 32 bits of the result are intercepted and stored in Rz. The result of the multiplication is added to 0x8000 0000 to implement the rounding operation.
affect Chi Bit:	unaffected
Limitations:	not have
Exception:	not have

32-bit finger honorific title	
Operation:	Rz[31:0]= {Rx[31:0] X Ry[31:0]+ 32'h8000 0000}[63:32]
Grammar:	mul.s32.rh rz, rx, ry
Description:	The 64-bit result obtained by multiplying Rx and Ry is added to 0x8000 0000, and the upper 32 bits of the result are intercepted and stored in Rz. The result of the multiplication is added to 0x8000 0000 to implement the rounding operation.
affect Chi Bit:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 0	1 1 0 0 0 0	Rz	

Figure 15.87: MUL.S32.RH

15.88 RMUL.S32.H - 32-bit Signed Decimal Multiplication Fetch High 32-bit Instruction

standardization directives	
grammatical	rmul.s32.h rz, rx, ry
manipulate	if(Rx[31:0] == 32'h8000 0000 && Ry[31:0]== 32'h8000 0000) Rz[31:0]= 32'h7FFF FFFF else Rz[31:0]= {Rx[31:0] X Ry[31:0]}[62:31]
compilation	Only 32-bit instructions exist. rmul.s32.h rz, rx, ry

Description:	Rx and Ry are regarded as 32-bit signed decimals, in which the 31st bit is the sign bit and the 30th to 0th bits are decimal places. Rx=0x8000 0000且 Ry=0x8000 0000, the result is 0x7FFF FFFF; in other cases, take the 62nd digit of the result of multiplying Rx and Ry. to 31 bits are deposited into Rz.
Impact Sign s Bit:	unaffected
Limitations:	not have
Exception:	not have

32-bit directives	
Operation:	if(Rx[31:0] == 32'h8000 0000 && Ry[31:0]== 32'h8000 0000) Rz[31:0]= 32'h7FFF FFFF else Rz[31:0]= {Rx[31:0] X Ry[31:0]}[62:31]
Grammar:	rmul.s32.h rz, rx, ry
Description:	Rx and Ry are regarded as 32-bit signed decimals, in which the 31st bit is the sign bit and the 30th to 0th bits are decimal places. Rx=0x8000 0000且 Ry=0x8000 0000, the result is 0x7FFF FFFF; in other cases, take the 62nd digit of the result of multiplying Rx and Ry. to 31 bits are deposited into Rz.
Impact Signs Bit:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

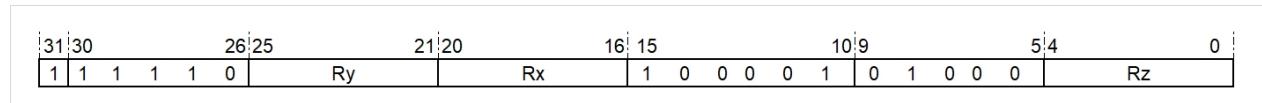


Figure 15.88: RMUL.

15.89 RMUL.S32.RH - 32-bit Signed Decimal Multiplication with Rounding Takes High 32 bit command (computing)

unify chemistry calibration honorific title	
--	--

USER'S Manual atrical	rmul.s32.rh rz, rx, ry
manip ulate	<pre>if(Rx[31:0] == 32'h8000 0000 && Ry[31:0]== 32'h8000 0000) Rz[31:0]= 32'h7FFF FFFF else Rz[31:0]= {Rx[31:0] X Ry[31:0]+ 32'h4000 0000}[62:31]</pre>
Trans lation in the end	<p>Only 32-bit instructions exist.</p> <p>rmul.s32.rh rz, rx, ry</p>

Exp lana tion :	Rx and Ry are regarded as 32-bit signed decimals, in which the 31st bit is the sign bit and the 30th~0th bits are decimal places. When Rx=0x8000 0000且 Ry=0x8000 0000, the result is 0x7FFF FF FF; in the rest of the cases, the 64-bit result obtained by multiplying Rx and Ry is added with 0x4000 0000, and bits 62~31 of the result are intercepted and stored in Rz. Among them, the result of the multiplication is added with 0x4000 0000 and stored in Rz. Implement rounding operations in decimal form.
M ar k er s of in fl u e n ce Bit:	unaffected
set a limit (on) Syst em:	not have
discr imin ate Reg ular:	not have

32 in d e x (m at h.) hon orifi c title	
--	--

Ope User's rate :	if(Rx[31:0] == 32'h8000 0000 && Ry[31:0]== 32'h8000 0000) Rz[31:0]= 32'h7FFF FFFF else Rz[31:0]= {Rx[31:0] X Ry[31:0]+ 32'h4000 0000}[62:31]
tell to Law:	rmul.s32.rh rz, rx, ry
Exp lana tion :	Rx and Ry are regarded as 32-bit signed decimals, in which the 31st bit is the sign bit and the 30th~0th bits are decimal places. When Rx=0x8000 0000且 Ry=0x8000 0000, the result is 0x7FFF FF FF; in the rest of the cases, the 64-bit result obtained by multiplying Rx and Ry is added with 0x4000 0000, and bits 62~31 of the result are intercepted and stored in Rz. Among them, the result of the multiplication is added with 0x4000 0000 and stored in Rz. Implement rounding operations in decimal form.
M ar k er s of in fl u e n ce Bit:	unaffected
set a limit (on) Syst em:	not have
discr imin ate Reg ular:	not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 0 1	1 1 0 0 0	Rz	

Figure 15.89: RMUL.S32.RH

15.90 MULA.S32.HS - 32-bit signed multiply-accumulate with saturation operation to take the high 32 bits directives

Harmonization Directive	
grammatical	mula.s32.hs rz, rx, ry
manipulate	$Rz[31:0] = \text{Saturate}(Rz[31:0] + \{Rx[31:0] \times Ry[31:0]\}[63:32])$
Compilation results	Only 32-bit instructions exist. mula.s32.hs rz, rx, ry

Explanation:	The 64-bit result obtained by multiplying Rx and Ry is taken as the higher 32 bits and added with Rz. The result is saturated and stored in Rz. The process of saturation is as follows: if the result is greater than the upper saturation value of 0x7FFF FFFF, the result is the upper saturation value; if the result is smaller than the upper saturation value, the result is the upper saturation value; if the result is smaller than the lower saturation value, the result is the lower saturation value. The result is the lower saturated value 0x8000 0000, in all other cases it is the result of the addition itself.
Markers of influence Bit:	unaffected
set a limit (on) System	not have

User's Manual	
discriminate Regulator:	not have

32 in d e x (m at h.) hono rific title	
drill (prac tice) Mak e:	Rz[31:0]= Saturate(Rz[31:0]+ {Rx[31:0] X Ry[31:0]}[63:32])
tell to Law:	mula.s32.hs rz, rx, ry
Exp lana tion : :	The 64-bit result obtained by multiplying Rx and Ry is taken as the higher 32 bits and added with Rz. The result is saturated and stored in Rz. The process of saturation is as follows: if the result is greater than the upper saturation value of 0x7FFF FFFF, the result is the upper saturation value; if the result is smaller than the upper saturation value, the result is the lower saturation value; if the result is smaller than the lower saturation value, the result is the lower saturation value. The result is the lower saturated value 0x8000 0000, in all other cases it is the result of the addition itself.
Mar k er s of in fl u e n ce Bit:	unaffected
set a limit (on) Syste m:	not have

discriminate	Manual
minate	
Regula	
lar:	

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 0	1 0 1 1 0 0	Rz	

Figure 15.90: MULA.S32.HS

15.91 MULS.S32.HS - 32-bit signed multiply-accumulate with saturation operation to fetch the high 32 bits directives

Harmonization Directive	
grammatical	muls.s32.hs rz, rx, ry
manipulate	$Rz[31:0] = \text{Saturate}(Rz[31:0] - \{Rx[31:0] \times Ry[31:0]\}[63:32])$
Compilation results	Only 32-bit instructions exist. muls.s32.hs rz, rx, ry

Exp lana tion :	The 64-bit result obtained by multiplying Rx and Ry is taken as the higher 32 bits and subtracted from Rz. The result of the subtraction is saturated and stored in Rz. The saturation process is as follows: if the result of the subtraction is greater than the upper saturation value of 0x7FFF FFFF, the result is the upper saturation value; if the result of the subtraction is smaller than the upper saturation value, the result is the lower saturation value. In the case of the lower saturation value 0x8000 0000, the result is the lower saturation value, in all other cases it is the result of the subtraction itself.
M ar k er s of in fl u e n ce Bit:	unaffected
set a limit (on) Syste m:	not have
discri mina te Regu lar:	not have

32 in d e x (m at h.) hono rific title	
--	--

drill User's Manual (practice) Make:	Rz[31:0]= Saturate(Rz[31:0] - {Rx[31:0] X Ry[31:0]}[63:32])
tell to Law:	muls.s32.hs rz, rx, ry
Explana- tion: tion:	The 64-bit result obtained by multiplying Rx and Ry is taken as the higher 32 bits and subtracted from Rz. The result of the subtraction is saturated and stored in Rz. The saturation process is as follows: if the result of the subtraction is greater than the upper saturation value of 0x7FFF FFFF, the result is the upper saturation value; if the result of the subtraction is smaller than the upper saturation value, the result is the lower saturation value. In the case of the lower saturation value 0x8000 0000, the result is the lower saturation value, in all other cases it is the result of the subtraction itself.
Mark- er s of in- fl- u- e- n- ce Bit:	unaffected
set a limit (on) System:	not have
discri- mina- te Regu- lar:	not have

Command Format:

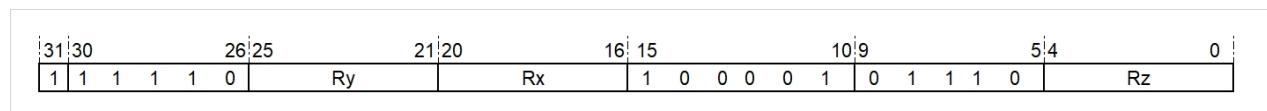


Figure 15.91: MULS.S32.HS

15.92 MULA.S32.RHS - 32-bit signed multiply-accumulate fetch with rounding and saturation operations

High 32-bit instructions

collectively known as "harmonization" or "harmonization index" (math.) honorific title	
grammatical	mula.s32.rhs rz, rx, ry
manipulate	$Rz[31:0] = \text{Saturate}(\{Rz[31:0], 32'h0000 0000\} + Rx[31:0] \times Ry[31:0] + 32'h8000 0000)[63:32]$
Compilation results	Only 32-bit instructions exist. mula.s32.rhs rz, rx, ry

Explanation :	The 64-bit result of multiplying Rx and Ry is added to $\{Rz, 32'h0000 0000\}$, and the result is added to $0x8000 0000$, the upper 32 bits of the result are intercepted and saturated, and then stored in Rz. The result is added to $0x8000 0000$ to implement rounding operation. The process of saturation processing is that if the addition result is greater than the upper saturation value $0x7FFF FFFF$, the result is the upper saturation value, if the addition result is less than the lower saturated value $0x8000 0000$, the result is the lower saturated value, otherwise it is the addition result itself.
Markers of influence	unaffected

User's Manual Bit:	
set a limit (on) System:	not have
discriminate Register:	not have

32 in d e x (m at h.) hon orifi c title	
drill (prac tic e) Ma ke:	Rz[31:0]= Saturate(({Rz[31:0], 32'h0000 0000}+ Rx[31:0] X Ry[31:0]+ 32'h8000 0000)[63:32])
tell to Law :	mula.s32.rhs Rz, Rx, Ry
Exp lana tion : :	The 64-bit result of multiplying Rx and Ry is added to {Rz, 32'h0000 0000}, and the result is added to 0x8000 0000, the upper 32 bits of the result are intercepted and saturated, and then stored in Rz. The result is added to 0x8000 0000 to implement rounding operation. The process of saturation processing is that if the addition result is greater than the upper saturation value 0x7FFF FFFF , the result is the upper saturation value, if the addition result is less than the lower saturated value 0x8000 0000, the result is the lower saturated value, otherwise it is the addition result itself.
M ar k er s of in fl u e n ce Bit:	unaffected

User's Method	not have
limit (on) System:	discriminate Regular :

Command Format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1	1 1 1 0 0	Rz	

Figure 15.92: MULA.S32.RHS

15.93 MULS.S32.RHS - 32-bit signed multiply-accumulate fetch with rounding and saturation operations

High 32-bit instructions

collectively known as "harmonization" or "harmonization index" (math.) honorific title	
grammatical	muls.s32.rhs rz, rx, ry
manipulate	Rz[31:0]= Saturate(({Rz[31:0], 32'h0000 0000} - Rx[31:0] X Ry[31:0]+ 32'h8000 0000)[63:32])
Compilation results	Only 32-bit instructions exist. muls.s32.rhs rz, rx, ry

Explanation :	The 64-bit result of multiplying Rx and Ry is subtracted from {Rz, 32'h0000 0000}, and the result is added with 0x8000 0000, the upper 32 bits of the result are intercepted and saturated and stored into Rz, where the result is added with 0x8000 0000 to implement rounding operation. The process of saturation processing is that if the addition result is greater than the upper saturation value 0x7FFF FFFF , the result is the upper saturation value, if the addition result is less than the lower saturated value 0x8000 0000, the result is the lower saturated value, otherwise it is the addition result itself.
Markers of influence	unaffected

User's Manual Bit:	
set a limit (on) System:	not have
discriminate Register:	not have

32 in d e x (m at h.) hon orifi c title	
drill (prac tic e) Ma ke:	Rz[31:0]= Saturate(({Rz[31:0], 32'h0000 0000} - Rx[31:0] X Ry[31:0]+ 32'h8000 0000)[63:32])
tell to Law :	muls.s32.rhs rz, rx, ry
Exp lana tion : : : :	The 64-bit result of multiplying Rx and Ry is subtracted from {Rz, 32'h0000 0000}, and the result is added with 0x8000 0000, the upper 32 bits of the result are intercepted and saturated and stored into Rz, where the result is added with 0x8000 0000 to implement rounding operation. The process of saturation processing is that if the addition result is greater than the upper saturation value 0x7FFF FFFF , the result is the upper saturation value, if the addition result is less than the lower saturated value 0x8000 0000, the result is the lower saturated value, otherwise it is the addition result itself.
M ar k er s of in fl u e n ce Bit:	unaffected

User's Manual	not have
limit (on) System:	discriminate Register:

Command Format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1	1 1 1 1 0	Rz	

Figure 15.93: MULS.S32.RHS

15.94 MULXL.S32 - 32 bit signed low halfword unaligned multiply instruction

Harmonization Directive	
grammatical	mulxl.s32 Rz, Rx, Ry
manipulate	$Rz[31:0] = \{Rx[31:0] \times Ry[15:0]\}[47:16]$
Compilation results	Only 32-bit instructions exist. mulxl.s32 Rz, Rx, Ry

Description:	Rx is multiplied by the lower half word of Ry and the higher 32 bits of the result are stored in Rz.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	not have

32-bit instruction:	
Operation:	$Rz[31:0] = \{Rx[31:0] \times Ry[15:0]\}[47:16]$
Grammar:	mulxl.s32 Rz, Rx, Ry
Description:	Rx is multiplied by the lower half word of Ry and the higher 32 bits of the result are stored in Rz.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

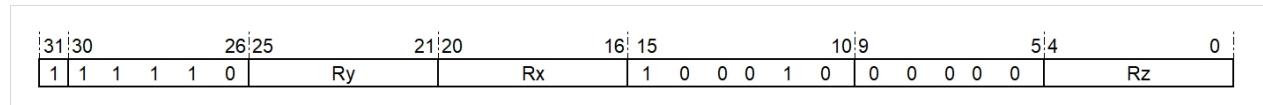


Figure 15.94: MULXL.S32

15.95 MULXL.S32.R - 32-bit Signed Low Halfword Unaligned Multiply with Rounding Operation

directives

Harmonization Directive	
grammatical	mulxl.s32.r rz, rx, ry
manipulate	$Rz[31:0] = \{Rx[31:0] \times Ry[15:0] + 16'h8000\}[47:16]$
Compilation results	Only 32-bit instructions exist. mulxl.s32.r rz, rx, ry

Description:	Multiply Rx with the lower half word of Ry and add 0x8000, take the upper 32 bits of the result and store them in Rz. Add 0x8000 to the multiplication result. Used to implement rounding operations.
Markers of influence Bit:	unaffected
Limitations:	not have
Exceptions:	not have

32-bit fingerpri nt honorific title	
Operation :	$Rz[31:0] = \{Rx[31:0] \times Ry[15:0] + 16'h8000\}[47:16]$
Grammar:	mulxl.s32.r rz, rx, ry
Description:	Multiply Rx with the lower half word of Ry and add 0x8000, take the upper 32 bits of the result and store them in Rz. Add 0x8000 to the multiplication result. Used to implement rounding operations.
Markers of influenc e Bit:	unaffected
Limitation s:	not have
Exception :	not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 0	1 0 0 0 0	Rz	

Figure 15.95: MULXL.S32.

15.96 MULXH.S32 - 32 bit signed high halfword unaligned multiply instruction

Harmoniza tion Directive	
grammatica l	mulxh.s32 rz, rx, ry
manipulate	$Rz[31:0] = \{Rx[31:0] \times Ry[31:16]\}[47:16]$
Compilatio n results	Only 32-bit instructions exist. mulxh.s32 rz, rx, ry

Description:	Rx is multiplied by the high halfword of Ry, and the high 32 bits of the result are stored in Rz.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	not have

32-bit instruction	
Operation:	$Rz[31:0] = \{Rx[31:0] \times Ry[31:16]\}[47:16]$
Grammar:	mulhx.s32 rz, rx, ry
Description:	Rx is multiplied by the high halfword of Ry, and the high 32 bits of the result are stored in Rz.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1	1 0 0 0 0	Rz	

Figure 15.96: MULXH.S32

15.97 MULXH.S32.R - 32-bit Signed High Halfword Unaligned Multiplication with Rounding Operations directives

Harmonization Directive	
grammatical	mulxh.s32.r rz, rx, ry
manipulate	$Rz[31:0] = \{Rx[31:0] X Ry[31:16] + 16'h8000\}[47:16]$
Compilation results	Only 32-bit instructions exist. mulxh.s32.r rz, rx, ry

Description:	Multiply Rx with the high half word of Ry and add 0x8000, take the high 32 bits of the result and store them in Rz. Add 0x8000 to the multiplication result. Used to implement rounding operations.
Markers of influence Bit:	unaffected
Limitations:	not have
Exceptions:	not have

32-bit fingerpri nt honorific title	
Operation	$Rz[31:0] = \{Rx[31:0] X Ry[31:16] + 16'h8000\}[47:16]$

User's Manual	
Grammar:	mulxh.s32.r rz, rx, ry
Description:	Multiply Rx with the high half word of Ry and add 0x8000, take the high 32 bits of the result and store them in Rz. Add 0x8000 to the multiplication result. Used to implement rounding operations.
Markers of influence Bit:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 1	1 0 0 0 0	Rz	

Figure 15.97: MULXH.S32.

15.98 RMULXL.S32 - 32-bit signed low half-word unaligned decimal multiply instruction

standardization directives	
grammatical	rmulxl.s32 rz, rx, ry
manipulate	if(Rx[31:0] == 32'h8000 0000 && Ry[15:0] == 16'h8000) Rz[31:0]= 32'h7FFF FFFF else Rz[31:0]= {Rx[31:0] X Ry[15:0]}[46:15]
compilation result resolute	Only 32-bit instructions exist. rmulxl.s32 rz, rx, ry

Explanation:	The lower half words of Rx and Ry are regarded as signed decimals, where the 31st bit of Rx is a sign bit and the 30th~0th bits are decimal bits, and the 15th of Ry is a sign bit and the 14th~0th bits are decimal bits. Rx[31:0]=0x8000 0000且 Ry[15:0]=0x8000, the result will be 0x7FFF. FFFF; in the rest of the cases, take the 46th to 15th bits of the result of multiplying Rx and Ry and deposit them into Rz.
Marks of influence Bit:	unaffected

Set a limit (on) System:	not have
discriminate Regular:	

32 in d e x (m at h.) hono rific title	
Ope rate : tell to Law:	<pre>if(Rx[31:0] == 32'h8000 0000 && Ry[15:0] == 16'h8000) Rz[31:0]= 32'h7FFF FFFF else Rz[31:0]= {Rx[31:0] X Ry[15:0]}[46:15]</pre> <p>rmulxl.s32 rz, rx, ry</p>
Exp lana tion : Bit:	<p>The lower half words of Rx and Ry are regarded as signed decimals, where the 31st bit of Rx is a sign bit and the 30th~0th bits are decimal bits, and the 15th of Ry is a sign bit and the 14th~0th bits are decimal bits. Rx[31:0]=0x8000 0000且 Ry[15:0]=0x8000, the result will be 0x7FFF. FFFF; in the rest of the cases, take the 46th to 15th bits of the result of multiplying Rx and Ry and deposit them into Rz.</p> <p>unaffected</p>
set a limit (on) Syste m: discrimina te Regu	<p>not have</p> <p>not have</p>

Command Format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 0	0 1 0 0 0	Rz	

Figure 15.98: RMULXL.S32

15.99 RMULXL.S32.R - 32-bit Signed Low Halfword Unaligned Small with Rounding Operation

number multiplication instruction

standardization directives	
grammatical	rmulxl.s32.r rz, rx, ry
manipulate	if(Rx[31:0] == 32'h8000 0000 && Ry[15:0] == 16'h8000) Rz[31:0]= 32'h7FFF FFFF else Rz[31:0]= {Rx[31:0] X Ry[15:0]+ 16'h4000}[46:15]
compilation result resolute	Only 32-bit instructions exist. rmulxl.s32.r rz, rx, ry

Explanation :	The lower half words of Rx and Ry are regarded as signed decimals, where the 31st bit of Rx is the sign bit and the 30th~0th bits are decimal bits, and the 15th bit of Ry is the sign bit and the 14th~0th bits are decimal bits. When Rx[31:0]=0x8000 0000且 Ry[15:0]=0x8000, the result is 0x7FFF FFFF; in the rest of the cases, multiply Rx with the lower half word of Ry and add 0x4000, and take the 46th to 15th bit of the result and store it into Rz. The result of the method is added to 0x4000 to implement rounding in decimal form.
Mark er s of in fl ue n ce Bit:	unaffected
set a limit (on)	not have

Users Manual
em:
disc
rimi
nate
Reg
ular
:

32 in d e x (m at h.) hon orifi c title	
Ope rate : tell to Law :	<pre>if(Rx[31:0] == 32'h8000 0000 && Ry[15:0] == 16'h8000) Rz[31:0]= 32'h7FFF FFFF else Rz[31:0]= {Rx[31:0] X Ry[15:0]+ 16'h4000}[46:15]</pre>
Explaina tion : Mark er s of in fl u e n ce Bit:	<p>The lower half words of Rx and Ry are regarded as signed decimals, where the 31st bit of Rx is the sign bit and the 30th~0th bits are decimal bits, and the 15th bit of Ry is the sign bit and the 14th~0th bits are decimal bits. When Rx[31:0]=0x8000 0000且 Ry[15:0]=0x8000, the result is 0x7FFF FFFF; in the rest of the cases, multiply Rx with the lower half word of Ry and add 0x4000, and take the 46th to 15th bit of the result and store it into Rz.</p> <p>The result of the method is added to 0x4000 to implement rounding in decimal form.</p>
set a limi t (on) Syst	unaffected

User's Manual	
disc rimi nate Reg ular :	not have

Command Format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 1 1 1 0	Ry	Rx	1 0 0 0 1 0	1 1 0 0 0	Rz	

Figure 15.99: RMULXL.S32.

15.100 RMULXH.S32 - 32 bit signed high half-word unaligned decimal multiply instruction

standardization directives	
grammatical	rmulxh.s32 rz, rx, ry
manipulate	<pre>if(Rx[31:0]== 32'h8000 0000 && Ry[31:16]== 16'h8000) Rz[31:0]= 32'h7FFF FFFF else Rz[31:0]= {Rx[31:0] X Ry[31:16]}[46:15]</pre>
compilation result resolute	<p>Only 32-bit instructions exist.</p> <p>rmulxh.s32 rz, rx, ry</p>

Explanation :	The lower half of Rx and Ry are considered as signed decimals, where the 31st bit of Rx is the sign bit, the 30th to 0th bits are decimal places, and the 31st bit of Ry is the sign bit, the 30th to 0th bits are decimal places, and the 31st bit of Ry is the sign bit. The 30th to 16th bits are sign bits, and the 30th to 16th bits are decimal places. Rx[31:0]=0x8000 0000 且 Ry[31:16]=0x8000, the result is 0x7FFF FFFF; for the rest of the cases, bits 46 to 15 of the multiplication result of Rx and Ry are taken and stored in Rz.
Mark ers of influence Bit:	unaffected
set a limit (on) Syste	not have

User's Manual	
discriminate Regulator:	not have

32 in d e x (m at h.) hono rific title	
Ope rate : tell to Law:	<pre>if(Rx[31:0]== 32'h8000 0000 && Ry[31:16]== 16'h8000) Rz[31:0]= 32'h7FFF FFFF else Rz[31:0]= {Rx[31:0] X Ry[31:16]}[46:15]</pre> <p>rmulxh.s32 rz, rx, ry</p>
Exp lana tion : M ar k er s of in fl u e n ce Bit:	<p>The lower half of Rx and Ry are considered as signed decimals, where the 31st bit of Rx is the sign bit, the 30th to 0th bits are decimal places, and the 31st bit of Ry is the sign bit, the 30th to 0th bits are decimal places, and the 31st bit of Ry is the sign bit.</p> <p>The 30th to 16th bits are sign bits, and the 30th to 16th bits are decimal places. Rx[31:0]=0x8000 0000 且 Ry[31:16]=0x8000, the result is 0x7FFF FFFF; for the rest of the cases, bits 46 to 15 of the multiplication result of Rx and Ry are taken and stored in Rz.</p> <p>unaffected</p>
set a limit (on) Syste m:	not have

discri
Users Manual
mina
te
Regu
lar:

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 1	0 1 0 0 0	Rz	

Figure 15.100: RMULXH.S32

15.101 RMULXH.S32.R - 32 bits with rounding operation Signed High Halfword Unaligned fractional multiplication instruction

standardization directives	
grammatical	rmulxh.s32.r rz, rx, ry
manipulate	<pre>if(Rx[31:0]== 32'h8000 0000 && Ry[31:16]== 16'h8000) Rz[31:0]= 32'h7FFF FFFF else Rz[31:0]= {Rx[31:0] X Ry[31:16]+ 16'h4000}[46:15]</pre>
compilation result resolute	<p>Only 32-bit instructions exist.</p> <p>rmulxh.s32.r rz, rx, ry</p>

Explanation :	The lower half words of Rx and Ry are regarded as signed decimals, where the 31st bit of Rx is the sign bit and the 30th~0th bits are decimal bits, and the 15th bit of Ry is the sign bit and the 14th~0th bits are decimal bits. When Rx[31:0]=0x8000 0000且 Ry[15:0]=0x8000, the result is 0x7FFF FFFF; in the rest of the cases, multiply Rx with the lower half word of Ry and add 0x4000, and take the 46th to 15th bit of the result and store it into Rz. The result of the method is added to 0x4000 to implement rounding in decimal form.
Mark er s of in fl ue n ce Bit:	unaffected
set a limi t (on)	not have

Users Manual
em:
disc
rimi
nate
Reg
ular
:

32 in d e x (m at h.) hon orifi c title	
Ope rate : tell to Law :	<pre>if(Rx[31:0]== 32'h8000 0000 && Ry[31:16]== 16'h8000) Rz[31:0]= 32'h7FFF FFFF else Rz[31:0]= {Rx[31:0] X Ry[31:16]+ 16'h4000}[46:15]</pre>
Explaina tion : Mark er s of in fl u e n ce Bit:	<p>The lower half words of Rx and Ry are regarded as signed decimals, where the 31st bit of Rx is the sign bit and the 30th~0th bits are decimal bits, and the 15th bit of Ry is the sign bit and the 14th~0th bits are decimal bits. When Rx[31:0]=0x8000 0000且 Ry[15:0]=0x8000, the result is 0x7FFF FFFF; in the rest of the cases, multiply Rx with the lower half word of Ry and add 0x4000, and take the 46th to 15th bit of the result and store it into Rz.</p> <p>The result of the method is added to 0x4000 to implement rounding in decimal form.</p>
set a limi t (on) Syst	unaffected
not have	

User's Manual	
discriminate	not have
Register:	

Command Format:

:31:30	26:25	21:20	16:15	10:9	5:4	0
[1 1 1 1 0]	Ry	Rx	[1 0 0 0 1 1]	[1 1 0 0 0]	Rz	

Figure 15.101: RMULXH.S32.

15.102 MULAXL.S32.S - 32-bit Signed Low Halfword Unaligned with Saturation Operation multiply-accumulate instruction

Harmonization Directive	
grammatical	mulaxl.s32.s rz, rx, ry
manipulation	Rz[31:0]= Saturate(Rz[31:0]+ {Rx[31:0] X Ry[15:0]}[47:16])
Compilation results	Only 32-bit instructions exist. mulaxl.s32.s rz, rx, ry

Explana tion: :	Rx is multiplied by the lower half word of Ry and the higher 32 bits of the result are added to Rz, and the result is saturated and stored in Rz. The process of saturation processing is that if the addition result is greater than the upper saturation value 0x7FFF FFFF, the result is the upper saturation value, if the addition result is smaller The result is the lower saturated value 0x8000 0000, in all other cases it is the result of the addition itself.
Mark ers of infl uen ce Bit:	unaffected
set a limit (on) Syste m:	not have
discri mina te Regu lar:	not have

32 inde x (mat h.) hono rific title	
drill (prac tice) Mak e:	Rz[31:0]= Saturate(Rz[31:0]+ {Rx[31:0] X Ry[15:0]}[47:16])
tell to Law:	mulaxl.s32.s rz, rx, ry
Explana tion	Rx is multiplied by the lower half word of Ry and the higher 32 bits of the result are added to Rz, and the result is saturated and stored in Rz. The process of saturation processing is that if the addition result is greater than the upper

User's Manual:	saturation value 0x7FFF FFFF , the result is the upper saturation value, if the addition result is smaller The result is the lower saturated value 0x8000 0000, in all other cases it is the result of the addition itself.
Markers of influence Bit:	unaffected
set a limit (on) System:	not have
discriminate Regular:	not have

Command Format:

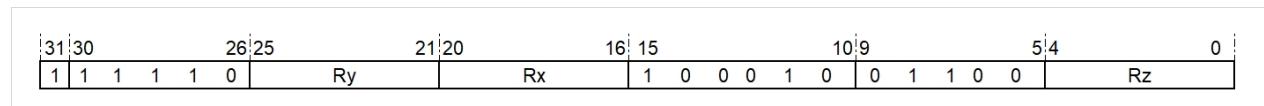


Figure 15.102: MULAXL.S32.

15.103 MULAXL.S32.RS - 32 bit signed lower half with rounding and saturation operations

Word unaligned multiply-accumulate instruction

Harmonization Directive	
grammatical	mulaxl.s32.rs rz, rx, ry
manipulate	Rz[31:0]= Saturate(({Rz[31:0], 16'h0000}+ Rx[31:0] X Ry[15:0]+ 16'h8000)[47:16])
Compilation results	Only 32-bit instructions exist. mulaxl.s32.rs rz, rx, ry

Explanation:	Rx is multiplied by the lower half word of Ry and added to {Rz, 16'h0000} and the result is added to 0x8000. The upper 32 bits of the result are saturated and stored in Rz. The result is added to 0x8000 to realize the rounding operation. The process of saturation processing is that if the addition result is greater than the upper saturation value 0x7FFF FFFF , the result is the upper saturation value, if the addition result is less than the lower saturated value 0x8000 0000, the result is the lower saturated value, otherwise it is the addition result itself.
Mark Bit:	unaffected
set a limit (on) System:	not have
discriminate Register:	not have

32 in d e x (m at h.) hon orifi c title	
drill (pra ctice) Mak e:	Rz[31:0]= Saturate(({Rz[31:0], 16'h0000}+ Rx[31:0] X Ry[15:0]+ 16'h8000)[47:16])
tell to Law : Exp lana tion : :	<p>mulaxl.s32.rs rz, rx, ry</p> <p>Rx is multiplied by the lower half word of Ry and added to {Rz, 16'h0000} and the result is added to 0x8000. The upper 32 bits of the result are saturated and stored in Rz. The result is added to 0x8000 to realize the rounding operation.</p> <p>The process of saturation processing is that if the addition result is greater than the upper saturation value 0x7FFF FFFF , the result is the upper saturation value, if the addition result is less than the lower saturated value 0x8000 0000, the result is the lower saturated value, otherwise it is the addition result itself.</p>
M ar k er s of in fl u e n ce Bit:	unaffected

User's Method	have
discriminate Regular: :	not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 0	1 1 1 0 0	Rz	

Figure 15.103: MULAXL.S32.RS

15.104 MULAXH.S32.S - 32 bits with saturation operation Signed High Halfword Unaligned multiply-accumulate instruction

Harmonization Directive	
grammatical	mulaxh.s32.s rz, rx, ry
manipulate	$Rz[31:0] = \text{Saturate}(Rz[31:0] + \{Rx[31:0] \times Ry[31:16]\}[47:16])$
Compilation results	Only 32-bit instructions exist. mulaxh.s32.s rz, rx, ry

User's Manual

Explana tion: :	Rx is multiplied by the high half-word of Ry and the high 32 bits of the result are added to Rz, and the result is saturated and stored in Rz. The process of saturation processing is that if the addition result is greater than the upper saturation value 0x7FFF FFFF, the result is the upper saturation value, if the addition result is smaller The result is the lower saturated value 0x8000 0000, in all other cases it is the result of the addition itself.
Mark ers of infl uen ce Bit:	unaffected
set a limit (on) Syste m:	not have
discri mina te Regu lar:	not have

32 inde x (mat h.) hono rific title	
drill (prac tice) Mak e:	Rz[31:0]= Saturate(Rz[31:0]+ {Rx[31:0] X Ry[31:16]}[47:16])
tell to Law:	mulaxh.s32.s rz, rx, ry
Explana tion	Rx is multiplied by the high half-word of Ry and the high 32 bits of the result are added to Rz, and the result is saturated and stored in Rz. The process of saturation processing is that if the addition result is greater than the upper

User's Manual:	saturation value 0x7FFF FFFF , the result is the upper saturation value, if the addition result is smaller The result is the lower saturated value 0x8000 0000, in all other cases it is the result of the addition itself.
Markers of influence Bit:	unaffected
set a limit (on) System:	not have
discriminate Regular:	not have

Command Format:

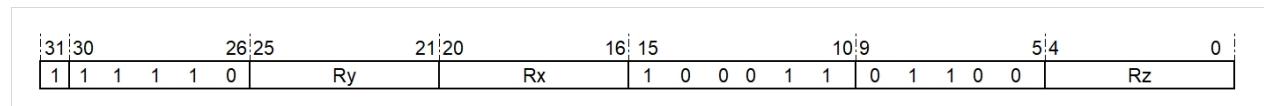


Figure 15.104: MULAXH.S32.

15.105 MULAXH.S32.RS - 32 bit signed high half with rounding and saturation operations

Word unaligned multiply-accumulate instruction

Harmonization means honorific title	
grammatical	mulaxh.s32.rs rz, rx, ry
manipulate	Rz[31:0] = Saturate(({Rz[31:0], 16'h0000} + Rx[31:0] X Ry[31:16] + 16'h8000)[47:16])
Compilation results	Only 32-bit instructions exist. mulaxh.s32.rs rz, rx, ry

Explanation:	Rx is multiplied by the high halfword of Ry and to {Rz, 16'h0000}, and the result is added to 0x8000. The high 32 bits of the result are saturated and stored in Rz. The result is then saturated and stored in Rz. The result is added to 0x8000 to realize the rounding operation. The process of saturation processing is that if the addition result is greater than the upper saturation value 0x7FFF FFFF , the result is the upper saturation value, if the addition result is less than the Lower saturated value 0x8000 0000, the result is the lower saturated value, otherwise it is the addition result itself.
Mark ers of in fl u e n ce Bit:	unaffected
set a limit (on) System:	not have
disc rimi nate	not have

User Manual
ular
:

32 in d e x (m at h.) hon orifi c title	
drill (pra ctice) Mak e:	Rz[31:0]= Saturate(({Rz[31:0], 16'h0000}+ Rx[31:0] X Ry[31:16]+ 16'h8000)[47:16])
tell to Law :	mulaxh.s32.rs rz, rx, ry
Exp lana tion : : : :	<p>Rx is multiplied by the high halfword of Ry and to {Rz, 16'h0000}, and the result is added to 0x8000. The high 32 bits of the result are saturated and stored in Rz. The result is then saturated and stored in Rx. The result is added to 0x8000 to realize the rounding operation.</p> <p>The process of saturation processing is that if the addition result is greater than the upper saturation value 0x7FFF FFFF , the result is the upper saturation value, if the addition result is less than the lower saturation value 0x8000 0000, the result is the lower saturated value, otherwise it is the addition result itself.</p>
M ar k er s of in fl u e n ce Bit:	unaffected

User's Method	have
limit (on)	
Syst em:	
disc rimi nate	not have
Reg ular	
:	

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1	1 1 1 0 0	Rz	

Figure 15.105: MULAXH.S32.RS

15.106 MULLL.S16 - 16-bit Signed Low Halfword Multiply Instruction

Harmoniza tion Directive	
grammatica l	mulll.s16 rz, rx, ry
manipulate	Rz[31:0]= Rx[15:0] X Ry[15:0]
Compilatio n results	Only 32-bit instructions exist. mulll.s16 rz, rx, ry

Description:	The lower half-word of Rx is multiplied by the lower half-word of Ry and the result is deposited into Rz.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	not have

32-bit instruction	
Operation:	$Rz[31:0] = Rx[15:0] \times Ry[15:0]$
Grammar:	mulll.s16 rz, rx, ry
Description:	The lower half-word of Rx is multiplied by the lower half-word of Ry and the result is deposited into Rz.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 0	0 0 0 0 1	Rz	

Figure 15.106: MULLL.S16

15.107 MULHH.S16 - 16 bit signed high half-word multiply instructions

Harmonization Directive	
grammatical	mulhh.s16 rz, rx, ry
manipulate	$Rz[31:0] = Rx[31:16] \times Ry[31:16]$
Compilation results	Only 32-bit instructions exist. mulhh.s16 rz, rx, ry

Description:	The high half-word of Rx is multiplied by the high half-word of Ry and the result is deposited into Rz.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	not have

32-bit instruction	
Operation:	$Rz[31:0] = Rx[31:16] \times Ry[31:16]$
Grammar:	mulhh.s16 rz, rx, ry
Description:	The high half-word of Rx is multiplied by the high half-word of Ry and the result is deposited into Rz.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

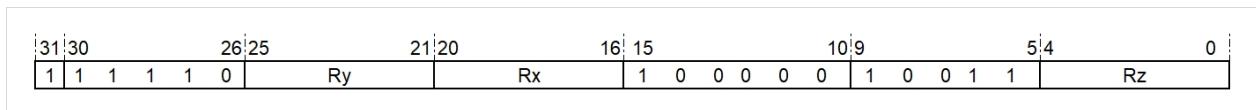


Figure 15.107: MULHH.S16

15.108 MULHL.S16 - 16 bit signed high and low half word multiply instructions

Harmonization Directive	
grammatical	mulhl.s16 rz, rx, ry
manipulate	$Rz[31:0] = Rx[31:16] \times Ry[15:0]$
Compilation results	Only 32-bit instructions exist. mulhl.s16 rz, rx, ry

Description:	The high half-word of Rx is multiplied by the low half-word of Ry and the result is deposited into Rz.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	not have

32-bit instruction	
Operation:	Rz[31:0]= Rx[31:16] X Ry[15:0]
Grammar:	mulhl.s16 rz, rx, ry
Description:	The high half-word of Rx is multiplied by the low half-word of Ry and the result is deposited into Rz.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 0	1 0 0 0 1	Rz	

Figure 15.108: MULHL.S16

15.109 RMULLL.S16 - 16-bit Signed Low Half-Word Decimal Multiplication Instructions

Unification directives	
grammatical	rmulll.s16 rz, rx, ry
manipulate	if(Rx[15:0]== 16'h8000 && Ry[15:0]== 16'h8000) Rz[31:0]= 32'h7FFF FFFF else Rz[31:0] = {Rx[15:0] X Ry[15:0]}<<1
Translation resolute	Only 32-bit instructions exist. rmulll.s16 rz, rx, ry

Description:	The lower half words of Rx and Ry are considered as signed decimals, where the 15th bit is the sign bit and the 14th to 0th bits are decimal places. Rx [15:0]=0x8000且 Ry[15:0]=0x8000, the result is 0x7FFF FF FF FF; in other cases, the result of multiplying Rx and Ry 0x7FFF FF FF. Shift left one bit and deposit into Rz.
Impact Signs Bit:	unaffected
Limitations:	not have
Exception:	not have

32-bit directives	
Operation:	if(Rx[15:0]== 16'h8000 && Ry[15:0]== 16'h8000) Rz[31:0]= 32'h7FFF FFFF else Rz[31:0] = {Rx[15:0] X Ry[15:0]}<< 1
Grammar:	rmulll.s16 rz, rx, ry
Description:	The lower half words of Rx and Ry are considered as signed decimals, where the 15th bit is the sign bit and the 14th to 0th bits are decimal places. Rx [15:0]=0x8000且 Ry[15:0]=0x8000, the result is 0x7FFF FF FF FF; in other cases, the result of multiplying Rx and Ry 0x7FFF FF FF. Shift left one bit and deposit into Rz.
Impact Signs Bit:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

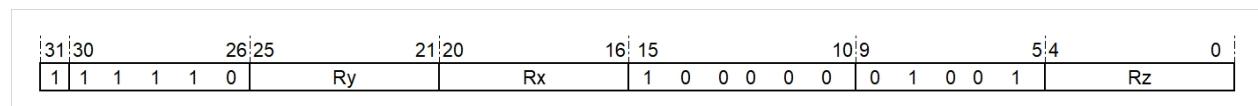


Figure 15.109: RMULLL.S16

15.110 RMULHH.S16 - 16 bit signed high half-word decimal multiply instruction

Unified instruction directives	
grammatical	rmulhh.s16 rz, rx, ry

User manual	if(Rx[31:16]== 16'h8000 && Ry[31:16]== 16'h8000)
ate	Rz[31:0]= 32'h7FFF FFFF else Rz[31:0] = {Rx[31:16] X Ry[31:16]}<< 1
Translat ion resolute	Only 32-bit instructions exist. rmulhh.s16 rz, rx, ry

Description:	The higher half words of Rx and Ry are considered as signed decimal, where the 31st bit is the sign bit and the 30th~16th bits are the decimal places. Rx[31:16]=0x8000且 Ry[31:16]=0x8000, the result is 0x7FFF FF FF FF; in the rest of the cases, the result of multiplying Rx and Ry is 0x7FFF FF FF. Shift left one bit and deposit into Rz.
Impact Sign s Bit:	unaffected
Limitat ions:	not have
Excepti on:	not have

32-bit direct ives	
Operati on:	if(Rx[31:16]== 16'h8000 && Ry[31:16]== 16'h8000) Rz[31:0]= 32'h7FFF FFFF else Rz[31:0] = {Rx[31:16] X Ry[31:16]}<< 1
Gram mar:	rmulhh.s16 rz, rx, ry
Description:	The higher half words of Rx and Ry are considered as signed decimal, where the 31st bit is the sign bit and the 30th~16th bits are the decimal places. Rx[31:16]=0x8000且 Ry[31:16]=0x8000, the result is 0x7FFF FF FF FF; in the rest of the cases, the result of multiplying Rx and Ry is 0x7FFF FF FF. Shift left one bit and deposit into Rz.
Impact Sign s Bit:	unaffected
Limitat ions:	not have
Excepti on:	not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 0	1 1 0 1 1	Rz	

15.111 RMULHL.S16 - 16-bit Signed High/Low Half-Word Decimal Multiplication Instructions

Unification directives	
grammatical	rmulhl.s16 rz, rx, ry
manipulate	if(Rx[31:16]== 16'h8000 && Ry[15:0]== 16'h8000) Rz[31:0]= 32'h7FFF FFFF else Rz[31:0] = {Rx[31:16] X Ry[15:0]}<<1
Translation resolute	Only 32-bit instructions exist. rmulhl.s16 rz, rx, ry

Explanation :	The high half-word of Rx and the low half-word of Ry are regarded as signed decimals, where the 31st bit of Rx is the sign bit and the 30th~16th bits are decimal bits, and the 15th bit of Ry is the sign bit and the 14th~0th bits are decimal bits. When Rx[31:16]=0x8000 且 Ry[15:0]=0x8000, the result is 0x7FFF FF FF; for the rest of the cases, the result of multiplying Rx and Ry is shifted one bit to the left and stored in Rz.
Mark ers of influence Bit:	unaffected
set a limit (on) System:	not have
discriminate Regu	not have

32 in d e x (m at h.) hono rific title	
Ope rate : :	if(Rx[31:16]== 16'h8000 && Ry[15:0]== 16'h8000) Rz[31:0]= 32'h7FFF FFFF else Rz[31:0] = {Rx[31:16] X Ry[15:0]}<< 1
tell to Law:	rmulhl.s16 rz, rx, ry
Exp lana tion : :	The high half-word of Rx and the low half-word of Ry are regarded as signed decimals, where the 31st bit of Rx is the sign bit and the 30th~16th bits are decimal bits, and the 15th bit of Ry is the sign bit and the 14th~0th bits are decimal bits. When Rx[31:16]=0x8000 且 Ry[15:0]=0x8000, the result is 0x7FFF FF FF; for the rest of the cases, the result of multiplying Rx and Ry is shifted one bit to the left and stored in Rz.
M ar k er s of in fl u e n ce Bit:	unaffected
set a limit (on) Syste m: :	not have
discr imin ate Regu	not have

Command Format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 0	1 1 0 0 1	Rz	

Figure 15.111: RMULHLS16

15.112 MULAHHS16.S - 16-Bit Signed High Half-Word Multiply Accumulate with Saturation Operation directives

Harmonization Directive	
grammatical	mulahhs16.s rz, rx, ry
manipulate	$Rz[31:0] = \text{Saturate}(Rz[31:0] + Rx[31:16] \times Ry[31:16])$
Compilation results	Only 32-bit instructions exist. mulahhs16.s rz, rx, ry

User's Manual

Explana tion: :	The high half-word of Rx is multiplied by the high half-word of Ry, the result of the multiplication is added to Rz, and the result of the addition is saturated and stored in Rz. The process of saturation processing is that if the addition result is greater than the upper saturation value 0x7FFF FFFF , the result is the upper saturation value, if the addition result is less than the The result is the lower saturated value 0x8000 0000, in all other cases it is the result of the addition itself.
Mark ers of infl uen ce Bit:	unaffected
set a limit (on) Syste m:	not have
discri mina te Regu lar:	not have

32 index (mat h.) hono rific title	
drill (prac tice) Mak e:	Rz[31:0]= Saturate(Rz[31:0]+ Rx[31:16] X Ry[31:16])
tell to Law:	mulahh.s16.s rz, rx, ry
Explana tion: :	The high half-word of Rx is multiplied by the high half-word of Ry, the result of the multiplication is added to Rz, and the result of the addition is saturated and stored in Rz. The process of saturation processing is that if the addition result is greater than the upper saturation value 0x7FFF FFFF , the result is the upper saturation value, if the addition result is

User's Manual	The result is the lower saturated value 0x8000 0000, in all other cases it is the result of the addition itself.
Mark ers of infl uen ce Bit:	unaffected
set a limit (on) Syste m:	not have
discri mina te Regu lar:	not have

Command Format:

31 30	26 25	21 20	16 15	10 9	5 4	0
[1 1 1 1 0]	Ry	Rx	[1 0 0 0 0]	[1 1 1 1 1]	Rz	

Figure 15.112: MULAHH.S16.

15.113 MULAHL.S16.S - 16-Bit Signed High-Low Half-Word Multiply Accumulate with Saturation Operation

add instruction

Harmonization Directive	
grammatical	mulahl.s16.s Rz, Rx, Ry
manipulate	Rz[31:0]= Saturate(Rz[31:0]+ Rx[31:16] X Ry[15:0])
Compilation results	Only 32-bit instructions exist. mulahl.s16.s Rz, Rx, Ry

Explanation:	The high half-word of Rx is multiplied by the low half-word of Ry, the result of the multiplication is added to Rz, and the result of the addition is saturated and stored in Rz. The process of saturation processing is that if the addition result is greater than the upper saturation value 0x7FFF FFFF , the result is the upper saturation value, if the addition result is smaller The result is the lower saturated value 0x8000 0000, in all other cases it is the result of the addition itself.
Markers of influence Bit:	unaffected
set a limit (on) System:	not have
discriminate Regular:	not have
order for m Style:	

32 index (mat h.) hono rific title	
drill (prac tice) Mak e:	Rz[31:0]= Saturate(Rz[31:0]+ Rx[31:16] X Ry[15:0])
tell to Law:	mulahl.s16.s Rz, Rx, Ry
Exp lana tion :	The high half-word of Rx is multiplied by the low half-word of Ry, the result of the multiplication is added to Rz, and the result of the addition is saturated and stored in Rz. The process of saturation processing is that if the addition result is greater than the upper saturation value 0x7FFF FFFF , the result is the upper saturation value, if the addition result is smaller The result is the lower saturated value 0x8000 0000, in all other cases it is the result of the addition itself.
Mar kers of infl uen ce Bit:	unaffected
set a limit (on) Syste m:	not have
discri mina te Regu lar:	not have

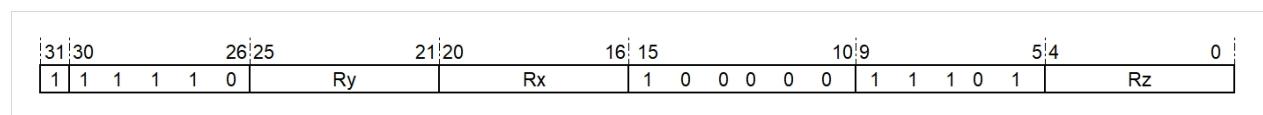
Command Format:

Figure 15.113: MULAHLS16.

15.114 MULALL.S16.E - 16-Bit Signed Low Half-Word Multiply Accumulate with Extended Operation directives

Harmonization Directive	
grammatical	mulall.s16.e rz, rx, ry
manipulate	{Rz+1[31:0],Rz[31:0]}= {Rz+1[31:0],Rz[31:0]} + Rx[15:0] X Ry[15:0]
Compilation results	Only 32-bit instructions exist. mulall.s16.e rz, rx, ry

Description:	The lower halfword of Rx is multiplied by the lower halfword of Ry, the result added to {Rz+1, Rz}, and the upper 32 bits of the result are stored in Rz+1. The lower 32 bits are stored in Rz.
Markers of influence Bit:	unaffected
Limitations:	not have
Exception:	not have

32-bit fingerpri nt honorific title	
Operation :	{Rz+1[31:0],Rz[31:0]}= {Rz+1[31:0],Rz[31:0]}+ Rx[15:0] X Ry[15:0]
Grammar:	mulall.s16.e rz, rx, ry
Description:	The lower halfword of Rx is multiplied by the lower halfword of Ry, the result added to {Rz+1, Rz}, and the upper 32 bits of the result are stored in Rz+1. The lower 32 bits are stored in Rz.
Markers of influence Bit:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

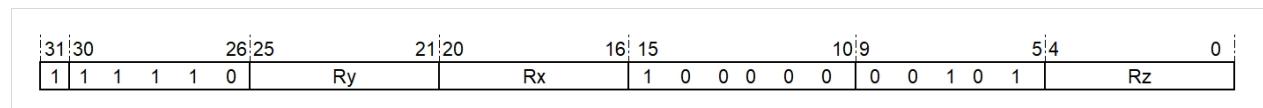


Figure 15.114: MULALL.S16.

15.115 MULAHH.S16.E - 16-Bit Signed High Half-Word Multiply

User's Manual **Accumulate with Extended Operation****directives**

Harmonization Directive	
grammatical	mulahh.s16.e rz, rx, ry
manipulate	{Rz+1[31:0],Rz[31:0]}= {Rz+1[31:0],Rz[31:0]}+ Rx[31:16] X Ry[31:16]
Compilation results	Only 32-bit instructions exist. mulahh.s16.e rz, rx, ry

Description:	The high half-word of Rx is multiplied by the high half-word of Ry, the result of the multiplication added to {Rz+1, Rz}, and the high 32 bits of the result are stored in Rz+1. The lower 32 bits are stored in Rz.
Markers of influence Bit:	unaffected
Limitations:	not have
Exception:	not have

32-bit fingerpri nt honorific title	
Operation :	{Rz+1[31:0],Rz[31:0]}= {Rz+1[31:0],Rz[31:0]}+ Rx[31:16] X Ry[31:16]
Grammar:	mulahh.s16.e rz, rx, ry
Description:	The high half-word of Rx is multiplied by the high half-word of Ry, the result of the multiplication added to {Rz+1, Rz}, and the high 32 bits of the result are stored in Rz+1. The lower 32 bits are stored in Rz.
Markers of influence Bit:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

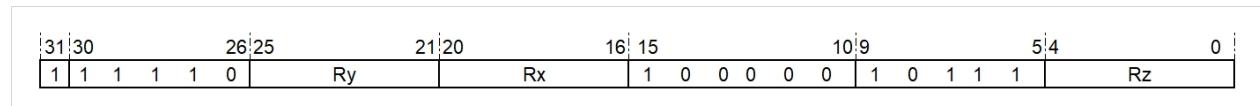


Figure 15.115: MULAHH.S16.

15.116 MULAHL.S16.E - 16-Bit Signed High-Low Half-Word Multiply

User's Manual **Accumulate with Extended Operations****add instruction**

Harmonization Directive	
grammatical	mulahl.s16.e rz, rx, ry
manipulate	{Rz+1[31:0],Rz[31:0]}= {Rz+1[31:0],Rz[31:0]}+ Rx[31:16] X Ry[15:0]
Compilation results	Only 32-bit instructions exist. mulahl.s16.e rz, rx, ry

Description:	The high halfword of Rx is multiplied by the low halfword of Ry, the result added to {Rz+1, Rz}, and the high 32 bits of the result are stored in Rz+1. The lower 32 bits are stored in Rz.
Markers of influence Bit:	unaffected
Limitations:	not have
Exception:	not have

32-bit fingerpri nt honorific title	
Operation :	{Rz+1[31:0],Rz[31:0]}= {Rz+1[31:0],Rz[31:0]}+ Rx[31:16] X Ry[15:0]
Grammar:	mulahl.s16.e rz, rx, ry
Description:	The high halfword of Rx is multiplied by the low halfword of Ry, the result added to {Rz+1, Rz}, and the high 32 bits of the result are stored in Rz+1. The lower 32 bits are stored in Rz.
Markers of influence Bit:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

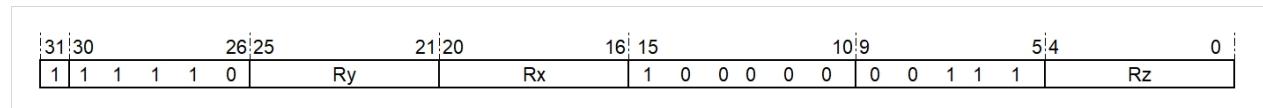


Figure 15.116: MULAHLS16.

15.117 PMUL.(U/S)16 - 16-bit parallel (no/have) signed multiply

Harmonization Directive	
grammatical	pmul.u16 rz, rx, ry pmul.s16 rz, rx, ry
manipulate	Rz+1[31:0]= Rx[31:16] X Ry[31:16] Rz[31:0]= Rx[15:0] X Ry[15:0]
Compilation results	Only 32-bit instructions exist. pmul.u16 rz, rx, ry pmul.s16 rz, rx, ry

Description:	The high half-word of Rx is multiplied by the high half-word of Ry and the result is deposited into Rz+1; The lower half word of Rx is multiplied by the lower half word of Ry and the result is stored in Rz.
Impact Marker Bit:	unaffected
Limitations :	not have
Exception:	not have

32-bit instruction	
Operation:	$Rz+1[31:0] = Rx[31:16] \times Ry[31:16]$ $Rz[31:0] = Rx[15:0] \times Ry[15:0]$
Grammar:	pmul.u16 rz, rx, ry pmul.s16 rz, rx, ry
Description:	The high half-word of Rx is multiplied by the high half-word of Ry and the result is deposited into Rz+1; The lower half word of Rx is multiplied by the lower half word of Ry and the result is stored in Rz.
Impact Marker Bit:	unaffected
Limitations :	not have
Exception:	not have

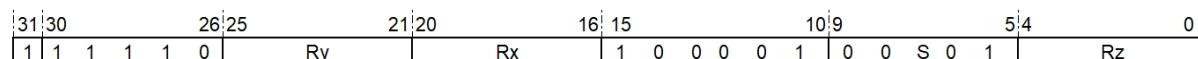
Command Format:

Figure 15.117: PMUL.(US)16

Note: S is 1 for unsign and 0 for sign.
--

15.118 PMULX.(U/S)16 - 16-bit parallel (no/have) signed cross multiply instruction

Harmonization Directive	
grammatical	pmulx.u16 rz, rx, ry pmulx.s16 rz, rx, ry
manipulate	Rz+1[31:0]= Rx[31:16] X Ry[15:0] Rz[31:0]= Rx[15:0] X Ry[31:16]
Compilation results	Only 32-bit instructions exist. pmulx.u16 rz, rx, ry pmulx.s16 rz, rx, ry

Description:	The high half-word of Rx is multiplied by the low half-word of Ry, and the result is stored in Rz+1; The low half-word of Rx is multiplied by the high half-word of Ry and the result is stored in Rz;
Impact Marker Bit:	unaffected
Limitations :	not have
Exception:	not have

32-bit instruction	
Operation:	Rz+1[31:0]= Rx[31:16] X Ry[15:0] Rz[31:0]= Rx[15:0] X Ry[31:16]
Grammar:	pmulx.u16 rz, rx, ry pmulx.s16 rz, rx, ry
Description:	The high half-word of Rx is multiplied by the low half-word of Ry, and the result is stored in Rz+1; The low half-word of Rx is multiplied by the high half-word of Ry and the result is stored in Rz;
Impact Marker Bit:	unaffected
Limitations :	not have

Command Format:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1	1	1	0	Ry	Rx	1	0	0	0	1	0

Figure 15.118: PMULX.(US)16

Note: S is 1 for unsign and 0 for sign.

15.119 PRMUL.S16 - 16 bit parallel signed decimal multiplication instructions

c o l e c t i v e l y k n o w n a s "h a r m o n i z a t i o n" "n" o r "h a r m o n i z a t i o n in d e x" (m at h.	
--	--

User's Manual	
hono rific title	
tell to Budd hist teac hing	prmul.s16 rz, rx, ry
m a ni p ul at e c o m pi la ti o n re s ul t resol ute	<pre> if(Rx[31:16]== 16'h8000 && Ry[31:16]== 16'h8000) Rz+1[31:0]= 32'h7FFF FFFF else Rz+1[31:0] = {Rx[31:16] X Ry[31:16]} << 1 if(Rx[15:0] == 16'h8000 && Ry[15:0] == 16'h8000) Rz[31:0]= 32'h7FFF FFFF else Rz[31:0]= {Rx[15:0] X Ry[15:0]}<< 1 </pre> <p>Only 32-bit instructions exist. prmul.s16 rz, rx, ry</p>

Exp lana tion : :	<p>The upper half words of Rx and Ry are considered as signed decimals, where the 31st bit is the sign bit and the 30th to 16th bits are decimal places; Rx and Ry are considered as signed decimals; Rx and Ry are considered as signed decimals.</p> <p>The lower half of the word is regarded as a signed decimal, where the 15th bit is the sign bit and the 14th to the 0th bits are decimal places.</p> <p>Rx[31:16]=0x8000且 Ry[31:16]=0x8000, the result is 0x7FFF FFFF; in the rest of the cases, the high halfword of Rx is identical to the</p> <p>Ry is multiplied by the higher half-word, and the result of the multiplication is shifted one place to the left and stored in Rz+1.</p> <p>Rx [15:0]=0x8000且 Ry[15:0]=0x8000, the result is 0x7FFF FFFF; in the rest of the cases, the lower halfword of Rx is the same as the Ry</p> <p>The lower half-word is multiplied, and the result of the multiplication is shifted one place to the left and deposited into Rz.</p>
M ar k er s of in fl u e n ce Bit:	unaffected
set a limi t (on) Syst em:	not have
disc rimi nate Reg ular :	not have

32 in d e x (m at h.) hon orifi c title	
Ope rate : :	<pre>if(Rx[31:16]== 16'h8000 && Ry[31:16]== 16'h8000) Rz+1[31:0]= 32'h7FFF FFFF else Rz+1[31:0] = {Rx[31:16] X Ry[31:16]} << 1 if(Rx[15:0] == 16'h8000 && Ry[15:0] == 16'h8000) Rz[31:0]= 32'h7FFF FFFF else Rz[31:0]= {Rx[15:0] X Ry[15:0]}<< 1</pre>
tell to Law : :	prmuls16 rz, rx, ry
Exp lana tion : :	<p>The upper half words of Rx and Ry are considered as signed decimals, where the 31st bit is the sign bit and the 30th to 16th bits are decimal places; Rx and Ry are considered as signed decimals; Rx and Ry are considered as signed decimals.</p> <p>The lower half of the word is regarded as a signed decimal, where the 15th bit is the sign bit and the 14th to the 0th bits are decimal places.</p> <p>Rx[31:16]=0x8000 且 Ry[31:16]=0x8000, the result is 0x7FFF FFFF; in the rest of the cases, the high halfword of Rx is identical to the Ry is multiplied by the higher half-word, and the result of the multiplication is shifted one place to the left and stored in Rz+1.</p> <p>Rx [15:0]=0x8000 且 Ry[15:0]=0x8000, the result is 0x7FFF FFFF; in the rest of the cases, the lower halfword of Rx is the same as the Ry</p> <p>The lower half-word is multiplied, and the result of the multiplication is shifted one place to the left and deposited into Rz.</p>
M ar k er s	unaffected

User's Manual	
Bit: set a limit (on) System:	not have
discriminate Regular :	not have

Command Format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 1 1 1 0	Ry	Rx	1 0 0 0 0 1	0 1 0 0 1	Rz	

Figure 15.119: PRMUL.

15.120 PRMULX.S16 - 16 bit parallel signed cross-decimal multiply instructions

c ol le ct iv el y k n o w n as "h ar m o ni za ti o n" or "h ar m o ni za ti o n in d e x" (m at h.) hono rific title	
---	--

tell User's Manual to Budd hist teac hing	prmulx.s16 rz, rx, ry
m a ni p ul at e	<pre> if(Rx[31:16]== 16'h8000 && Ry[15:0]== 16'h8000) Rz+1[31:0]= 32'h7FFF FFFF else Rz+1[31:0]= {Rx[31:16] X Ry[15:0]}<<1 if(Rx[15:0]== 16'h8000 && Ry[31:16]== 16'h8000) Rz[31:0]= 32'h7FFF FFFF else Rz[31:0]= {Rx[15:0] X Ry[31:16]}<<1 </pre>
c o m pi la ti o n re s ul t resol ute	<p>Only 32-bit instructions exist.</p> <p>prmulx.s16 rz, rx, ry</p>

Exp lana tion :	<p>The upper half words of Rx and Ry are considered as signed decimals, where the 31st bit is the sign bit and the 30th to 16th bits are decimal places; Rx and Ry are considered as signed decimals; Rx and Ry are considered as signed decimals.</p> <p>The lower half of the word is regarded as a signed decimal, where the 15th bit is the sign bit and the 14th to the 0th bits are decimal places.</p> <p>Rx[31:16]=0x8000 且 Ry[15:0]=0x8000, the result is 0x7FFF FFFF; in the rest of the cases, the high half-word of Rx is the same as the Ry</p> <p>The lower half-word is multiplied, and the result of the multiplication is shifted one place to the left and deposited into Rz+1.</p> <p>Rx[15:0]=0x8000 且 Ry[31:16]=0x8000, the result is 0x7FFF FFFF; in the rest of the cases, the lower halfword of Rx is the same as the Ry</p> <p>The high half-word is multiplied, and the result of the multiplication is shifted one place to the left and deposited into Rz.</p>
--------------------------	--

User's Manual	Manufactured
ar k er s of in fl u e n ce Bit:	
set a limi t (on) Syst em:	not have
disc rimi nate Reg ular :	not have

32 in d e x (m at h.) hon orifi c title	
Ope rate : :	<pre> if(Rx[31:16]== 16'h8000 && Ry[15:0]== 16'h8000) Rz+1[31:0]= 32'h7FFF FFFF else Rz+1[31:0]= {Rx[31:16] X Ry[15:0]}<< 1 if(Rx[15:0]== 16'h8000 && Ry[31:16]== 16'h8000) Rz[31:0]= 32'h7FFF FFFF else Rz[31:0]= {Rx[15:0] X Ry[31:16]}<< 1 </pre>
tell to Law : :	prmulx.s16 rz, rx, ry
Exp lana tion : :	<p>The upper half words of Rx and Ry are considered as signed decimals, where the 31st bit is the sign bit and the 30th to 16th bits are decimal places; Rx and Ry are considered as signed decimals; Rx and Ry are considered as signed decimals.</p> <p>The lower half of the word is regarded as a signed decimal, where the 15th bit is the sign bit and the 14th to the 0th bits are decimal places.</p> <p>Rx[31:16]=0x8000 且 Ry[15:0]=0x8000, the result is 0x7FFF FFFF; in the rest of the cases, the high half-word of Rx is the same as the Ry</p> <p>The lower half-word is multiplied, and the result of the multiplication is shifted one place to the left and deposited into Rz+1.</p> <p>Rx[15:0]=0x8000 且 Ry[31:16]=0x8000, the result is 0x7FFF FFFF; in the rest of the cases, the lower halfword of Rx is the same as the Ry</p> <p>The high half-word is multiplied, and the result of the multiplication is shifted one place to the left and deposited into Rz.</p>
M ar k er s	unaffected

User's Manual	
Bit: set a limit (on) System:	not have
discriminate Regular :	not have

Command Format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 1 1 1 0	Ry	Rx	1 0 0 0 0 1	0 1 0 1 1	Rz	

Figure 15.120: PRMULX.

15.121 PRMUL.S16.H - 16-bit Parallel Signed Decimal Multiplication**Fetch High 16-bit Instruction**

c ol le ct iv el y k n o w n as "h ar m o ni za ti o n" or "h ar m o ni za ti o n in d e x" (m at h.) hono rific title	
---	--

tell to User's Manual	prmul.s16.h rz, rx, ry
Budd hist teach ing	<pre> if(Rx[31:16]== 16'h8000 && Ry[31:16]== 16'h8000) Rz[31:16]= 16'h7FFF else Rz[31:16]= {Rx[31:16] X Ry[31:16}][30:15] if(Rx[15:0]== 16'h8000 && Ry[15:0]== 16'h8000) Rz[15:0]= 16'h7FFF else Rz[15:0]= {Rx[15:0] X Ry[15:0}][30:15]</pre>
c o m pi la ti o n re s ul t resol ute	<p>Only 32-bit instructions exist.</p> <p>prmul.s16.h rz, rx, ry</p>

Exp lanation :	<p>The upper half words of Rx and Ry are considered as signed decimals, where the 31st bit is the sign bit and the 30th to 16th bits are decimal places; Rx and Ry are considered as signed decimals; Rx and Ry are considered as signed decimals.</p> <p>The lower half of the word is regarded as a signed decimal, where the 15th bit is the sign bit and the 14th to the 0th bits are decimal places.</p> <p>Rx[31:16]=0x8000且 Ry[31:16]=0x80 00, the result is 0x7FFF; in the rest of the cases, the high half word of Rx is multiplied with the high half word of Ry, and the 30th~15th bits of the multiplication result are taken and stored into the high half word of Rz.</p> <p>Rx[15:0]=0x8000且 Ry[15:0]=0x80 00, the result is 0x7FFF; in the rest of the cases, the lower half word of Rx and the lower halfword of Ry are the same.</p> <p>The multiplication is done by taking the 30th~15th bits of the multiplication result and depositing them into the lower half word of Rz.</p>
M ar k er s of	unaffected

in User's fl u e n ce Bit:	Manual
set a limi t (on) Syst em:	not have
disc rimi nate Reg ular :	not have

32 in d e x (m at h.) hon orifi c title	
Ope rate : :	<pre> if(Rx[31:16]== 16'h8000 && Ry[31:16]== 16'h8000) Rz[31:16]= 16'h7FFF else Rz[31:16]= {Rx[31:16] X Ry[31:16}][30:15] if(Rx[15:0]== 16'h8000 && Ry[15:0]== 16'h8000) Rz[15:0]= 16'h7FFF else Rz[15:0]= {Rx[15:0] X Ry[15:0}][30:15]</pre>
tell to Law : :	prmul.s16.h rz, rx, ry
Exp lana tion : :	<p>The upper half words of Rx and Ry are considered as signed decimals, where the 31st bit is the sign bit and the 30th to 16th bits are decimal places; Rx and Ry are considered as signed decimals; Rx and Ry are considered as signed decimals.</p> <p>The lower half of the word is regarded as a signed decimal, where the 15th bit is the sign bit and the 14th to the 0th bits are decimal places.</p> <p>Rx[31:16]=0x8000 且 Ry[31:16]=0x80 00, the result is 0x7FFF; in the rest of the cases, the high half word of Rx is multiplied with the high half word of Ry, and the 30th~15th bits of the multiplication result are taken and stored into the high half word of Rz.</p> <p>Rx[15:0]=0x8000 且 Ry[15:0]=0x80 00, the result is 0x7FFF; in the rest of the cases, the lower half word of Rx and the lower halfword of Ry are the same.</p> <p>The multiplication is done by taking the 30th~15th bits of the multiplication result and depositing them into the lower half word of Rz.</p>
M ar k er s of in	unaffected

User's Manual	
Bit: set a limi t (on) Syst em:	not have
disc rimi nate Reg ular :	not have

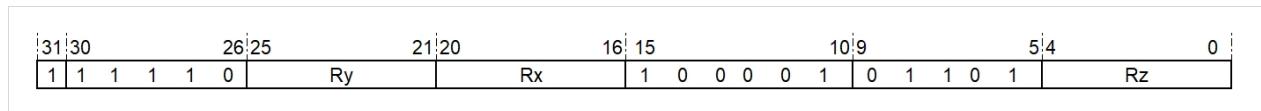
Command Format:

Figure 15.121: PRMUL.S16.

15.122 PRMUL.S16.RH - 16 bit parallel signed decimal multiplication with rounding operations

Fetch High 16-bit Instruction

c ol le ct iv el y k n o w n as "h ar m o ni za ti o n" or "h ar m o ni za ti o n in d e x" (m at h.) hono	
---	--

User Manual title	
tell to Bud dhist teaching	prmul.s16.rh rz, rx, ry
m a ni p ul at e	<pre> if(Rx[31:16]== 16'h8000 && Ry[31:16]== 16'h8000) Rz[31:16]= 16'h7FFF else Rz[31:16] = {Rx[31:16] X Ry[31:16] + 16'h4000}[30:15] if(Rx[15:0] == 16'h8000 && Ry[15:0] == 16' h8000) Rz[15:0]= 16'h7FFF else Rz[15:0]= {Rx[15:0] X Ry[15:0]+ 16'h4000}[30:15]</pre>
c o m pi la ti o n re s ul t resolute	<p>Only 32-bit instructions exist.</p> <p>prmul.s16.rh rz, rx, ry</p>

Exp lana tion : :	<p>The upper half words of Rx and Ry are considered as signed decimals, where the 31st bit is the sign bit and the 30th to 16th bits are decimal places; Rx and Ry are considered as signed decimals; Rx and Ry are considered as signed decimals.</p> <p>The lower half of the word is regarded as a signed decimal, where the 15th bit is the sign bit and the 14th to the 0th bits are decimal places.</p> <p>Rx[31:16]=0x8000且 Ry[31:16]=0x8000, the result is 0x7FFF; in the rest of the cases, the high half word of Rx is multiplied by the high half word of Ry and the multiplication result is added with 0x4000, and the 30th~15th bit of the result is taken and deposited into the high half word of Rz. The multiplication result plus 0x4000 is used to realize the rounding operation.</p> <p>Rx[15:0]=0x8000且 Ry[15:0]=0x8000, the result is 0x7FFF; in the rest of the cases, the lower half word of Rx is multiplied with the lower half word of Ry, the multiplication result is added with 0x4000, and the 30th~15th bit of the addition result is taken and deposited into the lower half word of Rz. Add 0x4000 to the multiplication result.</p> <p>Used to implement rounding operations.</p>
M ar k er s of in fl u e n ce Bit:	unaffected
set a limi t (on) Syst em:	not have
disc rimi nate Reg ular :	not have

32 in d e x (m at h.) hon orifi c title	
Operate rate : tell to Law : Explana tion : Mark ers	<pre> if(Rx[31:16]== 16'h8000 && Ry[31:16]== 16'h8000) Rz[31:16]= 16'h7FFF else Rz[31:16] = {Rx[31:16] X Ry[31:16] + 16'h4000}[30:15] if(Rx[15:0] == 16'h8000 && Ry[15:0] == 16' h8000) Rz[15:0]= 16'h7FFF else Rz[15:0]= {Rx[15:0] X Ry[15:0]+ 16'h4000}[30:15]</pre> <p>prmuls16.rh rz, rx, ry</p> <p>The upper half words of Rx and Ry are considered as signed decimals, where the 31st bit is the sign bit and the 30th to 16th bits are decimal places; Rx and Ry are considered as signed decimals; Rx and Ry are considered as signed decimals.</p> <p>The lower half of the word is regarded as a signed decimal, where the 15th bit is the sign bit and the 14th to the 0th bits are decimal places.</p> <p>Rx[31:16]=0x8000 且 Ry[31:16]=0x8000, the result is 0x7FFF; in the rest of the cases, the high half word of Rx is multiplied by the high half word of Ry and the multiplication result is added with 0x4000, and the 30th~15th bit of the result is taken and deposited into the high half word of Rz. The multiplication result plus 0x4000 is used to realize the rounding operation.</p> <p>Rx[15:0]=0x8000 且 Ry[15:0]=0x8000, the result is 0x7FFF; in the rest of the cases, the lower half word of Rx is multiplied with the lower half word of Ry, the multiplication result is added with 0x4000, and the 30th~15th bit of the addition result is taken and deposited into the lower half word of Rz. Add 0x4000 to the multiplication result.</p> <p>Used to implement rounding operations.</p> <p>unaffected</p>

User's Manual	
Bit: set a limit (on) System:	not have
discriminate Regular :	not have

Command Format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 1 1 1 0	Ry	Rx	1 0 0 0 0 1	1 1 1 0 1	Rz	

Figure 15.122: PRMUL.S16.RH

15.123 PRMULX.S16.H - 16 Bit Parallel Signed Cross Decimal Multiplication Taking High 16 bit command (computing)

c ol le ct iv el y k n o w n as "h ar m o ni za ti o n" or "h ar m o ni za ti o n in d e x" (m at h.) hono	
---	--

title	Users Manual
tell to Budd hist teaching	prmulp.s16.h rz, rx, ry
m a ni pul at e	<pre> if(Rx[31:16]== 16'h8000 && Ry[15:0]== 16'h8000) Rz[31:16]= 16'h7FFF else Rz[31:16]= {Rx[31:16] X Ry[15:0]}[30:15] if(Rx[15:0]== 16'h8000 && Ry[31:16]== 16'h8000) Rz[15:0]= 16'h7FFF else Rz[15:0]= {Rx[15:0] X Ry[31:16]}[30:15]</pre>
c o m pi la ti o n re s ul t resol ute	<p>Only 32-bit instructions exist.</p> <pre>prmulp.s16.h rz, rx, ry</pre>

Explanation :	<p>The upper half words of Rx and Ry are considered as signed decimals, where the 31st bit is the sign bit and the 30th to 16th bits are decimal places; Rx and Ry are considered as signed decimals; Rx and Ry are considered as signed decimals.</p> <p>The lower half of the word is regarded as a signed decimal, where the 15th bit is the sign bit and the 14th to the 0th bits are decimal places.</p> <p>Rx[31:16]=0x8000 且 Ry[31:16]=0x80 00, the result is 0x7FFF; in the rest of the cases, the high half word of Rx is multiplied with the high half word of Ry, and the 30th~15th bits of the multiplication result are taken and stored into the high half word of Rz.</p> <p>Rx[15:0]=0x8000 且 Ry[15:0]=0x80 00, the result is 0x7FFF; in the rest of the cases, the lower half word of Rx and the lower halfword of Ry are the same.</p> <p>The multiplication is done by taking the 30th~15th bits of the multiplication result and depositing them into the lower half word of Rz.</p>
---------------	--

User's Manual	Manufactured
ar k er s of in fl u e n ce Bit:	
set a limi t (on) Syst em:	not have
disc rimi nate Reg ular :	not have

32 in d e x (m at h.) hon orifi c title	
Ope rate : :	<pre> if(Rx[31:16]== 16'h8000 && Ry[15:0]== 16'h8000) Rz[31:16]= 16'h7FFF else Rz[31:16]= {Rx[31:16] X Ry[15:0]}[30:15] if(Rx[15:0]== 16'h8000 && Ry[31:16]== 16'h8000) Rz[15:0]= 16'h7FFF else Rz[15:0]= {Rx[15:0] X Ry[31:16]}[30:15]</pre>
tell to Law : :	prmulx.s16.h rz, rx, ry
Exp lana tion : :	<p>The upper half words of Rx and Ry are considered as signed decimals, where the 31st bit is the sign bit and the 30th to 16th bits are decimal places; Rx and Ry are considered as signed decimals; Rx and Ry are considered as signed decimals.</p> <p>The lower half of the word is regarded as a signed decimal, where the 15th bit is the sign bit and the 14th to the 0th bits are decimal places.</p> <p>Rx[31:16]=0x8000 且 Ry[31:16]=0x80 00, the result is 0x7FFF; in the rest of the cases, the high half word of Rx is multiplied with the high half word of Ry, and the 30th~15th bits of the multiplication result are taken and stored into the high half word of Rz.</p> <p>Rx[15:0]=0x8000 且 Ry[15:0]=0x80 00, the result is 0x7FFF; in the rest of the cases, the lower half word of Rx and the lower halfword of Ry are the same.</p> <p>The multiplication is done by taking the 30th~15th bits of the multiplication result and depositing them into the lower half word of Rz.</p>
M ar k er s of in	unaffected

User's Manual	
Bit: set a limi t (on) Syst em:	not have
disc rimi nate Reg ular :	not have

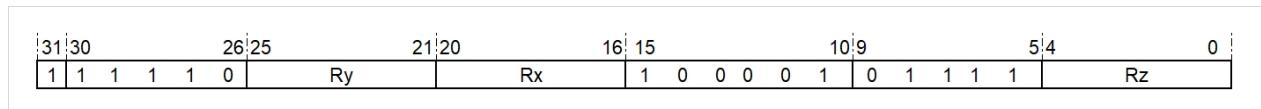
Command Format:

Figure 15.123: PRMULX.S16.

15.124 PRMULX.S16.RH - 16 bit parallel signed cross small with rounding operation

Number Multiply Higher 16-Bit Instruction

c ol le ct iv el y k n o w n as "h ar m o ni za ti o n" or "h ar m o ni za ti o n in d e x" (m at h.) hono	
---	--

title	Users Manual
tell to Bud dhist teac hing	prmulx.s16.rh rz, rx, ry
m a ni p ul at e	<pre>if(Rx[31:16]== 16'h8000 && Ry[15:0]== 16'h8000) Rz[31:16]= 16'h7FFF else Rz[31:16]= {Rx[31:16] X Ry[15:0]+ 16'h4000}[30:15] if(Rx[15:0]== 16'h8000 && Ry[31:16]== 16'h8000) Rz[15:0]= 16'h7FFF else Rz[15:0]= {Rx[15:0] X Ry[31:16]+ 16'h4000}[30:15]</pre>
c o m pi la ti o n re s ul t resol ute	<p>Only 32-bit instructions exist.</p> <pre>prmulx.s16.rh rz, rx, ry</pre>

Exp lana tion : :	<p>The upper half words of Rx and Ry are considered as signed decimals, where the 31st bit is the sign bit and the 30th to 16th bits are decimal places; Rx and Ry are considered as signed decimals; Rx and Ry are considered as signed decimals.</p> <p>The lower half of the word is regarded as a signed decimal, where the 15th bit is the sign bit and the 14th to the 0th bits are decimal places.</p> <p>Rx[31:16]=0x8000且 Ry[15:0]=0x8000, the result is 0x7FFF; in the rest of the cases, the high half word of Rx is multiplied by the low half word of Ry and the multiplication result is added with 0x4000, and the 30th~15th bit of the result is taken and deposited into the high half word of Rz. The multiplication result plus 0x4000 is used to realize the rounding operation.</p> <p>Rx[15:0]=0x8000且 Ry[31:16]=0x8000, the result is 0x7FFF; in the rest of the cases, the low half word of Rx is multiplied with the high half word of Ry, the multiplication result is added with 0x4000, and the 30th~15th bit of the addition result is taken and deposited into the low half word of Rz. Add 0x4000 to the multiplication result.</p> <p>Used to implement rounding operations.</p>
M ar k er s of in fl u e n ce Bit:	unaffected
set a limi t (on) Syst em:	not have
disc rimi nate Reg ular :	not have

32 in d e x (m at h.) hon orifi c title	
Ope rate : :	<pre>if(Rx[31:16]== 16'h8000 && Ry[15:0]== 16'h8000) Rz[31:16]= 16'h7FFF else Rz[31:16]= {Rx[31:16] X Ry[15:0]+ 16'h4000}[30:15] if(Rx[15:0]== 16'h8000 && Ry[31:16]== 16'h8000) Rz[15:0]= 16'h7FFF else Rz[15:0]= {Rx[15:0] X Ry[31:16]+ 16'h4000}[30:15]</pre>
tell to Law : :	prmulp.s16.rh rz, rx, ry
Exp lana tion : :	<p>The upper half words of Rx and Ry are considered as signed decimals, where the 31st bit is the sign bit and the 30th to 16th bits are decimal places; Rx and Ry are considered as signed decimals; Rx and Ry are considered as signed decimals.</p> <p>The lower half of the word is regarded as a signed decimal, where the 15th bit is the sign bit and the 14th to the 0th bits are decimal places.</p> <p>Rx[31:16]=0x8000 且 Ry[15:0]=0x8000, the result is 0x7FFF; in the rest of the cases, the high half word of Rx is multiplied by the low half word of Ry and the multiplication result is added with 0x4000, and the 30th~15th bit of the result is taken and deposited into the high half word of Rz. The multiplication result plus 0x4000 is used to realize the rounding operation.</p> <p>Rx[15:0]=0x8000 且 Ry[31:16]=0x8000, the result is 0x7FFF; in the rest of the cases, the low half word of Rx is multiplied with the high half word of Ry, the multiplication result is added with 0x4000, and the 30th~15th bit of the addition result is taken and deposited into the low half word of Rz. Add 0x4000 to the multiplication result.</p> <p>Used to implement rounding operations.</p>
M ar k er s	unaffected

User's Manual	
Bit: set a limit (on) System:	not have
discriminate Regular :	not have

Command Format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 1 1 1 0	Ry	Rx	1 0 0 0 0 1	1 1 1 1 1 1	Rz	

Figure 15.124: PRMULX.S16.RH

15.125 MULCA.S16.S - 16-bit signed multiply-chain-add instruction with saturation operation

standard directives	
grammatical	mulca.s16.s rz, rx, ry
manipulate	if(Rx[31:0]= 32'h80008000 && Ry[31:0]= 32'h80008000) Rz[31:0]= 32'h7FFF FFFF else Rz[31:0]= Rx[31:16] X Ry[31:16]+ Rx[15:0] X Ry[15:0]
compilation result resolute	Only 32-bit instructions exist. mulca.s16.s rz, rx, ry

Description:	The lower half word of Rx is multiplied by the lower half word of Ry, the higher half word of Rx is multiplied by the higher half word of Ry, and the two multiplication results are added together. The result of the two multiplications is added together when Rx=0x80008000且 Ry= 0x80008000 The multiply-chain-add result overflows and is saturated to 0x7FFF FFFF.
impact Logo Bit:	unaffected
Limitations:	not have
Exception:	not have

32-bit directives	
Operation:	if(Rx[31:0]= 32'h80008000 && Ry[31:0]= 32'h80008000) Rz[31:0]= 32'h7FFF FFFF else Rz[31:0]= Rx[31:16] X Ry[31:16]+ Rx[15:0] X Ry[15:0]
Gramm:	mulca.s16.s rz, rx, ry

Description:	The lower half word of Rx is multiplied by the lower half word of Ry, the higher half word of Rx is multiplied by the higher half word of Ry, and the two multiplication results are added together. The result of the two multiplications is added together when Rx=0x80008000且 Ry=0x80008000 The multiply-chain-add result overflows and is saturated to 0x7FFF FFFF.
impact Logo Bit:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 0 1	1 1 1 1 1	Rz	

Figure 15.125: MULCA.S16.

15.126 MULCAX.S16.S - 16 bit signed cross-multiplication chain with saturation operation plus finger honorific title

standard directives	
grammatical	mulcax.s16.s rz, rx, ry
manipulate	if(Rx[31:0]= 32'h80008000 && Ry[31:0]= 32'h80008000) Rz[31:0]= 32'h7FFF FFFF else Rz[31:0]= Rx[31:16] X Ry[15:0]+ Rx[15:0] X Ry[31:16]
compilation result resolute	Only 32-bit instructions exist. mulcax.s16.s rz, rx, ry

Description:	The lower half-word of Rx is multiplied by the higher half-word of Ry, the higher half-word of Rx is multiplied by the lower half-word of Ry, and the two multiplication results are added together. The result of the two multiplications is added together when Rx=0x80008000且 Ry= 0x80008000 The multiply-chain-add result overflows and is saturated to 0x7FFF FFFF.
impact Logo Bit:	unaffected
Limitations:	not have
Exception:	not have
Directives Format:	

32-bit directives	
Operation:	if(Rx[31:0]==32'h80008000 && Ry[31:0]==32'h80008000) Rz[31:0]=32'h7FFF FFFF else Rz[31:0]=Rx[31:16] X Ry[15:0]+ Rx[15:0] X Ry[31:16]
Grammar:	mulcax.s16.s rz, rx, ry
Description:	The lower half-word of Rx is multiplied by the higher half-word of Ry, the higher half-word of Rx is multiplied by the lower half-word of Ry, and the two multiplication results are added together. The result of the two multiplications is added together when Rx=0x80008000且 Ry=0x80008000 The multiply-chain-add result overflows and is saturated to 0x7FFF FFFF.
Impact Logo Bit:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

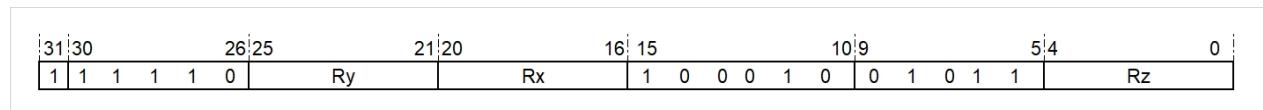


Figure 15.126: MULCAX.S16.

15.127 MULCS.S16 - 16 bit signed multiply chain subtract instructions

Harmonization Directive	
grammatical	mulcs.s16 rz, rx, ry
manipulate	Rz[31:0]= Rx[15:0] X Ry[15:0] - Rx[31:16] X Ry[31:16]
Compilation results	Only 32-bit instructions exist. mulcs.s16 rz, rx, ry

User's Manual

Description:	The lower half-word of Rx is multiplied by the lower half-word of Ry, the higher half-word of Rx is multiplied by the higher half-word of Ry, and the two multiplication results are subtracted.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	not have

32-bit instruction	
Operation:	$Rz[31:0] = Rx[15:0] \times Ry[15:0] - Rx[31:16] \times Ry[31:16]$
Grammar:	mulcs.s16 rz, rx, ry
Description:	The lower half-word of Rx is multiplied by the lower half-word of Ry, the higher half-word of Rx is multiplied by the higher half-word of Ry, and the two multiplication results are subtracted.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

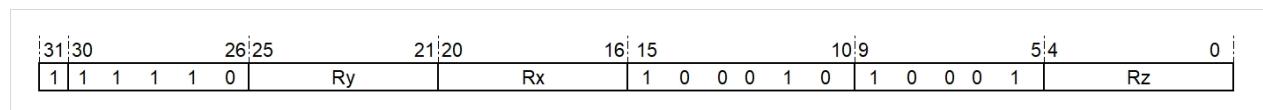


Figure 15.127: MULCS.

15.128 MULCSR.S16 - 16 bit signed reverse multiply chain subtract instructions

Harmonization Directive	
grammatical	mulcsr.s16 rz, rx, ry
manipulate	$Rz[31:0] = Rx[31:16] \times Ry[31:16] - Rx[15:0] \times Ry[15:0]$
Compilation results	Only 32-bit instructions exist. mulcsr.s16 rz, rx, ry

Description:	The high half-word of Rx is multiplied by the high half-word of Ry, the low half-word of Rx is multiplied by the low half-word of Ry, and the two multiplication results are subtracted.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	not have

32-bit	
---------------	--

User Manual	
Operation:	Rz[31:0]= Rx[31:16] X Ry[31:16] - Rx[15:0] X Ry[15:0]
Grammar:	mulcsr.s16 rz, rx, ry
Description:	The high half-word of Rx is multiplied by the high half-word of Ry, the low half-word of Rx is multiplied by the low half-word of Ry, and the two multiplication results are subtracted.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

:31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1	0 1 0 0 1 1	Rz	

Figure 15.128: MULCSR.S16

15.129 MULCSX.S16 - 16 bit signed cross-multiply-chain-subtract instructions

Harmonization Directive	
grammatical	mulcsx.s16 rz, rx, ry
manipulate	$Rz[31:0] = Rx[15:0] \times Ry[31:16] - Rx[31:16] \times Ry[15:0]$
Compilation results	Only 32-bit instructions exist. mulcsx.s16 rz, rx, ry

Description:	The low word of Rx is multiplied by the high half word of Ry, the high half word of Rx is multiplied by the low half word of Ry, and the two multiplication results are subtracted.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	not have

32-bit instruction	
Operation:	$Rz[31:0] = Rx[15:0] \times Ry[31:16] - Rx[31:16] \times Ry[15:0]$
Grammar:	mulcsx.s16 rz, rx, ry
Description:	The low word of Rx is multiplied by the high half word of Ry, the high half word of Rx is multiplied by the low half word of Ry, and the two multiplication results are subtracted.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

:31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 1	0 0 0 0 1	Rz	

15.130 MULACA.S16.S - 16-Bit Signed Multiply Chain Plus Accumulator Finger with Saturation Operation

honorific title

Harmonization Directive	
grammatical	mulaca.s16.s rz, rx, ry
manipulate	$Rz[31:0] = \text{Saturate}(Rz[31:0] + Rx[31:16] \times Ry[31:16] + Rx[15:0] \times Ry[15:0])$
Compilation results	Only 32-bit instructions exist. mulaca.s16.s rz, rx, ry

Explanation:	The low halfword of Rx is multiplied by the low halfword of Ry, the high halfword of Rx is multiplied by the high halfword of Ry, and the results of the two multiplications are added and compared with Rz Accumulate. The result is saturated and stored in Rz. The process of saturation processing is as follows: if the accumulated result is greater than the upper saturation value 0x7FFF FFFF, the result is the upper saturation value, if the accumulated result is less than the lower saturated value 0x8000 0000, the result is the lower saturated value, otherwise it is the accumulation result itself.
Marker Bit:	unaffected
set a limit (on) System:	not have
discrimin	not have

User's Manual
Reg
ular:

32 in d e x (m at h.) hon orifi c title	
drill (pra ctice) Mak e:	Rz[31:0]= Saturate(Rz[31:0]+ Rx[31:16] X Ry[31:16]+ Rx[15:0] X Ry[15:0])
tell to Law:	mulaca.s16.s rz, rx, ry
Exp lana tion : :	The low halfword of Rx is multiplied by the low halfword of Ry, the high halfword of Rx is multiplied by the high halfword of Ry, and the results of the two multiplications are added and compared with Rz. Accumulate. The result is saturated and stored in Rz. The process of saturation processing is as follows: if the accumulated result is greater than the upper saturation value 0x7FFF FFFF , the result is the upper saturation value, if the accumulated result is less than the Lower saturated value 0x8000 0000, the result is the lower saturated value, otherwise it is the accumulation result itself.
M ar k er s of in fl u e n ce Bit:	unaffected

User's Manual	not have
System limit (on)	not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 0	0 1 1 0 1	Rz	

Figure 15.130: MULACA.S16.

15.131 MULACAX.S16.S - 16 bit signed cross multiply chain add with saturation operation accumulator command (computing)

Harmonization Directive	
grammatical	mulacax.s16.s rz, rx, ry
manipulation	$Rz[31:0] = \text{Saturate}(Rz[31:0] + Rx[31:16] \times Ry[15:0] + Rx[15:0] \times Ry[31:16])$
Compilation results	Only 32-bit instructions exist. mulacax.s16.s rz, rx, ry

Exp lana tion : M ar k er s of in fl u e n ce Bit:	The lower half-word of Rx is multiplied by the higher half-word of Ry, and the higher half-word of Rx is multiplied by the lower half-word of Ry, and the two multiplication results are added and combined with Rz Accumulate. The result is saturated and stored in Rz. The process of saturation processing is as follows: if the accumulated result is greater than the upper saturation value 0x7FFF FFFF , the result is the upper saturation value, if the accumulated result is less than the Lower saturated value 0x8000 0000, the result is the lower saturated value, otherwise it is the accumulation result itself.
set a limit (on) Syst em:	not have
discr imin ate Reg ular:	not have

32 in d e x (m at h.) hon orifi	
--	--

User's title	Manual
drill (practice) Mak e:	Rz[31:0]= Saturate(Rz[31:0]+ Rx[31:16] X Ry[15:0]+ Rx[15:0] X Ry[31:16])
tell to Law:	mulacax.s16.s rz, rx, ry
Explana tion :	<p>The lower half-word of Rx is multiplied by the higher half-word of Ry, and the higher half-word of Rx is multiplied by the lower half-word of Ry, and the two multiplication results are added and combined with Rz</p> <p>Accumulate. The result is saturated and stored in Rz.</p> <p>The process of saturation processing is as follows: if the accumulated result is greater than the upper saturation value 0x7FFF FFFF , the result is the upper saturation value, if the accumulated result is less than the</p> <p>Lower saturated value 0x8000 0000, the result is the lower saturated value, otherwise it is the accumulation result itself.</p>
Mark ers of in fl u e n ce Bit:	unaffected
set a limit (on) System:	not have
discriminate Regular:	not have

Command Format:

[31:30]	[26:25]	[21:20]	[16:15]	[10:9]	[5:4]	0
[1 1 1 1 0 Ry]	Rx		[1 0 0 0 1 0]	[0 1 1 1 1 Rz]		

Figure 15.131: MULACAX.S16.

15.132 MULACS.S16.S - 16-bit Signed Multiply-Chain-Subtract Accumulator Finger with Saturation Operation

honorific title

Harmonization Directive	
grammatical	mulacs.s16.s rz, rx, ry
manipulate	$Rz[31:0] = \text{Saturate}(Rz[31:0] + Rx[15:0] \times Ry[15:0] - Rx[31:16] \times Ry[31:16])$
Compilation results	Only 32-bit instructions exist. mulacs.s16.s rz, rx, ry

Explanation:	The lower half-word of Rx is multiplied by the lower half-word of Ry, the higher half-word of Rx is multiplied by the higher half-word of Ry, and the results of the two multiplications are subtracted and combined with Rz. Accumulate. The result is saturated and stored in Rz. The process of saturation processing is as follows: if the accumulated result is greater than the upper saturation value 0x7FFF FFFF, the result is the upper saturation value, if the accumulated result is less than the lower saturated value 0x8000 0000, the result is the lower saturated value, otherwise it is the accumulation result itself.
Marker s of influence Bit:	unaffected
set a limit	not have

(On) User's Manual System:	
discriminate Regular:	not have

32 in d e x (m at h.) hon orifi c title	
drill (pra ctice) Mak e:	Rz[31:0]= Saturate(Rz[31:0]+ Rx[15:0] X Ry[15:0] - Rx[31:16] X Ry[31:16])
tell to Law:	mulacs.s16.s rz, rx, ry
Exp lana tion : :	<p>The lower half-word of Rx is multiplied by the lower half-word of Ry, the higher half-word of Rx is multiplied by the higher half-word of Ry, and the results of the two multiplications are subtracted and combined with Rz</p> <p>Accumulate. The result is saturated and stored in Rz.</p> <p>The process of saturation processing is as follows: if the accumulated result is greater than the upper saturation value 0x7FFF FFFF , the result is the upper saturation value, if the accumulated result is less than the</p> <p>Lower saturated value 0x8000 0000, the result is the lower saturated value, otherwise it is the accumulation result itself.</p>
M ar k er s of in fl u e n ce Bit:	unaffected

Set a limit (on) System:	not have
discriminate Register:	

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 0	1 1 1 0 1	Rz	

Figure 15.132: MULACS.S16.

15.133 MULACSR.S16.S - Signed inverse multiply-chain-subtract with saturation operation 16 bit accumulator command (computing)

Harmonization Directive	
grammatical	mulacsr.s16.s rz, rx, ry
manipulation	$Rz[31:0] = \text{Saturate}(Rz[31:0] + Rx[31:16] \times Ry[31:16] - Rx[15:0] \times Ry[15:0])$
Compilation results	Only 32-bit instructions exist. mulacsr.s16.s rz, rx, ry

Exp lana tion : M ar k er s of in fl u e n ce Bit:	The high half-word of Rx is multiplied by the high half-word of Ry, and the low half-word of Rx is multiplied by the low half-word of Ry. The two multiplication results are subtracted and combined with Rz. Accumulate. The result is saturated and stored in Rz. The process of saturation processing is as follows: if the accumulated result is greater than the upper saturation value 0x7FFF FFFF , the result is the upper saturation value, if the accumulated result is less than the Lower saturated value 0x8000 0000, the result is the lower saturated value, otherwise it is the accumulation result itself.
set a limit (on) Syst em:	not have
discr imin ate Reg ular:	not have

32 in d e x (m at h.) hon orifi	
--	--

User's title	Manual
drill (practice) Mak e:	Rz[31:0]= Saturate(Rz[31:0]+ Rx[31:16] X Ry[31:16] - Rx[15:0] X Ry[15:0])
tell to Law:	mulacsrs16.s rz, rx, ry
Explana tion :	<p>The high half-word of Rx is multiplied by the high half-word of Ry, and the low half-word of Rx is multiplied by the low half-word of Ry. The two multiplication results are subtracted and combined with Rz.</p> <p>Accumulate. The result is saturated and stored in Rz.</p> <p>The process of saturation processing is as follows: if the accumulated result is greater than the upper saturation value 0x7FFF FFFF , the result is the upper saturation value, if the accumulated result is less than the lower saturated value 0x8000 0000, the result is the lower saturated value, otherwise it is the accumulation result itself.</p>
Mark ers of in fl u e n ce Bit:	unaffected
set a limit (on) System:	not have
discriminate Regular:	not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 0	1 1 1 1 1	Rz	

Figure 15.133: MULACSR.S16.

15.134 MULACSX.S16.S - Signed Cross Multiply Chain Subtract with Saturated Operations 16 Bits accumulator command (computing)

Harmonization Directive	
grammatical	mulacsx.s16.s rz, rx, ry
manipulate	$Rz[31:0] = \text{Saturate}(Rz[31:0] + Rx[15:0] \times Ry[31:16] - Rx[31:16] \times Ry[15:0])$
Compilation results	Only 32-bit instructions exist. mulacsx.s16.s rz, rx, ry

Exp lana tion :	<p>The lower half-word of Rx is multiplied by the higher half-word of Ry, and the higher half-word of Rx is multiplied by the lower half-word of Ry. The two multiplication results are subtracted and combined with Rz.</p> <p>Accumulate. The result is saturated and stored in Rz.</p> <p>The process of saturation processing is as follows: if the accumulated result is greater than the upper saturation value 0x7FFF FFFF, the result is the upper saturation value, if the accumulated result is less than the</p> <p>Lower saturated value 0x8000 0000, the result is the lower saturated value, otherwise it is the accumulation result itself.</p>
M ar k er s of in fl u e n ce Bit:	unaffected
set a limit (on) Syst em:	not have
discr imin ate Reg ular:	not have
32 in d e x (m at h.) hon orifi c	

User	Manual
drill (prac- tice) Mak- e:	Rz[31:0]= Saturate(Rz[31:0]+ Rx[15:0] X Ry[31:16] - Rx[31:16] X Ry[15:0])
tell to Law:	mulacsx.s16.s rz, rx, ry
Explana- tion: :	<p>The lower half-word of Rx is multiplied by the higher half-word of Ry, and the higher half-word of Rx is multiplied by the lower half-word of Ry. The two multiplication results are subtracted and combined with Rz</p> <p>Accumulate. The result is saturated and stored in Rz.</p> <p>The process of saturation processing is as follows: if the accumulated result is greater than the upper saturation value 0x7FFF FFFF , the result is the upper saturation value, if the accumulated result is less than the</p> <p>Lower saturated value 0x8000 0000, the result is the lower saturated value, otherwise it is the accumulation result itself.</p>
Mark- er s of in- fl- u- e- n- ce Bit:	unaffected
set a limit (on) Syst- em:	not have
discrim- inate Reg- ular:	not have

Command Format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 1 1 1 0 Ry		Rx	1 0 0 0 1 1 0 1 1 0 1 Rz			

Figure 15.134: MULACSX.S16.

15.135 MULSCA.S16.S - 16-bit Signed Multiply Chain Plus Accumulate and Decrease Finger with Saturation Operation

honorific title

Harmonization Directive	
grammatical	mulsca.s16.s rz, rx, ry
manipulate	Rz[31:0]= Saturate(Rz[31:0] - Rx[31:16] X Ry[31:16] - Rx[15:0] X Ry[15:0])
Compilation results	Only 32-bit instructions exist. mulsca.s16.s rz, rx, ry

Explanation:	The low halfword of Rx is multiplied by the low halfword of Ry, the high halfword of Rx is multiplied by the high halfword of Ry, and the results of the two multiplications are added and compared with Rz Cumulative subtraction. The result is saturated and stored in Rz. The process of saturation processing is as follows: if the result of accumulation and decimation is greater than the upper saturation value 0x7FFF FFFF , the result is the upper saturation value, if the result of accumulation and decimation is less than the lower saturated value 0x8000 0000, the result is the lower saturated value, in other cases it is the accumulated result itself.
Markers of influence Bit:	unaffected
set a limit (on) System:	no have
discrimin	no have

User's Manual
Reg
ular:

32 in d e x (m at h.) hon orifi c title	
drill (pra ctice) Mak e:	Rz[31:0]= Saturate(Rz[31:0] - Rx[31:16] X Ry[31:16] - Rx[15:0] X Ry[15:0])
tell to Law:	mulscas16.s rz, rx, ry
Exp lana tion : :	The low halfword of Rx is multiplied by the low halfword of Ry, the high halfword of Rx is multiplied by the high halfword of Ry, and the results of the two multiplications are added and compared with Rz Cumulative subtraction. The result is saturated and stored in Rz. The process of saturation processing is as follows: if the result of accumulation and decimation is greater than the upper saturation value 0x7FFF FFFF , the result is the upper saturation value, if the result of accumulation and decimation is less than the Lower saturated value 0x8000 0000, the result is the lower saturated value, in other cases it is the accumulated result itself.
M ar k er s of in fl u e n ce Bit:	unaffected

Set up limit (on) Syst em:	Machine not have
discr imin ate Reg ular:	

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 1	0 1 1 1 1	Rz	

Figure 15.135: MULSCA.S16.

15.136 MULSCAX.S16.S - 16 bit signed cross multiply chain add with saturation operation accumulate and decrease instruction

Harmoniza tion Directive	
grammatica l	mulscax.s16.s rz, rx, ry
manipulate	Rz[31:0]= Saturate(Rz[31:0] - Rx[31:16] X Ry[15:0] - Rx[15:0] X Ry[31:16])
Compilatio n results	Only 32-bit instructions exist. mulscax.s16.s rz, rx, ry

Exp lana tion : M ar k er s of in fl u e n ce Bit:	The high half-word of Rx is multiplied by the low half-word of Ry, and the low half-word of Rx is multiplied by the high half-word of Ry. The two multiplication results are added and combined with Rz. Cumulative subtraction. The result is saturated and stored in Rz. The process of saturation processing is as follows: if the result of accumulation and decimation is greater than the upper saturation value 0x7FFF FFFF , the result is the upper saturation value, if the result of accumulation and decimation is less than the Lower saturated value 0x8000 0000, the result is the lower saturated value, in other cases it is the accumulated result itself.
set a limit (on) Syst em:	not have
discr imin ate Reg ular:	not have

32 in d e x (m at h.) hon orifi	
--	--

User's title	Manual
drill (practice) Mak e:	Rz[31:0]= Saturate(Rz[31:0] - Rx[31:16] X Ry[15:0] - Rx[15:0] X Ry[31:16])
tell to Law:	mulscax.s16.s rz, rx, ry
Explana tion :	<p>The high half-word of Rx is multiplied by the low half-word of Ry, and the low half-word of Rx is multiplied by the high half-word of Ry. The two multiplication results are added and combined with Rz</p> <p>Cumulative subtraction. The result is saturated and stored in Rz.</p> <p>The process of saturation processing is as follows: if the result of accumulation and decimation is greater than the upper saturation value 0x7FFF FFFF , the result is the upper saturation value, if the result of accumulation and decimation is less than the Lower saturated value 0x8000 0000, the result is the lower saturated value, in other cases it is the accumulated result itself.</p>
Mark ers of in fl u e n ce Bit:	unaffected
set a limit (on) System:	not have
discriminate Regular:	not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 1	1 1 1 0 1	Rz	

Figure 15.136: MULSCAX.S16.

15.137 MULACA.S16.E - 16-Bit Signed Multiply Chain Plus Accumulator Finger with Extended Operations

honorific title

Harmonization means honorific title	
grammatical	mulaca.s16.e rz, rx, ry
manipulate	$\{Rz+1[31:0], Rz[31:0]\} = \{Rz+1[31:0], Rz[31:0]\} + Rx[31:16] \times Ry[31:16] + Rx[15:0] \times Ry[15:0]$
Compilation results	Only 32-bit instructions exist. mulaca.s16.e rz, rx, ry

Description:	The lower halfword of Rx is multiplied by the lower halfword of Ry, and the higher halfword of Rx is multiplied by the higher halfword of Ry. The results of the two multiplications are added together and combined with the $\{Rz+1, Rz\}$ is totalized. The high 32 bits of the result are stored in $Rz+1$ and the low 32 bits are stored in Rz .
affect Chi Bit:	unaffected
Limitations:	not have
Exception:	not have

32 bits directives	
Operation:	$\{Rz+1[31:0], Rz[31:0]\} = \{Rz+1[31:0], Rz[31:0]\} + Rx[31:16] \times Ry[31:16] + Rx[15:0] \times Ry[15:0]$
Grammar:	mulaca.s16.e rz, rx, ry

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 1	1 1 1 0 1	Rz	

Description:	The lower halfword of Rx is multiplied by the lower halfword of Ry, and the higher halfword of Rx is multiplied by the higher halfword of Ry. The results of the two multiplications are added together and combined with the {Rz+1, Rz} is totalized. The high 32 bits of the result are stored in Rz+1 and the low 32 bits are stored in Rz.
affect Chi Bit:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 1	1 1 1 0 1	Rz	

Figure 15.137: MULACA.S16.

15.138 MULACAX.S16.E - 16 bit signed cross multiply chain add with extended operation accumulator command (computing)

Harmonization means honorific title	
grammatical	mulacax.s16.e rz, rx, ry
manipulate	{Rz+1[31:0],Rz[31:0]}= {Rz+1[31:0],Rz[31:0]}+ Rx[31:16] X Ry[15:0]+ Rx[15:0] X Ry [31:16]
Compilation results	Only 32-bit instructions exist. mulacax.s16.e rz, rx, ry

Description:	The high half-word of Rx is multiplied by the low half-word of Ry, and the low half-word of Rx is multiplied by the high half-word of Ry. The results of the two multiplications are added together and combined with the {Rz+1, Rz} is accumulated. The high 32 bits of the result are stored in Rz+1 and the low 32 bits are stored in Rz.
affect Chi Bit:	unaffected
Limitations:	not have
Exception:	not have

32 bits directives	
Operation:	{Rz+1[31:0],Rz[31:0]}= {Rz+1[31:0],Rz[31:0]}+ Rx[31:16] X Ry[15:0]+ Rx[15:0] X Ry[31:16]
Grammar:	mulacax.s16.e rz, rx, ry

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 1	1 1 1 0 1	Rz	

Description:	The high half-word of Rx is multiplied by the low half-word of Ry, and the low half-word of Rx is multiplied by the high half-word of Ry. The results of the two multiplications are added together and combined with the {Rz+1, Rz} is totalized. The high 32 bits of the result are stored in Rz+1 and the low 32 bits are stored in Rz.
affect Chi Bit:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 1	1 1 1 0 1	Rz	

Figure 15.138: MULACAX.S16.

15.139 MULACS.S16.E - 16-Bit Signed Multiply Chain Subtract Accumulator Finger with Extended Operations honorific title

Harmonization means honorific title	
grammatical	mulacs.s16.e rz, rx, ry
manipulate	$\{Rz+1[31:0], Rz[31:0]\} = \{Rz+1[31:0], Rz[31:0]\} + Rx[15:0] \times Ry[15:0] - Rx[31:16] \times Ry[31:16]$
Compilation results	Only 32-bit instructions exist. mulacs.s16.e rz, rx, ry

Description:	The lower half-word of Rx is multiplied by the lower half-word of Ry, and the higher half-word of Rx is multiplied by the higher half-word of Ry. The two multiplication results are subtracted and compared with $\{Rz+1, Rz\}$. The high 32 bits of the result are stored in $Rz+1$ and the low 32 bits are stored in Rz .
affect Chi Bit:	unaffected
Limitations:	not have
Exception:	not have

32 bits directives	
Operation:	$\{Rz+1[31:0], Rz[31:0]\} = \{Rz+1[31:0], Rz[31:0]\} + Rx[15:0] \times Ry[15:0] - Rx[31:16] \times Ry[31:16]$
Grammar:	mulacs.s16.e rz, rx, ry

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 1	1 1 1 0 1	Rz	

Description:	The lower half-word of Rx is multiplied by the lower half-word of Ry, and the higher half-word of Rx is multiplied by the higher half-word of Ry. The two multiplication results are subtracted and compared with {Rz+1, Rz} is totalized. The high 32 bits of the result are stored in Rz+1 and the low 32 bits are stored in Rz.
affect Chi Bit:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 1	1 1 1 0 1	Rz	

Figure 15.139: MULACAX.S16.

15.140 MULACSR.S16.E - Signed inverse multiply-chain-subtract with extended operation 16 bits accumulator command (computing)

Harmonization means honorific title	
grammatical	mulacsr.s16.e rz, rx, ry
manipulate	$\{Rz+1[31:0], Rz[31:0]\} = \{Rz+1[31:0], Rz[31:0]\} + Rx[31:16] \times Ry[31:16] - Rx[15:0] \times Ry[15:0]$
Compilation results	Only 32-bit instructions exist. mulacsr.s16.e rz, rx, ry

Description:	The high half-word of Rx is multiplied by the high half-word of Ry, and the low half-word of Rx is multiplied by the low half-word of Ry. The two multiplication results are subtracted and compared with $\{Rz+1, Rz\}$ is totalized. The high 32 bits of the result are stored in $Rz+1$ and the low 32 bits are stored in Rz .
affect Chi Bit:	unaffected
Limitations:	not have
Exception:	not have

32 bits directives	
Operation:	$\{Rz+1[31:0], Rz[31:0]\} = \{Rz+1[31:0], Rz[31:0]\} + Rx[31:16] \times Ry[31:16] - Rx[15:0] \times Ry[15:0]$
Grammar:	mulacsr.s16.e rz, rx, ry

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 1	1 1 1 0 1	Rz	

Description:	The high half-word of Rx is multiplied by the high half-word of Ry, and the low half-word of Rx is multiplied by the low half-word of Ry. The two multiplication results are subtracted and compared with {Rz+1, Rz} is totalized. The high 32 bits of the result are stored in Rz+1 and the low 32 bits are stored in Rz.
affect Chi Bit:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 1	1 1 1 0 1	Rz	

Figure 15.140: MULACSR.S16.

15.141 MULACSRX.S16.E - 16-bit signed cross multiplication chain with extended operations minus accumulate instruction

Harmonization means honorific title	
grammatical	mulacsx.s16.e rz, rx, ry
manipulate	$\{Rz+1[31:0], Rz[31:0]\} = \{Rz+1[31:0], Rz[31:0]\} + Rx[15:0] \times Ry[31:16] - Rx[31:16] \times Ry[15:0]$
Compilation results	Only 32-bit instructions exist. mulacsx.s16.e rz, rx, ry

Description:	The lower half-word of Rx is multiplied by the higher half-word of Ry, and the higher half-word of Rx is multiplied by the lower half-word of Ry. The results of the two multiplications are subtracted and combined with the $\{Rz+1, Rz\}$ is totalized. The high 32 bits of the result are stored in $Rz+1$ and the low 32 bits are stored in Rz .
affect Chi Bit:	unaffected
Limitations:	not have
Exception:	not have

32 bits directives	
Operation:	$\{Rz+1[31:0], Rz[31:0]\} = \{Rz+1[31:0], Rz[31:0]\} + Rx[15:0] \times Ry[31:16] - Rx[31:16] \times Ry[15:0]$
Grammar:	mulacsx.s16.e rz, rx, ry

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 1	1 1 1 0 1	Rz	

Description:	The lower half-word of Rx is multiplied by the higher half-word of Ry, and the higher half-word of Rx is multiplied by the lower half-word of Ry. The results of the two multiplications are subtracted and combined with the {Rz+1, Rz} is totalized. The high 32 bits of the result are stored in Rz+1 and the low 32 bits are stored in Rz.
affect Chi Bit:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 1	1 1 1 0 1	Rz	

Figure 15.141: MULACSRX.S16.

15.142 MULSCA.S16.E - 16-Bit Signed Multiply Chain Plus Accumulator and Decreaser Finger with Extended Operations

honorific title

Harmonization means honorific title	
grammatical	mulscas16.e rz, rx, ry
manipulate	{Rz+1[31:0],Rz[31:0]}= {Rz+1[31:0],Rz[31:0]} - Rx[31:16] X Ry[31:16] - Rx[15:0] X Ry[15:0]
Compilation results	Only 32-bit instructions exist. mulscas16.e rz, rx, ry

Description:	The lower halfword of Rx is multiplied by the lower halfword of Ry, and the higher halfword of Rx is multiplied by the higher halfword of Ry. The results of the two multiplications are added together and combined with the {Rz+1, Rz} Decrement. The high 32 bits of the result are stored in Rz+1 and the low 32 bits are stored in Rz.
affect Chi Bit:	unaffected
Limitations:	not have
Exception:	not have

32 bits directives	
Operation:	{Rz+1[31:0],Rz[31:0]}= {Rz+1[31:0],Rz[31:0]} - Rx[31:16] X Ry[31:16] - Rx[15:0] X Ry[15:0]
Grammar:	mulscas16.e rz, rx, ry

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 1	1 1 1 0 1	Rz	

Description:	The lower halfword of Rx is multiplied by the lower halfword of Ry, and the higher halfword of Rx is multiplied by the higher halfword of Ry. The results of the two multiplications are added together and combined with the {Rz+1, Rz} Decrement. The high 32 bits of the result are stored in Rz+1 and the low 32 bits are stored in Rz.
affect Chi Bit:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 1	1 1 1 0 1	Rz	

Figure 15.142: MULSCA.S16.

15.143 MULSCAX.S16.E - 16 bit signed cross multiply chain add with extended operation accumulate and decrease instruction

Harmonization means honorific title	
grammatical	mulscax.s16.e rz, rx, ry
manipulate	{Rz+1[31:0],Rz[31:0]}= {Rz+1[31:0],Rz[31:0]} - Rx[31:16] X Ry[15:0] - Rx[15:0] X Ry [31:16]
Compilation results	Only 32-bit instructions exist. mulscax.s16.e rz, rx, ry

Description:	The lower half-word of Rx is multiplied by the higher half-word of Ry, and the higher half-word of Rx is multiplied by the lower half-word of Ry. The results of the two multiplications are added together and combined with the {Rz+1, Rz} Decrement. The high 32 bits of the result are stored in Rz+1 and the low 32 bits are stored in Rz.
affect Chi Bit:	unaffected
Limitations:	not have
Exception:	not have

32 bits directives	
Operation:	{Rz+1[31:0],Rz[31:0]}= {Rz+1[31:0],Rz[31:0]} - Rx[31:16] X Ry[15:0] - Rx[15:0] X Ry[31:16]
Grammar:	mulscax.s16.e rz, rx, ry

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 1	1 1 1 0 1	Rz	

Description:	The lower half-word of Rx is multiplied by the higher half-word of Ry, and the higher half-word of Rx is multiplied by the lower half-word of Ry. The results of the two multiplications are added together and combined with the {Rz+1, Rz} Decrement. The high 32 bits of the result are stored in Rz+1 and the low 32 bits are stored in Rz.
affect Chi Bit:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 1	1 1 1 0 1	Rz	

Figure 15.143: MULSCAX.S16.

15.144 PSABSA.U8 - 8 Bit Parallel Unsigned Subtracted Absolute Value Chaining Instructions

Unification directives	
grammatical	psabsa.u8 rz, rx, ry
manipulate	$Rz[31:0] = \text{abs}(Rx[7:0] - Ry[7:0]) + \text{abs}(Rx[15:8] - Ry[15:8]) + \text{abs}(Rx[23:16] - Ry[23:16]) + \text{abs}(Rx[31:24] - Ry[31:24])$
Translation resolute	Only 32-bit instructions exist. psabsa.u8 rz, rx, ry

Description:	Subtract each of 4 bytes of Rx from the 4 bytes Ry in bytes, and chain the subtraction results to the absolute value. The result of chaining is deposited in Rz.
Markers of influence Bit:	unaffected
Limitations:	not have
Exception:	not have

32 Bitmap honorific title	
Operations:	$Rz[31:0] = \text{abs}(Rx[7:0] - Ry[7:0]) + \text{abs}(Rx[15:8] - Ry[15:8]) + \text{abs}(Rx[23:16] - Ry[23:16]) + \text{abs}(Rx[31:24] - Ry[31:24])$
Grammar:	psabsa.u8 rz, rx, ry

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 1	1 1 1 0 1	Rz	

Description:	In byte units, the 4 bytes Rx are subtracted from the 4 bytes Ry, and the subtraction result is taken as the absolute value and chained together. The result of the addition is deposited in Rz.
Markers of influence Bit:	unaffected
Limitations:	not have
Exception :	not have

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 1	1 1 1 0 1	Rz	

Figure 15.144: PSABSA.U8

15.145 PSABSAA.U8 - 8 Bit Parallel Signed or Unsigned Subtracted Absolute Value Chaining Accumulation Instruction

Unification directives	
grammatical	psabsaa.u8 rz, rx, ry
manipulate	$Rz[31:0] = Rz[31:0] + \text{abs}(Rx[7:0] - Ry[7:0]) + \text{abs}(Rx[15:8] - Ry[15:8]) + \text{abs}(Rx[23:16] - Ry[23:16]) + \text{abs}(Rx[31:24] - Ry[31:24])$
Translation resolute	Only 32-bit instructions exist. psabsaa.u8 rz, rx, ry

Description:	In byte units, the 4 bytes Rx are subtracted from the 4 bytes Ry, and the subtraction result is taken as the absolute value and chained together, chained The result is accumulated with Rz and stored in Rz.
Markers of influence Bit:	unaffected
Limitations:	not have
Exception:	not have

32-bit finger honorific title	
Operation:	$Rz[31:0] = Rz[31:0] + \text{abs}(Rx[7:0] - Ry[7:0]) + \text{abs}(Rx[15:8] - Ry[15:8]) + \text{abs}(Rx[23:16] - Ry[23:16]) + \text{abs}(Rx[31:24] - Ry[31:24])$
Grammar:	psabsaa.u8 rz, rx, ry

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 1	1 1 1 0 1	Rz	

Description:	In byte units, the 4 bytes Rx are subtracted from the 4 bytes Ry, and the subtraction result is taken as the absolute value and chained together, chained. The result is accumulated with Rz and stored in Rz.
impact indicator:	unaffected
Chi Bit:	
Limitations:	not have
Exception:	not have
Command Forms Style:	

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 1	1 1 1 0 1	Rz	

Figure 15.145: PSABSAA.U8

15.146 DIVSL - Signed Long Division Instruction

Harmonization Directive	
grammatical	divsl rz, rx, ry
manipulate	signed division $\{Rz+1[31:0], Rz[31:0]\} = \{Rx+1[31:0], Rx[31:0]\} / Ry[31:0]$
Compilation results	32-bit instructions only divsl rz, rx, ry

Explanation:	{Rx+1,Rx} consists of a 64-bit signed number divided by the value of RY. The upper 32 bits of the result are stored in Rz+1, and the lower 32 bits are stored in Rz+1. Bits are deposited into Rz. The values of Rx+1, Ry, and Rz+1 are all considered to be signed numbers. Note that 0x80000000_00000000 is divided by 0xFFFF FFFF, the result is undefined.
Markers of influence Bit:	unaffected
set a limit (on) System:	not have
discriminate Regular:	zero removal anomaly (math.)

32 inde x (mat h.) honor ific title	
drill (pract ice) Make :	signed division $\{Rz+1[31:0], Rz[31:0]\} = \{Rx+1[31:0], Rx[31:0]\} / Ry[31:0]$
tell to Law:	divs132 rz, rx, ry
Exp lana tion :	{Rx+1,Rx} consists of a 64-bit signed number divided by the value of RY. The upper 32 bits of the result are stored in Rz+1, and the lower 32 bits are stored in Rz+1. Bits are deposited into Rz. The values of Rx+1, Ry, and Rz+1 are all considered to be signed numbers. Note that 0x80000000_00000000 is divided by 0xFFFF FFFF, the result is undefined.
Mar ker s of infl uen ce Bit:	unaffected
set a limit (on) Syste m:	not have
discri minat e Regul ar:	zero removal anomaly (math.)

Command Format:

31 30	26 25	21 20	16 15	10 9	5 4	0
[1 1 1 1 1 0]	Ry	Rx	[1 1 1 0 0 0]	[1 0 1 1 1]	Rz	

Figure 15.146: DIVSL

15.147 DIVUL - Unsigned Long Division Instruction

Harmonization Directive	
grammatical	divul rz, rx, ry
manipulate	unsigned division $\{Rz+1[31:0], Rz[31:0]\} = \{Rx+1[31:0], Rx[31:0]\} / Ry[31:0]$
Compilation results	32-bit instructions only divul rz, rx, ry

Description:	Form {Rx+1,Rx} into a 64-bit unsigned number, divide by the value of RY, and store the high 32 bits of the result in Rz+1 and the low 32 bits in Rz+1. 32 bits are deposited into Rz. The values of Rx+1, Ry, and Rz+1 are all considered unsigned numbers.
affect Chi Bit:	unaffected
Limitations:	not have
Exception:	zero removal anomaly (math.)

32-bit 32-bit (computing) directives	
Operation:	unsigned division $\{Rz+1[31:0], Rz[31:0]\} = \{Rx+1[31:0], Rx[31:0]\} / Ry[31:0]$
Grammar:	divul rz, rx, ry
Description:	Form {Rx+1,Rx} into a 64-bit unsigned number, divide by the value of RY, and store the high 32 bits of the result in Rz+1 and the low 32 bits in Rz+1. 32 bits are deposited into Rz. The values of Rx+1, Ry, and Rz+1 are all considered unsigned numbers.
affect Chi Bit:	unaffected
Limitations:	not have
Exception:	zero removal anomaly (math.)
order form Style:	

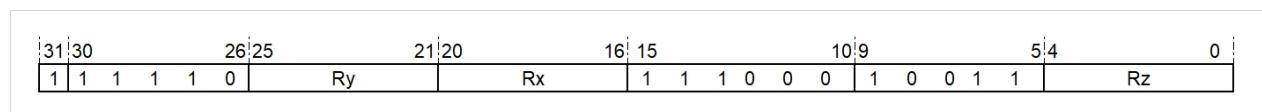


Figure 15.147: DIVUL

15.148 MULACA.S8 - 8 bit parallel signed multiply chain plus

accumulate instructions

collectively known as "harmonization" or "harmonization index" (math.) honorific title	
grammatical	mulaca.s8 rz, rx, ry
manipulate	Rz= Rx[7:0] X Ry[7:0]+ Rx[15:8] X Ry[15:8]+ Rx[23:16] X Ry[23:16]+ Rx[31:24] X Ry [31:24]
Compilation results	Only 32-bit instructions exist. mulaca.s8 rz, rx, ry

Description:	Multiply each of 4 bytes of Rx with the 4 bytes Ry in bytes, and the resultant chain of multiplication is thickened and deposited into Rz.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	not have

32-bit instruction	
Operation:	$Rz = Rx[7:0] * Ry[7:0] + Rx[15:8] * Ry[15:8] + Rx[23:16] * Ry[23:16] + Rx[31:24] * Ry[31:24]$.
Grammar:	mulaca.s8 rz, rx, ry
Description :	Multiply each of 4 bytes of Rx with each of 4 bytes of Ry in bytes, and the resultant chain of multiplication is thickened and deposited into Rz.
Impact Marker Bit:	unaffected
Limitations:	not have
Exception:	not have

Command Format:

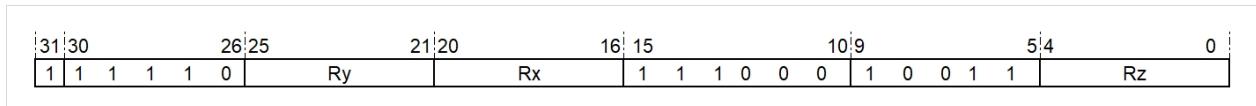


Figure 15.148: MULACA.S8

15.149 BLOOP - Loop Body Acceleration Instruction

unify chem icali zatio n honori fic title	
gram matica l	bloop rx, label1, label2

User's Manual	<p>Rx: loop count register label1: start address of the loop body label2: end address of the loop body</p> <p>Determine whether Rx is equal to 0. If it is not equal to 0, iteratively execute the program fragments on [label1,label2], and the number of executions is determined by the number of times Rx is equal to 0.</p> <p>Values are determined.</p>
Translation in the end	<p>Only 32-bit instructions exist. bloop rx, label1, label2</p>

Exp lana tion :	The last two instructions of a typical loop body are: loop count reduction instruction and comparison jump instruction, BLOOP instruction is used to accelerate the loop body, which avoids the execution of loop count reduction instruction and comparison jump instruction in the loop body. Specific use [label1,label2] on the The program fragment should not include loop traversal register self-decrement instructions, comparison jump instructions.
Mar kers of infl uen ce Bit:	not have
set a limit (on) Syste m:	The program fragments on [label1,label2] should not modify Rx, otherwise the program results are not expected. The loop count Rx is not greater than 0x10000.
discri mina te Regu lar:	not have
ord er for m Style:	

32 index (mat h.) hono rific title	
Ope rate :	Rx: loop count register label1: start address of the loop body label2: end address of the loop body Determine whether Rx is equal to 0, if not equal to 0, repeatedly execute the program fragment on [label 1,label2], and the number of executions is determined by the value of Rx. Values are determined.

User Manual	
tell to Law:	bloop rx, label1, label2
Explana- tion: :	The last two instructions of a typical loop body are: loop count reduction instruction and comparison jump instruction, BLOOP instruction is used to accelerate the loop body, which avoids the execution of loop count reduction instruction and comparison jump instruction in the loop body. Specific use [label1,label2] on the The program fragment should not include loop traversal register self-decrement instructions, comparison jump instructions.
Mark- ers of infl- uen- ce Bit:	not have
set a limit (on) Syste- m:	The program fragments on [label1,label2] should not modify Rx, otherwise the program results are not expected. The loop count Rx is not greater than 0x10000.
discri- mina- te Regu- lar:	not have

Command Format:

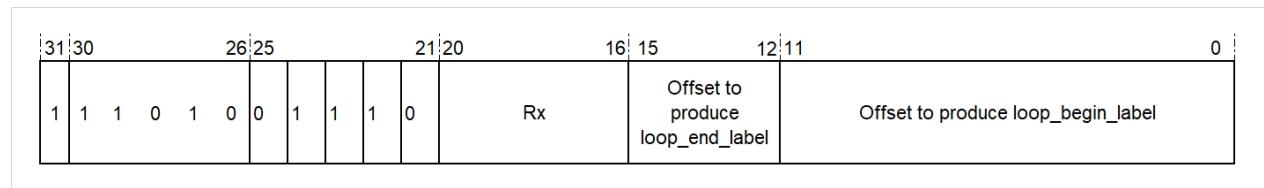


Figure 15.149: BLOOP

15.150 LDBI.W - Address self-incrementing word load instruction

Harmonization Directive	
grammatical	ldbi.w rz, (rx)
manipulate	Rz[31:0] <- mem(Rx) Rx[31:0] <- Rx[31:0]+ 4
Compilation results	Only 32-bit instructions exist. ldbi.w rz, (rx)

Description:	A word is loaded from the address corresponding to Rx and deposited into Rz, while Rx plus 4 is deposited into Rx.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	Access Error Exception

32-bit instruction	
Operation:	Rz[31:0] <- mem(Rx) Rx[31:0] <- Rx[31:0]+ 4
Grammar:	ldbi.w rz, (rx)
Description:	A word is loaded from the address corresponding to Rx and deposited into Rz, while Rx plus 4 is deposited into Rx.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	Access Error Exception

Command Format:

31	30		26	25		21	20		16	15		10	9		5	4		0	
1	1	0	1	0	0	0		0	0	0	0	Rx	1	0	0	0	1	0	0

Figure 15.150: LDBI.

15.151 LDBI.H - Unsigned half-word load instruction with self-incrementing address

Harmonization Directive	
grammatical	ldbi.h rz, (rx)
manipulate	Rz[31:0] <- zero_extend(mem(Rx)) Rx[31:0] <- Rx[31:0]+2
Compilation results	Only 32-bit instructions exist. ldbi.h rz, (rx)

Description:	A half word is loaded from the address corresponding to Rx, the high 16 bits are added to 0 and stored in Rz, and Rx is added to 2 and stored in Rx.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	Access Error Exception

32-bit instruction	
Operation:	Rz[31:0] <- zero_extend(mem(Rx)) Rx[31:0] <- Rx[31:0]+2
Grammar:	ldbi.h rz, (rx)
Description:	A half word is loaded from the address corresponding to Rx, the high 16 bits are added to 0 and stored in Rz, and Rx is added to 2 and stored in Rx.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	Access Error Exception

Command Format:

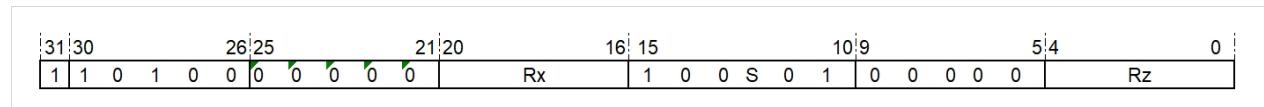


Figure 15.151: LDBI.

15.152 LDBI.HS - address self-incrementing signed half-word

load instruction

Harmonization Directive	
grammatical	ldbi.hs rz, (rx)
manipulate	$Rz[31:0] \leftarrow \text{sign_extend}(\text{mem}(Rx))$ $Rx[31:0] \leftarrow Rx[31:0] + 2$
Compilation results	Only 32-bit instructions exist. ldbi.hs rz, (rx)

Description:	Load a half-word from the address corresponding to Rx, expand the sign bit by the high 16 bits, and store the result of the expansion into Rz, while Rx Add 2 and deposit into Rx.
Markers of influence Bit:	unaffected
Limitations:	not have
Exception:	Access Error Exception

32-bit instruction	
Operation :	Rz[31:0] <- sign_extend(mem(Rx)) Rx[31:0] <- Rx[31:0]+ 2
Grammar:	ldbi.hs rz, (rx)
Description:	Load a half-word from the address corresponding to Rx, expand the sign bit by the high 16 bits, and store the result of the expansion into Rz, while Rx Add 2 and deposit into Rx.
Markers of influence Bit:	unaffected
Limitations:	not have
Exception:	Access Error Exception

Command Format:

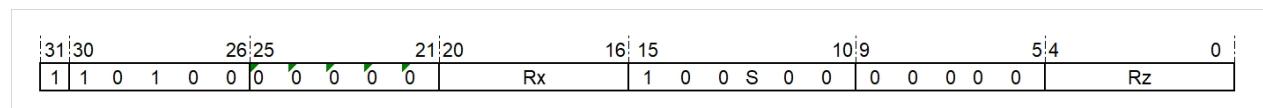


Figure 15.152: LDBI.HS

15.153 LDBI.B - Unsigned byte load instruction with self-incrementing address

Harmonization Directive	
grammatical	ldbi.b rz, (rx)
manipulate	Rz[31:0] <- zero_extend(mem(Rx)) Rx[31:0] <- Rx[31:0]+ 1
Compilation results	Only 32-bit instructions exist. ldbi.b rz, (rx)

Description:	One byte is loaded from the address corresponding to Rx, the high 24 bits are zero-complemented and deposited into Rz, while Rx is added by one and deposited into Rx.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	Access Error Exception

32-bit instruction	
Operation:	Rz[31:0] <- zero_extend(mem(Rx)) Rx[31:0] <- Rx[31:0]+1
Grammar:	ldbi.b rz, (rx)
Description:	One byte is loaded from the address corresponding to Rx, the high 24 bits are zero-complemented and deposited into Rz, while Rx is added by one and deposited into Rx.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	Access Error Exception

Command Format:

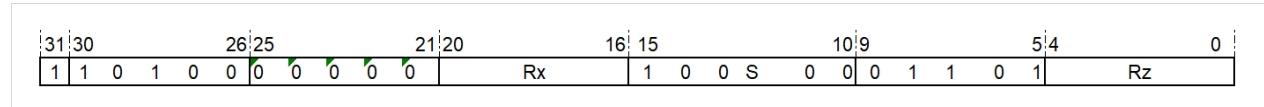


Figure 15.153: LDBI.

15.154 LDBI.BS - address self-incrementing signed byte load instruction

Harmonization Directive	
grammatical	ldbi.bs rz, (rx)
manipulate	Rz[31:0] <- sign_extend(mem(Rx)) Rx[31:0] <- Rx[31:0]+1
Compilation results	Only 32-bit instructions exist. ldbi.bs rz, (rx)

Description:	One byte is loaded from the address corresponding to Rx, the high 24 bits are sign bit expanded, and the result of the expansion is stored in Rz, while Rx Add 1 and store it in Rx.
Markers of influence	unaffected

User's Manual	
Limitations:	not have
Exception:	Access Error Exception

32-bit instruction	
Operation :	Rz[31:0] <- sign_extend(mem(Rx)) Rx[31:0] <- Rx[31:0]+1
Grammar:	ldbi.bs rz, (rx)
Description:	One byte is loaded from the address corresponding to Rx, the high 24 bits are sign bit expanded, and the result of the expansion is stored in Rz, while Rx Add 1 and store it in Rx.
Markers of influence Bit:	unaffected
Limitations:	not have
Exception:	Access Error Exception

Command Format:

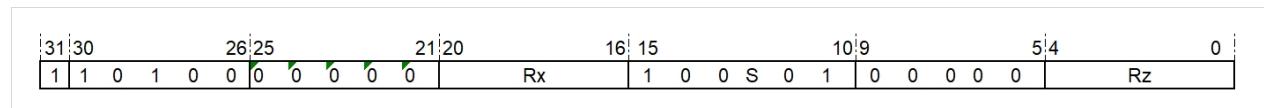


Figure 15.154: LDBI.BS

15.155 PLDBI.D - Address self-incrementing double-word load instruction

standardization directives	
grammatical	pldbi.d rz, (rx)
manipulate	Rz[31:0] <- mem(Rx) Rz+1[31:0] <- mem(Rx+4) Rx[31:0] <- Rx[31:0]+8 and prefetch the data on mem(Rx+8), mem(Rx+12) into the processor's prefetch cache.

Serial	Only 32-bit instructions exist.
Users Manual	pldbi.d rz, (rx)
action	
result	
resolute	

Description:	Load a word from the address corresponding to Rx and store it in Rz, and load a word from the address corresponding to Rx+1 and store it in Rz+1, and at the same time, Rx plus This instruction also prefetches the data on mem(Rx+8), mem(Rx+12) and stores it in the processor's prefetch cache. Center.
impact Logo Bit:	unaffected
Limitations:	not have
Exception:	Unaligned access exception, access error exception
Directives Format:	

32-bit directives	
Operation:	Rz[31:0] <- mem(Rx) Rz+1[31:0] <- mem(Rx+4) Rx[31:0] <- Rx[31:0]+ 8 and prefetch the data on mem(Rx+8), mem(Rx+12) into the processor's prefetch cache.
Grammar:	pldbi.d rz, (rx)
Description:	Load a word from the address corresponding to Rx and store it in Rz, and load a word from the address corresponding to Rx+1 and store it in Rz+1, and at the same time, Rx plus This instruction also prefetches the data on mem(Rx+8), mem(Rx+12) and stores it in the processor's prefetch cache. Center.
impact Logo Bit:	unaffected
Limitations:	not have
Exception:	Unaligned access exception, access error exception

Command Format:

31	30		26	25		21	20		16	15		10	9		5	4		0
1	1	0	1	0	0	0	0	0	1	0	0	0	1	1	0	0	0	Rz

Figure 15.155: PLDBI.

15.156 STBI.W - Address self-incrementing word store instruction

Harmonization Directive	
grammatical	stbi.w rz, (rx)
manipulate	mem(Rx) <- Rz[31:0] Rx[31:0] <- Rx[31:0]+4
Compilation results	Only 32-bit instructions exist. stbi.w rz, (rx)

Description:	Rz is stored in the address corresponding to Rx, and Rx is stored in Rx adding 4 to Rx.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	Access Error Exception

32-bit instruction	
Operation:	mem(Rx) <- Rz[31:0] Rx[31:0] <- Rx[31:0]+4
Grammar:	stbi.w rz, (rx)
Description:	Rz is stored in the address corresponding to Rx, and Rx is stored in Rx adding 4 to Rx.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	Access Error Exception

Command Format:

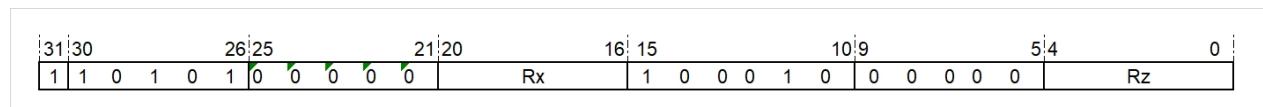


Figure 15.156: STBI.

15.157 STBI.H - address self-incrementing half-word store

instruction

Harmonization Directive	
grammatical	stbi.h rz, (rx)
manipulate	mem(Rx) <- Rx[15:0] Rx[31:0] <- Rx[31:0]+ 2
Compilation results	Only 32-bit instructions exist. stbi.h rz, (rx)

Description:	The lower half-word of Rx is stored in the address corresponding to Rx, and Rx is stored in Rx adding 2.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	Access Error Exception

32-bit instruction	
Operation:	mem(Rx) <- Rx[15:0] Rx[31:0] <- Rx[31:0]+2
Grammar:	stbi.h rz, (rx)
Description:	The lower half-word of Rx is stored in the address corresponding to Rx, and Rx is stored in Rx adding 2.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	Access Error Exception

Command Format:

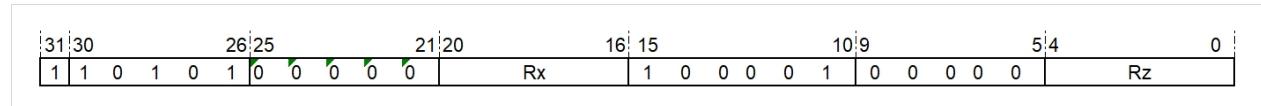


Figure 15.157: STBI.

15.158 STBI.B - Address self-incrementing byte store instruction

Harmonization Directive	
grammatical	stbi.b rz, (rx)
manipulate	mem(Rx) <- Rx[7:0] Rx[31:0] <- Rx[31:0]+1
Compilation results	Only 32-bit instructions exist. stbi.b rz, (rx)

User's Manual	Description: The lower 8 bits of Rz are stored in the address corresponding to Rx, and Rx is written to Rx by adding 1 Rx.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	Access Error Exception

32-bit instruction	
Operation:	mem(Rx) <- Rx[7:0] Rx[31:0] <- Rx[31:0]+ 1
Grammar:	stbi.b rz, (rx)
Description:	The lower 8 bits of Rz are stored in the address corresponding to Rx, and Rx is written to Rx by adding 1 Rx.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	Access Error Exception

Command Format:

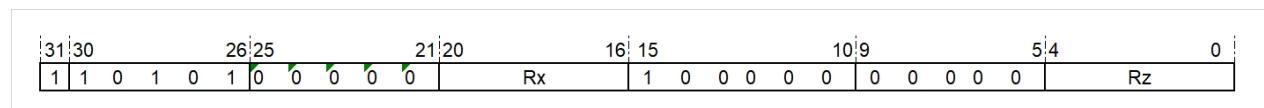


Figure 15.158: STBI.

15.159 LDBIR.W - register address self-incrementing word load instruction

Harmonization Directive	
grammatical	ldbir.w rz, (rx), ry
manipulate	Rz[31:0] <- mem(Rx) Rx[31:0] <- Rx[31:0]+ Ry[31:0]
Compilation results	Only 32-bit instructions exist. ldbir.w rz, (rx), ry

Description:	A word is loaded from the address corresponding to Rx and deposited into Rz, while Rx is added to Ry and deposited into Rx.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	Access Error Exception

32-bit instruction	
Operation:	Rz[31:0] <- mem(Rx) Rx[31:0] <- Rx[31:0]+ Ry[31:0]
Grammar:	ldbir.w rz, (rx), ry
Description:	A word is loaded from the address corresponding to Rx and deposited into Rz, while Rx is added to Ry and deposited into Rx.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	Access Error Exception

Command Format:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 0 1 0 0	Ry	Rx	1 0 1 0 1 0	0 0 0 0 0 0	Rz	

Figure 15.159: LDBIR.

15.160 LDBIR.H - unsigned half-word load instruction with register address self-incrementation

Harmonization Directive	
grammatical	ldbir.h rz, (rx), ry
manipulate	Rz[31:0] <- zero_extend(mem(Rx)) Rx[31:0] <- Rx[31:0]+ Ry[31:0]
Compilation results	Only 32-bit instructions exist. ldbir.h rz, (rx), ry

Description:	A half word is loaded from the address corresponding to Rx, and the high 16 bits are added to 0 and stored in Rz, while Rx is added to Ry and stored in Rx.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	Access Error Exception

32-bit instruction	
Operation:	Rz[31:0] <- zero_extend(mem(Rx)) Rx[31:0] <- Rx[31:0]+ Ry[31:0]
Grammar:	ldbir.h rz, (rx), ry
Description:	A half word is loaded from the address corresponding to Rx, and the high 16 bits are added to 0 and stored in Rz, while Rx is added to Ry and stored in Rx.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	Access Error Exception

Command Format:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1	0	1	0	0	Ry	Rx	1	0	1	S	0

Figure 15.160: LDBIR.H

15.161 LDBIR.HS - register address self-incrementing signed half-word load instruction

Harmonization Directive	
grammatical	ldbir.hs rz, (rx), ry
manipulate	Rz[31:0] <- zero_extend(mem(Rx)) Rx[31:0] <- Rx[31:0]+ Ry[31:0]
Compilation results	Only 32-bit instructions exist. ldbir.hs rz, (rx), ry

Description:	Load a half-word from the address corresponding to Rx, expand the sign bit by the high 16 bits, and store the result of the expansion into Rz, while Rx Add Ry and deposit into Rx.
Markers of influence Bit:	unaffected
Limitations:	not have
Exception:	Access Error Exception

32-bit fingerpri nt honorific title	
Operation :	Rz[31:0] <- zero_extend(mem(Rx)) Rx[31:0] <- Rx[31:0]+ Ry[31:0]
Grammar:	ldbir.hs rz, (rx), ry
Description:	Load a half-word from the address corresponding to Rx, expand the sign bit by the high 16 bits, and store the result of the expansion into Rz, while Rx Add Ry and deposit into Rx.
Markers of influence Bit:	unaffected
Limitation	not have

Command Format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 1 0 0	Ry	Rx	1 0 1 S 0 1	0 0 0 0 0	Rz	

Figure 15.161: LDBIR.HS

15.162 LDBIR.B - register address self-incrementing unsigned byte load instruction

Harmonization Directive	
grammatical	ldbir.b rz, (rx), ry
manipulate	$Rz[31:0] \leftarrow \text{zero_extend}(\text{mem}(Rx))$ $Rx[31:0] \leftarrow Rx[31:0] + Ry[31:0]$
Compilation results	Only 32-bit instructions exist. ldbir.b rz, (rx), ry

Description:	One byte is loaded from the address corresponding to Rx, the high 24 bits are zero-complemented and stored in Rz, while Rx is added to Ry and stored in Rx.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	Access Error Exception

32-bit instruction	
Operation:	$Rz[31:0] \leftarrow \text{zero_extend}(\text{mem}(Rx))$ $Rx[31:0] \leftarrow Rx[31:0] + Ry[31:0]$
Grammar:	ldbir.b rz, (rx), ry
Description:	One byte is loaded from the address corresponding to Rx, the high 24 bits are zero-complemented and stored in Rz, while Rx is added to Ry and stored in Rx.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	Access Error Exception

Command Format:

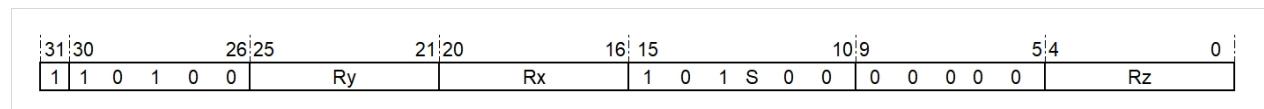


Figure 15.162: LDBIR.

15.163 LDBIR.BS - register address self-incrementing signed byte load instruction

Harmonization Directive	
grammatical	ldbir.bs rz, (rx), ry
manipulate	Rz[31:0] <- sign_extend(mem(Rx)) Rx[31:0] <- Rx[31:0]+ Ry[31:0]
Compilation results	Only 32-bit instructions exist. ldbir.bs rz, (rx), ry

Description:	One byte is loaded from the address corresponding to Rx, the high 24 bits are sign bit expanded, and the result of the expansion is stored in Rz, while Rx Add Ry and deposit into Rx.
Markers of influence Bit:	unaffected
Limitations:	not have
Exception:	Access Error Exception

32-bit fingerpri nt honorific title	
Operation :	Rz[31:0] <- sign_extend(mem(Rx)) Rx[31:0] <- Rx[31:0]+ Ry[31:0]
Grammar:	ldbir.bs rz, (rx), ry
Description:	One byte is loaded from the address corresponding to Rx, the high 24 bits are sign bit expanded, and the result of the expansion is stored in Rz, while Rx Add Ry and deposit into Rx.
Markers of influence Bit:	unaffected
Limitations:	not have
Exception:	Access Error Exception

Command Format:

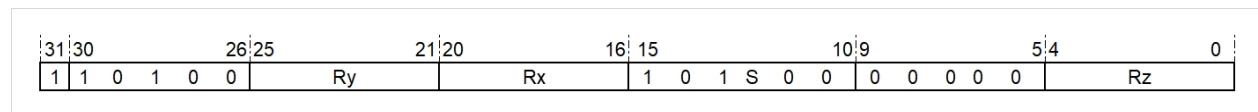


Figure 15.163: LDBIR.BS

15.164 PLDBIR.D - register address self-incrementing double-

word load instruction

unify chemicalization honorable title	
grammatical	pldbir.d rz, (rx), ry
manipulate	Rz[31:0] <- mem(Rx) Rz+1[31:0] <- mem(Rx+Ry) Rx[31:0] <- Rx[31:0]+ 2XRy and prefetch the data on mem(Rx+2XRy), mem(Rx+3XRy) to be stored in the processor's prefetch buffer.
Translation in the end	Only 32-bit instructions exist. pldbir.d rz, (rx), ry

Description:	Load a word from the address corresponding to Rx into Rz, and load a word from the address corresponding to Rx+Ry into Rz+1, while R x This instruction also prefetches the data on mem(Rx+2XRy), mem(Rx+3XRy) and stores it in the location in the prefetch cache of the processor.
Impact Sign s Bit:	unaffected
Limitati ons:	not have
Excepti on:	Unaligned access exception, access error exception

32-bit direct ives	
Operati on:	Rz[31:0] <- mem(Rx) Rz+1[31:0] <- mem(Rx+Ry) Rx[31:0] <- Rx[31:0]+ 2XRy and prefetch the data on mem(Rx+2XRy), mem(Rx+3XRy) to be stored in the processor's prefetch buffer.
Gramm ar:	pldbir.d rz, (rx) , ry
Description:	Load a word from the address corresponding to Rx into Rz, and load a word from the address corresponding to Rx+Ry into Rz+1, while R x This instruction also prefetches the data on mem(Rx+2XRy), mem(Rx+3XRy) and stores it in the location in the prefetch cache of the processor.
Impact Sign s Bit:	unaffected
Limitati ons:	not have
Excepti on:	Unaligned access exception, access error exception

Command Format:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1	0	1	0	0	Ry	Rx	1	0	1	0	Rz

Figure 15.164: PLDBIR.

15.165 STBIR.W - register address self-incrementing word store instruction

Harmonization Directive	
grammatical	stbir.w rz, (rx) , ry
manipulate	mem(Rx) <- Rx[31:0] Rx[31:0] <- Rx[31:0]+ Ry[31:0]
Compilation results	Only 32-bit instructions exist. stbir.w rz, (rx) , ry

Description:	Rz is stored in the address corresponding to Rx, and Rx is added to Ry and stored in Rx.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	Access Error Exception

32-bit instruction	
Operation:	mem(Rx) <- Rx[31:0] Rx[31:0] <- Rx[31:0]+ Ry[31:0]
Grammar:	stbir.w rz, (rx) , ry
Description:	Rz is stored in the address corresponding to Rx, and Rx is added to Ry and stored in Rx.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	Access Error Exception

Command Format:

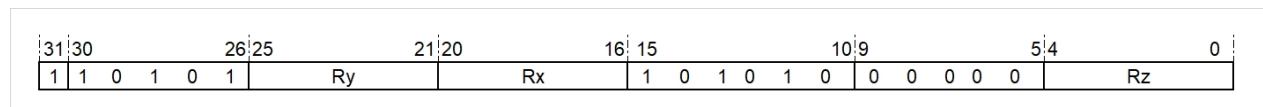


Figure 15.165: STBIR.

15.166 STBIR.H - register address self-incrementing half-word

store instruction

Harmonization Directive	
grammatical	stbir.h rz, (rx) , ry
manipulate	mem(Rx) <- Rx[15:0] Rx[31:0] <- Rx[31:0]+ Ry[31:0]
Compilation results	Only 32-bit instructions exist. stbir.h rz, (rx) , ry

Description:	The lower half-word of Rz is stored in the address corresponding to Rx, and Rx is added to Ry and stored in Rx.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	Access Error Exception

32-bit instruction	
Operation:	mem(Rx) <- Rx[15:0] Rx[31:0] <- Rx[31:0]+ Ry[31:0]
Grammar:	stbir.h rz, (rx) , ry
Description:	The lower half-word of Rz is stored in the address corresponding to Rx, and Rx is added to Ry and stored in Rx.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	Access Error Exception

Command Format:

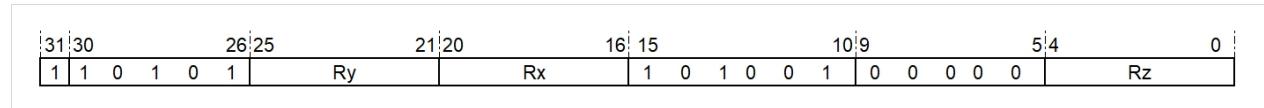


Figure 15.166: STBIR.

15.167 STBIR.B - register address self-incrementing byte store instruction

Harmonization Directive	
grammatical	stbir.b rz, (rx) , ry
manipulate	mem(Rx) <- Rx[7:0] Rx[31:0] <- Rx[31:0]+ Ry[31:0]
Compilation results	Only 32-bit instructions exist. stbir.b rz, (rx) , ry

User's Manual

	Description: The lower 8 bits of Rx are stored in the address corresponding to Rx, and Rx is added to Ry and stored in Rx.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	Access Error Exception

32-bit instruction	
Operation:	mem(Rx) <- Rx[7:0] Rx[31:0] <- Rx[31:0]+ Ry[31:0]
Grammar:	stbir.b rz, (rx) , ry
Description:	The lower 8 bits of Rz are stored in the address corresponding to Rx, and Rx is added to Ry and stored in Rx.
Impact flag bits:	unaffected
Limitations:	not have
Exception:	Access Error Exception

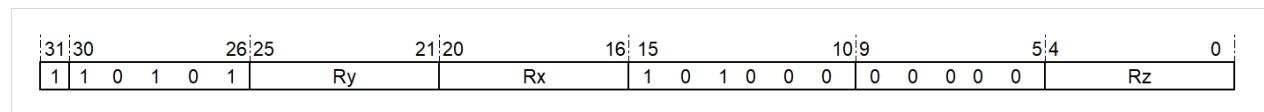
Command Format:

Figure 15.167: STBIR.