

玄铁 C860MP 用户手册

2024 年 04 月 24 日

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. 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 permission of Hangzhou C-SKY MicroSystems Co., Ltd.

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.

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. 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 contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

杭州中天微系统有限公司 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 杭州中天微系统有限公司，保留所有权利。

本文档的所有权及知识产权归属于杭州中天微系统有限公司及其关联公司（下称“中天微”）。本文档仅能分派给：(i) 拥有合法雇佣关系，并需要本文档的信息的中天微员工，或 (ii) 非中天微组织但拥有合法合作关系，并且其需要本文档的信息的合作方。对于本文档，未经杭州中天微系统有限公司明示同意，则不能使用该文档。在未经中天微的书面许可的情形下，不得复制本文档的任何部分，传播、转录、储存在检索系统中或翻译成任何语言或计算机语言。

商标申明

中天微的 LOGO 和其它所有商标（如 XuanTie 玄铁）归杭州中天微系统有限公司及其关联公司所有，未经杭州中天微系统有限公司的书面同意，任何法律实体不得使用中天微的商标或者商业标识。

注意

您购买的产品、服务或特性等应受中天微商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，中天微对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。杭州中天微系统有限公司不对任何第三方使用本文档产生的损失承担任何法律责任。

杭州中天微系统有限公司 Hangzhou C-SKY MicroSystems Co., LTD

地址：中国浙江省杭州市网商路 699 号 5 号楼 2 楼 201 室

网址：www.xrvm.cn

版本历史

版本	描述	日期
01	第一次正式发布。	2020.07.28
02	手册勘误，优化文档。	2020.09.07
03	更新模板。	2021.05.14
04	修正了 TS bit 的描述。	2022.03.11
05	虚拟地址空间、MEL 描述更新。	2022.04.29
06	增加了说明文字：指令 mter 和 mfcr 在普通用户模式下可以访问 FCR 和 FESR。	2022.07.29
07	修改未对齐访问异常相关文字描述	2023.09.07

玄铁 C860MP 用户手册

第一章 概述	1
1.1 简介	1
1.2 特点	1
1.3 可配置选项	2
1.4 可测性设计	3
1.5 可调试性设计	3
1.6 命名规则	3
1.6.1 符号	4
1.6.2 术语	4
第二章 微体系结构	6
2.1 结构框图	6
2.2 流水线介绍	9
第三章 编程模型	11
3.1 工作模式及寄存器视图	11
3.2 通用寄存器	12
3.2.1 程序计数器	13
3.2.2 条件码 / 进位标志位	14
3.3 可选择寄存器	14
3.4 浮点寄存器	14
3.4.1 与通用寄存器传输数据	14
3.4.2 维护寄存器精度的一致	14
3.5 系统控制寄存器	15
3.5.1 处理器状态寄存器 (PSR, CR<0,0>)	16
3.5.1.1 更新 PSR	18
3.5.2 复位向量基址寄存器 (RVBR, CR<28,0>)	18
3.5.3 复位管理寄存器 (RMR, CR<29,0>)	19
3.5.4 向量基址寄存器 (VBR, CR<1,0>)	19
3.5.5 异常保留寄存器 (CR<2,0> ~ CR<5,0>)	19
3.5.6 全局控制寄存器 (GCR, CR<11,0>)	19
3.5.7 全局状态寄存器 (GSR, CR<12,0>)	20
3.5.8 产品序号寄存器 (CPUIDR, CR<13,0>)	20

3.5.9	高速缓存功能设置寄存器 (CFR, CR<17,0>)	20
3.5.10	高速缓存配置寄存器 (CCR, CR<18,0>)	21
3.5.11	二级高速缓存配置寄存器 (CCR2, CR<23,0>)	22
3.5.12	二级高速缓存校验错误寄存器 (CER2, CR<25,0>)	24
3.5.13	多核编号寄存器 (MPID, CR<30,0>)	25
3.5.14	隐式操作寄存器 (HINT, CR<31,0>)	25
3.5.15	普通用户堆栈指针寄存器 (R14(User SP), CR<14,1>)	26
3.5.16	Cache 访问索引寄存器 (CINDEX, CR<26,1>)	27
3.5.17	Cache 访问数据寄存器 0~3 (CDATA0~CDATA3, CR<27,1>~CR<30,1>)	27
3.5.18	Cache 访问指令寄存器 (CINS, CR<31,1>)	28
3.5.19	浮点版本寄存器 (FID, CR<0,2>)	28
3.5.20	浮点控制寄存器 (FCR, CR<1,2>)	29
3.5.21	浮点异常状态寄存器 (FESR, CR<2,2>)	31
3.6	数据格式	32
3.6.1	浮点数据格式	32
3.6.2	整型数据格式	32
3.6.3	大小端	32
3.7	内存模型	34
3.7.1	内存一致性模型	34
第四章	异常处理	36
4.1	异常处理概述	36
4.2	异常类型	38
4.2.1	重启异常 (向量偏移 0X0)	39
4.2.2	未对齐访问异常 (向量偏移 0X4)	39
4.2.3	访问错误异常 (向量偏移 0X8)	40
4.2.4	除以零异常 (向量偏移 0X0C)	40
4.2.5	非法指令异常 (向量偏移 0X10)	40
4.2.6	特权违反异常 (向量偏移 0X14)	40
4.2.7	跟踪异常 (向量偏移 0X18)	40
4.2.8	断点异常 (向量偏移 0X1C)	41
4.2.9	不可恢复错误异常 (向量偏移 0X20)	41
4.2.10	IDLY 异常 (异常偏移 0X24)	41
4.2.11	中断异常	41
4.2.11.1	普通中断 (INT)	42
4.2.11.2	快速中断 (FINT)	42
4.2.11.3	中断处理过程	43
4.2.12	TLB 不可恢复异常 (向量偏移 0X34)	43
4.2.13	TLB 失配异常 (向量偏移 0X38)	43
4.2.14	TLB 修改异常 (向量偏移 0X3C)	43
4.2.15	陷阱指令异常 (向量偏移 0X40 - 0X4C)	43
4.2.16	TLB 读无效异常 (向量偏移 0X50)	44

4.2.17	TLB 写无效异常 (向量偏移 0X54)	44
4.2.18	浮点异常 (向量偏移 0X78)	44
4.2.18.1	非规格化数输入异常	44
4.2.18.2	非法操作异常	45
4.2.18.3	除零异常	45
4.2.18.4	上溢异常	46
4.2.18.5	下溢异常	46
4.2.18.6	非精确异常	46
4.2.18.7	特殊指令的异常	47
4.3	异常优先级	47
4.3.1	发生待处理的异常时调试请求	48
4.4	异常返回	48
第五章	指令集	49
5.1	32 位指令	49
5.1.1	32 位指令功能分类	49
5.1.1.1	数据运算指令	49
5.1.1.2	分支跳转指令	60
5.1.1.3	内存存取指令	61
5.1.1.4	特权指令	63
5.1.1.5	特殊功能指令	64
5.2	16 位指令	65
5.2.1	16 位指令功能分类	65
5.2.1.1	数据运算指令	65
5.2.1.2	分支跳转指令	67
5.3	指令集列表	68
5.4	指令执行延迟	82
第六章	内存管理	100
6.1	虚拟地址空间介绍	100
6.2	MMU 相关控制寄存器	101
6.2.1	MMU 索引寄存器 (MIR, CR<0,15>)	101
6.2.2	MMU EntryLo0 和 EntryLo1 寄存器 (MEL0 & MEL1, CR<2,15> & CR<3,15>)	102
6.2.3	MMU EntryHi/Bad VPN 寄存器 (MEH, CR <4,15>)	103
6.2.4	MMU 页掩码寄存器 (MPR, CR<6,15>)	104
6.2.5	MMU 控制指令寄存器 (MCIR, CR<8,15>)	104
6.2.6	MMU PGD 基址寄存器 (MPGD0 & MPGD1, CR<CR28,15> & CR<29,15>)	106
6.2.7	MMU SSEG0 配置寄存器 (MSA0, CR<30,15>)	106
6.2.8	MMU SSEG1 配置寄存器 (MSA1, CR<31,15>)	107
6.3	TLB 表项结构	108
第七章	内存子系统	109
7.1	指令高速缓存子系统	109

7.1.1	指令预取	109
7.1.2	路预测	109
7.1.3	循环加速缓存器	110
7.1.4	分支历史表	110
7.1.5	分支跳转目标预测器	110
7.1.6	间接分支预测器	111
7.1.7	返回地址预测器	111
7.2	数据高速缓存子系统	111
7.2.1	Cache 一致性	112
7.2.2	多通道数据预取	112
7.2.3	自适应的写分配机制	113
7.2.4	独占式访问 (local monitor 和 global monitor)	113
7.2.5	Preload 指令	113
7.3	二级高速缓存子系统	114
7.3.1	组织形式	114
7.3.2	RAM 延时	115
7.3.3	ECC 校验和奇偶校验	116
7.3.4	L2 Cach 预取机 e 制	116
第八章	总线接口协议	117
8.1	主设备接口	117
8.1.1	特点	117
8.1.2	协议内容	117
8.1.2.1	支持传输类型	117
8.1.2.2	支持响应类型	118
8.1.3	不同总线响应下的行为	118
8.1.4	AXI4.0 协议的主设备接口信号	118
8.2	从设备接口	121
8.2.1	特点	121
8.2.2	协议内容	121
8.2.2.1	支持传输类型	121
8.2.2.2	支持响应类型	122
8.2.3	不同行为发出的响应	122
8.2.4	AXI4.0 协议的从设备接口信号	122
第九章	中断控制器	126
9.1	概述	126
9.2	相关寄存器	128
9.2.1	寄存器地址映射	129
9.2.2	INTC 基地址寄存器 (INTC_BAR, CR<31,14>)	130
第十章	系统计时器	132
10.1	概述	132

10.2	私有计时器相关寄存器	133
10.2.1	(PTIM_CTLR, 64-bit)	133
10.2.2	计时器当前计数值寄存器 (PTIM_CCVR, 64-bit)	134
10.2.3	计时器比较值寄存器 (PTIM_CMVR, 64-bit)	134
10.2.4	计时器装载寄存器 (PTIM_LVR, 32-bit)	134
10.2.5	计时器状态寄存器 (PTIM_TSR, 32-bit)	135
10.3	系统计数器实现参考	135
10.3.1	计数器控制寄存器 (CNT_CTLR, 32-bit)	136
10.3.2	计数器初始值寄存器 (CNT_ICVR, 64-bit)	136
第十一章	调试接口	138
11.1	概述	138
11.2	多核调试框架	138
11.3	外部接口	140
第十二章	工作模式转换	143
12.1	C860 工作模式及其转换	143
12.2	正常工作模式	143
12.2.1	低功耗模式	144
12.2.1.1	DOZE 工作模式	144
12.2.1.2	STOP 工作模式	144
12.2.1.3	WAIT 工作模式	144
12.2.2	调试模式	144
12.2.2.1	进入调试模式	144
12.2.2.2	退出调试模式	145
第十三章	程序参考示例	146
13.1	MMU 设置示例	146
13.2	高速缓存设置示例	147
13.3	多核启动示例	149
13.4	同步原语示例	150
第十四章	附录 A 基础指令术语表	151
14.1	ABS——绝对值指令	151
14.2	ADDC——无符号带进位加法指令	152
14.3	ADDI——无符号立即数加法指令	153
14.4	ADDI(SP)——无符号（堆栈指针）立即数加法指令	157
14.5	ADDU——无符号加法指令	159
14.6	AND——按位与指令	161
14.7	ANDI——立即数按位与指令	162
14.8	ANDN——按位非与指令	163
14.9	ANDNI——立即数按位非与指令	164
14.10	ASR——算术右移指令	165

14.11 ASRC——立即数算术右移至 C 位指令	167
14.12 ASRI——立即数算术右移指令	169
14.13 BAR——内存访问同步指令	170
14.14 BCLRI——立即数位清零指令	171
14.15 BEZ——寄存器等于零分支指令	172
14.16 BF——C 为 0 分支指令	174
14.17 BGENI——立即数位产生指令	175
14.18 BGENR——寄存器位产生指令	176
14.19 BHSZ——寄存器大于等于零分支指令	177
14.20 BHZ——寄存器大于零分支指令	178
14.21 BKPT——断点指令	179
14.22 BLSZ——寄存器小于等于零分支指令	180
14.23 BLZ——寄存器小于零分支指令	181
14.24 BMASKI——立即数位屏蔽产生指令	182
14.25 BNEZ——寄存器不等于零分支指令	184
14.26 BNEZAD——寄存器自减大于零分支指令	185
14.27 BR——无条件跳转指令	187
14.28 BREV——位倒序指令	188
14.29 BSETI——立即数位置位指令	190
14.30 BSR——跳转到子程序指令	191
14.31 BT——C 为 1 分支指令	193
14.32 BTSTI——立即数位测试指令	194
14.33 CLRF——C 为 0 清零指令	195
14.34 CLRT——C 为 1 清零指令	196
14.35 CMPHS——无符号大于等于比较指令	197
14.36 CMPHSI——立即数无符号大于等于比较指令	199
14.37 CMPLT——有符号小于比较指令	202
14.38 CMPLTI——立即数有符号小于比较指令	204
14.39 CMPNE——不等比较指令	207
14.40 CMPNEI——立即数不等比较指令	208
14.41 DCACHE——DCACHE 操作指令	211
14.42 DECF——C 为 0 立即数减法指令	213
14.43 DECGT——减法大于零置 C 位指令	214
14.44 DECLT——减法小于零置 C 位指令	215
14.45 DECNE——减法不等于零置 C 位指令	216
14.46 DECT——C 为 1 立即数减法指令	217
14.47 DIVS——有符号除法指令	218
14.48 DIVSL——有符号长除法指令	219
14.49 DIVU——无符号除法指令	220
14.50 DIVUL——无符号长除法指令	221
14.51 DOZE——进入低功耗睡眠模式指令	222
14.52 FF0——快速找 0 指令	223

14.53	FF1——快速找 1 指令	224
14.54	FABS.T——浮点绝对值	225
14.55	FADD.T——浮点加法指令	226
14.56	FCMPHS.T——浮点大于等于比较	227
14.57	FCMPHSZ.T——浮点大于等于零比较指令	228
14.58	FCMPHZ.T——浮点大于零比较指令	229
14.59	FCMPLSZ.T——浮点小于等于零比较	230
14.60	FCMPLT.T——浮点小于比较	231
14.61	FCMPLTZ.T——浮点小于零比较指令	232
14.62	FCMPNE.T——浮点不等于比较指令	233
14.63	FCMPNEZ.T——浮点不等于零比较指令	234
14.64	FCMPUO.T——浮点双操作数非数判断指令	235
14.65	FCMPUOZ.T——浮点单操作数非数判断指令	236
14.66	FDIV.T——浮点除法指令	237
14.67	FDTOS——双精度浮点单精度浮点转换指令	238
14.68	FFMULA.T——浮点融合乘累加指令	239
14.69	FFMULS.T——浮点融合乘取负累加指令	240
14.70	FFNMULA.T——浮点融合乘取负累减指令	241
14.71	FFNMULS.T——浮点融合乘累减指令	242
14.72	FFTOFI.T.RM——带舍入模式的浮点转换为整型浮点指令	243
14.73	FFTOI.T1.T2——浮点整数转换指令	245
14.74	FFTOI.T1.T2.RM——带舍入模式的浮点整数转换指令	246
14.75	FFTOX.T1.T2——浮点定点转换指令	248
14.76	FITOF.T1.T2——整数浮点转换指令	250
14.77	FINS.32——浮点寄存器间低字高半字传送指令	251
14.78	FLD.T——浮点加载指令	252
14.79	FLDM.T——连续浮点加载指令	254
14.80	FLDMU.T——连续浮点加载地址自增指令	256
14.81	FLDR.T——寄存器移位寻址浮点加载指令	258
14.82	FLRW.T——浮点存储器读入指令	261
14.83	FMAXNM.T——浮点规范最大值选择指令	263
14.84	FMFVR.64——浮点寄存器双字读传送指令	264
14.85	FMFVR.32.1——浮点寄存器低字读传送指令	265
14.86	FMFVR.32.2——浮点双寄存器低字读传送指令	266
14.87	FMFVRH——浮点寄存器高字读传送指令	267
14.88	FMINNM.T——浮点规范最小值选择指令	268
14.89	FMOV.T——浮点寄存器间传送指令	269
14.90	FMOVX.32——浮点寄存器间低半字传送指令	270
14.91	FMTVR.64——浮点双寄存器写传送指令	271
14.92	FMTVR.32.1——浮点寄存器低字写传送指令	271
14.93	FMTVR.32.2——浮点双寄存器低字写传送指令	272
14.94	FMTVRH——浮点寄存器高字写传送指令	273

14.95 FMUL.T——浮点乘法指令	274
14.96 FMULA.T——浮点乘累加指令	275
14.97 FMULS.T——浮点乘取负累加指令	276
14.98 FNMUL.T——浮点乘法取负指令	277
14.99 FNMULA.T——浮点乘取负累减指令	278
14.100 FNMULS.T——浮点乘累减指令	279
14.101 FNEG.T——浮点取反指令	280
14.102 FRECIP.T——浮点倒数指令	281
14.103 FSEL.T——浮点寄存器 C 位选择传送指令	282
14.104 FSQRT.T——浮点平方根指令	283
14.105 FST.T——浮点存储指令	284
14.106 FSTM.T——连续浮点存储指令	286
14.107 FSTMU.T——连续浮点存储地址自增指令	288
14.108 FSTOD——单精度浮点双精度浮点转换指令	290
14.109 FSTR.T——寄存器移位寻址浮点存储指令	291
14.110 FSUB.T——浮点减法指令	293
14.111 FXTOF.T1.T2——定点浮点转换指令	295
14.112 GRS——符号产生指令	297
14.113 ICACHE——ICACHE 操作指令	298
14.114 IDLY——中断识别禁止指令	299
14.115 INCF——C 为 0 立即数加法指令	301
14.116 INCT——C 为 1 立即数加法指令	302
14.117 INS——位插入指令	303
14.118 IXH——索引半字指令	305
14.119 IXW——索引字指令	306
14.120 IXD——索引双字指令	306
14.121 JMP——寄存器跳转指令	307
14.122 JMPI——间接跳转指令	308
14.123 JSR——寄存器跳转到子程序指令	309
14.124 JSRI——间接跳转到子程序指令	311
14.125 LD.B——无符号扩展字节加载指令	313
14.126 LD.BS——有符号扩展字节加载指令	315
14.127 LD.D——双字加载指令	317
14.128 LD.H——无符号扩展半字加载指令	319
14.129 LD.HS——有符号扩展半字加载指令	321
14.130 LD.W——字加载指令	323
14.131 LDEX.W——独占式字加载指令	326
14.132 LDM——连续多字加载指令	328
14.133 LDQ——连续四字加载指令	330
14.134 LDR.B——寄存器移位寻址无符号扩展字节加载指令	332
14.135 LDR.D——寄存器移位寻址双字加载指令	333
14.136 LDR.BS——寄存器移位寻址有符号扩展字节加载指令	335

14.137 LDR.H——寄存器移位寻址无符号扩展半字加载指令	336
14.138 LDR.HS——寄存器移位寻址有符号扩展半字加载指令	338
14.139 LDR.W——寄存器移位寻址字加载指令	339
14.140 LRS.B——字节符号加载指令	341
14.141 LRS.H——半字符符号加载指令	342
14.142 LRS.W——字符符号加载指令	344
14.143 LRW——存储器读入指令	345
14.144 LSL——逻辑左移指令	347
14.145 LSLC——立即数逻辑左移至 C 位指令	348
14.146 LSLI——立即数逻辑左移指令	350
14.147 LSR——逻辑右移指令	351
14.148 LSRC——立即数逻辑右移至 C 位指令	353
14.149 LSRI——立即数逻辑右移指令	354
14.150 L2 CACHE——L2 CACHE 操作指令	356
14.151 MFCR——控制寄存器读传送指令	357
14.152 MOV——数据传送指令	357
14.153 MOVF——C 为 0 数据传送指令	358
14.154 MOVI——立即数数据传送指令	359
14.155 MOVIH——立即数高位数据传送指令	360
14.156 MOVT——C 为 1 数据传送指令	361
14.157 MTCR——控制寄存器写传送指令	362
14.158 MUL.S32——32 位有符号乘法指令	363
14.159 MUL.U32——32 位无符号乘法指令	364
14.160 MULA.32.L——32 位有符号乘累加取低 32 位指令	365
14.161 MULA.S32——32 位有符号乘累加指令	365
14.162 MULA.U32——32 位无符号乘累加指令	366
14.163 MULALL.S16.S——带饱和操作的 16 位有符号低半字乘累加指令	367
14.164 MULSH——16 位有符号乘法指令	369
14.165 MULSW——有符号数 16x32 乘法指令	370
14.166 MULT——乘法指令	372
14.167 MVC——C 位传送指令	373
14.168 MCV——C 位取反传送	374
14.169 NOR——按位或非指令	375
14.170 NOT——按位非指令	376
14.171 OR——按位或指令	377
14.172 ORI——立即数按位或指令	378
14.173 PLDR——读数据预取指令	379
14.174 PLDW——写数据预取指令	380
14.175 POP——出栈指令	382
14.176 PSRCLR——PSR 位清零指令	386
14.177 PSRSET——PSR 位置位指令	388
14.178 PUSH——压栈指令	390

14.179	REVB——字节倒序指令	394
14.180	REVBH——半字节倒序指令	395
14.181	RFI——快速中断返回指令	397
14.182	ROTL——循环左移指令	398
14.183	ROTLI——立即数循环左移指令	399
14.184	RSUB——反向减法指令	400
14.185	RTS——子程序返回指令	401
14.186	RTE——异常和普通中断返回指令	402
14.187	SEXT——位提取并有符号扩展指令	403
14.188	SEXTB——字节提取并有符号扩展指令	405
14.189	SEXTH——半字提取并有符号扩展指令	406
14.190	SRS.B——字节符号存储指令	407
14.191	SRS.H——半字符号存储指令	409
14.192	SRS.W——字符号存储指令	410
14.193	ST.B——字节存储指令	412
14.194	ST.D——双字存储指令	414
14.195	ST.H——半字存储指令	416
14.196	ST.W——字存储指令	418
14.197	STCPR——协处理器字存储指令	421
14.198	STEX.W——独占式字存储指令	422
14.199	STM——连续多字存储指令	424
14.200	STOP——进入低功耗暂停模式指令	426
14.201	STQ——连续四字存储指令	427
14.202	STR.B——寄存器移位寻址字节存储指令	429
14.203	STR.D——寄存器移位寻址双字存储指令	430
14.204	STR.H——寄存器移位寻址半字存储指令	432
14.205	STR.W——寄存器移位寻址字存储指令	433
14.206	SUBC——无符号带借位减法指令	435
14.207	SUBI——无符号立即数减法指令	436
14.208	SUBI(SP)——无符号（堆栈指针）立即数减法指令	439
14.209	SUBU——无符号减法指令	440
14.210	SYNC——CPU 同步指令	442
14.211	TLBI——TLB 无效指令	443
14.212	TRAP——操作系统陷阱指令	444
14.213	TST——零测试指令	445
14.214	STNBZ——无字节等于零寄存器测试指令	446
14.215	WAIT——进入低功耗等待模式指令	448
14.216	XOR——按位异或指令	449
14.217	XORI——立即数按位异或指令	450
14.218	XSR——扩展右移指令	451
14.219	XTRB0——提取字节 0 并无符号扩展指令	453
14.220	XTRB1——提取字节 1 并无符号扩展指令	454

14.221	XTRB2——提取字节 2 并无符号扩展指令	455
14.222	XTRB3——提取字节 3 并无符号扩展指令	456
14.223	ZEXT——位提取并无符号扩展指令	457
14.224	ZEXTB——字节提取并无符号扩展指令	459
14.225	ZEXTH——半字提取并无符号扩展指令	460
第十五章 附录 B 矢量浮点指令术语表		462
15.1	VABS.T——矢量浮点绝对值指令	462
15.2	VADD.T——矢量浮点加法指令	463
15.3	VASX.T——矢量浮点交叉加减法指令	464
15.4	VCMPHS.T——矢量浮点大于等于比较指令	466
15.5	VCMPHSZ.T——矢量浮点大于等于零比较指令	468
15.6	VCMPHZ.T——矢量浮点大于零比较指令	470
15.7	VCMPLSZ.T——矢量浮点小于等于零比较指令	472
15.8	VCMPLT.T——矢量浮点小于比较指令	474
15.9	VCMPLTZ.T——矢量浮点小于零比较指令	476
15.10	VCMPNE.T——矢量浮点不等于比较指令	478
15.11	VCMPNEZ.T——矢量浮点不等于零比较指令	480
15.12	VDTOS——矢量双精度浮点单精度浮点转换指令	482
15.13	VEXPE.T——矢量浮点倒数逼近指令	483
15.14	VFCMUL.T——矢量浮点复数乘法指令	484
15.15	VFCMULA.T——矢量浮点复数乘累加指令	486
15.16	VFCMULC.T——矢量浮点复数共轭乘法指令	488
15.17	VFCMULCA.T——矢量浮点复数共轭乘累加指令	490
15.18	VFCMULCN.T——矢量浮点复数共轭取负乘法指令	492
15.19	VFCMULCNA.T——矢量浮点复数共轭乘累加指令	494
15.20	VFCMULN.T——矢量浮点复数取负乘法指令	496
15.21	VFCMULNA.T——矢量浮点复数取负乘累加指令	498
15.22	VFMULA.T——矢量浮点融合乘累加指令	500
15.23	VFMULS.T——矢量浮点乘取负累加指令	501
15.24	VFMULXAA.T——矢量浮点复数乘累加实部虚部部分计算指令	503
15.25	VFMULXAAL.T——矢量浮点索引复数乘累加实部虚部部分计算指令	505
15.26	VFMULXAS.T——矢量浮点复数乘累加实部虚部部分计算指令	507
15.27	VFMULXASL.T——矢量浮点复数乘累加实部虚部部分计算指令	509
15.28	VFMULXSA.T——矢量浮点复数乘累加实部虚部部分计算指令	511
15.29	VFMULXSAI.T——矢量浮点复数乘累加实部虚部部分计算指令	513
15.30	VFMULXSS.T——矢量浮点复数乘累加实部虚部部分计算指令	515
15.31	VFMULXSSI.T——矢量浮点索引复数乘累加实部虚部计算指令	517
15.32	VFNMULA.T——矢量浮点融合乘取负累减指令	519
15.33	VFNMULS.T——矢量浮点乘累减指令	521
15.34	VFTOFL.T.RM——矢量带舍入模式的浮点转换为整型浮点指令	523
15.35	VFTOIT1.T2——矢量浮点整数转换指令	525

15.36	VFTOI.T1.T2.RM——矢量带舍入模式的浮点整数转换指令	526
15.37	VFTOX.T1.T2——矢量浮点定点转换指令	528
15.38	VITOF.T1.T2——矢量整数浮点转换指令	530
15.39	VMAX.T——矢量浮点最大值选择指令	531
15.40	VMAXNM.T——矢量浮点规范最大值选择指令	533
15.41	VMIN.T——矢量浮点最小值选择指令	535
15.42	VMINNM.T——矢量浮点规范最小值选择指令	537
15.43	VMUL.T——矢量乘法指令	539
15.44	VMULA.T——矢量浮点乘累加指令	540
15.45	VMULAI.T——矢量浮点索引乘累加指令	541
15.46	VMULI.T——矢量索引乘法指令	543
15.47	VMULS.T——矢量浮点乘取负累加指令	545
15.48	VMULSI.T——矢量浮点索引乘取负累加指令	547
15.49	VNEG.T——矢量浮点取反指令	549
15.50	VPADD.T——矢量浮点耦合加法指令	550
15.51	VPMAX.T——矢量浮点相邻最大值选择指令	551
15.52	VPMIN.T——矢量浮点相邻最小值选择指令	553
15.53	VRECPE.T——矢量倒数指令	555
15.54	VRECPS.T——矢量浮点倒数逼近指令	556
15.55	VRSQRTE.T——矢量浮点倒数开方指令	557
15.56	VRSQRTS.T——矢量浮点开方逼近指令	558
15.57	VSABS.T——矢量浮点减法绝对值指令	559
15.58	VSAX.T——矢量浮点交叉减加法指令	560
15.59	VSUB.T——矢量单精度浮点减法指令	561
15.60	VXTOF.T1.T2——矢量定点浮点转换指令	562
第十六章 附录 C 矢量多媒体指令术语表		564
16.1	VABS.T——矢量取绝对值指令	565
16.2	VABS.T.S——矢量饱和取绝对值指令	566
16.3	VADD.T——矢量加法指令	568
16.4	VADD.T.E——矢量扩展加法指令	570
16.5	VADD.T.H——矢量加法取高半指令	572
16.6	VADD.T.RH——矢量加法取高半指令	574
16.7	VADD.T.S——矢量加法饱和指令	576
16.8	VADD.T.X——矢量非对齐加法指令	578
16.9	VADDH.T——矢量加法平均指令	580
16.10	VADDH.T.R——矢量加法平均并舍入指令	581
16.11	VAND.T——矢量与指令	583
16.12	VANDN.T——矢量非与指令	584
16.13	VASX.T.S——矢量错位加减指令	586
16.14	VASXH.T——矢量错位加减平均指令	588
16.15	VCLIP.T——矢量裁剪饱和指令	590

16.16	VCLS.T——矢量符号位连续相同指令	592
16.17	VCLZ.T——矢量最高位连续 0 个数指令	593
16.18	VCMPHS.T——矢量大于等于比较指令	594
16.19	VCMPHSZ.T——矢量大于等于 0 比较指令	596
16.20	VCMPHZ.T——矢量大于零比较指令	598
16.21	VCMPLSZ.T——矢量小于等于零比较指令	600
16.22	VCMPLT.T——矢量小于比较指令	602
16.23	VCMPLTZ.T——矢量小于 0 比较指令	604
16.24	VCMPNE.T——矢量不等于比较指令	606
16.25	VCMPNEZ.T——矢量不等于 0 比较指令	608
16.26	VCNT1.T——矢量数据 1 个数指令	610
16.27	VDTL.T.2——二元去交织指令	611
16.28	VDUP.T.1——一元矢量源目的寄存器之间的拷贝指令	612
16.29	VDUP.T.2——二元矢量源目的寄存器之间的拷贝指令	614
16.30	VDUP.T.3——三元矢量源目的寄存器之间的拷贝指令	616
16.31	VDUP.T.4——四元矢量源目的寄存器之间的拷贝指令	618
16.32	VDUPG.T——通用源矢量目的寄存器拷贝指令	620
16.33	VEXH.T——矢量高位数据插入指令	621
16.34	VEXL.T——矢量低位数据插入指令	622
16.35	VEXPE.T——矢量 e 指数指令	625
16.36	VEXT.T——寄存器矢量数据拼接指令	628
16.37	VEXTI.T——立即数矢量数据拼接指令	630
16.38	VINS.T.1——一元矢量插入指令	632
16.39	VINS.T.2——二元矢量插入指令	633
16.40	VINS.T.3——三元矢量插入指令	635
16.41	VINS.T.4——四元矢量插入指令	637
16.42	VITL.T.2——二元交织指令	639
16.43	VLD.T.N——固定长度矢量加载指令	641
16.44	VLDM.T——连续矢量加载指令	642
16.45	VLDMRU.T——跳跃矢量加载基址更新指令	644
16.46	VLDMU.T——连续矢量加载基址更新指令	646
16.47	VLDRU.T.N——矢量加载基址跳跃更新指令	648
16.48	VLDU.T.N——矢量加载基址更新指令	649
16.49	VLDX.T——可变长度矢量加载指令	650
16.50	VLRW.32.N——矢量存储器读入指令	651
16.51	VMASKI.8.L——矢量低半部分立即数扩展传输指令（按字节元素）	652
16.52	VMASKI.8.H——矢量高半部分立即数扩展传输指令（按字节元素）	653
16.53	VMASKI.16——矢量立即数扩展传输指令（按半字元素）	654
16.54	VMAX.T——矢量最大值选择指令	655
16.55	VMFVR.T——矢量写传送指令	656
16.56	VMIN.T——矢量最小值选择指令	658
16.57	VMOV.T.E——矢量扩展传输指令	659

16.58 VMOV.T.H——矢量高位传输指令	661
16.59 VMOV.T.L——矢量低位传输指令	663
16.60 VMOV.T.RH——矢量高位 round 传输指令	665
16.61 VMOV.T.SL——矢量低位饱和传输指令	667
16.62 VMOVL.8——矢量立即数传输指令（按字节元素）	669
16.63 VMOVL.(U/S)16——矢量（有无符号）立即数传输指令（按半字元素）	670
16.64 VMOVL.(U/S)32——矢量（有无符号）立即数传输指令（按字元素）	672
16.65 VMTVR.T.1——矢量写传送指令	673
16.66 VMTVR.T.2——矢量写传送指令	675
16.67 VMUL.T——矢量乘法指令	676
16.68 VMUL.T.E——矢量扩展乘法指令	678
16.69 VMUL.T.H——矢量取高半乘法指令	680
16.70 VMULA.T——矢量乘累加指令	682
16.71 VMULA.T.E——矢量扩展乘累加指令	684
16.72 VMULACA.T——矢量链乘累加指令	686
16.73 VMULACAA.T——矢量链乘累加加指令	688
16.74 VMULACAAIT——矢量带索引链乘累加加指令	690
16.75 VMULACAIT——矢量带索引链乘累加指令	692
16.76 VMULAIT——矢量带索引乘累加指令	694
16.77 VMULAIT.E——矢量扩展带索引乘累加指令	696
16.78 VMULI.T——矢量带索引乘法指令	698
16.79 VMULI.T.E——矢量扩展带索引乘法指令	700
16.80 VMULI.T.H——矢量带索引取高半乘法指令	702
16.81 VMULS.T——矢量乘累减指令	704
16.82 VMULS.T.E——矢量扩展乘累减指令	706
16.83 VMULSI.T——矢量带索引乘累减指令	708
16.84 VMULSI.T.E——矢量带索引扩展乘累减指令	710
16.85 VNEG.T——矢量取反指令	712
16.86 VNEG.T.S——矢量饱和取反指令	714
16.87 VNOT.T——矢量取反指令	715
16.88 VOR.T——矢量或指令	716
16.89 VORN.T——矢量非或指令	717
16.90 VPADD.T——矢量链加法指令	718
16.91 VPADD.T.E——矢量扩展链加法指令	720
16.92 VPADD.T.S——矢量链加法指令	722
16.93 VPADDA.T.E——矢量扩展链累加指令	724
16.94 VPKG.T.2——二元封装指令	726
16.95 VPKG.T.3——三元封装指令	730
16.96 VPKG.T.4——四元封装指令	734
16.97 VPMAX.T——矢量相邻最大值选择指令	737
16.98 VPMIN.T——矢量相邻最小值选择指令	739
16.99 VRCMUL.T.RS——矢量带饱和和复数乘法取高半舍入指令	741

16.100	VRCMULA.T.E——矢量复数乘法右移扩位累加指令	743
16.101	VRCMULC.T.RS——矢量带饱和复数共轭乘法取高半舍入指令	746
16.102	VRCMULCA.T.E——矢量共轭复数乘法右移扩位累加指令	748
16.103	VRCMULN.T.RS——矢量带饱和复数取负乘法取高半舍入指令	751
16.104	VRCMULNA.T.E——矢量复数取负乘法右移扩位累加指令	753
16.105	VRCMULCN.T.RS——矢量带饱和复数共轭取负乘法取高半舍入指令	756
16.106	VRCMULCNA.T.E——矢量共轭复数取负乘法右移扩位累加指令	758
16.107	VRECPE.T——矢量倒数指令	762
16.108	VRECPS.T——矢量倒数逼近指令	766
16.109	VREVD——矢量数据双字倒序指令	768
16.110	VREVBH——矢量数据半字倒序指令	769
16.111	VREVQ——矢量数据字节倒序指令	771
16.112	VREVBW——矢量数据字倒序指令	772
16.113	VRMUL.T.SE——矢量扩展带饱和小数乘法指令	773
16.114	VRMULH.T.RS——矢量带饱和取高半舍入小数乘法指令	775
16.115	VRMULH.T.S——矢量带饱和取高半小数乘法指令	777
16.116	VRMULHA.T.RS——矢量带饱和取高半舍入小数乘累加指令	778
16.117	VRMULHAI.T.RS——矢量带索引带饱和取高半舍入小数乘累加指令	780
16.118	VRMULHI.T.RS——矢量带索引带饱和取高半舍入小数乘法指令	782
16.119	VRMULHI.T.S——矢量带索引带饱和取高半小数乘法指令	784
16.120	VRMULHS.T.RS——矢量带饱和取高半舍入小数乘累减指令	786
16.121	VRMULHSI.T.RS——矢量带索引带饱和取高半舍入小数乘累减指令	788
16.122	VRMULI.T.SE——矢量带索引扩展带饱和和小数乘法指令	790
16.123	VRMULSA.T.E——矢量扩展带移位小数乘累加指令	792
16.124	VRMULSAI.T.E——矢量带索引扩展带移位小数乘累加指令	794
16.125	VRMULSHR.T.E——矢量扩展带移位小数乘法指令	796
16.126	VRMULSHRI.T.E——矢量带索引扩展带移位小数乘法指令	798
16.127	VRMULSS.T.E——矢量扩展带移位小数乘负累加指令	800
16.128	VRMULSSI.T.E——矢量带索引扩展带移位小数乘负累加指令	802
16.129	VRMULXAA.T.RS——矢量带饱和复数实部虚部交叉相乘累加累加取高半舍入指令	804
16.130	VRMULXAAI.T.RS——矢量带索引带饱和复数实部虚部交叉相乘累加累加取高半舍入指令	806
16.131	VRMULXAS.T.RS——矢量带饱和复数实部虚部交叉相乘累加累减取高半舍入指令	808
16.132	VRMULXASI.T.RS——矢量带索引带饱和复数实部虚部交叉相乘累加累减取高半舍入指令	810
16.133	VRMULXSA.T.RS——矢量带饱和复数实部虚部交叉相乘累减累加取高半舍入指令	813
16.134	VRMULXSAI.T.RS——矢量带索引带饱和复数实部虚部交叉相乘累加累减取高半舍入指令	815
16.135	VRMULXSS.T.RS——矢量带饱和复数实部虚部交叉相乘累减累减取高半舍入指令	817
16.136	VRMULXSSI.T.RS——矢量带索引带饱和复数实部虚部交叉相乘累减累减取高半舍入指令	819
16.137	VRSQRTE.T——矢量倒数开方指令	823
16.138	VRSQRTS.T——矢量倒数开方逼近指令	826
16.139	VSABS.T.E——矢量取绝对值指令	828
16.140	VSABS.T.S——矢量减法取绝对值并累加指令	830
16.141	VSABSA.T——矢量减法取绝对值并累加指令	832

16.142	VSABSA.T.E——矢量扩展减法取绝对值并累加指令	834
16.143	VSAX.T.S——矢量错位减加指令	837
16.144	VSAXH.T——矢量错位减加平均指令	839
16.145	VSEL.T——矢量位选择指令	841
16.146	VSEXT.T——矢量数据符号位读传送指令	842
16.147	VSHL.T——矢量寄存器左移指令	844
16.148	VSHL.T.S——矢量寄存器左移取饱和指令	845
16.149	VSHLI.T——矢量立即数左移指令	847
16.150	VSHLI.T.E——矢量扩展立即数左移指令	848
16.151	VSHLI.T.S——矢量立即数左移取饱和指令	850
16.152	VSHR.T——矢量寄存器右移指令	851
16.153	VSHR.T.R——矢量寄存器右移取 round 指令	853
16.154	VSHRI.T——矢量立即数右移指令	854
16.155	VSHRI.T.L——矢量立即数右移取低半指令	856
16.156	VSHRI.T.LR——矢量立即数右移 Round 取低半指令	858
16.157	VSHRI.T.LRS——矢量立即数右移 Round 取低半饱和指令	860
16.158	VSHRI.T.LS——矢量立即数右移取低半饱和指令	862
16.159	VSHRI.T.R——矢量立即数左移取 Round 指令	864
16.160	VSHRIA.T——矢量立即数右移累加指令	865
16.161	VSHRIA.T.R——矢量立即数右移取 Round 累加指令	867
16.162	VSHT.T——矢量移位指令	869
16.163	VSHT.T.R——矢量移位 Round 指令	870
16.164	VSHT.T.RS——矢量移位饱和 Round 指令	872
16.165	VSHT.T.S——矢量移位饱和指令	874
16.166	VST.T.N——固定长度矢量存储指令	875
16.167	VSTM.T——连续矢量存储指令	877
16.168	VSTMURU.T——跳跃矢量存储基址更新指令	879
16.169	VSTMU.T——连续矢量存储基址更新指令	881
16.170	VSTRU.T.N——矢量存储基址跳跃更新指令	883
16.171	VSTU.T.N——矢量存储基址更新指令	884
16.172	VSTX.T——可变长度矢量存储指令	885
16.173	VSUB.T——矢量减法指令	886
16.174	VSUB.T.E——矢量扩展减法指令	888
16.175	VSUB.T.H——矢量减法取高半指令	890
16.176	VSUB.T.RH——矢量减法取高半指令	892
16.177	VSUB.T.S——矢量减法饱和指令	894
16.178	VSUB.T.X——矢量非对齐减法指令	896
16.179	VSUBH.T——矢量减法平均指令	898
16.180	VSUBH.T.R——矢量减法平均并舍入指令	899
16.181	VSWP.T——矢量数据交换指令	901
16.182	VTBL.T——矢量数据链接指令	902
16.183	VTBX.T——矢量数据链接指令	904

16.184	VTRN.T——矢量数据交叉传输指令	905
16.185	VTST.T——矢量按位与置位指令	907
16.186	VXOR.T——矢量异或指令	908

第一章 概述

1.1 简介

玄铁 C860MP 是面向嵌入式系统和 SoC 应用领域的 32 位超高性能嵌入式多核心处理器，具有出色的功耗与性能表现。C860MP 采用 16/32 位混合编码的 RISC 指令集，主要面向对性能要求严格的高端嵌入式应用，如人工智能、机器视觉、视频监控、自动驾驶、移动智能终端、高性能通信、信息安全等等。

C860MP 采用同构多核架构，支持 1~4 个 C860 核心可配置。每个 C860 核心采用自主设计的体系结构和微体系结构，并重点针对性能进行优化，引入 3 发射 8 执行的超标量架构、强大的矢量运算加速引擎和多通道的数据预取等高性能技术。系统管理方面，C860MP 集成片上功耗管理单元，支持多电压和多时钟管理的低功耗技术。此外，C860 核心支持实时检测并关断内部空闲功能模块，进一步降低处理器动态功耗。

1.2 特点

C860MP 处理器体系结构的主要特点如下：

- 同构多核架构，支持 1~4 个 C860 核心可配置；
- 支持 AXI4.0 的 Master 接口，128B 的总线宽度；
- 支持 1 个 AXI4.0 的 Slave 接口可配置，128B 的总线宽度；
- 两级高缓结构，哈佛结构一级高缓和共享的二级高缓；
- 一级缓存支持 MESI 的一致性协议；
- 二级高缓支持 8 路组相联和 16 路组相联可配置，可配置的 ECC 和奇偶校验机制；
- 二级高缓大小可配置，支持 128KB、256KB、512KB 和 1MB，缓存行为 64B；
- 多核中断控制器，最多支持 224 个中断请求，支持 16 个私有中断、16 个软件中断和 192 个公有中断；
- 支持私有计时器；
- 支持多核调试框架；

每个 C860 核心的主要特征点如下：

- 精简指令集计算机结构（RISC）；

- 支持大端模式和小端模式可配置；
- 9~12 级深流水架构；
- 32 位数据长度，32 位或 16 位可变的指令长度；
- 3 发射 8 执行的超标量架构，对软件完全透明；
- 按序取指，乱序发射，乱序完成和按序退休；
- 两级 TLB 内存管理单元，实现虚实地址转换与内存管理；
- 指令高缓和数据高缓大小可配置，支持 32KB、64KB，缓存行为 64B；
- 指令预取功能，硬件自动检测并动态启动；
- 指令高缓路预测的低功耗访问技术；
- 短循环低功耗执行技术；
- 64Kbit 的 2 级多路并行分支预测器；
- 1024/2048 表项可配的分支目标缓存器；
- 支持 12 层的硬件返回地址堆栈；
- 256 表项的间接跳转分支预测器；
- 非阻塞发射，投机猜测执行；
- 基于物理寄存器的重命名技术；
- 支持 0 延时 move 指令；
- 双发射、全乱序执行的 load、store 指令；
- 支持读写各 8 路并发的总线访问；
- 支持写合并；
- 支持 8 个通道的数据缓存硬件预取，支持 stride 的预取方式；
- 浮点执行单元可配置，支持单精度和双精度；
- 可选的矢量执行单元可配置，支持单精度浮点和 8 位/16 位/32 位定点的矢量运算。

C860 单核心主要的工艺和性能参数如下：

- TSMC28nm 工艺下，工作频率高于 1.2GHz（最恶劣情况）；
- 性能 4.1 DMIPS/MHz，4.4 coremark/MHz。

1.3 可配置选项

C860MP 可配置选项如 表 1.1 所示。

表 1.1: C860MP 可配置选项

可配置单元	配置选项	详细
C860 核心数量	1/2/3/4	C860MP 提供 1-4 个 C860 核心可配。
VDSP	无/有	矢量多媒体运算单元。
Slave IF	无/有	用于外设对片上高速缓存的访问, 实现数据一致性, 可用于连接 DMA。
L1 DCache	32K/64K	可以配置 32KB、64KB。
L1 Icache	32K/64K	可以配置 32KB、64KB。
L2 Cache	128K/256K/512K/1M 8way/16way	可以配置 128KB 8 路组相连、128KB 16 路组相连 可以配置 256KB 8 路组相连、256KB 16 路组相连 可以配置 512KB 8 路组相连、512KB 16 路组相连 可以配置 1MB 8 路组相连、1MB 16 路组相连
L2 ECC	无/有	L2 Data RAM 的 ECC 校验。
JTLB	512/1024	MMU 可以配置为 512 个 TLB 表项、1024 个 TLB 表项。
分支目标预测器 (BTB)	无/1024/2048	可以配置为无 BTB、有 BTB 且表项为 1024、有 BTB 且表项为 2048。
间接跳转目标预测器	无/有	实现对间接跳转目标地址进行预测。
循环缓冲器	无/有	实现对短循环的加速。
中断控制器	无/有	内嵌中断控制器。
私有计时器	无/有	用于计时。
性能监测单元	无/有	用于性能监测。

1.4 可测性设计

C860MP 支持扫描链测试 (SCAN), 用于测试处理器内部的组合和时序逻辑是否存在制造错误。C860MP 的扫描链数目可由客户指定。

C860MP 不提供内建自测试, 由客户自行提供。内建自测试用于测试高速缓存是否存在制造错误。

1.5 可调试性设计

C860MP 使用 JTAG 标准设计硬件调试接口。C860MP 支持所有常见的调试功能, 包括软断点、内存断点, 改寄存器检查和存储器检查和修改, 指令单步跟踪与多步跟踪、程序流跟踪等。具体请详见第十一章——调试接口。

1.6 命名规则

1.6.1 符号

本文档用到的标准符号和操作符如 图 1.1 所示。

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
rd	整型目的寄存器
rs1	整型源寄存器1
rs2	整型源寄存器2
rs3	整型源寄存器3
fd	浮点目的寄存器
fs1	浮点源寄存器1
fs2	浮点源寄存器2
fs3	浮点源寄存器3
vd	矢量目的寄存器
vs1	矢量源寄存器1
vs2	矢量源寄存器2
vs3	矢量源寄存器3

图 1.1: 符号列表

1.6.2 术语

- 逻辑 1 是指对应于布尔逻辑真的电平值。

- 逻辑 0 是指对应于布尔逻辑伪的电平值。
- 置位是指使得某个或某几个位达到逻辑 1 对应的电平值。
- 清除是指使得某个或某几个位达到逻辑 0 对应的电平值。
- 保留位是为功能的扩展而预留的，没有特殊说明时其值为 0。
- 信号是指通过它的状态或状态间的转换来传递信息的电气值。
- 引脚是表示一种外部电气物理连接，同一个引脚可以连接多个信号。
- 使能是指使某个离散信号处在有效的状态：
 - 低电平有效信号从高电平切换到低电平；
 - 高电平有效信号从低电平切换到高电平。
- 禁止是指使某个处在使能状态的信号状态改变：
 - 低电平有效信号从低电平切换到高电平；
 - 高电平有效信号从高电平切换到低电平。
- LSB 代表最低有效位,MSB 代表最高有效位。

存储单元和寄存器当 “pad_cpu_bigend_b=0” 时采用小端模式，其字节次序是高字节在最低位的排列。一个字中的所有位是从最高有效位 (第 31 位) 开始往下排列。

- 当 “pad_cpu_bigend_b=1” 时，采用大端模式。
- 信号，位域，控制位的表示都使用一种通用的规则。
- 标识符后来跟着表示范围的数字，从高位到低位表示一组信号，比如 addr[4:0] 就表示一组地址总线，最高位是 addr[4]，最低位是 addr[0]。
- 单个的标识符就表示单个信号，例如 pad_cpu_rst_b 就表示单独的一个信号。有时候会在标识符后加上数字表示一定的意义，比如 addr15 就表示一组总线中的第 16 位。

第二章 微体系结构

2.1 结构框图

C860MP 结构框图如 图 2.1 所示。

指令提取单元 (IFU)，一次可最多提取八条指令并对其并行处理；配备高速缓存，在高速缓存缺失时采用关键指令先取和发射，以及后续指令旁路技术；可以开启指令高缓路预测技术，只访问两路指令高缓中大概率命中的一路，避免两路指令高缓的同时访问以降低功耗；配备指令暂存器缓存预取指令；选配循环加速缓存器，实现短循环取指加速；采用 Bi-mode 指令分支跳转预测，实现极高预测精度；可以配备间接分支预测器，对间接分支目标地址进行精准预测；整个指令提取单元拥有低功耗，高分支预测准确率，高指令预取效率的特点。

指令译码单元 (IDU) 可以同时三条指令进行译码并检测数据相关性。指令译码单元根据后级流水线执行情况，及时更新指令的数据相关性信息，并将指令乱序发送至下级流水线执行。指令译码单元支持指令的乱序执行调度，并通过投机性的发射缓解因数据相关性造成的性能损失。此外，指令译码单元还能够分解 LDM/STM/PUSH/POP 等复杂指令，简化执行逻辑。

执行单元包含整型单元 (IU) 和浮点单元 (FPU)，整型单元包含若干算术逻辑单元 (ALU)、乘法单元 MULT、除法单元 DIV 和跳转单元 BJU。其中，ALU 执行标准的 32 位整数操作，大部分常用指令单周期产生运算结果，如加减法、移位、逻辑运算等；少部分复杂指令被拆分成多条单周期指令，如提取、插入指令等。ALU 支持快速查找 1/0 算法 (FF1/FF0)，支持移位加操作 (IXH, IXW, IXD) 等。一个条件/进位位 (C) 用于条件测试和多于 32 位的算术逻辑运算。通常，C 位只在显式的测试/比较运算中被改变，而不是正常指令运算的副作用。ALU 通过操作数前馈减少数据真相关，单周期 ALU 指令不存在数据真相关停顿延时。MULT 支持 16*16, 16*32, 32*32 整数乘法。除法器的设计采用了快速算法，执行周期 6~35 不等。BJU 可以在单周期内完成分支预测错误处理，加速处理器性能。

浮点单元包含若干个浮点算术逻辑单元 (FALU)、两个浮点乘累加单元 (FMAU) 和一个浮点除法开方单元 (FDSU)，支持单精度、双精度和矢量运算。浮点算术逻辑单元 (FALU) 负责加减、比较、转换、寄存器传输等操作，支持单精度/双精度/矢量运算。浮点加减法操作采用 Triple Path 算法，可在 3 个周期完成。转换操作支持定点数、整型和浮点类型操作数，可在 3 个周期完成。寄存器传输、取绝对值、取反等其他操作同样需要 3 个周期完成。浮点除法开方单元 (FDSU) 负责浮点除法、浮点开方以及浮点倒数等操作。浮点除法开方单元采用基-4 SRT 算法，每个周期产生 2 位运算结果。单精度除法开方运算最长为 19 个执行周期，双精度最长为 32 个执行周期。浮点融合乘累加单元 (FMAU) 负责普通乘法、融合乘累加、e 指数快速运算及逼近等操作，支持单精度/双精度/矢量运算。普通乘法指令在第 4 周期产生计算结果，融合乘累加指令、e 指数快速运算以及逼近指令在第 5 周期产生计算结果。

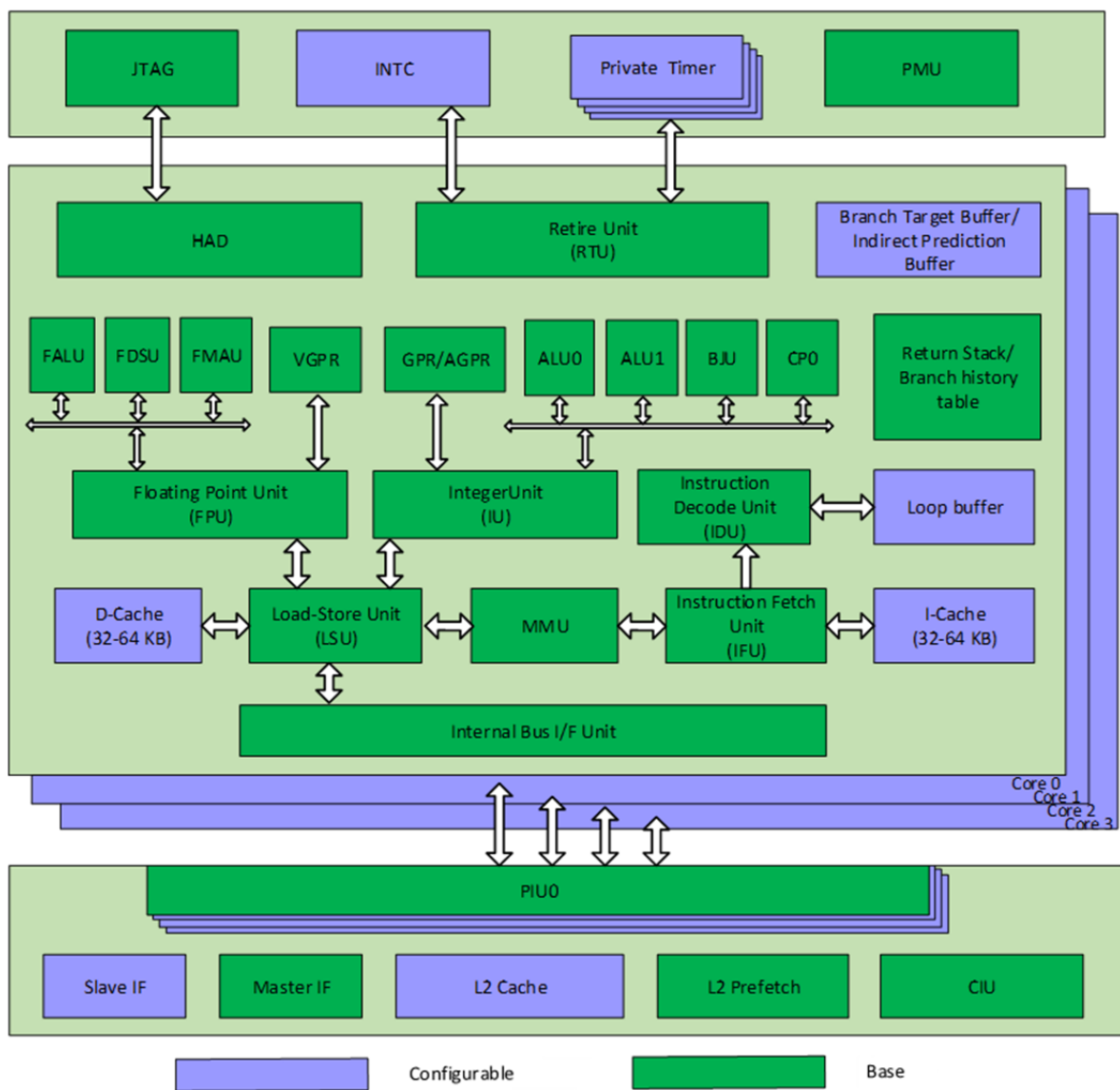


图 2.1: C860MP 微体系结构图

矢量多媒体运算单元包含若干个矢量算术逻辑单元、一个乘累加单元，一个移位单元，一个杂类运算单元，支持矢量运算。矢量算术逻辑单元负责加减、比较、寄存器传输等操作（部分为拆分指令），（每条拆分指令的结果）在 3-4 周期完成，在第 5 周期回写，支持内部 forward。乘累加单元负责整数以及小数的乘法和乘累加（部分为拆分指令），（每条拆分指令的结果）在 3-5 周期完成，在第 5 周期回写，16bit 复数乘法，16bit、32bit 的复数乘累加在 6 周期完成，支持内部 forward。移位单元负责元素的移位、传输等操作（部分为拆分指令），（每条拆分指令的结果）在 3-4 周期完成，在第 5 周期回写。杂类运算单元负责元素的插入、打包、交织、去交织等操作（部分为拆分指令），（每条拆分指令的结果）在 2 周期完成，在第 5 周期回写。

存储载入单元（LSU）支持存储/加载指令的双发射及全乱序执行，支持高速缓存的非阻塞访问。具有内部前馈机制，消除载入指令回写数据的相关性。支持字节、半字、字和双字的存储/载入指令，并支持字节和半字的载入指令的符号位和 0 扩展。存储/加载指令可以流水执行，使得数据吞吐量达到一个周期存取一个数据。支持 8 通道的数据预取技术，提前放入 L1 数据高缓中。当数据高缓缺失后，支持总线的并行访问。

内存管理单元（MMU），具有 16 表项全相联的数据 μ TLB、16 表项全相联的指令 μ TLB 和 512/1024 表项 4 路组相联 jTLB。 μ TLB 提升转换速度，对用户透明；jTLB 提高匹配率，用户可配置。如果在 μ TLB 中没有 TLB 失配，当前周期产生物理地址；如果在 μ TLB 中存在 TLB 失配，但在 jTLB 中没有 TLB 失配，3 个时钟周期产生物理地址；在 MMU 硬件回填没使能时，如果在 jTLB 存在 TLB 失配，软件支持填充 jTLB；如果 MMU 硬件回填使能时，如果在 jTLB 存在 TLB 失配，硬件支持填充 jTLB。

指令退休单元（RTU）包括一个重排序缓冲器与一个物理寄存器堆。其中，重排序缓冲器负责指令的乱序回收与按序退休，物理寄存器堆负责结果的乱序回收和传递。通过支持指令并行回收与快速退休提高指令退休效率。指令退休单元每个时钟周期并行退休三条指令，保证精确异常，支持普通中断和快速中断。16 个硬件可配置的可选择寄存器用于减少在中断异常处理程序的上下文切换的时间。支持矢量和自动矢量中断。

内部总线接口单元（IBIU）支持 AXI 协议，支持关键字优先的地址访问。

数据一致性接口单元（CIU），采用 MESI 的写无效协议维护各个 L1 数据高缓的一致性；CIU 单元通过一组处理器接口单元（PIU）连接各个 C860 核心，完成请求的接收和转换，实现 CIU 与 C860 核心的完全解耦合，从而可灵活配置 C860 的核心数量；设置两路监听缓冲器，可并行处理多个监听请求，最大化利用监听带宽；采用高效的数据旁路机制，当监听请求命中被监听的 L1 数据高缓时，直接将数据旁路给请求发起核心；另外，CIU 单元支持 TLB 和 ICACHE 无效操作请求的广播，简化了 TLB/ICache 与 DCache 数据一致性的软件维护成本。

二级高速缓存，紧耦合于 CIU 单元，实现和 L1 数据高缓的同步访问；采用分块的流水线架构，每条流水线拥有独立的控制逻辑和 RAM 资源，单周期可并行处理两个访问请求，最大访问带宽可达到 1024 比特。二级高速缓存采用和 C860 相同的工作频率，因此，TAG RAM 和 DATA RAM 的访问延时提供软件可配置；另外，设置了 16 个回填/牺牲缓冲器，可支持 16 条缓存行的回填或者 16 条牺牲缓存行的内存回写。

主设备接口单元，支持 AXI4.0 协议，支持关键字优先的地址访问，可以在不同的系统时钟与 CPU 时钟比例（1:1，1:2，1:3，1:4，1:5，1:6，1:7，1:8）下工作。

可配置的 AXI4.0 从设备接口，实现外设（例如 DMA）对 L1 数据高缓和 L2 高缓的访问；支持 128 比特的总线宽度；支持 Len 为 0 和 3 的 INCR 传输。

内嵌中断控制器（INTC），支持 16 个软件中断、16 个私有中断和 192 个公有中断，可通过 C860MP 提供的内部总线访问。具体请参考第九章中断控制器。

私有计时器，每个 C860 核心可配置一个私有定时器，各个私有计时器共享一个系统计数器，通过采集

系统计数器的计数值与软件设置的私有计时器的值进行比较，产生计时信号，具体请参考第十章系统计时器。

C860MP 采用多核单端口的调试框架，每个 C860 核心拥有独立调试辅助单元（HAD），各个核心共享一个 JTAG 单元。HAD 单元支持各种调试方式，包括软件设置断点方式、硬件设置断点方式、单步和多步指令跟踪、跳转指令跟踪、被动进入等 7 种方式，可以在线调试 CPU、通用寄存器（GPR）、可选择寄存器（AGPR）、协处理器 0（CP0）和内存。

2.2 流水线介绍

每个 C860 核心设计了 12 级流水线：

即指令访问、指令预译码、指令缓存、指令译码、指令重命名、指令发射、寄存器堆访问、指令执行/地址产生、数据访问、地址比较、数据封装和回写。12 级流水线的作用如下表所示。

表 2.1: 流水线描述

流水线名称	缩写	流水线作用
指令访问	IF	1、访问指令高速缓存，获取 128 位指令数据； 2、程序计数器（PC）由虚拟地址到物理地址转换。
指令预译码	IP	1、指令预译码； 2、预测条件分支指令的跳转方向和目标地址。
指令缓存	IB	1、提取并封装出 3 条有效指令； 2、缓存预译码指令； 3、间接跳转目标地址和返回地址预测； 4、访问短循环缓冲器。
指令译码	ID	1、指令译码 1； 2、指令拆分。
指令重命名	IR	1、指令译码 2； 2、指令相关性检测，寄存器重命名。
指令发射	IS	1、指令发射队列创建； 2、指令乱序调度。
寄存器堆访问	RF	1、指令译码 3； 2、访问物理寄存器堆获取操作数； 3、操作数前馈。
指令执行 1 (地址产生)	EX1 (AG)	1、指令执行（单周期 ALU 指令和分支指令产生结果）； 2、存储加载指令产生访问地址； 3、访问地址由虚拟地址到物理地址转换。
指令执行 2 (数据访问)	EX2 (DC)	1、双周期 ALU 指令产生结果； 2、乘除法指令执行； 3、访问数据高速缓存。

表 2.2: 流水线描述

流水线名称	缩写	流水线作用
指令执行 3 (地址比较)	EX3 (DA)	1、乘除法指令产生结果； 2、存储加载指令地址比较； 3、数据封装 1。
指令执行 4 (数据封装)	EX4 (FMT)	1、数据封装 2； 2、存储加载指令产生结果。
回写	WB	1、指令退休 2、将指令执行结果回写到寄存器堆。

第三章 编程模型

3.1 工作模式及寄存器视图

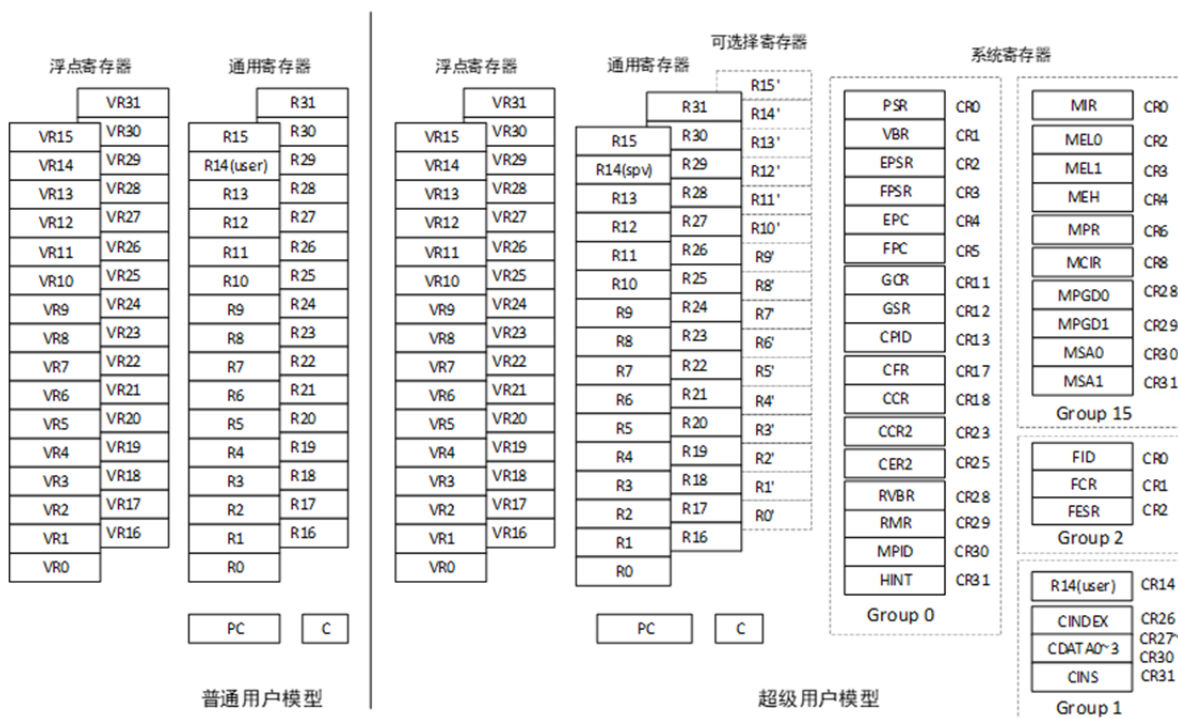


图 3.1: 编程模型

C860 定义了两种编程权限：超级用户模式和普通用户模式。两种运行模式对应不同的操作权限，区别主要体现在以下几个方面：

1. 对寄存器的访问；
2. 特权指令的使用；
3. 对内存空间的访问。

普通用户程序只允许访问指定给普通用户模式的寄存器；工作在超级用户模式下，系统软件则可以访问所有的寄存器，使用控制寄存器来进行超级用户操作。避免了普通用户程序接触特权信息，而操作系统通过协调与普通用户程序的行为来为普通用户程序提供管理和服务。

普通用户模式和超级用户模式的 R14 是独立的，在普通用户模式下只能访问 R14（user）；在超级用户模式下，不仅可以访问 R14（spv），还可以访问普通用户的 R14（user）。通常，R14 用于当前的堆栈指针。

大多数指令在两种模式下都能执行，但是一些对系统产生重大影响的特权指令只能工作在超级用户模式下执行。特权指令包括 STOP、WAIT、DOZE、MFCR、MTCR、PSRSET、PSRCLR、RTE、RFI、DCACHE、ICACHE、L2CACHE 和 TLBI 指令。其中，(1) 指令 ICACHE.IVA 和 DCACHE.CVAL1 在 CCR.UCME 有效时，允许普通用户模式执行；(2) 指令 MFCR 和 MTCR 允许在普通用户模式下访问 FCR 和 FESR。其他指令都只允许在超级用户模式下执行。

玄铁 CPU 提供 4GB 的的虚拟内存地址空间，可运行在两个权限级别上：普通用户模式和超级用户模式。其中，普通用户只能访问 0x00000000~0x7FFFFFFF 表示的 2GB 空间，在访问 0x80000000~0xFFFFFFFF 表示的 2GB 空间时，出现访问错误；而超级用户模式可以访问 4GB 全地址空间。

处理器的工作模式由处理器状态寄存器（PSR）的 S 位控制。当 PSR 的 S 位被置位时，处理器工作在超级用户模式；当 S 位被清零时，处理器工作在普通用户模式。在普通用户模式下，处理器使用普通用户编程模型。当异常处理时，处理器把模式从普通用户模式切换到超级用户模式。异常处理时，处理器把当前的 PSR 的值存放在 EPSR 或 FPSR 影子控制寄存器中，然后设置 PSR 的 S 位，强制处理器进入超级用户模式。在异常处理后，为返回到以前的工作模式，系统函数执行 rte(从异常返回) 或 rfi（从快速中断返回），清空流水线，并从异常发生的地方重新取指执行。作为系统调用指令，TRAP #n 指令为普通用户程序提供了访问操作系统服务程序的可控制接口。普通用户程序可通过 TRAP #n 产生系统调用异常，并强制处理器进入超级用户模式。

普通用户模式可以操作的寄存器包括 32 个 32 位的通用寄存器、32 位程序计数器（PC）、条件/进位位（C）。其中，C 位位于 PSR 的最低位，是 PSR 中唯一能被在普通用户模式下被访问的数据位。除了普通用户模式可以访问的寄存器外，超级用户模式还包括含有操作控制和状态信息的 PSR 寄存器，一套用来在异常发生时保存 PSR、PC 的异常影子寄存器 EPSR、EPC 和一套用来节省在快速中断中上下文切换时间的快速中断影子寄存器 FPSR、FPC。超级用户程序还可以利用一个寄存器 VBR 保存中断向量表的基地址、一个全局状态寄存器 GSR 和一个全局控制寄存器 GCR，以及其他相关控制寄存器。

3.2 通用寄存器

C860 拥有 32 个 32 位的通用寄存器，如 表 3.1 所示。

表 3.1: 普通用户编程模式寄存器

名称	功能
R0	不确定，函数调用时第一个参数
R1	不确定，函数调用时第二个参数
R2	不确定，函数调用时第三个参数
R3	不确定，函数调用时第四个参数
R4	不确定
R5	不确定
R6	不确定

下页继续

表 3.1 – 续上页

名称	功能
R7	不确定
R8	不确定
R9	不确定
R10	不确定
R11	不确定
R12	不确定
R13	不确定
R14(user)	堆栈指针（普通用户模式）
R14' (Spv SP)	堆栈指针（超级用户模式）
R15	链接寄存器
R16	不确定
R17	不确定
R18	不确定
R19	不确定
R20	不确定
R21	不确定
R22	不确定
R23	不确定
R24	不确定
R25	不确定
R26	不确定
R27	不确定
R28	不确定
R29	不确定
R30	不确定
R31	线程本地储存寄存器

通用寄存器用于保存指令操作数和指令执行结果以及地址信息。

C860 为普通用户模式和超级用户模式分别设计了堆栈指针 R14。普通用户模式只能访问用户程序的 R14 (User SP)；在超级用户模式下，系统软件不仅可以访问系统程序的 R14 (Spv SP)，还可以访问普通用户程序的 R14 (User SP)。超级用户模式下，对通用寄存器 R14 的索引将会使用超级用户模式的寄存器 R14(Spv SP)。若用户要在超级用户模式下访问 R14(User SP)，可通过 MFCR/MTCR 访问 CR<14,1> 完成。

3.2.1 程序计数器

程序计数器记录了当前执行指令的地址。在指令执行和异常处理期间，处理器会根据程序运行的情况自动的调整程序计数器值或放置一新值到程序计数器中。对一些指令来说，程序计数器能被用来作为相对地址

计算。此外，程序计数器的低位一直为零。

3.2.2 条件码 / 进位标志位

条件码 / 进位标志位代表了一次操作后的结果。条件码 / 进位标志位能够作为比较操作指令的结果被设置，或者作为另一些高精度算术或逻辑指令的结果而被设置。

3.3 可选择寄存器

在特定配置下，超级用户编程模式下的附加资源包含 16 个可选择寄存器。在转向时间关键任务时可选择寄存器可用来减少因当前内容转换和保护现场引起的响应延迟时间。当 PSR (AF) 为 1，则可选择寄存器被选中，所有的指令以前使用通用寄存器 R0~R15 现在都将使用可选择寄存器 R0' ~R15'。反之，若 PSR (AF) 为 0，则可选择寄存器不被选中。一些重要的参数和指针值可以保存在这些可选择寄存器中，只要在优先级高的任务执行时可选择寄存器被选中，就可以使用这些重要的数据。

在异常出现并执行异常服务程序时，异常服务程序入口地址（异常向量表所指向的内存值）的低位将被拷贝到 AF 位中，以用来选择哪一组寄存器。

通用寄存器 R16~R31 没有对应的可选择寄存器。

3.4 浮点寄存器

C860 浮点单元拥有独立的浮点寄存器，包含 32 个 64 位浮点寄存器，矢量配置下将扩展为 16 个 128 位的矢量寄存器 (vr0~vr15) 和 16 个 64 位浮点寄存器 (vr16~vr31)，可在普通用户模式与特权模式下被访问。

3.4.1 与通用寄存器传输数据

通用寄存器跟浮点寄存器之间的数据传输可以通过浮点寄存器传送指令实现。浮点寄存器传送指令包括：

- FMTVRH/FMTVR.32.1/FMTVR.64 浮点寄存器高/低/双字节写传送指令。
- FMFVRH/FMFVR.32.1/ FMFVR.64 浮点寄存器高/低/双字节读传送指令。

从通用寄存器传送一个单/双精度浮点数据到浮点寄存器中，数据格式不会因为传输而改变，所以程序可以直接使用这些寄存器而不必经过类型转换。

3.4.2 维护寄存器精度的一致

浮点寄存器可存储单精度浮点数、双精度浮点数和整形数据。举例说明，在浮点寄存器 FR1 中所存的数据类型，取决于上一次的写操作，可能是三种数据类型中的任何一种。

浮点单元在硬件上不对数据类型做任何数据格式上的检测，硬件对浮点寄存器中数据格式的解析只取决于执行的浮点指令本身，而不关心这个寄存器上次的写操作用的数据格式。这完全是靠编译器或者程序本身来保证寄存器中的数据精度的一致性。

3.5 系统控制寄存器

系统编程人员在超级用户编程模式下设置系统操作功能，I/O 控制，以及其他受限操作。超级用户编程模式可操作的系统控制寄存器如 表 3.2 所示。

表 3.2: 系统控制寄存器

名称	类型	控制寄存器编号	描述
PSR	读/写	CR<0,0>	处理器状态寄存器
VBR	读/写	CR<1,0>	向量基址寄存器
EPSR	读/写	CR<2,0>	异常保留处理器状态寄存器
FPSR	读/写	CR<3,0>	快速中断保留处理器状态寄存器
EPC	读/写	CR<4,0>	异常保留程序计数器
FPC	读/写	CR<5,0>	快速中断保留程序计数器
GCR	只写	CR<11,0>	32 位全控制寄存器
GSR	只读	CR<12,0>	32 位全状态寄存器
CPUIDR	只读	CR<13,0>	产品序号寄存器
CFR	读/写	CR<17,0>	高速缓存功能设置寄存器
CCR	读/写	CR<18,0>	高速缓存配置寄存器
CCR2	读/写	CR<23,0>	二级高速缓存配置寄存器
CER2	读/写	CR<25,0>	二级高速缓存校验错误寄存器
RVBR	读/写	CR<28,0>	复位向量基址寄存器
RMR	读/写	CR<29,0>	复位管理寄存器
MPID	只读	CR<30,0>	多核编号寄存器
HINT	读/写	CR<31,0>	隐式操作寄存器
R14 (User SP)	读/写	CR<14,1>	普通用户堆栈指针寄存器
CINDEX	读/写	CR<26,1>	高速缓存访问索引寄存器
CDATA0	只读	CR<27,1>	高速缓存访问数据 0 号寄存器
CDATA1	只读	CR<28,1>	高速缓存访问数据 1 号寄存器
CDATA2	只读	CR<29,1>	高速缓存访问数据 2 号寄存器
CDATA3	只读	CR<30,1>	高速缓存访问数据 3 号寄存器
CINS	只写	CR<31,1>	高速缓存访问指令寄存器
FID	只读	CR<0,2>	浮点版本寄存器
FCR	读/写	CR<1,2>	浮点控制寄存器
FESR	读/写	CR<2,2>	浮点异常状态寄存器

3.5.1 处理器状态寄存器（PSR，CR<0,0>）

处理器状态寄存器（PSR）存储了当前处理器的状态和控制信息，包括 C 位、中断有效位和其他控制位。在超级用户编程模式下，软件可以访问处理器状态寄存器（PSR）。控制位为处理器指出了以下的状态：跟踪模式（TM [1:0]）、超级用户模式或者普通用户模式（S 位）、以及通用寄存器或者可选择寄存器 *（AF 位）。它们同样也指出了异常保留寄存器是否可用来保存当前相应的内容，以及中断申请是否有效。

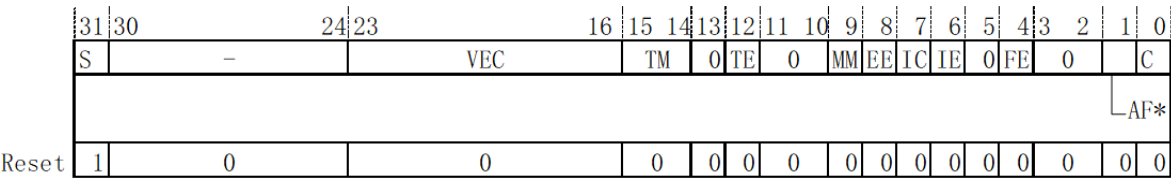


图 3.2: 处理器状态寄存器

S-超级用户模式设置位：

当 S 为 0 时，处理器工作在普通用户模式；

当 S 为 1 时，处理器工作在超级用户模式；

S 位在被 reset 和进入异常处理时由硬件置 1。

VEC[7:0]-异常事件向量值：

当异常出现时，这些位被用来计算异常服务程序向量入口地址，且会在被 reset 时清零。

TM[1:0]-跟踪模式位：

在指令跟踪模式下，每一条指令执行完后，CPU 都将会进入跟踪异常服务程序；在跳转跟踪模式下，当碰到内含有跳转（不管是跳转还是不跳转）的指令执行完，CPU 都将会进入跟踪异常服务程序。这些位在被 reset 清零和进入异常服务程序时由硬件清零。表 3.3 示出 TM[1:0] 编码与相对应的工作模式。

表 3.3: TM[1:0] 编码与相对应的工作模式

值	描述
00	正常执行模式
01	指令跟踪模式
10	未定义
11	跳转跟踪模式

跟踪模式具体的操作请参考[异常处理](#)。

TE-传输控制位：

该位在芯片的管脚上有对应的管脚信号 corex_pad_te_b，可以通过设置该位来控制传输，它会被 reset 清零，也称为 TC。

MM-不对齐异常掩盖位：

当 MM 为 0 时，读取或存储的地址不对齐，异常将正常发生，不会被掩盖即异常会被响应；

当 MM 为 1 时，读取或存储的地址不对齐，异常将会被掩盖，加载存储指令将以非对齐方式访问存储器。

以下是一些例外情况：

该位不能掩盖 JMPI 或 JSRI 指令的不对齐异常的发生，且不受其它异常影响，但会被 reset 清零。

该位不能掩盖矢量读取或者存储指令的不对齐异常发生条件，所有矢量读取或者存储指令的非对齐异常产生条件都为访问地址没有按照 element size 对齐。

当页面属性为 strong order，所有访问地址应当按照指令对应的数据 size 对齐，否则产生不对齐异常。

该位不能掩盖原子操作指令的不对齐异常的发生。

EE-异常有效控制位：

当 EE 为 0 时，异常无效，此时除了普通中断和快速中断之外的任何异常一旦发生，都会被 C860 认为是不可恢复的异常；

当 EE 为 1 时，异常有效，所有的异常都会正常的响应和使用 EPSR 与 EPC。

IC-中断控制位：

当 IC 为 0 时，中断只能在指令之间被响应；

当 IC 为 1 时，表明中断可在长时间、多周期的指令执行完之前被响应；

会被 reset 清零，不受其它异常影响。

IE-中断有效控制位：

当 IE 为 0 时，中断无效，即 pad_corex_int_b 不起作用，EPC 和 EPSR 都无效；

当 IE 为 1 时，中断有效，即 pad_corex_int_b 起作用；

该位会被 reset 清零，也在进入异常服务程序时被清零。

FE-快速中断有效控制位：

当 FE 为 0 时，快速中断无效，即 pad_corex_fint_b 不起作用，FPC 和 FPSR 都无效；

当 FE 为 1（不必考虑 EE 位）时，快速中断有效，即 pad_corex_fint_b 起作用；

该位会被 reset 清零，也在进入快速中断服务程序时被清零，但不受其它异常的影响。

AF-可选择寄存器有效控制位 *：

当 AF 为 0 时，通用寄存器被选中，可选择寄存器不被选中；

当 AF 为 1 时，通用寄存器不被选中，可选择寄存器被选中；

此位被 reset 清零。当异常发生时，异常入口地址（异常向量表所指向的内存值）的最低位被拷贝到该位用来选择哪一组寄存器以便在异常服务程序中使用。

注：AF 位仅在配置可选择寄存器时有效，否则该位保留为零。

注：玄铁 800 处理器架构中，异常入口地址是异常向量表所指向的内存值，并非异常向量表本身的值。所以异常向量表所指向的内存处存的是异常入口的跳转地址，而非跳转指令，此异常入口地址为软件所配置，其最低位可指定异常发生时 AF 的值。

C-条件码 / 进位位

该位用作条件判断位为一些指令服务。它在 reset 和在被拷贝到 EPSR 或 FPSR 之后不确定。

3.5.1.1 更新 PSR

PSR 可以通过几种不同的方式被更新，对 PSR 中控制位的更改所产生的影响也多种多样。PSR 通常可以通过异常响应，异常处理和执行 MTCR、PSRSET、PSRCLR、RTE、RFI 指令被修改，这些修改的实现有四个方面。

- 异常响应和异常处理更新 PSR：
更新 PSR 是异常响应和异常服务程序入口地址计算中的一部分，它将更新 PSR 中 S, TM, VEC, IE, FE, EE 以及 AF 位。对 S, TM, VEC, IE, FE 以及 EE 位的改动优先于异常服务程序向量入口地址的取址。对 VEC 以及 AF 位的改动优先于异常服务程序中的第一条指令的执行。
- RTE 和 RFI 指令更新 PSR：
更新 PSR 作为 RTE 和 RFI 指令执行的一部分，可能会对 PSR 中的所有位都改动。其中对 S, TM, TP, IE, FE 和 EE 的改动优先于对返回 PC 的取址，对 VEC, MM, IC, AF 和 C 位的改动优先于程序返回后第一条指令的执行。
- MTCR 指令更新 PSR：
若目标寄存器是 CR<0,0> 的话，更新 PSR 将会作为 mtcrr 指令执行的一部分。这种更新将可能会改变 PSR 中所有位的值，紧接着的指令、异常事件和中断响应将会采用新的 PSR 值。
- PSRCLR、PSRSET 指令更新 PSR：
更新 PSR 作为 PSRCLR 和 PSRSET 指令执行的一部分，紧接着的指令、异常事件和中断响应将会采用新的 PSR 值。

3.5.2 复位向量基址寄存器 (RVBR, CR<28,0>)

RVBR 寄存器用来保存复位异常向量的基址。该寄存器包含 22 个高位有效位，10 个保留位（其值为 0）。RVBR 的复位值为 0X00000000。各个 C860 核心共享一个 RVBR 寄存器，因此对该寄存器的设置会影响所有核心的复位向量基址。

	31	10	9	0
	RESET VECTOR BASE			—
Reset	0			0

图 3.3: 复位向量基址寄存器

3.5.3 复位管理寄存器 (RMR, CR<29,0>)

复位管理寄存器用于多核启动时各个 C860 核心的复位释放使能。各个处理器核心共享一个 RMR 寄存器，因此某个处理器核心可以通过设置 RMR 将其它处理器核心从复位状态释放。

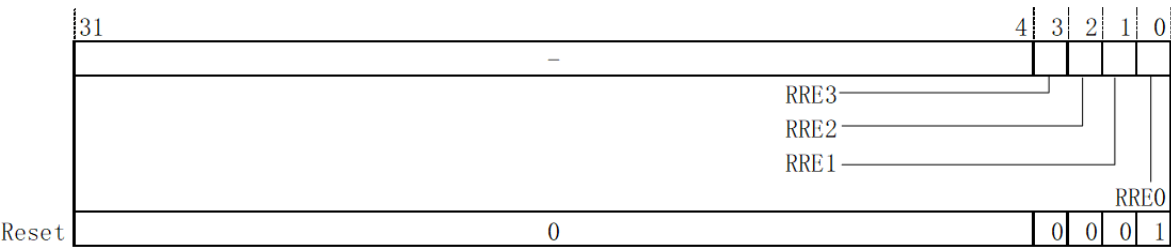


图 3.4: 复位管理寄存器

RRE3/2/1/0-复位释放使能位:

控制各个核心复位释放使能位，
当 RREx 为 0 时，对应 C860 核心处于复位状态；
当 RREx 为 1 时，对应 C860 核心复位释放。

3.5.4 向量基址寄存器 (VBR, CR<1,0>)

VBR 寄存器用来保存除复位异常之外所有异常向量的基址（复位异常的基址由复位向量基址寄存器指定，具体信息见 RVBR 描述）。该寄存器包含 22 个高位有效位，10 个保留位（其值为 0）。VBR 的复位值为 0X00000000。

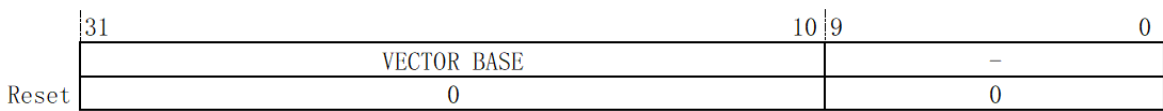


图 3.5: 基址向量寄存器

3.5.5 异常保留寄存器 (CR<2,0> ~ CR<5,0>)

EPSR, EPC, FPSR 和 FPC 这些寄存器在遇到异常情况时被用来保存当前处理器执行的内容。更详细的信息请参考[异常处理](#)。

3.5.6 全局控制寄存器 (GCR, CR<11,0>)

12 位的全局控制寄存器是用来控制外部设备和事件。它通过芯片口上提供的 12 位平行输出接口实现指定控制。一般来说，可以通过简单设置 GCR 来管理功耗，设备控制，事件安排处理以及其它的基本功能。至于 GCR 中每一位对应的控制功能，用户可以根据情况自行定义。全局控制寄存器是只可写的。

3.5.7 全局状态寄存器（GSR，CR<12,0>）

12 位的全局状态寄存器是用来标记外围设备和事件的。它通过芯片口上提供的 12 位平行输入接口将外部状态送入到 C860 内部，从而实现监测。一般来说，可以通过查看 GSR 来检测外围设备状态和事件。全局状态寄存器是只读的。

3.5.8 产品序号寄存器（CPUIDR，CR<13,0>）

该寄存器用于存放杭州中天微系统有限公司产品的内部编号。产品序号寄存器是只读的，其复位值由产品本身决定。

3.5.9 高速缓存功能设置寄存器（CFR，CR<17,0>）

高速缓存功能设置寄存器用来控制高速缓存，让相应高速缓存内的数据全部无效。

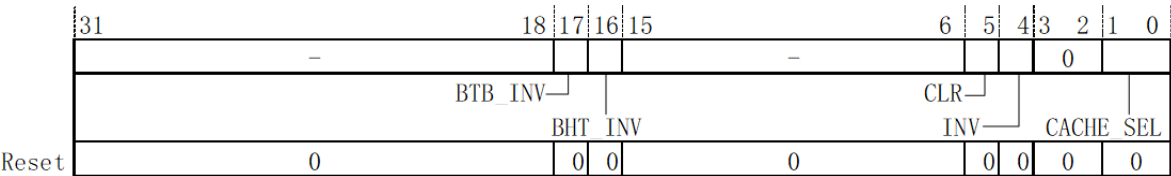


图 3.6: 高速缓存功能设置寄存器

BTB_INV-BTB 无效设置位：

当 BTB_INV 为 1 时，分支目标缓冲器内的数据将失效。

BHT_INV-BHT 无效设置位：

当 BHT_INV 为 1 时，分支历史表内的数据将失效。

CLR-脏表项清除设置位：

当 CLR 为 1 时，高速缓存内的被标记为脏的缓存行将被写出到片外内存中。

INV-无效设置位：

当 INV 为 1 时，高速缓存内的数据将失效。

CACHE_SEL-高速缓存选择位：

- 当 SEL 为 01 时，选中指令高速缓存；
- 当 SEL 为 10 时，选中数据高速缓存；
- 当 SEL 为 11 时，选中数据和指令高速缓存

3.5.10 高速缓存配置寄存器 (CCR, CR<18,0>)

高速缓存配置寄存器用来配置内存，高速缓存有效 / 无效，内存保护区，Endian 模式，以及系统和处理器的时钟比。

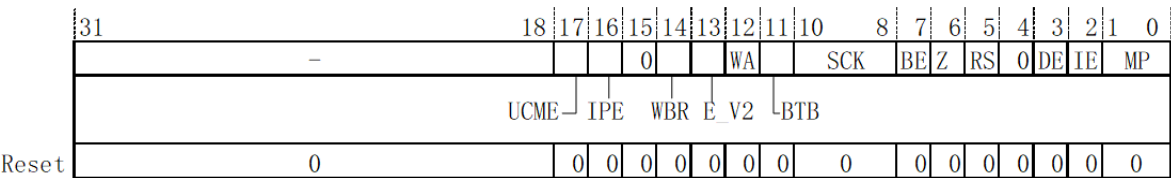


图 3.7: 高速缓存配置寄存器

UCME-普通用户模式执行 CACHE 指令使能位：

- 当 UCME 为 0 时，普通用户模式执行 CACHE 指令产生特权违反异常；
- 当 UCME 为 1 时，普通用户模式允许执行 DCACHE.CVAL1 和 ICACHE.IVA 指令，不产生特权违反异常。

IPE-间接分支跳转预测使能位：

- 当 IPE 为 0 时，间接分支跳转预测关闭；
- 当 IPE 为 1 时，间接分支跳转预测开启。

WBR-写突发传输使能位：

- 当 WBR 为 0 时，不支持写突发传输；
 - 当 WBR 为 1 时，支持写突发传输。
- C860 的 WBR 默认为 1，不可设置。

E_V2-大小端版本选择位：

- 当 E_V2 为 0 时，支持 V1 版本大小端模式；
- 当 E_V2 为 1 时，支持 V2 版本大小端模式。

WA-高速缓存写分配有效设置：

- 该位表示处理器执行 ST 指令且数据缓冲器缺失时，是否在数据高速缓存中分配缺失的缓存行。
- 当 WA 为 0 时，数据高速缓存为 write non-allocate 模式；
- 当 WA 为 1 时，数据高速缓存为 write allocate 模式。

BTB-分支目标预测使能位：

- 当 BTB 为 0 时，分支目标预测关闭；
- 当 BTB 为 1 时，分支目标预测开启。

SCK-系统和处理器的时钟比：

该位用来表示系统和处理器的时钟比，其计算公式为：时钟比 = $SCK + 1$ ，CPU 上有对应引脚引出。SCK 在 reset 时被配置且不能在之后改变。

BE-Endian 模式：

当 BE 为 0 时，Little endian；

当 BE 为 1 时，Big endian；

BE 在 reset 时被配置且不能在之后改变，CPU 上有对应引脚引出。

Z-允许预测跳转设置位：

当 Z 为 0 时，预测跳转关闭；

当 Z 为 1 时，预测跳转开启。

RS-地址返回栈设置位：

当 RS 为 0 时，返回栈关闭；

当 RS 为 1 时，返回栈开启。

DE-数据高速缓存设置位：

当 DE 为 0 时，数据高速缓存关闭；

当 DE 为 1 时，数据高速缓存开启。

IE-指令高速缓存设置位：

当 IE 为 0 时，指令高速缓存关闭；

当 IE 为 1 时，指令高速缓存开启。

MP-内存保护设置位：

MP 用来设置 MMU 是否有效，如 表 3.4。

表 3.4: C860 内存保护设置

MP	功能
00/10	MMU 无效
01/11	MMU 有效

3.5.11 二级高速缓存配置寄存器 (CCR2, CR<23,0>)

二级高速缓存配置寄存器用来配置共享的二级高速缓存种各个存储器的访问延时，二级高速缓存有效 / 无效，指令预取能力和 TLB 预取使能，以及 ECC 校验使能。

TPRF-L2 CACHE TLB 预取使能：

当 TPRF 为 0 时，TLB 预取功能关闭；

当 TPRF 为 1 时，TLB 预取功能开启。

	31	30	29	28	26	25	24	22	21	20	19	18	16	15		4	3	2	1	0
	IPRF		0			TLTNCY		0			DLTNCY		-				0			
	TPRF				TSETUP				DSETUP				L2EN				ECCEN		RFE	
Reset	0	0	0		0	0	0	0	0	0	0	0					0	0	0	

图 3.8: 二级高速缓存配置寄存器

IPRF-L2 CACHE 指令预取能力:

指示取指请求访问 L2 CACHE 缺失时预取的缓存行数量:

- 当 IPRF 为 0 时, 指令预取功能关闭;
- 当 IPRF 为 1 时, 预取 1 条缓存行;
- 当 IPRF 为 2 时, 预取 2 条缓存行;
- 当 IPRF 为 3 时, 预取 3 条缓存行。

TSETUP-L2 CACHE TAG RAM 的 setup 配置位:

- 当 TSETUP 为 0 时, TAG RAM 不需要额外的 setup 周期;
- 当 TSETUP 为 1 时, TAG RAM 需要额外的 1 个 setup 周期。

TLTNCY-L2 CACHE TAG RAM 的访问周期配置位:

- 当 TLTNCY 为 0 时, TAG RAM 访问周期为 1;
- 当 TLTNCY 为 1 时, TAG RAM 访问周期为 2;
- 当 TLTNCY 为 2 时, TAG RAM 访问周期为 3;
- 当 TLTNCY 为 3 时, TAG RAM 访问周期为 4;
- 当 TLTNCY 为 4 时, TAG RAM 访问周期为 5。

DSETUP-L2 CACHE DATA RAM 的 setup 配置位

- 当 DSETUP 为 0 时, DATA RAM 不需要额外的 setup 周期;
- 当 DSETUP 为 1 时, DATA RAM 需要额外的一个 setup 周期。

DLTNCY-L2 CACHE DATA RAM 的访问周期配置位:

- 当 DLTNCY 为 0 时, DATA RAM 访问周期为 1;
- 当 DLTNCY 为 1 时, DATA RAM 访问周期为 2;
- 当 DLTNCY 为 2 时, DATA RAM 访问周期为 3;
- 当 DLTNCY 为 3 时, DATA RAM 访问周期为 4;
- 当 DLTNCY 为 4 时, DATA RAM 访问周期为 5;
- 当 DLTNCY 为 5 时, DATA RAM 访问周期为 6;

当 DLTNCY 为 6 时，DATA RAM 访问周期为 7；
当 DLTNCY 为 7 时，DATA RAM 访问周期为 8。

L2EN-L2CACAHE 使能位：

当 L2EN 为 0 时，L2 CACHE 关闭；
当 L2EN 为 1 时，L2 CACHE 开启。

ECCEN-L2CACHE ECC 使能位：

当 ECCEN 为 0 时，L2 CACHE ECC 关闭；
当 ECCEN 为 1 时，L2 CACHE ECC 开启。

RFE-L2CACHE 数据访问读分配使能位：

数据访问 L2 CACHE 缺失，载入的缓存行是否回填 L2 CACHE；
当 RFE 为 0 时，数据访问在 L2 CACHE 缺失时不回填 L2 CACHE；
当 RFE 为 1 时，数据访问在 L2 CACHE 缺失时回填 L2 CACHE。
注：指令访问 L2 CACHE 缺失时需要回填 L2 CACHE。

3.5.12 二级高速缓存校验错误寄存器（CER2，CR<25,0>）

二级高速缓存支持可配置的 ECC，实现 1 比特错误可纠正、2 比特错误可检测的功能，当发现是 2 比特错误时，硬件自动设置 CER2 寄存器的 ECC_ERR 比特，供软件查询，软件可通过写入 0 的方式清除 ECC_ERR 比特，但不能对其置 1。

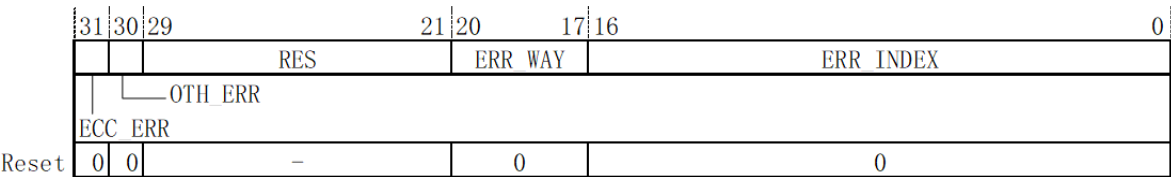


图 3.9: 二级高速缓存校验错误寄存器

ECC_ERR-L2 CACHE 校验错误位：

当 ECC_ERR 为 0，L2 CACHE 无 2 比特 ECC 错误和奇偶校验错误；
当 ECC_ERR 为 1，L2 CACHE 存在 2 比特 ECC 错误或者奇偶校验错误。

注：软件可在异常服务程序中清除该错误位，但无法置高该位。

OTH_ERR-L2 CACHE 其他校验错误位：

当 OTH_ERR 为 0，L2 CACHE 无其他位置的 2 比特 ECC 错误和奇偶校验错误；
当 OTH_ERR 为 1，L2 CACHE 存在其他位置的 2 比特 ECC 错误或者奇偶校验错误。

注：软件可在异常服务程序中清除该错误位，当无法置高该位。

ERR_WAY-L2 CACHE 校验错误的路位置信息：

记录 L2 CACHE 第一次出现 2 比特校验错误的路位置，在第一次错误没有被软件处理前再次发生的错误不会更新该信息。

ERR_INDEX-L2 CACHE 校验错误的索引信息：

记录 L2 CACHE 第一次出现 2 比特校验错误的索引位置，在第一次错误没有被软件处理前再次发生的错误不会更新该信息。

3.5.13 多核编号寄存器 (MPID, CR<30,0>)

多核标号寄存器用于指示当前 C860 核心的编号，该寄存器为只读寄存器。

	31	16	15	8	7	0
	-		CLUSTER ID		CORE ID	
Reset	0		0		*	

图 3.10: 多核编号寄存器

CLUSTER_ID- Cluster 编号：

指示当前 CPU 的 Cluster 编号。

CORE_ID- CORE 编号：

指示当前 CORE 的编号。

当 CORE_ID 为 0 时，表示 CORE0；

当 CORE_ID 为 1 时，表示 CORE1；

当 CORE_ID 为 2 时，表示 CORE2；

当 CORE_ID 为 3 时，表示 CORE3。

3.5.14 隐式操作寄存器 (HINT, CR<31,0>)

	31	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	RES					D	DIS	0				0		0			RES
						SRE			LPE			DPLD					
						IWPE			IPLD			AMR2			AMR		
Reset	-					0	-	0	0	0	0	-	0	0	0	0	-

图 3.11: CPU HINT 寄存器

D_DIS-DCACHE 预取缓存行数量：

当 D_DIS 为 0 时，预取 1 条缓存行；

当 D_DIS 为 1 时，预取 2 条缓存行；

当 D_DIS 为 2 时，预取 4 条缓存行；

当 D_DIS 为 3 时，保留；

默认为 0。

SRE-单退休模式位：

当 SRE 为 0 时，单退休模式关闭；

当 SRE 为 1 时，单退休模式开启。

IWPE-ICACHE 路预测使能位：

当 IWPE 为 0 时，ICACHE 路预测关闭；

当 IWPE 为 1 时，ICACHE 路预测开启。

LPE-循环加速使能位：

当 LPE 为 0 时，循环加速关闭；

当 LPE 为 1 时，循环加速开启。

IPLD-ICACHE 预取使能位：

当 IPLD 为 0 时，ICACHE 预取关闭；

当 IPLD 为 1 时，ICACHE 预取开启。

AMR2-L2cache 写分配策略自动调整使能位：

当 AMR2 为 0 时，写分配策略由访问地址的页面属性 WA 决定；

当 AMR2 为 1 时，在出现连续 12 条缓存行的存储操作时后续连续地址的存储操作不再写入 L2 Cache；

AMR-L1cache 写分配策略自动调整使能位：

当 AMR 为 0 时，写分配策略由访问地址的页面属性 WA 决定；

当 AMR 为 1 时，在出现连续 3 条缓存行的存储操作时后续连续地址的存储操作不再写入 L1 Cache；

DPLD-DCACHE 预取使能位：

当 DPLD 为 0 时，DCACHE 预取关闭；

当 DPLD 为 1 时，DCACHE 预取开启。

3.5.15 普通用户堆栈指针寄存器 (R14(User SP), CR<14,1>)

普通用户模式的通用寄存器 14 映射为控制寄存器 CR<14,1>。

3.5.16 Cache 访问索引寄存器 (CINDEX, CR<26,1>)

Cache 访问索引寄存器用来提供 CACHE 访问的 RAM 信息，包含 ICACHE、DCACHE 和 L2CACHE，以及指定 RAM 具体的 set/way 信息。

CACHE 访问索引寄存器如 图 3.12 所示。

	31	28 27	21 20	17 16	0
	RID	—	WAY	INDEX	
Reset	0	—	0	0	

图 3.12: CACHE 访问索引寄存器

RID-RAM 标识位：

指示访问的 RAM 信息：

- 当 RID 为 0 时，表示访问的是 ICACHE TAG RAM；
- 当 RID 为 1 时，表示访问的是 ICACHE DATA RAM；
- 当 RID 为 2 时，表示访问的是 DCACHE TAG RAM；
- 当 RID 为 3 时，表示访问的是 DCACHE DATA RAM；
- 当 RID 为 4 时，表示访问的是 L2CACHE TAG RAM；
- 当 RID 为 5 时，表示访问的是 L2CACHE DATA RAM；
- 当 RID 为 6 时，表示访问的是 L2CACHE ECC RAM。

WAY-Cache 路信息：

指示 RAM 访问的路位置信息。

INDEX-CACHE 索引：

指示 RAM 访问的索引位置信息。

3.5.17 Cache 访问数据寄存器 0~3 (CDATA0~CDATA3, CR<27,1>~CR<30,1>)

Cache 访问数据寄存器用于存储从 CACHE 中读取的内容，不同 RAM 存储的数据含义不同，CDATA0~CDATA3 为只读寄存器。

	31	0
	DATA	
Reset	0	

图 3.13: CACHE 访问数据寄存器

CDATA0~CDATA3 内容具体如 表 3.5 所示。

表 3.5: CACHE 访问数据寄存器内容

RAM 类型	CDATA 内容
ICACHE TAG	CDATA0[31:12]: TAG CDATA0[0]: VALID
ICACHE DATA	CDATA0~CDATA3: 128bit DATA
DCACHE TAG	CDATA0[31:12]: TAG CDATA0[2]: SHARED CDATA0[1]: DIRTY CDATA0[0]: VALID
DCACHE DATA	CDATA0~CDATA3: 128bit DATA
L2CACHE TAG	CDATA0[31:12]: TAG CDATA[1]: DIRTY CDATA0[0]: VALID
L2CACHE DATA	CDATA0~CDATA3: 128bit DATA
L2CACHE ECC	CDATA[10:0]: ECC

3.5.18 Cache 访问指令寄存器 (CINS, CR<31,1>)

Cache 访问指令寄存器用于实现 CACHE 读操作的命令，是一个只写寄存器。该操作通过 MTCR 指令实现。

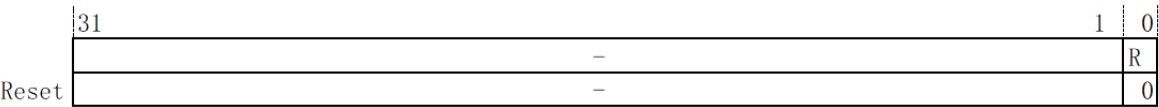


图 3.14: CACHE 访问指令寄存器

R-CACHE 读访问:

该指令用于从 CACHE 读出一个数据。由 CINDEX 指向的 CACHE 内容被装载到 CDATA0~CDATA3 寄存器。如果 CINDEX 寄存器的值无效时，处理器的操作将不可预知。

3.5.19 浮点版本寄存器 (FID, CR<0,2>)

浮点版本寄存器是一个只读寄存器，软件用户可通过查询浮点版本寄存器确定当前处理器是否配置了浮点单元以及版本信息。

Implement-浮点单元的实现方式:

- Implement 为 0，未实现硬件浮点单元，处理器将通过软件模拟方式实现浮点功能；
- Implement 为 1，实现了硬件浮点单元。

当 DQNaN 为 1 时，计算输出的 QNaN 值跟 IEEE754 标准一致。

当 DQNaN 为 0 时，计算输出的 QNaN 值为默认的固定值。

FRPOS-定点数小数点位置：

定点和浮点数相互转换时，指示定点数的小数点位置。

IDE-非规格化数异常使能位：

当 IDE 为 1 时，若输入非规格数，处理器将终止程序执行并抛出非规格化数异常，不回写任何结果；

当 IDE 为 0 时，若输入非规格数，处理器将根据 FM 指示位，对非规格化数进行处理，并继续运算。

IXE-非精确异常使能位：

当 IXE 为 1 时，若运算结果非精确，处理器将终止程序执行并抛出非精确异常，不回写任何结果；

当 IXE 为 0 时，若运算结果非精确，处理器将根据 RM 指示位，对非规格化数进行舍入处理，并继续运算。

UFE-下溢异常使能位：

当 UFE 为 1 时，若运算结果下溢出，处理器将终止程序执行并抛出下溢异常，不回写任何结果；

当 UFE 为 0 时，若运算结果下溢出，处理器将根据 FM 指示位，对下溢结果进行处理，并继续运算。

OFE-上溢异常使能位：

当 OFE 为 1 时，若运算结果上溢出，处理器将终止程序执行并抛出上溢异常，不回写任何结果；

当 OFE 为 0 时，若运算结果上溢出，处理器将根据 RM 指示位，对上溢结果进行处理，并继续运算。

DZE-除 0 异常使能位：

当 DZE 为 1 时，若发生除零运算，处理器将终止程序执行并抛出除以零异常，不回写任何结果；

当 DZE 为 0 时，若发生除零运算，处理器将以正/负无穷浮点数作为运算结果，并继续运算。

IOE-非法操作异常使能位：

当 IOE 为 1 时，若发生非法操作，处理器将终止程序执行并抛出非法操作异常，不回写任何结果；

当 IOE 为 0 时，若发生非法操作，处理器将以 QNaN 作为运算结果，并继续运算。

3.5.21 浮点异常状态寄存器 (FESR, CR<2,2>)

FESR 提供浮点的异常状态信息，指示发生的浮点异常类型。FESR 在超级用户模式和普通用户态都可以被读写。

	31	16														15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
	Res															FEA	Res	IDA	IXA	UFA	OFA	DZA	IOA	FEC	Res	IDC	IXC	UFC	OFC	DZC	IOC		
Reset	0															0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

图 3.17: 浮点异常状态寄存器

FEA-浮点异常累积位：

当有任何一个浮点异常发生并且相应的使能位未开启时，该位将被置为 1。

IDA-输入非规格化数异常累积位：

在发生输入非规格数异常且 IDE 位为 0 时，该位被置为 1。

IXA-非精确异常累积位：

在发生非精确异常且 IXE 位为 0 时，该位被置为 1。

UFA-下溢异常累积位：

在发生下溢异常且 UFE 位为 0 时，该位被置为 1。

OFA-上溢异常累积位：

在发生上溢异常且 OFE 位为 0 时，该位被置为 1。

DZA-除 0 异常累积位：

在发生除 0 异常且 DZE 位为 0 时，该位被置为 1。

IOA-非法操作异常累积位：

在发生非法操作异常且 IOE 位为 0 时，该位被置为 1。

FEC-浮点异常位：

当前任何一个浮点异常发生时，该位将被置为 1。

IDC-输入非规格化数异常累积位：

当前发生输入非规格化数异常时，该位被置为 1。

IXC-非精确异常累积位：

当前发生非精确异常时，该位被置为 1。

UFC-下溢异常累积位：

当前发生下溢异常时，该位被置为 1。

OFC-上溢异常累积位：

当前发生上溢异常时，该位被置为 1。

DZC-除 0 异常累积位：

当前发生除 0 异常时，该位被置为 1。

IOC-非法操作异常累积位：

当前发生非法异常时，该位被置为 1。

3.6 数据格式

3.6.1 浮点数据格式

C860 浮点单元兼容 ANSI/IEEE Std 754 浮点标准，支持单精度和双精度浮点运算，矢量配置下可支持 128 位矢量运算，四路单精度数据并行运算，如 图 3.18 所示。

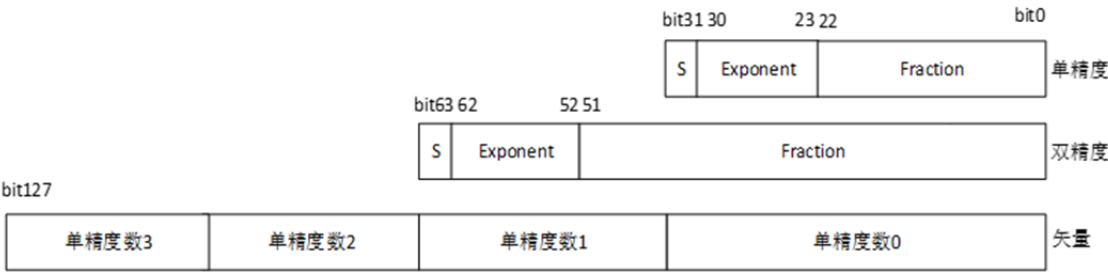


图 3.18: 寄存器中的浮点数据组织结构

3.6.2 整型数据格式

寄存器内部的数值并没有大小端之分，只有有符号和无符号的区别。其格式均为从右至左表示逻辑低位到高位 的排布，如 图 3.19 所示。

3.6.3 大小端

大小端的概念是相对于存储器数据存储的格式而提出的，高地址字节存放至物理内存的低位被定义大端；高地址字节存放至物理内存的高位被定义为小端，如 图 3.20 所示。

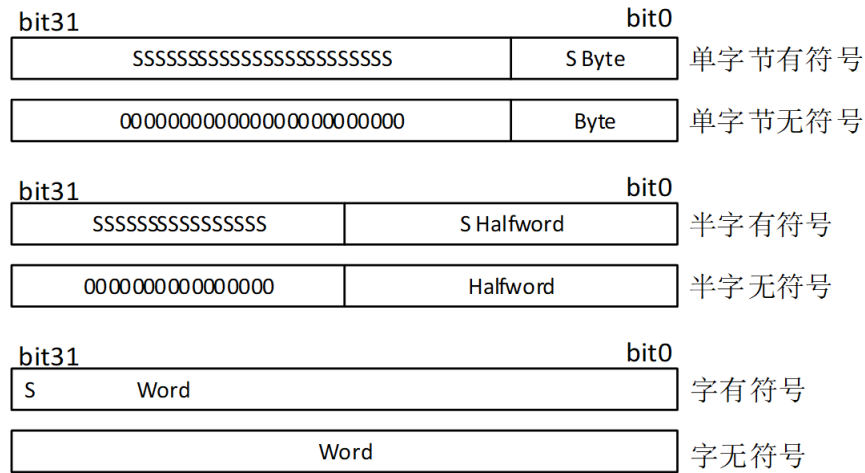


图 3.19: 寄存器中的整型数据组织结构

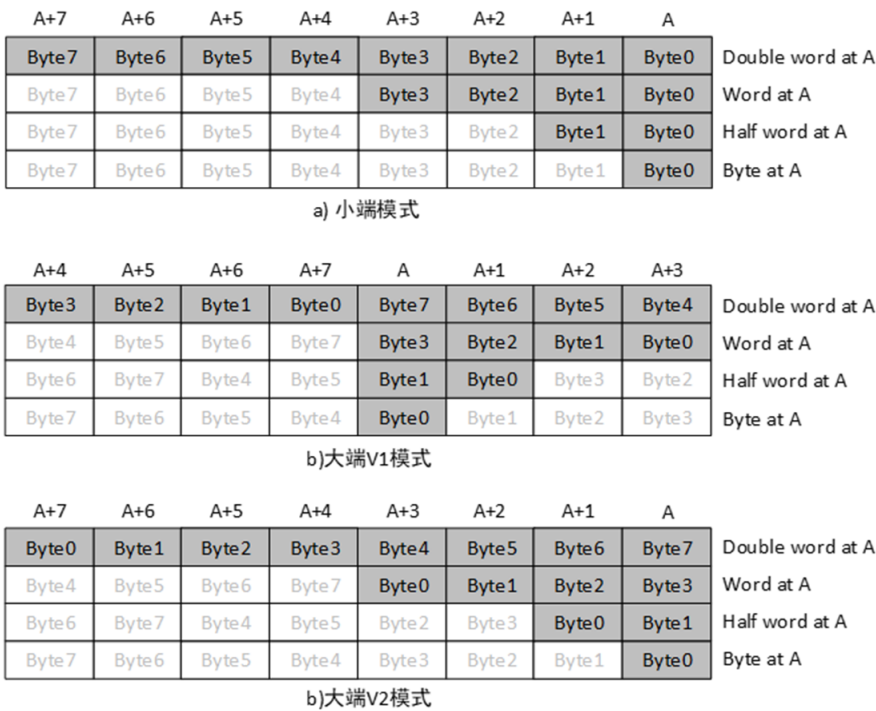


图 3.20: 内存中的数据组织形式

C860 定义了 2 个版本的大小端模式，由接口信号 pad_cpu_bigend_b 和 pad_cpu_endian_v2 指定。其中，小端 V1 版本和小端 V2 版本相同，统一位小端模式；大端 V1 版本和大端 V2 版本在数据组织形式上不相同，因此，有大端 V1 和大端 V2 之分。在小端模式下，无论存取指令的访问大小为字节/半字/字/双字，始终以最低位字节数据对应于地址的最低位；在大端 V1 模式下，根据 32bit 为边界将高位字节存放在低位地址上；而大端 V2 模式，则根据访问大小为边界，将高位字节存放在低位地址上。图 3.20 给出了以不同访问大小访问地址 A，数据在内存中的位置和各个字节的排列顺序。

C860 支持标准补码的 2 进制整数。每个指令操作数的长度可以明确地编码在程序中 (load/store 指令)，也可以隐含在指令操作中 (index operation, byte extraction)。通常，指令接收 32 位操作数，产生 32 位结果。

3.7 内存模型

C860 支持两种内存类型，分别是内存 (memory) 和外设 (device) (由 SO 位区分)。其中，memory 类型的特点是支持投机执行和乱序执行，根据是否可高缓 (Cacheable, C) 进一步分为可高缓内存 (cacheable memory) 和不可高缓内存 (non-cacheable memory)。device 类型的特点为不可投机执行且必须按序执行，因此 device 一定带有不可高缓的属性。device 根据是否可缓存 (Bufferable, B) 分为可缓存外设 (bufferable device) 和不可缓存外设 (non-bufferable device) (bufferable 表示 slave 允许在某个中间节点快速返回写完成；反之，non-bufferable 表示 slave 只有在最终设备真正写完成后才返回写响应)。

为了支持多核之间数据共享，C860 增加了可共享的页面属性 (Shareable, SH)。对于可共享的页面，表示该页面多核间共享，由硬件维护数据的一致性；对于不可共享的页面，表示被某个单核独占，不要求硬件维护数据的一致性。所有可缓存区域都被认定为可共享。

表 3.6 给出了各个内存类型对应的 TLB 页面属性。页面属性的配置可通过操作 MEL0/1 和 MSA0/1 寄存器实现。

表 3.6: 内存类型分类

内存类型	SO	C	B
可高缓内存	0	1	1
不可高缓内存	0	0	1
可缓存外设	1	0	1
不可缓存外设	1	0	0

3.7.1 内存一致性模型

C860MP 采用宽松的内存一致性模型 (Weak Memory Ordering)，该模型具体定义如下：

- 各个 CORE 保证相同地址访问的顺序性，包括读后读、写后写、读后写和写后读；
- 各个 CORE 放松不同地址访问的顺序性，包括读后读、写后写、读后写和写后读；
- 保证 other-multi-copy 的原子性，即要求当一个核能获得另一个核的写数据时，保证其他核此时也能够获得该写数据；而一个核能够获得自己核的写数据时，不要求其他核此时也能够获得该写数据。

由于 weak memory ordering 的模型，导致多核之间内存实际的读写顺序和程序给定的访问顺序会不一致。因此，提供了 BAR 和 SYNC 指令供软件强制规定内存访问的顺序性。

BAR 指令限定了 LD、ST 指令执行的顺序，保证了 BAR 指令之前的 LD、ST 指令一定在 BAR 之后的 LD、ST 指令之前执行。BAR 指令可以指定前后指令的类型和作用域的范围。其中，指令类型可以指定前序指令的类型和后序指令的类型，组合起来有前 LD/ST 后 LD/ST、前 LD 后 LD 和前 ST 后 ST 3 种类型；作用域有本核和多核 2 种，本核只影响执行该 BAR 指令的 CORE，而多核不仅影响执行该 BAR 指令的 CORE，对其他 CORE 发到总线上的内存访问顺序也做了顺序化处理。具体指令如 表 3.7 所示。

表 3.7: BAR 指令描述

助记符	前序指令操作类型	后序指令操作类型	作用域
BAR.BRWARW	LD/ST	LD/ST	本核
BAR.BRAR	LD	LD	本核
BAR.BWAW	ST	ST	本核
BAR.BRWARWS	LD/ST	LD/ST	多核
BAR.BRARS	LD	LD	多核
BAR.BWAWS	ST	ST	多核

而 SYNC 指令限定了所有指令的执行顺序，保证了 SYNC 指令之前的所有指令一定在 SYNC 指令执行之前完成。另外，SYNC 指令还可以额外同步指令内存，即在 SYNC 指令前序指令完成时清空流水线，重新取指。具体指令如 表 3.8 所示。

表 3.8: SYNC 指令描述

助记符	指令描述	作用域
SYNC.IS	Synchronize data and instruction memory	多核
SYNC.I	Synchronize data and instruction memory	本核
SYNC.S	Synchronize data memory	多核
SYNC	Synchronize data memory	本核

第四章 异常处理

异常处理（包括指令异常和外部中断）是处理器的一项重要技术，在某些异常事件产生时，处理器会停止当前指令的执行，并转入对异常事件的处理。这些事件包括硬件错误，指令执行错误，和用户请求服务等。本章主要描述异常种类、异常优先级、异常向量表、异常返回和总线错误恢复等内容。

4.1 异常处理概述

异常处理是处理器根据内部或外部的异常事件从正常的程序处理转入特定的异常处理程序。引起异常的外部事件包括：外部设备的中断请求、读写访问错误和硬件重启；引起异常的内部事件包括：非法指令、不对齐错误（misaligned error）、特权异常、浮点异常和指令跟踪，TRAP 和 BKPT 指令正常执行时也会产生异常。异常处理利用异常向量表和一组影子寄存器跳转到异常服务程序的入口。

异常处理的关键就是在异常发生时，保存 CPU 当前指令运行的状态，在退出异常处理时恢复异常处理前的状态。异常能够在指令流水线的各个阶段被识别，并使后面的指令不会改变 CPU 的状态。异常在指令的边界上被处理，即 CPU 在指令退休时响应异常，并保存退出异常处理时下一条被执行的指令的地址。即使异常在 IF 阶段（未对齐访问、访问错误）或 ID 阶段（非法指令、TRAP 和 BKPT）被识别，异常也要在相应的指令退休时才会被处理。为了异常处理不影响 CPU 的性能，CPU 在异常处理结束后要避免重复执行以前的指令。C860 根据异常识别时的指令是否完成决定异常地址寄存器存储哪一条指令的地址。例如，如果异常事件是外部中断服务请求，被中断的指令将正常退休并改变 CPU 的状态，它的下一条指令的地址将被保存在异常地址寄存器中作为中断返回时指令的入口；如果异常事件是由除以零的除法指令产生的，因为这条指令不能完成。它将异常退休但不改变 CPU 的状态（即不改变寄存器的值），这条除法指令的地址将被保存在异常地址寄存器中，CPU 从中断服务程序返回时继续执行这条除法指令。

异常按以下步骤被处理：

第一步，处理器更新 PSR 中的 VEC，然后保存 PSR 和 PC 到影子寄存器中。对于快速中断，PSR 和 PC 被保存到 FPSR 和 FPC 中；对于其它的异常，它们被保存到 EPSR 和 EPC 中。如果 PSR 的 EE 位被清零了，异常事件（除了普通中断和快速中断）将导致不可恢复的错误异常。PSR 和 PC 被保存后，PSR 的 EE 位被清零，处理器就只能处理不可恢复的错误异常。不可恢复的错误异常发生时，EPSR 和 EPC 也会被更新。

第二步，处理器设置 PSR 的 S 位进入超级用户模式，并且把 PSR 的 TM 位清零防止异常服务程序被跟踪。中断使能位 PSR 的 IE 位也被清零禁止响应中断。如果异常是快速中断或重启，快速中断使能 PSR 的 FE 位会被清零，但是其它的异常不影响 PSR 的 FE 位。

传输控制位 PSR 的 TE 位被清零防止外部存储器管理单元对在异常入口地址计算时进行地址转译，使下面的访问以非转译的方式进行。

处理器还要决定对应的异常向量。对于向量中断，异常向量由中断控制器提供；对于其它的异常，处理器根据内部逻辑决定异常向量。异常向量用来计算异常服务程序的入口地址，并被保存在 PSR 的 VEC 中以支持共享异常服务的情况。

最后一步，处理器计算异常服务程序的第一条指令的地址并将 CPU 的控制权转交给异常服务程序。处理器把异常向量乘以 4 再加上异常向量基准地址（复位异常基准地址存在复位向量基址寄存器种，其它异常基准地址存在向量基准地址寄存器中）就得到了异常向量表的地址，处理器用这个异常向量表地址从存储器中读取一个字，便得到异常入口地址，这个异常入口地址的 [31:1] 转载到程序计数器中作为异常服务程序的第一条指令的地址（PC 的最低位始终是 0，与异常向量表中取得的异常入口地址值的最低位无关）。在特定硬件配置下，处理器把这个异常入口地址（异常向量表所指向的内存值）的最低位装载到 PSR（AF）位，它用来控制异常处理程序使用哪一个寄存器组。就这样，处理器开始执行新的指令。（注：玄铁 800 处理器架构中，软件所维护的异常向量表中存放的是异常入口的跳转地址，而并非跳转指令，其最低位可指定异常发生时 AF 的值。）

所有的异常向量存放在超级用户地址空间，并以指令空间索引（TC1 = 1）访问。在处理器地址映射中，只有重启向量是固定的。一旦处理器完成初始化，VBR 允许异常向量表的基准地址被重载。

C860 支持 1024 个字节的向量表包含 256 个异常向量（见表 4.1）。开始的 31 个向量是用作在处理器内部识别的向量。第 32 个向量是留给软件的，用作指向系统描述符的指针。其余的 224 个向量是留给外部设备的。外部设备通过 8 位的中断向量和中断请求使处理器响应中断服务。处理器响应中断请求时锁存这个中断向量。对那些不能提供中断向量的设备，处理器为一般中断和快速中断提供了自动向量。

表 4.1: 异常向量分配

向量号	向量偏移（十六进制）	向量分配
0	000	重启异常。
1	004	未对齐访问异常。
2	008	访问错误异常。
3	00C	除以零异常。
4	010	非法指令异常。
5	014	特权违反异常。
6	018	跟踪异常。
7	01C	断点异常。
8	020	不可恢复错误异常/TLB 不可恢复异常。
9	024	Idly 异常。
10	028	普通中断。
11	02C	快速中断。
12	030	保留（HAI）。
13	034	保留（FP）。
14	038	保留（TLB 失配异常，仅在 TLB 软件回填功能使能情况下存在）。
15	03C	保留（TLB 修改异常）。
16 - 19	040 - 04C	陷阱指令异常（TRAP # 0-3）。
20	050	保留（TLB 读无效异常）。
21	054	保留（TLB 写无效异常）。
22 - 29	058 - 074	保留。
30	078	浮点异常。
31	07C	系统描述符指针。
32 - 255	080 - 3FC	保留给向量中断控制器使用。

4.2 异常类型

本节描述外部中断异常和在 C860 内部产生的异常。C860 处理的异常有以下几类：

- 重启异常；
- 未对齐访问异常；
- 访问错误异常；
- 除以零异常；
- 非法指令异常；
- 特权违反异常；
- 跟踪异常；

- 断点异常；
- 不可恢复错误异常/TLB 不可恢复异常；
- Idly4 异常；
- 普通中断；
- 快速中断；
- TLB 失配异常；
- TLB 修改异常；
- 陷阱指令异常；
- TLB 读无效异常；
- TLB 写无效异常；
- 浮点异常。

4.2.1 重启异常（向量偏移 0X0）

重启异常是所有异常中优先级最高的，它是用于系统初始化和发生重大故障后恢复系统。重启会中止处理器的所有操作，被中止的操作是不可恢复的。重启也在测试时用于初始化扫描链和时钟控制逻辑中锁存器的值，它也同时对处理器进行上电初始化。

重启异常设置 PSR (S) 为高电平使处理器工作在超级用户模式，并且把 PSR (TM) 清零禁止跟踪异常。重启异常也会把 PSR (IE) 和 PSR (FE) 清零以禁止中断响应。同时，复位向量基址寄存器也被顶层输入复位基址值重置，CPU 从复位向量基址寄存器提供的异常向量表中以偏移地址 0X0 为偏移地址读取异常向量，并把它装载到程序计数器 (PC)。异常处理器把控制权转移到 PC 指向的地址。

4.2.2 未对齐访问异常（向量偏移 0X4）

处理器试图在与访问大小不一致的地址边界上执行访问操作，就会发生地址未对齐访问异常。EPC 指向试图进行未对齐访问的指令。

对于地址属性 SO 为 0 的区域，若将 PSR 寄存器的 MM 位开启，则不会发生未对齐访问异常。

对于地址属性 SO 为 1 的区域，无论 MM 位是否开启，未对齐访问均会触发异常。

当配置 VDSP 时，CPU 支持 VDSP 指令，VDSP 指令的对齐方式包括 element size 对齐和访问长度对齐 (即 $T \times N$, T 为 element size, $N=1/2/4$)。当访问长度对齐时，element size 一定也会是对齐的。CPU 对于 VDSP 指令未对齐访问的处理为：

1. 对于地址属性 SO 为 0 的区域，VDSP 指令不支持 element size 未对齐访问，无论 MM 位是否开启，均会触发未对齐访问异常。
2. 对于地址属性 SO 为 1 的区域，VDSP 指令只支持访问长度对齐，否则无论 MM 位是否开启，均会触发未对齐访问异常。

4.2.3 访问错误异常（向量偏移 0X8）

如果 pad_biu_rresp[1] 为 1，就意味着发生了访问错误异常。当普通用户模式下访问特权用户模式的地址空间，会发生访问错误异常。

总线上的错误都会引起访问错误异常；其中，由数据访问产生的总线访问错误异常是一个非精确异常。为了加速载入指令的退休速度，在没有得到总线错误信息时可能已经退休，因此总线访问错误异常可能由后续任一指令响应。

4.2.4 除以零异常（向量偏移 0X0C）

当处理器发现除法指令的除数是零时，处理器进行异常处理而不执行该除法指令。EPC 指向该除法指令。

4.2.5 非法指令异常（向量偏移 0X10）

处理器译码时如果发现了非法指令或没有实现的指令，该指令不会被执行而进行异常处理。EPC 指向该非法指令。

4.2.6 特权违反异常（向量偏移 0X14）

为了保护系统安全，一些指令被授予了特权，它们只能在超级用户模式下被执行。试图在用户模式下执行下面的特权指令都会产生特权违反异常：MFCR、MTCR、PSRSET、PSRCLR、RFI、RTE、STOP、WAIT、DOZE、ICACHE、DCACHE、L2CACHE、TLBI。

处理器如果发现了特权违反异常，在执行该指令前进行异常处理。EPC 指向该特权指令。

4.2.7 跟踪异常（向量偏移 0X18）

为了便于程序开发调试，C860 提高了对每条指令或对改变控制流指令的跟踪能力。在指令跟踪模式下，每条指令在执行完后都会产生一个跟踪异常，以便于调试程序监测程序的执行。在跳转跟踪模式下，每条改变控制流的指令都会产生一个跟踪异常。对于条件跳转指令，不管程序有没有跳转，跟踪异常都会发生。

如果处理器工作在跳转跟踪模式下，以下指令都会引起跟踪异常：JMP, JSR, JMPI, JSRI, BR, BT, BF, BSR, BE, BNE, BEZ, BNEZ, BHZ, BLSZ, BLZ, BHSZ。无论条件跳转指令的执行结果如何，跟踪异常都被使能了。

如果被跟踪的指令由于发生了其它的异常而没有完成，跟踪异常不会被处理或被标记为等待处理。如果在指令完成时外部中断被探测到了，跟踪异常将延时处理，在 EPSR 中被标记为等待处理的跟踪异常。

PSR 的 TM 位控制跟踪模式。TM 的状态决定了指令退休时是否产生跟踪异常。请参考第三章 PSR 的 TM 位的定义。

跟踪异常处理起始于被跟踪的指令退休之后且在下一条指令退休之前。EPC 指向下一条指令。

以下控制相关的指令不能被跟踪：RTE, RFI, TRAP, STOP, WAIT, DOZE 和 BKPT。如果 EPSR (TP) 有效，跟踪异常作为 RTE 或 RFI 正常执行的一部分被处理，而不管 PSR 或 EPSR 的 TM 位的状态。

如果在被跟踪的指令退休时处理器发现有中断等待处理，中断的优先级比跟踪异常高，PSR 的影子寄存器的 TP（等待处理的跟踪异常）将有效，处理器响应中断。中断服务程序的 RTE 或 RFI 退休时，等待处理的跟踪异常会被处理。等待处理的跟踪异常是用来跟踪前面的指令的，这条指令已经退休，为了避免这个异常被丢失，它的优先级是最高的仅次于重启。

4.2.8 断点异常（向量偏移 0X1C）

C860 提供了断点指令 BKPT，退休时产生断点异常。断点异常发生时，EPC 指向该指令。

4.2.9 不可恢复错误异常（向量偏移 0X20）

当 PSR (EE) 为零时，除了快速中断外的异常会产生不可恢复的异常，因为这时用于异常恢复的信息（存于 EPC 和 EPSR）由于不可恢复的错误而被重写了。

由于所写的软件在 PSR (EE) 为零时应该排除了异常事件发生的可能，这种错误一般意味着有系统错误。在不可恢复错误异常的服务程序中，引起不可恢复错误异常的异常类型是不确定的。

4.2.10 IDLY 异常（异常偏移 0X24）

idly 异常用来指示在 idly 指令序列中发生了传输错误。在该异常服务程序中，EPC 指向引起传输错误的指令。异常服务程序应该分析发生传输错误的原因，并备份 EPC 的值以便在必要时重新执行 idly 指令序列。

4.2.11 中断异常

当外部设备需要向处理器请求服务或发送处理器需要的信息时，它可以用中断请求信号和相应的中断向量信号向处理器请求中断异常。

一般中断在指令的边界上被确认，如果 PSR (IC) 位设置了，处理器也可以通过允许许多周期指令被中断的方式来缩短中断响应的延时，例如：DIVS, DIVU, LDM, STM, LDQ, STQ, PUSH 和 POP 指令可以被中断而不等它们完成。

当 C860MP 没有配置内嵌中断控制器时，由外部中断控制器提供中断请求信号，C860MP 支持自动提供中断向量号和由外设显式地提供中断向量号的方式。图 4.1 显示了和中断相关的到处理器核的接口信号。为了支持向量化的中断，外设在中断请求时，用 7 位的中断向量信号提供中断向量号，或设置 pad_corex_avec_b 使用预先定义好的中断向量号。如果 pad_corex_avec_b 有效，pad_corex_vec_b[7:0] 上的输入信号被忽略。pad_corex_int_b, pad_corex_fint_b 和 pad_corex_avec_b 都是低电平有效。

当 C860MP 配置了内嵌中断控制器时，由内嵌中断控制器给各个 C860 核心提供中断请求信号；而中断控制器负责接收外设中断源，包括 128 个公有中断和 15 个私有中断，其中每个 C860 核心都由一组私有中断 pad_intc_pintx。具体如图 4.2 所示。

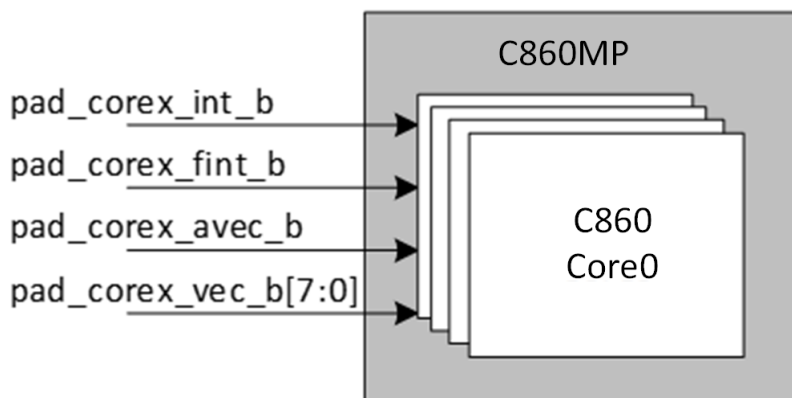


图 4.1: 没有配置中断控制器的中断接口信号

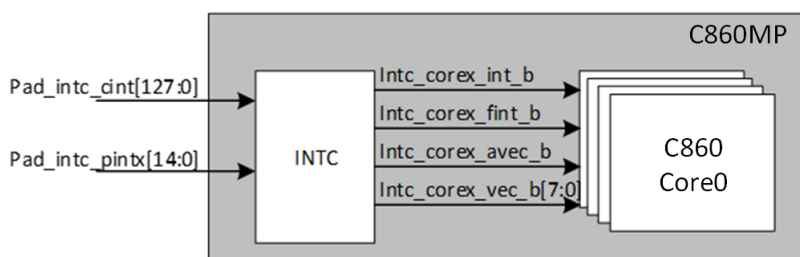


图 4.2: 配置中断控制器的中断接口信号

4.2.11.1 普通中断 (INT)

当没有配置中断控制器时，普通中断请求来自外部输入的引脚 `pad_corex_int_b`；当配置中断控制器时，普通中断请求来自于中断控制器输出给各个 CORE 的 `intc_corex_int_b`，后续统一成为 `int_b`。如果 PSR (IE) 被清零了，`int_b` 输入信号被屏蔽，处理器不响应异常。普通中断使用 EPSR 和 EPC 这一组异常影子寄存器，它也可以被 PSR (EE) 屏蔽。当 `int_b` 有效时，如果 `avec_b` 也有效，处理器使用自动向量号，它的向量偏移是 0x28，否则，处理器使用 `vec_b[7:0]` 上提供的向量号，它可以是 32 - 255 中的任意一个（硬件上不排除使用 0 - 31）。

4.2.11.2 快速中断 (FINT)

当没有配置中断控制器时，普通中断请求来自外部输入的引脚 `pad_corex_fint_b`；当配置中断控制器时，普通中断请求来自于中断控制器输出给各个 CORE 的 `intc_corex_fint_b`，后续统一成为 `fint_b`。如果快速中断被使能（PSR (FE) 有效），它的优先级比普通中断请求高。快速中断使用 FPSR 和 FPC 这一组异常影子寄存器，它不会被 PSR (EE) 屏蔽。如果 PSR (FE) 清零了，快速中断就被屏蔽了。当 `fint_b` 有效时，如果 `avec_b` 也有效，处理器使用自动向量号，它的向量偏移是 0x2C，否则，处理器使用 `vec_b[7:0]` 上提供的向量号，它可以是 32 - 255 中的任意一个（硬件上不排除使用 0 - 31）。

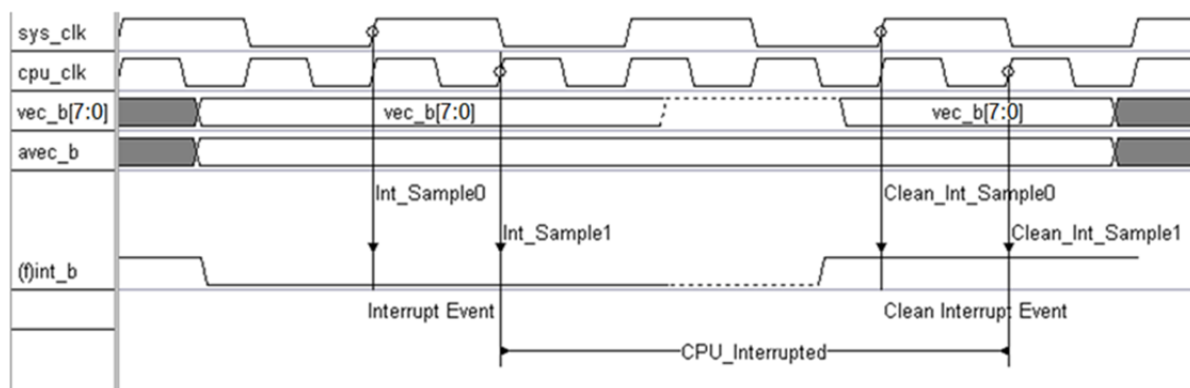


图 4.3: 中断处理过程

4.2.11.3 中断处理过程

在图 4.3 中，当中断向量号 `vec_b[7:0]` 和自动中断向量 `avec_b` 都已经准备好时，拉低普通中断信号线 `int_b`，该信号经系统时钟 `sys_clk` 和处理器内部时钟 `cpu_clk` 的上升沿依次采样后，处理器内部收到中断，并根据 `avec_b` 的设置取得中断向量，进入中断服务程序。在中断服务程序中，应该清除外部中断，即拉高 `int_b`，此信号亦需经此两个时钟依次采样后才会退出中断。快速中断 (`fint`) 的操作亦是如此。

4.2.12 TLB 不可恢复异常（向量偏移 0X34）

在物理地址转换或 jTLB 查找过程中，如果发现多个匹配的表项存在，将发生 TLB 不可恢复异常。

4.2.13 TLB 失配异常（向量偏移 0X38）

在处理器取指或数据访问时，如果 jTLB 中没有与虚拟地址匹配的表项，将发生 TLB 失配异常，软件需要负责再填充 jTLB。

C860 增加了 TLB 硬件回填的功能，当 MMU 的 TLB 硬件回填功能使能时，该异常不发生。

4.2.14 TLB 修改异常（向量偏移 0X3C）

在处理器写数据时，如果 MMU 的 jTLB 表项与虚拟地址匹配，但是匹配的表项关闭了页面 `dirty` 位时发生 TLB 修改异常。

4.2.15 陷阱指令异常（向量偏移 0X40 - 0X4C）

一些指令可以用来显示地产生陷阱异常。`TRAP # n` 指令可以强制产生异常，它可以用于用户程序的系统调用。在异常服务程序中，EPC 指向 `TRAP` 指令。

4.2.16 TLB 读无效异常（向量偏移 0X50）

在处理器取指或读数据时，如果 MMU 的 jTLB 表项与虚拟地址匹配，但是匹配的表项关闭了页面有效位时发生 TLB 读无效异常。

4.2.17 TLB 写无效异常（向量偏移 0X54）

在处理器写数据时，如果 MMU 的 jTLB 表项与虚拟地址匹配，但是匹配的表项关闭了页面有效位时发生 TLB 写无效异常。

4.2.18 浮点异常（向量偏移 0X78）

浮点运算会产生 6 种浮点算术类异常，除了浮点协议标准（ANSI/IEEE Std 754）定义的 5 种浮点运算异常外，还包括非规格化数输入异常。浮点指令的异常响应过程与主流水线类似，所有的 6 类浮点算术异常共享同一个异常向量号 30。

处理器在响应浮点算术异常之后，软件将通过查询浮点异常寄存器中的指示位，即 Current 位来判断当前指令发生的异常，并依据浮点异常优先级进行先后的处理。

浮点算术异常包括（按优先级由高到低排列）：

- 非规格化数输入异常（Input Denormalized Exception）；
- 非法操作异常（Invalid Operation Exception）；
- 除以零异常（Divide by Zero Exception）；
- 下溢异常（Underflow Exception）；
- 上溢异常（Overflow Exception）；
- 非精确异常（Inexact Exception）。

4.2.18.1 非规格化数输入异常

C860 浮点单元不支持非规划化数运算，当输入非规格化数时，IDC 标志位（FESR[5]）置 1。具体异常处理方式分为非规划化数输入异常使能和不使能两种情况：

非规划化数输入异常使能（IDE 为 1）时，处理器将终止正常的程序执行并抛出 30 号异常，同时 FEC 位（FESR[7]）和 IDC 位（FESR[5]）置 1，供异常服务程序查询与处理，此次计算结果不回写。

非规划化数输入异常不使能（IDE 为 0）时，浮点单元会根据 FM 位将非规格数替换成一个近似数。如果 FM 位为 1，则非规格数将会被刷成一个最小的浮点数，符号和原先保持不变；如果 FM 为 0，则非规格数将会被刷成 0，符号和原先相同。之后浮点单元会使用这个近似数完成指令运算，并将运算结果写回到寄存器中。浮点异常状态寄存器的 FEA（FESR[15]）、IDA（FESR[13]）、FEC（FESR[7]）和 IDC（FESR[5]）会被置 1。

4.2.18.2 非法操作异常

当输入操作数非法时，非法操作异常将被触发。如下表格所示为可以产生非法操作异常的运算和相应的操作数，非法操作异常的处理方式分为异常使能与异常不使能两种情况。

表 4.2: 异常优先级

指令类型	非法操作异常
浮点运算指令（除取绝对值/取反/传输类）	操作数中包含 SNaN
加法指令	(+INF) + (-INF) 或 (-INF) + (+INF)
减法指令	(+INF) - (+INF) 或 (-INF) - (-INF)
比较指令	任意的 NaN 操作数
浮点转整型/定点指令	任意 NaN 操作数或者结果发生上/下溢出
乘法指令	Zero×INF
除法指令	Zero/Zero 或 INF/INF
开方指令	操作数小于 0（包含-0）

注：如果 FM 位是 0 并且输入的操作数是非规格数（Denormalized Number）时，非规格化数将会被刷成 0，也有可能满足 IO 异常条件。

非法操作异常使能（IOE 为 1）时，处理器将会终止执行并产生 30 号异常，同时设置 FEC（FESR[7]）和 IOC（FESR[0]）为 1，供异常服务程序查询与处理，此次计算结果不回写。

非法操作异常不使能（IOE 为 0）时，浮点运算指令（除浮点转整型/定点操作之外）将会产生 qNaN 作为运算结果并回写至目的寄存器。FEA（FESR[15]）、IOA（FESR[8]）、FEC（FESR[7]）以及 IOC（FESR[0]）会被置 1。对于浮点转整型/定点指令，当发生非法操作时的默认结果如下表所示。

表 4.3: 浮点转整型/定点非法输入的计算结果

指令类型	舍入后的输入值	默认结果
浮点转有符号整型/定点	NaN +INF or $x > 2^{31}$ -INF or $x < -2^{31}$	0x00000000 0x7fffffff 0x80000000
浮点转无符号整型/定点	NaN +INF or $x > 2^{31}$ -INF or $x < -2^{31}$	0x00000000 0xffffffff 0x00000000

4.2.18.3 除零异常

当被除数是一个非零数而除数是 0 时，将产生除零异常。当除数是一个非规格化数，而 FM 位是 0，此时除数将会被刷为 0，同样也会引起除零异常。除零异常的处理方式分为异常使能与异常不使能两种情况。

除零异常使能（DZE 为 1）时，处理器将会终止执行并产生 30 号异常，FEC（FESR[7]）和 DZC（FESR[1]）将会置 1，供异常服务程序查询与处理，此次计算结果不回写。

除零异常不使能（DZE 为 0）时，将会向目标寄存器写入一个带符号的无穷大。FEA（FESR[15]）、DZA（FESR[9]）、FEC（FESR[7]）以及 DZC（FESR[1]）会被置 1。

4.2.18.4 上溢异常

浮点单元在舍入操作之后检测运算结果是否发生上溢异常。当结果超过了相应格式的最大有限数时，将会产生上溢异常。根据 IEEE754 标准，发生上溢出异常时，结果将不能被准确的表示出来，因此非精确异常也会被置起，无论此次运算的 GRS 是何值。上溢异常的处理方式分为异常使能与异常不使能两种情况。

上溢异常使能 (OFE 为 1) 时，处理器将会终止执行并产生 30 号异常，同时设置 FEC (FESR[7])、IXC (FESR[4]) 和 OFC (FESR[2]) 为 1，此次计算结果不回写。

上溢异常不使能 (OFE 为 0) 时，浮点单元根据不同的舍入模式，将舍入后的运算结果写回目标寄存器。FEA (FESR[15])、IXA (FESR[12])、OFA (FESR[10])、FEC (FESR[7])、IXC (FESR[4]) 和 OFC (FESR[2]) 会被置 1。

4.2.18.5 下溢异常

IEEE754 标准规定了两种下溢异常检测方式，分别为：舍入前检测和舍入后检测。C860 浮点单元选用舍入后检测方式，即对运算结果舍入之后进行下溢异常的判定。跟上溢异常一样，当发生下溢异常时，结果将不能被准确的表示出来，非精确异常也会被置起。下溢异常的处理方式分为异常使能与异常不使能两种情况。

下溢异常使能 (UFE 为 1) 时，处理器将会终止执行并产生 30 号异常，同时置 FEC (FESR[7])、IXC (FESR[4]) 和 UFC (FESR[3]) 为 1，此次计算结果不回写。

下溢异常使能 (UFE 为 0) 时，若 FM 位为 1 时，结果将被刷成同符号的相应格式的最小有限数，否则结果将会是一个带相同符号的 0。FEA (FESR[15])、IXA (FESR[12])、UFA (FESR[11])、FEC (FESR[7])、IXC (FESR[4]) 和 UFC (FESR[3]) 会被置 1。

4.2.18.6 非精确异常

当计算结果不能被准确表示时，会发生非精确异常，以下情况会导致非精确异常：

- 发生上溢异常
- 发生下溢异常
- 舍入前后的结果不一致

注意：非精确异常在浮点运算中出现频繁。如果非精确异常使能位 (IXE) 打开，将会极大地降低浮点运算性能。如果应用本身对精度要求不高，推荐关闭精确异常使能位。

非精确异常使能 (IXE 为 1) 时，处理器将会终止执行并产生 30 号异常，同时置 FEC (FESR[7]) 和 IXC (FESR[4]) 为 1，此次计算结果不回写。

非精确异常使能 (IXE 为 0) 时，浮点单元将根据舍入模式，对浮点运算结果进行舍入处理，并将舍入之后的结果回写到目的寄存器。FEA (FESR[15])、IXA (FESR[12])、FEC (FESR[7])、IXC (FESR[4]) 会被置 1。

4.2.18.7 特殊指令的异常

对于不需要保留乘法全部精度的普通乘累加/乘累减指令，将会被拆分成两条指令执行，浮点单元视作两次独立的浮点运算操作，浮点异常状态寄存器将记录乘法操作与加减法操作过程中发生的所有异常，以最大限度的帮助软件记录与进行异常处理。

对于单指令多数据（矢量）类指令，浮点单元视为若干并行的浮点运算操作，浮点异常状态寄存器将记录并行运算过程中发生的所有异常，以最大限度的帮助软件记录与进行异常处理。譬如，VADD.F32 指令，支持四路单精度加法并行运算，其中第一路加法运算发生上溢出异常，第二路加法运算发生非精确异常，浮点异常状态寄存器将记录这条指令发生了上溢出和非精确异常。

4.3 异常优先级

如表 4.4 所示，根据异常的特性和被处理的先后关系，C860 把优先级分为 10 级。在表 4.4 中，1 代表最高优先级，10 代表了最低优先级。值得注意的是，在第 9 组中，几个异常共享一个优先级，因为它们之间相互有排斥性。

在 C860 里，多个异常可以同时发生。重启异常是很特别的，它有最高的优先级。

所有其它的异常按表 4.4 中的优先级关系进行处理。

如果 PSR（EE）被清零了，当异常发生时，处理器处理的是不可恢复异常。

如果多个异常同时发生，拥有最高优先级的异常最先被处理。处理器在异常返回后，重新执行产生异常的指令时，其余的异常可以重现。

表 4.4: 异常优先级

优先级	异常与它相关的优先级	特征
1	重启异常	处理器中止所有程序运行，初始化系统。
2	待处理的跟踪异常	如果 TP=1，在 rte 或 rfi 指令退休后，处理器处理待处理的跟踪异常。
3	硬断点请求	在相关的指令退休后，硬断点请求被处理。
4	Idly4 错误	在相关的指令退休后，处理器保存上下文并处理异常。
5	不对齐错误	在相关的指令退休后，处理器保存上下文并处理异常。
6.0 6.1	快速中断 普通中断	如果 IC 为 0，中断在指令退休后被响应； 如果 IC 为 1，处理器允许中断在指令完成之前就被响应。
7.0 7.1 7.2 7.3 7.4	不可恢复错误异常/TLB 不可恢复异常 TLB 读无效异常/TLB 写无效异常 TLB 修改异常 TLB 失配异常 访问错误	在相关的指令退休后，处理器保存上下文并处理异常。
8	非法指令 特权异常 禁止硬件加速器 除以零 陷阱指令 断点指令 浮点异常	在相关的指令退休后，处理器保存上下文并处理异常。
9	跟踪异常	在相关的指令退休后，处理器保存上下文并处理异常。

4.3.1 发生待处理的异常时调试请求

处理器如果在异常发生的同时接收到了调试请求信号，先进入调试模式。异常延后处理，直到处理器退出调试模式和产生异常的指令重新被执行。

4.4 异常返回

根据正在处理的异常类型，处理器通过执行 RFI 或 RTE 指令从异常服务程序中返回。RFI 指令利用保存在 FPSR 和 FPC 影子寄存器中的上下文从异常服务程序中返回；RTE 指令利用保存在 EPSR 和 EPC 影子寄存器中的上下文从异常服务程序中返回。

第五章 指令集

5.1 32 位指令

本章主要介绍 C860 实现的玄铁 CPU V2 的 32 位指令集。

5.1.1 32 位指令功能分类

C860 实现的玄铁 CPUV2 的 32 位指令集按照指令实现的功能来划分，可以分为：

- 数据运算指令
- 分支跳转指令
- 内存存取指令
- 特权指令
- 特殊功能指令

5.1.1.1 数据运算指令

数据运算类指令可以进一步分为：加减法指令、数据类型转换指令、立即数生成指令、逻辑操作指令、移位指令、比较指令、数据传输指令、比特操作指令、乘除法指令、杂类运算指令。

表 5.1: 32 位加减法指令列表

加减法指令集	
ADDU32	无符号加法指令
ADDC32	无符号带进位加法指令
ADDI32	无符号立即数加法指令
SUBU32	无符号减法指令
SUBC32	无符号带借位减法指令
SUBI32	无符号立即数减法指令
RSUB32	反向减法指令
IXH32	索引半字指令

下页继续

表 5.1 – 续上页

IXW32	索引字指令
IXD32	索引双字指令
INCF32	C 为 0 立即数加法指令
INCT32	C 为 1 立即数加法指令
DECF32	C 为 0 立即数减法指令
DECT32	C 为 1 立即数减法指令
DECGT32	减法大于零置 C 位指令
DECLT32	减法小于零置 C 位指令
DECNE32	减法不等于零置 C 位指令
FADD.T	单精度/双浮点数加法指令
FSUB.T	单精度/双精度浮点数减法指令
矢量浮点扩展指令集	
VADD.T	矢量单精度浮点数加法指令
VSUB.T	矢量单精度浮点数减法指令
VPADD.T	矢量单精度浮点数耦合加法指令
VASX.T	矢量单精度浮点数交叉加减指令
VSAX.T	矢量单精度浮点数交叉减加指令
VSABS.T	矢量单精度浮点数相减取绝对值指令
矢量多媒体扩展指令集	
VADD.T	矢量加法指令
VADD.T.S	矢量饱和加法指令
VADD.T.H	矢量加法取高半指令
VADD.T.RH	矢量加法取高半舍入指令
VADD.T.E	矢量扩展加法指令
VADD.T.X	矢量非对齐加法指令
VADDDH.T	矢量加法平均指令
VADDDH.T.R	矢量加法平均并舍入指令
VPADD.T	矢量链加法指令
VPADD.T.S	矢量饱和链加法指令
VPADD.T.E	矢量扩展链加法指令
VPADDA.T.E	矢量扩展链累法指令
VASX.T.S	矢量错位加减指令
VSAX.T.S	矢量错位减加指令
VASXH.T	矢量错位加减平均指令
VSAXH.T	矢量错位减加平均指令
VADDDH.T	矢量加法平均指令
VADDDH.T.R	矢量加法平均并舍入指令
VSUB.T	矢量减法指令

下页继续

表 5.1 – 续上页

VSUB.T.S	矢量饱和减法指令
VSUB.T.H	矢量减法取高半指令
VSUB.T.RH	矢量减法取高半舍入指令
VSUB.T.E	矢量扩展减法指令
VSUB.T.X	矢量非对齐减法指令
VSUBH.T.R	矢量减法平均并舍入指令
VSUBH.T	矢量减法平均指令
VABS.T	矢量取绝对值指令
VABS.T.S	矢量饱和取绝对值指令
VSABS.T.S	矢量饱和减法取绝对值指令
VSABS.T.E	矢量扩展减法取绝对值指令
VSABSA.T	矢量减法取绝对值并累加指令
VSABSA.T.E	矢量扩展减法取绝对值并累加指令
VNEG.T	矢量取反指令
VNEG.T.S	矢量饱和取反指令

表 5.2: 32 位数据类型转换指令列表

数据类型转换指令	
FSTOD	单精度浮点数转换成双精度浮点数指令
FDTOS	双精度浮点数转换成单精度浮点数指令
FFTOX.T1.T2	浮点定点转换指令
FXTOF.T1.T2	定点浮点转换指令
FFTOL.T1.T2	浮点整型转换指令
FITOF.T1.T2	整型浮点转换指令
FFTOL.T1.T2.RM	带舍入模式的浮点整型转换指令
FFTOFL.T.RM	带舍入模式的浮点数转换为整型浮点指令。
矢量浮点扩展指令集	
VDTOS	矢量双精度浮点单精度浮点转换指令
VFTOFL.T.RM	矢量带舍入模式的浮点舍弃小数指令
VFTOL.T1.T2	矢量浮点符号整数转换指令
VFTOL.T1.T2.RM	矢量带舍入模式的浮点有符号整数转换指令
VFTOX.T1.T2	矢量浮点定点转换指令
VITOF.T1.T2	矢量整数浮点转换指令
VXTOF.T1.T2	矢量定点浮点转换指令

表 5.3: 32 位立即数生成指令

向量多媒体扩展指令集	
VMOVL.8	向量立即数传输指令（按字节元素）
VMOVL.16	向量 (有/无符号) 立即数传输指令（半字元素）
VMOVL.32	向量 (有/无符号) 立即数传输指令（按字元素）
VMASKI.8.L	向量低半部分立即数扩展传输指令（字节元素）
VMASKI.8.H	向量高半部分立即数扩展传输指令（字节元素）
VMASKI.16	向量立即数扩展传输指令（按半字元素）

表 5.4: 32 位逻辑操作指令列表

逻辑操作指令	
AND32	按位与指令
ANDI32	立即数按位与指令
ANDN32	按位非与指令
ANDNI32	立即数按位非与指令
OR32	按位或指令
ORI32	立即数按位或指令
XOR32	按位异或指令
XORI32	立即数按位异或指令
NOR32	按位或非指令
NOT32	按位非指令
向量多媒体扩展指令集	
VAND.T	向量与指令
VANDN.T	向量与非指令
VXOR.T	向量异或指令
VOR.T	向量或指令
VORN.T	向量或非指令
VNOT.T	向量取反指令
VTST.T	向量按位与置位指令
VSEL.T	向量位选择指令
VCLS.T	向量符号位连续相同指令
VCLZ.T	向量最高位连续 0 个数指令
VCNT1.T	向量数据 1 个数指令

表 5.5: 32 位移位指令列表

移位指令	
LSL32	逻辑左移指令

下页继续

表 5.5 – 续上页

LSLI32	立即数逻辑左移指令
LSLC32	立即数逻辑左移至 C 位指令
LSR32	逻辑右移指令
LSRI32	立即数逻辑右移指令
LSRC32	立即数逻辑右移至 C 位指令
ASR32	算术右移指令
ASRI32	立即数算术右移指令
ASRC32	立即数算术右移至 C 位指令
ROTL32	循环左移指令
ROTLI32	立即数循环左移指令
XSR32	扩展右移指令
矢量多媒体扩展指令集	
VSHT.T	矢量移位指令
VSHT.T.S	矢量移位饱和指令
VSHT.T.R	矢量移位舍入指令
VSHT.T.RS	矢量移位饱和舍入指令
VSHL.T	矢量寄存器左移指令
VSHL.T.S	矢量寄存器左移饱和指令
VSHLI.T	矢量立即数左移指令
VSHLI.T.S	矢量立即数左移饱和指令
VSHLI.T.E	矢量扩展立即数左移指令
VSHR.T	矢量寄存器右移指令
VSHR.T.R	矢量寄存器右移舍入指令
VSHRI.T	矢量立即数右移指令
VSHRI.T.R	矢量立即数左移舍入指令
VSHRI.T.L	矢量立即数右移取元素低半部分指令
VSHRI.T.LR	矢量立即数右移舍入取元素低半部分指令
VSHRI.T.LS	矢量立即数右移取元素低半部分饱和指令
VSHRI.T.LRS	矢量立即数右移舍入取元素低半部分饱和指令
VSHRIA.T	矢量立即数右移累加指令
VSHRIA.T.R	矢量立即数右移舍入累加指令
VEXH.T	矢量低位数据插入指令
VEXL.T	矢量高位数据插入指令

表 5.6: 32 位比较指令列表

比较指令	
CMPNE32	不等比较指令
CMPNEI32	立即数不等比较指令

下页继续

表 5.6 – 续上页

CMPHS32	无符号大于等于比较指令
CMPHSI32	立即数无符号大于等于比较指令
CMPLT32	有符号小于比较指令
CMPLTI32	立即数有符号小于比较指令
FCMPHS.T	浮点大于等于比较指令
FCMPHSZ.T	浮点大于等于零比较指令
FCMPHZ.T	浮点大于零比较指令
FCMPLSZ.T	浮点小于等于零比较指令
FCMPLT.T	浮点小于比较指令
FCMPLTZ.T	浮点小于零比较指令
FCMPNE.T	浮点不等于比较指令
FCMPNEZ.T	浮点不等于零比较指令
FCMPUO.T	浮点双操作数非数判断指令
FCMPUOZ.T	浮点单操作数非数判断指令
FMAXNM.T	浮点规范最大值选择指令
FMINNM.T	浮点规范最小值选择指令
矢量浮点扩展指令集	
VCMPHS.T	矢量浮点大于等于比较指令
VCMPHSZ.T	矢量浮点大于等于零比较指令
VCMPHZ.T	矢量浮点大于零比较指令
VCMPLSZ.T	矢量浮点小于等于零比较指令
VCMPLT.T	矢量浮点小于比较指令
VCMPLTZ.T	矢量浮点小于零比较指令
VCMPNE.T	矢量浮点不等于比较指令
VCMPNEZ.T	矢量浮点不等于零比较指令
VMAX.T	矢量浮点最大值选择指令
VMAXNM.T	矢量浮点规范最大值选择指令
VMIN.T	矢量浮点最小值选择指令
VMINNM.T	矢量浮点规范最小值选择指令
VPMAX.T	矢量浮点相邻最大值选择指令
VPMIN.T	矢量浮点相邻最小值选择指令
矢量多媒体扩展指令集	
VCMPHS.T	矢量大于等于比较指令
VCMPLT.T	矢量小于比较指令
VCMPNE.T	矢量不等于比较指令
VCMPHSZ.T	矢量大于等于 0 比较指令
VCMPLTZ.T	矢量小于 0 比较指令
VCMPNEZ.T	矢量不等于 0 比较指令

下页继续

表 5.6 – 续上页

VCMPHZ.T	矢量大于 0 比较指令
VCMPLSZ.T	矢量小于等于 0 比较指令
VMAX.T	矢量最大值选择指令
VMIN.T	矢量最小值选择指令
VPMAX.T	矢量相邻最大值选择指令
VPMIN.T	矢量相邻最小值选择指令
VCLIP.T	矢量裁剪饱和指令

表 5.7: 32 位数据传输指令列表

数据传输指令	
MOV32	数据传送指令
MOVF32	C 为 0 数据传送指令
MOVT32	C 为 1 数据传送指令
MOVI32	立即数数据传送指令
MOVIH32	立即数高位数据传送指令
MVCV32	C 位取反传送指令
MVC32	C 位传送指令
CLRF32	C 为 0 清零指令
CLRT32	C 为 1 清零指令
LRW32	存储器读入指令
GRS32	符号产生指令
FMTVR.64	浮点寄存器双字写传送指令
FMTVR.32.1	浮点寄存器低字写传送指令
FMTVR.32.2	浮点双寄存器低字写传送指令
FMFVR.64	浮点寄存器双字读传送指令
FMFVR.32.1	浮点寄存器低字读传送指令
FMFVR.32.2	浮点双寄存器低字读传送指令
FMFVRH	浮点寄存器高字读传送指令
FMTVRH	浮点寄存器高字写传送指令
FMOV.32	浮点寄存器间低字传送指令
FMOV.64	浮点寄存器间双字传送指令
FMOVX.32	浮点寄存器间低字低半字传送指令
FINS.32	浮点寄存器间低字高半字传送指令
FSEL.32	C 位选择低字传送指令
FSEL.64	C 位选择双字传送指令
矢量多媒体扩展指令集	
VMTVR.T.1	矢量写传送指令
VMTVR.T.2	矢量写传送指令

下页继续

表 5.7 – 续上页

VMFVR.T	矢量写传送指令
VSEXT.T	矢量数据符号位读传送指令
VDUPG.T	通用源矢量目的寄存器之间的拷贝指令
VMOV.T.E	矢量扩展传输指令
VMOV.T.L	矢量低位传输指令
VMOV.T.H	矢量高位传输指令
VMOV.T.SL	矢量低位饱和传输指令
VMOV.T.RH	矢量高位 round 传输指令
VLRW.32.N	矢量存储器读入指令

表 5.8: 32 位比特操作指令列表

比特操作指令	
BCLRl32	立即数位清零指令
BSETl32	立即数位位置指令
BTSTl32	立即数位测试指令

表 5.9: 32 位提取插入指令列表

提取插入指令	
ZEXT32	位提取并无符号扩展指令
SEXT32	位提取并有符号扩展指令
INS32	位插入指令
ZEXTB32	字节提取并无符号扩展指令
ZEXTH32	半字提取并无符号扩展指令
SEXTB32	字节提取并有符号扩展指令
SEXTH32	半字提取并有符号扩展指令
XTRB0.32	提取字节 0 并无符号展指令
XTRB1.32	提取字节 1 并无符号扩展指令
XTRB2.32	提取字节 2 并无符号扩展指令
XTRB3. 32	提取字节 3 并无符号扩展指令
BREV32	位倒序指令
REVB32	字节倒序指令
REVBH32	半字内字节倒序指令
矢量多媒体扩展指令集	
VSWP.T	矢量数据交换指令
VTRN.T	矢量数据交叉传输指令
VREVBH	矢量数据半字倒序指令
VREVW	矢量数据字倒序指令

下页继续

表 5.9 – 续上页

VREVD	矢量数据双字倒序指令
VREVQ	矢量数据字节倒序指令
VEXTL.T	立即数矢量数据拼接指令
VEXT.T	寄存器矢量数据拼接指令
VTBL.T	矢量数据链接指令
VTBX.T	矢量数据链接指令
VDUP.T.1	一元矢量源目的寄存器之间的拷贝指令
VDUP.T.2	二元矢量源目的寄存器之间的拷贝指令
VDUP.T.3	三元矢量源目的寄存器之间的拷贝指令
VDUP.T.4	四元矢量源目的寄存器之间的拷贝指令
VINS.T.1	一元矢量插入指令
VINS.T.2	二元矢量插入指令
VINS.T.3	三元矢量插入指令
VINS.T.4	四元矢量插入指令
VPKG.T.2	二元封装指令
VPKG.T.3	三元封装指令
VPKG.T.4	四元封装指令
VITL.T.2	二元交织指令
VDTL.T.2	二元去交织指令

表 5.10: 32 位乘除法指令列表

指令名称	指令描述
MULSW32	有符号数乘法指令
MULT32	乘法指令
MULSH32	16 位有符号乘法指令
MULA.32.L	32 位有符号乘累加区低 32 位指令
MULALL.S16.S	带饱和操作的 16 位有符号低半字乘累加指令
MUL.U32	32 位无符号乘法指令
MULA.U32	32 位无符号乘累加法指令
MUL.S32	32 位有符号乘法指令
MULA.S32	32 位有符号乘累加法指令
DIVU32	无符号除法指令
DIVS32	有符号除法指令
DIVUL	无符号长除法指令
DIVSL	有符号长除法指令
FMUL.T	浮点乘法指令
FDIV.T	浮点除法指令
FNMUL.T	浮点乘法取负指令

下页继续

表 5.10 – 续上页

指令名称	指令描述
FMULA.T	浮点乘累加指令 (乘法结果舍入后累加)
FMULS.T	浮点乘取负累加指令 (乘法结果舍入后累减)
FNMULA.T	浮点乘取负累减指令 (乘法结果舍入后累加)
FNMULS.T	浮点乘累减指令 (乘法结果舍入后累减)
FFMULA.T	浮点融合乘累加指令 (乘法结果保留全部精度累加)
FFMULS.T	浮点融合乘取负累加指令 (乘法结果保留全部精度累减)
FFNMULA.T	浮点融合乘取负累减指令 (乘法结果保留全部精度累加)
FFNMULS.T	浮点融合乘累减指令 (乘法结果保留全部精度累减)
FRECIP.T	浮点倒数指令
FSQRT.T	浮点平方根指令
矢量浮点扩展指令集	
VMUL.T	矢量浮点乘法指令
VMULI.T	矢量浮点索引乘法指令
VMULA.T	矢量浮点乘累加指令 (乘法结果舍入后累加)
VMULAI.T	矢量浮点索引乘累加指令 (乘法结果舍入后累加)
VMULS.T	矢量浮点乘累减指令 (乘法结果舍入后累减)
VMULSI.T	矢量浮点索引乘累减指令 (乘法结果舍入累减)
VFMULA.T	矢量浮点融合乘累加指令 (乘法结果保留全部精度累加)
VFMULS.T	矢量浮点融合乘累减指令 (乘法结果保留全部精度累减)
VFMULXAA.T	矢量浮点复数乘累加实部虚部部分计算 (乘法保留全部精度累加)
VFMULXAAI.T	矢量浮点索引复数乘累加实部虚部部分计算 (乘法保留全部精度累加)
VFMULXAS.T	矢量浮点复数乘累加实部虚部部分计算 (乘法保留全部精度累加)
VFMULXASI.T	矢量浮点索引复数乘累加实部虚部部分计算 (乘法保留全部精度累加)
VFMULXSS.T	矢量浮点复数乘累减实部虚部部分计算 (乘法保留全部精度累减)
VFMULXSSI.T	矢量浮点索引复数乘累减实部虚部部分计算 (乘法保留全部精度累减)
VFMULXSA.T	矢量浮点复数乘累减实部虚部部分计算 (乘法保留全部精度累减)
VFMULXSAI.T	矢量浮点索引复数乘累减实部虚部部分计算 (乘法保留全部精度累减)
VFCMUL.T	矢量浮点复数乘法 (乘法结果舍入后累加)
VFCMULA.T	矢量浮点复数乘累加 (乘法结果舍入后累加)
VFCMULC.T	矢量浮点复数共轭乘法 (乘法结果舍入后累加)
VFCMULCA.T	矢量浮点复数共轭乘累加 (乘法结果舍入后累加)
VFCMULN.T	矢量浮点复数取负乘法 (乘法结果舍入后累加)
VFCMULNA.T	矢量浮点复数取负乘累加 (乘法结果舍入后累加)
VFCMULCN.T	矢量浮点复数共轭取负乘法 (乘法结果舍入后累加)
VFCMULCNA.T	矢量浮点复数共轭取负乘累加 (乘法结果舍入后累加)
VRECPS.T	矢量浮点倒数逼近
VRECPE.T	矢量浮点倒数指令

下页继续

表 5.10 – 续上页

指令名称	指令描述
VRSQRTE.T	矢量浮点倒数开方
VRSQRTS.T	矢量浮点开方逼近
VEXPE.T	矢量浮点快速 e 指数计算
矢量多媒体扩展指令集	
VMUL.T	矢量乘法指令
VMULI.T	矢量带索引取高半乘法指令
VMUL.T.H	矢量取高半乘法指令
VMUL.T.E	矢量扩展乘法指令
VMULI.T.E	矢量扩展带索引乘法指令
VMULA.T	矢量乘累加指令
VMULAI.T	矢量带索引乘累加指令
VMULA.T.E	矢量扩展乘累加指令
VMULAI.T.E	矢量扩展带索引乘累加指令
VMULS.T	矢量乘累减指令
VMULSI.T	矢量带索引乘累减指令
VMULS.T.E	矢量扩展乘累减指令
VMULSI.T.E	矢量扩展带索引乘累减指令
VMULACA.T	矢量链乘累加指令
VMULACA.T	矢量链带索引乘累加指令
VMULACAA.T	矢量链乘累加加指令
VMULACAAI.T	矢量链带索引乘累加加指令
VRMUL.T.SE	矢量扩展带饱和和小数乘法指令
VRMULI.T.SE	矢量带索引扩展带饱和和小数乘法指令
VRMULH.T.S	矢量带饱和和取高半小数乘法指令
VRMULHI.T.S	矢量带索引带饱和和取高半小数乘法指令
VRMULH.T.RS	矢量带饱和和取高半舍入小数乘法指令
VRMULHI.T.RS	矢量带索引带饱和和取高半舍入小数乘法指令
VRMULHA.T.RS	矢量带饱和和取高半舍入小数乘累加指令
VRMULHAI.T.RS	矢量带索引带饱和和取高半舍入小数乘累加指令
VRMULHS.T.RS	矢量带饱和和取高半舍入小数乘累减指令
VRMULHSI.T.RS	矢量带索引带饱和和取高半舍入小数乘累减指令
VRMULSHR.T.E	矢量扩展带移位小数乘法指令
VRMULSHRI.T.E	矢量带索引扩展带移位小数乘法指令
VRMULSA.T.E	矢量扩展带移位小数乘累加指令
VRMULSAI.T.E	矢量带索引扩展带移位小数乘累加指令
VRMULSS.T.E	矢量扩展带移位小数乘负累加指令
VRMULSSI.T.E	矢量带索引扩展带移位小数乘负累加指令

下页继续

表 5.10 – 续上页

指令名称	指令描述
VRMULXAA.T.RS	矢量带饱和和复数实部虚部交叉相乘累加累加取高半舍入指令
VRMULXAAL.T.RS	矢量带索引带饱和和复数实部虚部交叉相乘累加累加取高半舍入指令
VRMULXAS.T.RS	矢量带饱和和复数实部虚部交叉相乘累加累减取高半舍入指令
VRMULXASI.T.RS	矢量带索引带饱和和复数实部虚部交叉相乘累加累减取高半舍入指令
VRMULXSS.T.RS	矢量带饱和和复数实部虚部交叉相乘累减累减取高半舍入指令
VRMULXSSI.T.RS	矢量带索引带饱和和复数实部虚部交叉相乘累减累减取高半舍入指令
VRMULXSA.T.RS	矢量带饱和和复数实部虚部交叉相乘累减累加取高半舍入指令
VRMULXSAI.T.RS	矢量带索引带饱和和复数实部虚部交叉相乘累加累减取高半舍入指令
VRCMUL.T.RS	矢量带饱和和复数乘法取高半舍入指令
VRCMULA.T.E	矢量复数乘法右移扩位累加指令
VRCMULC.T.RS	矢量带饱和和复数共轭乘法取高半舍入指令
VRCMULCA.T.E	矢量共轭复数乘法右移扩位累加指令
VRCMULN.T.RS	矢量带饱和和复数取负乘法取高半舍入指令
VRCMULNA.T.E	矢量复数取负乘法右移扩位累加指令
VRCMULCN.T.RS	矢量带饱和和复数共轭取负乘法取高半舍入指令
VRCMULCNA.T.E	矢量共轭复数取负乘法右移扩位累加指令

表 5.11: 32 位杂类运算指令列表

杂类运算指令	
ABS32	绝对值指令
FF0.32	快速找 0 指令
FF1.32	快速找 1 指令
BMASKI32	立即数位屏蔽产生指令
BGENR32	寄存器位产生指令
BGENI32	立即数位产生指令
矢量浮点扩展指令集	
VRECPE.T	矢量倒数指令
VRECPS.T	矢量倒数逼近指令
VRSQRTE.T	矢量倒数开方指令
VRSQRTS.T	矢量倒数开方逼近指令
VEXPE.T	矢量 e 指数指令

5.1.1.2 分支跳转指令

分支跳转指令可以进一步分为：分支指令、跳转指令

表 5.12: 32 位分支指令列表

分支指令	
BT32	C 为 1 分支指令
BF32	C 为 0 分支指令
BEZ32	寄存器等于零分支指令
BNEZ32	寄存器不等于零分支指令
BNEZAD32	寄存器自减大于零分支指令
BHZ32	寄存器大于零分支指令
BLSZ32	寄存器小于等于零分支指令
BLZ32	寄存器小于零分支指令
BHSZ32	寄存器大于等于零分支指令

表 5.13: 32 位跳转指令列表

跳转指令	
BR32	无条件跳转指令
BSR32	跳转到子程序指令
JMPI32	间接跳转指令
JSRI32	间接跳转到子程序指令
JMP32	寄存器跳转指令
JSR32	寄存器跳转到子程序指令
RTS32	链接寄存器跳转指令

5.1.1.3 内存存取指令

内存存取指令可以进一步分为：立即数偏移存取指令、向量寄存器偏移存取指令、多寄存器存取指令、独占存取指令、符号存取指令。

表 5.14: 32 位立即数偏移存取指令列表

立即数偏移存取指令	
LD32.B	无符号扩展字节加载指令
LD32.BS	有符号扩展字节加载指令
LD32.H	无符号扩展半字加载指令
LD32.HS	有符号扩展半字加载指令
LD32.W	字加载指令
LD32.D	双字加载指令
FLD.T	单/双精度浮点加载指令
ST32.B	字节存储指令
ST32.H	半字存储指令
ST32.W	字存储指令
ST32.D	双字存储指令
FST.T	单/双精度浮点存储指令
矢量多媒体扩展指令集	
VLD.T.N	固定长度矢量加载指令
VLDRU.T.N	矢量加载基址跳跃更新指令
VLDU.T.N	矢量加载基址更新指令
VLDX.T	可变长度矢量加载指令
VST.T.N	固定长度矢量存储指令
VSTRU.T.N	矢量存储基址跳跃更新指令
VSTU.T.N	矢量存储基址更新指令
VSTX.T	可变长度矢量存储指令

表 5.15: 32 位向量寄存器偏移存取指令列表

向量寄存器偏移存取指令	
LDR32.B	寄存器移位寻址无符号扩展字节加载指令
LDR32.BS	寄存器移位寻址有符号扩展字节加载指令
LDR32.H	寄存器移位寻址无符号扩展半字加载指令
LDR32.HS	寄存器移位寻址有符号扩展半字加载指令
LDR32.W	寄存器移位寻址字加载指令
LDR32.D	寄存器移位寻址双字加载指令
FLDR.T	寄存器移位寻址单、双精度浮点加载指令
STR32.B	寄存器移位寻址字节存储指令
STR32.H	寄存器移位寻址半字存储指令
STR32.W	寄存器移位寻址字存储指令
STR32.D	寄存器移位寻址双字存储指令
FSTR.T	寄存器移位寻址单、双精度浮点存储指令

表 5.16: 32 位多寄存器存取指令列表

多寄存器存取指令	
LDQ32	连续四字加载指令
LDM32	连续多字加载指令
FLDM.T	连续单/双精度浮点加载指令
FLDMU.T	连续单/双精度浮点加载指令地址自增指令
STQ32	连续四字存储指令
STM32	连续多字存储指令
FSTM.T	连续单/双精度浮点存储指令
FSTMU.T	连续单/双精度浮点存储地址自增指令
PUSH32	压栈指令
POP32	出栈指令
矢量多媒体扩展指令集	
VLDM.T	连续矢量加载指令
VLDMRU.T	跳跃矢量加载基址更新指令
VLDMU.T	连续矢量加载基址更新指令
VSTM.T	连续矢量存储指令
VSTMURU.T	跳跃矢量存储基址更新指令
VSTMU.T	连续矢量存储基址更新指令

表 5.17: 32 位独占存取指令列表

独占存取指令	
LDEX32.W	独占式字加载指令
STEX32.W	独占式字存储指令

表 5.18: 32 位符号存取指令列表

符号存取指令	
LRS32.B	字节符号加载指令
LRS32.H	半字节符号加载指令
LRS32.W	字符符号加载指令
SRS32.B	字节符号存储指令
SRS32.H	半字节符号存储指令
SRS32.W	字符符号存储指令

5.1.1.4 特权指令

特权指令可以进一步分为：控制寄存器操作指令、低功耗指令、CACHE 操作指令、异常返回指令。

表 5.19: 32 位控制寄存器操作指令列表

控制寄存器操作指令	
MFCR32	控制寄存器读传送指令
MTCR32	控制寄存器写传送指令
PSRSET32	PSR 位置位指令
PSRCLR32	PSR 位清零指令

表 5.20: 32 位低功耗指令列表

低功耗指令	
WAIT32	进入低功耗等待模式指令
DOZE32	进入低功耗睡眠模式指令
STOP32	进入低功耗暂停模式指令

表 5.21: 32 位低功耗指令列表

CACHE 操作指令	
ICACHE32	指令高缓操作指令
DCACHE32	数据高缓操作指令
TLBI32	TLB 无效指令
L2CACHE32	二级高缓操作指令

表 5.22: 32 位异常返回指令列表

异常返回指令	
RTE32	异常和普通中断返回指令
RFI32	快速中断返回指令

5.1.1.5 特殊功能指令

表 5.23: 32 位特殊功能指令列表

特殊功能指令	
SYNC32	CPU 同步指令
BAR32	内存同步指令
BKPT32	断点指令
IDLY32	中断识别禁止指令
TRAP32	无条件操作系统陷阱指令
PLDR32	读数据预取指令
PLDW32	写数据预取指令

5.2 16 位指令

本节主要介绍玄铁 CPU V2 的 16 位指令集。

5.2.1 16 位指令功能分类

玄铁 CPU V2 的 16 位指令集按照指令实现的功能来划分，可以分为：

- 数据运算指令
- 分支跳转指令
- 内存存取指令

5.2.1.1 数据运算指令

数据运算类指令可以进一步分为：加减法指令、逻辑操作指令、移位指令、比较指令、数据传输指令、比特操作指令、提取插入指令、乘除法指令。

表 5.24: 16 位加减法指令列表

加减法指令	
ADDU16	无符号加法指令
ADDC16	无符号带进位加法指令
ADDI16	无符号立即数加法指令
SUBU16	无符号减法指令
SUBC16	无符号带借位减法指令
SUBI16	无符号立即数减法指令

表 5.25: 16 位逻辑操作指令列表

逻辑操作指令	
AND16	按位与指令
ANDN16	按位非与指令
OR16	按位或指令
XOR16	按位异或指令
NOR16	按位或非指令
NOT16	按位非指令

表 5.26: 16 位移位指令列表

移位指令	
LSL16	逻辑左移指令
LSLI16	立即数逻辑左移指令
LSR16	逻辑右移指令
LSRI16	立即数逻辑右移指令
ASR16	算术右移指令
ASRI16	立即数算术右移指令
ROTL16	循环左移指令

表 5.27: 16 位比较指令列表

比较指令	
CMPNE16	不等比较指令
CMPNEI16	立即数不等比较指令
CMPHS16	无符号大于等于比较指令
CMPHSI16	立即数无符号大于等于比较指令
CMPLT16	有符号小于比较指令
CMPLTI16	立即数有符号小于比较指令
TST16	零测试指令
TSTNBZ16	无字节等于零寄存器测试指令

表 5.28: 16 位数据传输指令列表

数据传输指令	
MOV16	数据传送指令
MOVI16	立即数数据传送指令
MVCV16	C 位取反传送指令
LRW16	存储器读入指令

表 5.29: 16 位比特操作指令列表

比特操作指令	
BCLRI16	立即数位清零指令
BSETI16	立即数位置位指令
BTSTI16	立即数位测试指令

表 5.30: 16 位提取插入指令列表

提取插入指令	
ZEXTB16	字节提取并无符号扩展指令
ZEXTH16	半字提取并无符号扩展指令
SEXTB16	字节提取并有符号扩展指令
SEXTH16	半字提取并有符号扩展指令
REVB16	字节倒序指令
REVB16	半字内字节倒序指令

表 5.31: 16 位乘法指令列表

乘除法指令	
MULT16	乘法指令
MULSH16	16 位有符号乘法指令

5.2.1.2 分支跳转指令

分支跳转指令可以进一步分为：跳转指令、分支指令。

表 5.32: 16 位分支指令列表

分支指令	
BT16	C 为 1 分支指令
BF16	C 为 0 分支指令

表 5.33: 16 位跳转指令列表

跳转指令	
BR16	无条件跳转指令
JMP16	寄存器跳转指令
JSR16	寄存器跳转到子程序指令
RTS16	链接寄存器跳转指令

内存存取指令 内存存取指令可以进一步分为：立即数偏移存取指令、多寄存器存取指令。

表 5.34: 16 位立即数偏移存取指令列表

立即数偏移存取指令	
LD16.B	无符号扩展字节加载指令
LD16.H	无符号扩展半字加载指令
LD16.W	字加载指令
ST16.B	字节存储指令
ST16.H	半字存储指令
ST16.W	字存储指令

表 5.35: 16 位多寄存器存取指令列表

多寄存器存取指令	
POP16	出栈指令
PUSH16	压栈指令

5.3 指令集列表

C860 指令集具有高级语言特性，并为一些频繁执行的指令进行了优化。该指令集包括标准的算术逻辑指令、位操作指令、字节提取指令、数据转移指令、控制流改变指令和条件执行指令，这些条件执行指令有助于减少短跳转的条件转移。

表 5.36 列出了 C860 指令集中所有 16 位 32 位指令。

表 5.36: C860 基础指令集

汇编指令	32 位	16 位	指令描述
ADDU	O	O	无符号加法指令
ADDC	O	O	无符号带进位加法指令
ADDI	O	O	无符号立即数加法指令
SUBU	O	O	无符号减法指令
SUBC	O	O	无符号带借位减法指令
SUBI	O	O	无符号立即数减法指令
RSUB	O	×	反向减法指令
IXH	O	×	索引半字指令
IXW	O	×	索引字指令
IXD	O	×	索引双字指令
INCF	O	×	C 为 0 立即数加法指令
INCT	O	×	C 为 1 立即数加法指令
DECF	O	×	C 为 0 立即数减法指令

下页继续

表 5.36 – 续上页

汇编指令	32 位	16 位	指令描述
DECT	O	×	C 为 1 立即数减法指令
DECGT	O	×	减法大于零置 C 位指令
DECLT	O	×	减法小于零置 C 位指令
DECNE	O	×	减法不等于零置 C 位指令
AND	O	O	按位与指令
ANDI	O	×	立即数按位与指令
ANDN	O	O	按位非与指令
ANDNI	O	×	立即数按位非与指令
OR	O	O	按位或指令
ORI	O	×	立即数按位或指令
XOR	O	O	按位异或指令
XORI	O	×	立即数按位异或指令
NOR	O	O	按位或非指令
NOT	O	O	按位非指令
LSL	O	O	逻辑左移指令
LSLI	O	O	立即数逻辑左移指令
LSLC	O	×	立即数逻辑左移至 C 位指令
LSR	O	O	逻辑右移指令
LSRI	O	O	立即数逻辑右移指令
LSRC	O	×	立即数逻辑右移至 C 位指令
ASR	O	O	算术右移指令
ASRI	O	O	立即数算术右移指令
ASRC	O	×	立即数算术右移至 C 位指令
ROTL	O	O	循环左移指令
ROTLI	O	×	立即数循环左移指令
XSR	O	×	扩展右移指令
CMPNE	O	O	不等比较指令
CMPNEI	O	O	立即数不等比较指令
CMPHS	O	O	无符号大于等于比较指令
CMPHSI	O	O	立即数无符号大于等于比较指令
CMPLT	O	O	有符号小于比较指令
CMPLTI	O	O	立即数有符号小于比较指令
TST	O	O	零测试指令
TSTNBZ	O	O	无字节等于零寄存器测试指令
MOV	O	O	数据传送指令
MOVF	O	×	C 为 0 数据传送指令
MOVT	O	×	C 为 1 数据传送指令

下页继续

表 5.36 – 续上页

汇编指令	32 位	16 位	指令描述
MOVI	O	O	立即数数据传送指令
MOVIH	O	×	立即数高位数据传送指令
LRW	O	O	存储器读入指令
GRS	O	×	符号产生指令
MVCV	O	O	C 位取反传送指令
MVC	O	×	C 位传送指令
CLRf	O	×	C 为 0 清零指令
CLRT	O	×	C 为 1 清零指令
BCLRI	O	O	立即数位清零指令
BSETI	O	O	立即数位置位指令
BTSTI	O	×	立即数位测试指令
ZEXT	O	×	位提取并无符号扩展指令
SEXT	O	×	位提取并有符号扩展指令
INS	O	×	位插入指令
ZEXTB	O	O	字节提取并无符号扩展指令
ZEXTH	O	O	半字提取并无符号扩展指令
SEXTB	O	O	字节提取并有符号扩展指令
SEXTH	O	O	半字提取并有符号扩展指令
XTRB0	O	×	提取字节 0 并无符号扩展指令
XTRB1	O	×	提取字节 1 并无符号扩展指令
XTRB2	O	×	提取字节 2 并无符号扩展指令
XTRB3	O	×	提取字节 3 并无符号扩展指令
BREV	O	×	位倒序指令
REVB	O	O	字节倒序指令
REVBH	O	O	半字内字节倒序指令
MULT	O	O	乘法指令
MULSH	O	O	有符号数乘法指令
MULSW	O	×	有符号数 16x32 乘法指令
MULA.32.L	O	×	32 位有符号乘累加区低 32 位指令
MULALL.S16.S	O	×	带饱和操作的 16 位有符号低半字乘累加指令
MUL.U32	O	×	32 位无符号乘法指令
MULA.U32	O	×	32 位无符号乘累加法指令
MUL.S32	O	×	32 位有符号乘法指令
MULA.S32	O	×	32 位有符号乘累加法指令
FCMPNE.T	O	×	浮点不等于比较指令
FCMPNEZ.T	O	×	浮点不等于零比较指令

下页继续

表 5.36 – 续上页

汇编指令	32 位	16 位	指令描述
FCMPHS.T	O	×	浮点大于等于比较指令
FCMPHSZ.T	O	×	浮点大于等于零比较指令
FCMPHZ.T	O	×	浮点大于零比较指令
FCMPLT.T	O	×	浮点小于比较指令
FCMPLTZ.T	O	×	浮点小于零比较指令
FCMPLSZ.T	O	×	浮点小于等于零比较指令
FCMPUO.T	O	×	浮点双操作数非数判断指令
FCMPUOZ.T	O	×	浮点单操作数非数判断指令
FMAXNM.T	O	×	浮点规范最大值选择指令
FMINNM.T	O	×	浮点规范最小值选择指令
FMUL.T	O	×	浮点乘法指令
FNMUL.T	O	×	浮点乘法取负指令
FMULA.T	O	×	浮点乘累加指令 (乘法结果舍入后累加)
FMULS.T	O	×	浮点乘取负累加指令 (乘法结果舍入后累减)
FNMULA.T	O	×	浮点乘取负累减指令 (乘法结果舍入后累加)
FNMULS.T	O	×	浮点乘累减指令 (乘法结果舍入后累减)
FFMULA.T	O	×	浮点融合乘累加指令 (乘法结果保留全部精度累加)
FFMULS.T	O	×	浮点融合乘取负累加指令 (乘法结果保留全部精度累减)
FFNMULA.T	O	×	浮点融合乘取负累减指令 (乘法结果保留全部精度累加)
FFNMULS.T	O	×	浮点融合乘累减指令 (乘法结果保留全部精度累减)
FADD.T	O	×	单精度/ 双精度浮点数加法指令
FSUB.T	O	×	单精度/双精度浮点数减法指令
FSTOD	O	×	单精度浮点数转换成双精度浮点数指令
FDTOS	O	×	双精度浮点数转换成单精度浮点数指令
FFTOX.T1.T2	O	×	浮点定点转换指令
FXTOF.T1.T2	O	×	定点浮点转换指令
FFTOI.T1.T2	O	×	浮点整型转换指令
FITOF.T1.T2	O	×	整型浮点转换指令
FFTOI.T1.T2.RM	O	×	带舍入模式的浮点整型转换指令
FFTOFI.T.RM	O	×	带舍入模式的浮点转换为整型浮点指令。
FMTVR.64	O	×	浮点寄存器双字写传送指令
FMTVR.32.1	O	×	浮点寄存器低字写传送指令

下页继续

表 5.36 – 续上页

汇编指令	32 位	16 位	指令描述
FMTVR.32.2	O	×	浮点双寄存器低字写传送指令
FMFVR.64	O	×	浮点寄存器双字读传送指令
FMFVR.32.1	O	×	浮点寄存器低字读传送指令
FMFVR.32.2	O	×	浮点双寄存器低字读传送指令
FMFVRH	O	×	浮点寄存器高字读传送指令
FMTVRH	O	×	浮点寄存器高字写传送指令
FMOV.32	O	×	浮点寄存器间低字传送指令
FMOV.64	O	×	浮点寄存器间双字传送指令
FMOVX.32	O	×	浮点寄存器间低字低半字传送指令
FINS.32	O	×	浮点寄存器间低字高半字传送指令
FSEL.32	O	×	C 位选择低字传送指令
FSEL.64	O	×	C 位选择双字传送指令
DIVU	O	×	无符号除法指令
DIVS	O	×	有符号除法指令
DIVUL	O	×	无符号长除法指令
DIVSL	O	×	有符号长除法指令
ABS	O	×	绝对值指令
FF0	O	×	快速找 0 指令
FF1	O	×	快速找 1 指令
BMASKI	O	×	立即数位屏蔽产生指令
BGENR	O	×	寄存器位产生指令
BGENI	O	×	立即数位产生指令
BT	O	O	C 为 1 分支指令
BF	O	O	C 为 0 分支指令
BEZ	O	×	寄存器等于零分支指令
BNEZ	O	×	寄存器不等于零分支指令
BNEZAD	O	×	寄存器自减大于零分支指令
BHZ	O	×	寄存器大于零分支指令
BLSZ	O	×	寄存器小于等于零分支指令
BLZ	O	×	寄存器小于零分支指令
BHSZ	O	×	寄存器大于等于零分支指令
BR	O	O	无条件跳转指令
BSR	O	O	跳转到子程序指令
JMPI	O	×	间接跳转指令
JSRI	O	×	间接跳转到子程序指令
JMP	O	O	寄存器跳转指令
JSR	O	O	寄存器跳转到子程序指令

下页继续

表 5.36 – 续上页

汇编指令	32 位	16 位	指令描述
RTS	O	O	链接寄存器跳转指令
LD.B	O	O	无符号扩展字节加载指令
LD.BS	O	×	有符号扩展字节加载指令
LD.H	O	O	无符号扩展半字加载指令
LD.HS	O	×	有符号扩展半字加载指令
LD.W	O	O	字加载指令
LD.D	O	×	双字加载指令
FLD.T	O	×	单 / 双精浮点加载指令
ST.B	O	O	字节存储指令
ST.H	O	O	半字存储指令
ST.W	O	O	字存储指令
ST.D	O	×	双字存储指令
FST.T	O	×	浮点存储指令
LDR.B	O	×	寄存器移位寻址无符号扩展字节加载指令
LDR.BS	O	×	寄存器移位寻址有符号扩展字节加载指令
LDR.H	O	×	寄存器移位寻址无符号扩展半字加载指令
LDR.HS	O	×	寄存器移位寻址有符号扩展半字加载指令
LDR.W	O	×	寄存器移位寻址字加载指令
LDR.D	O	×	寄存器移位寻址双字加载指令
FLDR.T	O	×	寄存器移位寻址单 / 双精度浮点加载指令
STR.B	O	×	寄存器移位寻址字节存储指令
STR.H	O	×	寄存器移位寻址半字存储指令
STR.W	O	×	寄存器移位寻址字存储指令
STR.D	O	×	寄存器移位寻址双字存储指令
FSTR.T	O	×	寄存器移位寻址单 / 双精度浮点存储指令
LDEX.W	O	×	独占式加载指令
STEX.W	O	×	独占式存储指令
LRS.B	O	×	字节符号加载指令
LRS.H	O	×	半字符号加载指令
LRS.W	O	×	字符号加载指令
SRS.B	O	×	字节符号存储指令
SRS.H	O	×	半字符号存储指令

下页继续

表 5.36 – 续上页

汇编指令	32 位	16 位	指令描述
SRS.W	O	×	字符存储指令
LDQ	O	×	连续四字加载指令
LDM	O	×	连续多字加载指令
FLDM.T	O	×	连续单 / 双精度浮点加载指令
FLDMU.T	O	×	连续单 / 双精度浮点加载地址自增指令
STQ	O	×	连续四字存储指令
STM	O	×	连续多字存储指令
FSTM.T	O	×	连续单 / 双精度浮点存储指令
FSTMU.T	O	×	连续单 / 双精度浮点存储地址自增指令
POP	O	O	出栈指令
PUSH	O	O	压栈指令
MFCR	O	×	控制寄存器读传送指令
MTCR	O	×	控制寄存器写传送指令
PSRSET	O	×	PSR 位置位指令
PSRCLR	O	×	PSR 位清零指令
WAIT	O	×	进入低功耗等待模式指令
DOZE	O	×	进入低功耗睡眠模式指令
STOP	O	×	进入低功耗暂停模式指令
RTE	O	×	异常和普通中断返回指令
RFI	O	×	快速中断返回指令
ICACHE	O	×	指令 CACHE 操作指令
DCACHE	O	×	数据 CACHE 操作指令
TLBI	O	×	TLB 无效指令
L2CACHE	O	×	二级 CACHE 操作指令
SYNC	O	×	CPU 同步指令
BAR	O	×	内存同步指令
BKPT	×	O	断点指令
IDLY	O	×	中断识别禁止指令
TRAP	O	×	无条件操作系统陷阱指令
PLDR	O	×	读数据预取指令
PLDW	O	×	写数据预取指令

表 5.37 列出了 C860 矢量浮点指令集中所有 32 位指令。

表 5.37: C860 矢量浮点扩展指令集

汇编指令	32 位	16 位	指令描述
VCMPEZ.T	O	×	矢量浮点不等于零比较指令

下页继续

表 5.37 – 续上页

汇编指令	32 位	16 位	指令描述
VCMPNE.T	O	×	矢量浮点不等于比较指令
VCMPHSZ.T	O	×	矢量浮点大于等于零比较指令
VCMPHS.T	O	×	矢量浮点大于等于比较指令
VCMPLTZ.T	O	×	矢量浮点小于零比较指令
VCMPLT.T	O	×	矢量浮点小于比较指令
VCMPHZ.T	O	×	矢量浮点大于零比较指令
VCMPLSZ.T	O	×	矢量浮点小于等于零比较指令
VMAX.T	O	×	矢量浮点最大值选择指令
VMAXNM.T	O	×	矢量浮点规范最大值选择指令
VMIN.T	O	×	矢量浮点最小值选择指令
VMINNM.T	O	×	矢量浮点规范最小值选择指令
VPMAX.T	O	×	矢量浮点相邻最大值选择指令
VPMIN.T	O	×	矢量浮点相邻最小值选择指令
VMUL.T	O	×	矢量单精度乘法指令
VMULI.T	O	×	矢量单精度索引乘法指令
VMULA.T	O	×	矢量单精度乘累加指令（乘法结果舍入后累加）
VMULAI.T	O	×	矢量单精度索引乘累加指令（乘法结果舍入后累加）
VMULS.T	O	×	矢量单精度乘累减指令（乘法结果舍入后累减）
VMULSI.T	O	×	矢量单精度索引乘累减指令（乘法结果舍入累减）
VFMULA.T	O	×	矢量单精度融合乘累加指令（乘法结果保留全部精度累加）
VFMULS.T	O	×	矢量单精度融合乘累减指令（乘法结果保留全部精度累减）
VFMULXAA.T	O	×	矢量单精度复数乘累加实部虚部部分计算（乘法保留全部精度累加）
VFMULXAAL.T	O	×	矢量单精度索引复数乘累加实部虚部部分计算（乘法保留全部精度累加）
VFMULXAS.T	O	×	矢量单精度复数乘累加实部虚部部分计算（乘法保留全部精度累加）
VFMULXASL.T	O	×	矢量单精度索引复数乘累加实部虚部部分计算（乘法保留全部精度累加）
VFMULXSS.T	O	×	矢量单精度复数乘累减实部虚部部分计算（乘法保留全部精度累减）

下页继续

表 5.37 – 续上页

汇编指令	32 位	16 位	指令描述
VFMULXSSI.T	O	×	矢量单精度索引复数乘累减实部虚部部分计算 (乘法保留全部精度累减)
VFMULXSA.T	O	×	矢量单精度复数乘累减实部虚部部分计算 (乘法保留全部精度累减)
VFMULXSAI.T	O	×	矢量单精度索引复数乘累减实部虚部部分计算 (乘法保留全部精度累减)
VFCMUL.T	O	×	矢量单精度复数乘法 (乘法结果舍入后累加)
VFCMULA.T	O	×	矢量单精度复数乘累加 (乘法结果舍入后累加)
VFCMULC.T	O	×	矢量单精度复数共轭乘法 (乘法结果舍入后累加)
VFCMULCA.T	O	×	矢量单精度复数共轭乘累加 (乘法结果舍入后累加)
VFCMULN.T	O	×	矢量单精度复数取负乘法 (乘法结果舍入后累加)
VFCMULNA.T	O	×	矢量单精度复数取负乘累加 (乘法结果舍入后累加)
VFCMULCN.T	O	×	矢量单精度复数共轭取负乘法 (乘法结果舍入后累加)
VFCMULCNA.T	O	×	矢量单精度复数共轭取负乘累加 (乘法结果舍入后累加)
VRECPS.T	O	×	矢量单精度倒数逼近
VRSQRTS.T	O	×	矢量单精度开方逼近
VEXPE.T	O	×	矢量单精度快速 e 指数计算
VDTOS	O	×	矢量双精度浮点单精度浮点转换指令
VFTOFL.T.RM	O	×	矢量带舍入模式的浮点舍弃小数指令
VFTOI.T1.T2	O	×	矢量浮点符号整数转换指令
VFTOI.T1.T2.RM	O	×	矢量带舍入模式的浮点有符号整数转换指令
VFTOX.T1.T2	O	×	矢量浮点定点转换指令
VITOF.T1.T2	O	×	矢量整数浮点转换指令
VXTOF.T1.T2	O	×	矢量定点浮点转换指令

表 5.38 列出了 C860 矢量多媒体指令集中所有 32 位指令。

表 5.38: C860 矢量多媒体扩展指令集

汇编指令	32 位	16 位	指令描述
VADD.T	O	×	矢量加法指令
VADD.T.S	O	×	矢量饱和加法指令
VADD.T.H	O	×	矢量加法取高半指令
VADD.T.RH	O	×	矢量加法取高半舍入指令
VADD.T.E	O	×	矢量扩展加法指令
VADD.T.X	O	×	矢量非对齐加法指令
VADDH.T	O	×	矢量加法平均指令
VADDH.T.R	O	×	矢量加法平均并舍入指令
VPADD.T	O	×	矢量链加法指令
VPADD.T.S	O	×	矢量饱和链加法指令
VPADD.T.E	O	×	矢量扩展链加法指令
VPADDA.T.E	O	×	矢量扩展链累法指令
VASX.T.S	O	×	矢量错位加减指令
VSAX.T.S	O	×	矢量错位减加指令
VASXH.T	O	×	矢量错位加减平均指令
VSAXH.T	O	×	矢量错位减加平均指令
VSUB.T	O	×	矢量减法指令
VSUB.T.S	O	×	矢量饱和减法指令
VSUB.T.H	O	×	矢量减法取高半指令
VSUB.T.RH	O	×	矢量减法取高半舍入指令
VSUB.T.E	O	×	矢量扩展减法指令
VSUB.T.X	O	×	矢量非对齐减法指令
VSUBH.T.R	O	×	矢量减法平均并舍入指令
VSUBH.T	O	×	矢量减法平均指令
VABS.T	O	×	矢量取绝对值指令
VABS.T.S	O	×	矢量饱和和取绝对值指令
VSABS.T.S	O	×	矢量饱和和减法取绝对值指令
VSABS.T.E	O	×	矢量扩展减法取绝对值指令
VSABSA.T	O	×	矢量减法取绝对值并累加指令
VSABSA.T.E	O	×	矢量扩展减法取绝对值并累加指令
VNEG.T	O	×	矢量取反指令
VNEG.T.S	O	×	矢量饱和和取反指令
VMOVL.8	O	×	矢量立即数传输指令
VMOVL.16	O	×	矢量立即数传输指令
VMOVL.32	O	×	矢量立即数传输指令
VMASKL.8.l	O	×	矢量立即数扩展传输指令
VMASKL.8.H	O	×	矢量立即数扩展传输指令

下页继续

表 5.38 – 续上页

汇编指令	32 位	16 位	指令描述
VMASKI.16	O	×	矢量立即数扩展传输指令
VAND.T	O	×	矢量与指令
VANDN.T	O	×	矢量与非指令
VXOR.T	O	×	矢量异或指令
VOR.T	O	×	矢量或指令
VORN.T	O	×	矢量或非指令
VNOT.T	O	×	矢量取反指令
VTST.T	O	×	矢量按位与置位指令
VSEL.T	O	×	矢量位选择指令
VCLS.T	O	×	矢量符号位连续相同指令
VCLZ.T	O	×	矢量最高位连续 0 个数指令
VCNT1.T	O	×	矢量数据 1 个数指令
VSHT.T	O	×	矢量移位指令
VSHT.T.S	O	×	矢量移位饱和指令
VSHT.T.R	O	×	矢量移位舍入指令
VSHT.T.RS	O	×	矢量移位饱和舍入指令
VSHL.T	O	×	矢量寄存器左移指令
VSHL.T.S	O	×	矢量寄存器左移饱和指令
VSHLI.T	O	×	矢量立即数左移指令
VSHLI.T.S	O	×	矢量立即数左移饱和指令
VSHLI.T.E	O	×	矢量扩展立即数左移指令
VSHR.T	O	×	矢量寄存器右移指令
VSHR.T.R	O	×	矢量寄存器右移舍入指令
VSHRI.T	O	×	矢量立即数右移指令
VSHRI.T.R	O	×	矢量立即数左移舍入指令
VSHRI.T.L	O	×	矢量立即数右移取元素低半部分指令
VSHRI.T.LR	O	×	矢量立即数右移舍入取元素低半部分指令
VSHRI.T.LS	O	×	矢量立即数右移取元素低半部分饱和指令
VSHRI.T.LRS	O	×	矢量立即数右移舍入取元素低半部分饱和指令
VSHRIA.T	O	×	矢量立即数右移累加指令
VSHRIA.T.R	O	×	矢量立即数右移舍入累加指令
VEXH.T	O	×	矢量低位数据插入指令
VEXL.T	O	×	矢量高位数据插入指令
VCLIP.T	O	×	矢量裁剪饱和指令
VCMPHS.T	O	×	矢量大于等于比较指令
VCMLPT.T	O	×	矢量小于比较指令
VCMPNE.T	O	×	矢量不等于比较指令

下页继续

表 5.38 – 续上页

汇编指令	32 位	16 位	指令描述
VCMPSZ.T	O	×	矢量大于等于 0 比较指令
VCMLTZ.T	O	×	矢量小于 0 比较指令
VCMPEZ.T	O	×	矢量不等于 0 比较指令
VCMPHZ.T	O	×	矢量大于 0 比较指令
VCMLSZ.T	O	×	矢量小于等于 0 比较指令
VMAX.T	O	×	矢量最大值选择指令
VMIN.T	O	×	矢量最小值选择指令
VPMAX.T	O	×	矢量相邻最大值选择指令
VPMIN.T	O	×	矢量相邻最小值选择指令
VCLIP.T	O	×	矢量裁剪饱和指令
VMTVR.T.1	O	×	矢量写传送指令
VMTVR.T.2	O	×	矢量写传送指令
VMFVR.T	O	×	矢量写传送指令
VSEXT.T	O	×	矢量数据符号位读传送指令
VMOV.T	O	×	矢量传输指令
VMOV.T.E	O	×	矢量扩展传输指令
VMOV.T.L	O	×	矢量低位传输指令
VMOV.T.H	O	×	矢量高位传输指令
VMOV.T.RH	O	×	矢量高位 round 传输指令
VSWP.T	O	×	矢量数据交换指令
VTRN.T	O	×	矢量数据交叉传输指令
VREVH.T	O	×	矢量数据半字倒序指令
VREVT.T	O	×	矢量数据字倒序指令
VREVD.T	O	×	矢量数据双字倒序指令
VREVQ.T	O	×	矢量数据字节倒序指令
VEXTI.T	O	×	立即数矢量数据拼接指令
VEXT.T	O	×	寄存器矢量数据拼接指令
VTBL.T	O	×	矢量数据链接指令
VTBX.T	O	×	矢量数据链接指令
VDUP.T.1	O	×	一元矢量源目的寄存器之间的拷贝指令
VDUP.T.2	O	×	二元矢量源目的寄存器之间的拷贝指令
VDUP.T.3	O	×	三元矢量源目的寄存器之间的拷贝指令
VDUP.T.4	O	×	四元矢量源目的寄存器之间的拷贝指令
VINS.T.1	O	×	一元矢量插入指令
VINS.T.2	O	×	二元矢量插入指令
VINS.T.3	O	×	三元矢量插入指令
VINS.T.4	O	×	四元矢量插入指令

下页继续

表 5.38 – 续上页

汇编指令	32 位	16 位	指令描述
VPKG.T.2	O	×	二元封装指令
VPKG.T.3	O	×	三元封装指令
VPKG.T.4	O	×	四元封装指令
VITL.T.2	O	×	二元交织指令
VDTL.T.2	O	×	二元去交织指令
VMUL.T	O	×	矢量乘法指令
VMULI.T	O	×	矢量带索引取高半乘法指令
VMUL.T.H	O	×	矢量取高半乘法指令
VMUL.T.E	O	×	矢量扩展乘法指令
VMULI.T.E	O	×	矢量扩展带索引乘法指令
VMULA.T	O	×	矢量乘累加指令
VMULAI.T	O	×	矢量带索引乘累加指令
VMULA.T.E	O	×	矢量扩展乘累加指令
VMULAI.T.E	O	×	矢量扩展带索引乘累加指令
VMULS.T	O	×	矢量乘累减指令
VMULSI.T	O	×	矢量带索引乘累减指令
VMULS.T.E	O	×	矢量扩展乘累减指令
VMULSI.T.E	O	×	矢量扩展带索引乘累减指令
VMULACA.T	O	×	矢量链乘累加指令
VMULACAI.T	O	×	矢量链带索引乘累加指令
VMULACAA.T	O	×	矢量链乘累加加指令
VMULACAAI.T	O	×	矢量链带索引乘累加加指令
VRMUL.T.SE	O	×	矢量扩展带饱和和小数乘法指令
VRMULI.T.SE	O	×	矢量带索引扩展带饱和和小数乘法指令
VRMULH.T.S	O	×	矢量带饱和和取高半小数乘法指令
VRMULHI.T.S	O	×	矢量带索引带饱和和取高半小数乘法指令
VRMULH.T.RS	O	×	矢量带饱和和取高半舍入小数乘法指令
VRMULHI.T.RS	O	×	矢量带索引带饱和和取高半舍入小数乘法指令
VRMULHA.T.RS	O	×	矢量带饱和和取高半舍入小数乘累加指令
VRMULHAI.T.RS	O	×	矢量带索引带饱和和取高半舍入小数乘累加指令
VRMULHS.T.RS	O	×	矢量带饱和和取高半舍入小数乘累减指令
VRMULHSI.T.RS	O	×	矢量带索引带饱和和取高半舍入小数乘累减指令
VRMULSHR.T.E	O	×	矢量扩展带移位小数乘法指令
VRMULSHRI.T.E	O	×	矢量带索引扩展带移位小数乘法指令
VRMULSA.T.E	O	×	矢量扩展带移位小数乘累加指令
VRMULSAI.T.E	O	×	矢量带索引扩展带移位小数乘累加指令
VRMULSS.T.E	O	×	矢量扩展带移位小数乘负累加指令

下页继续

表 5.38 – 续上页

汇编指令	32 位	16 位	指令描述
VRMULSSI.T.E	O	×	矢量带索引扩展带移位小数乘负累加指令
VRMULXAA.T.RS	O	×	矢量带饱和和复数实部虚部交叉相乘累加累加取高半舍入指令
VRMULXAAI.T.RS	O	×	矢量带索引带饱和和复数实部虚部交叉相乘累加累加取高半舍入指令
VRMULXAS.T.RS	O	×	矢量带饱和和复数实部虚部交叉相乘累加累减取高半舍入指令
VRMULXASI.T.RS	O	×	矢量带索引带饱和和复数实部虚部交叉相乘累加累减取高半舍入指令
VRMULXSS.T.RS	O	×	矢量带饱和和复数实部虚部交叉相乘累减累减取高半舍入指令
VRMULXSSI.T.RS	O	×	矢量带索引带饱和和复数实部虚部交叉相乘累减累减取高半舍入指令
VRMULXSA.T.RS	O	×	矢量带饱和和复数实部虚部交叉相乘累减累加取高半舍入指令
VRMULXSAI.T.RS	O	×	矢量带索引带饱和和复数实部虚部交叉相乘累加累减取高半舍入指令
VRCMUL.T.RS	O	×	矢量带饱和和复数乘法取高半舍入指令
VRCMULA.T.E	O	×	矢量复数乘法右移扩位累加指令
VRCMULC.T.RS	O	×	矢量带饱和和复数共轭乘法取高半舍入指令
VRCMULCA.T.E	O	×	矢量共轭复数乘法右移扩位累加指令
VRCMULN.T.RS	O	×	矢量带饱和和复数取负乘法取高半舍入指令
VRCMULNA.T.E	O	×	矢量复数取负乘法右移扩位累加指令
VRCMULCN.T.RS	O	×	矢量带饱和和复数共轭取负乘法取高半舍入指令
VRCMULCNA.T.E	O	×	矢量共轭复数取负乘法右移扩位累加指令
VRECPE.T	O	×	矢量倒数指令
VRECPS.T	O	×	矢量倒数逼近指令
VRSQRTE.T	O	×	矢量倒数开方指令
VRSQRTS.T	O	×	矢量倒数开方逼近指令
VEXPE.T	O	×	矢量 e 指数指令
VLD.T.N	O	×	固定长度矢量加载指令
VLDRU.T.N	O	×	矢量加载基址跳跃更新指令
VLDU.T.N	O	×	矢量加载基址更新指令
VLDX.T	O	×	可变长度矢量加载指令
VLDM.T	O	×	连续矢量加载指令
VLDMU.T	O	×	连续矢量加载基址更新指令
VLDMRU.T	O	×	跳跃矢量加载基址更新指令
VST.T.N	O	×	固定长度矢量存储指令

下页继续

表 5.38 – 续上页

汇编指令	32 位	16 位	指令描述
VSTRU.T.N	O	×	矢量存储基址跳跃更新指令
VSTU.T.N	O	×	矢量存储基址更新指令
VSTX.T	O	×	可变长度矢量存储指令
VSTM.T	O	×	连续矢量存储指令
VSTMU.T	O	×	连续矢量存储基址更新指令
VSTMURU.T	O	×	跳跃矢量存储基址更新指令
VLRW.32.N	O	×	矢量存储器读入指令

注解： O 表示相应指令集中存在该指令， × 表示相应指令集中不存在该指令。

5.4 指令执行延迟

表 5.39: C860 基础指令执行延迟

指令类型	指令	执行周期	备注
加减法指令	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	-
	DECF32	1	-
	DECT32	1	-
	DECGT32	1	-
	DECLT32	1	-
	DECNE32	1	-
	FADD.T	3	-
	FSUB.T	3	-
数据类型转换指令	FSTOD	3	-
	FDTOS	3	-

下页继续

表 5.39 – 续上页

指令类型	指令	执行周期	备注
	FFTOX.T1.T2	3	-
	FXTOF.T1.T2	3	-
	FFTOI.T1.T2	3	-
	FITOF.T1.T2	3	-
	FFTOI.T1.T2.RM	3	-
	FFTOFI.T.RM	3	-
逻辑操作指令	AND32/16	1	-
	ANDI32/16	1	-
	ANDN32	1	-
	ANDNI32	1	-
	OR32/16	1	-
	ORI32	1	-
	XOR32/16	1	-
	XORI32	1	-
	NOR32/16	1	-
	NOT32/16	1	-
移位指令	LSL32/16	1	-
	LSLI32/16	1	-
	LSLC32	1	-
	LSR32/16	1	-
	LSRI32/16	1	-
	LSRC32	1	-
	ASR32/16	1	-
	ASRI32/16	1	-
	ASRC32	1	-
	ROTL32/16	1	-
	ROTLI32	1	-
	XSR32	1	-
比较指令	CMPNE32/16	1	-
	CMPNEI32/16	1	-
	CMPHS32/16	1	-
	CMPHSI32/16	1	-
	CMPLT32/16	1	-
	CMPLTI32/16	1	-
	FCMPNE.T	3	-
	FCMPNEZ.T	3	-
	FCMPHS.T	3	-
	FCMPHSZ.T	3	-
	FCMPHZ.T	3	-

下页继续

表 5.39 – 续上页

指令类型	指令	执行周期	备注
	FCMPLT.T	3	-
	FCMPLTZ.T	3	-
	FCMPLSZ.T	3	-
	FCMPUO.T	3	-
	FCMPUOZ.T	3	-
	FMAXNM.T	3	-
	FMINNM.T	3	-
	TST32/16	1	-
	TSTNBZ32/16	1	-
数据传输指令	MOV32/16	1	-
	MOVF32	1	-
	MOVT32	1	-
	MOVI32/16	1	-
	MOVIH32	1	-
	MVCV32/16	1	-
	MVC32	1	-
	CLRF32	1	-
	CLRT32	1	-
	LRW32/16*	≥4	L1 cache 命中: 4 个周期 L2 cache 命中: ≥17 个周期 L2cache 缺失: 片外访问延迟
	FLRW.T	≥4	
	VLRW.32.N	≥4	
	GRS32	4	-
	FMTVR.64	3	-
	FMTVR.32.1	3	拆分成 2 条微指令
	FMTVR.32.2	3	拆分成 4 条微指令
	FMFVR.64	3	拆分成 2 条微指令
	FMFVR.32.1	3	-
	FMFVR.32.2	3	拆分成 2 条微指令
	FMFVRH	3	-
	FMTVRH	3	拆分成 2 条微指令
	FMOV.32	3	-
	FMOV.64	3	-
	FMOVX.32	3	-
	FINS.32	3	-
	FSEL.32	3	-
	FSEL.64	3	-
比特操作指令	BCLRI32/16	1	-

下页继续

表 5.39 – 续上页

指令类型	指令	执行周期	备注
提取插入指令	BSETI32/16	1	-
	BTSTI32/16	1	-
	ZEXT32/16	2	ZEXT 拆分为 2 条微指令
	SEXT32/16	2	SEXT 拆分为 2 条微指令
	INS32	3	INS 拆分为 3 条微指令
	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	-
	REVBH32/16	1	-
乘除法指令	MULT32/16	4	-
	MULSW32	4	-
	MULSH32/16	4	-
	MULA.32.L	4+1	拆成 PMULSL & PADD PMULSL:4 周期 PADD:4 周期
	MULALL.S16.S	4+1	拆成 MULSH & PADDS MULSH:4 周期 PADDS:1 周期
	MUL.U32	4+4	拆成 PMULUL&PMULUH PMULUL:4 周期 PMULUH:4 周期
	MULA.U32	4+1+4+1	拆成 PMULUL&PADDC1 &PMULUH&PADDC2 PMULUL:4 周期 PADDC1:1 周期 PMULUH:4 周期 PADDC2:1 周期
	MUL.S32	4	拆成 PMULSL&PMULSH PMULSL:4 周期 PMULSH:4 周期
	MULA.S32	4+1+4+1	拆成 PMULSL&PADDC1 &PMULSH&PADDC2 PMULSL:4 周期 PADDC1:1 周期 PMULSH:4 周期 PADDC2:1 周期
	FDIV.T	4~18/4~32	单精度除法:4~18 个周期 双精度除法:4~32 个周期
	FMUL.T	4	-
	FNMUL.T	4	-

下页继续

表 5.39 – 续上页

指令类型	指令	执行周期	备注
	FMULA.T	4+3	拆成 FMUL & FADD FMUL: 4 周期 FADD: 3 周期
	FMULS.T	4+3	拆成 FMUL & FADD FMUL: 4 周期 FADD: 3 周期
	FNMULA.T	4+3	拆成 FMUL & FSUB FMUL: 4 周期 FADD: 3 周期
	FNMULS.T	4+3	拆成 FMUL & FSUB FMUL: 4 周期 FADD: 3 周期
	FFMULA.T	5	-
	FFMULS.T	5	-
	FFNMULA.T	5	-
	FFNMULS.T	5	-
	FRECIP.T	4~18/4~32	单精度倒数:4~18 个周期 双精度倒数:4~32 个周期
	FSQRT.T	4~18/4~32	单精度开方:4~18 个周期 双精度开方:4~32 个周期
	DIVU32	6~35	-
	DIVS32	6~35	-
	DIVUL	6~67	拆成两条微指令
	DIVSL	6~67	拆成两条微指令
杂类运算指令	ABS32	1	-
	FF0. 32	1	-
	FF1. 32	1	-
	BMASKI32	1	-
	BGENR32	1	-
	BGENI32	1	-
分支指令	BT32/16	1	-
	BF32/16	1	-
	BEZ32	1	-
	BNEZ32	1	-
	BNEZAD32	1+1	拆分为 SUBI 及 BNEZ 指令
	BHZ32	1	-
	BLSZ32	1	-
	BLZ32	1	-
	BHSZ32	1	-
跳转指令	BR32/16	1	-
	BSR32	1	-
	JMPI32	>=4+1	拆分为 LRW 及 JMP 指令
	JSRI32	>=4+1	拆分为 LRW 及 JSR 指令

下页继续

表 5.39 – 续上页

指令类型	指令	执行周期	备注
	JMP32/16	1	-
	JSR32/16	1	-
	RTS32/16	1	-
立即数偏移存取指令	LD32/16.B	≥ 4	L1 cache 命中: 4 个周期 L2 cache 命中: ≥ 17 个周期 L2cache 缺失: 片外访问延迟
	LD32.BS	≥ 4	
	LD32/16.H	≥ 4	
	LD32.HS	≥ 4	
	LD32/16.W	≥ 4	
	FLD.T	≥ 4	
	LD32.D	$\geq 4+1$	拆分为 2 条 LD.W 指令
	ST32/16.B	4	-
	ST32/16.H	4	-
	ST32/16.W	4	-
	FST.T	4	-
	ST32.D	4+1	拆分为 2 条 ST.W 指令
寄存器偏移存取指令	LDR32.B	≥ 4	L1 cache 命中: 4 个周期 L2 cache 命中: ≥ 17 个周期 L2cache 缺失: 片外访问延迟
	LDR32.BS	≥ 4	
	LDR32.H	≥ 4	
	LDR32.HS	≥ 4	
	LDR32.W	≥ 4	
	LDR32.D	≥ 4	
	FLDR.T	≥ 4	
	STR32.B	4	-
	STR32.H	4	-
	STR32.W	4	-
	STR32.D	4	拆分为四条指令
	FSTR.T	4	-
多寄存器存取指令	LDQ32	$\geq 4+3$	拆分为 4 条 LD.W 指令。
	LDM32	$\geq 4+N-1$	拆分为 N 条 LD.W 指令。
	FLDM.T	$\geq 4+N-1$	拆分为 N 条 LD.W 指令。
	FLDMU.T	$\geq 4+N-1$	拆分为 N 条 LD.W 指令和 1 条 ADD 指令。
	STQ32	4+3	拆分为 4 条 ST.W 指令。
	STM32	4+N-1	拆分为 N 条 ST.W 指令。
	FSTM.T	$\geq 4+N-1$	拆分为 N 条 ST.W 指令。

下页继续

表 5.39 – 续上页

指令类型	指令	执行周期	备注
	FSTMU.T	$\geq 4+N-1$	拆分为 N 条 ST.W 指令和 1 条 ADD 指令。
	PUSH32/16	$4+N$	拆分为 SUB 和 N 条 ST.W 指令。
	POP32/16	$\geq 4+N+1$	拆分为 N 条 LD.W、ADD 和 RTS 指令。
独占存取指令	LDEX32.W	≥ 4	L1 cache 命中: 4 个周期 L2 cache 命中: ≥ 17 个周期 L2cache 缺失: 片外访问延迟
	STEX32.W	≥ 4	
符号存取指令	LRS32.B	≥ 4	L1 cache 命中: 4 个周期 L2 cache 命中: ≥ 17 个周期 L2cache 缺失: 片外访问延迟
	LRS32.H	≥ 4	
	LRS32.W	≥ 4	
	SRS32.B	4	-
	SRS32.H	4	-
	SRS32.W	4	-
控制寄存器操作指令	MFCCR32	*	阻塞执行
	MTCR32	*	阻塞执行
	PSRSET32	*	阻塞执行
	PSRCLR32	*	阻塞执行
低功耗指令	WAIT32	*	阻塞执行
	DOZE32	*	阻塞执行
	STOP32	*	阻塞执行
CACHE 操作指令	ICACHE32	*	取决于高速缓存负载
	DCACHE32	*	取决于高速缓存负载
	TLBI32	*	取决于高速缓存负载
	L2CACHE32	*	取决于高速缓存负载
异常返回指令	RTE32	*	阻塞执行
	RFI32	*	阻塞执行
	PLDW32	≥ 4	-

表 5.40: C860 矢量浮点指令执行延迟

指令类型	指令	执行周期	备注
加减法指令	VADD.T	3	
	VSUB.T	3	
	VPADD.T	3	
	VASX.T	3	
	VSAX.T	3	
	VSABS.T	3	
数据类型转换	VDTOS	3	
	VFTOFL.T.RM	3	
	VFTOLT.T1.T2	3	
	VFTOLT.T1.T2.RM	3	
	VFTOX.T1.T2	3	
	VITOF.T1.T2	3	
	VXTOF.T1.T2	3	
比较指令	VCMPNE.T	3	
	VCMPNEZ.T	3	
	VCMPHS.T	3	
	VCMPHSZ.T	3	
	VCMPLT.T	3	
	VCMPLTZ.T	3	
	VCMPHZ.T	3	
	VCMPLSZ.T	3	
	VMAX.T	3	
	VMAXNM.T	3	
	VMIN.T	3	
	VMINNM.T	3	
	VPMAX.T	3	
	VPMIN.T	3	
	VMUL.T	4	
	VMULI.T	4	

表 5.41: C860 矢量浮点指令执行延迟

指令类型	指令	执行周期	备注
乘除法指令	VMULA.T	4+3	拆成 VMUL & VADD VMUL:4 周期 VADD:3 周期
	VMULAI.T	4+3	拆成 VMULI & VADD VMULI:4 周期 VADD:3 周期
	VMULS.T	4+3	拆成 VMUL & VADD VMUL:4 周期 VADD:3 周期
	VMULSI.T	4+3	拆成 VMULI & VADD VMULI:4 周期 VADD:3 周期
	VFMULA.T	5	
	VFMULS.T	5	
	VF-MULXAA.T	5	
	VFMULX-AAI.T	5	
	VF-MULXAS.T	5	
	VFMULX-ASI.T	5	
	VF-MULXSS.T	5	
	VF-MULXSSI.T	5	
	VF-MULXSA.T	5	
	VFMULX-SAI.T	5	
	VFCMUL.T	4+5	拆成 VMUL & VFMULXAS VMUL: 4 周期 VFMULXAS: 5 周期
	VFCMULA.T	5+5	拆成 VFMULXAA & VFMULXSA VFMULXAA: 5 周期 VFMULXSA: 5 周期
	VFCMULC.T	4+5	拆成 VMUL & VFMULXAS VMUL:4 周期 VFMULXAS:5 周期
	VFC-MULCA.T	5+5	拆成 VFMULXAA & VFMULXSA VFMULXAA:5 周期 VFMULXSA:5 周期
www.xrvim.cn	VFCMULN.T	4+5	拆成 VMUL & VFMULXSA VMUL:4 周期 VFMULXSA:5 周期
	VFC-	5+5	拆成 VFMULXSS & VFMULXSA

表 5.42: C860 矢量多媒体指令执行延迟

指令类型	指令	执行周期	备注
加减法指令	VADD.T	2	
	VADD.T.S	3	
	VADD.T.H	2	
	VADD.T.RH	3	
	VADD.T.E	2	拆分 2 条微指令
	VADD.T.X	2	拆分 2 条微指令
	VADDH.T	2	
	VADDH.T.R	3	
	VPADD.T	2	
	VPADD.T.S	3	
	VPADD.T.E	2	
	VPADDA.T.E	3	
	VASX.T.S	3	
	VSAX.T.S	3	
	VASXH.T	2	
	VSAXH.T	2	
	VADDH.T	2	
	VADDH.T.R	3	
	VSUB.T	2	
	VSUB.T.S	3	
	VSUB.T.H	2	
	VSUB.T.RH	3	
	VSUB.T.E	2	拆分 2 条微指令
	VSUB.T.X	2	拆分 2 条微指令
	VSUBH.T.R	3	
	VSUBH.T	2	
	VABS.T	2	
	VABS.T.S	3	
	VSABS.T.S	3	
	VSABS.T.E	2	拆分 2 条微指令
	VSABSA.T	3	
	VSABSA.T.E	3	拆分 2 条微指令
	VNEG.T	2	
	VNEG.T.S	3	
立即数生成指令	VMOVL.8	3	
	VMOVL.16	3	
	VMOVL.32	3	
	VMASKL.8.1	3	

下页继续

表 5.42 – 续上页

指令类型	指令	执行周期	备注
逻辑操作指令	VMASKI.8.H	3	
	VMASKI.16	3	
	VAND.T	3	
	VANDN.T	3	
	VXOR.T	3	
	VOR.T	3	
	VORN.T	3	
	VNOT.T	3	
	VTST.T	3	
	VSEL.T		
	VCLS.T	3	
	VCLZ.T	3	
	VCNT1.T	3	
移位指令	VSHT.T	4	
	VSHT.T.S	4	
	VSHT.T.R	4	
	VSHT.T.RS	4	
	VSHL.T	3	
	VSHL.T.S	3	
	VSHLI.T	3	
	VSHLI.T.S	3	
	VSHLI.T.E	3	拆分 2 条微指令
	VSHR.T	3	
	VSHR.T.R	4	
	VSHRI.T	3	
	VSHRI.T.R	4	
	VSHRI.T.L	4	
	VSHRI.T.LR	4	
	VSHRI.T.LS	4	
	VSHRI.T.LRS	4	
	VSHRIA.T	4	
	VSHRIA.T.R	4	
	VEXH.T	3	
	VEXL.T	3	
比较指令	VCMPHS.T	2	
	VCMPLT.T	2	
	VCMPNE.T	2	
	VCMPHSZ.T	2	
	VCMPLTZ.T	2	

下页继续

表 5.42 – 续上页

指令类型	指令	执行周期	备注
	VCMPNEZ.T	2	
	VCMPHZ.T	2	
	VCMPLSZ.T	2	
	VMAX.T	2	
	VMIN.T	2	
	VPMAX.T	2	
	VPMIN.T	2	
	VCLIP.T	4	
数据传输指令	VMTVR.T.1	3	
	VMTVR.T.2	3	
	VMFVR.T	3	
	VSEXT.T	3	
	VMOV.T.E	3	拆分 2 条微指令
	VMOV.T.L	3	
	VMOV.T.H	3	
	VMOV.T.SL	3	
	VMOV.T.RH	4	
提取插入指令	VSWP.T	3	拆分 2 条微指令
	VTRN.T	3	拆分 2 条微指令
	VREXH.T	3	
	VREXW.T	3	
	VREXD.T	3	
	VREXQ.T	3	
	VEXTL.T	3	
	VEXT.T	3	
	VTBL.T	3	
	VTBX.T	3	
	VDUPG.T	3	
	VDUP.T	1	
	VDUP.T.2	1	拆分 2 条微指令
	VDUP.T.3	1	拆分 3 条微指令
	VDUP.T.4	1	拆分 4 条微指令
	VINS.T	1	
	VINS.T.2	1	拆分 2 条微指令
	VINS.T.3	1	拆分 3 条微指令
	VINS.T.4	1	拆分 4 条微指令
	VPKG.T.2	1	拆分 2 条微指令
	VPKG.T.3	1	拆分 3 条微指令
	VPKG.T.4	1	拆分 4 条微指令

下页继续

表 5.42 – 续上页

指令类型	指令	执行周期	备注
乘法指令	VITL.T.2	1	拆分 2 条微指令
	VDTL.T.2	1	拆分 2 条微指令
	VMUL.T	3-5	Size 为 8, 3 周期 Size 为 16, 4 周期 Size 为 32, 5 周期
	VMULI.T	3-5	Size 为 8, 3 周期 Size 为 16, 4 周期 Size 为 32, 5 周期
	VMUL.T.H	3-5	Size 为 8, 3 周期 Size 为 16, 4 周期 Size 为 32, 5 周期
	VMUL.T.E	3-5	拆分 2 条微指令 每条拆分指令: Size 为 8, 3 周期 Size 为 16, 4 周期 Size 为 32, 5 周期
	VMULI.T.E	3-5	拆分 2 条微指令 每条拆分指令: Size 为 8, 3 周期 Size 为 16, 4 周期 Size 为 32, 5 周期
	VMULA.T	4-5	Size 为 8, 4 周期 Size 为 16, 5 周期 Size 为 32, 5 周期
	VMULAI.T	4-5	Size 为 8, 4 周期 Size 为 16, 5 周期 Size 为 32, 5 周期
	VMULA.T.E	4-5	拆分 2 条微指令 每条拆分指令: Size 为 8, 4 周期 Size 为 16, 5 周期 Size 为 32, 5 周期
	VMULAI.T.E	4-5	拆分 2 条微指令 每条拆分指令: Size 为 8, 4 周期 Size 为 16, 5 周期 Size 为 32, 5 周期

下页继续

表 5.42 – 续上页

指令类型	指令	执行周期	备注
	VMULS.T	4-5	Size 为 8, 4 周期 Size 为 16, 5 周期 Size 为 32, 5 周期
	VMULSI.T	4-5	Size 为 8, 4 周期 Size 为 16, 5 周期 Size 为 32, 5 周期
	VMULS.T.E	4-5	拆分 2 条微指令 每条拆分指令: Size 为 8, 4 周期 Size 为 16, 5 周期 Size 为 32, 5 周期
	VMULSI.T.E	4-5	拆分 2 条微指令 每条拆分指令: Size 为 8, 4 周期 Size 为 16, 5 周期 Size 为 32, 5 周期
	VMULACA.T	4-5	Size 为 8, 4 周期 Size 为 16, 5 周期
	VMULACA.T	4-5	Size 为 8, 4 周期 Size 为 16, 5 周期
	VMULACAA.T	4-5	Size 为 8, 4 周期 Size 为 16, 5 周期
	VMULACAAI.T	4-5	Size 为 8, 4 周期 Size 为 16, 5 周期
	VRMUL.T.SE	3-5	拆分 2 条微指令 每条拆分指令: Size 为 8, 3 周期 Size 为 16, 4 周期 Size 为 32, 5 周期
	VRMULI.T.SE	3-5	拆分 2 条微指令 每条拆分指令: Size 为 8, 3 周期 Size 为 16, 4 周期 Size 为 32, 5 周期
	VRMULH.T.S	3-5	Size 为 8, 3 周期 Size 为 16, 4 周期 Size 为 32, 5 周期

下页继续

表 5.42 – 续上页

指令类型	指令	执行周期	备注
	VRMULHI.T.S	3-5	Size 为 8, 3 周期 Size 为 16, 4 周期 Size 为 32, 5 周期
	VRMULH.T.RS	3-5	Size 为 8, 3 周期 Size 为 16, 4 周期 Size 为 32, 5 周期
	VRMULHI.T.RS	3-5	Size 为 8, 3 周期 Size 为 16, 4 周期 Size 为 32, 5 周期
	VRMULHA.T.RS	4-5	Size 为 8, 4 周期 Size 为 16, 4 周期 Size 为 32, 5 周期
	VRMULHAI.T.RS	4-5	Size 为 8, 4 周期 Size 为 16, 4 周期 Size 为 32, 5 周期
	VRMULHS.T.RS	4-5	Size 为 8, 4 周期 Size 为 16, 4 周期 Size 为 32, 5 周期
	VRMULHSI.T.RS	4-5	Size 为 8, 4 周期 Size 为 16, 4 周期 Size 为 32, 5 周期
	VRMULSHR.T.E	3-5	拆分 2 条微指令 每条拆分指令: Size 为 8, 3 周期 Size 为 16, 4 周期 Size 为 32, 5 周期
	VRMULSHRI.T.E	3-5	拆分 2 条微指令 每条拆分指令: Size 为 8, 3 周期 Size 为 16, 4 周期 Size 为 32, 5 周期
	VRMULSA.T.E	4-5	拆分 2 条微指令 每条拆分指令: Size 为 8, 4 周期 Size 为 16, 5 周期 Size 为 32, 5 周期

下页继续

表 5.42 – 续上页

指令类型	指令	执行周期	备注
	VRMULSAI.T.E	4-5	拆分 2 条微指令 每条拆分指令： Size 为 8, 4 周期 Size 为 16, 5 周期 Size 为 32, 5 周期
	VRMULSS.T.E	4-5	拆分 2 条微指令 每条拆分指令： Size 为 8, 4 周期 Size 为 16, 5 周期 Size 为 32, 5 周期
	VRMULSSI.T.E	4-5	拆分 2 条微指令 每条拆分指令： Size 为 8, 4 周期 Size 为 16, 5 周期 Size 为 32, 5 周期
	VRMULXAA.T.RS	4-5	Size 为 8, 4 周期 Size 为 16, 5 周期 Size 为 32, 5 周期
	VRMULXAAI.T.RS	4-5	Size 为 8, 4 周期 Size 为 16, 5 周期 Size 为 32, 5 周期
	VRMULXAS.T.RS	4-5	Size 为 8, 4 周期 Size 为 16, 5 周期 Size 为 32, 5 周期
	VRMULXASI.T.RS	4-5	Size 为 8, 4 周期 Size 为 16, 5 周期 Size 为 32, 5 周期
	VRMULXSS.T.RS	4-5	Size 为 8, 4 周期 Size 为 16, 5 周期 Size 为 32, 5 周期
	VRMULXSSI.T.RS	4-5	Size 为 8, 4 周期 Size 为 16, 5 周期 Size 为 32, 5 周期
	VRMULXSA.T.RS	4-5	Size 为 8, 4 周期 Size 为 16, 5 周期 Size 为 32, 5 周期
	VRMULXSAL.T.RS	4-5	Size 为 8, 4 周期 Size 为 16, 5 周期 Size 为 32, 5 周期

下页继续

表 5.42 – 续上页

指令类型	指令	执行周期	备注
	VRCMUL.T.RS	5-6	Size 为 8, 5 周期 Size 为 16, 6 周期 Size 为 32, 6 周期
	VRCMULA.T.E	5-6	拆分 2 条微指令 每条拆分指令: Size 为 8, 5 周期 Size 为 16, 6 周期 Size 为 32, 6 周期
	VRCMULC.T.RS	5-6	Size 为 8, 5 周期 Size 为 16, 6 周期 Size 为 32, 6 周期
	VRCMULCA.T.E	5-6	拆分 2 条微指令 每条拆分指令: Size 为 8, 5 周期 Size 为 16, 6 周期 Size 为 32, 6 周期
	VRCMULN.T.RS	5-6	Size 为 8, 5 周期 Size 为 16, 6 周期 Size 为 32, 6 周期
	VRCMULNA.T.E	5-6	拆分 2 条微指令 每条拆分指令: Size 为 8, 5 周期 Size 为 16, 6 周期 Size 为 32, 6 周期
	VRCMULCN.T.RS	5-6	Size 为 8, 5 周期 Size 为 16, 6 周期 Size 为 32, 6 周期
	VRCMULCNA.T.E	5-6	拆分 2 条微指令 每条拆分指令: Size 为 8, 5 周期 Size 为 16, 6 周期 Size 为 32, 6 周期
杂类运算指令	VRECPE.T	3-5	Size 为 8, 5 周期 Size 为 16, 6 周期 Size 为 32, 6 周期
	VRECPS.T	3-5	Size 为 8, 5 周期 Size 为 16, 6 周期 Size 为 32, 6 周期

下页继续

表 5.42 – 续上页

指令类型	指令	执行周期	备注
	VRSQRTE.T	3-5	Size 为 8, 5 周期 Size 为 16, 6 周期 Size 为 32, 6 周期
	VRSQRTS.T	3-5	Size 为 8, 5 周期 Size 为 16, 6 周期 Size 为 32, 6 周期
	VEXPE.T	2	
	VLRW.32.N	≥ 4	L1 cache 命中: 4 个周期 L2 cache 命中: ≥ 17 个周期 L2cache 缺失: 片外访问延迟
	VLD.T.N	≥ 4	
	VLDX.T	≥ 4	
	VLDU.T.N	≥ 5	拆分成 VLD.T.N 和 ADDI 指令
	VLDRU.T.N	≥ 5	拆分成 VLD.T.N 和 ADD 指令
	VLDM.T	$\geq 4+N-1$	拆分为 N 条 VLD.T 指令。
	VLDMU.T	$\geq 4+N-1$	拆分为 N 条 VLD.T 指令和 1 条 ADDI 指令。
	VLDMRU.T	$\geq 4+N-1$	拆分为 N 条 VLD.T 指令和 N 条 ADD 指令
	VST.T.N	4	
	VSTX.T	4	
	VSTU.T.N	5	拆分成 VLD.T.N 和 ADDI 指令
	VSTRU.T.N	5	拆分成 VLD.T.N 和 ADD 指令
	VSTM.T	$4+N-1$	拆分为 N 条 VST.T 指令。
	VSTMU.T	$4+N-1$	拆分为 N 条 VST.T 指令和 1 条 ADDI 指令。
	VSTMRU.T	$4+N-1$	拆分为 N 条 VST.T 指令和 N 条 ADD 指令

第六章 内存管理

6.1 虚拟地址空间介绍

玄铁 CPU 的虚拟内存地址空间可运行在两个权限级别上：普通用户模式（也叫“用户态”）和超级用户模式（也叫“内核态”），如下图所示。

超级用户模式可以访问所有的 4GB 虚拟地址空间。

普通用户只能访问 0x00000000~0x7FFFFFFF 表示的 2GB 空间，在访问 0x80000000~0xFFFFFFFF 表示的 2GB 空间时，出现访问错误。

从虚拟地址映射的角度，4GB 空间被划分为 4 个区：

USEG (0x00000000 ~ 0x7FFFFFFF)：2GB 的用户段。用户态和内核态都能正常访问。用户态在 MMU 被初始化之前不能使用这个区域。

SSEG0 (0x80000000 ~ 0x9FFFFFFF)：512MB 内核模式下的不可映射、可 Cache 空间。这个空间的“不可映射”，指的是落入这个区的虚拟地址会被强制减掉 0x80000000 后作为物理地址直接返回（即物理地址是通过虚拟地址减去 0x80000000 获得）。上述为默认情况，其属性和对应的物理地址可以通过 (MSA0, CR<30, 15>) 来配置。

SSEG1 (0xA0000000 ~ 0xBFFFFFFF)：512MB 内核模式下的不可映射、不可 Cache 空间。这个空间的“不可映射”，指的是落入该区域的虚拟地址会被强制减掉 0xA0000000 后作为物理地址直接返回（即物理地址是通过虚拟地址减去 0xA0000000 获得）。由于它的不可 Cache 属性，该区通常作为外设的映射空间（若原外设的物理地址为 0x10000000，其虚拟地址应该设置为 0xB0000000）。上述为默认情况，其属性和对应的物理地址可以通过 (MSA1, CR<31, 15>) 来配置。

SSEG2 (0xC0000000 ~ 0xFFFFFFFF)：1GB 内核模式下的可映射、可 Cache 空间。这个空间允许通过 MMU 的 TLB 进行物理地址的灵活映射。

【注意】SSEG0 和 SSEG1 必须至少使用 1 个。中断向量入口地址必须要放在 SSEG0 或者 SSEG1 区域。

C860 通过第 15 组控制寄存器来实现内存管理单元 (MMU) 的功能。系统程序员在超级用户编程模式下来通过设置与 MMU 相关的状态寄存器，来进行 MMU 的管理和操作。这些寄存器包括：

- MMU 索引寄存器 (MIR)；
- MMU EntryLo0 寄存器 (MEL0)；
- MMU EntryLo1 寄存器 (MEL1)；

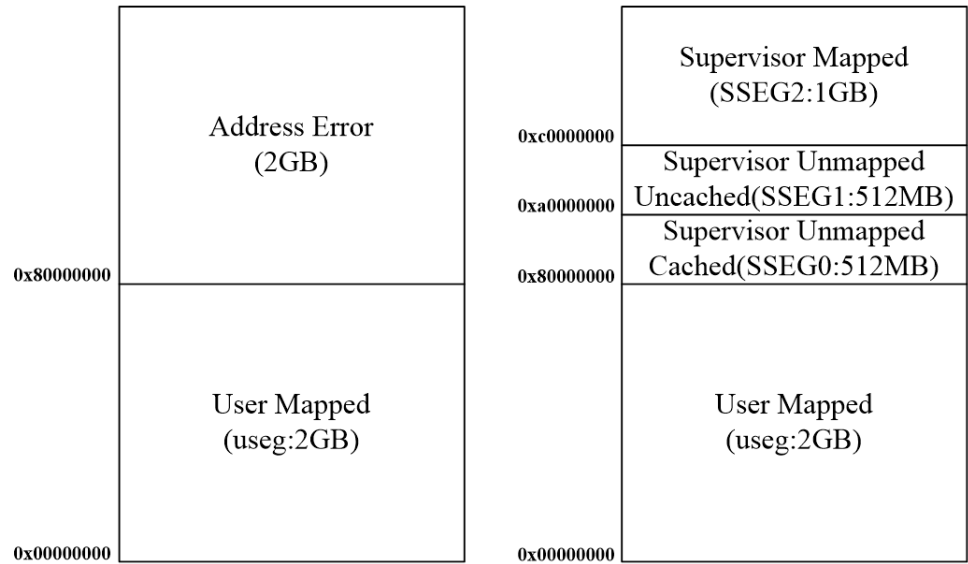


图 6.1: 虚拟地址空间划分图

- MMU EntryHi/Bad VPN 寄存器 (MEH);
- MMU 页掩码寄存器 (MPR);
- MMU 控制指令寄存器 (MCIR);
- MMU 用户模式 PGD 基址及硬件回填使能控制寄存器 (MPGD0) ;
- MMU 超级用户模式 PGD 基址及硬件回填使能控制寄存器 (MPGD1);
- MMU SSEG0 配置寄存器 (MSA0) ;
- MMU SSEG1 配置寄存器 (MSA1)。

6.2 MMU 相关控制寄存器

6.2.1 MMU 索引寄存器 (MIR, CR<0,15>)

MIR 寄存器用于指示读写操作的 jTLB 表项，或存放 TLBP 指令匹配的结果，其各位如下图所示：

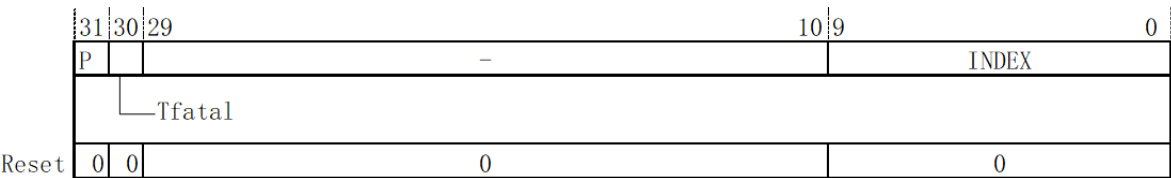


图 6.2: MMU 索引寄存器

P -匹配失败标志：

当执行 TLBP 指令时，硬件写该位指示 jTLB 是否匹配成功。

当 P 为 0 时，jTLB 匹配成功；

当 P 为 1 时，jTLB 匹配失败。

TFATAL-TLB 不可恢复异常标志：

该位用于表征是否出现 TLB 不可恢复异常；当 tfatal 为 1 时，表示出现 TLB 不可恢复异常。

INDEX-jTLB 索引：

软件写该域以提供 TLB 表项的索引，参考 TLBR 和 TLBWI 用法说明。

当执行 TLBP 指令时，硬件用匹配的 jTLB 表项的索引写该域。如果 TLBP 指令匹配失败，该域的内容不可预知。

jTLB 表可配置为 512 或 1024 个表项，jTLB 表项的索引号为 0 至 511/1023。

6.2.2 MMU EntryLo0 和 EntryLo1 寄存器 (MEL0 & MEL1, CR<2,15> & CR<3,15>)

MEL 寄存器保存 jTLB 访问的物理地址高位和页面属性信息，其各位如下图所示：

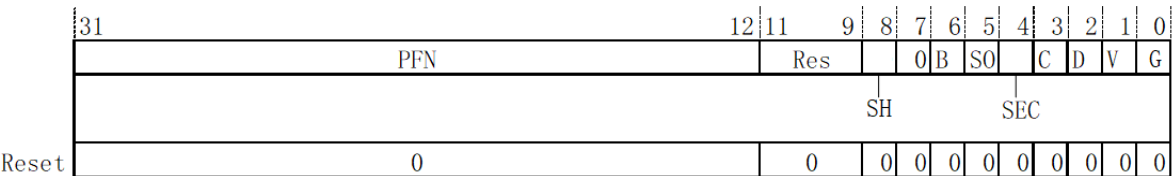


图 6.3: MMU EntryLo0 和 EntryLo1 寄存器

PFN-页帧号：

该域对应于物理地址的 31 位到 12 位。

SH-Shareable

表示页面的共享属性。

SH=0：不共享；

SH=1：共享。

B-页 bufferable 位：

指示当前页是否可 bufferable。

当 B 为 0 时，当前页不可暂存；

当 B 为 1 时，当前页可暂存。

SO-Strong Order 位：

该位使能时，总线上内存的读写访问顺序和程序流的读写访问顺序相同。

SEC-安全访问位:

指示当前页是否支持安全访问。

C-可高缓位:

指示当前页是否可高缓。

当 C 为 0 时，当前页不可高缓；

当 C 为 1 时，当前页可高缓。

D-Dirty 位:

指示当前页是否可写。

当 D 为 1 时，当前页可写；

当 D 为 0 时，当前页不可写，写该页将产生 TLB 修改异常。

V-有效位:

指示 jTLB 表项虚拟页映射是否有效。

当 V 为 1 时，该页有效，可访问；

当 V 为 0 时，该页无效，不可访问，访问该页将产生 TLB 无效异常。

G-全局位:

表示当前页是否为全局页。

当 G 为 0 时，表示当前页不是全局页，进程号 ASID 私有，TLB 匹配过程考虑 ASID。

当 G 为 1 时，表示当前页是全局页，TLB 匹配过程忽略 ASID。

6.2.3 MMU EntryHi/Bad VPN 寄存器 (MEH, CR <4,15>)

MEH 寄存器有两个功能：包含 TLB 访问的虚拟地址信息和 TLB 异常时当前 VPN 的值。ASID 表示当前页面对应的进程号。其各位如下图所示：

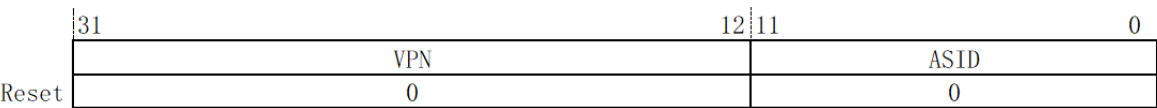


图 6.4: MMU EntryHi/Bad VPN 寄存器

VPN-虚拟页序号:

该域在 TLB 读和 TLB 异常时由硬件写，在软件写 TLB 表项之前由软件预先写入。

注意：在 TLB 访问时，该域的最低位无意义。在 TLB 异常时，该域的最低位有意义。软件将通过该位分辨哪个页面出现异常。例如，当产生 TLB 修改异常时，软件将通过该位找到对应的发生修改异常的页面，然后设置相应的 Dirty 位。

ASID-地址空间标识：

该域通常用来保存操作系统所看到的当前地址空间的标识，异常不会改变其值，因此在 TLB 异常发生后，它依然保存着当前运行进程的正确地址空间标识。

绝大部分软件系统会把当前进程的地址空间标识写入该域。在使用 TLBR 指令读取 TLB 表项时必须要小心，该操作会覆盖整个 MEH 寄存器，因此在执行该操作之后必须恢复当前进程地址空间的标识。

在软件读写 TLB 表项时，该域用于指示当前表项所属的进程号；在 MMU 进行地址映射时，该域用于指示当前进程号。

该域在软件读 TLB 表项时由硬件设置；在软件写 TLB 表项时由软件设置。

6.2.4 MMU 页掩码寄存器 (MPR, CR<6,15>)

MPR 寄存器用于配置页面的大小，其各位如下图所示：

	31	25	24	13	12	0
	–	Page Mask			–	
Reset	0	0			0	

图 6.5: MMU 页掩码寄存器

Mask-页掩码：

该域的位为 1 表示虚拟地址的对应位不用于 TLB 匹配。

页掩码的编码如 表 6.1 所示：

表 6.1: 页掩码编码

页掩码	页大小	奇偶块选择位
0000_0000_0000	4KB	VAddr[12]
0000_0000_0011	16KB	VAddr[14]
0000_0000_1111	64KB	VAddr[16]
0000_0011_1111	256KB	VAddr[18]
0000_1111_1111	1MB	VAddr[20]
0011_1111_1111	4MB	VAddr[22]
1111_1111_1111	16MB	VAddr[24]

如果软件装载到 MPR 寄存器的值与上面列出的值都不一样，处理器的操作将不可预知。

6.2.5 MMU 控制指令寄存器 (MCIR, CR<8,15>)

TLBP-TLB 查找：

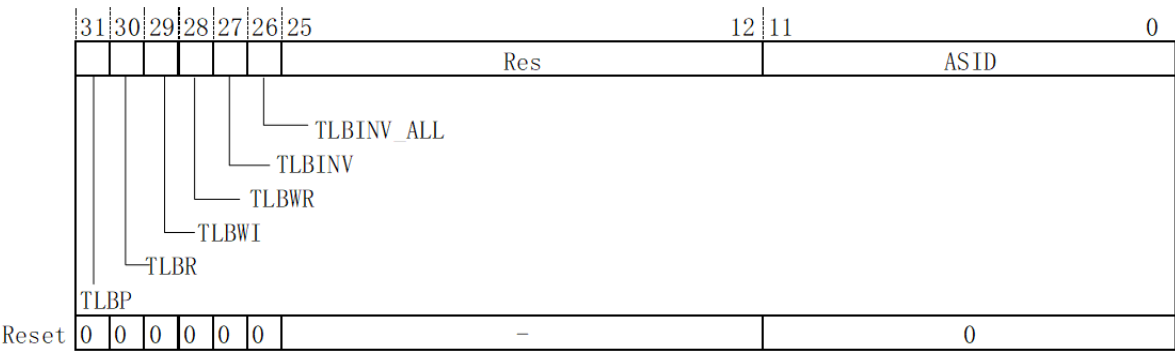


图 6.6: MMU 控制指令寄存器

该控制指令用于在由 MEH 寄存器 VPN 域低位指定的两路 jTLB 和全部 vTLB 表项中查找虚拟页号和 ASID (G 位有效时不比较 ASID) 跟 MEH 寄存器中内容相匹配的表项, 并把相应表项的索引值存入 MIR 寄存器。如果匹配失败, MIR 寄存器的 P 位被置 1。

TLBR-TLB 读:

该控制指令用于从 TLB 读一个表项。由 MIR 指向的 jTLB 表项的内容被装载到 MEH、MEL0 和 MEL1 寄存器。如果 MIR 寄存器的 Index 值无效时, 处理器的操作将不可预知。

TLBWI-TLB 写索引:

该控制指令用于写 MIR 寄存器指向的 TLB 表项, 即将 MEH、MEL0、MEL1 和 MPR 寄存器的内容写入到由 MIR 寄存器指向的 TLB 表项。如果 MIR 寄存器的 Index 值无效时, 处理器的操作将不可预知。

TLBWR-TLB 写随机:

该控制指令用于写 MEH 寄存器 VPN 域低位指向的 jTLB 表项, 即将 MEH、MEL0、MEL1 和 MPR 寄存器的内容写入到 MEH 寄存器 VPN 域低位指向的 jTLB 表项。

TLBINV-TLB 无效:

该控制指令用于删除与 ASID 域值相同的所有 TLB 表项。

TLBINV_ALL-TLB 无效全部表项:

该控制指令用于删除与 jTLB 中的所有 TLB 表项。

ASID-再循环 ASID:

该域仅仅对 TLB 无效指令有意义。如果存在多于 4096 个进程, 可能需要再循环 ASID。

MMU 指令的优先级: TLBINV_ALL>TLBINV>TLBP>TLBWI>TLBWR>TLBWR。当同时设置两个以上的指令域时, 按照优先级仅且一个 MMU 指令有效。

6.2.6 MMU PGD 基址寄存器 (MPGD0 & MPGD1, CR<CR28,15> & CR<29,15>)

MPGD 寄存器包含 PGDBase (物理地址) 和硬件回填使能控制位。在 TLB 缺失且硬件回填使能时, 不产生 TLB 失配异常, 操作系统从内存中读取回填失配的 TLB。其各位如下图所示:

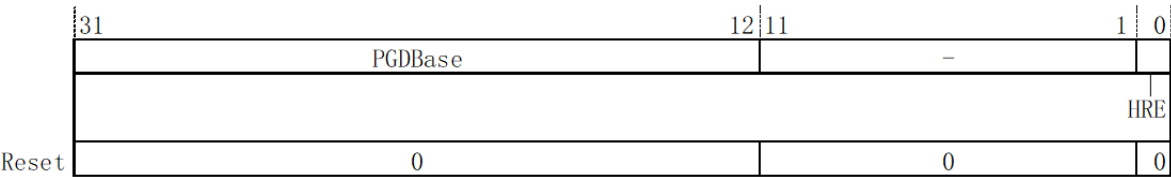


图 6.7: MMU PGD 基址寄存器

PGDBase-PGD 表项基址:

此为 PGD 表的基址, 为物理地址。

HRE-硬件回填使能:

此为硬件回填使能控制位, 当有效时 TLB 发生了失配, MMU 单元会通过 BIU 单元到内存中读取相关表项。

6.2.7 MMU SSEG0 配置寄存器 (MSA0, CR<30,15>)

MSA0 寄存器配置 SSEG0 区映射的物理地址和该区属性的信息, 其各位如下图所示:

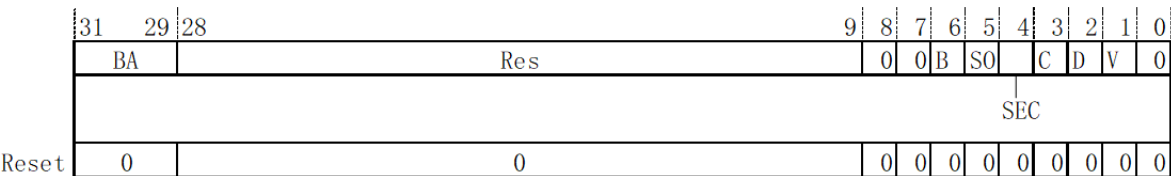


图 6.8: MMU MSA0 寄存器

BA-SSEG0 映射的物理地址:

该域对应于物理地址的 31 位到 29 位。

B-页 bufferable 位:

指示 SSEG0 区是否可 bufferable。

当 B 为 0 时, 当前页不可暂存;

当 B 为 1 时, 当前页可暂存。

SO-Strong Order 位:

该位使能时, 总线上内存的读写访问顺序和程序流的读写访问顺序相同。

SEC-安全访问位:

指示 SSEG0 区是否支持安全访问。

C-可高缓位:

指示 SSEG0 区是否可高缓。

当 C 为 0 时，当前页不可高缓；

当 C 为 1 时，当前页可高缓。

D-脏 (Dirty) 位:

指示 SSEG0 区是否可写。

当 D 为 1 时，当前页可写；

当 D 为 0 时，当前页不可写，写该页将产生 TLB 修改异常。

V-有效位:

指示 SSEG0 区映射是否有效。

当 V 为 1 时，该区有效，可访问；

当 V 为 0 时，该区无效，不可访问，访问该页将产生 TLB 无效异常。

6.2.8 MMU SSEG1 配置寄存器 (MSA1, CR<31,15>)

MSA1 寄存器配置 SSEG1 区映射的物理地址和该区属性的信息，其各位如下图所示：

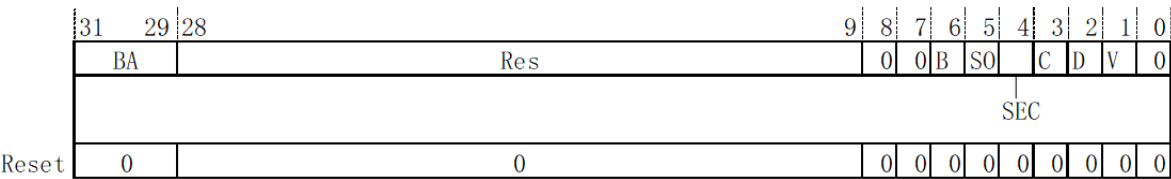


图 6.9: MMU MSA1 寄存器

BA-SSEG1 映射的物理地址:

该域对应于物理地址的 31 位到 29 位。

B-页 bufferable 位:

指示 SSEG1 区是否可 bufferable。

当 B 为 0 时，当前页不可暂存；

当 B 为 1 时，当前页可暂存。

SO-Strong Order 位:

该位使能时，总线上内存的读写访问顺序和程序流的读写访问顺序相同。

SEC-安全访问位:

指示 SSEG1 区是否支持安全访问。

C-可高缓位:

指示 SSEG1 区是否可高缓。

当 C 为 0 时，当前页不可高缓；

当 C 为 1 时，当前页可高缓。

D-脏 (Dirty) 位:

指示 SSEG1 区是否可写。

当 D 为 1 时，当前页可写；

当 D 为 0 时，当前页不可写，写该页将产生 TLB 修改异常。

V-有效位:

指示 SSEG1 区虚拟页映射是否有效。

当 V 为 1 时，该页有效，可访问；

当 V 为 0 时，该页无效，不可访问，访问该页将产生 TLB 无效异常。

6.3 TLB 表项结构

TLB 包含两部分：比较部分和物理转换部分。比较部分包括表项的虚拟页序号（因为每个表项都映射到两个物理页，所以事实上是虚拟页序号/2 (VPN2)，ASID 和 G 位。物理转换部分包括一对表项，每个表项都包含物理页帧序号 (PFN)，有效 (V) 位，dirty (D) 位和可高缓域 (C)。每个 jTLB 表项对应两个物理页面并包含一个表项有效位。TLB 表项的位定义如下图所示：

87	69	68	57	56	55	36	35	28	27	8	7	0
VPN2		ASID		G	PFN1			Flags1	PFN0			Flags0

图 6.10: jTLB 表项

第七章 内存子系统

C860MP 采用二级缓存架构：哈佛结构的一级高速缓存和共享的二级高速缓存，其中，一级高速缓存包含独立的指令高速缓存和数据高速缓存。

7.1 指令高速缓存子系统

L1 指令高速缓存的主要特征如下：

- 指令高速缓存大小硬件可配置，支持 32KB/64KB；
- 2 路组相联，缓存行大小为 64B；
- 虚拟地址索引，物理地址标记；
- 访问数据位宽为 128 比特；
- 采用先进先出的替换策略；
- 支持对整个指令高速缓存的无效操作，支持对单条缓存行的无效操作；
- 支持指令预取功能；
- 支持路预测。

7.1.1 指令预取

L1 指令高速缓存支持指令预取功能，通过配置隐式操作寄存器 HINT.IPLD 实现。在当前缓存行访问缺失时，开启下一条连续缓存行的预取，并将预取结果缓存到预取缓冲器中。当指令访问命中预取缓冲器时，直接从缓冲器获取指令并回填指令高速缓存，从而降低了取指延迟。

指令预取额外要求预取的缓存行与当前访问缓存行位为同一个页面，否则指令预取功能关闭，从而保证取指地址的安全。此外，读敏感的外设地址空间也禁止被分配到指令区空间。

7.1.2 路预测

C860 指令高速缓存采用两路组相联结构，为了减少并行访问两路缓存的功耗，C860 实现了指令高速缓存的路预测功能。在路预测信息有效时，关闭无效数据路的访问，仅访问预测路的数据。用户可以通过配置

隐式操作寄存器 HINT.IWPE 使能指令高速缓存的路预测功能。

根据取指行为的不同，路预测可分为以下两类：

- 顺序访问：当进行连续行内取指时，根据上次访问的路命中信息预测此次访问的路信息。
- 跳转访问：分支指令在获取跳转目标地址的同时获取了目标缓存行的路预测信息，并根据该信息访问其中一路缓存。

7.1.3 循环加速缓存器

针对程序中存在的大量短循环，C860 设置了一个 32B 的循环加速缓存器。当检测到短循环指令序列时，该循环体被加载到循环加速缓存器中；当后续取指命中该缓存器时，直接从缓存器中获取指令以及跳转的目标地址，关闭对指令高速缓存和分支历史表、分支跳转目标预测器的访问，从而降低了取指的动态功耗。用户可以通过配置隐式操作寄存器 HINT.LPE 使能短循环加速功能。

7.1.4 分支历史表

C860 采用分支历史表对条件分支的跳转方向进行预测。分支历史表容量为 64K Bit，使用 BI-MODE 预测器作为预测机制，每周期支持一条分支结果预测。

分支历史表由预测器和选择器两部分组成，其中预测器又分为跳转预测器和非跳转预测器，并根据分支历史信息对各预测器进行实时维护。分支历史表通过分支历史信息以及当前分支指令地址对各路进行索引，获得分支指令跳转方向的预测结果。

分支历史表进行预测的条件分支指令包括：

- BT16、BT32、BF16、BF32
- BEZ32、BNEZ32、BHZ32、BLSZ32、BLZ32、BHSZ32。

7.1.5 分支跳转目标预测器

C860 使用分支跳转目标预测器对分支指令的跳转目标地址进行预测。分支跳转目标预测器对分支指令历史目标地址进行记录，如果当前分支指令命中分支跳转目标预测器，则将记录目标地址作为当前分支指令预测目标地址。

分支跳转目标预测器主要特征包括：

- 硬件可配，支持 1024 表项和 2048 表项两种配置；
- 两路组相联结构，根据分支指令低位 PC 选择替换；
- 维护指令高速缓存路预测信息；
- 使用当前分支指令部分 PC 进行索引；

分支跳转目标预测器进行预测的分支指令包括：

- BT16、BT32、BF16、BF32;
- BEZ32、BNEZ32、BHZ32、BLSZ32、BLZ32、BHSZ32;
- BR16、BR32、BSR16、BSR32。

7.1.6 间接分支预测器

C860 使用间接分支预测器负责对间接分支的目标地址进行预测。间接分支指令通过寄存器获取目标地址，一条间接分支指令可包含多个分支目标地址，无法通过传统分支跳转目标预测器进行预测。因此 C860 采用基于分支历史的间接分支预测机制，将间接分支指令的历史目标地址与该分支之前的分支历史信息进行关联，用不同的分支历史信息将同一条间接分支的不同目标地址进行离散，从而实现多个不同目标地址的预测。

C860 间接分支预测器硬件可配，支持 256 表项间接分支预测器 IBP_PRO 和 8 表项间接分支预测器 IBP_LITE 选配。

间接分支指令包括：

- JSR16、JSR32、JMP16（除 R15）、JMP32（除 R15）;
- JMPI32、JSRI32。

7.1.7 返回地址预测器

返回地址预测器用于函数调用结束时，返回地址的快速准确预测。当取指单元译码得到有效的函数调用指令时，将函数返回地址压栈存入返回地址预测器；当取指单元译码得到有效的函数返回指令时，则从返回地址预测器弹栈，获取函数返回目标地址。

返回地址预测器最多支持 12 层函数调用嵌套，超出嵌套次数会导致目标地址预测错误。

函数调用指令包括：

- JSR16、JSR32;
- BSR16、BSR32;
- JSRI32。

函数返回指令包括：

- JMP16 R15, JMP32 R15;
- RTS16;
- POP16、POP32。

7.2 数据高速缓存子系统

L1 数据高速缓存的主要特征如下：

- 数据高速缓存大小硬件可配置，支持 32KB/64KB；
- 2 路组相联，缓存行大小为 64B；
- 物理地址索引，物理地址标记；
- 每次读访问的最大宽度为 128 比特，支持字节/半字/字/双字/四字访问；
- 每次写访问的最大宽度为 256 比特，支持任意字节组合的访问；
- 写策略支持写回-写分配模式和写回-写不分配模式；
- 高速缓存仅支持写回模式；
- 采用先进先出的替换策略；
- 支持对整个指令高速缓存的无效和清除操作，支持对单条缓存行的无效和清除操作；
- 指令多通道的数据预取功能；

7.2.1 Cache 一致性

对于页面属性配置为 cacheable 的请求将默认附带 shareable 属性，硬件将维护数据在不同核心的 L1 数据高速缓存的一致性。

C860MP 采用 MESI 协议维护多个处理器核心数据高速缓存的一致性。MESI 代表了每个缓存行在数据高速缓存上的 4 个状态，分别是：

M：表示缓存行仅位于此数据高速缓存中，且被写脏；(UniqueDirty)

E：表示缓存行仅位于此数据高速缓存中，且是干净的；(UniqueClean)

S：表示缓存行可能位于多个数据高速缓存中，且是干净的；(ShareClean)

I：表示缓存行不在该数据高速缓存中。(Invalid)

7.2.2 多通道数据预取

为了减少 DDR 等大内存的存储访问延时，C860 支持数据预取功能。通过检测数据高速缓存的缺失，匹配出固定的访问模式，然后硬件自动预取缓存行并回填 L1 数据高速缓存。C860 最多支持 8 个通道的数据预取，并实现了连续预取和间隔预取 (stride ≤ 32 个缓存行) 这 2 种不同的预取方式。此外，还实现了正向预取和反向预取 (即 stride 为负数)，从而支持各种可能的访问模式。

在处理器执行数据高速缓存无效和清除操作时，停止数据预取功能。用户通过设置隐式寄存器 HINT.DPLD 使能数据预取功能，并通过设置 HINT.DPLD_DIS 决定一次预取的缓存行数量。

支持数据预取的指令如下：

- LD16.B、LD32.B、LD32.BS、LD16.H、LD32.H、LD32.HS、LD16.W、LD32.W、LD32.D；
- LDR32.B、LDR32.BS、LDR32.H、LDR32.HS、LDR32.W；
- LRS32.B、LRS32.H、LRS32.W；

- LDQ32、LDM32。

7.2.3 自适应的写分配机制

C860 提供自适应的写分配机制，当处理器检测到连续的内存写入操作时，页面的写分配属性会自动关闭。用户可通过设置隐式寄存器 HINT.AMRL1 和 HINT.AMRL2 决定连续存储多少条缓存行后关闭 L1 数据高缓的写分配和二级高缓的写分配策略。

当执行数据高缓的无效或者清除操作时，处理器的自适应写分配机制会被关闭，在高速缓存操作完成后，重新检测内存连续写入行为。

支持自适应写分配的指令如下：

- ST16.B、ST32.B、ST16.H、ST32.H、ST16.W、ST32.W
- STR32.B、STR32.H、STR32.W
- SRS32.B、SRS32.H、SRS32.W

7.2.4 独占式访问 (local monitor 和 global monitor)

C860 支持独占式的内存访问指令 LDEX32.W 和 STEX32.W。用户可以使用这两条指令构成原子锁等同步原语实现同一个核不同进程之间或者不同核之间的同步。通过 LDEX 指令标记需要独占访问的地址，STEX 指令判断被标记的地址是否被其他进程抢占。C860 为每个核心上设置了一个位于 L1 数据高缓的局部监测器和一个位于二级高缓的全局监测器。每个监测器由一个状态机和一个地址缓存器组成，其中，状态机包含两个状态：IDLE 和 EXCLUSIVE。

对于属性设置为可高缓的页面，通过局部监测器就能够实现独占式访问。LDEX 指令在执行过程中设置局部监测器的状态机为 EXCLUSIVE 态并将访问的地址保存到底址缓存器中；STEX 指令在执行过程中读取局部监测器的状态和地址，如果状态为 EXCLUSIVE 并且地址匹配，那么执行该写操作，返回写成功，并清除状态机回到 IDLE 态；否则如果状态或者地址有一项不满足条件，或者数据高速缓存未使能时，不执行该写操作，返回写失败，并清除状态机回到 IDLE 态。其他核的写操作在地址匹配局部监测器时，也会将状态机清回到 IDLE 态；不同地址的独占访问不影响局部监测器。此外，在进程切换时需要清除局部监测器。

对于属性设置为不可高缓的页面，需要局部监测器和全局监测器共同作用实现独占式访问。LDEX 在执行过程中不仅要设置局部监测器还需要设置全局监测器；STEX 在局部监测器检查通过后需要进一步检查全局监测器，只有当全局监测器也通过检查，才执行写操作、返回写成功，清除状态机；否则不执行写操作，返回写失败，清除状态机。其他核的写操作在地址匹配某个全局监测器时，会将该全局监测器的状态清回 IDLE 态。

7.2.5 Preload 指令

C860 支持 PLDR 和 PLDW 预取指令，用于软件控制缓存行的预取。PLDR 和 PLDW 指令在检测到 L1 数据高缓缺失时发起缓存行的加载操作，然后将载回的缓存行回填数据高缓。PLDR 和 PLDW 指令不会产生任何异常，在执行过程中一旦检测到异常以 NOP 的方式处理。

PLDR 指令作为 LD 的隐式指令，只有在页面配置为 cacheable 时才会载入对应的缓存行，并且载入缓存行的状态可以是共享 (S) 也可以是独占 (E)；而 PLDW 指令是作为 ST 的隐式指令，在页面配置为 cacheable 且 write-allocate 时，才会载入相应的缓存行，并且载入缓存行的状态只能为独占 (E)。

7.3 二级高速缓存子系统

L2 高速缓存的主要特征如下：

- 高速缓存大小硬件可配置，支持 128KB/256KB/512KB/1MB；
- 8 路组相联/16 路组相联硬件可配置，缓存行大小为 64B；
- 物理地址索引，物理地址标记；
- 每次访问的最大宽度为 64B；
- 支持写回-写分配和写回-写不分配的写策略；
- 采用先进先出的替换策略；
- 可编程的 RAM 访问延时；
- 可选的 ECC 校验机制；
- 支持指令预取和 TLB 预取机制；
- 采用分块的流水线架构。

7.3.1 组织形式

C860MP L2 高速缓存在组织形式上采用了分块的流水线架构，将访问地址离散在两个不同的块中，允许多个访问的并行处理，从而提高 L2 高速缓存的访问效率。分块机制如 图 7.1 所示。

- TAG RAM 按照地址 PA[6] 分为两个标记子块，分别为 Tag bank0 和 Tag bank1，以支持同个时钟周期并行处理 2 个访问请求。
- 同样，DATA RAM 也按照地址 PA[6] 分为 2 个数据子块，分别为 Data bank0 和 Data bank1；对应每一个数据子块，又被进一步分为四个微块，每个微块的数据宽度为 128bit，从而实现并行获取一条缓存行的目的。

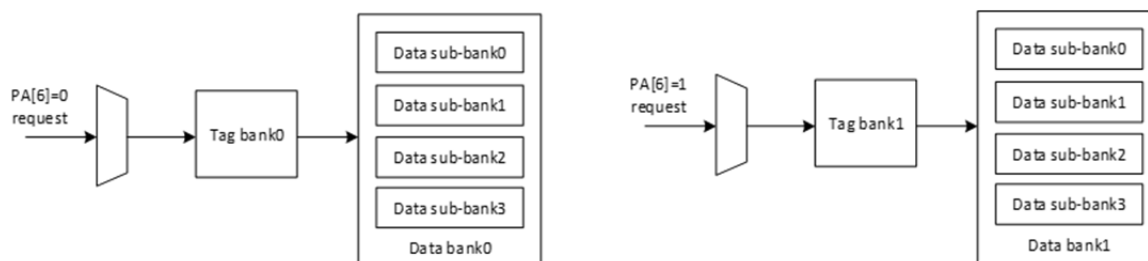


图 7.1: L2 Cache 组织形式

7.3.2 RAM 延时

由于 L2 Cache 较大，访问延迟较长，通常需要多个时钟周期才能完成访问。C860MP 提供了可配访问延迟，在不同工艺下能够根据所用 RAM 的 setup time 和 latency 进行手动设置。配置详情如 表 7.1 所示

表 7.1: RAM 访问延迟配置

配置选项	功能	说明
L2 TAG setup	L2 Cache Tag RAM setup: 1b0 0 cycle. 默认值. 1b1 1 cycle.	L2 Cache Tag RAM 的配置只影响 TAG RAM 的访问
L2 TAG latency	L2 Cache Tag RAM latency: 3b000 1 cycle. 默认值. 3b001 2 cycle. 3b010 3 cycle. 3b011 4 cycle. 3b1xx 5 cycle.	
L2 DATA setup	L2 Cache Data RAM setup: 1b0 0 cycle. 默认值. 1b1 1 cycle.	L2 Cache Data RAM 的配置只影响 DATA RAM 的访问
L2 DATA latency	L2 Data RAM latency: 3b000 1 cycle. 默认值. 3b001 2 cycle. 3b010 3 cycle. 3b011 4 cycle. 3b100 5 cycle. 3b101 6 cycle. 3b110 7 cycle. 3b111 8 cycle.	

用户根据所用 RAM 的访问时长，配置 latency; setup 默认值为 0，当所用 RAM 的 setup time 较长，或者绕线较长时，可以选择将 setup 配置成 1。在配置以上选项后，所得的访问周期数如 表 7.2 和 表 7.3 所示。

表 7.2: TAG RAM 有效访问延迟

TAG latency	TAG RAM 有效访问延迟	
-	TAG setup = 0	TAG setup = 1
000	1	2
001	2	3
010	3	4
011	4	5
1xx	5	5

表 7.3: DATA RAM 有效访问延迟

TAG latency	TAG RAM 有效访问延迟	
	TAG setup = 0	TAG setup = 1
-		
000	1	2
001	2	3
010	3	4
011	4	5
100	5	6
101	6	7
110	7	8
111	8	8

- L2 Tag latency 最大有效延迟为 5 周期；
- TAG setup 设置为 1 时增加 1 周期访问，访问 SRAM 之前会对 SRAM 输入端信号进行 flop；
- L2 Data latency 最大有效延迟为 8 周期；
- DATA setup 设置为 1 时增加 1 周期访问，访问 SRAM 之前会对 SRAM 输入端信号进行 flop。

7.3.3 ECC 校验和奇偶校验

L2 高速缓存支持可配置的 ECC 校验和奇偶校验机制，校验的数据位宽为 512 比特。开启校验机制后，L2 高速缓存在数据写入时进行 ECC 编码和奇偶编码，在数据读取时进行校验。当检测到 1-bit ECC 错误时，能够纠正错误，返回正确数据；当发生 1-bit 以上错误时，能够检测出数据错误，上报访问错误异常，返回错误的数据，并无效掉 L2 Cache 中出错的缓存行。软件可以查询 CER2 寄存器获取错误的相关信息，例如是否产生 2-bit 错误以及错误在 L2 Cache 发生的位置信息。

7.3.4 L2 Cach 预取机制

L2 高速缓存具有预取功能，能够处理取指与 TLB 访问的预取工作。

- 软件可配的指令预取数量 (0,1,2,3)，所有预取都会回填 L2 高速缓存；
- TLB 预取数固定为 1；
- 预取机制以 4KB 页表为边界，对于预取时发生跨 4KB 边界的地址会主动停止预取；
- 通过 CCR2 对预取机制进行配置。

第八章 总线接口协议

本章描述了 C860MP 处理器的接口总线协议。C860MP 的总线支持在处理器和其他外围设备之间同步传输数据。包括数据存取、地址控制和总线控制。

8.1 主设备接口

C860MP 的主设备接口支持 AMBA AXI4 协议 (请参考 AMBA 规格说明—AMBA® AXI™ and ACE™ Protocol Specification)。

8.1.1 特点

主设备接口负责 C860MP 和 AXI 系统总线之间的地址控制和数据传输，其基本特点包括：

- 支持 AMBA4.0 AXI 总线协议；
- 支持 128 位总线宽度；
- 支持系统动态变频，通过配置不同时钟频率比可动态变换时钟频率；
- 当执行 idly 指令时锁住总线；
- 所有的输出信号 flopped out、输入信号 flopped 以获得较好的时序。
- 读通道的 outstanding 能力为 $8*n+12$ ，写通道的 outstanding 能力为 $8*n+16$ 。(n 代表核心数量)

8.1.2 协议内容

8.1.2.1 支持传输类型

主设备接口支持的传输特性如下：

- BURST 支持 INCR 和 WRAP4 传输，其它突发类型均不支持；
- 支持独占式访问；
- SIZE 支持四字、字、半字和字节传输，其它传输大小不支持；
- 支持读和写操作。

8.1.2.2 支持响应类型

主设备接口接受从设备的响应类型为：

- OKAY;
- EXOKAY;
- SLVERR;
- DECERR。

8.1.3 不同总线响应下的行为

表 8.1 列出了总线上出现不同总线响应时 CPU 的行为。

表 8.1: 总线异常处理

RRESP/BRESP	结果
OKAY	普通传输访问成功，或 exclusive 传输访问失败。
EXOKAY	exclusive 访问成功。
SLVERR/DECERR	访问出错，读传输产生访问错误异常，写传输忽略此错误。

8.1.4 AXI4.0 协议的主设备接口信号

表 8.2: AXI 协议接口信号

信号名	I/O	Reset	定义
读地址通道相关接口			
biu_pad_arid[7:0]	O	0	读请求地址 ID
biu_pad_araddr[31:0]	O	0	读请求地址
biu_pad_arlen[7:0]	O	0	读请求 burst 长度 00000000: 1 拍 00000011: 4 拍
biu_pad_arsize[2:0]	O	0	读请求每拍数据位宽 000: 字节 001: 半字 010: 字 111:128bits
biu_pad_arburst[1:0]	O	0	读请求对应的传输类型： 01: INCR 10: WRAP

下页继续

表 8.2 – 续上页

信号名	I/O	Reset	定义
biu_pad_arlock	O	0	读请求对应的访问方式： 0: normal access 1: exclusive access
biu_pad_arcache[3:0]	O	0	读请求对应的 memory 访问类型： 0000: Device No n-bufferable (strong order) 0001: Device Bufferable (strong order) 0011: Normal Non-cacheable Bufferable (weak order) 1111: Cacheable
biu_pad_arprot[2:0]	O	0	读请求的保护类型： 0 1 [2]: Data Instruction [1]: Secure Non-secure [0]: User Privileged
biu_pad_arvalid	O	0	读地址通道有效信号
pad_biu_arready	I	-	读地址通道 slave ready 信号
读数据通道相关接口			
pad_biu_rid[7:0]	I	-	读请求数据 ID
pad_biu_rdata[127:0]	I	-	读请求数据
pad_biu_rresp[3:0]	I	-	读请求的 response 信息 [1:0]: 00: OKAY 01: EXOKAY 10: SLVERR 11: DECERR [2]: 无意义 [3]: 无意义
pad_biu_rlast	I	-	读请求的最后一拍数据
pad_biu_rvalid	I	-	读请求数据的有效信号
biu_pad_rready	O	1	读数据通道的 ready 信息
写地址通道相关接口			
biu_pad_awid[7:0]	O	0	写请求地址 ID
biu_pad_awaddr[31:0]	O	0	写请求地址
biu_pad_awlen[7:0]	O	0	写请求 burst 长度 00000000: 1 拍 00000011: 4 拍

下页继续

表 8.2 – 续上页

信号名	I/O	Reset	定义
biu_pad_awsiz[2:0]	O	0	写请求每拍数据位宽 000: 字节 001: 半字 010: 字 111:128bits
biu_pad_awburst[1:0]	O	0	写请求对应的传输类型: 01: INCR 10: WRAP
biu_pad_awlock	O	0	写请求对应的访问方式: 0: normal access 1: exclusive access
biu_pad_awcache[3:0]	O	0	写请求对应的 memory 访问类型: 0000: Device No n-bufferable (strong order) 0001: Device Bufferable (strong order) 0011: Normal Non-cacheable Bufferable (weak order) 0111: Write-back No-allocate 1111: Write-back Cacheable
biu_pad_awprot[2:0]	O	0	写请求的保护类型: 0 1 [2]: Data Instruction [1]: Secure Non-secure [0]: User Privileged
biu_pad_awvalid	O	0	写地址通道有效信号
pad_biu_arready	I	-	写地址通道 slave ready 信号
写数据通道相关接口			
biu_pad_wdata[127:0]	O	0	写请求数据
biu_pad_wstrb[15:0]	O	0	数据中有效数据域
biu_pad_wlast	O	0	最后一拍数据
biu_pad_wvalid	O	0	写数据通道有效信号
biu_pad_wready	I	-	写数据通道 slave ready 信号
写应答通道相关接口			
pad_biu_bid[7:0]	I	-	写应答的 ID

下页继续

表 8.2 – 续上页

信号名	I/O	Reset	定义
pad_biu_bresp[1:0]	I	-	写应答信息 读请求的 response 信息 [1:0]: 00: OKAY 01: EXOKAY 10: SLVERR 11: DECERR
pad_biu_bvalid	I	-	写应答通道的 valid 信号
biu_pad_bready	O	1	写应答通道的 ready 信号

8.2 从设备接口

C860MP 从设备接口是一个用户可配置的接口，用来完成外设对 L2 Cache 和 L1 数据 Cache 的访问，维护外设和处理器片上数据的一致性。从设备接口支持 AMBA AXI4 协议（请参考 AMBA 规格说明—AMBA® AXI™ and ACE™ Protocol Specification）。

8.2.1 特点

从设备接口的基本特点包括：

- 支持 AMBA4.0 AXI 总线协议；
- 支持 128 位总线宽度；
- 支持系统动态变频，通过配置不同时钟频率比可动态变换时钟频率；
- 所有的输出信号 flopped out、输入信号 flopped in 以获得较好的时序；
- 读写分别最多支持 8 个并发的传输。

8.2.2 协议内容

8.2.2.1 支持传输类型

从设备接口支持的传输特性如下：

- 仅支持 INCR 的传输模式，且 LEN 只支持 0 和 3；
- 要求 CACHE[3:0] 为 4’ b1111、4’ b1011、4’ b0111，其它的传输均返回 SLVERR；
- 不支持独占式访问；
- WSTRB 在 LEN 为 0 时支持任意字节有效，在 LEN 为 3 时要求全为 1；
- AxADDR 在 LEN 为 0 时 16B 边界对齐，在 LEN 为 3 时 64B 边界对齐；

- 支持 5bit 的 AxID；
- 支持读和写操作。

8.2.2.2 支持响应类型

从设备接口发出的从设备的响应类型为：

- OKAY；
- SLVERR。

8.2.3 不同行为发出的响应

表 8.3 列出了从设备接口返回的响应类型。

表 8.3: 从设备响应类型

RRESP/BRESP	结果
OKAY	传输访问成功，接收到的请求得到合适的处理。
SLVERR	访问出错，接收到不支持的传输类型。

8.2.4 AXI4.0 协议的从设备接口信号

表 8.4: AXI 协议接口信号

信号名	I/O	Reset	定义
读地址通道相关接口			
pad_slvif_araddr[31:0]	I	-	读地址总线： 32 位地址总线。
pad_slvif_arburst[1:0]	I	-	突发传输指示信号： 指示传输是一次突发传输的一部分： 00: SINGLE; 01: INCR; 10: WRAP4。
pad_slvif_arcache[3:0]	I	-	读请求对应的 cache 属性： [3]:Other Accocate [2]:Allocate [1]:Modifiable [0]:Bufferable
pad_slvif_arid[4:0]	I	-	读地址 ID。

下页继续

表 8.4 – 续上页

信号名	I/O	Reset	定义
pad_slvif_arlen[7:0]	I	-	突发传输长度： 00000000: 1 拍； 00000011: 4 拍。
pad_slvif_arlock	I	-	读请求对应的访问方式： 0: normal access 1: exclusive access
pad_slvif_arprot[2:0]	I	-	读请求的保护类型： 0 1 [2]: Data Instruction [1]: Secure Non-secure [0]: User Privileged
pad_slvif_arsize[2:0]	I	-	读请求每拍数据位宽： 000: 8bits; 001:16bits; 010:32bits; 011:64bits; 100: 128bits。
pad_slvif_arvalid	I	-	读地址有效信号。
slvif_pad_arready	O	1' b1	读地址通道 ready 信号。
读数据通道相关接口			
slvif_pad_rdata[127:0]	O	128' b0	读数据总线： 128 位数据总线。
slvif_pad_rid[4:0]	O	5' b0	读数据 ID。
slvif_pad_rresp[3:0]	O	4' b0	读响应信号： [1:0]: 00: OKAY 01: EXOKAY 10: SLVERR 11: DECERR [2]: 1: IsShared [3]: 暂无意义
slvif_pad_rlast	O	1' b0	读数据最后一拍指示信号。
slvif_pad_rvalid	O	1' b0	读数据有效信号。
pad_slvif_rready	I	-	读数据通道 ready 信号。
写地址通道相关接口			
pad_slvif_awaddr[31:0]	I	-	写地址总线： 32 位地址总线。

下页继续

表 8.4 – 续上页

信号名	I/O	Reset	定义
pad_slvif_awburst[1:0]	I	-	突发传输指示信号： 指示传输是一次突发传输的一部分： 00: SINGLE; 01: INCR; 10: WRAP4。
pad_slvif_awcache[3:0]	I	-	写请求对应的 cache 属性： [3]:Other Accocate [2]:Allocate [1]:Modifiable [0]:Bufferable
pad_slvif_awid[4:0]	I	-	写地址 ID。
pad_slvif_awlen[7:0]	I	-	突发传输长度： 00000000: 1 拍; 00000011: 4 拍。
pad_slvif_awlock	I	-	写请求对应的访问方式： 0: normal access 1: exclusive access
pad_slvif_awprot[2:0]	I	-	写请求的保护类型： 0 1 [2]: Data Instruction [1]: Secure Non-secure [0]: User Privileged
pad_slvif_awsz[2:0]	I	-	写请求每拍数据位宽： 000: 8bits; 001:16bits; 010:32bits; 011:64bits; 100: 128bits。
pad_slvif_awvalid	I	-	写地址有效信号。
slvif_pad_awready	O	1' b1	写地址通道 ready 信号。
写数据通道相关接口			
pad_slvif_wdata[127:0]	I	-	写数据总线： 128 位写数据总线。
pad_slvif_wstrb[15:0]	I	-	写数据字节有效信号。
pad_slvif_wlast	I	-	写数据最后一拍指示信号。
pad_slvif_wvalid	I	-	写数据有效信号。
slvif_pad_wready	O	1' b1	写数据通道 ready 信号。
写响应通道相关信号			

下页继续

表 8.4 – 续上页

信号名	I/O	Reset	定义
slvif_pad_bid[4:0]	O	5' b0	写响应 ID。
slvif_pad_bresp[1:0]	O	2' b0	写响应信号： 00: OKAY; 01: EXOKAY; 10: SLVERR; 11: DECERR。
slvif_pad_bvalid	O	1' b0	写响应有效信号。
pad_slvif_bready	I	-	写响应通道 ready 信号。

第九章 中断控制器

9.1 概述

中断控制器（以下简称 INTC）负责中断请求的接收、仲裁和分发。支持的中断类型包括：公有中断（common interrupt），私有中断（private interrupt）以及软件中断（software interrupt）。

- 公有中断：可以由多核环境中的任意一个核进行响应的中断。该类型中断可通过软件设置为发送到某特定核进行响应；也可设置为选取任意一个核进行响应，具体选取机制由硬件实现决定。
- 私有中断：每个核独立拥有的只能由该核进行响应的中断。典型的私有中断有如各个处理器核对应的计时中断（timer interrupt）。
- 软件中断：由软件触发产生并发送给多核环境中任意数量核进行响应的中断。主要用于核间通信。

INTC 最多可支持 224 个中断，中断号对应如 表 9.1 所示。

表 9.1: 不同中断的中断向量号

中断类型	中断向量号
软件中断	0 ~ 15
私有中断	16 ~ 31
公有中断	32 ~ 223

中断可被配置为两种类型：快速中断（FINT）或普通中断（INT）。一般来说，快速中断的优先级高于普通中断，INTC 最多支持 8bit 优先级，即 0-255，数值越小优先级越高，具体实现中可以选择少于 8bit 的优先级编码方案。各中断优先级可由用户自行设置。

表 9.2: 中断类型及其优先级区间

中断类型	中断优先级
快速中断	0 ~ 255
普通中断	0 ~ 255

图 9.1 为中断控制器框图：

INTC 可分为两部分，第一部分为所有核共用的根节点（General），第二部分为每个核私有的叶子节点（lieutenant）。两部分中所有寄存器均通过内存映射，软件可通过内存读写操作来访问 INTC 寄存器进而对

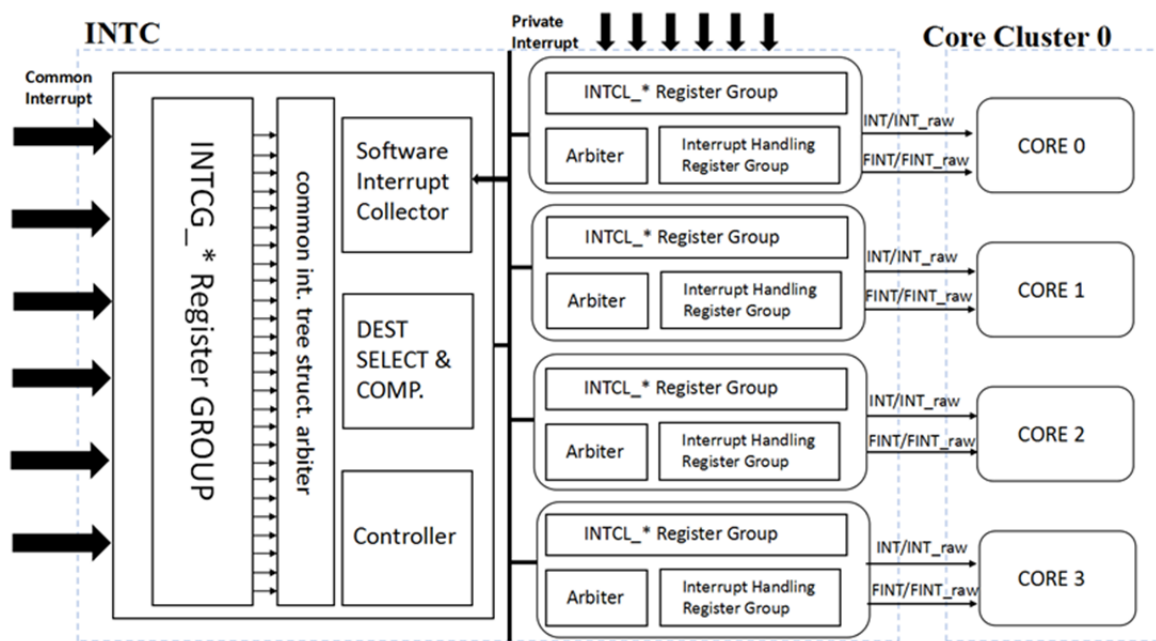


图 9.1: 中断控制器框图

中断进行管理和操作。

根节点的作用为：

- 接收公有中断；
- 对公有中断的使能位进行控制；
- 维护公有中断的状态；
- 设置公有中断的触发方式；
- 产生通过写内存方式触发的公有中断；
- 每一个公有中断的发送路线信息；
- 公有中断的优先级设置；
- 设置公有中断的中断类型（为快速中断或者普通中断）；
- 通过优先级比较将高优先级的中断按序派发给每一个核。

叶子节点的作用为：

- 接收私有中断和软件中断
- 产生软件中断及其发送路线信息并直接转发至目标各个核；
- 接收外部私有中断，来自根节点的公有中断以及软件中断；
- 对私有中断和软件中断的使能位进行控制；
- 维护私有中断和软件中断的中断状态；

- 设置私有中断的触发方式（软件中断触发方式始终为上升沿触发）；
- 私有中断和软件中断的优先级设置；
- 设置私有中断和软件中断的中断类型（为快速中断或者普通中断）；
- 通过优先级比较选出可响应的中断并让对应的 CORE 接收中断。

该中断控制器支持的功能有：

- 支持中断嵌套功能；
- 软件为每个中断配备 8 位的中断优先级编码（实际只使用高 5 位），数值越小优先级越高，在优先级比较时以高 5 位全部进行比较；
- 中断在处理的过程中支持优先级的动态调整，通过设置优先级阈值寄存器以较小的硬件代价实现中断优先级的反转，但对于公有中断，则无法调整已经发送给目标核的中断优先级，只能在下次触发该中断时使用新的优先级号；
- 支持电平触发和边缘触发的中断。

9.2 相关寄存器

图 9.2 为中断控制器所拥有的寄存器及其在架构上的位置。

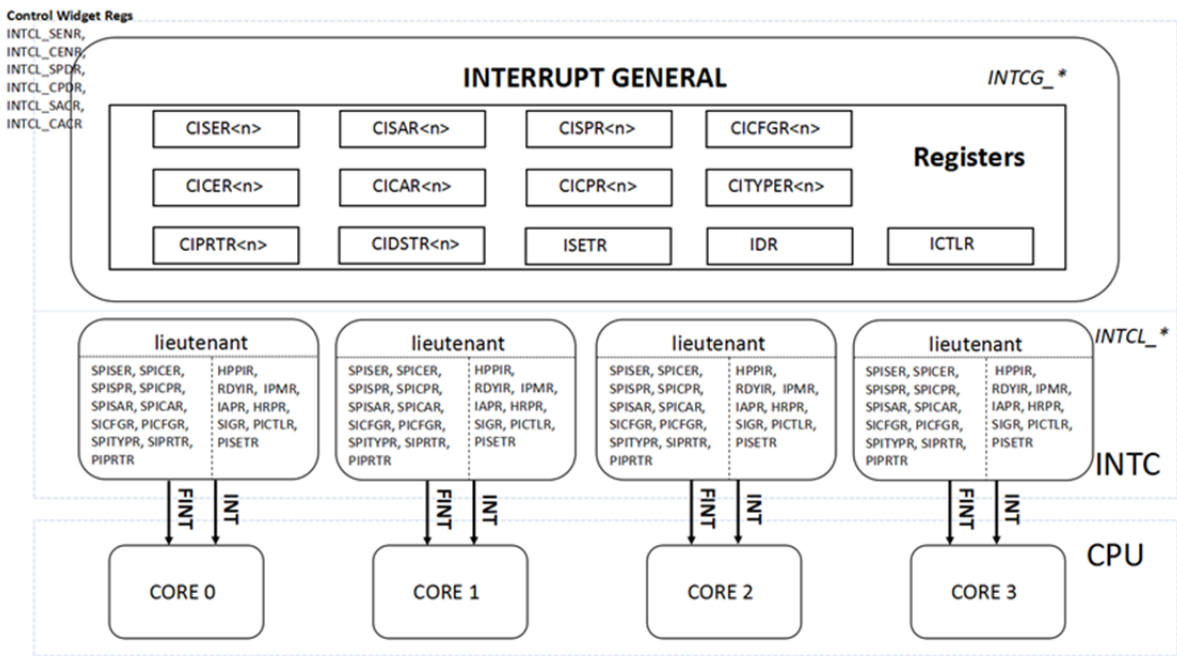


图 9.2: 中断控制器寄存器图

9.2.1 寄存器地址映射

中断控制器占据 64KB 内存空间, 基地址 (物理地址高 16 位) 通过输入信号 (pad_cpu_intc_base[15:0]) 在 SOC 层面硬件可配置, 软件可通过查询系统寄存器 INTC_BAR 获得。

根节点占据低 32KB 内存空间, 根节点上寄存器以 INTCG_* 作为开头。根节点寄存器映射如 表 9.3 所示, 偏移量为物理地址低 16 位, 拼接上 SOC 输入的高 16 位基地址构成完整 32 位物理地址。寄存器的具体描述请参考《C860MP 中断控制器用户手册》。

表 9.3: 根节点寄存器的地址映射

地址偏移量	名称	类型	初始值	描述
0x0000	ICTLR	读/写	0x00000003	中断控制寄存器
0x0004	-	-	-	-
0x0008	ISETR	只读	系统定义	中断控制器配置寄存器
0x000C	-	-	-	-
0x0010	IDR	只读	系统定义	中断控制器版本寄存器
0x0014-0x007C	-	-	-	-
0x0080-0x008C	CITYPR	读/写	0x00000000	公有中断类型配置寄存器
0x0100-0x011C	CICFGR	读/写	0x00000000	公有中断触发类型配置寄存器
0x0200-0x027C	CIPRTR	读/写	0x00000000	公有中断优先级配置寄存器
0x0600-0x060C	CISER	读/写	0x00000000	公有中断使能设置寄存器
0x0680-0x068C	CICER	读/写	0x00000000	公有中断使能清除寄存器
0x0700-0x070C	CISPR	读/写	0x00000000	公有中断等待设置寄存器
0x0780-0x078C	CICPR	读/写	0x00000000	公有中断等待清除寄存器
0x0800-0x080C	CISAR	读/写	0x00000000	公有中断响应设置寄存器
0x0880-0x088C	CICAR	读/写	0x00000000	公有中断响应清除寄存器
0x0900-0x09FC	为 tee 预留	-	-	-
0x1000-0x11FC	CIDSTR	读/写	0x00000000	公有中断目的地寄存器
0x1200-0x7FFC	Reserved	-	-	-

叶子节点占据高 32KB 内存空间, 每个叶子节点占据 4KB, 因此 INTC 最多支持 8 个叶子节点模块, 即支持 8 个 core, 叶子节点上的寄存器由 INTCL_* 作为开头。

叶子节点寄存器映射如 表 9.4 所示, 偏移量为物理地址低 12 位, 物理地址高 16 位由 SOC 提供, 而物理地址的 [15:12] 值由对应的核心编号确定, 例如 4' b1000 对应 core0, 4' b1001 对应 core1。寄存器的具体描述请参考《C860MP 中断控制器用户手册》。

表 9.4: 叶子节点寄存器的地址映射

地址	名称	类型	初始值	描述
0x000	PICTLR	读/写	0x00000003	叶子节点中断控制寄存器
0x004	-	-	-	-

下页继续

表 9.4 – 续上页

地址	名称	类型	初始值	描述
0x008	PISETR	只读	系统定义	叶子节点设置寄存器
0x00C	-	-	-	-
0x010	SPITYPR	读/写	0x00000000	软件及私有中断类型配置寄存器
0x014	SICFGR	读/写	0xAAAAAAAA	软件中断触发类型配置寄存器
0x018	PICFGR	读/写	0x00000000	私有中断触发类型配置寄存器
0x01C	-	-	-	-
0x020-0x02C	SIPRTR	读/写	0x00000000	软件中断优先级配置寄存器
0x030-0x03C	PIPRTR	读/写	0x00000000	私有中断优先级配置寄存器
0x040	SPISER	读/写	0x00000000	软件及私有中断使能设置寄存器
0x044	SPICER	读/写	0x00000000	软件及私有中断使能清除寄存器
0x048	SPISPR	读/写	0x00000000	软件及私有中断等待设置寄存器
0x04C	SPICPR	读/写	0x00000000	软件及私有中断等待清除寄存器
0x050	SPISAR	读/写	0x00000000	软件及私有中断响应设置寄存器
0x054	SPICAR	读/写	0x00000000	软件及私有中断响应清除寄存器
0x058	为 tee 预留	-	-	-
0x05C	为 tee 预留	-	-	-
0x060	SIGR	只写	-	软件中断生成寄存器
0x064	IPMR	只读	0x00000000	中断优先级掩膜寄存器
0x068	HPPIR	读/写	0x00000000	最高等待优先级中断寄存器
0x06C	RDYIR	只读	0x00000000	中断准备响应寄存器
0x070-0x08C	IAPR	只读	0x00000000	正在运行中断优先级寄存器
0x090	HRPR	只读	0x00000000	最高正在运行中断优先级寄存器
0x094-0x09C	-	-	-	-
0x0A0	SENR	只写	-	中断使能设置寄存器
0x0A4	CENR	只写	-	中断使能清除寄存器
0x0A8	SPDR	只写	-	中断等待设置寄存器
0x0AC	CPDR	只写	-	中断等待清除寄存器
0x0B0	SACR	只写	-	中断响应设置寄存器
0x0B4	CACR	只写	-	中断响应清除寄存器
0x0B8	为 tee 预留	-	-	-
0x0BC	为 tee 预留	-	-	-
0x0C0-0xFFC	Reserved	-	-	-

9.2.2 INTC 基地址寄存器 (INTC_BAR, CR<31,14>)

INTC_BAR 提供软件查询中断控制器各个寄存器的基地址，为一个 32bit 的只读寄存器，该寄存器的值由系统告知，INTC 基地址寄存器高 16 位有效，低 16 位恒为 0。

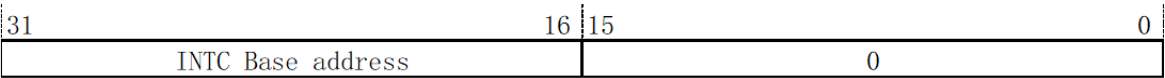


图 9.3: INTC 基地址寄存器

第十章 系统计时器

10.1 概述

计时器单元是 C860MP 的一个可选配模块。它主要用于计时并在计时结束时发送中断请求给中断控制器。系统计时器单元包含两大部分：一个核外输入的 56 位的系统计数器以及多个私有计时器。私有计时器为每一个 C860 核心配置一个，该计时器通过设定响应时间并用该时间与系统计数器提供的时间进行比较，从而产生中断请求。具体如 图 10.1 所示。需要注意的是，系统计数器由系统实现，推荐在 always-on 电压域进行工作。图 10.1 蓝色虚线边框部分不属于核内。

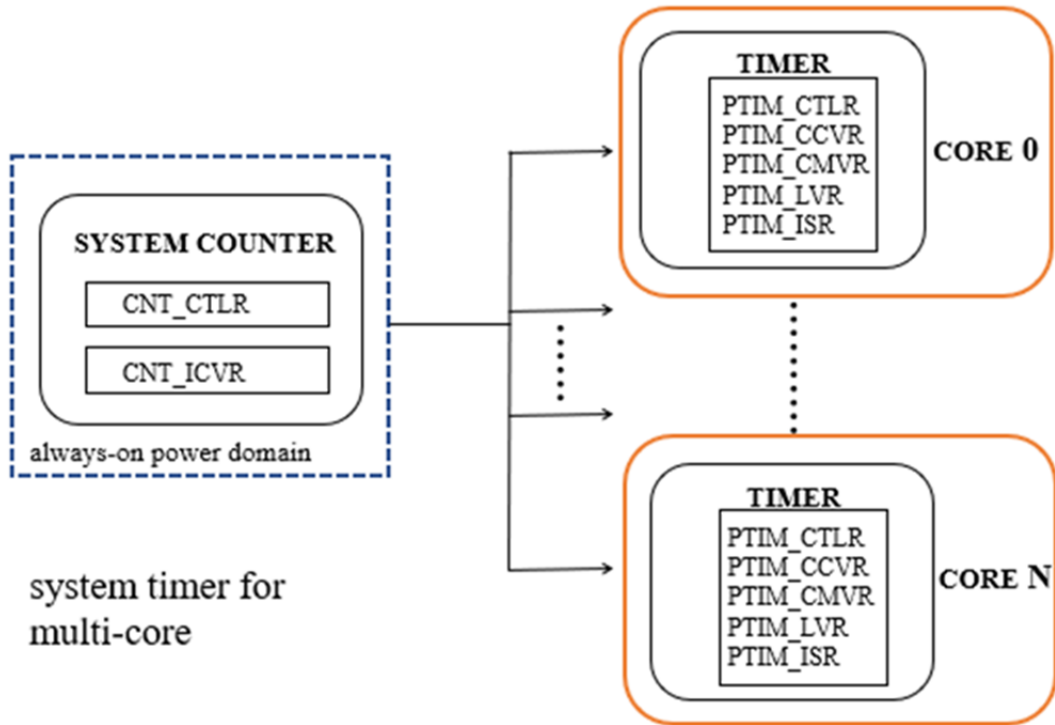


图 10.1: 多核系统计数器框图

10.2 私有计时器相关寄存器

而每一个核单独拥有的计时器寄存器为系统寄存器，位于 CP 第 14 组寄存器中，具体如 表 10.1 所示。

表 10.1: 私有计时器寄存器映射

系统寄存器位置	名称	类型	初始值	描述
CR<0,14>	PTIM_CTLR	读写	32' b0	计时器控制寄存器
CR<1,14>	PTIM_TSR	读写	32' b0	计时器状态寄存器
CR<2,14>	PTIM_CCVR_Hi	只读	32' b0	计时器当前计数值寄存器（高 32 位）
CR<3,14>	PTIM_CCVR_Lo	只读	32' b0	计时器当前计数值寄存器（低 32 位）
CR<4,14>	PTIM_CMVR_Hi	读写	32' b0	计时器比较值寄存器（高 32 位）
CR<5,14>	PTIM_CMVR_Lo	读写	32' b0	计时器比较值寄存器（低 32 位）
CR<6,14>	PTIM_LVR	只写	-	计时器装载寄存器

10.2.1 (PTIM_CTLR,64-bit)

计时器控制寄存器为每个核所私有的寄存器，它用来控制是否使能计时器以及是否触发对应的计时中断。其寄存器位分布和位定义如 图 10.2 和 表 10.2 所示。

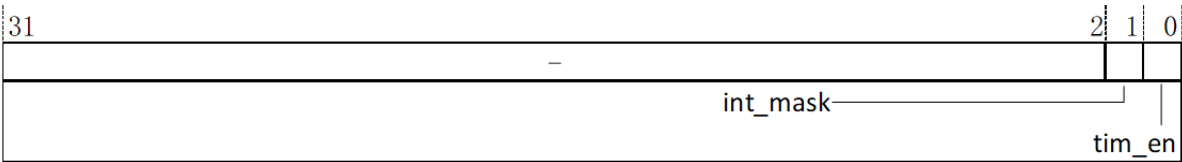


图 10.2: 计时器控制寄存器

表 10.2: 计时器控制寄存器域定义

位	名称	描述
1	int_mask	中断屏蔽。 读操作 0 计时器可以触发中断。 1 计时器不可触发中断 写操作 0 设置计时器可以触发中断。 1 设置计时器不可触发中断。 初始化为：1' b1
0	tim_en	计时器使能。该计时器必须在使能位置“1”时才可进行使用。 读操作 0 计时器未使能。 1 计时器已使能。 写操作 0 设置计时器不使能。 1 设置计时器使能。 初始化为：1' b0

10.2.2 计时器当前计数值寄存器 (PTIM_CCVR, 64-bit)

 计时器当前计数值寄存器存储当前计数值，每个 CORE 都各自有一个该寄存器。其寄存器位分布和位定义如 图 10.3 和 表 10.3 所示。

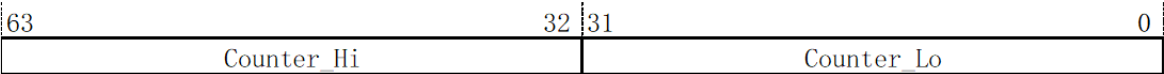


图 10.3: 计时器当前计数值寄存器

表 10.3: 计时器当前计数值寄存器域定义

位	名称	描述
[63:32]	Counter_Hi	存储从计数器获取的高 32 位计数器值。 读操作 读取从计数器获取的高 32 位计数器值。 写操作 无效。
[31:0]	Counter_Lo	存储从计数器获取的低 32 位计数器值。 读操作 读取从计数器获取的低 32 位计数器值。 写操作 无效。

10.2.3 计时器比较值寄存器 (PTIM_CMVR, 64-bit)

 计数器比较值寄存器用来写入用户想要与计数器值进行比较的计数值，通过比较可以触发中断。其寄存器位分布和位定义如 图 10.4 和 表 10.4 所示。

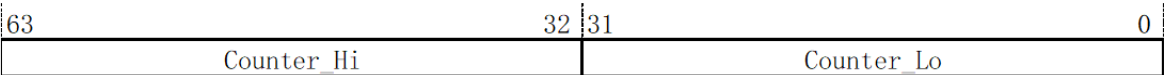


图 10.4: 计数器比较值寄存器

表 10.4: 计数器比较值寄存器域定义

位	名称	描述
[63:32]	Counter_Hi	软件写入用来与计数器值 (PTIM_CCV) 进行比较的高 32 位计数值。 读操作 读取当前 CMVR_Hi 的值 写操作 写入希望进行比较的计数值
[31:0]	Counter_Lo	软件写入用来与计数器值 (PTIM_CCV) 进行比较的低 32 位计数值。 读操作 读取当前 CMVR_Lo 的值 写操作 写入希望进行比较的计数值

10.2.4 计时器装载寄存器 (PTIM_LVR, 32-bit)

 计时器装载寄存器用来给用户写入一个相对计数值来触发中断。其寄存器位分布和位定义如 图 10.5 和 表 10.5 所示。

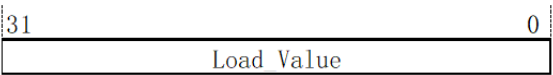


图 10.5: 计时器装载寄存器

表 10.5: 计时器装载寄存器域定义

位	名称	描述
[31:0]	Load_Value	计时器装载值为一个相对于当前计数器的一个偏移量值。通过对该寄存器的写入操作将 PTIM_LV+PTIM_CCV => PTIM_CMV。 读操作 无效。 写操作 写入当前的计时器装载值。

10.2.5 计时器状态寄存器 (PTIM_TSR, 32-bit)

计时器中断设置寄存器用来设置或查看对应 CORE 的计时中断是否触发。其寄存器位分布和位定义如图 10.6 和 表 10.6 所示。

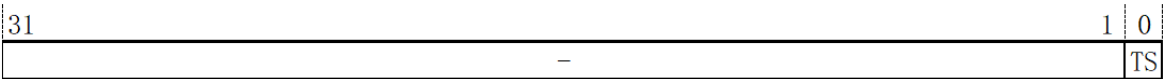


图 10.6: 计时器中断设置寄存器

表 10.6: 计时器中断设置寄存器域定义

位	名称	描述
0	TS	计时器有效位。 不管计时器中断屏蔽位是否有效，当计数器等于计时器比较值时置高该 TS 位。当计时器中断屏蔽位为低时，TS 为高触发一个高电平中断请求。 读操作 0 未达到计数条件。 1 达到计数条件。 写操作 1 没有效果。 0 清除 TS bit。 初始化为：1'b0

10.3 系统计数器实现参考

系统计数器并没有实现在处理器内部，本章仅作为参考设计说明。系统计时器包含一个控制寄存器和一个初始值寄存器。采用内存地址映射的方式，位于外设的地址空间，支持按字的访问方式。系统计数器的各个寄存器地址映射如 表 10.7 所示。

表 10.7: 系统计数器的地址映射

地址	名称	类型	初始值	描述
0x0	CNT_CTLR	读/写	0x00000000	计数器控制寄存器
0x4	CNT_ICVR_Hi	读/写	0x00000000	计数器初始值寄存器 (高 32 位)
0x8	CNT_ICVR_Lo	读/写	0x00000000	计数器初始值寄存器 (低 32 位)

10.3.1 计数器控制寄存器 (CNT_CTLR, 32-bit)

计数器控制寄存器用于控制计数器的分频操作以及使能位。其中分频操作主要与低功耗有关，开启后精度不受影响。其寄存器位分布和位定义如 图 10.7 表 10.8 所示。

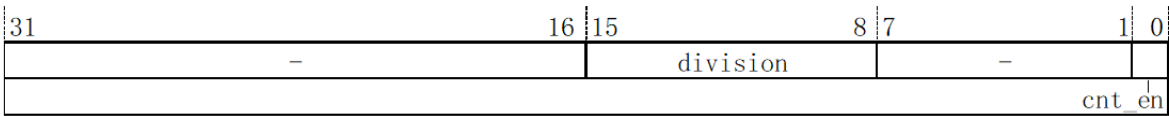


图 10.7: 计数器控制寄存器

表 10.8: 计数器控制寄存器域定义

位	名称	描述
[15:8]	division	计数器分频值。默认值为全 0，即 apbclk 每过一个 cycle，计数器就加“1”。当对计数器进行分频后，则可以实现在经过 K 个 cycle 后，计数器的值加上 K。 读操作 读取当前计数器的分频 写操作 设置当前计数器新的分频 该计数器的分频值为 (division+1)，即当 division 为 1 时为二分频，division 为 2 时为 3 分频。 初始化为 0
0	cnt_en	计数器的使能位。只有当该使能位开启时，该计数器才可正常进行工作。 读操作 0 计数器未使能。 1 计数器已使能。 写操作 0 设置计数器不使能。 1 设置计数器使能。 初始化为 1' b0

10.3.2 计数器初始值寄存器 (CNT_ICVR, 64-bit)

计数器初始值寄存器存储计数器的初始值。其寄存器位分布和位定义如 图 10.8 表 10.9 所示。

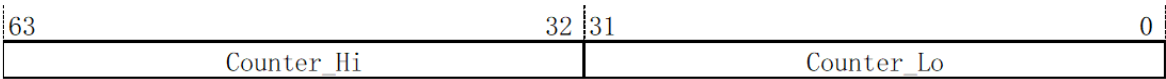


图 10.8: 初始计数器值寄存器

表 10.9: 初始计数器值寄存器域定义

位	名称	描述
[63:32]	Counter_Hi	存储计数器的初始值的高 32 位。 读操作 读取存储计数器的初始值。 写操作 修改存储计数器的初始值。 初始化为 0
[31:0]	Counter_Lo	存储计数器的初始值的低 32 位。 读操作 读取存储计数器的初始值。 写操作 修改存储计数器的初始值。 初始化为 0

第十一章 调试接口

11.1 概述

调试接口是软件与处理器交互的通道。用户可以通过调试接口获取 CPU 的寄存器以及存储器内容等信息，包括其他的片上设备信息。此外，程序下载等操作也可以通过调试接口完成的。

C860MP 支持兼容 IEEE-1149.1 标准的 JTAG 通信协议（通称 JTAG5），可以同已有的 JTAG 部件或独立的 JTAG 控制器集成在一起。另外，还实现了使用 2 根信号线进行通信的 JTAG_2 调试接口以节省引脚。

调试接口的主要特性如下：

- 使用标准的 JTAG 接口进行调试，同时支持两线制的 JTAG_2 调试接口；
- 支持同步调试和异步调试，保证在极端恶劣情况下使处理器进入调试模式；
- 支持软断点；
- 可以设置多个内存断点；
- 检查和设置 CPU 寄存器的值；
- 检查和改变内存值；
- 可进行指令单步执行或多步执行；
- 快速下载程序；
- 可在 CPU 复位之后进入调试模式。

玄铁 CPU 的调试工作是调试软件，调试代理服务程序，调试器和调试接口一起配合完成的，调试接口在整个 CPU 调试环境中的位置如 图 11.1 所示。其中，调试软件和调试代理服务程序通过网络互联，调试代理服务程序与调试器通过 USB 连接，调试器与 CPU 的调试接口以 JTAG 模式通信。

11.2 多核调试框架

C860MP 采用多核单端口的调试框架，通过一个共享的 JTAG 接口访问各个 CORE 的辅助调试单元 (HAD)，触发 CORE 进出调试模式和访问处理器资源。通过在 JTAG 接口中设置 HACR 的 CORESEL 域的方式指定目标 CORE，然后再配置目标 CORE 的 HAD 寄存器。

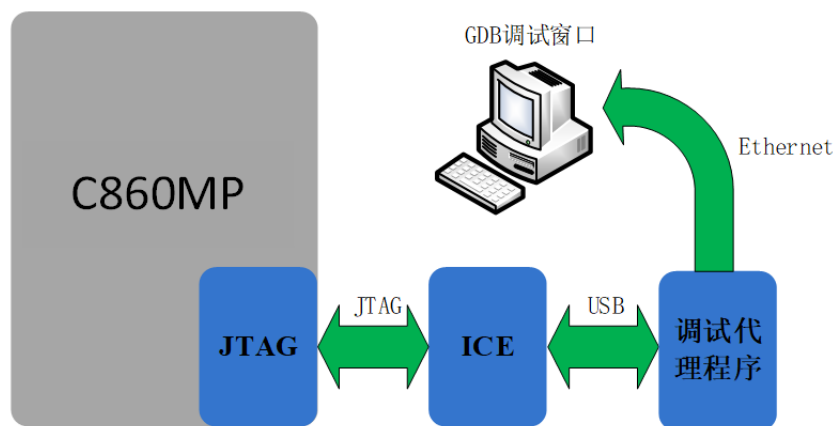


图 11.1: 调试接口在整个 CPU 调试环境中的位置

当某一个 CORE 进入调试时，需要将其他某个或者其他多个核都拉入调试；而当某个 CORE 退出调试时，也需要将其他某个或者其他多个核都拉出调试。因此，C860MP 设置了一个集中的事件传输模块 (ETM) 用于支持多核之间调试事件（进入调试和退出调试）的传输。当 C860 核心在接收到 ICE 发送的调试命令产生调试进入或者调试退出事件时，同时将该事件发送到 ETM，由 ETM 将该事件转给其他的 CORE，实现多个核同步进入和退出调试的目的。如图 11.2 所示（以两个核为例）。

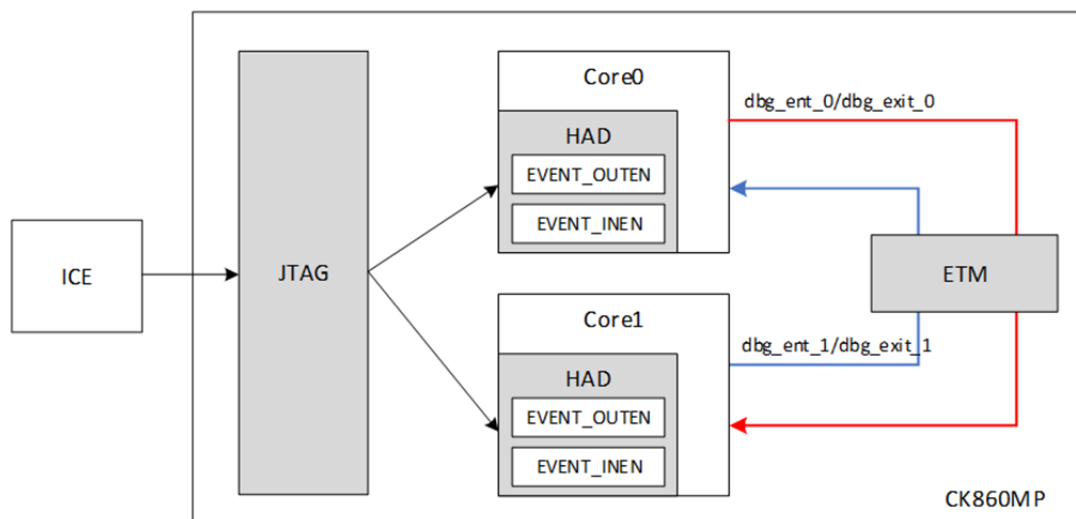


图 11.2: 多核调试整体框架

- 多核系统进入调试的场景：

某个 CORE 进入调试时会产生一个向外输出的 DBG_ENT 信号，该 CORE 的 EVENT_OUTEN 寄存器控制该 DBG 信号是否能够传输到 ETM 模块，当 EVENT_OUTEN 寄存器使能时，该 DBG 信号经 ETM 模块传输给其他 CORE，此时其他 CORE 是否被拉入调试状态取决于各自的 EVENT_INEN 寄存器。

- 多核系统退出调试的场景：

某个 CORE 退出调试时会产生一个向外输出的 DBG_EXIT 信号，该 CORE 的 EVENT_OUTEN 寄存器控制该 DBG_EXIT 信号是否能够传输到 ETM 模块，当 EVENT_OUTEN 寄存器使能时，该 DBG_EXIT

信号经 ETM 模块传输给其他 CORE，此时其他 CORE 是否被拉出调试状态取决于各自的 EVENT_CTRL 寄存器。

11.3 外部接口

调试模块与外部的接口主要是与 JTAG 相关的接口信号和调试相关的接口信号。表 11.1 列出了与调试相关的接口信号。

表 11.1: 调试模块与外部的接口信号

信号名	方向
(corex_pad_dbg_b)	输出
pad_corex_dbgrq_b	输入
pad_had_jdb_req_b	输入
corex_pad_jdb_ack_b	输出
pad_had_jtg_tap_en	输入
had_pad_jtg_tap_on	输出
corex_pad_jdb_pm[1:0]	输出
pad_had_jtg_tclk	输入
pad_had_jtg_trst_b	输入
pad_had_jtg_tdi	输入
had_pad_jtg_tdo	输出
had_pad_jtg_tdo_en	输出
pad_had_jtg_tms_i	输入
had_pad_jtg_tms_o	输出
had_pad_htg_tms_oe	输出
pad_had_jtag2_sel	输入

1. corex_pad_dbg_b

低电平表示对应 CORE 处于调试模式中。

2. pad_corex_dbgrq_b

同步进入调试模式请求信号。低电平有效。该信号是 CORE 的外部输入信号，在 CORE 中被系统时钟同步之后输入到 HAD 中，HAD 使用该信号欲让 CORE 同步进入调试模式。拉低该信号与设置 HAD 寄存器 HCR 的 DR 位有相同的作用。

3. pad_had_jdb_req_b 与 corex_pad_jdb_ack_b

pad_had_jdb_req_b 信号是让 CORE 异步进入调试状态的请求信号，该信号至少要维持低电平两个 JTAG 时钟周期才能保证 CORE 进入调试状态并且能够调试程序。如果该信号维持低电平不足两个 JTAG 时钟周期，那么可能会出现 CORE 已进入调试状态但不能调试程序的情况，因为该信号还会用于使能调试接口中的 TAP 状态机。

在 CORE 进入调试状态之后，corex_pad_jdb_ack_b 信号会维持两个 JTAG 时钟周期的低电平以作为应答。

4. pad_had_jtg_tap_en 与 had_pad_jtg_tap_on

pad_had_jtg_tap_en 信号为调试接口中 TAP 状态机的使能信号，维持该信号为高电平至少一个 JTAG 时钟周期可以使能调试接口中的 TAP 状态机。如果该信号在 CPU 上电之后一直无效，那么使用同步方式（如设置调试接口寄存器 HCR 中的 DR 位，断点等）使 CPU 进入调试状态时可能无法调试程序。

在 TAP 状态机启动之后，had_pad_jtg_tap_on 信号将拉高。

5. had_pad_jdb_pm[1:0]

had_pad_jdb_pm[1:0] 信号指示 CPU 当前工作模式，可以通过该信号确定 CPU 是否已进入调试模式。具体如 表 11.2 所示。

表 11.2: PM 指示的当前 CPU 状态

had_pad_jdb_pm[1:0]	说明
00	普通模式
01	低功耗模式（STOP，DOZE，WAIT）
10	调试模式
11	保留

6. pad_had_jtg_tclk

JTAG 时钟信号。该信号为外部输入信号，一般为调试器产生的时钟信号，要保证该时钟信号的频率低于 CPU 时钟信号的频率 1/2 才能保证调试模块与 CORE 之间的正常工作。

7. pad_had_jtg_trst_b

pad_had_jtg_trst_b 信号为 JTAG 复位信号，可以复位 TAP 状态机以及其他相关控制信号。

8. JTAG5 相关信号

pad_had_jtg_tdi 信号为 HAD 端 JTAG 串行输入信号，HAD 端在 JTAG 时钟信号 tclk 的上升沿对其采样，而外部调试器在 JTAG 时钟的下降沿设置该信号；

had_pad_jtg_tdo 信号为 HAD 端 JTAG 串行输出信号，HAD 端在 JTAG 时钟信号 tclk 的下降沿对其设置，而外部调试器在 JTAG 时钟的上升沿对其采样；

had_pad_jtg_tdo_en 信号表示 had_pad_jtg_tdo 信号有效，通常外部调试器监视该信号以确定 had_pad_jtg_tdo 信号是否有效，调试器也可以不观察该信号而通过调试器内部的 TAP 状态机的状态以决定对 had_pad_jtg_tdo 信号的采样。该信号主要用作在实现多个 TAP 状态机时，确定是哪个 JTAG 的输出；

pad_had_jtg_tms 信号为 JTAG 模式选择信号，该信号由调试器发出，用于控制 HAD 中 TAP 状态机的运作。pad_had_jtg_tms 信号的功能被 JTAG_2 的 3 个信号替代

9. JTAG_2 相关信号

pad_had_jtg_tms_i 信号为 2 线制 JTAG 串行数据输入信号，调试接口在 JTAG 时钟信号 TCLK 的上升沿对其采样，而外部调试器在 JTAG 时钟的下降沿设置该信号；

该信号在空闲时必须保持为高电平，同时空闲时时钟信号最好停止。用户可以利用该信号同步复位 HAD 逻辑：在实现 2 线制 JTAG 接口的调试模块中，如果时钟信号一直有效，用户只需保持该信号为高电平状态并维持 80 个时钟周期即可同步复位调试模块。复位调试模块后，调试模块的 TAP 状态机回到 RESET 态，同时调试模块寄存器 HACR 复位到 0x82（指向 ID 寄存器）。

had_pad_jtg_tms_o 信号为 2 线制 JTAG 串行数据输出信号，调试接口在 JTAG 时钟信号 TCLK 的下降沿对其设置，而外部调试器在 JTAG 时钟的上升沿对其采样；

had_pad_jtg_tms_oe 信号为 had_pad_jtg_tms_o 信号有效指示信号。CPU 外部应利用该信号将 pad_had_jtg_tms_i 和 had_pad_jtg_tms_o 信号合为一个双向端口信号。

10. pad_had_jtag2_sel

JTAG_2 接口选择信号。置高表示选择 JTAG_2 调试接口，置低表示选择 JTAG_5 调试接口。

第十二章 工作模式转换

每个 C860 处理器核总共有三类工作模式：正常运行模式、低功耗工作模式和调试模式，其中低功耗工作模式分为三种：STOP 模式、DOZE 模式、WAIT 模式，这三种低功耗模式的不同之处在于工作时停止的时钟不同。本章将详细解析 CPU 的工作模式以及模式之间的转换。

12.1 C860 工作模式及其转换

如 图 12.1 所示，C860 的工作模式有三类，即正常工作模式、低功耗工作模式和调试模式，CPU 处于哪种工作模式可以通过查询 `corex_pad_jdb_pm[1:0]` 信号得到。

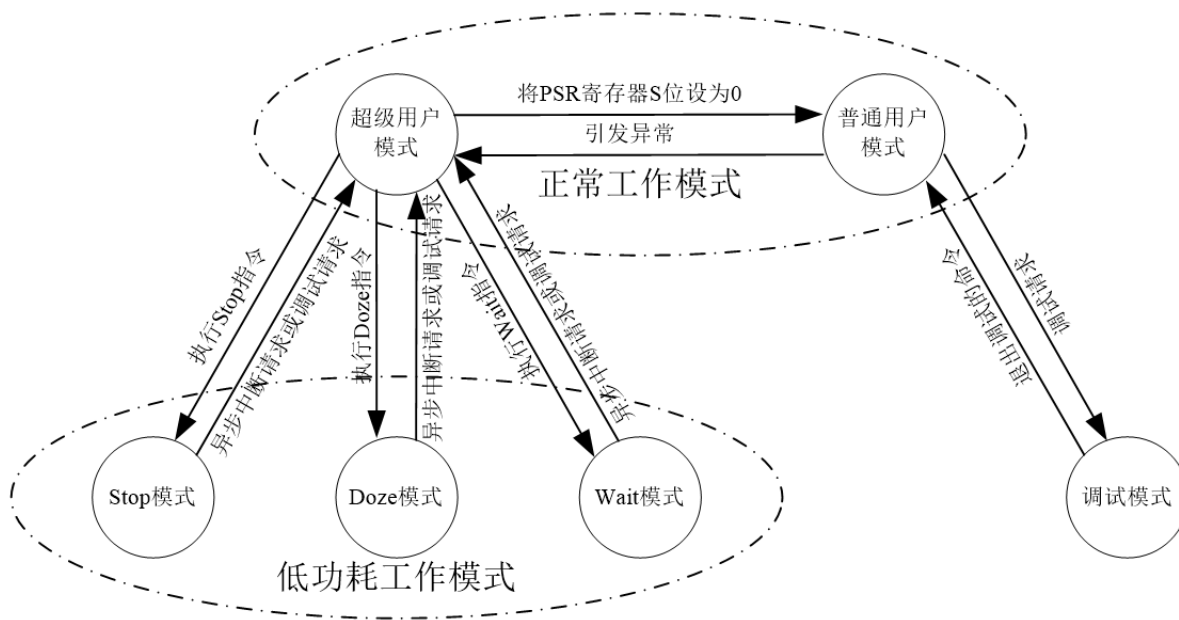


图 12.1: CPU 的各种工作状态示意图

12.2 正常工作模式

CPU 的正常工作模式可以分为两种，超级用户模式和普通用户模式，CPU 处于哪种正常工作模式通过查询 PSR 寄存器中的 S 位得到，当 S 位为 1 时，CPU 工作于超级用户模式；当 S 位为 0 时，CPU 工作

于普通用户模式。当 CPU 工作在超级用户模式时，可以通过将 S 位设置为 0 进入普通用户模式；当 CPU 工作于普通用户模式时，通过引发异常进入超级用户模式。

12.2.1 低功耗模式

当 CPU 执行完低功耗指令 (STOP、DOZE、WAIT) 之后，CPU 将进入相应的低功耗模式。CPU 进入低功耗模式之后，CPU 的时钟被停止，只有异步中断请求 (pad_biu_intraw_b 和 pad_biu_fintraw_b 信号) 或者调试请求才能正常地退出低功耗模式。当前的 CPU 工作于哪种低功耗模式可以通过查询信号线 biu_pad_lpmdb[1:0] 来得到。

12.2.1.1 DOZE 工作模式

当 CPU 执行完 DOZE 指令后，进入 DOZE 低功耗模式，此时 CPU 的时钟停止，哪些外设的时钟停掉与实现相关。

12.2.1.2 STOP 工作模式

当 CPU 执行完 STOP 指令后，进入 STOP 低功耗模式，此时 CPU 和大多数外设的时钟都被停止。

12.2.1.3 WAIT 工作模式

当 CPU 执行完 WAIT 指令之后，进入 WAIT 低功耗模式，此时 CPU 停止工作，大多数的外设仍然在运行并可以产生中断。

12.2.2 调试模式

12.2.2.1 进入调试模式

当 CPU 接到调试请求之后，进入调试模式，其中的调试请求源可以有以下 7 种：

- 在低功耗模式下，可以通过使能 pad_had_jdb_req_b 信号进入调试模式。
- 在正常工作模式下，可以通过使能 pad_corex_dbgrq_b 信号进入调试模式。
- 当 C860 CSR 的 FDB 位有效时，处理器在执行 bkpt 指令进入调试模式。
- 当 C860 HCR 的 DR 位有效时，处理器在完成当前指令后进入调试模式。
- 当 C860 HCR 的 TME 位有效时，处理器在跟踪计数器的值减到 0 后进入调试模式。
- 在 C860 存储器断点调试模式下，当 MBCA 的值为 0 时，如果当前执行的指令符合断点要求，处理器进入调试模式。

12.2.2.2 退出调试模式

如果 C860 HACR 的 GO、EX 位被置为 1 时，并且当前的操作是对 CPUSCR 或者是对“没有选中任何寄存器”进行读写，则执行指令时 CPU 退出调试模式，进入正常工作模式。

注意：由于在调试模式下 PC，CSR，PSR 是可变的，因此在退出调试模式时，CPUSCR 中的值必须是刚进入调试模式之时保存过的值。

第十三章 程序参考示例

13.1 MMU 设置示例

```
/******  
\* Function: An example of set C860MP MMU.  
\* Enable/disable C860MP memory space region.  
\* Memory space: Virtual address <-> physical address.  
\*  
\* virtual address: 0K-4K <-> physical address: 0K-4K  
\* virtual address: 4K-8K <-> physical address: 4K-8K  
\* virtual address: 8K-12K <-> physical address: 16K-20K  
\* virtual address: 12K-16K <-> physical address: 8K-12K  
\*  
\* Copyright © 2023 Hangzhou C-SKY MicroSystems Co., Ltd. All rights reserved.  
/******  
/*invalid all MMU TLB Entry*/  
  
movi r1,0x0  
bseti r1,26  
mtcr r1,cr<8,15>  
  
/* virtual address: 0K-4K <-> physical address: 0K-4K*/  
/* virtual address: 4K-8K <-> physical address: 4K-8K*/  
  
movi r1,0x1e  
mtcr r1,cr<2,15> //me10,PPN0:0x0  
lrw r1,0x101e  
mtcr r1,cr<3,15> //me11,PPN1:0x1  
movi r1,0  
mtcr r1,cr<4,15> //meh,VPN:0x0 & 0x1  
mtcr r1,cr<6,15> //page size:4K
```

(下页继续)

(续上页)

```
bseti r1,28
mtcr r1,cr<8,15> //write to jTLB

/* virtual address: 8K-12K <-> physical address: 16K-20K*/
/* virtual address: 12K-16K <-> physical address: 8K-12K*/

lrw r1,0x401e
mtcr r1,cr<2,15> //me10,PPN0:0x4
lrw r1,0x201e
mtcr r1,cr<3,15> //me11,PPN1:0x2
lrw r1,0x2000
mtcr r1,cr<4,15> //meh,VPN:0x2 & 0x3
movi r1,0
mtcr r1,cr<6,15> //page size:4K
bseti r1,28
mtcr r1,cr<8,15> //write to jTLB

/* enable mmu*/

mfcr r1,cr<18,0>
bseti r1,0x0
mtcr r1,cr<18,0> //enable mmu
```

13.2 高速缓存设置示例

高速缓存的开启示例

```
/*invalid I-cache*/

mfcr r1,cr17
bseti r1,0
bclri r1,1
bclri r1,2
bclri r1,3
bseti r1,4
mtcr r1,cr17

// it can also use icache instrucion to replace the invalid sequence.
// icache.iall
```

(下页继续)

(续上页)

```
// sync.is
/*enable I-cache*/

mfcr r1,cr18
bseti r1,2
mtcr r1,cr18

/*invalid D-cache*/

mfcr r1,cr17
bclri r1,0
bseti r1,1
bclri r1,2
bclri r1,3
bseti r1,4
mtcr r1,cr17

// it can also use dcache instruciton to replace the invalid sequence.
// dcache.iall
// sync.is
/*enable D-cache*/

mfcr r1,cr18
bseti r1,3
mtcr r1,cr18

/*invalid L2-cache*/

l2cache.iall
sync.is

/*enable L2-cache*/

mfcr r1,cr23
bseti r1,3
mtcr r1,cr23
```

指令高速缓存与数据高速缓存的同步示例

CPU0

```

st.w r1,(r0,0x0) // a new instruction defined in r1

// is stored to program memory address defined in r0.

dcache.cval1 r0 // clean the new instrcutioin to the shared L2 cache.
sync.s // ensure completion of clean operation.
icache.iva r0 // invalid icache according to shareable configuraiton.
sync.s // ensure completion in all CPUs.
st.w r2,(r3,0x0) // set flag to signal operation completion.
sync.is
jmp r0 // jmp to new code

```

CPU1~CPU4

WAIT_FINISH:

```

ld.w r4,(r3,0x0)
cmpne r4,r2
bt WAIT_FINISH // wait CPU0 modification finish.
sync.is
jmp r0 // jmp to new code

```

TLB 与数据高速缓存的同步示例

CPU0

```

st.w r1,(r0,0x0) // update a new translation table entry
sync.s // ensure completion of update operation.
tlbi.vaas r4 // two-level page table realized in

// memory address translation

sync.is // ensure completion of TLB invalidation and

// synchronises context

```

13.3 多核启动示例

CPU0

```
..... // CPU1~CPU3 are in reset mode and
// CPU0 executes system initialize operation.

lrw r1, RVBA
mtcr r1, cr28 // set reset vector base address
mfcr r1,cr29
bseti r1, 1 // release CPU1' s reset signal.
bseti r1, 2 // release CPU2' s reset signal.
bseti r1, 3 // release CPU3' s reset signal.
mtcr r1,cr29 // CPU1~CPU3 start to execute reset exception routine.
```

13.4 同步原语示例

CPU0

ACQUIRE_LOCK:

```
ldex.w r1,(r0,0x0) // read lock
cmpnei r1,0 // checks if 0
bt ACQUIRE_LOCK // if lock fail, try again
stex.w r2,(r0,0x0) // attempt to store new value
cmpnei r2,1 // checks if 1
bt ACQUIRE_LOCK // if lock fail, try again
sync.s
..... // critical region code
```

CPU1

```
movi r1,0
sync.s // ensure all operation are observed before clear lock.
st.w r1,(r0,0x0) // clear lock.
```

第十四章 附录 A 基础指令术语表

以下是每条 C860 实现的玄铁 CPU V2 整型和浮点标量指令的具体描述，下面根据指令英文字母顺序对每条指令进行详细说明。

每条指令助记符结尾以数字“32”或“16”表示指令位宽。例如，“addc32”表示该指令为 32 位无符号带进位加法指令，“addc16”表示该指令为 16 位无符号带进位加法指令。

如果省略助记符中的指令位宽（如“addc”），系统会自动将其汇编为最优化的指令。

其中，指令中文名称中带 # 的为伪指令。

14.1 ABS——绝对值指令

统一化指令	
语法	abs rz, rx
操作	$RZ \leftarrow RX $
编译结果	仅存在 32 位指令。 abs32 rz, rx
说明	取 RX 值的绝对值，并把结果存在 RZ。 注意，操作数 0x80000000 的结果为 80000000。
影响标志位	无影响
异常	无

32 位指令	
操作	$RZ \leftarrow RX $
语法	abs32 rz, rx
说明	取 RX 值的绝对值，并把结果存在 RZ。 注意，操作数 0x80000000 的结果为 80000000。
影响标志位	无影响
异常	无

32 位指令格式：

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

14.2 ADDC——无符号带进位加法指令

统一化指令		
语 法	addc rz, rx	addc rz, rx, ry
操 作	$RZ \leftarrow RZ + RX + C$, $C \leftarrow$ 进位	$RZ \leftarrow RX + RY + C$, $C \leftarrow$ 进位
编 译 结 果	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (x<16) and (z<16), then addc16 rz, rx; else addc32 rz, rz, rx;	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (y==z) and (x<16) and (z<16), then addc16 rz, rx; else addc32 rz, rx, ry;
说 明	将 RZ/RX、RX 与 C 位的值相加，并把结果存在 RZ，进位存在 C 位。	
影 响 标 志 位	$C \leftarrow$ 进位	
异 常	无	

16 位指令	
操作	$RZ \leftarrow RZ + RX + C$, $C \leftarrow$ 进位
语法	addc16 rz, rx
说明	将 RZ、RX 与 C 位的值相加，并把结果存在 RZ，进位存在 C 位。
影响标志位	$C \leftarrow$ 进位
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

15	14	10	9	6	5	2	1	0
0		11000		RZ		RX		01

32 位指令	
操作	$RZ \leftarrow RX + RY + C, C \leftarrow \text{进位}$
语法	addc32 rz, rx, ry
说明	将 RX、RY 与 C 位的值相加，并把结果存在 RZ，进位存在 C 位。
影响标志位	$C \leftarrow \text{进位}$
异常	无

32 位指令格式：

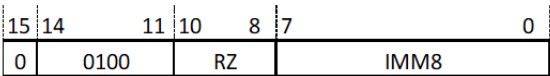
31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001		RY		RX		000000		00010			RZ

14.3 ADDI——无符号立即数加法指令

统一化指令			
语 法	addi rz, oimm12	addi rz, rx, oimm12	addi rz, r28, oimm18
操 作	$RZ \leftarrow RZ + \text{zero\_extend}(\text{OIMM12})$	$RZ \leftarrow RX + \text{zero\_extend}(\text{OIMM12})$	$RZ \leftarrow R28 + \text{zero\_extend}(\text{OIMM18})$
编 译 结 果	根据立即数和寄存器的范围编译为对应的 16 位或 32 位指令。 if (z<8) and (oimm12<257), addi16 rz, oimm8; else addi32 rz, rz, oimm12;	根据立即数和寄存器的范围编译为对应的 16 位或 32 位指令。 if (z<8) and (oimm12<257), addi16 rz, oimm8; else addi32 rz, rz, oimm12;	根据立即数和寄存器的范围编译为对应的 16 位或 32 位指令。 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;
说 明	将带偏置 1 的立即数零扩展至 32 位，然后与 RX/RZ 的值相加，把结果存入 RZ。		
影 响 标 志 位	无影响		
限 制	若源寄存器是 R28，立即数的范围为 0x1-0x40000。 若源寄存器不是 R28，立即数的范围为 0x1-0x1000。		

16 位指令 1	
操作	$RZ \leftarrow RZ + \text{zero\_extend}(\text{OIMM8})$
语法	addi16 rz, oimm8
说明	将带偏置 1 的 8 位立即数（OIMM8）零扩展至 32 位，然后与 RZ 的值相加，把结果存入 RZ。 注意：二进制操作数 IMM8 等于 OIMM8 - 1。
影响标志位	无影响
限制	寄存器的范围为 r0-r7；立即数的范围为 1-256。
异常	无

16 位指令格式 1：



IMM8 域：

指定不带偏置立即数的值。

注意：加到寄存器里的值 OIMM8 比起二进制操作数 IMM8 需偏置 1。

00000000：

加 1

00000001：

加 2

.....

11111111：

加 256

16 位指令 2	
操作	$RZ \leftarrow RX + \text{zero\_extend}(\text{OIMM3})$
语法	addi16 rz, rx, oimm3
说明	将带偏置 1 的 3 位立即数（OIMM3）零扩展至 32 位，然后与 RX 的值相加，把结果存入 RZ。 注意：二进制操作数 IMM3 等于 OIMM3 - 1。
影响标志位	无影响
限制	寄存器的范围为 r0-r7；立即数的范围为 1-8。
异常	无

16 位指令格式 2：

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

IMM3 域：

指定不带偏置立即数的值。

注意：加到寄存器里的值 OIMM3 比起二进制操作数 IMM3 需偏置 1。

000：

加 1

001：

加 2

.....

111：

加 8

32 位指令 1	
操作	$RZ \leftarrow RX + \text{zero\_extend}(OIMM12)$
语法	addi32 rz, rx, oimm12
说明	将带偏置 1 的 12 位立即数（OIMM12）零扩展至 32 位，然后与 RX 的值相加，把结果存入 RZ。 注意：二进制操作数 IMM12 等于 OIMM12 - 1。
影响标志位	无影响
限制	立即数的范围为 0x1-0x1000。
异常	无

32 位指令格式 1：

31	30	26	25	21	20	16	15	12	11	0
1	11001	RZ	RX	0000	IMM12					

IMM12 域：

指定不带偏置立即数的值。

注意：加到寄存器里的值 OIMM12 比起二进制操作数 IMM12 需偏置 1。

0000000000000000：

加 0x1

0000000000000001：

加 0x2

.....

11111111111111:

加 0x1000

32 位指令 2	
操作	$RZ \leftarrow R28 + \text{zero\_extend}(\text{OIMM18})$
语法	addi32 rz, r28, oimm18
说明	将带偏置 1 的 18 位立即数（OIMM18）零扩展至 32 位，然后与 R28 的值相加，把结果存入 RZ。 注意：二进制操作数 IMM18 等于 OIMM18 - 1。
影响标志位	无影响
限制	立即数的范围为 0x1-0x40000。
异常	无

32 位指令格式 2:

31	30	26	25	21	20	18	17	0
1	10011		RZ		111			IMM18

IMM18 域:

指定不带偏置立即数的值。

注意：加到寄存器里的值 OIMM18 比起二进制操作数 IMM18 需偏置 1。

0000000000000000:

加 0x1

0000000000000001:

加 0x2

.....

1111111111111111:

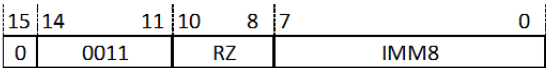
加 0x40000

14.4 ADDI(SP)——无符号（堆栈指针）立即数加法指令

统一化指令		
语法	addi rz, sp, imm	addi sp, sp, imm
操作	$RZ \leftarrow SP + \text{zero\_extend}(IMM)$	$SP \leftarrow SP + \text{zero\_extend}(IMM)$
编译结果	仅存在 16 位指令。 addi rz, sp, imm	仅存在 16 位指令。 addi sp, sp, imm
说明	将立即数（IMM）零扩展至 32 位，然后与堆栈指针（SP）的值相加，把结果存入 RZ 或者 SP。	
影响标志位	无影响	
限制	寄存器的范围为 r0-r7；立即数的范围为 0x0-0x3fc。	
异常	无	

16 位指令 1	
操作	$RZ \leftarrow SP + \text{zero\_extend}(IMM)$
语法	addi16 rz, sp, imm8
说明	将立即数（IMM）零扩展至 32 位，然后与堆栈指针（SP）的值相加，把结果存入 RZ。 注意：立即数（IMM）等于二进制操作数 IMM8 << 2。
影响标志位	无影响
限制	寄存器的范围为 r0-r7；立即数的范围为 (0x0-0xff) << 2。
异常	无

16 位指令格式 1：



IMM8 域：

指定不带移位立即数的值。

注意：加到寄存器里的值 IMM 比起二进制操作数 IMM8 需左移 2 位。

00000000：

加 0x0

00000001：

加 0x4

... ..

11111111：

加 0x3fc

16 位指令 2	
操作	SP ← SP + zero_extend(IMM)
语法	addi16 sp, sp, imm
说明	将立即数（IMM）零扩展至 32 位，然后与堆栈指针（SP）的值相加，把结果存入 RZ。 注意：立即数（IMM）等于二进制操作数 {IMM2, IMM5} << 2。
影响标志位	无影响
限制	源与目的寄存器均为堆栈指针寄存器（R14）；立即数的范围为 (0x0-0x7f) << 2。
异常	无

16 位指令格式 2：

15	14	11	10	9	8	7	5	4	0
0	0010			1	IMM2	000		IMM5	

IMM 域：

指定不带移位立即数的值。

注意：加到寄存器里的值 IMM 比起二进制操作数 {IMM2, IMM5} 需左移 2 位。

{00, 00000}

加 0x0

{00, 00001}

加 0x4

.....

{11, 11111}

加 0x1fc

14.5 ADDU——无符号加法指令

统一化指令		
语 法	addu rz, rx	addu rz, rx, ry
操 作	$RZ \leftarrow RZ + RX$	$RZ \leftarrow RX + RY$
编 译 结 果	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (z<16) and (x<16), then addu16 rz, rx; else addu32 rz, rz, rx;	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (z<8) and (x<8) and (y<8), then addu16 rz, rx, ry; elsif (y==z) and (x<16) and (z<16), then addu16 rz, rx; else addu32 rz, rx, ry;
说 明	将 RZ/RX 与 RX 的值相加，并把结果存在 RZ。	
影 响 标 志 位	无影响	
异 常	无	

16 位指令 1	
操作:	$RZ \leftarrow RZ + RX$
语法	addu16 rz, rx
说明:	将 RZ 与 RX 的值相加，并把结果存在 RZ。
影响标志位:	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式 1:

15	14	10	9	6	5	2	1	0
0	11000			RZ		RX		00

16 位指令 2	
操作	$RZ \leftarrow RX + RY$
语法	addu16 rz, rx, ry
说明	将 RX 与 RY 的值相加，并把结果存在 RZ。
影响标志位	无影响
限制	寄存器的范围为 r0-r7。
异常	无

16 位指令格式 2:

15	14	11	10	8	7	5	4	2	1	0
0	1011	RX	RZ	RY	00					

32 位指令	
操作	$RZ \leftarrow RX + RY$
语法	addu32 rz, rx, ry
说明	将 RX 与 RY 的值相加，并把结果存在 RZ。
影响标志位	无影响
异常	无

32 位指令格式:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001	RY		RX		000000		00001			RZ	

14.6 AND——按位与指令

统一化指令		
语 法	and rz, rx	and rz, rx, ry
操 作	RZ ← RZ and RX	RZ ← RX and RY
编 译 结 果	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (x<16) and (z<16), then and16 rz, rx; else and32 rz, rz, rx;	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (y==z) and (x<16) and (z<16), then and16 rz, rx; else and32 rz, rx, ry;
说 明	将 RZ/RX 与 RX 的值按位与，并把结果存在 RZ。	
影 响 标 志 位	无影响	
异 常	无	

16 位指令	
操作	RZ ← RZ and RX
语法	and16 rz, rx
说明	将 RZ 与 RX 的值按位与，并把结果存在 RZ。
影响标志位	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

15	14	10	9	6	5	2	1	0
0	11010	RZ		RX				00

32 位指令	
操作	$RZ \leftarrow RX \text{ and } RY$
语法	and32 rz, rx, ry
说明	将 RX 与 RY 的值按位与，并把结果存在 RZ。
影响标志位	无影响
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001		RY		RX		001000		00001		RZ	

14.7 ANDI——立即数按位与指令

统一化指令	
语法	andi rz, rx, imm16
操作	$RZ \leftarrow RX \text{ and zero\_extend}(IMM12)$
编译结果	仅存在 32 位指令 andi32 rz, rx, imm12
说明	将 12 位立即数零扩展至 32 位，然后与 RX 的值进行按位与操作，把结果存入 RZ。
影响标志位	无影响
限制	立即数的范围为 0x0-0xFFF。
异常	无

32 位指令	
操作	$RZ \leftarrow RX \text{ and zero\_extend}(IMM12)$
语法	andi32 rz, rx, imm12
说明	将 12 位立即数零扩展至 32 位，然后与 RX 的值进行按位与操作，把结果存入 RZ。
影响标志位	无影响
限制	立即数的范围为 0x0-0xFFF。
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	12	11	0
1	11001		RZ		RX		0010		IMM12	

14.8 ANDN——按位非与指令

统一化指令		
语 法	andn rz, rx	andn rz, rx, ry
操 作	$RZ \leftarrow RZ \text{ and } (!RX)$	$RZ \leftarrow RX \text{ and } (!RY)$
编 译 结 果	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (x<16) and (z<16), then andn16 rz, rx; else andn32 rz, rz, rx;	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (x==z) and (y<16) and (z<16), then andn16 rz, ry; else andn32 rz, rz, rx;
说 明	对于 andn rz, rx, 将 RZ 的值与 RX 的非值按位与, 并把结果存在 RZ; 对于 andn rz, rx, ry, 将 RX 的值与 RY 的非值按位与, 并把结果存在 RZ。	
影 响 标 志 位	无影响	
异 常	无	

16 位指令	
操作	$RZ \leftarrow RZ \text{ and } (!RX)$
语法	andn16 rz, rx
说明	将 RZ 的值与 RX 的非值按位与, 并把结果存在 RZ。
影响标志位	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式:

15	14	10	9	6	5	2	1	0
0	11010			RZ		RX		01

32 位指令	
操作	$RZ \leftarrow RX \text{ and } (!RY)$
语法	andn32 rz, rx, ry
说明	将 RX 的值与 RY 的非值按位与，并把结果存在 RZ。
影响标志位	无影响
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001		RY		RX		001000		00010		RZ	

14.9 ANDNI——立即数按位非与指令

统一化指令	
语法	andni rz, rx, imm16
操作	$RZ \leftarrow RX \text{ and } !(zero\_extend(IMM12))$
编译结果	仅存在 32 位指令 andni32 rz, rx, imm12
说明	将 12 位立即数零扩展至 32 位并取非，然后与 RX 的值进行按位与操作，把结果存入 RZ。
影响标志位	无影响
限制	立即数的范围为 0x0-0xFFF。
异常	无

32 位指令	
操作	$RZ \leftarrow RX \text{ and } !(zero\_extend(IMM12))$
语法	andni32 rz, rx, imm12
说明	将 12 位立即数零扩展至 32 位并取非，然后与 RX 的值进行按位与操作，把结果存入 RZ。
影响标志位	无影响
限制	立即数的范围为 0x0-0xFFF。
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	12	11	0
1	11001		RZ		RX		0011		IMM12	

14.10 ASR——算术右移指令

统一化指令		
语 法	asr rz, rx	asr rz, rx, ry
操 作	$RZ \leftarrow RZ \ggg RX[5:0]$	$RZ \leftarrow RX \ggg RY[5:0]$
编 译 结 果	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (x<16) and (z<16), then asr16 rz, rx; else asr32 rz, rz, rx;	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (x==z) and (y<16) and (z<16), then asr16 rz, ry; else asr32 rz, rx, ry;
说 明	对于 asr rz, rx, 将 RZ 的值进行算术右移（原值右移，左侧移入原符号位的拷贝），结果存入 RZ，右移位数由 RX 低 6 位（RX[5:0]）的值决定；如果 RX[5:0] 的值大于 30，那么 RZ 的值（0 或-1）由 RZ 原值的符号位决定； 对于 asr rz, rx, ry, 将 RX 的值进行算术右移（原值右移，左侧移入原符号位的拷贝），结果存入 RZ，右移位数由 RY 低 6 位（RY[5:0]）的值决定；如果 RY[5:0] 的值大于 30，那么 RZ 的值（0 或-1）由 RX 的符号位决定。	
影 响 标 志 位	无影响	
异 常	无	

16 位指令	
操作	$RZ \leftarrow RZ \ggg RX[5:0]$
语法	asr16 rz, rx
说明	将 RZ 的值进行算术右移（原值右移，左侧移入原符号位的拷贝），结果存入 RZ，右移位数由 R X 低 6 位（RX[5:0]）的值决定；如果 RX[5:0] 的值大于 30，那么 RZ 的值（0 或-1）由 RZ 原值的符号位决定。
影 响 标 志 位	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

15	14	10	9	6	5	2	1	0
0	11100	RZ		RX				10

32 位指令	
操作	$RZ \leftarrow RX \ggg RY[5:0]$
语法	asr32 rz, rx, ry
说明	将 RX 的值进行算术右移（原值右移，左侧移入原符号位的拷贝），结果存入 RZ，右移位数由 RY 低 6 位（RY[5:0]）的值决定；如果 RY[5:0] 的值大于 30，那么 RZ 的值（0 或-1）由 RX 的符号位决定。
影响标志位	无影响
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001		RY		RX		010000		00100			RZ

14.11 ASRC——立即数算术右移至 C 位指令

统一化指令	
语 法	asrc rz, rx, oimm5
操 作	$RZ \leftarrow RX \ggg OIMM5,$ $C \leftarrow RX[OIMM5 - 1]$
编 译 结 果	仅存在 32 位指令 asrc32 rz, rx, oimm5
说 明	将 RX 的值进行算术右移（原值右移，左侧移入原符号位的拷贝），把移出最末位存入条件位 C，移位结果存入 RZ，右移位数由带偏置 1 的 5 位立即数（OIMM5）的值决定。如果 OIMM5 的值等于 32，那么条件位 C 为 RX 的符号位（最高位），RZ 的值（0 或-1）由 RX 的符号位决定。
影 响 标 志 位	$C \leftarrow RX[OIMM5 - 1]$
限 制	立即数的范围为 1-32。
异 常	无

32 位指令	
操作	$RZ \leftarrow RX \ggg OIMM5, C \leftarrow RX[OIMM5 - 1]$
语法	asrc32 rz, rx, oimm5
说明	将 RX 的值进行算术右移（原值右移，左侧移入原符号位的拷贝），把移出最末位存入条件位 C，移位结果存入 RZ，右移位数由带偏置 1 的 5 位立即数（OIMM5）的值决定。如果 OIMM5 的值等于 32，那么条件位 C 为 RX 的符号位（最高位），RZ 的值（0 或-1）由 RX 符号位决定。注意：二进制操作数 IMM5 等于 OIMM5 - 1。
影响标志位	$C \leftarrow RX[OIMM5 - 1]$
限制	立即数的范围为 1-32。
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001	IMM5		RX		010011		00100			RZ	

IMM5 域：

指定不带偏置立即数的值。

注意：移位的值 OIMM5 比起二进制操作数 IMM5 需偏置 1。

00000：

移 1 位

00001：

移 2 位

....

11111：

移 32 位

14.12 ASRI——立即数算术右移指令

统一化指令	
语法	asri rz, rx, imm5
操作	$RZ \leftarrow RX \ggg IMM5$
编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (x<8) and (z<8), then asri16 rz, rx, imm5; else asri32 rz, rx, imm5;
说明	对 asri rz, rx, imm5 而言，将 RX 的值进行算术右移（原值右移，左侧移入原符号位的拷贝），结果存入 RZ，右移位数由 5 位立即数（IMM5）的值决定；如果 IMM5 的值等于 0，那么 RZ 的值将与 RX 相同。
影响标志位	无影响
异常	无

16 位指令	
操作	$RZ \leftarrow RX \ggg IMM5$
语法	asri16 rz, rx, imm5
说明	将 RX 的值进行算术右移（原值右移，左侧移入原符号位的拷贝），结果存入 RZ，右移位数由 5 位立即数（IMM5）的值决定；如果 IMM5 的值等于 0，那么 RZ 的值将不变。
影响标志位	无影响
限制	寄存器的范围为 r0-r7；立即数的范围为 0-31。
异常	无

16 位指令格式：

15	14	11	10	8	7	5	4	0
0	1010			RX		RZ		IMM5

32 位指令	
操作	RZ ← RX >>> IMM5
语法	asri32 rz, rx, imm5
说明	将 RX 的值进行算术右移（原值右移，左侧移入原符号位的拷贝），结果存入 RZ，右移位数由 5 位立即数（IMM5）的值决定；如果 IMM5 的值等于 0，那么 RZ 的值将与 RX 相同。
影响标志位	无影响
限制	立即数的范围为 0-31。
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001	IMM5		RX		010010		00100			RZ	

14.13 BAR——内存访问同步指令

统一化指令	
语法	bar.brwarw(s) bar.bwaw(s)
操作	该指令保证前序所有指令读写顺序比该指令更早被观察到，后续所有指令比该指令更晚被观察到。
说明	仅存在 32 位指令。
说明	S 表征是否广播到核外，“1”为有效。 b 表征前序集合 before，a 表征后续集合 after。两个集合均可具备 R,W,RW 三种读写属性。 br 和 bw 分别表征前序集合是否限制读、写操作； ar 和 aw 分别表征后续集合是否限制读、写操作。
影响标志位	无
异常	无

指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10000	S0000		00000		100001		00001		0	bw br aw ar	

14.14 BCLRI——立即数位清零指令

统一化指令		
语法	bclri rz, imm5	bclri rz, rx, imm5
操作	RZ ← RZ[IMM5] 清零	RZ ← RX[IMM5]
编译结果	清零根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (z<8), then bclri16 rz, imm5; else bclri32 rz, rz, imm5;	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (x==z) and (z<8), then bclri16 rz, imm5; else bclri32 rz, rx, imm5;
说明	将 RZ/RX 的值中，由 IMM5 域值所指示的位清零，其余位保持不变，把清零后的结果存入 RZ。	
影响标志位	无影响	
限制	立即数的范围为 0-31。	

16 位指令	
操作	RZ ← RZ[IMM5] 清零
语法	bclri16 rz, imm5
说明	将 RZ 的值中，由 IMM5 域值所指示的位清零，其余位保持不变，把清零后的结果存入 RZ。
影响标志位	无影响
限制	寄存器的范围为 r0-r7; 立即数的范围为 0-31。
异常	无

16 位指令格式：

15	14	11	10	8	7	5	4	0
0	0111	RZ		100	IMM5			

32 位指令	
操作	$RZ \leftarrow RX[IMM5]$ 清零
语法	bclri32 rz, rx, imm5
说明	将 RX 的值中, 由 IMM5 域值所指示的位清零, 其余位保持不变, 把清零后的结果存入 RZ。
影响标志位	无影响
限制	立即数的范围为 0-31。
异常	无

32 位指令格式:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001	IMM5		RX		001010		00001		RZ		

14.15 BEZ——寄存器等于零分支指令

统一化指令	
语法	bez rx, label
操作	寄存器等于零则程序转移 if (RX == 0) $PC \leftarrow PC + \text{sign\_extend}(\text{offset} \ll 1)$ else $PC \leftarrow PC + 4$
编译结果	仅存在 32 位指令 bez32 rx, label
说明	如果寄存器 RX 等于零, 则程序转移到 label 处执行; 否则程序执行下一条指令, 即 $PC \leftarrow PC + 4$ 。 Label 由当前程序 PC 加上左移 1 位的 16 位相对偏移量有符号扩展到 32 位后的值得到。BEZ 指令的转移范围是 $\pm 64\text{KB}$ 地址空间。
影响标志位	无影响
异常	无

32 位指令	
操作	寄存器等于零则程序转移 if(RX == 0) PC ← PC + sign_extend(offset << 1) else PC ← PC + 4
语法	bez32 rx, label
说明	如果寄存器 RX 等于零，则程序转移到 label 处执行；否则程序执行下一条指令，即 PC ← PC + 4。 Label 由当前程序 P C 加上左移 1 位的 16 位相对偏移量有符号扩展到 32 位后的值得到。BEZ 指令的转移范围是 ±64KB 地址空间。
影响标志位	无影响
异常	无

32 位指令格式：

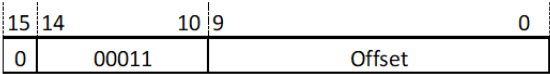
31	30	26	25	21	20	16	15	0
1	11010	01000		RX				Offset

14.16 BF——C 为 0 分支指令

统一化指令	
语法	bf label
操作	C 等于零则程序转移。 if(C==0) PC ← PC + sign_extend(offset << 1); else PC ← next PC;
编译结果	根据跳转的范围编译为对应的 16 位或 32 位指令。 if (offset<1KB), then bf16 label; else bf32 label;
说明	如果条件标志位 C 等于零，则程序转移到 label 处执行；否则程序执行下一条指令。 Label 由当前程序 PC 加上左移 1 位的相对偏移量有符号扩展到 32 位后的值得到。BF 指令的转移范围是 ±64KB 地址空间。
影响标志位	无影响
异常	无

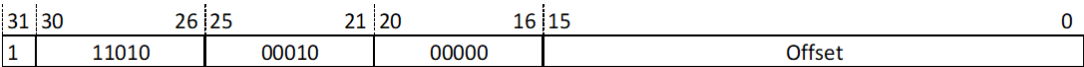
16 位指令	
操作	C 等于零则程序转移。 if(C==0) PC ← PC + sign_extend(offset << 1) else PC ← PC + 2
语法	bf16 label
说明	如果条件标志位 C 等于 0，则程序转移到 label 处执行；否则程序执行下一条指令，即 PC ← PC + 2。 Label 由当前程序 P C 加上左移 1 位的 10 位相对偏移量有符号扩展到 32 位后的值得到。BF16 指令的转移范围是 ±1KB 地址空间。
影响标志位	无影响
异常	无

16 位指令格式：



32 位指令	
操作	C 等于零则程序转移 if(C == 0) PC ← PC + sign_extend(offset << 1) else PC ← PC + 4
语法	bf32 label
说明	如果条件标志位 C 等于零，则程序转移到 label 处执行；否则程序执行下一条指令，即 PC ← PC + 4。 Label 由当前程序 PC 加上左移 1 位的 16 位相对偏移量有符号扩展到 32 位后的值得到。BF 指令的转移范围是 ±64KB 地址空间。
影响标志位	无影响
异常	无

32 位指令格式：



14.17 BGENI——立即数位产生指令

统一化指令	
语法	bgeni rz, imm5
操作	RZ ← (2) ^{IMM5}
编译结果	仅存在 32 位指令。 bgeni32 rz, imm5
说明	对由 5 位立即数确定的 RZ 的位 (RZ[IMM5]) 置位，并清除 RZ 的其它位。 注意，如果 IMM5 小于 16，该指令是 movi rz, (2) ^{IMM5} 的伪指令；如果 IMM5 大于等于 16，该指令是 movih rz, (2) ^{IMM5} 的伪指令。
影响标志位	无影响
限制	立即数的范围为 0-31。
异常	无

32 位指令	
操作	$RZ \leftarrow (2)^{IMM5};$
语法	bgeni32 rz, imm5
说明	对由 5 位立即数确定的 RZ 的位 (RZ[IMM5]) 置位，并清除 RZ 的其它位。 注意，如果 IMM5 小于 16，该指令是 movi32 rz, (2) ^{IMM5} 的伪指令；如果 IMM5 大于等于 16，该指令是 movih32 rz, (2) ^{IMM5} 的伪指令。
影响标志位	无影响
限制	立即数的范围为 0-31。
异常	无

32 位指令格式：

如果IMM5小于16:

31	30	26	25	21	20	16	15	0
1	11010	10000			RZ			(2) ^{IMM5}

如果IMM5大于等于16:

31	30	26	25	21	20	16	15	0
1	11010	10001			RZ			(2) ^{IMM5}

14.18 BGENR——寄存器位产生指令

统一化指令	
语法	bgenr rz, rx
操作	If (RX[5] == 0) , then $RZ \leftarrow 2^{RX[4:0]};$ else $RZ \leftarrow 0;$
编译结果	仅存在 32 位指令 bgenr32 rz, rx
说明	如果 RX[5] 为 0，那么置 RZ 由 RX 低五位 (RX[4:0]) 确定的寄存器位，并清除 RZ 所有其它的位；否则，对 RZ 清零。
影响标志位	无影响
异常	无

32 位指令	
操作	If (RX[5] == 0) , then RZ $\leftarrow 2^{RX[4:0]}$; else RZ $\leftarrow 0$;
语法	bgenr32 rz, rx
说明	如果 R X[5] 为 0, 那么置 RZ 由 RX 低五位 (RX[4:0]) 确定的寄存器位, 并清除 RZ 所有其它的位; 否则, 对 RZ 清零。
影 响 标 志 位	无影响
异常	无

32 位指令格式:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001	00000		RX		010100		00010			RZ	

14.19 BHSZ——寄存器大于等于零分支指令

统一化指令	
语法	bhsz rx, label
操作	寄存器大于等于零则程序转移 if(RX >= 0) PC $\leftarrow$ PC + sign_extend(offset << 1); else PC $\leftarrow$ PC + 4;
编译 结果	仅存在 32 位指令 bhsz32 rx, label
说明	如果寄存器 RX 大于或等于零, 则程序转移到 label 处执行; 否则程序执行下一条指令, 即 PC $\leftarrow$ PC + 4。 Label 由当前程序 PC 加上左移 1 位的 16 位相对偏移量有符号扩展到 32 位后的值得到。BHSZ 指令的转移范围是 $\pm 64\text{KB}$ 地址空间。
影响 标志 位	无影响
异常	无

32 位指令	
操作	寄存器大于等于零则程序转移 if(RX >= 0) PC ← PC + sign_extend(offset << 1) else PC ← PC + 4
语法	bhsz32 rx, label
说明	如果寄存器 RX 大于或等于零，则程序转移到 label 处执行；否则程序执行下一条指令，即 PC ← PC + 4。 Label 由当前程序 PC 加上左移 1 位的 16 位相对偏移量有符号扩展到 32 位后的值得到。BHSZ 指令的转移范围是 ±64KB 地址空间。
影响标志位	无影响
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	0
1	11010	01101		RX		Offset		

14.20 BHZ——寄存器大于零分支指令

统一化指令	
语法	bhz rx, label
操作	寄存器大于零则程序转移 if(RX > 0) PC ← PC + sign_extend(offset << 1) else PC ← PC + 4
编译结果	仅存在 32 位指令 bhz32 rx, label
说明	如果寄存器 RX 大于零，则程序转移到 label 处执行；否则程序执行下一条指令，即 PC ← PC + 4。 Label 由当前程序 PC 加上左移 1 位的 16 位相对偏移量有符号扩展到 32 位后的值得到。BHZ 指令的转移范围是 ±64KB 地址空间。
影响标志位	无影响
异常	无

32 位指令	
操作	寄存器大于零则程序转移 if(RX > 0) PC ← PC + sign_extend(offset << 1) else PC ← PC + 4
语法	bhz32 rx, label
说明	如果寄存器 RX 大于零，则程序转移到 label 处执行；否则程序执行下一条指令，即 PC ← PC + 4。 Label 由当前程序 P C 加上左移 1 位的 16 位相对偏移量有符号扩展到 32 位后的值得到。BHZ 指令的转移范围是 ±64KB 地址空间。
影响标志位	无影响
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	0
1	11010	01010		RX		Offset		

14.21 BKPT——断点指令

统一化指令	
操作	引起一个断点异常或者进入调试模式
编译结果	总是编译为 16 位指令。 bkpt16
说明	断点指令
影响标志位	无影响
异常	断点异常

16 位指令	
操作	引起一个断点异常或者进入调试模式
语法	bkpt16
说明	断点指令
影响标志位	无影响
异常	断点异常

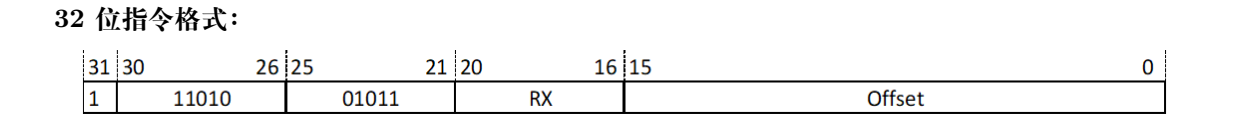
16 位指令格式：

15	14	10	9	0
0	00000	0000000000		

14.22 BLSZ——寄存器小于等于零分支指令

统一化指令	
语法	blsz rx, label
操作	寄存器小于等于零则程序转移 if(RX <= 0) PC ← PC + sign_extend(offset << 1) else PC ← PC + 4
编译结果	仅存在 32 位指令 blsz32 rx, label
说明	如果寄存器 RX 小于或等于零，则程序转移到 label 处执行；否则程序执行下一条指令，即 PC + 4。 Label 由当前程序 PC 加上左移 1 位的 16 位相对偏移量有符号扩展到 32 位后的值得到。BLSZ 指令的转移范围是 ±64KB 地址空间。
影响标志位	无影响
异常	无

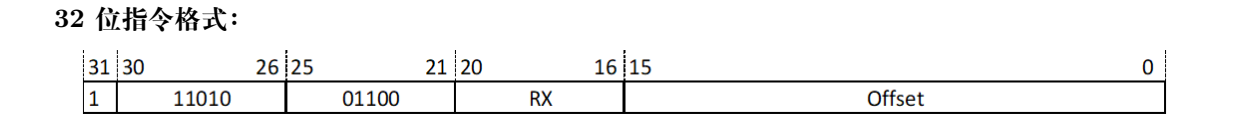
32 位指令	
操作	寄存器小于等于零则程序转移 if(RX <= 0) PC ← PC + sign_extend(offset << 1) else PC ← PC + 4
语法	blsz32 rx, label
说明	如果寄存器 RX 小于或等于零，则程序转移到 label 处执行；否则程序执行下一条指令，即 PC + 4。 Label 由当前程序 PC 加上左移 1 位的 16 位相对偏移量有符号扩展到 32 位后的值得到。BLSZ 指令的转移范围是 ±64KB 地址空间。
影响标志位	无影响
异常	无



14.23 BLZ——寄存器小于零分支指令

统一化指令	
语法	blz rx, label
操作	寄存器小于零则程序转移 if(RX < 0) PC ← PC + sign_extend(offset << 1) else PC ← PC + 4
编译结果	仅存在 32 位指令 blz32 rx, label
说明	如果寄存器 RX 小于零，则程序转移到 label 处执行；否则程序执行下一条指令，即 PC ← PC + 4。 Label 由当前程序 P C 加上左移 1 位的 16 位相对偏移量有符号扩展到 32 位后的值得到。BLZ 指令的转移范围是 ±64KB 地址空间。
影响标志位	无影响
异常	无

32 位指令	
操作	寄存器小于零则程序转移 if(RX < 0) PC ← PC + sign_extend(offset << 1) else PC ← PC + 4
语法	blz32 rx, label
说明	如果寄存器 RX 小于零，则程序转移到 label 处执行；否则程序执行下一条指令，即 PC ← PC + 4。 Label 由当前程序 P C 加上左移 1 位的 16 位相对偏移量有符号扩展到 32 位后的值得到。BLZ 指令的转移范围是 ±64KB 地址空间。
影响标志位	无影响
异常	无



14.24 BMASKI——立即数位屏蔽产生指令

统一化指令	
语 法	bmaski rz, oimm5
操 作	$RZ \leftarrow (2)^{OIMM5} - 1$
编 译 结 果	仅存在 32 位指令 bmaski32 rz, oimm5
说 明	产生连续低位为 1、高位为 0 的立即数，并将该立即数存入 RZ。带偏置的立即数 OIMM5 指定被置 1 的连续低位（RX[OIMM5-1:0]）的位数，其余高位清零。当 OIMM5 为 0 或 32 时，RX 所有位均被置 1。 注意，OIMM5 为 1-16 时由 movi 指令执行。
影 响 标 志 位	无影响
限 制	立即数的范围为 0，17-32；
异 常	无

32 位指令	
操作	$RZ \leftarrow (2)^{OIMM5} - 1$
语法	bmaski32 rz, oimm5
说明	产生连续低位为 1、高位为 0 的立即数，并将该立即数存入 RZ。带偏置的立即数 OIMM5 指定被置 1 的连续低位（RX[OIMM5-1:0]）的位数，其余高位清零。当 OIMM5 为 0 或 32 时，RX 所有位均被置 1。 注意，OIMM5 为 1 -16 时由 movi 指令执行；二进制操作数 IMM5 等于 OIMM5 - 1。
影响标志位	无影响
限制	立即数的范围为 0，17-32；
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001		IMM5		00000		010100		00001			RZ

IMM5 域：

指定被置 1 的连续低位的最高位。

注意：立即数 OIMM5 比起二进制操作数 IMM5 需偏置 1。

10000：

0-16 位置位

10001：

0-17 位置位

.....

11111：

0-31 位置位

14.25 BNEZ——寄存器不等于零分支指令

统一化指令	
语法	bnez rx, label
操作	寄存器不等于零则程序转移 if(RX != 0) PC ← PC + sign_extend(offset << 1) else PC ← PC + 4
编译结果	仅存在 32 位指令。 bnez32 rx, label
说明	如果寄存器 RX 不等于零，则程序转移到 label 处执行；否则程序执行下一条指令，即 PC ← PC + 4。 Label 由当前程序 PC 加上左移 1 位的 16 位相对偏移量有符号扩展到 32 位后的值得到。BNEZ 指令的转移范围是 ±64KB 地址空间。
影响标志位	无影响
异常	无

32 位指令	
操作	寄存器不等于零则程序转移 if(RX != 0) PC ← PC + sign_extend(offset << 1) else PC ← PC + 4
语法	bnez32 rx, label
说明	如果寄存器 RX 不等于零，则程序转移到 label 处执行；否则程序执行下一条指令，即 PC ← PC + 4。 Label 由当前程序 PC 加上左移 1 位的 16 位相对偏移量有符号扩展到 32 位后的值得到。BNEZ 指令的转移范围是 ±64KB 地址空间。
影响标志位	无影响
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	0
1	11010	01001		RX				Offset

14.26 BNEZAD——寄存器自减大于零分支指令

统一化指令	
语 法	bnezad rx, label
操 作	寄存器自减 1 后判断，大于零则程序转移，自减结果无条件更新寄存器。 $RX \leftarrow RX-1$ if($RX > 0$) $PC \leftarrow PC + sign\_extend(offset \ll 1)$ else $PC \leftarrow PC + 4$
编 译 结 果	仅存在 32 位指令。 bnezad rx, label
说 明	寄存器 RX 自减 1 后判断，若结果大于零，则程序转移到 label 处执行；否则程序执行下一条指令，即 $PC \leftarrow PC + 4$ 。自减结果无条件更新寄存器 RX。 Label 由当前程序 PC 加上左移 1 位的 16 位相对偏移量有符号扩展到 32 位后的值得到。BNEZAD 指令的转移范围是 $\pm 64KB$ 地址空间。
影 响 标 志 位	无影响
异 常	无

32 位指令	
操作	寄存器不等于零则程序转移 $RX \leftarrow RX-1$ if($RX > 0$) $PC \leftarrow PC + \text{sign\_extend}(\text{offset} \ll 1)$ else $PC \leftarrow PC + 4$
语法	bnezad32 rx, label
说明	寄存器 RX 自减 1 后判断，如果结果大于零，则程序转移到 label 处执行；否则程序执行下一条指令，即 $PC \leftarrow PC + 4$ 。自减结果无条件更新寄存器 RX。 Label 由当前程序 PC 加上左移 1 位的 16 位相对偏移量有符号扩展到 32 位后的值得到。BNEZAD 指令的转移范围是 $\pm 64\text{KB}$ 地址空间。
影响标志位	无影响
异常	无

32 位指令格式：

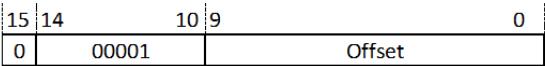
31	30	26	25	21	20	16	15	0
1	11010	00001		RX		Offset		

14.27 BR——无条件跳转指令

统一化指令	
语法	br label
操作	$PC \leftarrow PC + \text{sign\_extend}(\text{offset} \ll 1)$
编译结果	根据跳转的范围编译为对应的 16 位或 32 位指令 if(offset<1KB), then br16 label; else br32 label;
说明	程序无条件跳转到 label 处执行。 Label 由当前程序 PC 加上左移 1 位的相对偏移量有符号扩展到 32 位后的值得到。
影响标志位	无影响
异常	无

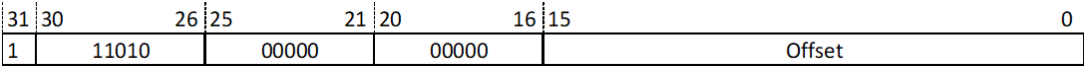
16 位指令	
操作	$PC \leftarrow PC + \text{sign\_extend}(\text{offset} \ll 1)$
语法	br16 label
说明	程序无条件跳转到 label 处执行。 Label 由当前程序 PC 加上左移 1 位的 10 位相对偏移量有符号扩展到 32 位后的值得到。BR16 指令的跳转范围是 ±1KB 地址空间。
影响标志位	无影响
异常	无

16 位指令格式：



32 位指令	
操作	$PC \leftarrow PC + \text{sign\_extend}(\text{offset} < 1)$
语法	br32 label
说明	程序无条件跳转到 label 处执行。 Label 由当前程序 PC 加上左移 1 位的 16 位相对偏移量有符号扩展到 32 位后的值得到。BR 指令的跳转范围是 $\pm 64\text{KB}$ 地址空间。
影响标志位	无影响
异常	无

32 位指令格式：



14.28 BREV——位倒序指令

统一化指令	
语法	brev rz, rx
操作	for i=0 to 31 RZ[i] $\leftarrow$ RX[31-i];
编译结果	仅存在 32 位指令。 brev32 rz, rx
说明	把 RX 的值按位取倒序，结果存入 RZ。 如果 RX 的值为” abc defghijklmnopqrstuvwxyz012345” ，按位取倒序后，RZ 的值变为” 543210zyxwvutsrqponmlkjihgfedcba”。
影响标志位	无影响
异常	无

32 位指令	
操作	for i=0 to 31 RZ[i] ← RX[31-i];
语法	brev32 rz, rx
说明	把 RX 的值按位取倒序，结果存入 RZ。 如果 RX 的值为” abc defghijklmnopqrstuvwxyz012345” ，按位取倒序后，RZ 的值变为” 543210zyxwvutsrqponmlkjihgfedcba”。
影 响 标 志 位	无影响
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001	00000		RX		011000		10000			RZ	

14.29 BSETI——立即数位置位指令

统一化指令		
语 法	bseti rz, imm5	bseti rz, rx, imm5
操 作	RZ ← RZ[IMM5] 置位	RZ ← RX[IMM5] 置位
编 译 结 果	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (z<8), then bseti16 rz, imm5; else bseti32 rz, rz, imm5;	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if((x==z) and (z<8)), then bseti16 rz, imm5; else bseti32 rz, rx, imm5;
说 明	将 RZ/RX 的值中，由 IMM5 域值所指示的位置 1，其余位保持不变，把置位后的结果存入 RZ。	
影 响 标 志 位	无影响	
限 制	立即数的范围为 0-31。	
异 常	无	

16 位指令	
操作	RZ ← RZ[IMM5] 置位
语法	bseti16 rz, imm5
说明	将 RZ 的值中，由 IMM5 域值所指示的位置 1，其余位保持不变，把置位后的结果存入 RZ。
影 响 标 志 位	无影响
限制	寄存器的范围为 r0-r7；立即数的范围为 0-31。
异常	无

16 位指令格式：

15	14	11	10	8	7	5	4	0
0	0111	RZ	101	IMM5				

32 位指令	
操作	$RZ \leftarrow RX[IMM5]$ 置位
语法	bseti32 rz, rx, imm5
说明	将 RX 的值中, 由 IMM5 域值所指示的位置 1, 其余位保持不变, 把置位后的结果存入 RZ。
影响标志位	无影响
限制	立即数的范围为 0-31。
异常	无

32 位指令格式：

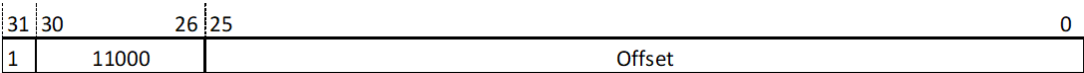
31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001	IMM5		RX		001010		00010			RZ	

14.30 BSR——跳转到子程序指令

统一化指令	
语法	bsr label
操作	链接并跳转到子程序： $R15 \leftarrow \text{next PC}$ $PC \leftarrow PC + \text{sign\_extend}(\text{offset} \ll 1)$
编译结果	仅存在 32 位指令 bsr32 label;
说明	子程序跳转，将子程序的返回地址（下一条指令的 PC）保存在链接寄存器 R15 中，程序转移到 label 处执行。 Label 由当前程序 PC 加上左移 1 位的相对偏移量有符号扩展到 32 位后的值得到。
影响标志位	无影响
异常	无

32 位指令	
操作	链接并跳转到子程序： $R15 \leftarrow PC+4$ $PC \leftarrow PC + \text{sign\_extend}(\text{offset} \ll 1)$
语法	bsr32 label
说明	子程序跳转，将子程序的返回地址（下一条指令的 PC，即当前 PC+4）保存在链接寄存器 R15 中，程序转移到 label 处执行。 Label 由当前程序 PC 加上左移 1 位的 26 位相对偏移量有符号扩展到 32 位后的值得到。BSR 指令的跳转范围是 $\pm 64\text{MB}$ 地址空间。
影响标志位	无影响
异常	无

32 位指令格式：

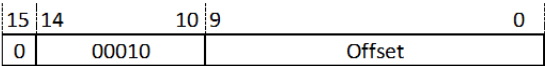


14.31 BT——C 为 1 分支指令

统一化指令	
语法	bt label
操作	if(C == 1) PC ← PC + sign_extend(offset << 1); else PC ← next PC;
编译结果	根据跳转的范围编译为对应的 16 位或 32 位指令。 if (offset<1KB), then bt16 label; else bt32 label;
说明	如果条件标志位 C 等于 1，则程序转移到 label 处执行；否则程序执行下一条指令。 Label 由当前程序 PC 加上左移 1 位的相对偏移量有符号扩展到 32 位后的值得到。BT 指令的转移范围是 ±64KB 地址空间。
影响标志位	无影响
异常	无

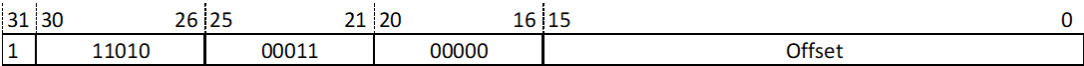
16 位指令	
操作	C 等于一则程序转移 if(C == 1) PC ← PC + sign_extend(offset << 1) else PC ← PC + 2
语法	bt16 label
说明	如果条件标志位 C 等于 1，则程序转移到 label 处执行；否则程序执行下一条指令，即 PC ← PC + 2。 Label 由当前程序 P C 加上左移 1 位的 10 位相对偏移量有符号扩展到 32 位后的值得到。BT16 指令的转移范围是 ±1KB 地址空间。
影响标志位	无影响
异常	无

16 位指令格式：



32 位指令	
操作	C 等于一则程序转移 if(C == 1) PC ← PC + sign_extend(offset << 1) else PC ← PC + 4
语法	bt32 label
说明	如果条件标志位 C 等于 1，则程序转移到 label 处执行；否则程序执行下一条指令，即 PC ← PC + 4。 Label 由当前程序 PC 加上左移 1 位的 16 位相对偏移量有符号扩展到 32 位后的值得到。BT 指令的转移范围是 ±64KB 地址空间。
影响标志位	无影响
异常	无

32 位指令格式：



14.32 BTSTI——立即数位测试指令

统一化指令	
语法	btsti rx, imm5
操作	C ← RX[IMM5]
编译结果	仅存在 32 位指令 btsti32 rx, imm5
说明	对由 IMM5 决定的 RX 的位 (RX[IMM5]) 进行测试，并使条件位 C 的值等于该位的值。
影响标志位	C ← RX[IMM5]
限制	立即数的范围为 0-31。
异常	无

32 位指令	
操作	$C \leftarrow \text{RX}[\text{IMM5}]$
语法	btsti32 rx, imm5
说明	对由 IMM5 决定的 RX 的位 ($\text{RX}[\text{IMM5}]$) 进行测试, 并使条件位 C 的值等于该位的值。
影响标志位	$C \leftarrow \text{RX}[\text{IMM5}]$
限制	立即数的范围为 0-31。
异常	无

32 位指令格式:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001	IMM5				RX		001010		00100		00000

14.33 CLRF——C 为 0 清零指令

统一化指令	
语法	clrf rz
操作	if $C == 0$, then $RZ \leftarrow 0$; else $RZ \leftarrow RZ$;
编译结果	仅存在 32 位指令 clrf32 rz
说明	如果 C 为 0, 寄存器 RZ 被清零; 否则, 寄存器 RZ 保持不变。
影响标志位	无影响
异常	无

32 位指令	
操作	if $C == 0$, then $RZ \leftarrow 0$; else $RZ \leftarrow RZ$;
语法	clrf32 rz
说明	如果 C 为 0, 寄存器 RZ 被清零; 否则, 寄存器 RZ 保持不变。
影响标志位	无影响
异常	无

32 位指令格式:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001		RZ		00000		001011		00001		00000	

14.34 CLRT——C 为 1 清零指令

统一化指令	
语法	clrt rz
操作	if C==1, then RZ $\leftarrow$ 0; else RZ $\leftarrow$ RZ;
编译结果	仅存在 32 位指令 clrt32 rz
说明	如果 C 为 1，寄存器 RZ 被清零；否则，寄存器 RZ 保持不变。
影响标志位	无影响
异常	无

32 位指令	
操作	if C==1, then RZ $\leftarrow$ 0; else RZ $\leftarrow$ RZ;
语法	clrt32 rz
说明	如果 C 为 1，寄存器 RZ 被清零；否则，寄存器 RZ 保持不变。
影响标志位	无影响
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001		RZ		00000		001011		00010		00000	

14.35 CMPHS——无符号大于等于比较指令

统一化指令	
语法	cmp _{phs} rx, ry
操作	RX 与 RY 作无符号比较。 If $RX \geq RY$, then $C \leftarrow 1$; else $C \leftarrow 0$;
编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if $(x < 16)$ and $(y < 16)$, then cmp _{phs16} rx, ry; else cmp _{phs32} rx, ry;
说明	将 RX 的值减去 RY 的值，结果与 0 作比较，并对 C 位进行更新。cmp _{phs} 进行无符号比较，即操作数被认为是无符号数。如果 RX 大于等于 RY，即减法结果大于等于 0，则设置条件位 C；否则，清除条件位 C。
影响标志位	根据比较结果设置条件位 C
异常	无

16 位指令	
操作	RX 与 RY 作无符号比较。 If $RX \geq RY$, then $C \leftarrow 1$; else $C \leftarrow 0$;
语法	cmp _{phs16} rx, ry
说明	将 RX 的值减去 RY 的值，结果与 0 作比较，并对 C 位进行更新。cmp _{phs16} 进行无符号比较，即操作数被认为是无符号数。如果 RX 大于等于 RY，即减法结果大于等于 0，则设置条件位 C；否则，清除条件位 C。
影响标志位	根据比较结果设置条件位 C
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

15	14	10	9	6	5	2	1	0
0	11001		RY		RX		00	

32 位指令	
操作	RX 与 RY 作无符号比较。 If RX >= RY, then C ← 1; else C ← 0;
语法	cmphs32 rx, ry
说明	将 RX 的值减去 RY 的值，结果与 0 作比较，并对 C 位进行更新。cmphs32 进行无符号比较，即操作数被认为是无符号数。如果 RX 大于等于 RY，即减法结果大于等于 0，则设置条件位 C；否则，清除条件位 C。
影响标志位	根据比较结果设置条件位 C
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1		10001			RY		RX		000001		00001	00000

14.36 CMPHSI——立即数无符号大于等于比较指令

统一化指令	
语 法	cmphsi rx, oimm16
操 作	RX 与立即数作无符号比较。 If RX >= zero_extend(OIMM16), C ← 1; else C ← 0;
编 译 结 果	根据立即数和寄存器的范围编译为对应的 16 位或 32 位指令。 if (oimm16<33) and (x<8),then cmphsi16 rx, oimm5; else cmphsi32 rx, oimm16;
说 明	将带偏置 1 的 16 位立即数（OIMM16）零扩展至 32 位，然后用 RX 的值减去该 32 位值，结果与 0 作比较，并对 C 位进行更新。cmphsi 进行无符号比较，即操作数被认为是无符号数。如果 RX 大于等于零扩展后的 OIMM16，即减法结果大于等于 0，则设置条件位 C；否则，清除条件位 C。
影 响 标 志 位	根据比较结果设置条件位 C
限 制	立即数的范围为 0x1-0x10000。
异 常	无

16 位指令	
操作	RX 与立即数作无符号比较。 If $RX \geq \text{zero\_extend}(\text{OIMM5})$, then $C \leftarrow 1$; else $C \leftarrow 0$;
语法	cmphsi16 rx, oimm5
说明	将带偏置 1 的 5 位立即数（OIMM5）零扩展至 32 位，然后用 RX 的值减去该 32 位值，结果与 0 作比较，并对 C 位进行更新。cmphsi16 进行无符号比较，即操作数被认为是无符号数。如果 RX 大于等于零扩展后的 OIMM5，即减法结果大于等于 0，则设置条件位 C；否则，清除条件位 C。 注意：二进制操作数 IMM5 等于 OIMM5 - 1。
影响标志位	根据比较结果设置条件位 C
限制	寄存器的范围为 r0-r7；立即数的范围为 1-32。
异常	无

16 位指令格式：

15	14	11	10	8	7	5	4	0
0	0111			RX		000		IMM5

IMM5 域：

指定不带偏置立即数的值。

注意：参与比较的立即数 OIMM5 比起二进制操作数 IMM5 需偏置 1。

00000：

与 1 比较

00001：

与 2 比较

... ..

11111：

与 32 比较

32 位指令	
操作	RX 与立即数作无符号比较。 If $RX \geq \text{zero\_extend}(\text{OIMM16})$, then $C \leftarrow 1$; else $C \leftarrow 0$;
语法	cmphi32 rx, oimm16
说明	将带偏置 1 的 16 位立即数 (OIMM16) 零扩展至 32 位, 然后用 RX 的值减去该 32 位值, 结果与 0 作比较, 并对 C 位进行更新。cmphi32 进行无符号比较, 即操作数被认为是无符号数。如果 RX 大于等于零扩展后的 OIMM16, 即减法结果大于等于 0, 则设置条件位 C; 否则, 清除条件位 C。 注意: 二进制操作数 IMM16 等于 OIMM16 - 1。
影响标志位	根据比较结果设置条件位 C
限制	立即数的范围为 0x1-0x10000。
异常	无

32 位指令格式:

31	30	26	25	21	20	16	15	0
1	11010	11000	11000	IMM16				

IMM16 域:

指定不帶偏置立即數的值。

注意：参与比较的立即数 OIMM16 比起二进制操作数 IMM16 需偏置 1。

00000000000000000000:

与 0x1 比较

00000000000000000001:

与 0x2 比较

• • • • •

111111111111111111:

与 0x10000 比较

14.37 CMPLT——有符号小于比较指令

统一化指令	
语法	cmplt rx, ry
操作	RX 与 RY 作有符号比较。 If $RX < RY$, then $C \leftarrow 1$; else $C \leftarrow 0$;
编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if $(x < 16)$ and $(y < 16)$, then cmplt16 rx, ry; else cmplt32 rx, ry;
说明	将 RX 的值减去 RY 的值，结果与 0 作比较，并对 C 位进行更新。cmplt 进行有符号比较，即操作数被认为是补码形式的有符号数。如果 RX 小于 RY，即减法结果小于 0，则设置条件位 C；否则，清除条件位 C。
影响标志位	根据比较结果设置条件位 C
异常	无

16 位指令	
操作	RX 与 RY 作有符号比较。 If $RX < RY$, then $C \leftarrow 1$; else $C \leftarrow 0$;
语法	cmplt16 rx, ry
说明	将 RX 的值减去 RY 的值，结果与 0 作比较，并对 C 位进行更新。cmplt16 进行有符号比较，即操作数被认为是补码形式的有符号数。如果 RX 小于 RY，即减法结果小于 0，则设置条件位 C；否则，清除条件位 C。
影响标志位	根据比较结果设置条件位 C
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

15	14	10	9	6	5	2	1	0
0	11001		RY		RX		01	

32 位指令	
操作	RX 与 RY 作有符号比较。 If RX < RY, then C ← 1; else C ← 0;
语法	cmplt32 rx, ry
说明	将 RX 的值减去 RY 的值，结果与 0 作比较，并对 C 位进行更新。cmplt32 进行有符号比较，即操作数被认为是补码形式的有符号数。如果 RX 小于 RY，即减法结果小于 0，则设置条件位 C；否则，清除条件位 C。
影响标志位	根据比较结果设置条件位 C
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001		RY		RX		000001		00010		00000	

14.38 CMPLTI——立即数有符号小于比较指令

统一化指令	
语 法	cmplti rx, oimm16
操 作	RX 与立即数作有符号比较。 If $RX < \text{zero\_extend}(OIMM16)$, $C \leftarrow 1$; else $C \leftarrow 0$;
编 译 结 果	根据立即数和寄存器的范围编译为对应的 16 位或 32 位指令。 if $(x < 8)$ and $(oimm16 < 33)$, then cmplti16 rx, oimm5; else cmplti32 rx, oimm16;
说 明	将带偏置 1 的 16 位立即数 (OIMM16) 零扩展至 32 位，然后用 RX 的值减去该 32 位值，结果与 0 作比较，并对 C 位进行更新。cmplti 进行有符号比较，即 RX 的值被认为是补码形式的有符号数。如果 RX 小于零扩展后的 OIMM16，即减法结果小于 0，则设置条件位 C；否则，清除条件位 C。
影 响 标 志 位	根据比较结果设置条件位 C
限 制	立即数的范围为 0x1-0x10000。
异 常	无

16 位指令	
操作	RX 与立即数作有符号比较。 If $RX < \text{zero\_extend}(\text{OIMM5})$, then $C \leftarrow 1$; else $C \leftarrow 0$;
语法	cmplti16 rx, oimm5
说明	将带偏置 1 的 5 位立即数（OIMM5）零扩展至 32 位，然后用 RX 的值减去该 32 位值，结果与 0 作比较，并对 C 位进行更新。cm plti16 进行有符号比较，即 RX 的值被认为是补码形式的有符号数。如果 RX 小于零扩展后的 OIMM5，即减法结果小于 0，则设置条件位 C；否则，清除条件位 C。 注意：二进制操作数 IMM5 等于 OIMM5 - 1。
影响标志位	根据比较结果设置条件位 C
限制	寄存器的范围为 r0-r7；立即数的范围为 1-32。
异常	无

16 位指令格式：

15	14	11	10	8	7	5	4	0
0	0111			RX	001		IMM5	

IMM5 域：

指定不带偏置立即数的值。

注意：参与比较的立即数 OIMM5 比起二进制操作数 IMM5 需偏置 1。

00000：

与 1 比较

00001：

与 2 比较

.....

11111：

与 32 比较

32 位指令	
操作	RX 与立即数作有符号比较。 If $RX < \text{zero\_extend}(\text{OIMM16})$, then $C \leftarrow 1$; else $C \leftarrow 0$;
语法	cmplti32 rx, oimm16
说明	将带偏置 1 的 16 位立即数 (OIMM16) 零扩展至 32 位, 然后用 RX 的值减去该 32 位值, 结果与 0 作比较, 并对 C 位进行更新。cm plti32 进行有符号比较, 即 RX 的值被认为是补码形式的有符号数。如果 RX 小于零扩展后的 OIMM16, 即减法结果小于 0, 则设置条件位 C; 否则, 清除条件位 C。 注意: 二进制操作数 IMM16 等于 OIMM16 - 1。
影响标志位	根据比较结果设置条件位 C
限制	立即数的范围为 0x1-0x10000。
异常	无

32 位指令格式:

31	30	26	25	21	20	16	15	0
1	11010	11001			RX			IMM16

IMM16 域:

指定不带偏置立即数的值。

注意: 参与比较的立即数 OIMM16 比起二进制操作数 IMM16 需偏置 1。

0000000000000000:

与 0x1 比较

0000000000000001:

与 0x2 比较

.....

1111111111111111:

与 0x10000 比较

14.39 CMPNE——不等比较指令

统一化指令	
语法	cmpne rx, ry
操作	RX 与 RY 作比较。 If RX != RY, then C ← 1; else C ← 0;
编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (x<16) and (y<16), then cmpne16 rx, ry; else cmpne32 rx, ry;
说明	将 RX 的值减去 RY 的值，结果与 0 作比较，并对 C 位进行更新。如果 RX 不等于 RY，即减法结果不等于 0，则设置条件位 C；否则，清除条件位 C。
影响标志位	根据比较结果设置条件位 C
异常	无

16 位指令	
操作	RX 与 RY 作比较。 If RX != RY, then C ← 1; else C ← 0;
语法	cmpne16 rx, ry
说明	将 RX 的值减去 RY 的值，结果与 0 作比较，并对 C 位进行更新。如果 RX 不等于 RY，即减法结果不等于 0，则设置条件位 C；否则，清除条件位 C。
影响标志位	根据比较结果设置条件位 C
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

15	14	10	9	6	5	2	1	0
0	11001			RY		RX		10

32 位指令	
操作	RX 与 RY 作比较。 If $RX \neq RY$, then $C \leftarrow 1$; else $C \leftarrow 0$;
语法	cmpne32 rx, ry
说明	将 RX 的值减去 RY 的值，结果与 0 作比较，并对 C 位进行更新。如果 RX 不等于 RY，即减法结果不等于 0，则设置条件位 C；否则，清除条件位 C。
影响标志位	根据比较结果设置条件位 C
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001		RY		RX		000001		00100		00000	

14.40 CMPNEI——立即数不等比较指令

统一化指令	
语法	cmpnei rx, imm16
操作	RX 与立即数作比较。 If $RX \neq \text{zero\_extend}(\text{imm16})$, $C \leftarrow 1$; else $C \leftarrow 0$;
编译结果	根据立即数和寄存器的范围编译为对应的 16 位或 32 位指令。 if $(x < 7)$ and $(\text{imm16} < 33)$, then cmpnei16 rx, imm5; else cmpnei32 rx, imm16;
说明	将 RX 的值减去零扩展至 32 位的 16 位立即数的值，结果与 0 作比较，并对 C 位进行更新。如果 RX 不等于零扩展后的 IMM16，即减法结果不等于 0，则设置条件位 C；否则，清除条件位 C。
影响标志位	根据比较结果设置条件位 C
限制	立即数的范围为 0x0-0xFFFF。
异常	无

16 位指令	
操作	RX 与立即数作比较。 If $RX \neq \text{zero\_extend}(\text{IMM5})$, then $C \leftarrow 1$; else $C \leftarrow 0$;
语法	cmpnei16 rx, imm5
说明	将 RX 的值减去零扩展至 32 位的 5 位立即数的值，结果与 0 作比较，并对 C 位进行更新。如果 RX 不等于零扩展后的 IMM5，即减法结果不等于 0，则设置条件位 C；否则，清除条件位 C。
影响标志位	根据比较结果设置条件位 C
限制	寄存器的范围为 r0-r7； 立即数的范围为 0-31。
异常	无

16 位指令格式：

15	14	11	10	8	7	5	4	0
0	0111			RX	010		IMM5	

32 位指令	
操作	RX 与立即数作比较。 If $RX \neq \text{zero\_extend}(\text{imm16})$, then $C \leftarrow 1$; else $C \leftarrow 0$;
语法	cmpnei rx, imm16
说明	将 RX 的值减去零扩展至 32 位的 16 位立即数的值，结果与 0 作比较，并对 C 位进行更新。如果 RX 不等于零扩展后的 IMM16，即减法结果不等于 0，则设置条件位 C；否则，清除条件位 C。
影响标志位	根据比较结果设置条件位 C
限制	立即数的范围为 0x0-0xFFFF。
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	0
1	11010			11010		RX	IMM16	

14.41 DCACHE——DCACHE 操作指令

统一化指令	
语 法	dcache.iall dcache.iva rx dcache.isw rx dcache.call dcache.cva rx dcache.cval1 rx dcache.csw rx dcache.ciall dcache.civa rx dcache.cisw rx
操 作	删除所有 dcache 表项， 删除虚拟地址对应的 dcache 和 L2Cache 表项， 删除指定 SET/WAY/LEVEL 的 dcache 表项， CLEAR 所有 dcache 表项， CLEAR 虚拟地址对应的 dcache 和 l2cache 表项， CLEAR 虚拟地址对应的 dcache 表项 CLEAR 指定 SET/WAY/LEVEL 的 dcache 表项， CLEAR 并删除所有 dcache 表项， CLEAR 并删除虚拟地址对应的 dcache 和 l2cache 表项， CLEAR 并删除指定 SET/WAY/LEVEL 的 dcache 表项。 其中，当操作 VA 时，RX[31:0] 表征 VA[31:0] 当操作为 set/way 时，RX[31] 表征 way，rx[L:6] 表征 set，D-Cache 32K 时，L=13；64K 时，L=14
说 明	仅存在 32 位指令。 dcache32.iall, dcache32.iva rx, dcache32.isw rx, dcache32.call, dcache32.cva rx, dcache32.csw rx, dcache32.ciall, dcache32.civa rx, dcache32.cisw rx
说 明	dcache.iva、dcache.cva、dcache.cval1 和 dcache.civa 指令根据虚拟地址的共享属性判断是否广播给其他 CPU 的 dcache。
影 响 标 志 位	无
异 常	TLB 不可恢复异常、TLB 失配异常、TLB 无效异常

指令格式：

dcache.iall

31	30	26	25	21	20	16	15	10	9	5	4	0
1		10000		01000		00000		100101		00001		00000

dcache.iva rx

31	30	26	25	21	20	16	15	10	9	5	4	0
1		10000		01011		RX		100101		00001		00000

dcache.isw rx

31	30	26	25	21	20	16	15	10	9	5	4	0
1		10000		01010		RX		100101		00001		00000

dcache.call

31	30	26	25	21	20	16	15	10	9	5	4	0
1		10000		00100		00000		100101		00001		00000

dcache.cva rx

31	30	26	25	21	20	16	15	10	9	5	4	0
1		10000		00111		RX		100101		00001		00000

dcache.cval1 rx

31	30	26	25	21	20	16	15	10	9	5	4	0
1		10000		10111		RX		100101		00001		00000

dcache.csw rx

31	30	26	25	21	20	16	15	10	9	5	4	0
1		10000		00110		RX		100101		00001		00000

dcache.ciall

31	30	26	25	21	20	16	15	10	9	5	4	0
1		10000		01100		00000		100101		00001		00000

dcache.civa rx

31	30	26	25	21	20	16	15	10	9	5	4	0
1		10000		01111		RX		100101		00001		00000

dcache.cisw rx

31	30	26	25	21	20	16	15	10	9	5	4	0
1		10000		01110		RX		100101		00001		00000

14.42 DECF——C 为 0 立即数减法指令

统一化指令	
语法	decf rz, rx, imm5
操作	if C==0, then RZ ← RX - zero_extend(IMM5); else RZ ← RZ;
编译结果	仅存在 32 位指令 decf32 rz, rx, imm5
说明	如果条件位 C 为 0，将 5 位立即数零扩展至 32 位，用 RX 的值减去该 32 位值，把结果存在 RZ；否则，RZ 与 RX 的值不变。
影响标志位	无影响
限制	立即数的范围为 0-31。
异常	无

32 位指令	
操作	if C==0, then RZ ← RX - zero_extend(IMM5); else RZ ← RZ;
语法	decf32 rz, rx, imm5
说明	如果条件位 C 为 0，将 5 位立即数零扩展至 32 位，用 RX 的值减去该 32 位值，把结果存在 RZ；否则，RZ 与 RX 的值不变。
影响标志位	无影响
限制	立即数的范围为 0-31。
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001		RZ		RX		000011		00100		IMM5	

14.43 DECGT——减法大于零置 C 位指令

统一化指令	
语法	decdgt rz, rx, imm5
操作	$RZ \leftarrow RX - \text{zero\_extend}(IMM5);$ If $RZ > 0$, then $C \leftarrow 1;$ else $C \leftarrow 0;$
编译结果	仅存在 32 位指令 decdgt32 rz, rx, imm5
说明	将 5 位立即数零扩展至 32 位，把 RX 减去该 32 位值的结果存在 RZ。减法结果被认为是补码形式的有符号数。如果结果大于 0，设置条件位 C；否则清除条件位 C。
影响标志位	如果减法结果大于 0，设置条件位 C；否则清除条件位 C。
限制	立即数的范围为 0-31。
异常	无

32 位指令	
操作	$RZ \leftarrow RX - \text{zero\_extend}(IMM5);$ If $RZ > 0$, then $C \leftarrow 1;$ else $C \leftarrow 0;$
语法	decdgt32 rz, rx, imm5
说明	将 5 位立即数零扩展至 32 位，把 RX 减去该 32 位值的结果存在 RZ。减法结果被认为是补码形式的有符号数。如果结果大于 0，设置条件位 C；否则清除条件位 C。
影响标志位	如果减法结果大于 0，设置条件位 C；否则清除条件位 C。
限制	立即数的范围为 0-31。
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001	IMM5		RX		000100		00001			RZ	

14.44 DECLT——减法小于零置 C 位指令

统一化指令	
语法	declt rz, rx, imm5
操作	$RZ \leftarrow RX - \text{zero\_extend}(IMM5);$ If $RZ < 0$, then $C \leftarrow 1;$ else $C \leftarrow 0;$
编译结果	仅存在 32 位指令 declt32 rz, rx, imm5
说明	将 5 位立即数零扩展至 32 位，把 RX 减去该 32 位值的结果存在 RZ。减法结果被认为是补码形式的有符号数。如果结果小于 0，设置条件位 C；否则清除条件位 C。
影响标志位	如果减法结果小于 0，设置条件位 C；否则清除条件位 C。
限制	立即数的范围为 0-31。
异常	无

32 位指令	
操作	$RZ \leftarrow RX - \text{zero\_extend}(IMM5);$ If $RZ < 0$, then $C \leftarrow 1;$ else $C \leftarrow 0;$
语法	declt32 rz, rx, imm5
说明	将 5 位立即数零扩展至 32 位，把 RX 减去该 32 位值的结果存在 RZ。减法结果被认为是补码形式的有符号数。如果结果小于 0，设置条件位 C；否则清除条件位 C。
影响标志位	如果减法结果小于 0，设置条件位 C；否则清除条件位 C。
限制	立即数的范围为 0-31。
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001				IMM5				RX			
								000100				RZ

14.45 DECNE——减法不等于零置 C 位指令

统一化指令	
语法	decne rz, rx, imm5
操作	$RZ \leftarrow RX - \text{zero\_extend}(IMM5);$ If $RZ \neq 0$, then $C \leftarrow 1;$ else $C \leftarrow 0;$
编译结果	仅存在 32 位指令 decne32 rz, rx, imm5
说明	将 5 位立即数零扩展至 32 位，把 RX 减去该 32 位值的结果存在 RZ。如果结果不等于 0，设置条件位 C；否则清除条件位 C。
影响标志位	如果减法结果不等于 0，设置条件位 C；否则清除条件位 C。
限制	立即数的范围为 0-31。
异常	无

32 位指令	
操作	$RZ \leftarrow RX - \text{zero\_extend}(IMM5);$ If $RZ \neq 0$, then $C \leftarrow 1;$ else $C \leftarrow 0;$
语法	decne32 rz, rx, imm5
说明	将 5 位立即数零扩展至 32 位，把 RX 减去该 32 位值的结果存在 RZ。如果结果不等于 0，设置条件位 C；否则清除条件位 C。
影响标志位	如果减法结果不等于 0，设置条件位 C；否则清除条件位 C。
限制	立即数的范围为 0-31。
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001	IMM5		RX		000100		00100			RZ	

14.46 DECT——C 为 1 立即数减法指令

统一化指令	
语法	dect rz, rx, imm5
操作	if C==1, then RZ ← RX - zero_extend(IMM5); else RZ ← RZ;
编译结果	仅存在 32 位指令 dect32 rz, rx, imm5
说明	如果条件位 C 为 1，将 5 位立即数零扩展至 32 位，用 RX 的值减去该 32 位值，把结果存在 RZ；否则，RZ 与 RX 的值不变。
影响标志位	无影响
限制	立即数的范围为 0-31。
异常	无

32 位指令	
操作	if C==1, then RZ ← RX - zero_extend(IMM5); else RZ ← RZ;
语法	dect32 rz, rx, imm5
说明	如果条件位 C 为 1，将 5 位立即数零扩展至 32 位，用 RX 的值减去该 32 位值，把结果存在 RZ；否则，RZ 与 RX 的值不变。
影响标志位	无影响
限制	立即数的范围为 0-31。
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001		RZ		RX		000011		01000		IMM5	

14.47 DIVS——有符号除法指令

统一化指令	
语法	divs rz, rx, ry
操作	有符号除法 $RZ = RX / RY$
编译结果	仅存在 32 位指令 divs32 rz, rx, ry
说明	有符号寄存器除法指令将寄存器 RX 的值除以寄存器 RY 的值，得到的商存放于 RZ 中。RX、RY 和 RZ 的值均被认为是 32 位有符号数。 注意，对于 0x80 000000 除以 0xffffffff 这种情况，结果没有定义。
影响标志位	不影响
异常	除零异常

32 位指令	
操作	有符号除法 $RZ = RX / RY$
语法	divs32 rz, rx, ry
说明	有符号寄存器除法指令将寄存器 RX 的值除以寄存器 RY 的值，得到的商存放于 RZ 中。RX、RY 和 RZ 的值均被认为是 32 位有符号数。 注意，对于 0x80000000 除以 0xffffffff 这种情况，结果没有定义。
影响标志位	不影响
异常	除零异常

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001			RY		RX		100000		00010		RZ

14.48 DIVSL——有符号长除法指令

统一化指令	
语法	divsl rz, rx, ry
操作	有符号除法 $\{Rz+1[31:0],Rz[31:0]\} = \{Rx+1[31:0],Rx[31:0]\} / Ry[31:0]$
编译结果	仅存在 32 位指令 divsl rz, rx, ry
说明	{Rx+1,Rx} 组成一个 64 位的有符号数，除以 RY 的值，除法结果的高 32 位存入 Rz+1，低 32 位存入 Rz。Rx+1、Ry 和 Rz+1 的值均被认为是符号数。 注意，当 0x80000000_00000000 除以 0xFFFFFFFF 时，结果没有定义。
影响标志位	无影响
限制	rz 和 rz+1 不能等于 rx 或 ry
异常	除零异常

32 位指令	
操作	有符号除法 $\{Rz+1[31:0],Rz[31:0]\} = \{Rx+1[31:0],Rx[31:0]\} / Ry[31:0]$
语法	divsl32 rz, rx, ry
说明	{Rx+1,Rx} 组成一个 64 位的有符号数，除以 RY 的值，除法结果的高 32 位存入 Rz+1，低 32 位存入 Rz。Rx+1、Ry 和 Rz+1 的值均被认为是符号数。 注意，当 0x80000000_00000000 除以 0xFFFFFFFF 时，结果没有定义。
影响标志位	无影响
限制	rz 和 rz+1 不能等于 rx 或 ry
异常	除零异常

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11110			RY		RX		111000		10111		RZ

14.49 DIVU——无符号除法指令

统一化指令	
语法	divu rz, rx, ry
操作	无符号除法 $RZ = RX / RY$
说明	仅存在 32 位指令 divu32 rz, rx, ry
说明	无符号寄存器除法指令将寄存器 RX 的值除以寄存器 RY 的值，得到的商存放于 RZ 中。RX、RY 和 RZ 的值均被认为是 32 位无符号数。
影响标志位	不影响
异常	除零异常

32 位指令	
操作	无符号除法 $RZ = RX / RY$
语法	divu32 rz, rx, ry
说明	无符号寄存器除法指令将寄存器 RX 的值除以寄存器 RY 的值，得到的商存放于 RZ 中。RX、RY 和 RZ 的值均被认为是 32 位无符号数。
影响标志位	不影响
异常	除零异常

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001		RY		RX		100000		00001		RZ	

14.50 DIVUL——无符号长除法指令

统一化指令	
语法	divul rz, rx, ry
操作	无符号除法 $\{Rz+1[31:0],Rz[31:0]\} = \{Rx+1[31:0],Rx[31:0]\} / Ry[31:0]$
编译结果	仅存在 32 位指令 divul rz, rx, ry
说明	将 {Rx+1,Rx} 组成一个 64 位的无符号数，除以 RY 的值，除法结果的高 32 位存入 Rz+1，低 32 位存入 Rz。Rx+1、Ry 和 Rz+1 的值均被认为是无符号数。
影响标志位	无影响
限制	rz 和 rz+1 不能等于 rx 或 ry
异常	除零异常

32 位指令	
操作	无符号除法 $\{Rz+1[31:0],Rz[31:0]\} = \{Rx+1[31:0],Rx[31:0]\} / Ry[31:0]$
语法	divul rz, rx, ry
说明	将 {Rx+1,Rx} 组成一个 64 位的无符号数，除以 RY 的值，除法结果的高 32 位存入 Rz+1，低 32 位存入 Rz。Rx+1、Ry 和 Rz+1 的值均被认为是无符号数。
影响标志位	无影响
限制	rz 和 rz+1 不能等于 rx 或 ry
异常	除零异常

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11110		RY		RX		111000		10011		RZ	

14.51 DOZE——进入低功耗睡眠模式指令

统一化指令	
语法	doze
操作	进入低功耗睡眠模式
编译结果	仅存在 32 位指令 doze32
说明	此指令使处理器进入低功耗睡眠模式，并等待一个中断来退出这个模式。此时，CPU 时钟停止，相应的外围设备也被停止。
影响标志位	不影响
异常	特权违反异常

32 位指令	
操作	进入低功耗睡眠模式
语法	doze32
属性：	特权指令
说明	此指令使处理器进入低功耗睡眠模式，并等待一个中断来退出这个模式。此时，CPU 时钟停止，相应的外围设备也被停止。
影响标志位	不影响
异常	特权违反异常

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10000	00000	00000	010100	00001	00000						

14.52 FF0——快速找 0 指令

统一化指令	
语法	ff0 rz, rx
操作	RZ ← find_first_0(RX);
编译结果	ff0.32 rz, rx
说明	查找 RX 第一个为 0 的位，并把查找结果返回到 RZ。查找顺序是从 RX 的最高位扫描到最低位。如果 RX 的最高位（RX[31]）为 0，返回 RZ 的值为 0。如果在 RX 没有为 0 的位，返回 RZ 的值为 32。
影响标志位	无影响
异常	无

32 位指令	
操作	RZ ← find_first_0(RX);
语法	ff0.32 rz, rx
说明	查找 RX 第一个为 0 的位，并把查找结果返回到 RZ。查找顺序是从 RX 的最高位扫描到最低位。如果 RX 的最高位（RX[31]）为 0，返回 RZ 的值为 0。如果在 RX 没有为 0 的位，返回 RZ 的值为 32。
影响标志位	无影响
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001	00000		RX		011111		00001			RZ	

14.53 FF1——快速找 1 指令

统一化指令	
语法	ff1 rz, rx
操作	RZ ← find_first_1(RX);
编译结果	ff1.32 rz, rx
说明	查找 RX 第一个为 1 的位，并把查找结果返回到 RZ。查找顺序是从 RX 的最高位扫描到最低位。如果 RX 的最高位（RX[31]）为 1，返回 RZ 的值为 0。如果在 RX 没有为 1 的位，返回 RZ 的值为 32。
影响标志位	无影响
异常	无

32 位指令	
操作	RZ ← find_first_1(RX);
语法	ff1.32 rz, rx
说明	查找 RX 第一个为 1 的位，并把查找结果返回到 RZ。查找顺序是从 RX 的最高位扫描到最低位。如果 RX 的最高位（RX[31]）为 1，返回 RZ 的值为 0。如果在 RX 没有为 1 的位，返回 RZ 的值为 32。
影响标志位	无影响
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001	00000		RX		011111		00010			RZ	

14.54 FABS.T——浮点绝对值

统一化指令	
语法	fabs.T vrz, vrx (T=32/64)
操作	vrz= vrx
编译结果	仅存在 32 位指令。 fabs.T vrz, vrx
数据类型	单精度/双精度
说明	将 vrx 中的浮点数取绝对值，保存至 vrz 中。
影响标志位	无
限制	寄存器范围为 vr0-vr31。
异常	无

32 位指令	
操作	vrz= vrx
语法	fabs.T vrz, vrx
数据类型	单精度/双精度
说明	将 vrx 中的浮点数取绝对值，保存至 vrz 中。
影响标志位	无
限制	寄存器范围为 vr0-vr31。
异常	无

32 位指令格式：

fabs.32											
31	26	25	24	21	20	16	15	5	4	0	
11110		1	0000		VRX		00000000110			VRZ	

fabs.64											
31	26	25	24	21	20	16	15	5	4	0	
111101		0	0000		VRX		00001000110			VRZ	

14.55 FADD.T——浮点加法指令

统一化指令	
语法	fadd.T vrz, vrx,vry (T=32/64)
操作	vrz= vrx+vry
编译结果	仅存在 32 位指令。 fadd.T vrz, vrx,vry
数据类型	单精度/双精度
说明	将 vrx 和 vry 中的浮点数相加，保存至 vrz 中。
影响标志位	无
限制	寄存器范围为 vr0-vr31。
异常	ID,IO,UF,OF,IX

32 位指令	
操作	vrz= vrx+vry
语法	fadd.T vrz, vrx,vry
数据类型	单精度/双精度
说明	将 vrx 和 vry 中的浮点数相加，保存至 vrz 中。
影响标志位	无
限制	寄存器范围为 vr0-vr31。
异常	ID,IO,UF,OF,IX

32 位指令格式：

fadd.32									
31	26	25	21	20	16	15	5	4	0
111101		VRY		VRX		0000000000		VRZ	

fadd.64									
31	26	25	21	20	16	15	5	4	0
111101		VRY		VRX		0000100000		VRZ	

14.56 FCMPHS.T——浮点大于等于比较

统一化指令	
语法	fcmphs.T vrx (T=32/64)
操作	if vrx>=vry, then Set C; Else Clear C;
编译结果	仅存在 32 位指令 fcmphsz.T vrx
数据类型	单精度/双精度
说明	比较 vrx 和 vry，若 vrx>=vry 则条件位 C 置位，否则清零。
影响标志位	根据比较结果设置条件位 C。
限制	寄存器范围为 vr0-vr31。
异常	IO、ID

32 位指令	
操作	if vrx>=vry, then Set C; else Clear C;
语法	fcmphsz.T vrx
数据类型	单精度/双精度
说明	比较 vrx 和 vry，若 vrx>=vry 则条件位 C 置位，否则清零。
影响标志位	根据比较结果设置条件位 C。
限制	寄存器范围为 vr0-vr31。
异常	IO、ID

32 位指令格式：

fcmphs.32																
31	30	26	25	21	20	16	15	10	9	5	4	0				
1	11101			VRY			VRX			000000			01100		00000	
fcmphs.64																
31	30	26	25	21	20	16	15	10	9	5	4	0				
1	11101			VRY			VRX			000010			01100		00000	

14.57 FCMPSZ.T——浮点大于等于零比较指令

统一化指令	
语法	fcmphsz.T vrx (T=32/64)
操作	if vrx>=0, then Set C; Else Clear C;
编译结果	仅存在 32 位指令 fcmphsz.T vrx
数据类型	单精度/双精度
说明	若 vrx>0 则条件位 C 置位，否则清零。
影响标志位	根据比较结果设置条件位 C。
限制	寄存器范围为 vr0-vr31。
异常	IO、ID

32 位指令	
操作	if vrx>=0, then Set C; else Clear C;
语法	fcmphsz.T vrx
数据类型	单精度/双精度
说明	若 vrx>=0 则条件位 C 置位，否则清零。
影响标志位	根据比较结果设置条件位 C。
限制	寄存器范围为 vr0-vr31。
异常	IO、ID

32 位指令格式：

fcmphsz.32																								
31	30	26			25	21			20	16			15	10			9	5		4	0			
1	11101				00000				VRX				000000				01000				00000			
fcmphsz.64																								
31	30	26			25	21			20	16			15	10			9	5		4	0			
1	11101				00000				VRX				000010				01000				00000			

14.58 FCMPHZ.T——浮点大于零比较指令

统一化指令	
语法	fcmphz.T vrx (T=32/64)
操作	if vrx>0, then Set C; else Clear C;
编译结果	仅存在 32 位指令 fcmphz.T vrx, vry
数据类型	单精度/双精度
说明	若 vrx>0 则条件位 C 置位，否则清零。
影响标志位	根据比较结果设置条件位 C。
限制	寄存器范围为 vr0-vr31。
异常	IO、ID

32 位指令	
操作	if vrx>0, then Set C; else Clear C;
语法	fcmphz.T vrx,vry
操作	单精度/双精度
说明	若 vrx>0 则条件位 C 置位，否则清零。
影响标志位	根据比较结果设置条件位 C。
限制	寄存器范围为 vr0-vr31。
异常	IO、ID

32 位指令格式：

fcmphz.32																
31	30	26	25	21	20	16	15	10	9	5	4	0				
1	11101			00000			VRX			000001			01010		00000	
fcmphz.64																
31	30	26	25	21	20	16	15	10	9	5	4	0				
1	11101			00000			VRX			000011			01010		00000	

14.59 FCMPLSZ.T——浮点小于等于零比较

统一化指令	
语法	fcmplsz.T vrx (T=32/64)
操作	if vrx<=0, then Set C; Else Clear C;
编译结果	仅存在 32 位指令 fcmplsz.T vrx
数据类型	单精度/双精度
说明	若 vrx 小于等于零，则条件位 C 置位，否则清零
影响标志位	根据比较结果设置条件位 C。
限制	寄存器范围为 vr0-vr31。
异常	IO、ID

32 位指令	
操作	if vrx<=0, then Set C; else Clear C;
语法	fcmplsz.T vrx
数据类型	单精度/双精度
说明	若 vrx 小于等于零，则条件位 C 置位，否则清零。
影响标志位	根据比较结果设置条件位 C。
限制	寄存器范围为 vr0-vr31。
异常	IO、ID

32 位指令格式：

fcmplsz.32																
31	30	26	25	21	20	16	15	10	9	5	4	0				
1	11101			00000			VRX			000001			01011		00000	
fcmplsz.64																
31	30	26	25	21	20	16	15	10	9	5	4	0				
1	11101			00000			VRX			000011			01011		00000	

14.60 FCMPLT.T——浮点小于比较

统一化指令	
语法	fcmplt.T vr _x ,vr _y (T=32/64)
操作	if vr _x <vr _y , then Set C; Else Clear C;
编译结果	仅存在 32 位指令 fcmplt.T vr _x ,vr _y
数据类型	单精度/双精度
说明	比较 vr _x 和 vr _y ，若 vr _x 小于 vr _y ，则条件位 C 置位，否则清零。
影响标志位	根据比较结果设置条件位 C。
限制	寄存器范围为 vr0-vr31。
异常	IO、ID

32 位指令	
操作	if vr _x <vr _y , then Set C; else Clear C;
语法	fcmplt.T vr _x .vr _y
数据类型	单精度/双精度
说明	比较 vr _x 和 vr _y ，若 vr _x 小于 vr _y ，则条件位 C 置位，否则清零。
影响标志位	根据比较结果设置条件位 C。
限制	寄存器范围为 vr0-vr31。
异常	IO、ID

32 位指令格式：

fcmplt.32

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11101	VR _Y	VR _X	000000	01101	00000						

fcmplt.64

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11101	VR _Y			VR _X		000010		01101		00000	

14.61 FCMPLTZ.T——浮点小于零比较指令

统一化指令	
语法	fcmpltz.T vrx (T=32/64)
操作	if vrx<0 then Set C; Else Clear C;
编译结果	仅存在 32 位指令 fcmpltz.T vrx
数据类型	单精度/双精度
说明	若 vrx 小于零，则条件位 C 置位，否则清零。
影响标志位	根据比较结果设置条件位 C。
限制	寄存器范围为 vr0-vr31。
异常	IO、ID

32 位指令	
操作	if vrx<0, then Set C; else Clear C;
语法	fcmpltz.T vrx
数据类型	单精度/双精度
说明	若 vrx 小于零，则条件位 C 置位，否则清零。
影响标志位	根据比较结果设置条件位 C。
限制	寄存器范围为 vr0-vr31。
异常	IO、ID

32 位指令格式：

fcmpltz.32																
31	30	26	25	21	20	16	15	10	9	5	4	0				
1	11101			000000			VRX			000000			01001		00000	
fcmpltz.64																
31	30	26	25	21	20	16	15	10	9	5	4	0				
1	11101			000000			VRX			000010			01001		00000	

14.62 FCOMPNE.T——浮点不等于比较指令

+=====+									
统一化指令 +=====+									
语法 fcmpne.T vr _x ,vr _y (T=32/64) +-----+-----									
-+ 操作 if vr _x != vr _y , then Set C; Else Clear C; +-----+-----									
-----+ 编译结果 仅存在 32 位指令 fcmpne.T vr _x ,vr _y									
+-----+-----+ 数据类型 单精度/双精度 +-----									
-+-----+ 说明 比较 vr _x 和 vr _y ，若 vr _x 不等于 vr _y ，则条件位 C									
置位，否则清零。 +-----+-----+ 影响标志位 根据比较									
结果设置条件位 C。 +-----+-----+ 限制 寄存器范围为									
vr ₀ -vr ₃₁ 。 +-----+-----+ 异常 IO、ID +-----+-----									
-----+									

32 位指令	
操作	if vr _x != vr _y , then Set C; else Clear C;
语法	fcmpne.T vr _x .vr _y
数据类型	单精度/双精度
说明	比较 vr _x 和 vr _y ，若 vr _x 不等于 vr _y ，则条件位 C 置位，否则清零。
影响标志位	根据比较结果设置条件位 C。
限制	寄存器范围为 vr ₀ -vr ₃₁ 。
异常	IO、ID

32 位指令格式：

fcmpne.32													
31	30	26	25	21	20	16	15	10	9	5	4	0	
1	11101			VRY			VRX			000000		01110	00000
fcmpne.64													
31	30	26	25	21	20	16	15	10	9	5	4	0	
1	11101			VRY			VRX			000010		01110	00000

14.63 FCMPEZ.T——浮点不等于零比较指令

统一化指令	
语法	fcmpnez.T vrx (T=32/64)
操作	if vrx != 0, then Set C; Else Clear C;
编译结果	仅存在 32 位指令 fcmpnez.T vrx
数据类型	单精度/双精度
说明	若 vrx 不等于零，则条件位 C 置位，否则清零。
影响标志位	根据比较结果设置条件位 C。
限制	寄存器范围为 vr0-vr31
异常	IO、ID

32 位指令	
操作	if vrx != 0, then Set C; else Clear C;
语法	fcmpnez.T vrx
数据类型	单精度/双精度
说明	若 vrx 不等于零，则条件位 C 置位，否则清零。
影响标志位	根据比较结果设置条件位 C。
限制	寄存器范围为 vr0-vr31
异常	IO、ID

32 位指令格式：

fcmpnez.32																
31	30	26	25	21	20	16	15	10	9	5	4	0				
1	11101			00000			VRX			000000			01010		00000	
fcmpnez.64																
31	30	26	25	21	20	16	15	10	9	5	4	0				
1	11101			00000			VRX			000010			01010		00000	

14.64 FCMPUO.T——浮点双操作数非数判断指令

统一化指令	
语法	fcmpuo.T vrx, vry (T=32/64)
操作	If vrx==NaN vry==NaN set C; else clear C;
编译结果	仅存在 32 位指令。 fcmpuo.T vrx, vry
数据类型	单精度/双精度
说明	若 vrx 非数或者 vry 非数，则条件位 C 置位，否则清零。
影响标志位	根据比较结果设置条件位 C。
限制	寄存器范围为 vr0-vr31。
异常	IO、ID

32 位指令	
操作	If vrx == NaN vry == NaN set C; else clear C。
语法	fcmpuo.T vrx, vry
数据类型	单精度/双精度
说明	若 vrx 非数或者 vry 非数，则条件位 C 置位，否则清零。
影响标志位	根据比较结果设置条件位 C。
限制	寄存器范围为 vr0-vr31。
异常	IO、ID

32 位指令格式：

fcmpuo.32																
31	30	26	25	21	20	16	15	10	9	5	4	0				
1	11101				VRY				VRX				000000			
fcmpuo.64																
31	30	26	25	21	20	16	15	10	9	5	4	0				
1	11101				VRY				VRX				000010			

14.65 FCMPUOZ.T——浮点单操作数非数判断指令

统一化指令	
语法	fcmpuoz.T vrx (T=32/64)
操作	If vrx==NaN set C; else clear C。
编译结果	仅存在 32 位指令。 fcmpuoz.T vrx
数据类型	单精度/双精度
说明	若 vrx 非数，则条件位 C 置位，否则清零。
影响标志位	根据比较结果设置条件位 C。
限制	寄存器范围为 vr0-vr31。
异常	IO、ID

32 位指令	
操作	If vrx==NaN set C; else clear C。
语法	fcmpuoz.T vrx
数据类型	单精度/双精度
说明	若 vrx 非数，则条件位 C 置位，否则清零。
影响标志位	根据比较结果设置条件位 C。
限制	寄存器范围为 vr0-vr31。
异常	IO、ID

32 位指令格式：

fcmpuoz.32													
31	30	26	25	21	20	16	15	10	9	5	4		0
1		11101		00000		VRX		000000		01011		00000	

fcmpuoz.64													
31	30	26	25	21	20	16	15	10	9	5	4		0
1		11101		00000		VRX		000010		01011		00000	

14.66 FDIV.T——浮点除法指令

统一化指令	
语法	fdiv.T vrz, vrx,vry (T=32/64)
操作	vrz= vrx/vry
编译结果	仅存在 32 位指令。 fdiv.T vrz, vrx,vry
数据类型	单精度/双精度
说明	将 vrx 和 vry 中的单精度浮点数相除，保存至 vrz 中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr31。
异常	IO、ID、UF、OF、IX、DZ

32 位指令	
操作	vrz= vrx/vry
语法	fdiv.T vrz, vrx,vry
数据类型	单精度/双精度
说明	将 vrx 和 vry 中的单精度浮点数相除，保存至 vrz 中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr31。
异常	IO、ID、UF、OF、IX、DZ

32 位指令格式：

fdiv.32

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11101	VRY				VRX		000000		11000		VRZ

fdiv.64

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11101	VRY	VRX	000010	11000	VRZ						

14.67 FDTOS——双精度浮点单精度浮点转换指令

统一化指令	
语法	fdtos vrz, vrx
操作	vrz= (float)vrx
编译结果	仅存在 32 位指令。 fdtos vrz, vrx
数据类型	双精度浮点数
说明	将 vrx 中的双精度浮点数转换成单精度浮点数，保存至 vrz 中。
影响标志位	无
限制	寄存器范围为 vr0-vr31。
异常	ID,IO,UF,OF,IX

32 位指令	
操作	vrz= (float)vrx
语法	fdtos vrz, vrx
数据类型	双精度浮点数
说明	将 vrx 中的双精度浮点数转换成单精度浮点数，保存至 vrz 中。
影响标志位	无
限制	寄存器范围为 vr0-vr31。
异常	ID,IO,UF,OF,IX

32 位指令格式：

31	26	25	24	21	20	16	15	5	4	0
111101			0	0000		VRX		00011010110		VRZ

14.68 FFMULA.T——浮点融合乘累加指令

统一化指令	
语法	ffmula.t vrz, vrx, vry (t=32/64)
操作	$vrz = vrz + vrx * vry$
编译结果	仅存在 32 位指令 ffmula.t vrz, vrx, vry
数据类型	单精度/双精度
说明	将 vrx 中的单精度浮点数与 vry 中的单精度浮点数相乘得到乘积，保留乘积所有精度，与 vrz 中的单精度浮点数相加，将结果写回 vrz 中
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令	
操作	$vrz = vrz + vrx * vry$
数据类型	单精度/双精度
语法	ffmula.t vrz, vrx, vry
说明	将 vrx 中的单精度浮点数与 vry 中的单精度浮点数相乘得到乘积，保留乘积所有精度，与 vrz 中的单精度浮点数相加，将结果写回 vrz 中
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令格式：

ffmula.32															
31	30	26	25	21	20	16	15	10	9	5	4	0			
1	11101			VRY			VRX			000001			10000		VRZ

ffmula.64															
31	30	26	25	21	20	16	15	10	9	5	4	0			
1	11101			VRY			VRX			000011			10000		VRZ

14.69 FFMULS.T——浮点融合乘取负累加指令

统一化指令	
语法	ffmul.s t vrz, vrx, vry (t=32/64)
操作	$vrz = vrz - vrx * vry$
编译结果	仅存在 32 位指令 ffmul.s t vrz, vrx, vry
数据类型	单精度/双精度
说明	将 vrx 中的单精度浮点数与 vry 中的单精度浮点数相乘得到乘积，保留乘积全部精度并进行取负操作后，与 vrz 中的单精度浮点数相加，将结果写回 vrz 中
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令	
操作	$vrz = vrz - vrx * vry$
数据类型	单精度/双精度
语法	ffmul.s t vrz, vrx, vry
说明	将 vrx 中的单精度浮点数与 vry 中的单精度浮点数相乘得到乘积，保留乘积全部精度并进行取负操作后，与 vrz 中的单精度浮点数相加，将结果写回 vrz 中
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令格式：

ffmul.s.32															
31	30	26	25	21	20	16	15	10	9	5	4	0			
1	11101			VRX			000001			10001		VRZ			

ffmul.s.64															
31	30	26	25	21	20	16	15	10	9	5	4	0			
1	11101			VRX			000011			10001		VRZ			

14.70 FFMULA.T——浮点融合乘取负累减指令

统一化指令	
语法	ffnmula.t vrz, vrx, vry (t=32/64)
操作	vrz = -vrz - vrx * vry
编译结果	仅存在 32 位指令 ffnmula.t vrz, vrx, vry
数据类型:	单精度/双精度
说明	将 vrx 中的单精度浮点数与 vry 中的单精度浮点数相乘得到乘积，保留乘积全部精度并进行取负操作后，与 vrz 中的单精度浮点数相减，将结果写回 vrz 中
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令	
操作	vrz = -vrz - vrx * vry
数据类型	单精度/双精度
语法	ffnmula.t vrz, vrx, vry
说明	将 vrx 中的单精度浮点数与 vry 中的单精度浮点数相乘得到乘积，保留乘积全部精度并进行取负操作后，与 vrz 中的单精度浮点数相减，将结果写回 vrz 中
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令格式：

ffnmula.32															
31	30	26	25	21	20	16	15	10	9	5	4	0			
1	11101			VRY			VRX			000001			10010		VRZ

ffnmula.64															
31	30	26	25	21	20	16	15	10	9	5	4	0			
1	11101			VRY			VRX			000011			10010		VRZ

14.71 FFMULS.T——浮点融合乘累减指令

统一化指令	
语法	ffnmuls.t vrz, vrx, vry (t=32/64)
操作	$vrz = -vrz + vrx * vry$
编译结果	仅存在 32 位指令 ffnmuls.t vrz, vrx, vry
数据类型	单精度/双精度
说明	将 vrx 中的单精度浮点数与 vry 中的单精度浮点数相乘得到乘积，保留乘积全部精度，与 vrz 中的单精度浮点数相减，将结果写回 vrz 中
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令	
操作	$vrz = -vrz + vrx * vry$
数据类型	单精度/双精度
语法	ffnmuls.t vrz, vrx, vry
说明	将 vrx 中的单精度浮点数与 vry 中的单精度浮点数相乘得到乘积，保留乘积全部精度，与 vrz 中的单精度浮点数相减，将结果写回 vrz 中
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令格式：

ffnmuls.32															
31	30	26	25	21	20	16	15	10	9	5	4	0			
1	11101			VRY			VRX			000001			10011		VRZ

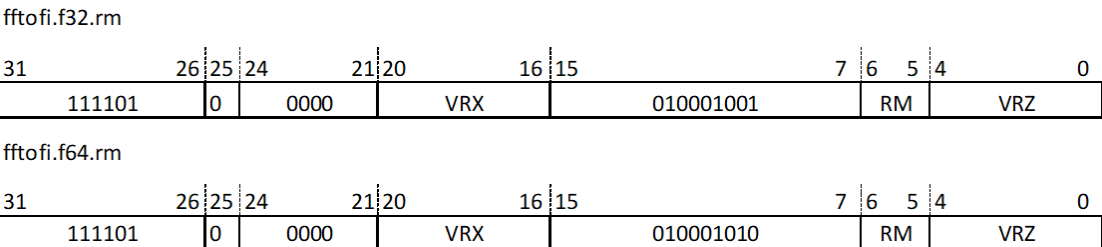
ffnmuls.64															
31	30	26	25	21	20	16	15	10	9	5	4	0			
1	11101			VRY			VRX			000011			10011		VRZ

14.72 FFTOFL.T.RM——带舍入模式的浮点转换为整型浮点指令

统一化指令	
语法	fftofi.T.rm vrz, vrx (T=f32/f64)
操作	vrz= (int_float)vrx
编译结果	仅存在 32 位指令。 fftofi.T.rm vrz, vrx
数据类型	单精度/双精度
说明	将 vrx 中的浮点舍入为整数结果再用浮点数表示，保存至 vrz 中。 rm 为舍入方式： 2’ b00: 就近舍入模式，对应的汇编指令为 fftoi.T.rm 2’ b01: 向 0 舍入模式，对应的汇编指令为 fftoi.T.rz 2’ b10: 向正无穷舍入模式，对应的汇编指令为 fftoi.T.rpi 2’ b11: 向负无穷舍入模式，对应的汇编指令为 fftoi.T.rni
影响标志位	无
限制	寄存器范围为 vr0-vr31。
异常	ID,IO,IX

32 位指令	
操作	vrz= (float16)vrx
语法	fftofi.T.rm vrz, vrx
数据类型	单精度/双精度
说明	将 vrx 中的浮点舍入为整数结果再用浮点数表示，保存至 vrz 中。 rm 为舍入方式： 2’ b00: 就近舍入模式，对应的汇编指令为 fftoi.T.rn 2’ b01: 向 0 舍入模式，对应的汇编指令为 fftoi.T.rz 2’ b10: 向正无穷舍入模式，对应的汇编指令为 fftoi.T.rpi 2’ b11: 向负无穷舍入模式，对应的汇编指令为 fftoi.T.rni
影响标志位	无
限制	寄存器范围为 vr0-vr31。
异常	ID,IO,IX

32 位指令格式：



14.73 FFTOI.T1.T2——浮点整数转换指令

统一化指令	
语法	fftoi.T1.T2 vrz, vrx (T1=f32/f64) (T2=u32/s32)
操作	vrz= (int)vrx
编译结果	仅存在 32 位指令。 fftoi.T1.T2 vrz, vrx
数据类型	单精度/双精度
说明	将 vrx 中的浮点数转换成整数，保存至 vrz 中。
影响标志位	无
限制	寄存器范围为 vr0-vr31。
异常	ID,IO,IX

32 位指令	
操作	vrz= (int)vrx
语法	fftoi.T1.T2 vrz, vrx
数据类型	单精度/双精度
说明	将 vrx 中的浮点数转换成整数，保存至 vrz 中。
影响标志位	无
限制	寄存器范围为 vr0-vr31。
异常	ID,IO,IX

32 位指令格式：

fftoi.f32.s32

31	26	25	24	21	20	16	15	5	4	0
111101	0	0000		VRX		01000011011			VRZ	

fftoi.f32.u32

31	26	25	24	21	20	16	15	5	4	0
111101	0	0000		VRX		01000011010			VRZ	

fftoi.f64.s32

31	26	25	24	21	20	16	15	5	4	0
111101	0	0000		VRX		01000011101			VRZ	

fftoi.f64.u32

31	26	25	24	21	20	16	15	5	4	0
111101	0	0000		VRX		01000011100			VRZ	

14.74 FFTOI.T1.T2.RM——带舍入模式的浮点整数转换指令

统一化指令	
语 法	fftoi.T1.T2.rm vrz, vrx (T1=f32/f64) (T2=u32/s32)
操 作	vrz= (int)vrx
编 译 结 果	仅存在 32 位指令。 fftoi.T1.T2 vrz, vrx
数 据 类 型	单精度/双精度
说 明	将 vrx 中的浮点数转换成整数，保存至 vrz 中。 rm 为舍入方式： 2’ b00: 就近舍入模式，对应的汇编指令为 fftoi.T1.T2.rm 2’ b01: 向 0 舍入模式，对应的汇编指令为 fftoi.T1.T2.rz 2’ b10: 向正无穷舍入模式，对应的汇编指令为 fftoi.T1.T2.rpi 2’ b11: 向负无穷舍入模式，对应的汇编指令为 fftoi.T1.T2.rni
影 响 标 志 位	无
限 制	寄存器范围为 vr0-vr31。
异 常	ID,IO,IX

32 位指令	
操作	vrz = (int)vrx
语法	fftoi.T1.T2 vrz, vrx
数据类型	单精度/双精度
说明	将 vrx 中的浮点数转换成整数，保存至 vrz 中。 rm 为舍入方式： 2' b00: 就近舍入模式，对应的汇编指令为 fftoi.T1.T2.rn 2' b01: 向 0 舍入模式，对应的汇编指令为 fftoi.T1.T2.rz 2' b10: 向正无穷舍入模式，对应的汇编指令为 fftoi.T1.T2.rpi 2' b11: 向负无穷舍入模式，对应的汇编指令为 fftoi.T1.T2.rni
影响标志位	无
限制	寄存器范围为 vr0-vr31。
异常	ID,IO,IX

32 位指令格式：

31	26	25	24	21	20	16	15	7	6	5	4	0
111101	0		0000		VRX		000110000		RM		VRZ	
fftoi.f32.u32												
31	26	25	24	21	20	16	15	7	6	5	4	0
111101	0		0000		VRX		000110001		RM		VRZ	
fftoi.f64.s32												
31	26	25	24	21	20	16	15	7	6	5	4	0
111101	0		0000		VRX		000110010		RM		VRZ	
fftoi.f64.u32												
31	26	25	24	21	20	16	15	7	6	5	4	0
111101	0		0000		VRX		000110011		RM		VRZ	

14.75 FFTOX.T1.T2——浮点定点转换指令

统一化指令	
语 法	fftox.T1.T2 vrz, vrx (T1=f32/f64) (T2=u32/s32)
操 作	vrz= (fx)vrx
编 译 结 果	仅存在 32 位指令。 fftox.T1.T2 vrz, vrx
数 据 类 型	单精度/双精度
说 明	将 vrx 中的浮点数转换成定点数，保存至 vrz 中。 FCR[20:16].frpos[4:0] 指定定点数的小数点位置。 16bit 定点数采用 frpos[3:0] 指示 1~16 位的小数部分。frpos[3:0]=0 时，说明有 1bit 小数部分。 frpos[3:0]=15 时，说明有 16bit 小数部分。 32bit 定点数采用 frpos[4:0] 指示 1~32 位的小数部分。frpos[4:0]=0 时，说明有 1bit 小数部分。 frpos[4:0]=31 时，说明有 32bit 小数部分。
影 响 标 志 位	无
限 制	寄存器范围为 vr0-vr31。
异 常	ID,IO,IX

32 位指令	
操作	$vrz = (fx)vrz$
语法	<code>fftox.T1.T2 vrz, vrz</code>
数据类型	单精度/双精度
说明	将 <code>vrz</code> 中的浮点数转换成定点数，保存至 <code>vrz</code> 中。 FCR[20:16].frpos[4:0] 指定定点数的小数点位置。 16bit 定点数采用 frpos[3:0] 指示 1~16 位的小数部分。frpos[3:0]=0 时，说明有 1bit 小数部分。 frpos[3:0]=15 时，说明有 16bit 小数部分。 32bit 定点数采用 frpos[4:0] 指示 1~32 位的小数部分。frpos[4:0]=0 时，说明有 1bit 小数部分。 frpos[4:0]=31 时，说明有 32bit 小数部分。
影响标志位	无
限制	寄存器范围为 <code>vr0-vr31</code> 。
异常	ID,IO,IX

32 位指令格式：

fftox.f32.s32									
31	26	25	21	20	16	15	5	4	0
111101		00000		VRX		01000001011		VRZ	
fftox.f32.u32									
31	26	25	21	20	16	15	5	4	0
111101		00000		VRX		01000001010		VRZ	
fftox.f64.s32									
31	26	25	21	20	16	15	5	4	0
111101		00000		VRX		01000001101		VRZ	
fftox.f64.u32									
31	26	25	21	20	16	15	5	4	0
111101		00000		VRX		01000001100		VRZ	

14.76 FITOF.T1.T2——整数浮点转换指令

统一化指令	
语法	fitof.T1.T2 vrz, vrx (T1=s32/u32) (T2=f32/f64)
操作	vrz= (float32/double)vrx
编译结果	仅存在 32 位指令。 fitof.T1.T2 vrz, vrx
数据类型	32bit 有/无符号整数
说明	将 vrx 中的整数转换成浮点数，保存至 vrz 中。
影响标志位	无
限制	寄存器范围为 vr0-vr31。
异常	IX。

32 位指令	
操作	vrz= (float16/float32/double)vrx
语法	fitof.T1.T2 vrz, vrx
数据类型	32bit 有/无符号整数
说明	将 vrx 中的整数转换成浮点数，保存至 vrz 中。
影响标志位	无
限制	寄存器范围为 vr0-vr31。
异常	IX

32 位指令格式：

fitof.s32.f32										
31	26	25	24	21	20	16	15	5	4	0
111101	0		0000		VRX		01001011011			VRZ
fitof.u32.f32										
31	26	25	24	21	20	16	15	5	4	0
111101	0		0000		VRX		01001011010			VRZ
fitof.s32.f64										
31	26	25	24	21	20	16	15	5	4	0
111101	0		0000		VRX		01001011101			VRZ
fitof.u32.f64										
31	26	25	24	21	20	16	15	5	4	0
111101	0		0000		VRX		01001011100			VRZ

14.77 FINS.32——浮点寄存器间低字高半字传送指令

统一化指令	
语法	fins.32 vrz, vrx
操作	Vz[31:0]={Vx[15:0],Vz[15:0]};
编译结果	仅存在 32 位指令。 fins.32 vrz, vrx
数据类型	单精度浮点数
说明	将浮点寄存器 vrx 的低字低半字搬运到浮点寄存器 vrz 的低字高半字中。低字低半字不变，高字为 0。
影响标志位	无
限制	寄存器范围为 vr0-vr31。
异常	无

32 位指令	
操作	Vz[31:0]={Vx[15:0],Vz[15:0]};
语法	fins.32 vrz, vrx
数据类型	单精度浮点数
说明	将浮点寄存器 vrx 的低字低半字搬运到浮点寄存器 vrz 的低字高半字中。低字低半字不变，高字为 0。。
影响标志位	无
限制	寄存器范围为 vr0-vr31。低字低半字不变，高字为 0。
异常	无

32 位指令格式：

31	26	25	24	21	20	16	15	5	4	0
111101	0	0000	VRX	00000011011	VRZ					

14.78 FLD.T——浮点加载指令

统一化指令	
语 法	fld.32 vrz, (rx, disp) fld.64 vrz, (rx, disp)
操 作	VRZ[len-1:0] ← MEM[RX + zero_extend(offset << 2)]; VRZ[127:len] ← 0;
编 译 结 果	仅存在 32 位指令。 fld.32 vrz, (rx, disp) fld.64 vrz, (rx, disp)
说 明	从存储器加载单、双精度浮点数到矢量寄存器 VRZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移两位的 8 位相对偏移量无符号扩展到 32 位后的值得到。FLD.T 指令可以寻址 +1KB 地址空间。 注意，偏移量 DISP 是二进制操作数 {IMM4H,IMM4L} 左移两位得到的。
影 响 标 志 位	无影响
异 常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操 作	从存储器加载单、双精度浮点数到矢量寄存器 VRZ[len-1:0] ← MEM[RX + zero_extend(offset << 2)]; VRZ[127:len] ← 0;
语 法	fld.32 vrz, (rx, disp) fld.64 vrz, (rx, disp)
说 明	从存储器加载单、双精度浮点数到矢量寄存器 VRZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移两位的 8 位相对偏移量无符号扩展到 32 位后的值得到。FLD.32 指令可以寻址 +1KB 地址空间。 注意，偏移量 DIS P 是二进制操作数 {IMM4H,IMM4L} 左移两位得到的。
影 响 标 志 位	无影响
异 常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

fld.32 vrz, (rx, disp)														
31	30	26	25	24	21	20	16	15	8	7	4	3	0	
1	11101				IMM4H		RX		00100000			IMM4L		VRZL
VRZH														

fld.64 vrz, (rx, disp)														
31	30	26	25	24	21	20	16	15	8	7	4	3	0	
1	11101				IMM4H		RX		00100001			IMM4L		VRZL
VRZH														

14.79 FLDM.T——连续浮点加载指令

统一化指令	
语 法	fldm.32 vry-vrz, (rx) fldm.64 vry-vrz, (rx)
操 作	从存储器加载连续的多个单、双精度浮点数到一片连续的矢量寄存器堆中 dst ← Y; addr ← RX; for (n = 0; n <= (Z-Y); n++){ VRdst[len-1:0] ← MEM[addr]; VRdst[127:len] ← 0; dst ← dst + 1; addr ← addr + len/8; }
编 译 结 果	仅存在 32 位指令。 fldm.32 vry-vrz, (rx); fldm.64 vry-vrz, (rx);
说 明	从存储器依次加载连续的多个单、双精度浮点数到矢量寄存器 VRY 开始的一片连续寄存器堆中，即将存储器指定地址开始的第一个字、双字加载到矢量寄存器 VRY 中，第二个字、双字加载到寄存器 VR(Y+1) 中，依次类推，最后一个字、双字加载到寄存器 VRZ 中。存储器的有效地址由基址寄存器 RX 的内容决定。
影 响 标 志 位	无影响
限 制	VRZ 应当大于等于 VRY。
异 常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	从存储器加载连续的多个单、双精度浮点数到一片连续的矢量寄存器堆中 dst Y; addr RX; for (n = 0; n <= (Z-Y); n++){ VRdst[len-1:0] MEM[addr]; VRdst[127:len] 0; dst dst + 1; addr addr + len/8; }
语法	fldm.32 vry-vrz, (rx) fldm.64 vry-vrz, (rx)
说明	从存储器依次加载连续的多个单、双精度浮点数到矢量寄存器 VRY 开始的一片连续寄存器堆中，即将存储器指定地址开始的第一个字、双字加载到矢量寄存器 VRY 中，第二个字、双字加载到寄存器 VR(Y+1) 中，依次类推，最后一个字、双字加载到寄存器 VRZ 中。存储器的有效地址由基址寄存器 RX 的内容决定。
影响标志位	无影响
限制	VRZ 应当大于等于 VRY。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

fldm.32 vry-vrz, (rx)															
31	30	26	25	21	20	16	15	8	7	4	3	0			
1	11101			IMM5			RX		00110000			000		VRY	

fldm.64 vry-vrz, (rx)															
31	30	26	25	21	20	16	15	8	7	4	3	0			
1	11101			IMM5			RX		00110001			000		VRY	

IMM5 域：

指定目标寄存器的个数，IMM5 = Z - Y。

00000：

1 个目的寄存器

00001：

2 个目的寄存器

.....

11111:

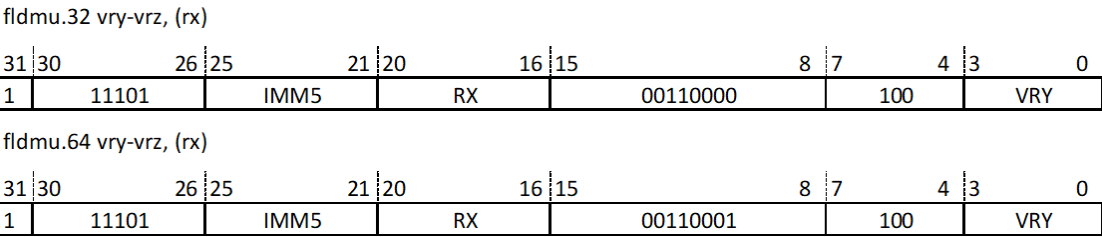
32 个目的寄存器

14.80 FLDMU.T——连续浮点加载地址自增指令

统一化指令	
语 法	fldmu.32 vry-vrz, (rx) fldmu.64 vry-vrz, (rx)
操 作	从存储器加载连续的多个单、双精度浮点数到一片连续的矢量寄存器堆中，并更新基址寄存器 dst ← Y; addr ← RX; for (n = 0; n <= (Z-Y); n++){ VRdst[len-1:0] ← MEM[addr]; VRdst[127:len] ← 96’ b0; dst ← dst + 1; addr ← addr + len/8; } RX←addr
编 译 结 果	仅存在 32 位指令。 fldmu.32 vry-vrz, (rx); fldmu.64 vry-vrz, (rx);
说 明	从存储器依次加载连续的多个单、双精度浮点数到矢量寄存器 VRY 开始的一片连续寄存器堆中，并更新基址寄存器为本地加载的最后一个地址之后下一个地址的位置，即将存储器指定地址开始的第一个字、双字加载到矢量寄存器 VRY 中，第二个字、双字加载到寄存器 VR(Y+1) 中，依次类推，最后一个字、双字加载到寄存器 VRZ 中。存储器的有效地址由基址寄存器 RX 的内容决定。最后，更新基址寄存器 RX 为 VRZ 的基址的下一个字、双字地址。
影 响 标 志 位	无影响
限 制	VRZ 应当大于等于 VRY。
异 常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	从存储器加载连续的多个单精度浮点数到一片连续的矢量寄存器堆中，并更新基址寄存器 $dst \leftarrow Y; addr \leftarrow RX;$ for ($n = 0; n \leq (Z-Y); n++$){ $VRdst[len-1:0] \leftarrow MEM[addr];$ $VRdst[127:len] \leftarrow 96'b0;$ $dst \leftarrow dst + 1;$ $addr \leftarrow addr + len/8;$ } $RX \leftarrow addr$
语法	$fldmu.32\ vry-vrz, (rx)$ $fldmu.64\ vry-vrz, (rx)$
说明	从存储器依次加载连续的多个单、双精度浮点数到矢量寄存器 VR _Y 开始的一片连续寄存器堆中，并更新基址寄存器为本地加载的最后一个地址之后下一个地址的位置，即将存储器指定地址开始的第一个字、双字加载到矢量寄存器 VR _Y 中，第二个字、双字加载到寄存器 VR _(Y+1) 中，依次类推，最后一个字、双字加载到寄存器 VR _Z 中。存储器的有效地址由基址寄存器 RX 的内容决定。最后，更新基址寄存器 RX 为 VR _Z 的基址的下一个字、双字地址。
影响标志位	无影响
限制	VR _Z 应当大于等于 VR _Y 。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：



IMM5 域：

指定目标寄存器的个数， $IMM5 = Z - Y$ 。

00000：

1 个目的寄存器

00001:
2 个目的寄存器
.....
11111:
32 个目的寄存器

14.81 FLDR.T——寄存器移位寻址浮点加载指令

统一化指令	
语法	fldr.32 vrz, (rx, ry << 0) fldr.32 vrz, (rx, ry << 1) fldr.32 vrz, (rx, ry << 2) fldr.32 vrz, (rx, ry << 3) fldr.64 vrz, (rx, ry << 0) fldr.64 vrz, (rx, ry << 1) fldr.64 vrz, (rx, ry << 2) fldr.64 vrz, (rx, ry << 3)
操作	从存储器加载单、双精度浮点数到寄存器 $VRZ[len-1:0] \leftarrow MEM[RX + RY \ll IMM2];$ $VRZ[127:len] = 0;$
编译结果	仅存在 32 位指令。 fldr.32 vrz, (rx, ry << 0) fldr.32 vrz, (rx, ry << 1) fldr.32 vrz, (rx, ry << 2) fldr.32 vrz, (rx, ry << 3) fldr.64 vrz, (rx, ry << 0) fldr.64 vrz, (rx, ry << 1) fldr.64 vrz, (rx, ry << 2) fldr.64 vrz, (rx, ry << 3)
说明	从存储器加载单、双精度浮点数到寄存器 VRZ 中。采用寄存器加寄存器移位的寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值为 0。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	从存储器加载单精度浮点数到寄存器 $VRZ[len-1:0] \leftarrow MEM[RX + RY \ll IMM2];$ $VRZ[127:len] = 0;$
语法	$fldr.32\ vrz, (rx, ry \ll 0)$ $fldr.32\ vrz, (rx, ry \ll 1)$ $fldr.32\ vrz, (rx, ry \ll 2)$ $fldr.32\ vrz, (rx, ry \ll 3)$ $fldr.64\ vrz, (rx, ry \ll 0)$ $fldr.64\ vrz, (rx, ry \ll 1)$ $fldr.64\ vrz, (rx, ry \ll 2)$ $fldr.64\ vrz, (rx, ry \ll 3)$
说明	从存储器加载单、双精度浮点数到寄存器 VRZ 中。采用寄存器加寄存器移位的寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值为 0。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

fldr.32 vrz, (rx, ry << 0)

31	30	26	25	21	20	16	15	8	7	6	5	4	0
1	11101		RY		RX		00101000	0		00		VRZ	

fldr.32 vrz, (rx, ry << 1)

31	30	26	25	21	20	16	15	8	7	6	5	4	0
1	11101		RY		RX		00101000	0		01		VRZ	

fldr.32 vrz, (rx, ry << 2)

31	30	26	25	21	20	16	15	8	7	6	5	4	0
1	11101		RY		RX		00101000	0		10		VRZ	

fldr.32 vrz, (rx, ry << 3)

31	30	26	25	21	20	16	15	8	7	6	5	4	0
1	11101		RY		RX		00101000	0		11		VRZ	

fldr.64 vrz, (rx, ry << 0)

31	30	26	25	21	20	16	15	8	7	6	5	4	0
1	11101		RY		RX		00101001	0		00		VRZ	

fldr.64 vrz, (rx, ry << 1)

31	30	26	25	21	20	16	15	8	7	6	5	4	0
1	11101		RY		RX		00101001	0		01		VRZ	

fldr.64 vrz, (rx, ry << 2)

31	30	26	25	21	20	16	15	8	7	6	5	4	0
1	11101		RY		RX		00101001	0		10		VRZ	

fldr.64 vrz, (rx, ry << 3)

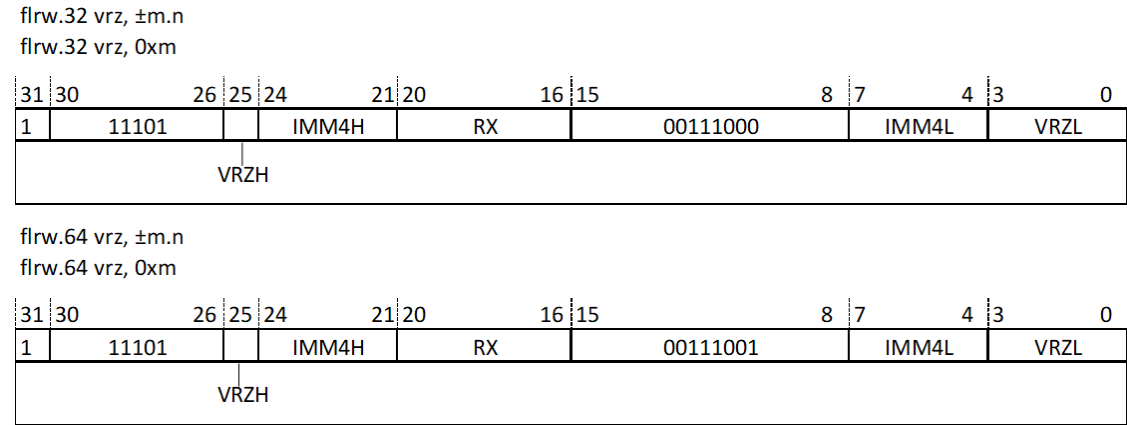
31	30	26	25	21	20	16	15	8	7	6	5	4	0
1	11101		RY		RX		00101001	0		10		VRZ	

14.82 FLRW.T——浮点存储器读入指令

统一化指令	
语 法	flrw.32 vrz, $\pm m.n$ flrw.32 vrz, 0xm flrw.64 vrz, $\pm m.n$ flrw.64 vrz, 0xm
操 作	VRZ[len-1:0] $\leftarrow$ MEM[(PC + zero_extend(offset << 2))&0xfffffc]; VRZ[127:len] $\leftarrow$ 0;
编 译 结 果	仅存在 32 位指令。 flrw.32 vrz, $\pm m.n$ flrw.32 vrz, 0xm flrw.64 vrz, $\pm m.n$ flrw.64 vrz, 0xm
说 明	从存储器加载 32 位单精度浮点、64 位双精度浮点数 ($\pm m.n$) 或 32、64 位十六进制数到浮点寄存器 VRZ 中。存储器的有效地址根据 PC 加左移两位的相对偏移量无符号扩展到 32 位后的值得到。FLRW.32 指令的加载范围是全部 4GB 地址空间。 注意：由于精度不够或者超出表示范围的浮点立即数，编译器将给出 error。
影 响 标 志 位	无影响
异 常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	从存储器加载单精度浮点数到浮点寄存器 $VRZ[len-1:0] \leftarrow MEM[(PC + zero\_extend(offset << 2))\&0xffffffffc];$ $VRZ[127:len] \leftarrow 0;$
语法	$flrw.32\ vrz,\ \pm m.n$ $flrw.32\ vrz,\ 0xm$ $flrw.64\ vrz,\ \pm m.n$ $flrw.64\ vrz,\ 0xm$
说明	从存储器加载 32 位单精度浮点、64 位双精度浮点数 ($\pm m.n$) 或 32、64 位十六进制数到浮点寄存器 VRZ 中。存储器的有效地址根据 PC 加左移两位的相对偏移量无符号扩展到 32 位后的值得到。FLRW.32 指令的加载范围是全部 4GB 地址空间。 注意：由于精度不够或者超出表示范围的浮点立即数，编译器将给出 error。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：



14.83 FMAXNM.T——浮点规范最大值选择指令

统一化指令	
语法	fmaxnm.T vrz, vrx,vry (T=32/64)
操作	if vrx>=vry vrz = vrx; else vrz = vy;
编译结果	仅存在 32 位指令。 fmaxnm.T vrz, vrx,vry
数据类型	单精度/双精度
说明	取两元素中值较大的一个; 且当两个元素中一个为 quite NaN 而另一个为规范数时，取规范数的值作为输出。
影响标志位	无影响。
限制	寄存器范围为 vr0-vr31。
异常	IO、ID

32 位指令	
操作	if vrx>=vry vrz = vrx; else vrz = vy;
语法	fmaxnm.T vrz, vrx,vry
数据类型	单精度/双精度
说明	取两元素中值较大的一个; 且当两个元素中一个为 quite NaN 而另一个为规范数时，取规范数的值作为输出。
影响标志位	无影响。
限制	寄存器范围为 vr0-vr31。
异常	IO、ID

32 位指令格式：

fmaxnm.32													
31	30	26	25	21	20	16	15	10	9	5	4	0	
1	11101			VRX		VRX			000001		01000		VRZ

fmaxnm.64													
31	30	26	25	21	20	16	15	10	9	5	4	0	
1	11101			VRX		VRX			000011		01000		VRZ

14.84 FMFVR.64——浮点寄存器双字读传送指令

统一化指令	
语法	fmfvr.64 rx, ry, vrz
操作	Ry[31:0]=Vz[63:32]; Rx[31:0]=Vz{31:0};
编译结果	仅存在 32 位指令。 fmfvr.64 rx, ry, vrz
数据类型	双精度浮点数
说明	将浮点寄存器的低字搬运到 rx 中，高字搬运到 ry 中。
影响标志位	无
限制	寄存器范围为 vr0-vr31。
异常	无

32 位指令	
操作	Ry[31:0]=Vz[63:32]; Rx[31:0]=Vz{31:0};
语法	fmfvr.64 vrz, rx, ry
数据类型	双精度浮点数
说明	将浮点寄存器的低字搬运到 rx 中，高字搬运到 ry 中。
影响标志位	无
限制	寄存器范围为 vr0-vr31。
异常	无

32 位指令格式：

31	26	25	21	20	16	15	5	4	0			
111101				RY		VRZ		00011111000			RX	

14.85 FMFVR.32.1——浮点寄存器低字读传送指令

统一化指令	
语法	Fmfvr.32.1 rx, vrz
操作	rx= vrz[31:0]
编译结果	仅存在 32 位指令。 Fmfvr.32.1 rx, vrz
数据类型	单精度浮点数
说明	将浮点寄存器的低字搬运到 rx 中。
影响标志位	无
限制	寄存器范围为 vr0-vr31。
异常	无

32 位指令	
操作	rx= vrz[31:0]
语法	fmfvr.32.1 rx, vrz
数据类型	单精度浮点数
说明	将浮点寄存器的低字搬运到 rx 中。
影响标志位	无
限制	寄存器范围为 vr0-vr31。
异常	无

32 位指令格式：

31	26	25	24	21	20	16	15	5	4	0
111101			0	0000		VRZ		00011011001		RX

14.86 FMFVR.32.2——浮点双寄存器低字读传送指令

统一化指令	
语法	fmfvr.32.2 rx,ry, vrz
操作	Rx[31:0]=Vrz[31:0]; Ry[31:0]= Vrz+1[31:0]
编译结果	仅存在 32 位指令。 fmfvr.32.2 rx,ry, vrz
数据类型	单精度浮点数
说明	将浮点寄存器 vrz 的低字搬运到 rx 中。 将浮点寄存器 vrz+1 的低字搬运到 ry 中。
影响标志位	无
限制	寄存器范围为 vr0-vr31。
异常	无

32 位指令	
操作	Rx[31:0]=Vrz[31:0]; Ry[31:0]= Vrz+1[31:0]
语法	fmfvr.32.2 rx,ry, vrz
数据类型	单精度浮点数
说明	将浮点寄存器 vrz 的低字搬运到 rx 中。 将浮点寄存器 vrz+1 的低字搬运到 ry 中。
影响标志位	无
限制	寄存器范围为 vr0-vr31。
异常	无

32 位指令格式：

31	26	25	21	20	16	15	5	4	0
111101			RY		VRZ		0001111010		RX

14.87 FMFVRH——浮点寄存器高字读传送指令

统一化指令	
语法	fmfvrh rz, vr _x
操作	rz= vr _x [63:32]
编译结果	仅存在 32 位指令。 fmfvrh rz, vr _x
数据类型	单精度浮点数
说明	将浮点寄存器的高字搬运到 rz 中。
影响标志位	无
限制	寄存器范围为 vr ₀ -vr ₃₁ 。
异常	无

32 位指令	
操作	rz= vr _x [63:32]
语法	fmfvrh rz, vr _x
数据类型	单精度浮点数
说明	将浮点寄存器的高字搬运到 rz 中。
影响标志位	无
限制	寄存器范围为 vr ₀ -vr ₃₁ 。
异常	无

32 位指令格式：

31	26	25	24	21	20	16	15	5	4	0
111101			0	0000		VRX		00011011000		RZ

14.88 FMINNM.T——浮点规范最小值选择指令

统一化指令	
语法	fminnm.T vrz, vrx,vry (T=32/64)
操作	If vrx <= vry vrz = vrx; else vrz = vry;
编译结果	仅存在 32 位指令。 fminnm.T vrz, vrx,vry
数据类型	单精度/双精度
说明	取两元素中值较小的一个; 且当两个元素中一个为 quite NaN 而另一个为规范数时，取规范数的值作为输出。
影响标志位	无影响。
限制	寄存器范围为 vr0-vr31。
异常	IO、ID

32 位指令	
操作	if vrx>=vry vrz = vrx; else vrz = vy;
语法	fminnm.T vrz, vrx,vry
数据类型	单精度/双精度
说明	取两元素中值较小的一个; 且当两个元素中一个为 quite NaN 而另一个为规范数时，取规范数的值作为输出。
影响标志位	无影响。
限制	寄存器范围为 vr0-vr31。
异常	IO、ID

32 位指令格式：

fminnm.32																				
31	30	26	25	21	20	16	15	10	9	5	4	0								
1	11101				VRY				VRX				000001			01001		VRZ		
fminnm.64																				
31	30	26	25	21	20	16	15	10	9	5	4	0								
1	11101				VRY				VRX				000011			01001		VRZ		

14.89 FMOV.T——浮点寄存器间传送指令

统一化指令	
语法	fmov.T vrz, vrx (T=32/64)
操作	Vrz[len-1:0]=Vrx[len-1:0];
编译结果	仅存在 32 位指令。 fmov.T vrz, vrx
数据类型	单精度/双浮点数
说明	将浮点寄存器 vr x[len-1:0] 的值搬运到浮点寄存器 vrz[len-1:0] 中。
影响标志位	无
限制	寄存器范围为 vr0-vr31。
异常	无

32 位指令	
操作	Vrz[len-1:0]=Vrx[len-1:0];
语法	fmov.T vrz, vrx
数据类型	单精度/双浮点数
说明	将浮点寄存器 vrx[len-1:0] 的值搬运到浮点寄存器 vrz[len-1:0] 中。
影响标志位	无
限制	寄存器范围为 vr0-vr31。
异常	无

32 位指令格式：

Fmov.32											
31	26	25	24	21	20	16	15	5	4	0	
111101		0	0000		VRX		00000000100			VRZ	
Fmov.64											
31	26	25	24	21	20	16	15	5	4	0	
111101		0	0000		VRX		00001000100			VRZ	

14.90 FMOVX.32——浮点寄存器间低半字传送指令

统一化指令	
语法	fmovx.32 vrz, vrx
操作	Vrz[31:0]={16’ b0, Vrx[31:16]};
编译结果	仅存在 32 位指令。 fmovx.32 vrz, vrx
数据类型	单精度浮点数
说明	将浮点寄存器 vrx 的低字高半字搬运到浮点寄存器 vrz 的低字低半字中，低字高半字和高字为 0。
影响标志位	无
限制	寄存器范围为 vr0-vr31。
异常	无

32 位指令	
操作	Vrz[31:0]={16’ b0, Vrx[31:16]};
语法	fmovx.32 vrz, vrx
数据类型	单精度浮点数
说明	将浮点寄存器 vrx 的低字高半字搬运到浮点寄存器 vrz 的低字低半字中，低字高半字和高字为 0。
影响标志位	无
限制	寄存器范围为 vr0-vr31。低字高半字和高字为 0。
异常	无

指令格式：

31	26	25	24	21	20	16	15	5	4	0
111101			0	0000		VRX		00001000101		VRZ

14.91 FMTVR.64——浮点双寄存器写传送指令

统一化指令	
语法	fmtvr.64 vrz, rx, ry
操作	Vrz[63:0]={Ry[31:0], Rx[31:0]};
编译结果	仅存在 32 位指令。 fmtvr.64 vrz, rx, ry
数据类型	双精度浮点数
说明	将 rx 搬运到浮点寄存器的低字中，ry 搬运到浮点寄存器的高字中。
影响标志位	无
限制	寄存器范围为 vr0-vr31。
异常	无

32 位指令	
操作	Vz[63:0]={Ry[31:0], Rx[31:0]};
语法	fmtvr.64 vrz, rx, ry
数据类型	双精度浮点数
说明	将 rx 搬运到浮点寄存器的低字中，ry 搬运到浮点寄存器的高字中。
影响标志位	无
限制	寄存器范围为 vr0-vr31。
异常	无

32 位指令格式：

31	26	25	21	20	16	15	5	4	0
111101			RY		RX		0001111100		VRZ

14.92 FMTVR.32.1——浮点寄存器低字写传送指令

统一化指令	
语法	fmtvr.32.1 vrz, rx
操作	vrz[31:0] = rx
编译结果	仅存在 32 位指令。 fmtvr.32.1 vrz, rx
数据类型	单精度浮点数
说明	将 rx 搬运到浮点寄存器的低字中，高字保持不变。
影响标志位	无
限制	寄存器范围为 vr0-vr31。
异常	无

32 位指令	
操作	vrz[31:0] = rx
语法	fmtvr.32.1 vrz, rx
数据类型	单精度浮点数
说明	将 rx 搬运到浮点寄存器的低字中。
影响标志位	无
限制	寄存器范围为 vr0-vr31。高字保持不变。
异常	无

32 位指令格式：

31	26	25	24	21	20	16	15	5	4	0
111101	0		0000		RX		00011011011		VRZ	

14.93 FMTVR.32.2——浮点双寄存器低字写传送指令

统一化指令	
语法	fmtvr.32.2 vrz, rx,ry
操作	Vz[31:0]=Rx[31:0]; Vz+1[31:0]=Ry[31:0];
编译结果	仅存在 32 位指令。 fmtvr.32.2 vrz, rx,ry
数据类型	单精度浮点数
说明	将 rx 搬运到浮点寄存器 Vrz 的低字中，高字保持不变。 将 ry 搬运到浮点寄存器 Vrz+1 的低字中，高字保持不变。 当 Vrz 为 31 时，Vrz+1 为 0。
影响标志位	无
限制	寄存器范围为 vr0-vr31。
异常	无

32 位指令	
操作	$vrz[31:0] = rx$ $Vz+1[31:0]=Ry[31:0];$
语法	fmtvr.32.2 vrz, rx,ry
数据类型	单精度浮点数
说明	将 rx 搬运到浮点寄存器 Vrz 的低字中，高字保持不变。 将 ry 搬运到浮点寄存器 Vrz+1 的低字中，高字保持不变。 当 Vrz 为 31 时，Vrz+1 为 0。
影响标志位	无
限制	寄存器范围为 vr0-vr31。高字保持不变。
异常	无

32 位指令格式：

31	26	25	21	20	16	15	5	4	0
111101		RY		RX		0001111110		VRZ	

14.94 FMTVRH——浮点寄存器高字写传送指令

统一化指令	
语法	fmtvrh vrz, rx
操作	$vrz[63:32] = rx$
编译结果	仅存在 32 位指令。 fmtvrh vrz, rx
数据类型	单精度浮点数
说明	将 rx 搬运到浮点寄存器的高字中。
影响标志位	无
限制	寄存器范围为 vr0-vr31。
异常	无

32 位指令	
操作	vrz[63:32] = rx
语法	fmtvrh vrz, rx
数据类型	单精度浮点数
说明	将 rx 搬运到浮点寄存器的高字中。
影响标志位	无
限制	寄存器范围为 vr0-vr31。
异常	无

32 位指令格式：

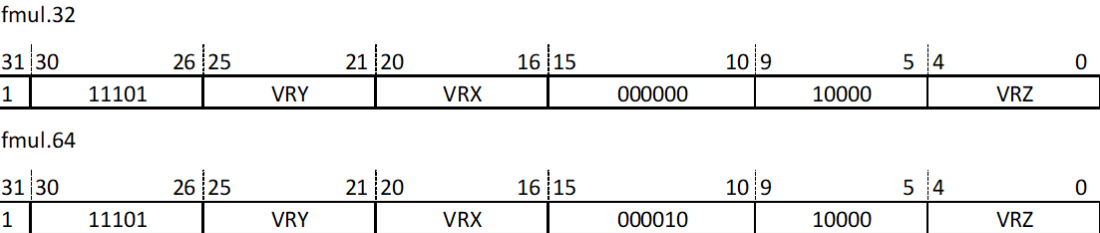
31	26	25	24	21	20	16	15	5	4	0
111101	0	0000		RX		00011011010		VRZ		

14.95 FMUL.T——浮点乘法指令

统一化指令	
语法	fmul.t vrz, vr _x , vr _y (t=32/64)
操作	vrz = vr _x * vr _y
编译结果	仅存在 32 位指令 fmul.t vrz, vr _x , vr _y
数据类型	单精度/双精度
说明	单精度浮点乘法指令将寄存器 vr _x 的值乘以寄存器 vr _y 的值，得到的积存放于 vrz 中。vr _x 、vr _y 和 vrz 的值均被认为是 32 位有符号数。
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令	
操作	$vrz = vrx * vry$
数据类型	单精度/双精度
语法	<code>fmul.t vrz, vrx, vry</code>
说明	单精度浮点乘法指令将寄存器 <code>vrx</code> 的值乘以寄存器 <code>vry</code> 的值，得到的积存放于 <code>vrz</code> 中。 <code>vrx</code> 、 <code>vry</code> 和 <code>vrz</code> 的值均被认为是 32 位有符号数
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令格式：



14.96 FMULA.T——浮点乘累加指令

统一化指令	
语法	<code>fmula.t vrz, vrx, vry</code> ($t=32/64$)
操作	$vrz = vrz + vrx * vry$
编译结果	仅存在 32 位指令 <code>fmula.t vrz, vrx, vry</code>
数据类型	单精度/双精度
说明	将 <code>vrx</code> 中的单精度浮点数与 <code>vry</code> 中的单精度浮点数相乘得到乘积，对乘积进行舍入操作后与 <code>vrz</code> 中的单精度浮点数相加，将结果写回 <code>vrz</code> 中
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令	
操作	$vrz = vrz + vrx * vry$
数据类型	单精度/双精度
语法	fmula.t vrz, vrx, vry
说明	将 vrx 中的单精度浮点数与 vry 中的单精度浮点数相乘得到乘积，对乘积进行舍入操作后与 vrz 中的单精度浮点数相加，将结果写回 vrz 中
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令格式：

fmula.32															
31	30	26	25	21	20	16	15	10	9	5	4	0			
1	11101			VRY			VRX			000000			10100		VRZ
fmula.64															
31	30	26	25	21	20	16	15	10	9	5	4	0			
1	11101			VRY			VRX			000010			10100		VRZ

14.97 FMULS.T——浮点乘取负累加指令

统一化指令	
语法	fmuls.t vrz, vrx, vry (t=32/64)
操作	$vrz = vrz - vrx * vry$
编译结果	仅存在 32 位指令 fmuls.t vrz, vrx, vry
数据类型	单精度/双精度
说明	将 vrx 中的单精度浮点数与 vry 中的单精度浮点数相乘得到乘积，对乘积进行舍入操作并取负后与 vrz 中的单精度浮点数相加，将结果写回 vrz 中
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令	
操作	$VRZ = VRZ - VRX * VRy$
数据类型	单精度/双精度
语法	<code>fmuls.t vrz, vrx, vry</code>
说明	将 <code>vrx</code> 中的单精度浮点数与 <code>vry</code> 中的单精度浮点数相乘得到乘积，对乘积进行舍入操作并取负后与 <code>vrz</code> 中的单精度浮点数相加，将结果写回 <code>vrz</code> 中
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令格式：

fmuls.32															
31	30	26	25	21	20	16	15	10	9	5	4	0			
1	11101			VRy			VRX			000000			10110		VRZ
fmuls.64															
31	30	26	25	21	20	16	15	10	9	5	4	0			
1	11101			VRy			VRX			000010			10110		VRZ

14.98 FNMUL.T——浮点乘法取负指令

统一化指令	
语法	<code>fnmul.t vrz, vrx, vry</code> (t=32/64)
操作	$VRZ = -(VRX * VRy)$
编译结果	仅存在 32 位指令 <code>fnmul.t vrz, vrx, vry</code>
数据类型	单精度/双精度
说明	单精度浮点乘法取负指令将寄存器 <code>vrx</code> 的值乘以寄存器 <code>vry</code> 的值，得到的积取负后存放于 <code>vrz</code> 中。 <code>vrx</code> 、 <code>vry</code> 和 <code>vrz</code> 的值均被认为是 32 位有符号数。
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令	
操作	$vrz = -(vrx * vry)$
数据类型	单精度/双精度
语法	<code>fmul.t vrz, vrx, vry</code>
说明	单精度浮点乘法取负指令将寄存器 <code>vrx</code> 的值乘以寄存器 <code>vry</code> 的值，得到的积存取负后存放于 <code>vrz</code> 中。 <code>vrx</code> 、 <code>vry</code> 和 <code>vrz</code> 的值均被认为是 32 位有符号数。
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令格式：

fmmul.32															
31	30	26	25	21	20	16	15	10	9	5	4	0			
1	11101			VRY			VRX			000000			10001		VRZ
fmmul.64															
31	30	26	25	21	20	16	15	10	9	5	4	0			
1	11101			VRY			VRX			000010			10001		VRZ

14.99 FNMULA.T——浮点乘取负累减指令

统一化指令	
语法	<code>fnmula.t vrz, vrx, vry</code> (<code>t=32/64</code>)
操作	$vrz = -vrz - vrx * vry$
编译结果	仅存在 32 位指令 <code>fnmula.t vrz, vrx, vry</code>
数据类型	单精度/双精度
说明	将 <code>vrx</code> 中的单精度浮点数与 <code>vry</code> 中的单精度浮点数相乘得到乘积，对乘积进行舍入操作并取负后，与 <code>vrz</code> 中的单精度浮点数相减，将结果写回 <code>vrz</code> 中
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令	
操作	$VRZ = -VRZ - VRX * VRY$
数据类型	单精度/双精度
语法	fnmula.t vrz, vrx, vry
说明	将 vrx 中的单精度浮点数与 vry 中的单精度浮点数相乘得到乘积，对乘积进行舍入操作并取负后，与 vrz 中的单精度浮点数相减，将结果写回 vrz 中
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令格式：

fnmula.32												
31	30	26	25	21	20	16	15	10	9	5	4	0
1	11101			VRY		VRX		000000		10111		VRZ
fnmula.64												
31	30	26	25	21	20	16	15	10	9	5	4	0
1	11101			VRY		VRX		000010		10111		VRZ

14.100 FNMULS.T——浮点乘累减指令

统一化指令	
语法	fnmul.s.t vrz, vrx, vry (t=32/64)
操作	$VRZ = -VRZ + VRX * VRY$
编译结果	仅存在 32 位指令 fnmul.s.t vrz, vrx, vry
数据类型	单精度/双精度
说明	将 vrx 中的单精度浮点数与 vry 中的单精度浮点数相乘得到乘积，对乘积进行舍入操作后，与 vrz 中的单精度浮点数相减，将结果写回 vrz 中
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令	
操作	$VRZ = -VRZ + VRX * VRY$
数据类型	单精度/双精度
语法	fnmuls.t vrz, vrx, vry
说明	将 vrx 中的单精度浮点数与 vry 中的单精度浮点数相乘得到乘积，对乘积进行舍入操作后，与 vrz 中的单精度浮点数相减，将结果写回 vrz 中
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令格式：

fnmuls.32															
31	30	26	25	21	20	16	15	10	9	5	4	0			
1	11101			VRY			VRX			000000			10101		VRZ
fnmuls.64															
31	30	26	25	21	20	16	15	10	9	5	4	0			
1	11101			VRY			VRX			000010			10101		VRZ

14.101 FNEG.T——浮点取反指令

统一化指令	
语法	fneg.T vrz, vrx (T=32/64)
操作	$VRZ = -VRX$
编译结果	仅存在 32 位指令。 fneg.T vrz, vrx
数据类型	单精度/双精度
说明	将 vrx 中的浮点数取反，保存至 vrz 中。
影响标志位	无
限制	寄存器范围为 vr0-vr31。
异常	无

32 位指令	
操作	$VRZ = -VRX$
语法	<code>fneg.T vrz, vrx</code>
数据类型	单精度/双精度
说明	将 <code>vrx</code> 中的浮点数取反，保存至 <code>vrz</code> 中。
影响标志位	无
限制	寄存器范围为 <code>vr0-vr31</code> 。
异常	无

32 指令格式：

fneg.32

31	26	25	24	21	20	16	15	5	4	0	
111101			0	0000			VRX	00000000111			VRZ

fneg.64

31	26	25	24	21	20	16	15	5	4	0	
111101			0	0000		VRX		00001000111			VRZ

14.102 **FRECIP.T——浮点倒数指令**

统一化指令	
语法	<code>frecip.T vrz, vrx</code> ($T=32/64$)
操作	$vrz = 1/vrx$
编译结果	仅存在 32 位指令。 <code>frecip.T vrz, vrx</code>
数据类型	单精度/双精度
说明	将 <code>vrx</code> 中的单精度浮点数取倒数，保存至 <code>vrz</code> 中。
影响标志位	无影响
限制	寄存器范围为 <code>vr0-vr31</code> 。
异常	IO、ID 、UF 、IX 、DZ

32 位指令	
操作	$vrz = 1/vrx$
语法	frecip.T vrz, vrx
数据类型	单精度/双精度
说明	将 vrx 中的单精度浮点数取倒数，保存至 vrz 中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr31。
异常	IO、ID 、UF 、IX 、DZ

32 位指令格式：

frecip.32

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11101	00000	VRX	000000	11001	VRZ						

frecip.64

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11101	00000	VRX	000010	11001	VRZ						

14.103 FSEL.T——浮点寄存器 C 位选择传送指令

统一化指令	
语法	fsel.T vrz, vrx, vry (T=32/64)
操作	$Vrz[len-1:0] = C ? Vrx[len-1:0] : Vry[len-1:0];$
编译结果	仅存在 32 位指令。 fsel.T vrz, vrx, vry
数据类型	浮点数
说明	根据 C 位值将浮点寄存器 vrx 或 vry 中的值搬运到浮点寄存器 vrz 中。
影响标志位	无
限制	寄存器范围为 vr0-vr31。
异常	无

32 位指令	
操作	$Vr_z[len-1:0]=C?Vr_x[len-1:0]:Vr_y[len-1:0];$
语法	<code>fsel.T vrz, vr_x, vr_y</code>
数据类型	浮点数
说明	根据 C 位值将浮点寄存器 vr _x 或 vr _y 中的值搬运到浮点寄存器 vr _z 中。
影响标志位	无
限制	寄存器范围为 vr0-vr31。
异常	无

32 位指令格式：

fsel.32																			
31	26	25	21	20	16	15	5	4	0										
111101				VR _Y				VR _X				00000111001				VR _Z			
fsel.64																			
31	26	25	21	20	16	15	5	4	0										
111101				VR _Y				VR _X				00001111001				VR _Z			

14.104 FSQRT.T——浮点平方根指令

统一化指令	
语法	<code>fsqrt.T vrz, vr_x</code> (T=32/64)
操作	$vr_z = \text{sqrt}(vr_x)$
编译结果	仅存在 32 位指令。 <code>fsqrt.T vrz, vr_x</code>
数据类型	单精度/双精度
说明	将 vr _x 中的单精度浮点数取平方根，保存至 vr _z 中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr31。
异常	IO、ID 、IX

32 位指令	
操作	vrz= sqrt(vrx)
语法	fsqrt.T vrz, vrx
数据类型	单精度/双精度
说明	将 vr _x 中的单精度浮点数取平方根，保存至 vr _z 中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr31。
异常	IO、ID、IX

32 位指令格式:

fsqrt.32												
31	30	26	25	21	20	16	15	10	9	5	4	0
1		11101		00000		VRX		000000		11010		VRZ

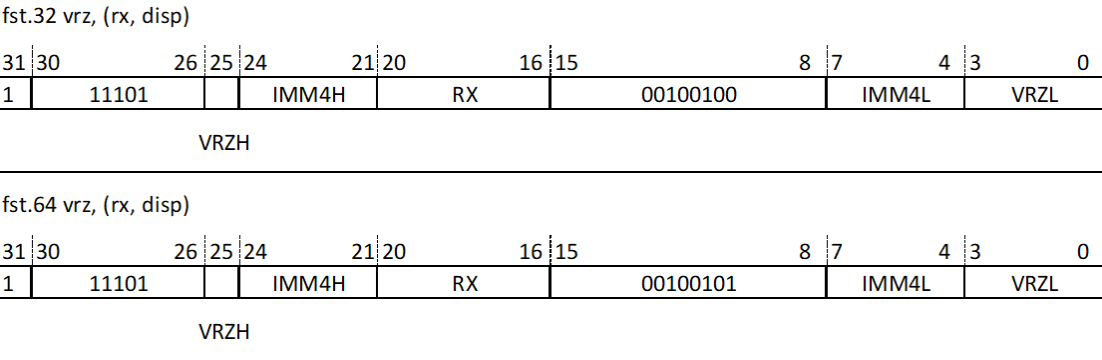
fsqrt.64												
31	30	26	25	21	20	16	15	10	9	5	4	0
1		11101		00000		VRX		000010		11010		VRZ

14.105 FST.T——浮点存储指令

统一化指令	
语法	fst.32 vrz, (rx, disp) fst.64 vrz, (rx, disp)
操作	MEM[RX + zero_extend(offset << 2)] ← VRZ[len-1:0];
编译结果	仅存在 32 位指令。 fst.32 vrz, (rx, disp) fst.64 vrz, (rx, disp)
说明	将寄存器 VRZ 中的单、双精度浮点数存储到存储器中。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移两位的 8 位相对偏移量无符号扩展到 32 位后的值得到。FST.T 指令可以寻址 +1KB 地址空间。 注意，偏移量 DIS P 是二进制操作数 {IMM4H, IMM4L} 左移两位得到的。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常、TLB 修改异常

32 位指令	
操作	将寄存器 VRZ 中的单、双精度浮点数存储到存储器中 MEM[RX + zero_extend(offset << 2)] ← VRZ[len-1:0];
语法	fst.32 vrz, (rx, disp) fst.64 vrz, (rx, disp)
说明	将寄存器 VRZ 中的单、双精度浮点数存储到存储器中。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移两位的 8 位相对偏移量无符号扩展到 32 位后的值得到。FST.T 指令可以寻址 +1KB 地址空间。 注意，偏移量 DIS P 是二进制操作数 {IMM4H,IMM4L} 左移两位得到的。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常、TLB 修改异常

32 位指令格式：

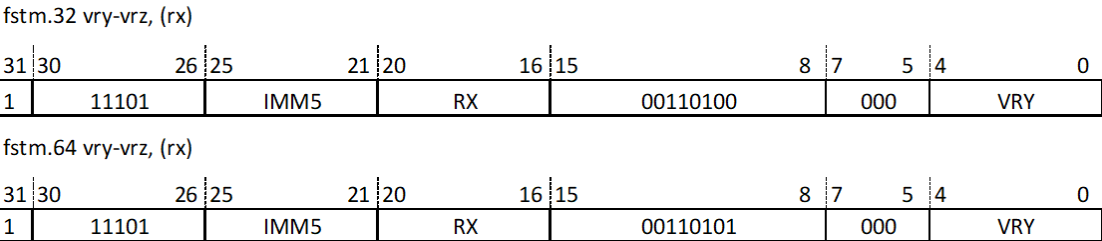


14.106 FSTM.T——连续浮点存储指令

统一化指令	
语 法	fstm.32 vry-vrz, (rx) fstm.64 vry-vrz, (rx)
操 作	将一片连续的寄存器堆中的单、双精度浮点数依次存储到一片连续的存储器地址上。 src ← Y; addr ← RX; for (n = 0; n <=(Z-Y); n++){ MEM[addr] ← VRsrc[len-1:0]; src ← src + 1; addr ← addr + len/8; }
编 译 结 果	仅存在 32 位指令。 fstm.32 vry-vrz, (rx); fstm.64 vry-vrz, (rx);
说 明	将从 VRY 开始的一片连续的寄存器堆中的单、双精度浮点数依次存储到一片连续的存储器地址上，即将寄存器 VRY 的内容存到存储器指定地址开始的第一个字、双字的地址上，寄存器 VR(Y+1) 的内容存到存储器指定地址开始的第二个字、双字的地址上，依次类推，将寄存器 VRZ 的内容存到存储器指定地址开始的最后一个字、双字的地址上。存储器的有效地址由基址寄存器 RX 的内容决定。
影 响 标 志 位	无影响
限 制	VRZ 应当大于等于 VRY。
异 常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常、TLB 修改异常

32 位指令	
操作	将一片连续的寄存器堆中的单、双精度浮点数依次存储到一片连续的存储器地址上。 src ← Y; addr ← RX; for (n = 0; n <=(Z-Y); n++){ MEM[addr] ← VRsrc[len-1:0]; src ← src + 1; addr ← addr + len/8; }
语法	fstm.32 vry-vrz, (rx) fstm.64 vry-vrz, (rx)
说明	将从 VRY 开始的一片连续的寄存器堆中的单、双精度浮点数依次存储到一片连续的存储器地址上，即将寄存器 VRY 的内容存到存储器指定地址开始的第一个字、双字的地址上，寄存器 VR(Y+1) 的内容存到存储器指定地址开始的第二个字、双字的地址上，依次类推，将寄存器 VRZ 的内容存到存储器指定地址开始的最后一个字、双字的地址上。存储器的有效地址由基址寄存器 RX 的内容决定。
影响标志位	无影响
限制	VRZ 应当大于等于 VRY。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常、TLB 修改异常

32 位指令格式：



IMM5 域：

指定目标寄存器的个数，IMM5 = Z - Y。

00000：

1 个目的寄存器

00001：

2 个目的寄存器

.....

11111:

32 个目的寄存器

14.107 FSTMU.T——连续浮点存储地址自增指令

统一化指令	
语 法	fstmu.32 vry-vrz, (rx) fstmu.64 vry-vrz, (rx)
操 作	将一片连续的寄存器堆中的单、双精度浮点数依次存储到一片连续的存储器地址上，并更新基址寄存器。 src ← Y; addr ← RX; for (n = 0; n <=(Z-Y); n++){ MEM[addr] ← VRsrc[len-1:0]; src ← src + 1; addr ← addr + len/8; } RX←addr
编 译 结 果	仅存在 32 位指令。 fstmu.32 vry-vrz, (rx); fstmu.64 vry-vrz, (rx);
说 明	将从 VRY 开始的一片连续的寄存器堆中的单、双精度浮点数依次存储到一片连续的存储器地址上，并更新基址寄存器为本地加载的最后一个地址之后一个字、双字的位置，即将寄存器 VRY 的内容存到存储器指定地址开始的第一个字、双字的地址上，寄存器 VR(Y+1) 的内容存到存储器指定地址开始的第二个字、双字的地址上，依次类推，将寄存器 VRZ 的内容存到存储器指定地址开始的最后一个字、双字的地址上。存储器的有效地址由基址寄存器 RX 的内容决定。最后，更新基址寄存器 RX 为 VRZ 的基址之后的字、双字的地址。
影 响 标 志 位	无影响
限 制	VRZ 应当大于等于 VRY。
异 常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常、TLB 修改异常

32 位指令	
操作	将一片连续的寄存器堆中的单、双精度浮点数依次存储到一片连续的存储器地址上，并更新基址寄存器。 $src \leftarrow Y; addr \leftarrow RX;$ for ($n = 0; n \leq (Z-Y); n++$){ $MEM[addr] \leftarrow VRsrc[len-1:0];$ $src \leftarrow src + 1;$ $addr \leftarrow addr + len/8;$ } $RX \leftarrow addr$
语法	$fstmu.32 \ vry-vrz, (rx)$ $fstmu.64 \ vry-vrz, (rx)$
说明	将从 VRY 开始的一片连续的寄存器堆中的单、双精度浮点数依次存储到一片连续的存储器地址上，并更新基址寄存器为本地加载的最后一个地址之后一个字、双字的位置，即将寄存器 VRY 的内容存到存储器指定地址开始的第一个字、双字的地址上，寄存器 VR(Y+1) 的内容存到存储器指定地址开始的第二个字、双字的地址上，依次类推，将寄存器 VRZ 的内容存到存储器指定地址开始的最后一个字、双字的地址上。存储器的有效地址由基址寄存器 RX 的内容决定。最后，更新基址寄存器 RX 为 VRZ 的基址之后的字、双字的地址。
影响标志位	无影响
限制	VRZ 应当大于等于 VRY。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常、TLB 修改异常

32 位指令格式：

fstmu.32 vry-vrz, (rx)												
31	30	26	25	21	20	16	15	8	7	5	4	0
1	11101			IMM5		RX		00110100		100		VRY

fstmu.64 vry-vrz, (rx)												
31	30	26	25	21	20	16	15	8	7	5	4	0
1	11101			IMM5		RX		00110101		100		VRY

IMM5 域：

指定目标寄存器的个数， $IMM5 = Z - Y$ 。

00000：

1 个目的寄存器

00001:

2 个目的寄存器

.....

11111:

32 个目的寄存器

14.108 FSTOD——单精度浮点双精度浮点转换指令

统一化指令	
语法	fstod vrz, vrx
操作	vrz= (double)vrx
编译结果	仅存在 32 位指令。 fstod vrz, vrx
数据类型	单精度浮点数
说明	将 vr _x 中的单精度浮点数转换成双精度浮点数，保存至 vr _z 中。
影响标志位	无
限制	寄存器范围为 vr ₀ -vr ₃₁ 。
异常	ID,IO

32 位指令	
操作	vrz= (double)vrx
语法	fstod vrz, vrx
数据类型	单精度浮点数
说明	将 vr _x 中的单精度浮点数转换成双精度浮点数，保存至 vr _z 中。
影响标志位	无
限制	寄存器范围为 vr ₀ -vr ₃₁ 。
异常	ID,IO

32 位指令格式:

31	26	25	24	21	20	16	15	5	4	0
111101			0	0000		VRX		00011010111		VRZ

14.109 FSTR.T——寄存器移位寻址浮点存储指令

统一化指令	
语法	fstr.32 vrz, (rx, ry << 0) fstr.32 vrz, (rx, ry << 1) fstr.32 vrz, (rx, ry << 2) fstr.32 vrz, (rx, ry << 3) fstr.64 vrz, (rx, ry << 0) fstr.64 vrz, (rx, ry << 1) fstr.64 vrz, (rx, ry << 2) fstr.64 vrz, (rx, ry << 3)
操作	存储单、双精度浮点数至存储器 MEM[RX + RY << IMM2] ← VRZ[len-1:0];
编译结果	仅存在 32 位指令。 fstr.32 vrz, (rx, ry << 0) fstr.32 vrz, (rx, ry << 1) fstr.32 vrz, (rx, ry << 2) fstr.32 vrz, (rx, ry << 3) fstr.64 vrz, (rx, ry << 0) fstr.64 vrz, (rx, ry << 1) fstr.64 vrz, (rx, ry << 2) fstr.64 vrz, (rx, ry << 3)
说明	从寄存器 VRZ 存储单、双精度浮点数到存储器中。采用寄存器加寄存器移位的寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值为 0。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常、TLB 修改异常

32 位指令	
操作	存储单、双精度浮点数至存储器 MEM[RX + RY << IMM2] ← VRZ[len-1:0];
语法	fstr.32 vrz, (rx, ry << 0) fstr.32 vrz, (rx, ry << 1) fstr.32 vrz, (rx, ry << 2) fstr.32 vrz, (rx, ry << 3) fstr.64 vrz, (rx, ry << 0) fstr.64 vrz, (rx, ry << 1) fstr.64 vrz, (rx, ry << 2) fstr.64 vrz, (rx, ry << 3)
说明	从寄存器 VRZ 存储单、双精度浮点数到存储器中。采用寄存器加寄存器移位的寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值为 0。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常、TLB 修改异常

32 位指令格式：

fstr.32 vrz, (rx, ry << 0)															
31	30	26	25	21	20	16	15	8	7	6	5	4			0
1		11101													
fstr.32 vrz, (rx, ry << 1)															
31	30	26	25	21	20	16	15	8	7	6	5	4			0
1		11101													
fstr.32 vrz, (rx, ry << 2)															
31	30	26	25	21	20	16	15	8	7	6	5	4			0
1		11101													
fstr.32 vrz, (rx, ry << 3)															
31	30	26	25	21	20	16	15	8	7	6	5	4			0
1		11101													
fstr.64 vrz, (rx, ry << 0)															
31	30	26	25	21	20	16	15	8	7	6	5	4			0
1		11101													
fstr.64 vrz, (rx, ry << 1)															
31	30	26	25	21	20	16	15	8	7	6	5	4			0
1		11101													
fstr.64 vrz, (rx, ry << 2)															
31	30	26	25	21	20	16	15	8	7	6	5	4			0
1		11101													
fstr.64 vrz, (rx, ry << 3)															
31	30	26	25	21	20	16	15	8	7	6	5	4			0
1		11101													

14.110 FSUB.T——浮点减法指令

统一化指令	
语法	fsub.T vrz, vrx,vry (T=32/64)
操作	vrz= vrx-vry
编译结果	仅存在 32 位指令。 fsub.T vrz, vrx,vry
数据类型	单精度/双精度
说明	将 vrx 和 vry 中的浮点数相减，保存至 vrz 中。
影响标志位	无
限制	寄存器范围为 vr0-vr31。
异常	ID,IO,UF,OF,IX

32 位指令	
操作	vrz= vr _x -vr _y
语法	fsub.T vr _z , vr _x ,vr _y
数据类型	单精度/双精度
说明	将 vr _x 和 vr _y 中的浮点数相减，保存至 vr _z 中。
影响标志位	无
限制	寄存器范围为 vr0-vr31。
异常	ID,IO,UF,OF,IX

32 位指令格式：

fsub.32

31	26	25	21	20	16	15	5	4	0
111101							00000000001		VRZ

fsub.64

31	26	25	21	20	16	15	5	4	0
111101							00001000001		VRZ

14.111 FXTOF.T1.T2——定点浮点转换指令

统一化指令	
语 法	fxtof.T1.T2 vrz, vrx (T1= u32/s32) (T2= f32/f64)
操 作	vrz= (float32/double)vrx
编 译 结 果	仅存在 32 位指令。 fxtof.T1.T2 vrz, vrx
数 据 类 型	32bit/有/无定点数
说 明	将 vrx 中的定点数转换浮点数，保存至 vrz 中。 FCR[20:16].frpos[4:0] 指定定点数的小数点位置。 16bit 定点数采用 frpos[3:0] 指示 1~16 位的小数部分。frpos[3:0]=0 时，说明有 1bit 小数部分。 frpos[3:0]=15 时，说明有 16bit 小数部分。 32bit 定点数采用 frpos[4:0] 指示 1~32 位的小数部分。frpos[4:0]=0 时，说明有 1bit 小数部分。 frpos[4:0]=31 时，说明有 32bit 小数部分。
影 响 标 志 位	无
限 制	寄存器范围为 vr0-vr31。
异 常	IX。

32 位指令	
操作	vrz= (float32/double)vrx
语法	fxtof.T1.T2 vrz, vrx
数据类型	32bit/有/无定点数
说明	将 vrx 中的 16bit 有符号定点数转换成浮点数，保存至 vrz 中。 FCR[20:16].frpos[4:0] 指定定点数的小数点位置。 16bit 定点数采用 frpos[3:0] 指示 1~16 位的小数部分。frpos[3:0]=0 时，说明有 1bit 小数部分。 frpos[3:0]=15 时，说明有 16bit 小数部分。
影响标志位	无
限制	寄存器范围为 vr0-vr31。
异常	IX。

32 位指令格式：

fxtof.s32.f32											
31	26	25	21	20	16	15	5	4	0		
111101		00000			VRX		01001001011			VRZ	
fxtof.u32.f32											
31	26	25	21	20	16	15	5	4	0		
111101		00000			VRX		01001001010			VRZ	
fxtof.s32.f64											
31	26	25	21	20	16	15	5	4	0		
111101		00000			VRX		01001001001			VRZ	
fxtof.u32.f64											
31	26	25	21	20	16	15	5	4	0		
111101		00000			VRX		01001001000			VRZ	

14.112 GRS——符号产生指令

统一化指令	
语法	grs rz, label grs rz, imm32
操作	$RZ \leftarrow PC + \text{sign\_extend}(\text{offset} \ll 1);$
编译结果	仅存在 32 位指令。 grs32 rz, label grs32 rz, imm32
说明	产生符号的值，该值由 label 所在位置，或 32 位立即数（IMM32）确定。符号的值由当前程序 PC 加上左移 1 位的 18 位相对偏移量有符号扩展到 32 位后的值得到。符号的值的范围是 $\pm 256\text{KB}$ 地址空间。
影响标志位	无影响
异常	无

32 位指令	
操作	$RZ \leftarrow PC + \text{sign\_extend}(\text{offset} \ll 1);$
语法	grs rz, label grs rz, imm32
说明	产生符号的值，该值由 label 所在位置，或 32 位立即数（IMM32）确定。符号的值由当前程序 PC 加上左移 1 位的 18 位相对偏移量有符号扩展到 32 位后的值得到。符号的值的范围是 $\pm 256\text{KB}$ 地址空间。
影响标志位	无影响
异常	无

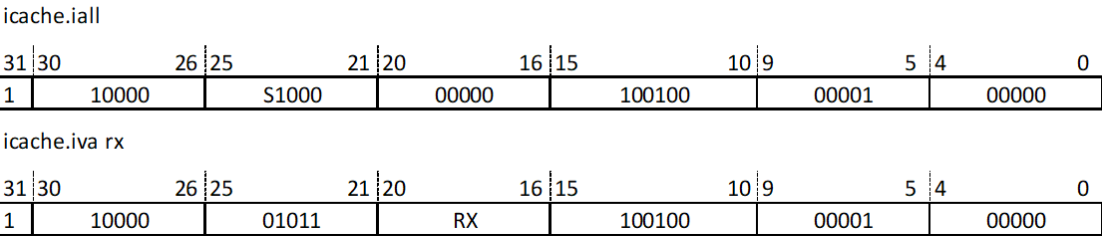
32 位指令格式：

31	30	26	25	21	20	18	17	0
1	10011		RZ		011			Offset

14.113 ICACHE——ICACHE 操作指令

统一化指令	
语法	icache.iall(s) icache.iva rx
操作	删除所有 icache 表项 删除虚拟地址对应的 icache 表项 其中，rx[31:0] 表示 VA
说明	仅存在 32 位指令。 icache32.iall(s) icache32.iva rx
说明	S 表征是否广播到其他 CPU，“1”为有效。 icache.iva 指令根据虚拟地址的共享属性判断是否广播。
影响标志位	无
异常	TLB 不可恢复异常、TLB 失配异常、TLB 无效异常

指令格式：



14.114 IDLY——中断识别禁止指令

统一化指令	
语 法	idly
操 作	禁止中断识别 4 个指令
编 译 结 果	仅存在 32 位指令 idly32
说 明	idly 指令后 4 条指令禁止中断识别，这样就允许一个不可中断指令序列在多任务环境中被执行。
影 响 标 志 位	标志位 C 在 idly 指令执行后被清零。如果在 idl y 指令执行以后的 4 个指令中发生异常（包括跟踪或断点异常），位 C 被置 1，中断指令序列就能被观测到。
限 制	idly 指令后面的指令只能是单时钟周期的算术、逻辑指令，ld, st, 或分支指令。为了使一些潜在中断的影响达到最小，如果是其他的指令，就不能保证不会被中断。如果在 idly 后面的指令序列中有另一个 idly 指令，那么它也将被忽略。但是如果是 rte, rfi, doze, wait, stop 等指令，那么他们就会引起 idly 指令序列的中止。 idly 指令不允许在一个小于 8 条指令的循环中出现。
异 常	无
备 注：	如果 idly 计数器停在不为零的状态时，中断就会被屏蔽。如果在 idly 指令序列中发生一个断点异常或一个跟踪异常，那么位 C 会被置 1，这样序列将会出现操作失败。在异常处理过程中，使中断屏蔽无效，从而让计数器清零。 idly 的计数器在使用 H AD 调试端口进行调试过程中不会变化，一旦处理器从调试模式中释放到正常的操作，那么计数就会继续。

32 位指令	
操作	禁止中断识别 4 个指令 disable_int_in_following(4);
语法	idly32
说明	idly 指令后 4 条指令禁止中断识别，这样就允许一个不可中断指令序列在多任务环境中被执行。
影响标志位	标志位 C 在 idly 指令执行后被清零。如果在 idly 指令执行以后的 4 个指令中发生异常（包括跟踪或断点异常），位 C 被置 1，中断指令序列就能被观测到。
限制	idly 指令后面的指令只能是单时钟周期的算术、逻辑指令，ld, st, 或分支指令。为了使一些潜在中断的影响达到最小，如果是其他的指令，就不能保证不会被中断。如果在 idly 后面的指令序列中有另一个 idly 指令，那么它也将被忽略。但是如果是 rte, rfi, doze, wait, stop 等指令，那么他们就会引起 idly 指令序列的中止。 idly 指令不允许在一个小于 8 条指令的循环中出现。
异常	无
备注：	如果 idly 计数器停在不为零的状态时，中断就会被屏蔽。如果在 idly 指令序列中发生一个断点异常或一个跟踪异常，那么位 C 会被置 1，这样序列将会出现操作失败。在异常处理过程中，使中断屏蔽无效，从而让计数器清零。 idly 的计数器在使用 HAD 调试端口进行调试过程中不会变化，一旦处理器从调试模式中释放到正常的操作，那么计数就会继续。

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10000	00000	00000	000111	00001	00000						

14.115 INCF——C 为 0 立即数加法指令

统一化指令	
语法	incf rz, rx, imm5
操作	if C==0, then RZ $\leftarrow$ RX + zero_extend(IMM5); else RZ $\leftarrow$ RZ;
编译结果	仅存在 32 位指令 incf32 rz, rx, imm5
说明	如果条件位 C 为 0，将 5 位立即数零扩展至 32 位，用该 32 位值加 RX 的值，把结果存在 RZ；否则，RZ 与 RX 的值不变。
影响标志位	无影响
限制	立即数的范围为 0-31。
异常	无

32 位指令	
操作	if C==0, then RZ $\leftarrow$ RX + zero_extend(IMM5); else RZ $\leftarrow$ RZ;
语法	incf32 rz, rx, imm5
说明	如果条件位 C 为 0，将 5 位立即数零扩展至 32 位，用该 32 位值加 RX 的值，把结果存在 RZ；否则，RZ 与 RX 的值不变。
影响标志位	无影响
限制	立即数的范围为 0-31。
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001				RZ	RX		000011		00001		IMM5

14.116 INCT——C 为 1 立即数加法指令

统一化指令	
语法	inct rz, rx, imm5
操作	if C==1, then RZ ← RX + zero_extend(IMM5); else RZ ← RZ;
编译结果	仅存在 32 位指令 inct32 rz, rx, imm5
说明	如果条件位 C 为 1，将 5 位立即数零扩展至 32 位，用该 32 位值加 RX 的值，把结果存在 RZ；否则，RZ 与 RX 的值不变。
影响标志位	无影响
限制	立即数的范围为 0-31。
异常	无

32 位指令	
操作	if C==1, then RZ ← RX + zero_extend(IMM5); else RZ ← RZ;
语法	inct32 rz, rx, imm5
说明	如果条件位 C 为 1，将 5 位立即数零扩展至 32 位，用该 32 位值加 RX 的值，把结果存在 RZ；否则，RZ 与 RX 的值不变。
影响标志位	无影响
限制	立即数的范围为 0-31。
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001		RZ		RX		000011		00010		IMM5	

14.117 INS——位插入指令

统一化指令	
语 法	ins rz, rx, msb, lsb
操 作	$RZ[MSB:LSB] \leftarrow RX[MSB-LSB:0]$
编 译 结 果	仅存在 32 位指令 ins32 rz, rx, msb, lsb
说 明	将 RX 的一段连续低位, 插入到由 2 个 5 位立即数 (MSB,LSB) 指定的 RZ 的一段连续位 (RZ[MSB:LSB]) 上, RZ 的其它位保持不变, RX 连续低位的宽度由 MSB 和 LSB 指定 (即 $RX[MSB-LSB:0]$)。如果 MSB 等于 31, 且 LSB 等于 0, 则 RZ 的值与 RX 相同。如果 MSB 等于 LSB, 则 RZ 的 MSB (即 LSB) 位为 RX 的最低位, 其它位保持不变。如果 MSB 小于 LSB, 该指令的行为不可预测。
影 响 标 志 位	无影响
限 制	MSB 的范围为 0-31, LSB 的范围为 0-31, 且 MSB 应当大于等于 LSB。
异 常	无

32 位指令	
操作	$RZ[MSB:LSB] \leftarrow RX[MSB-LSB:0]$
语法	ins32 rz, rx, msb, lsb
说明	将 RX 的一段连续低位, 插入到由 2 个 5 位立即数 (MSB,LSB) 指定的 RZ 的一段连续位 (RZ[MSB:LSB]) 上, RZ 的其它位保持不变, RX 连续低位的宽度由 MSB 和 LSB 指定 (即 $RX[MSB-LSB:0]$)。如果 MSB 等于 31, 且 LSB 等于 0, 则 RZ 的值与 RX 相同。如果 MSB 等于 LSB, 则 RZ 的 MSB (即 LSB) 位为 RX 的最低位, 其它位保持不变。如果 MSB 小于 LSB, 该指令的行为不可预测。
影响标志位	无影响
限制	MSB 的范围为 0-31, LSB 的范围为 0-31, 且 MSB 应当大于等于 LSB。
异常	无

32 位指令格式:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001		RZ		RX		010111		SIZE		LSB	

SIZE 域:

指定插入位的宽度。

注意: 二进制操作数 SIZE 等于 MSB-LSB。

00000:

1

00001:

2

....

11111:

32

LSB 域:

指定插入结束的位。

00000:

0 位
00001:
1 位
.....
11111:
31 位

14.118 IXH——索引半字指令

统一化指令	
语法	ixh rz, rx, ry
操作	$RZ \leftarrow RX + (RY \ll 1)$
编译结果	仅存在 32 位指令 ixh32 rz, rx, ry
说明	将 RY 的值左移一位后加上 RX 的值，并把结果存入 RZ。
影响标志位	无影响
异常	无

32 位指令	
操作	$RZ \leftarrow RX + (RY \ll 1)$
语法	ixh32 rz, rx, ry
说明	将 RY 的值左移一位后加上 RX 的值，并把结果存入 RZ。
影响标志位	无影响
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001		RY		RX		000010		00001		RZ	

14.119 IXW——索引字指令

统一化指令	
语法	ixw rz, rx, ry
操作	$RZ \leftarrow RX + (RY \ll 2)$
编译结果	仅存在 32 位指令 ixw32 rz, rx, ry
说明	将 RY 的值左移两位后加上 RX 的值，并把结果存入 RZ。
影响标志位	无影响
异常	无

32 位指令	
操作	$RZ \leftarrow RX + (RY \ll 2)$
语法	ixw32 rz, rx, ry
说明	将 RY 的值左移两位后加上 RX 的值，并把结果存入 RZ。
影响标志位	无影响
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001		RY		RX		000010		00010		RZ	

14.120 IXD——索引双字指令

统一化指令	
语法	ixd rz, rx, ry
操作	$RZ \leftarrow RX + (RY \ll 3)$
编译结果	仅存在 32 位指令 ixd32 rz, rx, ry
说明	将 RY 的值左移三位后加上 RX 的值，并把结果存入 RZ。
影响标志位	无影响
异常	无

32 位指令	
操作	$RZ \leftarrow RX + (RY \ll 3)$
语法	ixd32 rz, rx, ry
说明	将 RY 的值左移三位后加上 RX 的值，并把结果存入 RZ。
影响标志位	无影响
异常	无

32 位指令格式：

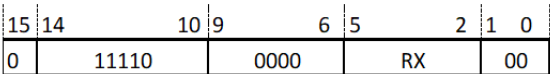
31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001		RY		RX		000010		00100		RZ	

14.121 JMP——寄存器跳转指令

统一化指令	
语法	jmp rx
操作	跳转到寄存器指定的位置 $PC \leftarrow RX \& 0\text{xffffffe}$
编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (x<16), then jmp16 rx; else jmp32 rx;
说明	程序跳转到寄存器 RX 指定的位置，RX 的最低位被忽略。JMP 指令的跳转范围是全部 4GB 地址空间。
影响标志位	无影响
异常	无

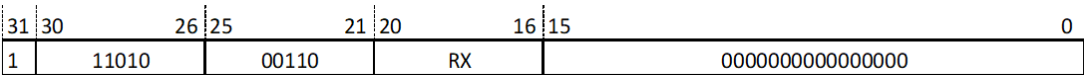
16 位指令	
操作	跳转到寄存器指定的位置 $PC \leftarrow RX \& 0\text{xffffffe}$
语法	jmp16 rx
说明	程序跳转到寄存器 RX 指定的位置，RX 的最低位被忽略。JMP 指令的跳转范围是全部 4GB 地址空间。
影响标志位	无影响
异常	无

16 位指令格式：



32 位指令	
操作	跳转到寄存器指定的位置 $PC \leftarrow RX \ \& \ 0\text{xffffffe}$
语法	jmp32 rx
说明	程序跳转到寄存器 RX 指定的位置，RX 的最低位被忽略。JMP 指令的跳转范围是全部 4GB 地址空间。
影响标志位	无影响
异常	无

32 位指令格式：



14.122 JMPI——间接跳转指令

统一化指令	
语法	jmp _i label
操作	程序跳转到存储器指定的位置 $PC \leftarrow \text{MEM}[(PC + \text{zero\_extend}(\text{offset} \ll 2)) \ \& \ 0\text{xffffffc}]$
编译结果	仅存在 32 位指令。 jmp _i 32 label
说明	程序跳转到 label 所在的位置，label 由存储器加载得到。存储器地址根据 PC 加上左移两位的 16 位相对偏移量无符号扩展到 32 位后，再经最低两位强制清零得到。JMPI 指令的跳转范围是全部 4GB 地址空间。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	程序跳转到存储器指定的位置 $PC \leftarrow MEM[(PC + zero\_extend(offset \ll 2)) \& 0xffffffffc]$
语法	jmp _i 32 label
说明	程序跳转到 label 所在的位置，label 由存储器加载得到。存储器地址根据 PC 加上左移两位的 16 位相对偏移量无符号扩展到 32 位后，再经最低两位强制清零得到。JMPI 指令的跳转范围是全部 4GB 地址空间。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

31	30	26	25	21	20	16	15	0
1	11010	10110	00000	Offset				

14.123 JSR——寄存器跳转到子程序指令

统一化指令	
语法	jsr rx
操作	链接并跳转到寄存器指定的子程序位置 $R15 \leftarrow PC + 4,$ $PC \leftarrow RX \& 0xffffffffe$
编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (x<16), then jsr ₁₆ rx; else jsr ₃₂ rx;
说明	子程序寄存器跳转，将子程序的返回地址（下一条指令的 PC，即当前 PC+4）保存在链接寄存器 R15 中，程序跳转到寄存器 RX 的内容指定的子程序位置处执行，RX 的最低位被忽略。JSR 指令的跳转范围是全部 4GB 地址空间。
影响标志位	无影响
异常	无

16 位指令	
操作	链接并跳转到寄存器指定的子程序位置 $R15 \leftarrow PC + 4, PC \leftarrow RX \& 0\text{xffffffe}$
语法	jsr16 rx
说明	子程序寄存器跳转，将子程序的返回地址（下一条指令的 PC，即当前 PC+4）保存在链接寄存器 R15 中，程序跳转到寄存器 RX 的内容指定的子程序位置处执行，RX 的最低位被忽略。JSR 指令的跳转范围是全部 4GB 地址空间。
影响标志位	无影响
异常	无

16 位指令格式：

15	14	10	9	6	5	2	1	0
0	11110			0000		RX		01

32 位指令	
操作	链接并跳转到寄存器指定的子程序位置 $R15 \leftarrow PC + 4, PC \leftarrow RX \& 0\text{xffffffe}$
语法	jsr32 rx
说明	子程序寄存器跳转，将子程序的返回地址（下一条指令的 PC，即当前 PC+4）保存在链接寄存器 R15 中，程序跳转到寄存器 RX 的内容指定的子程序位置处执行，RX 的最低位被忽略。JSR 指令的跳转范围是全部 4GB 地址空间。
影响标志位	无影响
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	0
1	11010			00111		RX		0000000000000000

14.124 JSRI——间接跳转到子程序指令

统一化指令	
语 法	jsri label
操 作	程序跳转到存储器指定的子程序位置 $R15 \leftarrow \text{next PC},$ $PC \leftarrow \text{MEM}[(PC + \text{zero\_extend}(\text{offset} \ll 2)) \& 0\text{xffffffffc}]$
编 译 结 果	仅存在 32 位指令。 jsri32 label;
说 明	子程序间接跳转，将子程序的返回地址（下一条指令的 PC）保存在链接寄存器 R15 中，程序跳转到 label 所在的位置处执行，label 由存储器加载得到。存储器地址根据 PC 加上左移两位的 16 位相对偏移量无符号扩展到 32 位后，再经最低两位强制清零得到。JSRI 指令的跳转范围是全部 4GB 地址空间。
影 响 标 志 位	无影响
异 常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	程序跳转到存储器指定的子程序位置 $R15 \leftarrow PC + 4, PC \leftarrow MEM[(PC + zero\_extend(offset \ll 2)) \& 0xffffffffc]$
语法	jsri32 label
说明	子程序间接跳转，将子程序的返回地址（下一条指令的 PC，即当前 PC+4）保存在链接寄存器 R15 中，程序跳转到 label 所在的位置处执行，label 由存储器加载得到。存储器地址根据 PC 加上左移两位的 16 位相对偏移量无符号扩展到 32 位后，再经最低两位强制清零得到。JSRI 指令的跳转范围是全部 4GB 地址空间。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

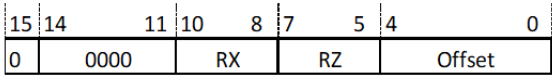
31	30	26	25	21	20	16	15	0
1	11010	10111	00000	Offset				

14.125 LD.B——无符号扩展字节加载指令

统一化指令	
语 法	ld.b rz,(rx, disp)
操 作	$RZ \leftarrow \text{zero\_extend}(\text{MEM}[RX + \text{zero\_extend}(\text{offset})])$
编 译 结 果	根据偏移量和寄存器的范围编译为对应的 16 位或 32 位指令。 if (disp<32)and(x<7) and (z<7), then ld16.b rz, (rx, disp); else ld32.b rz, (rx, disp);
说 明	从存储器加载的字节经零扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上 12 位相对偏移量无符号扩展到 32 位后的值得到。LD.B 指令可以寻址 +4KB 地址空间。 注意，偏移量 DISP 即二进制操作数 Offset。
影 响 标 志 位	无影响
异 常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

16 位指令	
操作	从存储器加载字节到寄存器，无符号扩展 $RZ \leftarrow \text{zero\_extend}(\text{MEM}[\text{RX} + \text{zero\_extend}(\text{offset})])$
语法	ld16.b rz, (rx, disp)
说明	从存储器加载的字节经零扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上 5 位相对偏移量无符号扩展到 32 位后的值得到。LD16.B 指令可以寻址 +32B 的地址空间。 注意，偏移量 DISP 即二进制操作数 Offset。
影响标志位	无影响
限制	寄存器的范围为 r0-r7。
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

16 位指令格式：



32 位指令	
操作	从存储器加载字节到寄存器，无符号扩展 $RZ \leftarrow \text{zero\_extend}(\text{MEM}[\text{RX} + \text{zero\_extend}(\text{offset})])$
语法	ld32.b rz, (rx, disp)
说明	从存储器加载的字节经零扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上 12 位相对偏移量无符号扩展到 32 位后的值得到。LD32.B 指令可以寻址 +4KB 地址空间。 注意，偏移量 DISP 即二进制操作数 Offset。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

31	30	26	25	21	20	16	15	12	11	0
1	10110	RZ		RX		0000		Offset		

14.126 LD.BS——有符号扩展字节加载指令

统一化指令	
语 法	ld.bs rz, (rx, disp)
操 作	$RZ \leftarrow \text{sign\_extend}(\text{MEM}[RX + \text{zero\_extend}(\text{offset})])$
编 译 结 果	仅存在 32 位指令。 ld32.bs rz, (rx, disp)
说 明	从存储器加载的字节经有符号扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上 12 位相对偏移量无符号扩展到 32 位后的值得到。LD.BS 指令可以寻址 +4KB 地址空间。 注意，偏移量 DISP 即二进制操作数 Offset。
影 响 标 志 位	无影响
异 常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	从存储器加载字节到寄存器，有符号扩展 $RZ \leftarrow \text{sign\_extend}(\text{MEM}[\text{RX} + \text{zero\_extend}(\text{offset})])$
语法	ld32.bs rz, (rx, disp)
说明	从存储器加载的字节经有符号扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上 12 位相对偏移量无符号扩展到 32 位后的值得到。LD32.BS 指令可以寻址 +4KB 地址空间。 注意，偏移量 DISP 即二进制操作数 Offset。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

31	30	26	25	21	20	16	15	12	11	0
1	10110		RZ		RX		0100		Offset	

14.127 LD.D——双字加载指令

统一化指令	
语 法	ld.d rz, (rx, disp)
操 作	$RZ \leftarrow MEM[RX + zero\_extend(offset \ll 2)]$ $RZ + 1 \leftarrow MEM[RX + zero\_extend(offset \ll 2) + 0x4]$
编 译 结 果	仅存在 32 位指令。 ld32.d rz, (rx, disp);
说 明	从存储器加载双字到寄存器 RZ、RZ + 1 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移两位的 12 位相对偏移量无符号扩展到 32 位后的值得到。LD.D 指令可以寻址 +16KB 地址空间。 注意：1. 偏移量 DISP 是二进制操作数 Offset 左移两位得到的。 2.rz 和 rx 不可以相同，否则结果将不可预期。
影 响 标 志 位	无影响
异 常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	从存储器加载双字到寄存器 $RZ \leftarrow \text{MEM}[RX + \text{zero\_extend}(\text{offset} \ll 2)]$ $RZ + 1 \leftarrow \text{MEM}[RX + \text{zero\_extend}(\text{offset} \ll 2) + 0x4]$
语法	ld32.d rz, (rx, disp)
说明	从存储器加载字到寄存器 RZ、RZ + 1 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移两位的 12 位相对偏移量无符号扩展到 32 位后的值得到。LD32.D 指令可以寻址 +16KB 地址空间。 注意： 1. 偏移量 DISP 是二进制操作数 Offset 左移两位得到的。 2.rz 和 rx 不可以相同，否则结果将不可预期。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

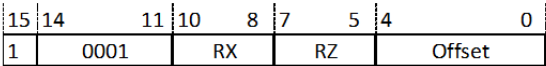
31	30	26	25	21	20	16	15	12	11	0
1	10110		RZ		RX		0011		Offset	

14.128 LD.H——无符号扩展半字加载指令

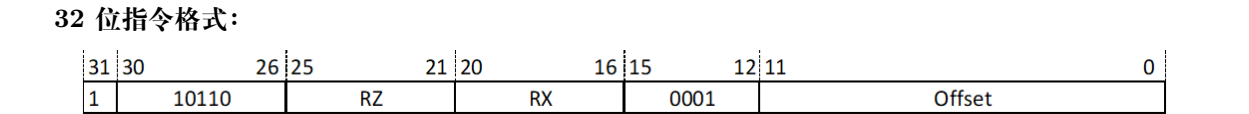
统一化指令	
语 法	ld.h rz, (rx, disp)
操 作	$RZ \leftarrow \text{zero\_extend}(\text{MEM}[\text{RX} + \text{zero\_extend}(\text{offset} \ll 1)])$
编 译 结 果	根据偏移量和寄存器的范围编译为对应的 16 位或 32 位指令。 if (disp<64)and(x<7) and (z<7), then ld16.h rz, (rx, disp); else ld32.h rz, (rx, disp);
说 明	从存储器加载的半字经零扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移 1 位的 12 位相对偏移量无符号扩展到 32 位后的值得到。LD.H 指令可以寻址 +8KB 地址空间。 注意：偏移量 DISP 是二进制操作数 Offset 左移 1 位得到的。
影 响 标 志 位	无影响
异 常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

16 位指令	
操作	从存储器加载半字到寄存器，无符号扩展 $RZ \leftarrow \text{zero\_extend}(\text{MEM}[\text{RX} + \text{zero\_extend}(\text{offset} \ll 1)])$
语法	ld16.h rz, (rx, disp)
说明	从存储器加载的半字经零扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移 1 位的 5 位相对偏移量无符号扩展到 32 位后的值得到。LD16.H 指令可以寻址 +64B 的地址空间。 注意，偏移量 DISP 是二进制操作数 Offset 左移 1 位得到的。
影响标志位	无影响
限制	寄存器的范围为 r0-r7。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

16 位指令格式：



32 位指令	
操作	从存储器加载半字到寄存器，无符号扩展 $RZ \leftarrow \text{zero\_extend}(\text{MEM}[\text{RX} + \text{zero\_extend}(\text{offset} \ll 1)])$
语法	ld32.h rz, (rx, disp)
说明	从存储器加载的半字经零扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移 1 位的 12 位相对偏移量无符号扩展到 32 位后的值得到。LD32.H 指令可以寻址 +8KB 地址空间。 注意，偏移量 DISP 是二进制操作数 Offset 左移 1 位得到的。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常



14.129 LD.HS——有符号扩展半字加载指令

统一化指令	
语 法	ld.hs rz, (rx, disp)
操 作	$RZ \leftarrow \text{sign\_extend}(\text{MEM}[\text{RX} + \text{zero\_extend}(\text{offset} \ll 1)])$
编 译 结 果	仅存在 32 位指令。 ld32.hs rz, (rx, disp)
说 明	从存储器加载的半字经有符号扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移 1 位的 12 位相对偏移量无符号扩展到 32 位后的值得到。LD.HS 指令可以寻址 +8KB 地址空间。 注意，偏移量 DISP 是二进制操作数 Offset 左移 1 位得到的。
影 响 标 志 位	无影响
异 常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	从存储器加载半字到寄存器，有符号扩展 $RZ \leftarrow \text{sign\_extend}(\text{MEM}[\text{RX} + \text{zero\_extend}(\text{offset} \ll 1)])$
语法	ld32.hs rz, (rx, disp)
说明	从存储器加载的半字经有符号扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移 1 位的 12 位相对偏移量无符号扩展到 32 位后的值得到。LD32.HS 指令可以寻址 +8KB 地址空间。 注意，偏移量 DISP 是二进制操作数 Offset 左移 1 位得到的。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

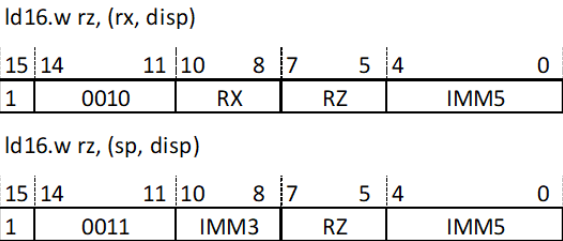
31	30	26	25	21	20	16	15	12	11	0
1	10110		RZ		RX		0101		Offset	

14.130 LD.W——字加载指令

统一化指令	
语 法	ld.w rz, (rx, disp)
操 作	$RZ \leftarrow MEM[RX + zero\_extend(offset \ll 2)]$
编 译 结 果	根据偏移量和寄存器的范围编译为对应的 16 位或 32 位指令。 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);
说 明	从存储器加载字到寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移两位的 12 位相对偏移量无符号扩展到 32 位后的值得到。LD.W 指令可以寻址 +16KB 地址空间。 注意，偏移量 DISP 是二进制操作数 Offset 左移两位得到的。
影 响 标 志 位	无影响
异 常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

16 位指令	
操作	从存储器加载字到寄存器 $RZ \leftarrow \text{MEM}[RX + \text{sign\_extend}(\text{offset} \ll 2)]$
语法	ld16.w rz, (rx, disp) ld16.w rz, (sp, disp)
说明	从存储器加载字到寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。当 RX 为 SP 时，存储器的有效地址由基址寄存器 RX 加上左移 2 位的 8 位相对偏移量无符号扩展到 32 位后的值得到。当 rx 为其它寄存器时，存储器的有效地址由基址寄存器 RX 加上左移 2 位的 5 位相对偏移量无符号扩展到 32 位后的值得到。LD16.W 指令可以寻址 +1KB 的地址空间。 注意，偏移量 DISP 是二进制操作数 IMM5 左移两位得到的。当基址寄存器 RX 为 SP 时，偏移量 DISP 是二进制操作数 {IMM3, IMM5} 左移两位得到的。
影响标志位	无影响
限制	寄存器的范围为 r0-r7。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常。

16 位指令格式：



32 位指令	
操作	从存储器加载字到寄存器 $RZ \leftarrow \text{MEM}[RX + \text{zero\_extend}(\text{offset} \ll 2)]$
语法	ld32.w rz, (rx, disp)
说明	从存储器加载字到寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移两位的 12 位相对偏移量无符号扩展到 32 位后的值得到。LD32.W 指令可以寻址 +16KB 地址空间。 注意，偏移量 DISP 是二进制操作数 Offset 左移两位得到的。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

31	30	26	25	21	20	16	15	12	11	0
1	10110		RZ		RX		0010		Offset	

14.131 LDEX.W——独占式字加载指令

统一化指令	
语 法	ldex.w rz, (rx, disp)
操 作	$RZ \leftarrow \text{MEM}[RX + \text{zero\_extend}(\text{offset} \ll 2)]$
编 译 结 果	仅存在 32 位指令。 ldex32.w rz, (rx, disp)
说 明	从存储器加载字到通用寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移两位的 12 位相对偏移量无符号扩展到 32 位后的值得到。LDEX.W 指令可以寻址 +16KB 地址空间。 该指令与 STEX.W 配对，用于多核通信时的原子“读存储器—修改—写存储器”操作。 注意，偏移量 DISP 是二进制操作数 Offset 左移两位得到的。
影 响 标 志 位	无影响
异 常	未对齐访问异常、未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	从存储器加载字到通用寄存器 $RZ \leftarrow \text{MEM}[RX + \text{zero\_extend}(\text{offset} \ll 2)]$
语法	ldex32.w rz, (rx, disp)
说明	从存储器加载字到通用寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移两位的 12 位相对偏移量无符号扩展到 32 位后的值得到。LDEX32.W 指令可以寻址 +16KB 地址空间。 该指令与 STEX32.W 配对，用于多核通信时的原子“读存储器—修改—写存储器”操作。 注意，偏移量 DISP 是二进制操作数 Offset 左移两位得到的。
影响标志位	无影响
异常	未对齐访问异常、未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

31	30	26	25	21	20	16	15	12	11	0
1	10110		RZ		RX		0111		Offset	

14.132 LDM——连续多字加载指令

统一化指令	
语 法	ldm ry-rz, (rx)
操 作	从存储器加载连续的多个字到一片连续的寄存器堆中 dst ← Y; addr ← RX; for (n = 0; n <= (Z-Y); n++){ Rdst ← MEM[addr]; dst ← dst + 1; addr ← addr + 4; }
编 译 结 果	仅存在 32 位指令。 ldm32 ry-rz, (rx);
说 明	从存储器依次加载连续的多个字到寄存器 RY 开始的一片连续寄存器堆中，即将存储器指定地址开始的第一个字加载到寄存器 RY 中，第二个字加载到寄存器 RY+1 中，依次类推，最后一个字加载到寄存器 RZ 中。存储器的有效地址由基址寄存器 RX 的内容决定。
影 响 标 志 位	无影响
限 制	RZ 应当大于等于 RY。 RY-RZ 范围内不应该包含基址寄存器 RX，否则结果不可预测。
异 常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	从存储器加载连续的多个字到一片连续的寄存器堆中 $dst \leftarrow Y; addr \leftarrow RX;$ for ($n = 0; n \leq IMM5; n++$) { $Rdst \leftarrow MEM[addr];$ $dst \leftarrow dst + 1;$ $addr \leftarrow addr + 4;$ }
语法	ldm32 ry-rz, (rx)
说明	从存储器依次加载连续的多个字到寄存器 RY 开始的一片连续寄存器堆中，即将存储器指定地址开始的第一个字加载到寄存器 RY 中，第二个字加载到寄存器 RY+1 中，依次类推，最后一个字加载到寄存器 RZ 中。存储器的有效地址由基址寄存器 RX 的内容决定。
影响标志位	无影响
限制	RZ 应当大于等于 RY。 RY-RZ 范围内不应该包含基址寄存器 RX，否则结果不可预测。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10100	RZ		RX		000111		00001		IMM5		

IMM5 域：

指定目标寄存器的个数， $IMM5 = Z - Y$ 。

00000：

1 个目的寄存器

00001：

2 个目的寄存器

....

11111：

32 个目的寄存器

14.133 LDQ——连续四字加载指令

统一化指令	
语 法	ldq r4-r7, (rx)
操 作	从存储器加载连续的四个字到寄存器 R4—R7 中 dst ← 4; addr ← RX; for (n = 0; n <= 3; n++){ Rdst ← MEM[addr]; dst ← dst + 1; addr ← addr + 4; }
编 译 结 果	仅存在 32 位指令。 ldq32 r4-r7, (rx);
说 明	从存储器依次加载连续的 4 个字到寄存器堆 [R4, R7]（包括边界）中，即将存储器指定地址开始的第一个字加载到寄存器 R4 中，第二个字加载到寄存器 R5 中，第三个字加载到寄存器 R6 中，第四个字加载到寄存器 R7 中。存储器的有效地址由基址寄存器 RX 的内容决定。 注意，该指令是 ldm r4-r7, (rx) 的伪指令。
影 响 标 志 位	无影响
限 制	R4-R7 范围内不应该包含基址寄存器 RX，否则结果不可预测。
异 常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	从存储器加载连续的四个字到寄存器 R4—R7 中 $dst \leftarrow 4; addr \leftarrow RX;$ for (n = 0; n <= 3; n++){ $Rdst \leftarrow MEM[addr];$ $dst \leftarrow dst + 1;$ $addr \leftarrow addr + 4;$ }
语法	ldq32 r4-r7, (rx)
说明	从存储器依次加载连续的 4 个字到寄存器堆 [R4, R7]（包括边界）中，即将存储器指定地址开始的第一个字加载到寄存器 R4 中，第二个字加载到寄存器 R5 中，第三个字加载到寄存器 R6 中，第四个字加载到寄存器 R7 中。存储器的有效地址由基址寄存器 RX 的内容决定。 注意：该指令是 ldm32 r4-r7, (rx) 的伪指令。
影响标志位	无影响
限制	R4-R7 范围内不应该包含基址寄存器 RX，否则结果不可预测。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10100	00100		RX		000111		00001		00011		

14.134 LDR.B——寄存器移位寻址无符号扩展字节加载指令

统一化指令	
语法	ldr.b rz, (rx, ry << 0) ldr.b rz, (rx, ry << 1) ldr.b rz, (rx, ry << 2) ldr.b rz, (rx, ry << 3)
操作	从存储器加载字节到寄存器，无符号扩展 $RZ \leftarrow \text{zero\_extend}(\text{MEM}[\text{RX} + \text{RY} \ll \text{IMM2}])$
编译结果	仅存在 32 位指令。 ldr32.b rz, (rx, ry << 0) ldr32.b rz, (rx, ry << 1) ldr32.b rz, (rx, ry << 2) ldr32.b rz, (rx, ry << 3)
说明	从存储器加载的字节经零扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加寄存器移位的寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值为 0。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	从存储器加载字节到寄存器，无符号扩展 $RZ \leftarrow \text{zero\_extend}(\text{MEM}[\text{RX} + \text{RY} \ll \text{IMM2}])$
语法	ldr32.b rz, (rx, ry << 0) ldr32.b rz, (rx, ry << 1) ldr32.b rz, (rx, ry << 2) ldr32.b rz, (rx, ry << 3)
说明	从存储器加载的字节经零扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加寄存器移位的寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值为 0。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

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						

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						

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						

ldr32.b rz, (rx, ry << 3)

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10100	RY	RX	000000	01000	RZ						

14.135 LDR.D——寄存器移位寻址双字加载指令

统一化指令	
语 法	ldr.d rz, (rx, ry << 0) ldr.d rz, (rx, ry << 1) ldr.d rz, (rx, ry << 2) ldr.d rz, (rx, ry << 3)
操 作	从存储器加载双字到寄存器 $RZ \leftarrow \text{MEM}[RX + \text{zero\_extend}(\text{offset} \ll 2)]$ $RZ + 1 \leftarrow \text{MEM}[RX + \text{zero\_extend}(\text{offset} \ll 2) + 0x4]$
编 译 结 果	仅存在 32 位指令。 ldr32.d rz, (rx, ry << 0) ldr32.d rz, (rx, ry << 1) ldr32.d rz, (rx, ry << 2) ldr32.d rz, (rx, ry << 3)
说 明	从存储器加载双字到寄存器 RZ 和 RZ+1 中。采用寄存器加寄存器移位的寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值为 0。 注意：rz、rz+1、rx 和 ry 不能相同，否则结果不可预期。
影 响 标 志 位	无影响
异 常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	从存储器加载双字到寄存器
语法	ldr32.d rz, (rx, ry << 0) ldr32.d rz, (rx, ry << 1) ldr32.d rz, (rx, ry << 2) ldr32.d rz, (rx, ry << 3)
说明	从存储器加载双字到寄存器 RZ 和 RZ+1 中。采用寄存器加寄存器移位的寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值为 0。 注意：rz、rz+1、rx 和 ry 不能相同，否则结果不可预期。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

ldr32.d rz, (rx, ry << 0)

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10100	RY	RX	000011	00001	RZ						

ldr32.d rz, (rx, ry << 1)

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10100	RY	RX	000011	00010	RZ						

ldr32.d rz, (rx, ry << 2)

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10100	RY	RX	000011	00100	RZ						

ldr32.d rz, (rx, ry << 3)

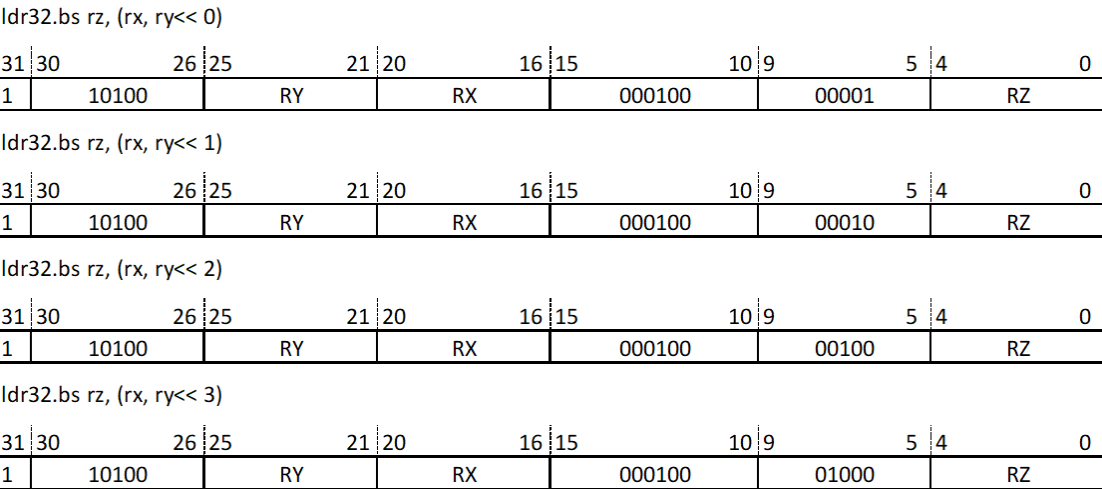
31	30	26	25	21	20	16	15	10	9	5	4	0
1	10100	RY	RX	000011	01000	RZ						

14.136 LDR.BS——寄存器移位寻址有符号扩展字节加载指令

统一化指令	
语法	ldr.bs rz, (rx, ry << 0) ldr.bs rz, (rx, ry << 1) ldr.bs rz, (rx, ry << 2) ldr.bs rz, (rx, ry << 3)
操作	从存储器加载字节到寄存器，有符号扩展 $RZ \leftarrow \text{sign\_extend}(\text{MEM}[RX + RY \ll IMM2])$
编译结果	仅存在 32 位指令。 ldr32.bs rz, (rx, ry << 0) ldr32.bs rz, (rx, ry << 1) ldr32.bs rz, (rx, ry << 2) ldr32.bs rz, (rx, ry << 3)
说明	从存储器加载的字节经有符号扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加寄存器移位的寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值为 0。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	从存储器加载字节到寄存器，有符号扩展 $RZ \leftarrow \text{sign\_extend}(\text{MEM}[RX + RY \ll IMM2])$
语法	ldr32.bs rz, (rx, ry << 0) ldr32.bs rz, (rx, ry << 1) ldr32.bs rz, (rx, ry << 2) ldr32.bs rz, (rx, ry << 3)
说明	从存储器加载的字节经有符号扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加寄存器移位的寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值为 0。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：



14.137 LDR.H——寄存器移位寻址无符号扩展半字加载指令

统一化指令	
语法	ldr.h rz, (rx, ry << 0) ldr.h rz, (rx, ry << 1) ldr.h rz, (rx, ry << 2) ldr.h rz, (rx, ry << 3)
操作	从存储器加载半字到寄存器，无符号扩展。 RZ ← zero_extend(MEM[RX + RY << IMM2])
编译结果	仅存在 32 位指令。 ldr32.h rz, (rx, ry << 0) ldr32.h rz, (rx, ry << 1) ldr32.h rz, (rx, ry << 2) ldr32.h rz, (rx, ry << 3)
说明	从存储器加载的半字经零扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加寄存器移位的寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值为 0。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	从存储器加载半字到寄存器，无符号扩展 $RZ \leftarrow \text{zero\_extend}(\text{MEM}[RX + RY \ll IMM2])$
语法	ldr32.h rz, (rx, ry << 0) ldr32.h rz, (rx, ry << 1) ldr32.h rz, (rx, ry << 2) ldr32.h rz, (rx, ry << 3)
说明	从存储器加载的半字经零扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加寄存器移位的寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值为 0。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

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						

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						

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						

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						

14.138 LDR.HS——寄存器移位寻址有符号扩展半字加载指令

统一化指令	
语法	ldr.hs rz, (rx, ry << 0) ldr.hs rz, (rx, ry << 1) ldr.hs rz, (rx, ry << 2) ldr.hs rz, (rx, ry << 3)
操作	从存储器加载半字到寄存器，有符号扩展 $RZ \leftarrow \text{sign\_extend}(\text{MEM}[RX + RY \ll IMM2])$
编译结果	仅存在 32 位指令。 ldr32.hs rz, (rx, ry << 0) ldr32.hs rz, (rx, ry << 1) ldr32.hs rz, (rx, ry << 2) ldr32.hs rz, (rx, ry << 3)
说明	从存储器加载的半字经有符号扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加寄存器移位的寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值为 0。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	从存储器加载半字到寄存器，有符号扩展 $RZ \leftarrow \text{sign\_extend}(\text{MEM}[RX + RY \ll IMM2])$
语法	ldr32.hs rz, (rx, ry << 0) ldr32.hs rz, (rx, ry << 1) ldr32.hs rz, (rx, ry << 2) ldr32.hs rz, (rx, ry << 3)
说明	从存储器加载的半字经有符号扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加寄存器移位的寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值为 0。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

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	
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	
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	
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	

14.139 LDR.W——寄存器移位寻址字加载指令

统一化指令	
语法	ldr.w rz, (rx, ry << 0) ldr.w rz, (rx, ry << 1) ldr.w rz, (rx, ry << 2) ldr.w rz, (rx, ry << 3)
操作	从存储器加载字到寄存器 $RZ \leftarrow \text{MEM}[RX + RY \ll IMM2]$
编译结果	仅存在 32 位指令。 ldr32.w rz, (rx, ry << 0) ldr32.w rz, (rx, ry << 1) ldr32.w rz, (rx, ry << 2) ldr32.w rz, (rx, ry << 3)
说明	从存储器加载字到寄存器 RZ 中。采用寄存器加寄存器移位的寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值为 0。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	从存储器加载字到寄存器 $RZ \leftarrow \text{MEM}[RX + RY \ll \text{IMM2}]$
语法	$\text{ldr32.w rz, (rx, ry} \ll 0)$ $\text{ldr32.w rz, (rx, ry} \ll 1)$ $\text{ldr32.w rz, (rx, ry} \ll 2)$ $\text{ldr32.w rz, (rx, ry} \ll 3)$
说明	从存储器加载字到寄存器 RZ 中。采用寄存器加寄存器移位的寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值为 0。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

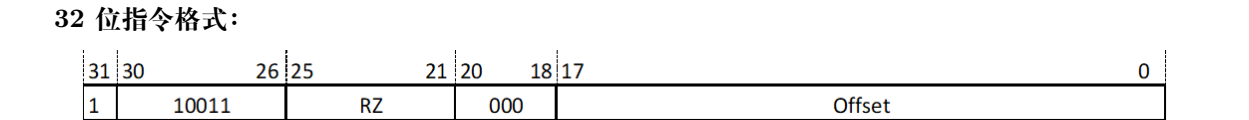
32 位指令格式：

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
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
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
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

14.140 LRS.B——字节符号加载指令

统一化指令	
语 法	lrs.b rz, [label]
操 作	从存储器加载字节到寄存器 $RZ \leftarrow \text{zero\_extend}(\text{MEM}[R28 + \text{zero\_extend}(\text{offset})])$
编 译 结 果	仅存在 32 位指令。 lrs32.b rz, [label]
说 明	加载 label 所在位置的字节符号，经零扩展到 32 位后存放于目的寄存器 RZ。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 R28 加上 18 位相对偏移量无符号扩展到 32 位后的值得到。LRS.B 指令可以寻址 +256KB 地址空间。 注意：偏移量 DISP 即二进制操作数 Offset。
影 响 标 志 位	无影响
异 常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操 作	从存储器加载字节符号到寄存器，无符号扩展 $RZ \leftarrow \text{zero\_extend}(\text{MEM}[R28 + \text{zero\_extend}(\text{offset})])$
语 法	lrs32.b rz, [label]
说 明	加载 label 所在位置的字节符号，经零扩展到 32 位后存放于目的寄存器 RZ。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 R28 加上 18 位相对偏移量无符号扩展到 32 位后的值得到。LRS.B 指令可以寻址 +256KB 地址空间。 注意：偏移量 DISP 即二进制操作数 Offset。
影 响 标 志 位	无影响
异 常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常



14.141 LRS.H——半字符号加载指令

统一化指令	
语 法	lrs.h rz, [label]
操 作	RZ ← zero_extend(MEM[R28 + zero_extend(offset << 1)])
编 译 结 果	仅存在 32 位指令。 lrs32.h rz, [label]
说 明	加载 label 所在位置的半字符号，经零扩展到 32 位后存放于目的寄存器 RZ。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 R2 8 加上左移 1 位的 18 位相对偏移量无符号扩展到 32 位后的值得到。LRS.H 指令可以寻址 +512KB 地址空间。 注意，偏移量 DISP 即二进制操作数 Offset。
影 响 标 志 位	无影响
异 常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	从存储器加载半字符到寄存器，无符号扩展 $RZ \leftarrow \text{zero\_extend}(\text{MEM}[\text{R28} + \text{zero\_extend}(\text{offset} \ll 1)])$
语法	lrs32.h rz, [label]
说明	加载 label 所在位置的半字符，经零扩展到 32 位后存放于目的寄存器 RZ。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 R2 8 加上左移 1 位的 18 位相对偏移量无符号扩展到 32 位后的值得到。LRS.H 指令可以寻址 +512KB 地址空间。 注意，偏移量 DISP 即二进制操作数 Offset。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

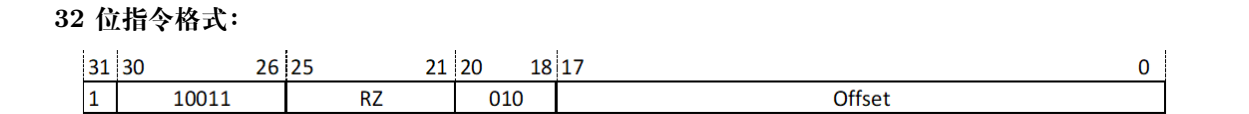
32 位指令格式：

31	30	26	25	21	20	18	17	0
1	10011		RZ		001			Offset

14.142 LRS.W——字符符号加载指令

统一化指令	
语 法	lrs.w rz, [label]
操 作	$RZ \leftarrow \text{zero\_extend}(\text{MEM}[R28 + \text{zero\_extend}(\text{offset} \ll 2)])$
编 译 结 果	仅存在 32 位指令。 lrs32.w rz, [label]
说 明	加载 label 所在位置的字符符号，经零扩展到 32 位后存放于目的寄存器 RZ。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 R2 8 加上左移 2 位的 18 位相对偏移量无符号扩展到 32 位后的值得到。LRS.W 指令可以寻址 +1024KB 地址空间。 注意，偏移量 DISP 即二进制操作数 Offset。
影 响 标 志 位	无影响
异 常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操 作	从存储器加载字符符号到寄存器，无符号扩展 $RZ \leftarrow \text{zero\_extend}(\text{MEM}[R28 + \text{zero\_extend}(\text{offset} \ll 2)])$
语 法	lrs32.w rz, [label]
说 明	加载 label 所在位置的字符符号，经零扩展到 32 位后存放于目的寄存器 RZ。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 R2 8 加上左移 2 位的 18 位相对偏移量无符号扩展到 32 位后的值得到。LRS.W 指令可以寻址 +1024KB 地址空间。 注意：偏移量 DISP 即二进制操作数 Offset。
影 响 标 志 位	无影响
异 常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常



14.143 LRW——存储器读入指令

统一化指令	
语法	lrw rz, label lrw rz, imm32
操作	从存储器加载字到寄存器 $RZ \leftarrow \text{zero\_extend}(\text{MEM}[(PC + \text{zero\_extend}(\text{offset} \ll 2)) \& 0\text{xfffffc}])$
编译结果	根据加载的范围编译为对应的 16 位或 32 位指令 if(offset<512B), then lrw16 label; lrw16 imm32; else lrw32 label; lrw32 imm32;
说明	加载 label 所在位置的字，或 32 位立即数（IMM32）至目的寄存器 RZ。存储器地址根据 PC 加左移两位的相对偏移量，并无符号扩展到 32 位后，再经最低两位强制清零得到。LRW 指令的加载范围是全部 4GB 地址空间。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

16 位指令	
操作	从存储器加载字到寄存器 $RZ \leftarrow \text{zero\_extend}(\text{MEM}[(\text{PC} + \text{zero\_extend}(\text{offset} \ll 2)) \& 0\text{xffffffc}])$
语法	lrw16 rz, label lrw16 rz, imm32
说明	加载 label 所在位置的字，或 32 位立即数 (IMM32) 至目的寄存器 RZ。存储器地址根据 PC 加左移两位的 10 位相对偏移量，并无符号扩展到 32 位后，再经最低两位强制清零得到。LRW 指令的加载范围是全部 4GB 地址空间。 注意，相对偏移量 Offset 等于二进制编码 {IMM2, IMM5}。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

16 位指令格式：

15	14	11	10	9	8	7	5	4	0
0	0010		0	IMM2		RZ		IMM5	

32 位指令	
操作	从存储器加载字到寄存器 $RZ \leftarrow \text{zero\_extend}(\text{MEM}[(\text{PC} + \text{zero\_extend}(\text{offset} \ll 2)) \& 0\text{xffffffc}])$
语法	lrw32 rz, label lrw32 rz, imm32
说明	加载 label 所在位置的字，或 32 位立即数 (IMM32) 至目的寄存器 RZ。存储器地址根据 PC 加左移两位的 16 位相对偏移量，并无符号扩展到 32 位后，再经最低两位强制清零得到。LRW 指令的加载范围是全部 4GB 地址空间。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

31	30	26	25	21	20	16	15	0
1	11010		10100		RZ		Offset	

14.144 LSL——逻辑左移指令

统一化指令		
语法	lsl rz, rx	lsl rz, rx, ry
操作	$RZ \leftarrow RZ \ll RX[5:0]$	$RZ \leftarrow RX \ll RY[5:0]$
编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (x<16) and (z<16), then lsl16 rz, rx; else lsl32 rz, rz, rx;	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (x==z) and (y<16) and (z<16), then lsl16 rz, ry else lsl32 rz, rx, ry
说明	对于 lsl rz, rx, 将 RZ 的值进行逻辑左移（原值左移，右侧移入 0），结果存入 RZ，左移位数由 RX 低 6 位（RX[5:0]）的值确定；如果 RX[5:0] 的值大于 31，那么 RZ 将被清零； 对于 lsl rz, rx, ry, 将 RX 的值进行逻辑左移（原值左移，右侧移入 0），结果存入 RZ，左移位数由 RY 低 6 位（RY[5:0]）的值确定；如果 RY[5:0] 的值大于 31，那么 RZ 将被清零。	
影响标志位	无影响	
异常	无	

16 位指令	
操作	$RZ \leftarrow RZ \ll RX[5:0]$
语法	lsl16 rz, rx
说明	将 RZ 的值进行逻辑左移（原值左移，右侧移入 0），结果存入 RZ，左移位数由 RX 低 6 位（RX[5:0]）的值确定；如果 RX[5:0] 的值大于 31，那么 RZ 将被清零。
影响标志位	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

15	14	10	9	6	5	2	1	0
0	11100			RZ	RX		00	

32 位指令	
操作	$RZ \leftarrow RX \ll RY[5:0]$
语法	lsl32 rz, rx, ry
说明	将 RX 的值进行逻辑左移（原值左移，右侧移入 0），结果存入 RZ，左移位数由 RY 低 6 位（RY[5:0]）的值确定；如果 RY[5:0] 的值大于 31，那么 RZ 将被清零。
影响标志位	无影响
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001		RY		RX		010000		00001		RZ	

14.145 LSLC——立即数逻辑左移至 C 位指令

统一化指令	
语法	lslc rz, rx, oimm5
操作	$RZ \leftarrow RX \ll OIMM5, C \leftarrow RX[32 - OIMM5]$
编译结果	仅存在 32 位指令。 lslc32 rz, rx, oimm5
说明	将 RX 的值进行逻辑左移（原值左移，右侧移入 0），把移出最末位存入条件位 C，移位结果存入 RZ，左移位数由带偏置 1 的 5 位立即数（OIMM5）的值决定。如果 OIMM5 的值等于 32，那么条件位 C 为 RX 的最低位，RZ 被清零。
影响标志位	$C \leftarrow RX[32 - OIMM5]$
限制	立即数的范围为 1-32。
异常	无

32 位指令	
操作	$RZ \leftarrow RX \ll OIMM5, C \leftarrow RX[32 - OIMM5]$
语法	lslc32 rz, rx, oimm5
说明	将 RX 的值进行逻辑左移（原值左移，右侧移入 0），把移出最末位存入条件位 C，移位结果存入 RZ，左移位数由带偏置 1 的 5 位立即数（OIMM5）的值决定。如果 OIMM5 的值等于 32，那么条件位 C 为 RX 的最低位，RZ 被清零。 注意：二进制操作数 IMM5 等于 OIMM5 - 1。
影响标志位	$C \leftarrow RX[32 - OIMM5]$
限制	立即数的范围为 1-32。
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001	IMM5		RX		010011		00001			RZ	

IMM5 域：

指定不带偏置立即数的值。

注意：移位的值 OIMM5 比起二进制操作数 IMM5 需偏置 1。

00000：

移 1 位

00001：

移 2 位

.....

11111：

移 32 位

14.146 LSLI——立即数逻辑左移指令

统一化指令	
语法	lsli rz, rx, imm5
操作	$RZ \leftarrow RX \ll IMM5$
编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (x<8) and (z<8), then lsli16 rz, rx, imm5 else lsli32 rz, rx, imm5
说明	将 RX 的值进行逻辑左移（原值左移，右侧移入 0），结果存入 RZ，左移位数由 5 位立即数 (IMM5) 的值决定；如果 IMM5 的值等于 0，那么 RZ 的值将不变。
影响标志位	无影响
限制	立即数的范围为 0-31。
异常	无

16 位指令	
操作	$RZ \leftarrow RX \ll IMM5$
语法	lsli16 rz, rx, imm5
说明	将 RX 的值进行逻辑左移（原值左移，右侧移入 0），结果存入 RZ，左移位数由 5 位立即数 (IMM5) 的值决定；如果 IMM5 的值等于 0，那么 RZ 的值将不变。
影响标志位	无影响
限制	寄存器的范围为 r0-r7； 立即数的范围为 0-31。
异常	无

16 位指令格式：

15	14	11	10	8	7	5	4	0
0	1000			RX		RZ		IMM5

32 位指令	
操作	$RZ \leftarrow RX \ll IMM5$
语法	lsli32 rz, rx, imm5
说明	将 RX 的值进行逻辑左移（原值左移，右侧移入 0），结果存入 RZ，左移位数由 5 位立即数（IMM5）的值决定；如果 IMM5 的值等于 0，那么 RZ 的值将与 RX 相同。
影响标志位	无影响
限制	立即数的范围为 0-31。
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001	IMM5		RX		010010		00001			RZ	

14.147 LSR——逻辑右移指令

统一化指令		
语法	lsr rz, rx	lsr rz, rx, ry
操作	$RZ \leftarrow RZ \gg RX[5:0]$	$RZ \leftarrow RX \gg RY[5:0]$
编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (z<16) and (x<16), then lsr16 rz, rx ; else lsr32 rz, rz, rx;	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (x==z) and (z<16) and (y<16), then lsr16 rz, ry ; else lsr32 rz, rx, ry;
说明	对于 lsr rz, rx，将 RZ 的值进行逻辑右移（原值右移，左侧移入 0），结果存入 RZ，右移位数由 RX 低 6 位（RX[5:0]）的值确定；如果 RX[5:0] 的值大于 31，那么 RZ 将被清零。 对于 lsr rz, rx, ry，将 RX 的值进行逻辑右移（原值右移，左侧移入 0），结果存入 RZ，右移位数由 RY 低 6 位（RY[5:0]）的值确定；如果 RY[5:0] 的值大于 31，那么 RZ 将被清零。	
影响标志位	无影响	
异常	无	

16 位指令	
操作	$RZ \leftarrow RZ \gg RX[5:0]$
语法	lsr16 rz, rx
说明	将 RZ 的值进行逻辑右移（原值右移，左侧移入 0），结果存入 RZ，右移位数由 RX 低 6 位（RX[5:0]）的值确定；如果 RX[5:0] 的值大于 31，那么 RZ 将被清零。
影 响 标 志 位	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

15	14	10	9	6	5	2	1	0
0	11100		RZ		RX		01	

32 位指令	
操作	$RZ \leftarrow RX \gg RY[5:0]$
语法	lsr32 rz, rx, ry
说明	将 RX 的值进行逻辑右移（原值右移，左侧移入 0），结果存入 RZ，右移位数由 RY 低 6 位（RY[5:0]）的值确定；如果 RY[5:0] 的值大于 31，那么 RZ 将被清零。
影 响 标 志 位	无影响
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001		RY		RX		010000		00010		RZ	

14.148 LSRC——立即数逻辑右移至 C 位指令

统一化指令	
语法	lsrc rz, rx, oimm5
操作	$RZ \leftarrow RX \gg OIMM5,$ $C \leftarrow RX[OIMM5 - 1]$
编译结果	仅存在 32 位指令。 lsrc32 rz, rx, oimm5
说明	将 RX 的值进行逻辑右移（原值右移，左侧移入 0），把移出最末位存入条件位 C，移位结果存入 RZ，右移位数由带偏置 1 的 5 位立即数（OIMM5）的值决定。如果 OIMM5 的值等于 32，那么条件位 C 为 RX 的最高位，RZ 被清零。
影响标志位	$C \leftarrow RX[OIMM5 - 1]$
限制	立即数的范围为 1-32。
异常	无

32 位指令	
操作	$RZ \leftarrow RX \gg OIMM5, C \leftarrow RX[OIMM5 - 1]$
语法	lsrc32 rz, rx, oimm5
说明	将 RX 的值进行逻辑右移（原值右移，左侧移入 0），把移出最末位存入条件位 C，移位结果存入 RZ，右移位数由带偏置 1 的 5 位立即数（OIMM5）的值决定。如果 OIMM5 的值等于 32，那么条件位 C 为 RX 的最高位，RZ 被清零。 注意：二进制操作数 IMM5 等于 OIMM5 - 1。
影响标志位	$C \leftarrow RX[OIMM5 - 1]$
限制	立即数的范围为 1-32。
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001	IMM5				RX				010011	00010	RZ

IMM5 域：

指定不带偏置立即数的值。

注意：移位的值 OIMM5 比起二进制操作数 IMM5 需偏置 1。

00000：

移 1 位

00001：

移 2 位

.....

11111：

移 32 位

14.149 LSRI——立即数逻辑右移指令

统一化指令	
语法	lsri rz, rx, imm5
操作	RZ ← RX >> IMM5
编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (x<8) and (z<8), then lsri16 rz, rx, imm5 else lsri32 rz, rx, imm5
说明	将 RX 的值进行逻辑右移（原值右移，左侧移入 0），结果存入 RZ，右移位数由 5 位立即数（IMM5）的值决定；如果 IMM5 的值等于 0，那么 RZ 的值不变或者将与 RX 相同。
影响标志位	无影响
限制	立即数的范围为 0-31。
异常	无

16 位指令	
操作	$RZ \leftarrow RX \gg IMM5$
语法	lsri16 rz, rx, imm5
说明	将 RX 的值进行逻辑右移（原值右移，左侧移入 0），结果存入 RZ，右移位数由 5 位立即数（IMM5）的值决定；如果 IMM5 的值等于 0，那么 RZ 的值将不变。
影 响 标 志 位	无影响
限制	寄存器的范围为 r0-r7；立即数的范围为 0-31。
异常	无

16 位指令格式：

15	14	11	10	8	7	5	4	0
0	1001		RX		RZ		IMM5	

32 位指令	
操作	$RZ \leftarrow RX \gg IMM5$
语法	lsri32 rz, rx, imm5
说明	将 RX 的值进行逻辑右移（原值右移，左侧移入 0），结果存入 RZ，右移位数由 5 位立即数（IMM5）的值决定；如果 IMM5 的值等于 0，那么 RZ 的值将与 RX 相同。
影 响 标 志 位	无影响
限制	立即数的范围为 0-31。
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001		IMM5		RX		010010		00010		RZ	

14.150 L2 CACHE——L2 CACHE 操作指令

统一化指令	
语法	L2cache.iall L2cache.call L2cache.ciall
操作	删除所有 l2cache 表项 CLEAR 所有 l2cache 表项 CLEAN 并删除 L2cache 表项
说明	仅存在 32 位指令。 L2cache32.iall L2cache.call L2cache.ciall
影响标志位	无
异常	无

指令格式：

L2cache.iall

31	30	26	25	21	20	16	15	10	9	5	4	0
1		10000		01000		00000		100110		00001		00000

L2cache.call

31	30	26	25	21	20	16	15	10	9	5	4	0
1		10000		00100		00000		100110		00001		00000

L2cache.ciall

31	30	26	25	21	20	16	15	10	9	5	4	0
1		10000		01100		00000		100110		00001		00000

14.151 MFCR——控制寄存器读传送指令

统一化指令	
语法	mfc r _Z , cr<x, sel>
操作	将控制寄存器的内容传送到通用寄存器中 $RZ \leftarrow CR<X, sel>$
编译结果	仅存在 32 位指令。 mfc ₃₂ r _Z , cr<x, sel>
属性：	特权指令
说明	将控制寄存器 CR<x, sel> 的内容传送到通用寄存器 RZ 中。
影响标志位	无影响
异常	特权违反异常（普通用户模式下访问 FCR 和 FESR 除外）

32 位指令	
操作	将控制寄存器的内容传送到通用寄存器中 $RZ \leftarrow CR<X, sel>$
语法	mfc ₃₂ r _Z , cr<x, sel>
属性：	特权指令
说明	将控制寄存器 CR<x, sel> 的内容传送到通用寄存器 RZ 中。
影响标志位	无影响
异常	特权违反异常（普通用户模式下访问 FCR 和 FESR 除外）

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10000	SEL		CRX		011000		00001			RZ	

14.152 MOV——数据传送指令

统一化指令	
语法	mov r _Z , r _X
操作	$RZ \leftarrow RX$
编译结果	总是编译为 16 位指令。 mov ₁₆ r _Z , r _X
说明	把 RX 中的值复制到目的寄存器 RZ 中。
影响标志位	无影响
异常	无

16 位指令	
操作	$RZ \leftarrow RX$
语法	mov16 rz, rx
说明	把 RX 中的值复制到目的寄存器 RZ 中。 注意，该指令寄存器索引范围为 r0-r31。
影响标志位	无影响
异常	无

16 蚊指令格式：

15	14	10	9	6	5	2	1	0
0	11011		RZ		RX			11

32 位指令	
操作	$RZ \leftarrow RX$
语法	mov32 rz, rx
说明	把 RX 中的值复制到目的寄存器 RZ 中。 注意，该指令是 lsl32 rz, rx, 0x0 的伪指令。
影响标志位	无影响
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001		00000		RX		010010		00001			RZ

14.153 MOVF——C 为 0 数据传送指令

统一化指令	
语法	movf rz, rx
操作	if C==0, then $RZ \leftarrow RX$; else $RZ \leftarrow RZ$;
编译结果	仅存在 32 位指令。 movf32 rz, rx
说明	如果 C 为 0，把 RX 的值复制到目的寄存器 RZ 中；否则，RZ 的值不变。 注意，该指令是 incf rz, rx, 0x0 的伪指令。
影响标志位	无影响
异常	无

32 位指令	
操作	if C==0, then RZ ← RX; else RZ ← RZ;
语法	movf32 rz, rx
说明	如果 C 为 0，把 RX 的值复制到目的寄存器 RZ 中；否则，RZ 的值不变。 注意，该指令是 incf32 rz, rx, 0x0 的伪指令。
影 响 标 志 位	无影响
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001		RZ		RX		000011		00001		00000	

14.154 MOVI——立即数数据传送指令

统一化指令	
语法	movi rz, imm
操作	RZ ← zero_extend(IMM);
编 译 结 果	根据立即数和寄存器的范围编译为对应的 16 位或 32 位指令。 if (imm16<256) and (z<7), then movi16 rz, imm8; else movi32 rz, imm16;
说明	将 16 位立即数零扩展至 32 位，然后传送至目的 RZ。
影 响 标 志 位	无影响
限制	立即数的范围为 0x0-0xFFFF。
异常	无

16 位指令	
操作	$RZ \leftarrow \text{zero\_extend}(\text{IMM8});$
语法	movi16 rz, imm8
说明	将 8 位立即数零扩展至 32 位，然后传送至目的 RZ。
影响标志位	无影响
限制	寄存器的范围为 r0-r7；立即数的范围为 0-255。
异常	无

16 位指令格式：

15	14	11	10	8	7	0
0	0110			RZ	IMM8	

32 位指令	
操作	$RZ \leftarrow \text{zero\_extend}(\text{IMM16});$
语法	movi32 rz, imm16
说明	将 16 位立即数零扩展至 32 位，然后传送至目的 RZ。
影响标志位	无影响
限制	立即数的范围为 0x0-0xFFFF。
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	0
1	11010		10000		RZ	IMM16		

14.155 MOVIH——立即数高位数据传送指令

统一化指令	
语法	movih rz, imm16
操作	$RZ \leftarrow \text{zero\_extend}(\text{IMM16}) \ll 16$
编译结果	仅存在 32 位指令。 movih32 rz, imm16
说明	将 16 位立即数零扩展至 32 位，然后逻辑左移 16 位，传送结果至目的 RZ。 该指令可配合 ori rz, rz, imm16 指令产生任意 32 位立即数。
影响标志位	无影响
限制	立即数的范围为 0x0-0xFFFF。
异常	无

32 位指令	
操作	$RZ \leftarrow \text{zero\_extend}(\text{IMM16}) \ll 16$
语法	movih32 rz, imm16
说明	将 16 位立即数零扩展至 32 位，然后逻辑左移 16 位，传送结果至目的 RZ。 该指令可配合 ori32 rz, rz, imm16 指令产生任意 32 位立即数。
影响标志位	无影响
限制	立即数的范围为 0x0-0xFFFF。
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	0
1	11010	10001		RZ		IMM16		

14.156 MOVTC——C 为 1 数据传送指令

统一化指令	
语法	movt rz, rx
操作	if C==1, then RZ ← RX; else RZ ← RZ;
编译结果	仅存在 32 位指令。 movt32 rz, rx
影响标志位	无影响
异常	无

32 位指令	
操作	if C==1, then RZ ← RX; else RZ ← RZ;
语法	movt32 rz, rx
说明	如果 C 为 1，把 RX 的值复制到目的寄存器 RZ 中；否则，RZ 的值不变。 注意，该指令是 inct32 rz, rx, 0x0 的伪指令。
影响标志位	无影响
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001		RZ		RX		000011		00010		00000	

14.157 MTCR——控制寄存器写传送指令

统一化指令	
语法	mter rx, cr<z, sel>
操作	将通用寄存器的内容传送到控制寄存器中 $CR<Z, sel> \leftarrow RX$
编译结果	仅存在 32 位指令。 mter32 rx, cr<z, sel>
属性：	特权指令
说明	将通用寄存器 RX 的内容传送到控制寄存器 CR<z, sel> 中。
影响标志位	如果目标控制寄存器不是 PSR，则该指令不会影响标志位。
异常	特权违反异常（普通用户模式下访问 FCR 和 FESR 除外）

32 位指令	
操作	将通用寄存器的内容传送到控制寄存器中 $CR<Z, sel> \leftarrow RX$
语法	mter32 rx, cr<z, sel>
属性：	特权指令
说明	将通用寄存器 RX 的内容传送到控制寄存器 CR<z, sel> 中。
影响标志位	如果目标控制寄存器不是 PSR，则该指令不会影响标志位。
异常	特权违反异常（普通用户模式下访问 FCR 和 FESR 除外）

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10000		SEL		RX		011001		00001		CRZ	

14.158 MUL.S32——32 位有符号乘法指令

统一化指令	
语法	mul.s32 rz, rx, ry
操作	{Rz+1[31:0],Rz[31:0]} = Rx[31:0] X Ry[31:0]
编译结果	仅存在 32 位指令。 mul.s32 rz, rx, ry
说明	有符号数 Rx 与有符号数 Ry 相乘，得到 64 位结果，其中高 32 位存入 Rz+1，低 32 位存入 Rz。
影响标志位	无影响
限制	RZ 和 rz+1 不能等于 rx 或 ry
异常	无

32 位指令	
操作	{Rz+1[31:0],Rz[31:0]} = Rx[31:0] X Ry[31:0]
语法	mul.s32 rz, rx, ry
说明	有符号数 Rx 与有符号数 Ry 相乘，得到 64 位结果，其中高 32 位存入 Rz+1，低 32 位存入 Rz。
影响标志位	无影响
限制	RZ 和 rz+1 不能等于 rx 或 ry
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11110			RY		RX		100000		10000		RZ

14.159 MUL.U32——32 位无符号乘法指令

统一化指令	
语法	mul.u32 rz, rx, ry
操作	{Rz+1[31:0],Rz[31:0]} = Rx[31:0] X Ry[31:0]
编译结果	仅存在 32 位指令。 mul.u32 rz, rx, ry
说明	无符号数 Rx 与无符号数 Ry 相乘，得到 64 位结果，其中高 32 位存入 Rz+1，低 32 位存入 Rz。
影响标志位	无影响
限制	RZ 和 rz+1 不能等于 rx 或 ry
异常	无

32 位指令	
操作	{Rz+1[31:0],Rz[31:0]} = Rx[31:0] X Ry[31:0]
语法	mul.u32 rz, rx, ry
说明	无符号数 Rx 与无符号数 Ry 相乘，得到 64 位结果，其中高 32 位存入 Rz+1，低 32 位存入 Rz。
影响标志位	无影响
限制	RZ 和 rz+1 不能等于 rx 或 ry
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11110		RY		RX		100000		00000		RZ	

14.160 MULA.32.L——32 位有符号乘累加取低 32 位指令

统一化指令	
语法	mula.32.l rz, rx, ry
操作	$Rz[31:0] \leftarrow Rz[31:0] + \{Rx[31:0] \times Ry[31:0]\}[31:0]$
编译结果	仅存在 32 位指令。 mula.32.l rz, rx, ry
说明	Rx 与 Ry 相乘得到 64 位结果，并与 Rz 相加，截取加法结果的低 32 位存入 Rz。
影响标志位	无影响
限制	无
异常	无

32 位指令	
操作	$Rz[31:0] \leftarrow Rz[31:0] + \{Rx[31:0] \times Ry[31:0]\}[31:0]$
语法	mula.32.l rz, rx, ry
说明	Rx 与 Ry 相乘得到 64 位结果，并与 Rz 相加，截取加法结果的低 32 位存入 Rz。
影响标志位	无影响
限制	无
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11110		RY		RX		100001		00010		RZ	

14.161 MULA.S32——32 位有符号乘累加指令

统一化指令	
语法	mula.s32 rz, rx, ry
操作	$\{Rz+1[31:0], Rz[31:0]\} = \{Rz+1[31:0], Rz[31:0]\} + Rx[31:0] \times Ry[31:0]$
编译结果	仅存在 32 位指令。 mula.s32 rz, rx, ry
说明	有符号数 Rx 与有符号数 Ry 相乘，得到 64 位结果，并与 {Rz+1, Rz} 相加。加法结果的高 32 位存入 Rz+1，低 32 位存入 Rz。
影响标志位	无影响
限制	RZ 和 rz+1 不能等于 rx 或 ry
异常	无

32 位指令	
操作	$\{Rz+1[31:0],Rz[31:0]\} = \{Rz+1[31:0],Rz[31:0]\} + Rx[31:0] \times Ry[31:0]$
语法	mula.s32 rz, rx, ry
说明	有符号数 Rx 与有符号数 Ry 相乘，得到 64 位结果，并与 {Rz+1, Rz} 相加。加法结果的高 32 位存入 Rz+1，低 32 位存入 Rz。
影响标志位	无影响
限制	RZ 和 rz+1 不能等于 rx 或 ry
异常	无

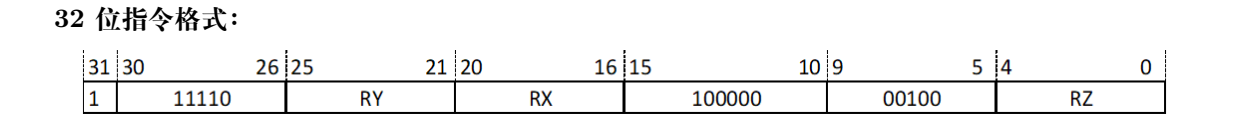
32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11110			RY		RX		100000		10100		RZ

14.162 MULA.U32——32 位无符号乘累加指令

统一化指令	
语法	mula.u32 rz, rx, ry
操作	$\{Rz+1[31:0],Rz[31:0]\} = \{Rz+1[31:0],Rz[31:0]\} + Rx[31:0] \times Ry[31:0]$
编译结果	仅存在 32 位指令。 mula.u32 rz, rx, ry
说明	无符号数 Rx 与无符号数 Ry 相乘，得到 64 位结果，并与 {Rz+1, Rz} 相加。加法结果的高 32 位存入 Rz+1，低 32 位存入 Rz。
影响标志位	无影响
限制	RZ 和 rz+1 不能等于 rx 或 ry
异常	无

32 位指令	
操作	$\{Rz+1[31:0],Rz[31:0]\} = \{Rz+1[31:0],Rz[31:0]\} + Rx[31:0] \times Ry[31:0]$
语法	mula.u32 rz, rx, ry
说明	无符号数 Rx 与无符号数 Ry 相乘，得到 64 位结果，并与 {Rz+1, Rz} 相加。加法结果的高 32 位存入 Rz+1，低 32 位存入 Rz。
影响标志位	无影响
限制	RZ 和 rz+1 不能等于 rx 或 ry
异常	无



14.163 MULALL.S16.S——带饱和操作的 16 位有符号低半字乘累加指令

统一化指令	
语 法	mulall.s16.s rz, rx, ry
操 作	$Rz[31:0] = \text{Saturate}(Rz[31:0] + Rx[15:0] \times Ry[15:0])$
编 译 结 果	仅存在 32 位指令。 mulall.s16.s rz, rx, ry
说 明	Rx 的低半字与 Ry 的低半字相乘，乘法结果与 Rz 相加，加法结果做饱和处理后存入 Rz。 饱和处理的过程为，若加法结果大于上饱和值 0x7FFFFFFF，结果为上饱和值，若加法结果小于下饱和值 0x80000000，结果为下饱和值，其它情况为加法结果本身。
影 响 标 志 位	无影响
限 制	无
异 常	无

32 位指令	
操作	$Rz[31:0] = \text{Saturate}(Rz[31:0] + Rx[15:0] \times Ry[15:0])$
语法	mulall.s16.s rz, rx, ry
说明	Rx 的低半字与 Ry 的低半字相乘，乘法结果与 Rz 相加，加法结果做饱和处理后存入 Rz。 饱和处理的过程为，若加法结果大于上饱和值 0x7FFFFFFF，结果为上饱和值，若加法结果小于下饱和值 0x80000000，结果为下饱和值，其它情况为加法结果本身。
影响标志位	无影响
限制	无
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11110			RY		RX		100000		01101		RZ

14.164 MULSH——16 位有符号乘法指令

统一化指令	
语法	mulsh rz, rx
操作	两个 16 位有符号数相乘，结果放入通用寄存器中 $RZ \leftarrow RX[15..0] \times RZ[15..0]$
编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (x<16) and (y<16), then mulsh16 rz, rx; else mulsh32 rz, rz, rx;
说明	将通用寄存器 RX 的低 16 位和 RZ/ RY 的低 16 位相乘后的结果存放到通用寄存器 RZ 中。寄存器的内容均被认为是有符号数，其中源寄存器的符号位是第 15 位，目标寄存器的符号位是第 31 位。
影响标志位	无影响
异常	无

16 位指令	
操作	两个 16 位有符号数相乘，结果放入通用寄存器中 $RZ \leftarrow RX[15..0] \times RZ[15..0]$
语法	mulsh16 rz, rx
说明	将通用寄存器 RX 的低 16 位和 RZ 的低 16 位相乘后得到的 32 位结果存放到通用寄存器 RZ 中。寄存器 RX 和 RZ 的内容均被认为是有符号数，其中源操作数的符号位是寄存器的第 15 位，结果的符号位是寄存器的第 31 位。
影响标志位	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

15	14	10	9	6	5	2	1	0
0	11111	RZ		RX			01	

32 位指令	
操作	两个 16 位有符号数相乘，结果放入通用寄存器中 $RZ \leftarrow RX[15..0] \times RY[15..0]$
语法	mulsh32 rz, rx, ry
说明	将通用寄存器 RX 的低 16 位和 RY 的低 16 位相乘后的结果存放 to 通用寄存器 RZ 中。寄存器 RX、RY 和 RZ 的内容均被认为是 有符号数，其中源寄存器 RX、RY 的符号位是第 15 位，目标寄存器 RZ 的符号位是第 31 位。
影响标志位	无影响
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001		RY		RX		100100		00001		RZ	

14.165 MULSW——有符号数 16x32 乘法指令

统一化指令	
语法	mulsw rz, rx, ry
操作	16 位有符号数和 32 位有符号数相乘，结果的高 32 位放入通用寄存器中 $RZ \leftarrow (RX[15:0] \times RY[31:0])[47:16];$
编译结果	仅存在 32 位指令。 mulsw32 rx, ry
说明	将通用寄存器 RX 的低 16 位和 RY 的内容相乘后结果的高 32 位存放到通用寄存器 RZ 中。通用寄存器 RX 和 RY 以及 RZ 中的值均被认为是 有符号数，其中寄存器 RX 的源操作数的符号位是寄存器的第 15 位，寄存器 RY 的源操作数和寄存器 RZ 中结果的符号位是寄存器的第 31 位。
影响标志位	无影响
异常	无

32 位指令	
操作	16 位有符号数和 32 位有符号数相乘，结果的高 32 位放入通用寄存器中 $RZ \leftarrow (RX[15:0] \times RY[31:0])[47:16];$
语法	mulsw32 rx, ry
说明	将通用寄存器 RX 的低 16 位和 RY 的内容相乘后结果的高 32 位存放到通用寄存器 RZ 中。通用寄存器 RX 和 RY 以及 RZ 中的值均被认为是带符号数，其中寄存器 RX 的源操作数的符号位是寄存器的第 15 位，寄存器 RY 的源操作数和寄存器 RZ 中结果的符号位是寄存器的第 31 位。
影响标志位	无影响
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001		RY		RX		100101		00001		RZ	

14.166 MULT——乘法指令

统一化指令		
语 法	mult rz, rx	mult rz, rx, ry
操 作	两个数相乘，结果的低 32 位放入通用寄存器中 $RZ \leftarrow RX \times RZ$	两个数相乘，结果的低 32 位放入通用寄存器中 $RZ \leftarrow RX \times RY$
编 译 结 果	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (x<16) and (z<16), then mult16 rz, rx; else mult32 rz, rz, rx;	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (y==z) and (x<16)and (z<16), then mult16 rz, rx; else mult32 rz, rx, ry;
说 明	将两个源寄存器的内容相乘后结果的低 32 位存放到目的寄存器中，结果的高 32 位舍弃。不管源操作数被认为是符号数还是无符号数，结果都相同。	
影 响 标 志 位	无影响	
异 常	无	

16 位指令	
操作	两个数相乘，结果的低 32 位放入通用寄存器中 $RZ \leftarrow RX \times RZ$
语法	mult16 rz, rx
说明	将通用寄存器 RX 和 RZ 的内容相乘后得到结果的低 32 位存放到通用寄存器 RZ 中，结果的高 32 位舍弃。不管源操作数被认为是符号数还是无符号数，结果都相同。
影 响 标 志 位	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

15	14	10	9	6	5	2	1	0
0	11111	RZ	RX	00				

32 位指令	
操作	两个数相乘，结果的低 32 位放入通用寄存器中 $RZ \leftarrow RX \times RY$
语法	mult32 rz, rx, ry
说明	将通用寄存器 RX 和 RY 的内容相乘后结果的低 32 位存放 to 通用寄存器 RZ 中，结果的高 32 位舍弃。不管源操作数被认为是符号数还是无符号数，结果都相同。
影响标志位	无影响
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001		RY		RX		100001		00001		RZ	

14.167 MVC——C 位传送指令

统一化指令	
语法	mvc rz
操作	$RZ \leftarrow C$
编译结果	仅存在 32 位指令。 mvc32 rz;
说明	把条件位 C 传送到 RZ 的最低位，RZ 的其它位清零。
影响标志位	无影响
异常	无

32 位指令	
操作	$RZ \leftarrow C$
语法	mvc32 rz
说明	把条件位 C 传送到 RZ 的最低位，RZ 的其它位清零。
影响标志位	无影响
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1		10001		00000		00000		000001		01000		RZ

14.168 MVCV——C 位取反传送

统一化指令	
语法	mvcv rz
操作	$RZ \leftarrow (!C)$
编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (z<16), then mvcv16 rz; else mvcv32 rz;
说明	把条件位 C 取反后传送到 RZ 的最低位，RZ 的其它位清零。
影响标志位	无影响
异常	无

16 位指令	
操作	$RZ \leftarrow (!C)$
语法	mvcv16 rz
说明	把条件位 C 取反后传送到 RZ 的最低位，RZ 的其它位清零。
影响标志位	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

15	14	10	9	6	5	2	1	0
0	11001		RZ		0000		11	

32 位指令	
操作	$RZ \leftarrow (!C)$
语法	mvcv32 rz
说明	把条件位 C 取反后传送到 RZ 的最低位，RZ 的其它位清零。
影响标志位	无影响
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001		00000		00000		000001		10000		RZ	

14.169 NOR——按位或非指令

统一化指令		
语 法	nor rz, rx	or rz, rx, ry
操 作	$RZ \leftarrow !(RZ \mid RX)$	$RZ \leftarrow !(RX \mid RY)$
编 译 结 果	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (x<16) and (z<16), then nor16 rz, rx; else nor32 rz, rz, rx;	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (y==z) and (x<16) and (z<16), then nor16 rz, rx else nor32 rz, rx, ry
说 明	将 RX 与 RY/RZ 的值按位或，然后按位取非，并把结果存在 RZ。	
影 响 标 志 位	无影响	
异 常	无	

16 位指令	
操作	$RZ \leftarrow !(RZ \mid RX)$
语法	nor16 rz, rx
说明	将 RZ 与 RX 的值按位或，然后按位取非，并把结果存在 RZ。
影响标志位	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

15	14	10	9	6	5	2	1	0
0	11011	RZ		RX		10		

32 位指令	
操作	$RZ \leftarrow !(RX \mid RY)$
语法	<code>nor32 rz, rx, ry</code>
说明	将 RX 与 RY 的值按位或，然后按位取非，并把结果存在 RZ。
影响标志位	无影响
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001		RY		RX		001001		00100		RZ	

14.170 NOT——按位非指令

统一化指令		
语法	<code>not rz</code>	<code>not rz, rx</code>
操作	$RZ \leftarrow !(RZ)$	$RZ \leftarrow !(RX)$
编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (z<16), then <code>not16 rz;</code> else <code>not32 rz, rz;</code>	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (x==z) and (z<16), then <code>not16 rz;</code> else <code>not32 rz, rx;</code>
说明	将 RZ/RX 的值按位取反，把结果存在 RZ。 注意，该指令是 <code>nor rz, rz</code> 和 <code>nor rz, rx, rx</code> 的伪指令。	
影响标志位	无影响	
异常	无	

16 位指令	
操作	$RZ \leftarrow !(RZ)$
语法	<code>not16 rz</code>
说明	将 RZ 的值按位取反，把结果存在 RZ。 注意，该指令是 <code>nor16 rz, rz</code> 的伪指令。
影响标志位	无影响
异常	无

16 位指令格式：

15	14	10	9	6	5	2	1	0
0	11011	RZ		RZ		10		

32 位指令	
操作	$RZ \leftarrow \neg(RX)$
语法	not32 rz, rx
说明	将 RX 的值按位取反，把结果存在 RZ。 注意，该指令是 nor32 rz, rx, rx 的伪指令。
影响标志位	无影响
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001	RX		RX		001001		00100		RZ		

14.171 OR——按位或指令

统一化指令		
语法	or rz, rx	or rz, rx, ry
操作	$RZ \leftarrow RZ \mid RX$	$RZ \leftarrow RX \mid RY$
编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (x<16) and (z<16), then or16 rz, rx ; else or32 rz, rz, rx;	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (y==z) and (x<16) and (z<16), then or16 rz, rx else or32 rz, rx, ry
说明	将 RX 与 RY/RZ 的值按位或，并把结果存在 RZ。	
影响标志位	无影响	
异常	无	

16 位指令	
操作	$RZ \leftarrow RZ \mid RX$
语法	or16 rz, rx
说明	将 RZ 与 RX 的值按位或，并把结果存在 RZ。
影响标志位	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

15	14	10	9	6	5	2	1	0
0	11011	RZ				RX		00

32 位指令	
操作	$RZ \leftarrow RX \mid RY$
语法	or rz, rx, ry
说明	将 RX 与 RY 的值按位或，并把结果存在 RZ。
影响标志位	无影响
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001			RY		RX		001001		00001		RZ

14.172 ORI——立即数按位或指令

统一化指令	
语法	ori rz, rx, imm16
操作	$RZ \leftarrow RX \mid \text{zero\_extend}(\text{IMM16})$
编译结果	仅存在 32 位指令。 ori32 rz, rx, imm16
说明	将 16 位立即数零扩展至 32 位，然后与 RX 的值进行按位或操作，把结果存入 RZ。
影响标志位	无影响
限制	立即数的范围为 0x0-0xFFFF。
异常	无

32 位指令	
操作	$RZ \leftarrow RX \mid \text{zero\_extend}(\text{IMM16})$
影响标志位	无影响
限制	立即数的范围为 0x0-0xFFFF。
语法	ori32 rz, rx, imm16
说明	将 16 位立即数零扩展至 32 位，然后与 RX 的值进行按位或操作，把结果存入 RZ。
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15					0
1	11011			RZ		RX		IMM16				

14.173 PLDR——读数据预取指令

统一化指令	
语法	pldr (rx, disp)
操作	读数据预取 MEM[RX + zero_extend(offset << 2)]
编译结果	仅存在 32 位指令。 pldr32 (rx, disp)
说明	读数据预取指令的目的是加速数据的加载行为。在数据加载前，使用 PLDR 指令将该数据行读入到 Cache 中；在加载数据的 Load 指令执行时，命中在 D-Cache 中，从而加快数据加载的效率。读数据预取的有效地址由基址寄存器 RX 加上左移两位的 12 相对偏移量无符号扩展到 32 位后的值得到。注意，偏移量 DISP 是二进制操作数 Offset 左移两位得到的。
影响标志位	无影响
异常	无

32 位指令	
操作	读数据预取
语法	pldr32 (rx, disp)
说明	读数据预取指令的目的是加速数据的加载行为。在数据加载前，使用 PLDR 指令将该数据行读入到 Cache 中；在加载数据的 Load 指令执行时，命中在 D-Cache 中，从而加快数据加载的效率。读数据预取的有效地址由基址寄存器 RX 加上左移两位的 12 相对偏移量无符号扩展到 32 位后的值得到。注意，偏移量 DISP 是二进制操作数 Offset 左移两位得到的。
影响标志位	无影响
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	12	11	0
1	10110	00000		RX		0110		Offset		

14.174 PLDW——写数据预取指令

统一化指令	
语 法	pldw (rx, disp)
操 作	写数据预取 MEM[RX + zero_extend(offset << 2)]
编 译 结 果	仅存在 32 指令。 pldw32 (r 位 x, disp)
说 明	写数据预取指令的目的是加速写分配 Cache 策略下数据的存储行为。在写分配 Cache 的策略下，存储指令 Cache 未命中会导致一次 Cache 回填操作。PLDW 指令在存储指令之前将即将存储的数据所在的行读入到 Cache；在存储数据的 Store 指令执行时，命中在 D-Cache 中，从而加快数据存储的效率。写数据预取的有效地址由基址寄存器 RX 加上左移两位的 12 相对偏移量无符号扩展到 32 位后的值得到。注意，偏移量 DISP 是二进制操作数 Offset 左移两位得到的。
影 响 标 志 位	无影响
异 常	无

32 位指令	
操作	写数据预取
语法	pldw32 (rx, disp)
说明	写数据预取指令的目的是加速写分配 Cache 策略下数据的存储行为。在写分配 Cache 的策略下，存储指令 Cache 未命中会导致一次 Cache 回填操作。PLDW 指令在存储指令之前将即将存储的数据所在的行读入到 Cache；在存储数据的 Store 指令执行时，命中在 D-Cache 中，从而加快数据存储的效率。写数据预取的有效地址由基址寄存器 RX 加上左移两位的 12 相对偏移量无符号扩展到 32 位后的值得到。注意，偏移量 DISP 是二进制操作数 Offset 左移两位得到的。
影响标志位	无影响
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	12	11	0
1	10111	00000		RX		0110		Offset		

14.175 POP——出栈指令

统一化指令	
语 法	pop reglist
操 作	从堆栈存储器加载连续的多个字到一片连续的寄存器堆中，然后更新堆栈寄存器到堆栈存储器的顶端，并子程序返回； dst ← {reglist}; addr ← SP; foreach (reglist){ Rdst ← MEM[addr]; dst ← next {reglist}; addr ← addr + 4; } sp ← addr; PC ← R15 & 0xffffffe;
编 译 结 果	根据寄存器的范围编译为对应的 16 位或 32 位指令 if ({reglist}<16), then pop16 reglist; else pop32 reglist;
说 明	从堆栈存储器加载连续的多个字到一片连续的寄存器堆中，更新堆栈指针寄存器，然后实现子程序返回功能，即程序跳转到链接寄存器 R15 指定的位置，链接寄存器的最低位被忽略。采用堆栈寄存器直接寻址方式。
影 响 标 志 位	无影响
异 常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

16 位指令	
操作	从堆栈存储器加载连续的多个字到一片连续的寄存器堆中，然后更新堆栈寄存器到堆栈存储器的顶端，并子程序返回。 dst ← {reglist}; addr ← SP; foreach (reglist){ Rdst ← MEM[addr]; dst ← next {reglist}; addr ← addr + 4; } sp ← addr; PC ← R15 & 0xffffffe;
语法	pop16 reglist
说明	从堆栈存储器加载连续的多个字到一片连续的寄存器堆中，更新堆栈指针寄存器，然后实现子程序返回功能，即程序跳转到链接寄存器 R15 指定的位置，链接寄存器的最低位被忽略。采用堆栈指针寄存器直接寻址方式。
影响标志位	无影响
限制	寄存器的范围为 r4 – r11, r15。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

16 位指令格式：

15	14	10	9	8	7	6	5	4	3	0
0	00101			0	0	10	0	R15	LIST1	

LIST1 域：

指定寄存器 r4-r11 是否在寄存器列表中。

0000：

r4-r11 不在寄存器列表中

0001：

r4 在寄存器列表中

0010：

r4-r5 在寄存器列表中

0011:

r4-r6 在寄存器列表中

.....

1000:

r4-r11 在寄存器列表中

R15 域:

指定寄存器 r15 是否在寄存器列表中。

0:

r15 不在寄存器列表中

1:

r15 在寄存器列表中

32 位指令	
操作	从堆栈存储器加载连续的多个字到一片连续的寄存器堆中 $dst \leftarrow \{reglist\}; addr \leftarrow SP;$ foreach (reglist){ $Rdst \leftarrow MEM[addr];$ $dst \leftarrow next \{reglist\};$ $addr \leftarrow addr + 4;$ } $sp \leftarrow addr;$ $PC \leftarrow R15 \& 0xffffffe;$
语法	pop32 reglist
说明	从堆栈存储器加载连续的多个字到一片连续的寄存器堆中，更新堆栈指针寄存器，然后实现子程序返回功能，即程序跳转到链接寄存器 R15 指定的位置，链接寄存器的最低位被忽略。采用堆栈指针寄存器直接寻址方式。
影响标志位	无影响
限制	寄存器的范围为 r4 – r11, r15, r16 - r17, r29。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令格式:

31	30	26	25	21	20	16	15	12	11	10	9	8	7	5	4	3	0
1		11010		11110		00000		0000	0	0	0	R29	LIST2	R15		LIST1	

LIST1 域:

指定寄存器 r4-r11 是否在寄存器列表中。

0000:

r4-r11 不在寄存器列表中

0001:

r4 在寄存器列表中

0010:

r4-r5 在寄存器列表中

0011:

r4-r6 在寄存器列表中

.....

1000:

r4-r11 在寄存器列表中

R15 域:

指定寄存器 r15 是否在寄存器列表中。

0:

r15 不在寄存器列表中

1:

r15 在寄存器列表中

LIST2 域:

指定寄存器 r16-r17 是否在寄存器列表中。

000:

r16-r19 不在寄存器列表中

001:

r16 在寄存器列表中

010:

r16-r17 在寄存器列表中

R29 域:

指定寄存器 r29 是否在寄存器列表中。

0:

r29 不在寄存器列表中

1:

r29 在寄存器列表中

14.176 PSRCLR——PSR 位清零指令

统一化指令	
语法	psrclr ee, ie, fe, af 或者操作数也可以为 ee、ie、fe、af 的任意组合。
操作	清除状态寄存器的某一位或几位。 $PSR(\{EE, IE, FE, AF\}) \leftarrow 0$
编译结果	仅存在 32 位指令。 psrclr32 ee, ie, fe, af
属性:	特权指令
说明	选中的 PSR 位被清零 (1 表示选中)。五位立即数 IMM5 用于编码要清除的控制位，对应关系如下： 立即数 IMM5 各位：对应的 PSR 控制位 Imm5[0]：AF Imm5[1]：FE Imm5[2]：IE Imm5[3]：EE Imm5[4]：保留
影响标志位	无影响
异常	特权违反异常

32 位指令	
操作	清除状态寄存器的某一位或几位 $PSR(\{EE, IE, FE, AF\}) \leftarrow 0$
语法	psrclr32 ee, ie, fe, af 或者操作数也可以为 ee、ie、fe、af 的任意组合。
属性:	特权指令
说明	选中的 PSR 位被清零 (1 表示选中)。五位立即数 IMM5 用于编码要清除的控制位，对应关系如下： 立即数 IMM5 各位：对应的 PSR 控制位 Imm5[0]: AF Imm5[1]: FE Imm5[2]: IE Imm5[3]: EE Imm5[4]: 保留
影响标志位	无影响
异常	特权违反异常

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10000	IMM5	00000	011100	00001	00000						

14.177 PSRSET——PSR 位置位指令

统一化指令	
语法	psrset ee, ie, fe, af 或者操作数也可以为 ee、ie、fe、af 的任意组合。
操作	设置状态寄存器的某几位 $PSR(\{EE, IE, FE, AF\}) \leftarrow 1$
编译结果	仅存在 32 位指令。 psrset32 ee, ie, fe, af
属性：	特权指令
说明	选中的 PSR 位被置位（1 表示选中）。五位立即数 IMM5 用于编码要清除的控制位，对应关系如下： 立即数 IMM5 各位：对应的 PSR 控制位 Imm5[0]：AF Imm5[1]：FE Imm5[2]：IE Imm5[3]：EE Imm5[4]：保留
影响标志位	无影响
异常	特权违反异常

32 位指令	
操作	设置状态寄存器的某几位 $PSR(\{EE, IE, FE, AF\}) \leftarrow 1$
语法	psrset32 ee, ie, fe, af 或者操作数也可以为 ee、ie、fe、af 的任意组合。
属性:	特权指令
说明	选中的 PSR 位被置位 (1 表示选中)。五位立即数 IMM5 用于编码要清除的控制位，对应关系如下： 立即数 IMM5 各位对应的 PSR 控制位 Imm5[0]: AF Imm5[1]: FE Imm5[2]: IE Imm5[3]: EE Imm5[4]: 保留
影响标志位	无影响
异常	特权违反异常

32 位指令格式:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10000	IMM5	00000	011101	00001	00000						

14.178 PUSH——压栈指令

统一化指令	
语法	push reglist
操作	将寄存器列表中的字存储到堆栈存储器中，然后更新堆栈寄存器到堆栈存储器的顶端； src ← {reglist}; addr ← SP; foreach (reglist){ MEM[addr] ← Rsrc; src ← next {reglist}; addr ← addr - 4; } sp ← addr;
编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令 if ({reglist}<16), then push16 reglist; else push32 reglist;
说明	将寄存器列表中的字存储到堆栈存储器中，然后更新堆栈寄存器到堆栈存储器的顶端。采用堆栈寄存器直接寻址方式。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

16 位指令	
操作	将寄存器列表中的字存储到堆栈存储器中 $src \leftarrow \{reglist\}; addr \leftarrow SP;$ foreach (reglist){ MEM[addr] $\leftarrow$ Rsrc; $src \leftarrow next \{reglist\};$ $addr \leftarrow addr - 4;$ } $sp \leftarrow addr$
语法	push16 reglist
说明	将寄存器列表中的字存储到堆栈存储器中，然后更新堆栈寄存器到堆栈存储器的顶端。采用堆栈寄存器直接寻址方式。
影 响 标 志 位	无影响
限制	寄存器的范围为 r4 – r11, r15。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

16 位指令格式：

15	14	10	9	8	7	6	5	4	3	0
0	00101			0	0	11	0	R15	LIST1	

LIST1 域：

指定寄存器 r4-r11 是否在寄存器列表中。

0000：

r4-r11 不在寄存器列表中

0001：

r4 在寄存器列表中

0010：

r4-r5 在寄存器列表中

0011：

r4-r6 在寄存器列表中

.....

1000：

r4-r11 在寄存器列表中

R15 域：

指定寄存器 r15 是否在寄存器列表中。

1:

r15 在寄存器列表中

32 位指令	
操作	从堆栈存储器加载连续的多个字到一片连续的寄存器堆中 src ← {reglist}; addr ← SP; foreach (reglist){ MEM[addr] ← Rsrc; src ← next {reglist}; addr ← addr - 4; } sp ← addr
语法	push32 reglist
说明	将寄存器列表中的字存储到堆栈存储器中，然后更新堆栈寄存器到堆栈存储器的顶端。采用堆栈寄存器直接寻址方式。
影 响 标 志 位	无影响
限制	寄存器的范围为 r4 – r11, r15, r16 - r17, r29。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令格式：

31	30	26	25	21	20	16	15	12	11	10	9	8	7	5	4	3	0		
1	11010			11111			00000			0000			0	0	0	R29	LIST2	R15	LIST1

LIST1 域：

指定寄存器 r4-r11 是否在寄存器列表中。

0000:

r4-r11 不在寄存器列表中

0001:

r4 在寄存器列表中

0010:

r4-r5 在寄存器列表中

0011:

r4-r6 在寄存器列表中

... ..

1000:

r4-r11 在寄存器列表中

R15 域:

指定寄存器 r15 是否在寄存器列表中。

0:

r15 不在寄存器列表中

1:

r15 在寄存器列表中

LIST2 域:

指定寄存器 r16-r17 是否在寄存器列表中。

000:

r16-r19 不在寄存器列表中

001:

r16 在寄存器列表中

010:

r16-r17 在寄存器列表中

R29 域:

指定寄存器 r29 是否在寄存器列表中。

0:

r29 不在寄存器列表中

1:

r29 在寄存器列表中

14.179 REVB——字节倒序指令

统一化指令	
语法	revb rz, rx
操作	RZ[31:24] ← RX[7:0]; RZ[23:16] ← RX[15:8]; RZ[15:8] ← RX[23:16]; RZ[7:0] ← RX[31:24];
编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (x<16) and (z<16), then revb16 rz, rx; else revb32 rz, rx;
说明	把 RX 的值按字节取倒序，各字节内部的位顺序保持不变，结果存入 RZ。
影响标志位	无影响
异常	无

16 位指令	
操作	RZ[31:24] ← RX[7:0]; RZ[23:16] ← RX[15:8]; RZ[15:8] ← RX[23:16]; RZ[7:0] ← RX[31:24];
语法	revb16 rz, rx
说明	把 RX 的值按字节取倒序，各字节内部的位顺序保持不变，结果存入 RZ。
影响标志位	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

15	14	10	9	6	5	2	1	0
0	11110		RZ		RX			10

32 位指令	
操作	$RZ[31:24] \leftarrow RX[7:0];$ $RZ[23:16] \leftarrow RX[15:8];$ $RZ[15:8] \leftarrow RX[23:16];$ $RZ[7:0] \leftarrow RX[31:24];$
语法	revb32 rz, rx
说明	把 RX 的值按字节取倒序，各字节内部的位顺序保持不变，结果存入 RZ。
影响标志位	无影响
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001	00000		RX		011000		00100			RZ	

14.180 REVH——半字字节倒序指令

统一化指令	
语法	revh rz, rx
操作	$RZ[31:24] \leftarrow RX[23:16];$ $RZ[23:16] \leftarrow RX[31:24];$ $RZ[15:8] \leftarrow RX[7:0];$ $RZ[7:0] \leftarrow RX[15:8];$
编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (x<16) and (z<16), then revh16 rz, rx; else revh32 rz, rx;
说明	把 RX 的值在半字内按字节取倒序，即分别交换高半字内的两个字节和低半字内的两个字节，两个半字间的顺序和各字节内的位顺序保持不变，结果存入 RZ。
影响标志位	无影响
异常	无

16 位指令	
操作	$RZ[31:24] \leftarrow RX[23:16];$ $RZ[23:16] \leftarrow RX[31:24];$ $RZ[15:8] \leftarrow RX[7:0];$ $RZ[7:0] \leftarrow RX[15:8];$
语法	revh16 rz, rx
说明	把 RX 的值在半字内按字节取倒序，即分别交换高半字内的两个字节和低半字内的两个字节，两个半字间的顺序和各字节内的位顺序保持不变，结果存入 RZ。
影响标志位	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

15	14	10	9	6	5	2	1	0
0	11110	RZ	RX	11				

32 位指令	
操作	$RZ[31:24] \leftarrow RX[23:16];$ $RZ[23:16] \leftarrow RX[31:24];$ $RZ[15:8] \leftarrow RX[7:0];$ $RZ[7:0] \leftarrow RX[15:8];$
语法	revh32 rz, rx
说明	把 RX 的值在半字内按字节取倒序，即分别交换高半字内的两个字节和低半字内的两个字节，两个半字间的顺序和各字节内的位顺序保持不变，结果存入 RZ。
影响标志位	无影响
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001	00000	RX	011000	01000	RZ						

14.181 RFI——快速中断返回指令

统一化指令	
语法	rfi
操作	快速中断返回 $PC \leftarrow FPC, PSR \leftarrow FPSR$
编译结果	仅存在 32 位指令。 rfi 32
属性：	特权指令
说明	PC 值恢复为保存在控制寄存器 FPC 中的值，PSR 值恢复为保存在 FPSR 的值，指令执行从新的 PC 地址处开始。
影响标志位	无影响
异常	特权违反异常

32 位指令	
操作	快速中断返回 $PC \leftarrow FPC, PSR \leftarrow FPSR$
语法	rfi32
属性：	特权指令
说明	PC 值恢复为保存在控制寄存器 FPC 中的值，PSR 值恢复为保存在 FPSR 的值，指令执行从新的 PC 地址处开始。
影响标志位	无影响
异常	特权违反异常

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10000	00000	00000	010001	00001	00000						

14.182 ROTL——循环左移指令

统一化指令		
语 法	rotl rz, rx	rotl rz, rx, ry
操 作	$RZ \leftarrow RZ \lllll RX[5:0]$	$RZ \leftarrow RX \lllll RY[5:0]$
编 译 结 果	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (x<16) and (z<16), then rotl16 rz, rx ; else rotl32 rz, rz, rx;	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (x==z) and (y<16) and (z<16), then rotl16 rz, ry else rotl32 rz, rx, ry
说 明	对于 rotl rz, rx, 将 RZ 的值进行循环左移（原值左移，右侧移入左侧移出的位），结果存入 RZ，左移位数由 RX 低 6 位（RX[5:0]）的值决定；如果 RX[5:0] 的值大于 31，那么 RZ 将被清零。 对于 rotl rz, rx, ry, 将 RX 的值进行循环左移（原值左移，右侧移入左侧移出的位），结果存入 RZ，左移位数由 RY 低 6 位（RY[5:0]）的值决定；如果 RY[5:0] 的值大于 31，那么 RZ 将被清零。	
影 响 标 志 位	无影响	
异 常	无	

16 位指令	
操作	$RZ \leftarrow RZ \lllll RX[5:0]$
语法	rotl16 rz, rx
说明	将 RZ 的值进行循环左移（原值左移，右侧移入左侧移出的位），结果存入 RZ，左移位数由 RX 低 6 位（RX[5:0]）的值决定；如果 RX[5:0] 的值大于 31，那么 RZ 将被清零。
影 响 标 志 位	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

15	14	10	9	6	5	2	1	0
0	11100			RZ		RX		11

32 位指令	
操作	$RZ \leftarrow RX \llll RY[5:0]$
语法	rotl32 rz, rx, ry
说明	将 RX 的值进行循环左移（原值左移，右侧移入左侧移出的位），结果存入 RZ，左移位数由 RY 低 6 位（RY[5:0]）的值决定；如果 RY[5:0] 的值大于 31，那么 RZ 将被清零。
影 响 标 志 位	无影响
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001		RY		RX		010000		01000		RZ	

14.183 ROTLI——立即数循环左移指令

统一化指令	
语法	rotli rz, rx, imm5
操作	$RZ \leftarrow RX \llll IMM5$
编 译 结果	rotli32 rz, rx, imm5;
说明	将 RX 的值进行循环左移（原值左移，右侧移入左侧移出的位），结果存入 RZ，左移位数由 5 位立即数（IMM5）的值决定；如果 IMM5 的值等于 0，那么 RZ 的值将与 RX 相同。
影 响 标 志 位	无影响
限制	立即数的范围为 0-31。
异常	无

32 位指令	
操作	$RZ \leftarrow RX \llll IMM5$
语法	rotli32 rz, rx, imm5
说明	将 RX 的值进行循环左移（原值左移，右侧移入左侧移出的位），结果存入 RZ，左移位数由 5 位立即数（IMM5）的值决定；如果 IMM5 的值等于 0，那么 RZ 的值将与 RX 相同。
影响标志位	无影响
限制	立即数的范围为 0-31。
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001	IMM5		RX		010010		01000			RZ	

14.184 RSUB——反向减法指令

统一化指令	
语法	rsub rz, rx, ry
操作	$RZ \leftarrow RY - RX$
编译结果	仅存在 32 位指令。 rsub32 rz, rx, ry
说明	将 RY 的值减去 RX 值，并把结果存在 RZ 中。 注意，该指令是 subu rz, ry, rx 的伪指令。
影响标志位	无影响
异常	无

32 位指令	
操作	$RZ \leftarrow RY - RX$
语法	rsub32 rz, rx, ry
说明	将 RY 的值减去 RX 值，并把结果存在 RZ 中。 注意，该指令是 subu32 rz, ry, rx 的伪指令。
影响标志位	无影响
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001	RX		RY		000000		00100		RZ		

14.185 RTS——子程序返回指令

统一化指令	
语法	rts
操作	程序跳转到链接寄存器指定的位置 $PC \leftarrow R15 \& 0\text{xffffffe}$
编译结果	总是编译为 16 位指令。 rts16
说明	程序跳转到链接寄存器 R15 指定的位置，链接寄存器的最低位被忽略。RTS16 指令的跳转范围是全部 4GB 地址空间。 该指令用于实现子程序返回功能。 注意，该指令是 jmp r15 的伪指令。
影响标志位	无影响
异常	无

16 位指令	
操作	程序跳转到链接寄存器指定的位置 $PC \leftarrow R15 \& 0\text{xffffffe}$
语法	rts16
说明	程序跳转到链接寄存器 R15 指定的位置，链接寄存器的最低位被忽略。RTS16 指令的跳转范围是全部 4GB 地址空间。 该指令用于实现子程序返回功能。 注意，该指令是 jmp16 r15 的伪指令。
影响标志位	无影响
异常	无

16 位指令格式：

15	14	10	9	6	5	2	1	0
0	11110	0000		1111		00		

32 位指令	
操作	程序跳转到链接寄存器指定的位置 $PC \leftarrow R15 \& 0\text{ffffffe}$
语法	rts32
说明	程序跳转到链接寄存器 R15 指定的位置，链接寄存器的最低位被忽略。RTS 指令的跳转范围是全部 4GB 地址空间。 该指令用于实现子程序返回功能。 注意，该指令是 jmp32 r15 的伪指令。
影 响 标 志 位	无影响
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	0
1	11010	00110	01111	0000000000000000				

14.186 RTE——异常和普通中断返回指令

统一化指令	
语法	rte
操作	异常和普通中断返回 $PC \leftarrow EPC, PSR \leftarrow EPSR$
编译结果	仅存在 32 位指令。 rte32
属性：	特权指令
说明	PC 值恢复为保存在控制寄存器 EPC 中的值，PSR 值恢复为保存在 EPSR 的值，指令执行从新的 PC 地址处开始。
影响标志位	无影响
异常	特权违反异常

32 位指令	
操作	异常和普通中断返回 $PC \leftarrow EPC, PSR \leftarrow EPSR$
语法	rte32
属性:	特权指令
说明	PC 值恢复为保存在控制寄存器 EPC 中的值, PSR 值恢复为保存在 EPSR 的值, 指令执行从新的 PC 地址处开始。
影响标志位	无影响
异常	特权违反异常

32 位指令格式:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10000	00000	00000	010000	00001	00000						

14.187 SEXT——位提取并有符号扩展指令

统一化指令	
语 法	sext rz, rx, msb, lsb
操 作	$RZ \leftarrow \text{sign\_extend}(RX[MSB:LSB])$
编 译 结 果	仅存在 32 位指令。 sext32 rz, rx, msb, lsb
说 明	提取由 2 个 5 位立即数 (MSB,LSB) 所指定的 RX 的一段连续位 (RX[MSB:LSB]), 符号扩展至 32 位, 并把结果存入 RZ。如果 MSB 等于 31, 且 LSB 等于 0, 则 RZ 的值与 RX 相同。如果 MSB 等于 LSB, 则 RZ 的值为 RX[MSB] (即 RX[LSB]) 一位符号扩展的结果。如果 MSB 小于 LSB, 该指令的行为不可预测。
影 响 标 志 位	无影响
限 制	MSB 的范围为 0-31, LSB 的范围为 0-31, 且 MSB 应当大于等于 LSB。
异 常	无

32 位指令	
操作	$RZ \leftarrow \text{sign\_extend}(RX[MSB:LSB])$
语法	sext32 rz, rx, msb, lsb
说明	提取由 2 个 5 位立即数 (MSB,LSB) 所指定的 RX 的一段连续位 (RX[MSB:LSB]), 符号扩展至 32 位, 并把结果存入 RZ。如果 MSB 等于 31, 且 LSB 等于 0, 则 RZ 的值与 RX 相同。如果 MSB 等于 LSB, 则 RZ 的值为 RX[MSB] (即 RX[LSB]) 一位符号扩展的结果。如果 MSB 小于 LSB, 该指令的行为不可预测。
影响标志位	无影响
限制	MSB 的范围为 0-31, LSB 的范围为 0-31, 且 MSB 应当大于等于 LSB。
异常	无

32 位指令格式:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001		LSB		RX		010110		MSB		RZ	

MSB 域

指定被提取开始的位。

LSB 域

指定被提取结束的位。

00000

0

00000

0 位

00001

1

00001

1 位

... ..

11111

31

11111

31 位

14.188 SEXTB——字节提取并有符号扩展指令

统一化指令	
语法	sextb rz, rx
操作	$RZ \leftarrow \text{sign\_extend}(RX[7:0]);$
编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (z<16) and (x<16), then sextb16 rz, rx; else sextb32 rz, rx;
说明	将 RX 的低字节（RX[7:0]）符号扩展至 32 位，结果存在 RZ 中。
影响标志位	无影响
异常	无

16 位指令	
操作	$RZ \leftarrow \text{sign\_extend}(RX[7:0]);$
语法	sextb16 rz, rx
说明	将 RX 的低字节（RX[7:0]）符号扩展至 32 位，结果存在 RZ 中。
影响标志位	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

15	14	10	9	6	5	2	1	0
0	11101		RZ		RX		10	

32 位指令	
操作	$RZ \leftarrow \text{sign\_extend}(RX[7:0]);$
语法	sextb32 rz, rx
说明	将 RX 的低字节（RX[7:0]）符号扩展至 32 位，结果存在 RZ 中。 注意，该指令是 sext32 rz, rx, 0x7, 0x0 的伪指令。
影响标志位	无影响
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001	00000			RX		010110		00111			RZ

14.189 SEXTH——半字提取并有符号扩展指令

统一化指令	
语法	sextb rz, rx
操作	$RZ \leftarrow \text{sign\_extend}(RX[15:0]);$
编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (z<16) and (x<16), then sextb16 rz, rx; else sextb32 rz, rx;
说明	将 RX 的低半字（RX[15:0]）符号扩展至 32 位，结果存在 RZ 中。
影响标志位	无影响
异常	无

16 位指令	
操作	$RZ \leftarrow \text{sign\_extend}(RX[15:0]);$
语法	sextb16 rz, rx
说明	将 RX 的低半字（RX[15:0]）符号扩展至 32 位，结果存在 RZ 中。
影响标志位	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

15	14	10	9	6	5	2	1	0
0	11101		RZ		RX			11

32 位指令	
操作	$RZ \leftarrow \text{sign\_extend}(RX[15:0]);$
语法	sext32 rz, rx
说明	将 RX 的低半字（RX[15:0]）符号扩展至 32 位，结果存在 RZ 中。 注意，该指令是 sext32 rz, rx, 0x15, 0x0 的伪指令。
影响标志位	无影响
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001	00000			RX		010110		01111			RZ

14.190 SRS.B——字节符号存储指令

统一化指令	
语 法	srs.b rz, [label]
操 作	将寄存器中的最低字节符号存储到存储器中 $MEM[R28 + \text{zero\_extend}(\text{offset})] \leftarrow RZ[7:0]$
编 译 结 果	仅存在 32 位指令。 srs32.b rz, [label]
说 明	将寄存器 RZ 中的最低字节符号存储到 label 所在位置。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上 18 位相对偏移量无符号扩展到 32 位后的值得到。SRS.B 指令可以寻址 +256KB 地址空间。 注意，偏移量 DISP 即二进制操作数 Offset。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	将寄存器中的最低字节符号存储到存储器中 $MEM[R28 + zero\_extend(offset)] \leftarrow RZ[7:0]$
语法	srs32.b rz, [label]
说明	将寄存器 RZ 中的最低字节符号存储到 label 所在位置。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上 18 位相对偏移量无符号扩展到 32 位后的值得到。SRS.B 指令可以寻址 +256KB 地址空间。 注意，偏移量 DISP 即二进制操作数 Offset。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

31	30	26	25	21	20	18	17	0
1	10011		RZ	100				Offset

14.191 SRS.H——半字符存储指令

统一化指令	
语 法	srs.h rz, [label]
操 作	将寄存器中的最低半字符存储到存储器中 $MEM[R28 + zero\_extend(offset << 1)] \leftarrow RZ[7:0]$
编 译 结 果	仅存在 32 位指令。 srs32.h rz, [label]
说 明	将寄存器 RZ 中的最低半字符存储到 label 所在位置。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移 1 位的 18 位相对偏移量无符号扩展到 32 位后的值得到。SRS.H 指令可以寻址 +512KB 地址空间。 注意：偏移量 DISP 即二进制操作数 Offset。
影 响 标 志 位	无影响
异 常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操 作	将寄存器中的最低半字符存储到存储器中 $MEM[R28 + zero\_extend(offset << 1)] \leftarrow RZ[7:0]$
语 法	srs32.h rz, [label]
说 明	将寄存器 RZ 中的最低半字符存储到 label 所在位置。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 R X 加上左移 1 位的 18 位相对偏移量无符号扩展到 32 位后的值得到。SRS.H 指令可以寻址 +512KB 地址空间。 注意，偏移量 DISP 即二进制操作数 Offset。
影 响 标 志 位	无影响
异 常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

31	30	26	25	21	20	18	17	0
1	10011		RZ		101			Offset

14.192 SRS.W——字符符号存储指令

统一化指令	
语 法	srs.w rz, [label]
操 作	将寄存器中的最低字符符号存储到存储器中 MEM[R28 + zero_extend(offset << 2)] ← RZ[7:0]
编 译 结 果	仅存在 32 位指令。 srs32.w rz, [label]
说 明	将寄存器 RZ 中的最低字符符号存储到到 label 所在位置。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 R X 加上左移 2 位的 18 位相对偏移量无符号扩展到 32 位后的值得到。SRS.W 指令可以寻址 +1024KB 地址空间。 注意，偏移量 DISP 即二进制操作数 Offset。
影 响 标 志 位	无影响
异 常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	将寄存器中的最低字符存储到存储器中 $MEM[R28 + zero\_extend(offset \ll 2)] \leftarrow RZ[7:0]$
语法	srs32.w rz, [label]
说明	将寄存器 RZ 中的最低字符存储到 label 所在位置。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 R X 加上左移 2 位的 18 位相对偏移量无符号扩展到 32 位后的值得到。SRS.W 指令可以寻址 +1024KB 地址空间。 注意，偏移量 DISP 即二进制操作数 Offset。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

31	30	26	25	21	20	18	17	0
1	10011		RZ	110	Offset			

14.193 ST.B——字节存储指令

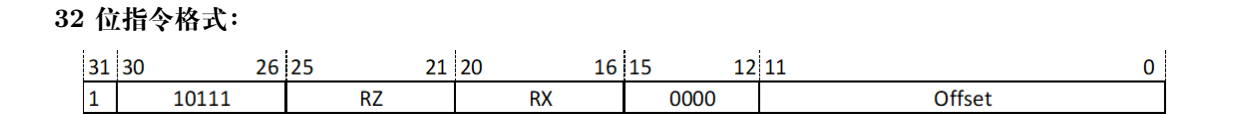
统一化指令	
语 法	st.b rz, (rx, disp)
操 作	将寄存器中的最低字节存储到存储器中 $MEM[RX + zero\_extend(offset)] \leftarrow RZ[7:0]$
编 译 结 果	根据偏移量和寄存器的范围编译为对应的 16 位或 32 位指令。 if (disp<32) and (x<7) and (z<7), then st16.b rz, (rx, disp) ; else st32.b rz, (rx, disp) ;
说 明	将寄存器 RZ 中的最低字节存储到存储器中。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上 12 位相对偏移量无符号扩展到 32 位后的值得到。ST.B 指令可以寻址 +4KB 地址空间。 注意，偏移量 DISP 即二进制操作数 Offset。
影 响 标 志 位	无影响
异 常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

16 位指令	
操作	将寄存器中的最低字节存储到存储器中 $MEM[RX + zero\_extend(offset)] \leftarrow RZ[7:0]$
语法	st16.b rz, (rx, disp)
说明	将寄存器 RZ 中的最低字节存储到存储器中。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上 5 位相对偏移量无符号扩展到 32 位后的值得到。ST16.B 指令可以寻址 +32B 的空间。 注意，偏移量 DISP 即二进制操作数 Offset。
影响标志位	无影响
限制	寄存器的范围为 r0-r7。
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

16 位指令格式：

15	14	11	10	8	7	5	4	0
1	0100			RX		RZ		IMM5

32 位指令	
操作	将寄存器中的最低字节存储到存储器中 $MEM[RX + zero\_extend(offset)] \leftarrow RZ[7:0]$
语法	st32.b rz, (rx, disp)
说明	将寄存器 RZ 中的最低字节存储到存储器中。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上 12 位相对偏移量无符号扩展到 32 位后的值得到。ST32.B 指令可以寻址 +4KB 地址空间。 注意，偏移量 DISP 即二进制操作数 Offset。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常



14.194 ST.D——双字存储指令

统一化指令	
语 法	st.d rz, (rx, disp)
操 作	将寄存器中的双字存储到存储器中 $MEM[RX + zero\_extend(offset \ll 2)] \leftarrow RZ[31:0]$ $MEM[RX + zero\_extend(offset \ll 2) + 0x4] \leftarrow RZ + 1[31:0]$
编 译 结 果	仅存在 32 位指令 st32.d rz, (rx, disp) ;
说 明	将寄存器 RZ、RZ + 1 中的双字存储到存储器中。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移两位的 12 位相对偏移量无符号扩展到 32 位后的值得到。ST.D 指令可以寻址 +16KB 地址空间。 注意，偏移量 DISP 是二进制操作数 Offset 左移两位得到的。
影 响 标 志 位	无影响
异 常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令	
操作	将寄存器中的双字存储到存储器中 $MEM[RX + zero\_extend(offset \ll 2)] \leftarrow RZ[31:0]$ $MEM[RX + zero\_extend(offset \ll 2) + 0x4] \leftarrow RZ + 1[31:0]$
语法	st32.d rz, (rx, disp)
说明	将寄存器 RZ、RZ + 1 中的双字存储到存储器中。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移两位的 12 位相对偏移量无符号扩展到 32 位后的值得到。ST32.D 指令可以寻址 +16KB 地址空间。 注意，偏移量 DISP 是二进制操作数 Offset 左移两位得到的。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令格式：

31	30	26	25	21	20	16	15	12	11	0
1	10111	RZ			RX	0011		Offset		

14.195 ST.H——半字存储指令

统一化指令	
语法	st.h rz, (rx, disp)
操作	将寄存器中的最低字节存储到存储器中 $MEM[RX + zero\_extend(offset << 1)] \leftarrow RZ[15:0]$
编译结果	根据偏移量和寄存器的范围编译为对应的 16 位或 32 位指令。 if (disp<64)and(x<7)and(z<7), then st16.h rz, (rx, disp) ; else st32.h rz, (rx, disp) ;
说明	将寄存器 RZ 中的低半字存储到存储器中。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移 1 位的 12 位相对偏移量无符号扩展到 32 位后的值得到。ST.H 指令可以寻址 +8KB 地址空间。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

16 位指令	
操作	将寄存器中的低半字存储到存储器中 $MEM[RX + zero\_extend(offset << 1)] \leftarrow RZ[15:0]$
语法	st16.h rz, (rx, disp)
说明	将寄存器 RZ 中的低半字存储到存储器中。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移 1 位的 5 位相对偏移量无符号扩展到 32 位后的值得到。ST16.H 指令可以寻址 +64B 的空间。 注意，偏移量 DISP 是二进制操作数 Offset 左移 1 位得到的。
影响标志位	无影响
限制	寄存器的范围为 r0-r7。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

16 位指令格式：

15	14	11	10	8	7	5	4	0
1	0101	RX	RZ	IMM5				

32 位指令	
操作	将寄存器中的低半字存储到存储器中 $MEM[RX + zero\_extend(offset \ll 1)] \leftarrow RZ[15:0]$
语法	st32.h rz, (rx, disp)
说明	将寄存器 RZ 中的低半字存储到存储器中。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移 1 位的 12 位相对偏移量无符号扩展到 32 位后的值得到。ST32.H 指令可以寻址 +8KB 地址空间。 注意，偏移量 DISP 是二进制操作数 Offset 左移 1 位得到的。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令格式：

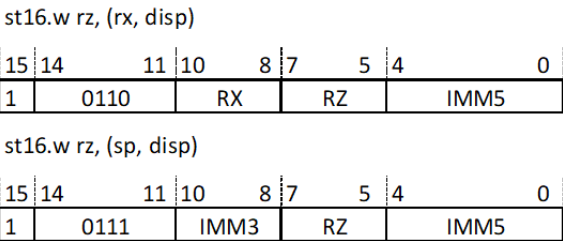
31	30	26	25	21	20	16	15	12	11	0
1	10111	RZ	RX	0001	Offset					

14.196 ST.W——字存储指令

统一化指令	
语法	st.w rz, (rx, disp)
操作	将寄存器中的字存储到存储器中 MEM[RX + zero_extend(offset<< 2)] ← RZ[31:0]
编译结果	根据偏移量和寄存器的范围编译为对应的 16 位或 32 位指令。 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) ; else st32.w rz, (rx, disp) ;
说明	将寄存器 RZ 中的字存储到存储器中。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移两位的 12 位相对偏移量无符号扩展到 32 位后的值得到。ST.W 指令可以寻址 +16KB 地址空间。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

16 位指令	
操作	将寄存器中的字存储到存储器中 $MEM[RX + zero\_extend(offset \ll 2)] \leftarrow RZ[31:0]$
语法	st16.w rz, (rx, disp) st16.w rz, (sp, disp)
说明	将寄存器 RZ 中的字存储到存储器中。采用寄存器加无符号立即数偏移量的寻址方式。当 rx= sp 时，存储器的有效地址由基址寄存器 RX 加上左移两位的 8 位相对偏移量无符号扩展到 32 位后的值得到。当 rx 为其它寄存器时，存储器的有效地址由基址寄存器 RX 加上左移两位的 5 位相对偏移量无符号扩展到 32 位后的值得到。ST16.W 指令可以寻址 +1KB 的空间。 注意，偏移量 DISP 是二进制操作数 IMM5 左移两位得到的。当基址寄存器 RX 为 SP 时，偏移量 DISP 是二进制操作数 {IMM3, IMM5} 左移两位得到的。
影响标志位	无影响
限制	寄存器的范围为 r0-r7。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

16 位指令格式：



32 位指令	
操作	将寄存器中的字存储到存储器中 $MEM[RX + zero\_extend(offset \ll 2)] \leftarrow RZ[31:0]$
语法	st32.w rz, (rx, disp)
说明	将寄存器 RZ 中的字存储到存储器中。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移两位的 12 位相对偏移量无符号扩展到 32 位后的值得到。ST32.W 指令可以寻址 +16KB 地址空间。 注意，偏移量 DISP 是二进制操作数 Offset 左移两位得到的。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令格式：

31	30	26	25	21	20	16	15	12	11	0
1	10111		RZ		RX		0010		Offset	

14.197 STCPR——协处理器字存储指令

统一化指令	
语法	stcpr <cpid, cprz>, (rx, offset)
操作	将协处理器通用寄存器中的字存储到存储器中 MEM[RX + sign_extend(offset << 2)] ← CPRZ
编译结果	仅存在 32 位指令。 stcpr32 <cpid, cprz>, (rx, offset)
说明	将协处理器通用寄存器 CPRZ 中的字存储到存储器中。采用寄存器加立即数偏移量的寻址方式。指令低 12 位编码空间中 8~11 位约定为协处理器号，用于指定预操作的协处理器，其余 8 位为相对偏移量。存储器的有效地址由主流水线基址寄存器 RX 加上左移两位的 8 位相对偏移量无符号扩展到 32 位后的值得到。STCPR 指令可以寻址 +1KB 地址空间。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

指令格式：

31	30	26	25	21	20	16	15	12	11	0
1	11010	CPRZ		RX		1000		User-define		

14.198 STEX.W——独占式字存储指令

统一化指令	
语 法	stex.w rz, (rx, disp)
操 作	if 独占式字存储成功, then MEM[RX + sign_extend(offset << 2)] ← RZ; RZ ← 1; else RZ ← 0;
编 译 结 果	仅存在 32 位指令。 stex32.w rz, (rx, disp)
说 明	将通用寄存器 RZ 中的字存储到存储器中，若独占存储成功，则源寄存器 RZ 返回 1；否则源寄存器返回 0 表示独占存储失败。STEX.W 采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移两位的 12 相对偏移量无符号扩展到 32 位后的值得到。STEX.W 指令可以寻址 +16KB 空间。 该指令与 LDEX.W 配对，用于多核通信时的原子“读存储器—修改—写存储器”操作。 注意，偏移量 DISP 是二进制操作数 Offset 左移两位得到的。
影 响 标 志 位	无影响
异 常	未对齐访问异常、未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令	
操作	if 独占式字存储成功, then MEM[RX + sign_extend(offset << 2)] ← RZ; RZ ← 1; else RZ ← 0;
语法	stex32.w rz, (rx, disp)
说明	将通用寄存器 RZ 中的字存储到存储器中，若独占存储成功，则源寄存器 RZ 返回 1；否则源寄存器返回 0 表示独占存储失败。STEX32.W 采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移两位的 12 相对偏移量无符号扩展到 32 位后的值得到。STEX32.W 指令可以寻址 +16KB 空间。 该指令与 LDEX32.W 配对，用于多核通信时的原子“读存储器—修改—写存储器”操作。 注意，偏移量 DISP 是二进制操作数 Offset 左移两位得到的。
影响标志位	无影响
异常	未对齐访问异常、未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令格式：

31	30	26	25	21	20	16	15	12	11	0
1	10111	RZ	RX	0111	Offset					

14.199 STM——连续多字存储指令

统一化指令	
语 法	stm ry-rz, (rx)
操 作	将一片连续的寄存器堆中内容依次存储到一片连续的存储器地址上。 $src \leftarrow Y$; $addr \leftarrow RX$; for ($n = 0$; $n \leq (Z-Y)$; $n++$){ MEM[addr] $\leftarrow$ Rsrc; $src \leftarrow src + 1$; $addr \leftarrow addr + 4$; }
编 译 结 果	仅存在 32 位指令。 stm32 ry-rz, (rx)
说 明	将从 RY 开始的一片连续的寄存器堆中的内容依次存储到一片连续的存储器地址上，即将寄存器 RY 的内容存到存储器指定地址开始的第一个字的地址上，寄存器 RY+1 的内容存到存储器指定地址开始的第二个字的地址上，依次类推，将寄存器 RZ 的内容存到存储器指定地址开始的最后一个字的地址上。存储器的有效地址由基址寄存器 RX 的内容决定。
影 响 标 志 位	无影响
限 制	RZ 应当大于等于 RY。
异 常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令	
操作	将一片连续的寄存器堆中内容依次存储到一片连续的存储器地址上。 src ← Y; addr ← RX; for (n = 0; n <= IMM5; n++){ MEM[addr] ← Rsrc; src ← src + 1; addr ← addr + 4; }
语法	stm32 ry-rz, (rx)
说明	将从 RY 开始的一片连续的寄存器堆中的内容依次存储到一片连续的存储器地址上，即将寄存器 RY 的内容存到存储器指定地址开始的第一个字的地址上，寄存器 RY+1 的内容存到存储器指定地址开始的第二个字的地址上，依次类推，将寄存器 RZ 的内容存到存储器指定地址开始的最后一个字的地址上。存储器的有效地址由基址寄存器 RX 的内容决定。
影响标志位	无影响
限制	RZ 应当大于等于 RY。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10101			RY		RX		000111		00001		IMM5

IMM5 域：

指定目标寄存器的个数，IMM5 = Z - Y。

00000：

1 个目的寄存器

00001：

2 个目的寄存器

.....

11111：

32 个目的寄存器

14.200 STOP——进入低功耗暂停模式指令

统一化指令	
语法	stop
操作	进入低功耗暂停模式
编译结果	仅存在 32 位指令。 stop32
说明	此指令使处理器进入低功耗模式，并等待一个中断来退出这个模式。此时，CPU 时钟停止，大部分外围设备也被停止。
影响标志位	无影响
异常	特权违反异常

32 位指令	
操作	进入低功耗暂停模式
语法	stop32
属性：	特权指令
说明	此指令使处理器进入低功耗模式，并等待一个中断来退出这个模式。此时，CPU 时钟停止，大部分外围设备也被停止。
影响标志位	无影响
异常	特权违反异常

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10000	00000	00000	010010	00001	00000						

14.201 STQ——连续四字存储指令

统一化指令	
语 法	stq r4-r7, (rx)
操 作	将寄存器 R 4—R7 中的字依次存储到一片连续的存储器地址上。 src ← 4; addr ← RX; for (n = 0; n <= 3; n++){ MEM[addr] ← Rsrc; src ← src + 1; addr ← addr + 4; }
编 译 结 果	仅存在 32 位指令。 stq32 r4-r7, (rx);
说 明	将寄存器堆 [R4,R7]（包括边界）中的字依次存储到一片连续的存储器地址上，即将寄存器 R4 的内容存到存储器指定地址开始的第一个字的地址上，寄存器 R5 的内容存到存储器指定地址开始的第二个字的地址上，寄存器 R6 的内容存到存储器指定地址开始的第三个字的地址上，寄存器 R7 的内容存到存储器指定地址开始的第四个字的地址上。存储器的有效地址由基址寄存器 RX 的内容决定。 注意，该指令是 stm r4-r7, (rx) 的伪指令。
影 响 标 志 位	无影响
异 常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令	
操作	将寄存器 R 4—R7 中的字依次存储到一片连续的存储器地址上。 src ← 4; addr ← RX; for (n = 0; n <= 3; n++){ MEM[addr] ← Rsrc; src ← src + 1; addr ← addr + 4; }
语法	stq32 r4-r7, (rx)
说明	将寄存器堆 [R4,R7]（包括边界）中的字依次存储到一片连续的存储器地址上，即将寄存器 R4 的内容存到存储器指定地址开始的第一个字的地址上，寄存器 R5 的内容存到存储器指定地址开始的第二个字的地址上，寄存器 R6 的内容存到存储器指定地址开始的第三个字的地址上，寄存器 R7 的内容存到存储器指定地址开始的第四个字的地址上。存储器的有效地址由基址寄存器 RX 的内容决定。 注意，该指令是 stm r4-r7, (rx) 的伪指令。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10101	00100		RX		000111		00001		00011		

14.202 STR.B——寄存器移位寻址字节存储指令

统一化指令	
语法	str.b rz, (rx, ry << 0) str.b rz, (rx, ry << 1) str.b rz, (rx, ry << 2) str.b rz, (rx, ry << 3)
操作	将寄存器中的最低字节存储到存储器中 $MEM[RX + RY \ll IMM2] \leftarrow RZ[7:0]$
编译结果	仅存在 32 位指令。 str32.b rz, (rx, ry << 0) str32.b rz, (rx, ry << 1) str32.b rz, (rx, ry << 2) str32.b rz, (rx, ry << 3)
说明	将寄存器 RZ 中的最低字节存储到存储器中。采用寄存器加寄存器移位寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值为 0。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令	
操作	将寄存器中的最低字节存储到存储器中 $MEM[RX + RY \ll IMM2] \leftarrow RZ[7:0]$
语法	str32.b rz, (rx, ry << 0) str32.b rz, (rx, ry << 1) str32.b rz, (rx, ry << 2) str32.b rz, (rx, ry << 3)
说明	将寄存器 RZ 中的最低字节存储到存储器中。采用寄存器加寄存器移位寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值为 0。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令格式：

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		
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		
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		
str32.b rz, (rx, ry << 3)															
31	30	26	25	21	20	16	15	10	9	5	4	0			
1	10101			RY		RX		000000			01000		RZ		

14.203 STR.D——寄存器移位寻址双字存储指令

统一化指令	
语 法	str.d rz, (rx, ry << 0) str.d rz, (rx, ry << 1) str.d rz, (rx, ry << 2) str.d rz, (rx, ry << 3)
操 作	将寄存器中的双字存储到存储器中 $MEM[RX + zero\_extend(offset << 2)] \leftarrow RZ[31:0]$ $MEM[RX + zero\_extend(offset << 2) + 0x4] \leftarrow RZ + 1[31:0]$
编 译 结 果	仅存在 32 位指令。 str32.d rz, (rx, ry << 0) str32.d rz, (rx, ry << 1) str32.d rz, (rx, ry << 2) str32.d rz, (rx, ry << 3)
说 明	将寄存器 RZ 和 RZ+1 中的字存储到存储器中。采用寄存器加寄存器移位寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值 0。注意：rz、rz+1、rx 和 ry 不能相同，否则结果不可预期。
影 响 标 志 位	无影响
异 常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令	
操作	将寄存器中的双字存储到存储器中
语法	str32.d rz, (rx, ry << 0) str32.d rz, (rx, ry << 1) str32.d rz, (rx, ry << 2) str32.d rz, (rx, ry << 3)
说明	将寄存器 RZ 和 RZ+1 中的字存储到存储器中。采用寄存器加寄存器移位寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值 0。注意：rz、rz+1、rx 和 ry 不能相同，否则结果不可预期。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令格式：

str32.w rz, (rx, ry << 0)															
31	30	26	25	21	20	16	15	10	9	5	4	0			
1	10101			RY		RX		000011		00001		RZ			
str32.w rz, (rx, ry << 1)															
31	30	26	25	21	20	16	15	10	9	5	4	0			
1	10101			RY		RX		000011		00010		RZ			
str32.w rz, (rx, ry << 2)															
31	30	26	25	21	20	16	15	10	9	5	4	0			
1	10101			RY		RX		000011		00100		RZ			
str32.w rz, (rx, ry << 3)															
31	30	26	25	21	20	16	15	10	9	5	4	0			
1	10101			RY		RX		000011		01000		RZ			

14.204 STR.H——寄存器移位寻址半字存储指令

统一化指令	
语法	str.h rz, (rx, ry << 0) str.h rz, (rx, ry << 1) str.h rz, (rx, ry << 2) str.h rz, (rx, ry << 3)
操作	将寄存器中的低半字存储到存储器中 $MEM[RX + RY \ll IMM2] \leftarrow RZ[15:0]$
编译结果	仅存在 32 位指令。 str32.h rz, (rx, ry << 0) str32.h rz, (rx, ry << 1) str32.h rz, (rx, ry << 2) str32.h rz, (rx, ry << 3)
说明	将寄存器 RZ 中的低半字存储到存储器中。采用寄存器加寄存器移位寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值为 0。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令	
操作	将寄存器中的低半字存储到存储器中 $MEM[RX + RY \ll IMM2] \leftarrow RZ[15:0]$
语法	str32.h rz, (rx, ry << 0) str32.h rz, (rx, ry << 1) str32.h rz, (rx, ry << 2) str32.h rz, (rx, ry << 3)
说明	将寄存器 RZ 中的低半字存储到存储器中。采用寄存器加寄存器移位寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值为 0。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令格式：

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
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
str32.h rz, (rx, ry << 2)													
31	30	26	25	21	20	16	15	10	9	5	4	0	
1	10101			RY		RX		000001			00100		RZ
str32.h rz, (rx, ry << 3)													
31	30	26	25	21	20	16	15	10	9	5	4	0	
1	10101			RY		RX		000001			01000		RZ

14.205 STR.W——寄存器移位寻址字存储指令

统一化指令	
语法	str.w rz, (rx, ry << 0) str.w rz, (rx, ry << 1) str.w rz, (rx, ry << 2) str.w rz, (rx, ry << 3)
操作	将寄存器中的字存储到存储器中 MEM[RX + RY << IMM2] ← RZ[31:0]
编译结果	仅存在 32 位指令。 str32.w rz, (rx, ry << 0) str32.w rz, (rx, ry << 1) str32.w rz, (rx, ry << 2) str32.w rz, (rx, ry << 3)
说明	将寄存器 RZ 中的字存储到存储器中。采用寄存器加寄存器移位寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值为 0。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令	
操作	将寄存器中的字存储到存储器中 $MEM[RX + RY \ll IMM2] \leftarrow RZ[31:0]$
语法	$str32.w\ rz, (rx, ry \ll 0)$ $str32.w\ rz, (rx, ry \ll 1)$ $str32.w\ rz, (rx, ry \ll 2)$ $str32.w\ rz, (rx, ry \ll 3)$
说明	将寄存器 RZ 中的字存储到存储器中。采用寄存器加寄存器移位寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值为 0。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令格式：

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
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
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
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

14.206 SUBC——无符号带借位减法指令

统一化指令		
语 法	subc rz, rx	subc rz, rx, ry
操 作	$RZ \leftarrow RZ - RX - (!C)$, $C \leftarrow$ 借位	$RZ \leftarrow RX - RY - (!C)$, $C \leftarrow$ 借位
编 译 结 果	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (x<16) and (z<16), then subc16 rz, rx; else subc32 rz, rz, rx;	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (x==z) and (y<16) and (z<16), then subc16 rz, ry; else subc32 rz, rx, ry;
说 明	对于 subc rz, rx, 将 RZ 的值减去寄存器 RX 的值和 C 位的非值; 对于 subcrz, rx, ry, 将 RX 的值减去寄存器 RY 的值和 C 位的非值。把结果存在 RZ, 借位存在 C 位。对于该减法指令来说, 如果发生借位, 将清 C 位, 反之置 C 位。	
影 响 标 志 位	$C \leftarrow$ 借位	
异 常	无	

16 位指令	
操作	$RZ \leftarrow RZ - RX - (!C)$, $C \leftarrow$ 借位
语法	subc16 rz, rx
说明	将 RZ 的值减去寄存器 RX 的值和 C 位的非值, 并把结果存在 RZ, 借位存在 C 位。对于该减法指令来说, 如果发生借位, 将清 C 位, 反之置 C 位。
影响标志位	$C \leftarrow$ 借位
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式:

15	14	10	9	6	5	2	1	0
0	11000			RZ		RX		11

32 位指令	
操作	$RZ \leftarrow RX - RY - (!C), C \leftarrow \text{借位}$
语法	subc32 rz, rx, ry
说明	将 RX 的值减去寄存器 RY 的值和 C 位的非值，并把结果存在 RZ，借位存在 C 位。对于该减法指令来说，如果发生借位，将清 C 位，反之置 C 位。
影响标志位	$C \leftarrow \text{借位}$
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001			RY		RX		000000		01000		RZ

14.207 SUBI——无符号立即数减法指令

统一化指令		
语 法	ubi rz, oimm12	subi rz, rx, oimm12
S 操作	$RZ \leftarrow RZ - \text{zero\_extend}(OIMM12)$	$RZ \leftarrow RX - \text{zero\_extend}(OIMM12)$
编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (oimm12<257) and (z<8), then subi16 rz, oimm8; else subi32 rz, rz, oimm12;	根据寄存器的范围编译为对应的 16 位或 32 位指令。 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;
说明	将带偏置 1 的 12 位立即数（OIMM12）零扩展至 32 位，然后用 RZ/RX 的值减去该 32 位数，把结果存入 RZ。	
影响标志位	无影响	
限制	立即数的范围为 0x1-0x1000。	
异常	无	

16 位指令 1	
操作	$RZ \leftarrow RZ - \text{zero\_extend}(OIMM8)$
语法	subi16 rz, oimm8
说明	将带偏置 1 的 8 位立即数（OIMM8）零扩展至 32 位，然后用 RZ 的值减去该 32 位数，把结果存入 RZ。 注意：二进制操作数 IMM8 等于 OIMM8 - 1。
影响标志位	无影响
限制	寄存器的范围为 r0-r7；立即数的范围为 1-256。
异常	无

16 位指令格式 1：

15	14	11	10	8	7	0
0	0101			RZ		IMM8

IMM8 域：

指定不带偏置立即数的值。

注意：寄存器减去的值 OIMM8 比起二进制操作数 IMM8 需偏置 1。

00000000：

减 1

00000001：

减 2

.....

11111111：

减 256

16 位指令 2	
操作	$RZ \leftarrow RX - \text{zero\_extend}(OIMM3)$
语法	subi16 rz, rx, oimm3
说明	将带偏置 1 的 3 位立即数（OIMM3）零扩展至 32 位，然后用 RX 的值减去该 32 位数，把结果存入 RZ。 注意：二进制操作数 IMM3 等于 OIMM3 - 1。
影响标志位	无影响
限制	寄存器的范围为 r0-r7；立即数的范围为 1-8。
异常	无

16 位指令格式 2:

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

IMM3 域

指定不带偏置立即数的值。

注意：寄存器减去的值 OIMM3 比起二进制操作数 IMM3 需偏置 1。

000

减 1

001

减 2

.....

111

减 8

32 位指令	
操作	$RZ \leftarrow RX - \text{zero\_extend}(OIMM12)$
语法	subi32 rz, rx, oimm12
说明	将带偏置 1 的 12 位立即数（OIMM12）零扩展至 32 位，然后用 RX 的值减去该 32 位数，把结果存入 RZ。 注意：二进制操作数 IMM12 等于 OIMM12 - 1。
影响标志位	无影响
限制	立即数的范围为 0x1-0x1000。
异常	无

32 位指令格式:

31	30	26	25	21	20	16	15	12	11	0
1	11001	RZ	RX	0001	IMM12					

IMM12 域

指定不带偏置立即数的值。

注意：寄存器减去的值 OIMM12 比起二进制操作数 IMM12 需偏置 1。

000000000000

减 0x1

000000000001

减 0x2

.....

111111111111

减 0x1000

14.208 SUBI(SP)——无符号（堆栈指针）立即数减法指令

统一化指令	
语法	subi sp, sp, imm
操作	SP ← SP- zero_extend(IMM)
编译结果	仅存在 16 位指令。 subi sp, sp, imm
说明	将立即数（IMM）零扩展至 32 位并左移 2 位，然后与堆栈指针（SP）的值相减，把结果存入 SP。
影响标志位	无影响
限制	立即数的范围为 0x0-0x1fc。
异常	无

16 位指令	
操作	SP ← SP - zero_extend(IMM)
语法	subi sp, sp, imm
说明	将立即数（IMM）零扩展至 32 位并左移 2 位，然后与堆栈指针（SP）的值相减，把结果存入堆栈指针。 注意：立即数（IMM）等于二进制操作数 {IMM2, IMM5} << 2。
影响标志位	无影响
限制	源与目的寄存器均为堆栈指令寄存器（R14）；立即数的范围为 (0x0-0x7f) << 2。
异常	无

16 位指令格式：

15	14	11	10	9	8	7	5	4	0
0	0010		1	IMM2		001		RZ	

IMM 域：

指定不带移位的立即数的值。

注意：加到寄存器里的值 IMM 比起二进制操作数 {IMM2, IMM5} 需左移 2 位。

{00, 00000}

减 0x0

{00, 00001}

减 0x4

....

{11, 11111}

减 0x1fc

14.209 SUBU——无符号减法指令

统一化指令		
语 法	subu rz, rx sub rz, rx	subu rz, rx, ry
操 作	RZ ← RZ - RX	RZ ← RX - RY
编 译 结 果	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (z<16) and (x<16), then subu16 rz, rx; else subu32 rz, rz, rx;	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (z<8) and (x<8) and (y<8), then subu16 rz, rx, ry; elsif (x==z) and (z<16) and (y<16), then subu16 rz, ry; else subu32 rz, rx, ry;
说 明	对于 subu rz, rx, 将 RZ 的值减去 RX 值, 并把结果存在 RZ 中。 对于 subu rz, rx, ry, 将 RX 的值减去 RY 值, 并把结果存在 RZ 中。	
影 响 标 志 位	无影响	
异 常	无	

16 位指令 1	
操作	$RZ \leftarrow RZ - RX$
语法	subu16 rz, rx sub16 rz, rx
说明	将 RZ 的值减去 RX 值，并把结果存在 RZ 中。
影响标志位	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式 1:

15	14	10	9	6	5	2	1	0
0	11000			RZ	RX		1	0

16 位指令 2	
操作	$RZ \leftarrow RX - RY$
语法	subu16 rz, rx, ry sub16 rz, rx, ry
说明	将 RX 的值减去 RY 值，并把结果存在 RZ 中。
影响标志位	无影响
限制	寄存器的范围为 r0-r7。
异常	无

16 位指令格式 2:

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

32 位指令	
操作	$RZ \leftarrow RX - RY$
语法	subu32 rz, rx, ry
说明	将 RX 的值减去 RY 值，并把结果存在 RZ 中。
影响标志位	无影响
异常	无

32 位指令格式:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001			RY		RX		000000		00100		RZ

14.210 SYNC——CPU 同步指令

统一化指令	
语法	sync.is sync.i sync.s sync
操作	该指令保证前序所有指令比该指令早退休，后续所有指令比该指令晚退休。
编译结果	仅存在 32 位指令。 sync32.is sync32.i sync32.s sync32
说明	S 表征是否广播到 CPU 核外，“1”为有效。 I 表征指令退休时是否同步取值，执行流水线清空操作，“1”为有效。
影响标志位	无影响
异常	无

32 位指令	
操作	使 CPU 同步
语法	sync32
说明	S 表征是否广播到 CPU 核外，“1”为有效。 I 表征指令退休时是否同步取值，执行流水线清空操作，“1”为有效。
影响标志位	无影响
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10000	5000I		00000		000001		00001		00000		

14.211 TLBI——TLB 无效指令

统 一 化 指 令	
语法	tlbi.vaa(s) rx tlbi.asid(s) rx tlbi.va(s) rx tlbi.all(s)
操作	删除指定 VA 的所有 TLB 表项， 删除指定 ASID 的所有 TLB 表项， 删除指定 ASID 和 VA 的 TLB 表项， 删除所有 TLB 表项。 其中，RX[31:0] = {vpn[19:0], asid[11:0]}
编译	仅存在 32 位指令。 tlbi32.vaa rx, tlbi32.asid rx, tlbi32.va rx, tlbi32.all
说明	TLBI 类型指令均可指定在共享空间内广播与否。S=1 表示需要广播到其他 CPU，为 0 表示只应用于当前 CPU。
影 响 标 志 位	无
异常	无

指令格式：

tlbi.vaa(s)												
31	30	26	25	21	20	16	15	10	9	5	4	0
1	10000	S0010		RX		100010		00001		00000		
tlbi.asid(s) rx												
31	30	26	25	21	20	16	15	10	9	5	4	0
1	10000	S0001		RX		100010		00001		00000		
tlbi.va(s) rx												
31	30	26	25	21	20	16	15	10	9	5	4	0
1	10000	S0011		RX		100010		00001		00000		
tlbi.all(s)												
31	30	26	25	21	20	16	15	10	9	5	4	0
1	10000	S0000		00000		100010		00001		00000		

14.212 TRAP——操作系统陷阱指令

统一化指令	
语法	trap 0, trap 1 trap 2, trap 3
操作	引起陷阱异常发生
编译结果说明	仅存在 32 位指令。 trap32 0, trap32 1 trap32 2, trap32 3 当处理器碰到 trap 指令时，发生陷阱异常操作。
影响标志位	无影响
异常	陷阱异常

32 位指令	
操作	引起陷阱异常发生
语法	trap32 0, trap32 1, trap32 2, trap32 3
说明	当处理器碰到 trap 指令时，发生陷阱异常操作。
影响标志位	无影响
异常	陷阱异常

32 位指令格式：	
trap32 0	
3130262521201615109540	
11000000000000000010000	
trap32 1	
3130262521201615109540	
11000000000000000010001	
trap32 2	
3130262521201615109540	
1100000000000000001010	
trap32 3	
3130262521201615109540	
1100000000000000001011	

14.213 TST——零测试指令

统一化指令	
语法	tst rx, ry
操作	If (RX & RY) != 0, then C ← 1; else C ← 0;
编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (x<16) and (y<16), then tst16 rx, ry; else tst32 rx, ry;
说明	测试 RX 和 RY 的值按位与的结果。 如果结果不等于 0，则设置条件位 C；否则，清除条件位 C。
影响标志位	根据按位与结果设置条件位 C
异常	无

16 位指令	
操作	If (RX & RY) != 0, then C ← 1; else C ← 0;
语法	tst16 rx, ry
说明	测试 RX 和 RY 的值按位与的结果。 如果结果不等于 0，则设置条件位 C；否则，清除条件位 C。
影响标志位	根据按位与结果设置条件位 C
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

15	14	10	9	6	5	2	1	0
0	11010			RY		RX		10

32 位指令	
操作	If (RX & RY) != 0, then C ← 1; else C ← 0;
语法	tst32 rx, ry
说明	测试 RX 和 RY 的值按位与的结果。 如果结果不等于 0，则设置条件位 C；否则，清除条件位 C。
影响标志位	根据按位与结果设置条件位 C
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001		RY		RX		001000		00100		00000	

14.214 STNBZ——无字节等于零寄存器测试指令

统一化指令	
语法	tstnbz16 rx
操作	If ((RX[31:24] != 0) &(RX[23:16] != 0) &(RX[15: 8] != 0) &(RX[7 : 0] != 0)), then C ← 1; else C ← 0;
编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (x<16), then tstnbz16 rx; else tstnbz32 rx;
说明	测试 RX 中是否没有字节等于零。如果 RX 没有字节等于零，则设置条件位 C；否则，清除条件位 C。
影响标志位	根据按位与结果设置条件位 C
异常	无

16 位指令	
操作	If ((RX[31:24] != 0) &(RX[23:16] != 0) &(RX[15: 8] != 0) &(RX[7 : 0] != 0)), then C ← 1; else C ← 0;
语法	tstnbz16 rx
说明	测试 RX 中是否没有字节等于零。如果 RX 没有字节等于零，则设置条件位 C；否则，清除条件位 C。
影响标志位	根据按位与结果设置条件位 C
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

15	14	10	9	6	5	2	1	0
0	11010	0000		RX			11	

32 位指令	
操作	If ((RX[31:24] != 0) &(RX[23:16] != 0) &(RX[15: 8] != 0) &(RX[7 : 0] != 0)), then C ← 1; else C ← 0;
语法	tstnbz32 rx
说明	测试 RX 中是否没有字节等于零。如果 RX 没有字节等于零，则设置条件位 C；否则，清除条件位 C。
影响标志位	根据按位与结果设置条件位 C
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001	00000		RX		001000		01000		00000		

14.215 WAIT——进入低功耗等待模式指令

统一化指令	
语法	wait
操作	进入低功耗等待模式
编译结果	仅存在 32 位指令。 wait32
属性：	特权指令
说明	此指令停止当前指令执行，并等待一个中断，此时 CPU 时钟停止。所有的外围设备都仍在继续运行，并有可能会产生中断而引起 CPU 从等待模式退出。
影响标志位	无影响
异常	特权违反指令

32 位指令	
操作	进入低功耗等待模式
语法	wait32
属性：	特权指令
说明	此指令停止当前指令执行，并等待一个中断，此时 CPU 时钟停止。所有的外围设备都仍在继续运行，并有可能会产生中断而引起 CPU 从等待模式退出。
影响标志位	无影响
异常	特权违反指令

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10000	00000	00000	010011	00001	00000						

14.216 XOR——按位异或指令

统一化指令		
语 法	xor rz, rx	xor rz, rx, ry
操 作	$RZ \leftarrow RZ \wedge RX$	$RZ \leftarrow RX \wedge RY$
编 译 结 果	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (x<16) and (z<16), then xor16 rz, rx; else xor32 rz, rz, rx;	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (y==z) and (z<16) and (x<16), then xor16 rz, rx; else xor32 rz, rx, ry;
说 明	将 RX 与 RZ/RY 的值按位异或，并把结果存在 RZ。	
影 响 标 志 位	无影响	
异 常	无	

16 位指令	
操作	$RZ \leftarrow RZ \wedge RX$
语法	xor16 rz, rx
说明	将 RZ 与 RX 的值按位异或，并把结果存在 RZ。
影响标志位	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

15	14	10	9	6	5	2	1	0
0	11011			RZ		RX		01

32 位指令	
操作	$RZ \leftarrow RX \wedge RY$
语法	xor32 rz, rx, ry
说明	将 RX 与 RY 的值按位异或，并把结果存在 RZ。
影响标志位	无影响
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001		RY		RX		001001		00010		RZ	

14.217 XORI——立即数按位异或指令

统一化指令	
语法	xori rz, rx, imm16
操作	$RZ \leftarrow RX \wedge \text{zero\_extend}(\text{IMM12})$
编译结果	仅存在 32 位指令。 xori32 rz, rx, imm12
说明	将 12 位立即数零扩展至 32 位，然后与 RX 的值进行按位异或操作，把结果存入 RZ。
影响标志位	无影响
限制	立即数的范围为 0x0-0xFFFF。
异常	无

32 位指令	
操作	$RZ \leftarrow RX \wedge \text{zero\_extend}(\text{IMM12})$
语法	xori32 rz, rx, imm12
说明	将 12 位立即数零扩展至 32 位，然后与 RX 的值进行按位异或操作，把结果存入 RZ。
影响标志位	无影响
限制	立即数的范围为 0x0-0xFFFF。
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	12	11	0
1	11001		RZ		RX		0100		IMM12	

14.218 XSR——扩展右移指令

统一化指令	
语 法	xsr rz, rx, oimm5
操 作	{RZ,C} ← {RX,C} >>>> OIMM5
编 译 结 果	仅存在 32 位指令。 xsr32 rz, rx, oimm5
说 明	将 RX 带条件位 C 的值（{RX,C}）进行循环右移（原值右移，左侧移入右侧移出的位），把移位结果的最低位（[0]）存入条件位 C，高位（[32:1]）存入 RZ，右移位数由带偏置 1 的 5 位立即数（OIMM5）的值决定。如果 OIMM5 的值等于 32，那么条件位 C 为 RX 的最高位。
影 响 标 志 位	$C \leftarrow RX[OIMM5 - 1]$
限 制	立即数的范围为 1-32。
异 常	无

32 位指令	
操作	$\{RZ,C\} \leftarrow \{RX,C\} >>>> OIMM5$
语法	xsr32 rz, rx, oimm5
说明	将 RX 带条件位 C 的值 ($\{RX,C\}$) 进行循环右移 (原值右移, 左侧移入右侧移出的位), 把移位结果的最低位 ([0]) 存入条件位 C, 高位 ([32:1]) 存入 RZ, 右移位数由带偏置 1 的 5 位立即数 (OIMM5) 的值决定。如果 OIMM5 的值等于 32, 那么条件位 C 为 RX 的最高位。 注意: 二进制操作数 IMM5 等于 OIMM5 - 1。
影响标志位	$C \leftarrow RX[OIMM5 - 1]$
限制	立即数的范围为 1-32。
异常	无

32 位指令格式:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001	IMM5		RX		010011		01000		RZ		

IMM5 域

指定不带偏置立即数的值。

注意: 移位的值 OIMM5 比起二进制操作数 IMM5 需偏置 1。

00000

移 1 位

00001

移 2 位

....

11111

移 32 位

14.219 XTRB0——提取字节 0 并无符号扩展指令

统一化指令	
语法	xtrb0 rz, rx
操作	$RZ \leftarrow \text{zero\_extend}(RX[31:24]);$ if ($RX[31:24] == 0$), then $C \leftarrow 0;$ else $C \leftarrow 1;$
编译结果	仅存在 32 位指令。 xtrb0.32 rz, rx
说明	提取 RX 的字节 0 ($RX[31:24]$) 到 RZ 的低位 ($RZ[7:0]$), 并进行零扩展。如果结果等于 0, 则清除 C 位, 反之设置 C 位。
影响标志位	如果结果等于 0, 则清除 C 位, 反之设置 C 位。
异常	无

32 位指令	
操作	$RZ \leftarrow \text{zero\_extend}(RX[31:24]);$ if ($RX[31:24] == 0$), then $C \leftarrow 0;$ else $C \leftarrow 1;$
语法	xtrb0.32 rz, rx
说明	提取 RX 的字节 0 ($RX[31:24]$) 到 RZ 的低位 ($RZ[7:0]$), 并进行零扩展。如果结果等于 0, 则清除 C 位, 反之设置 C 位。
影响标志位	如果结果等于 0, 则清除 C 位, 反之设置 C 位。
异常	无

32 位指令格式:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001	00000		RX		011100		00001			RZ	

14.220 XTRB1——提取字节 1 并无符号扩展指令

统一化指令	
语法	xtrb1 rz, rx
操作	$RZ \leftarrow \text{zero\_extend}(RX[23:16]);$ if ($RX[23:16] == 0$), then $C \leftarrow 0;$ else $C \leftarrow 1;$
编译结果	仅存在 32 位指令。 xtrb1.32 rz, rx
说明	提取 RX 的字节 1 ($RX[23:16]$) 到 RZ 的低位 ($RZ[7:0]$)，并进行零扩展。如果结果等于 0，则清除 C 位，反之设置 C 位。
影响标志位	如果结果等于 0，则清除 C 位，反之设置 C 位。
异常	无

32 位指令	
操作	$RZ \leftarrow \text{zero\_extend}(RX[23:16]);$ if ($RX[23:16] == 0$), then $C \leftarrow 0;$ else $C \leftarrow 1;$
语法	xtrb1.32 rz, rx
说明	提取 RX 的字节 1 ($RX[23:16]$) 到 RZ 的低位 ($RZ[7:0]$)，并进行零扩展。如果结果等于 0，则清除 C 位，反之设置 C 位。
影响标志位	如果结果等于 0，则清除 C 位，反之设置 C 位。
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001	00000		RX		011100		00010			RZ	

14.221 XTRB2——提取字节 2 并无符号扩展指令

统一化指令	
语法	xtrb2 rz, rx
操作	$RZ \leftarrow \text{zero\_extend}(RX[15:8]);$ if ($RX[15:8] == 0$), then $C \leftarrow 0;$ else $C \leftarrow 1;$
编译结果	仅存在 32 位指令。 xtrb2.32 rz, rx
说明	提取 RX 的字节 2 ($RX[15:8]$) 到 RZ 的低位 ($RZ[7:0]$), 并进行零扩展。如果结果等于 0, 则清除 C 位, 反之设置 C 位。
影响标志位	如果结果等于 0, 则清除 C 位, 反之设置 C 位。
异常	无

32 位指令	
操作	$RZ \leftarrow \text{zero\_extend}(RX[15:8]);$ if ($RX[15:8] == 0$), then $C \leftarrow 0;$ else $C \leftarrow 1;$
语法	xtrb2.32 rz, rx
说明	提取 RX 的字节 2 ($RX[15:8]$) 到 RZ 的低位 ($RZ[7:0]$), 并进行零扩展。如果结果等于 0, 则清除 C 位, 反之设置 C 位。
影响标志位	如果结果等于 0, 则清除 C 位, 反之设置 C 位。
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001	00000		RX		011100		00100			RZ	

14.222 XTRB3——提取字节 3 并无符号扩展指令

统一化指令	
语法	xtrb3 rz, rx
操作	$RZ \leftarrow \text{zero\_extend}(RX[7:0]);$ if ($RX[7:0] == 0$), then $C \leftarrow 0;$ else $C \leftarrow 1;$
编译结果	仅存在 32 位指令。 xtrb3.32 rz, rx
说明	提取 RX 的字节 3 ($RX[7:0]$) 到 RZ 的低位 ($RZ[7:0]$), 并进行零扩展。如果结果等于 0, 则清除 C 位, 反之设置 C 位。
影响标志位	如果结果等于 0, 则清除 C 位, 反之设置 C 位。
异常	无

32 位指令	
操作	$RZ \leftarrow \text{zero\_extend}(RX[7:0]);$ if ($RX[7:0] == 0$), then $C \leftarrow 0;$ else $C \leftarrow 1;$
语法	xtrb3.32 rz, rx
说明	提取 RX 的字节 3 ($RX[7:0]$) 到 RZ 的低位 ($RZ[7:0]$), 并进行零扩展。如果结果等于 0, 则清除 C 位, 反之设置 C 位。
影响标志位	如果结果等于 0, 则清除 C 位, 反之设置 C 位。
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001	00000		RX		011100		01000			RZ	

14.223 ZEXT——位提取并无符号扩展指令

统一化指令	
语 法	zext rz, rx, msb,
操 作 lsb	$RZ \leftarrow \text{zero\_extend}(RX[MSB:LSB])$
编 译 结 果	仅存在 32 位指令。 zext32 rz, rx, msb, lsb
说 明	提取由 2 个 5 位立即数 (MSB,LSB) 所指定的 RX 的一段连续位 ($RX[MSB:LSB]$), 零扩展至 32 位, 并把结果存入 RZ。如果 MSB 等于 31, 且 LSB 等于 0, 则 RZ 的值与 RX 相同。如果 MSB 等于 LSB, 则 RZ 的值为 $RX[MSB]$ (即 $RX[LSB]$) 一位零扩展的结果。如果 MSB 小于 LSB, 该指令的行为不可预测。
影 响 标 志 位	无影响
限 制	MSB 的范围为 0-31, LSB 的范围为 0-31, 且 MSB 应当大于等于 LSB。
异 常	无

32 位指令	
操作	$RZ \leftarrow \text{zero\_extend}(RX[MSB:LSB])$
语法	<code>zext32 rz, rx, msb, lsb</code>
说明	提取由 2 个 5 位立即数 (MSB,LSB) 所指定的 RX 的一段连续位 (RX[MSB:LSB]), 零扩展至 32 位, 并把结果存入 RZ。如果 MSB 等于 31, 且 LSB 等于 0, 则 RZ 的值与 RX 相同。如果 MSB 等于 LSB, 则 RZ 的值为 RX[MSB] (即 RX[LSB]) 一位零扩展的结果。如果 MSB 小于 LSB, 该指令的行为不可预测。
影响标志位	无影响
限制	MSB 的范围为 0-31, LSB 的范围为 0-31, 且 MSB 应当大于等于 LSB。
异常	无

32 位指令格式:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001		LSB		RX		010101		MSB		RZ	

MSB 域

指定被提取开始的位。

LSB 域

指定被提取结束的位。

00000

0

00000

0 位

00001

1

00001

1 位

.....

11111

31

11111

31 位

14.224 ZEXTB——字节提取并无符号扩展指令

统一化指令	
语法	zextb rz, rx
操作	$RZ \leftarrow \text{zero\_extend}(RX[7:0]);$
编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (x<16) and (z<16), then zextb16 rz, rx; else zextb32 rz, rx
说明	将 RX 的低字节 (RX[7:0]) 零扩展至 32 位，结果存在 RZ 中。
影响标志位	无影响
异常	无

16 位指令	
操作	$RZ \leftarrow \text{zero\_extend}(RX[7:0]);$
语法	zextb16 rz, rx
说明	将 RX 的低字节 (RX[7:0]) 零扩展至 32 位，结果存在 RZ 中。
影响标志位	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

15	14	10	9	6	5	2	1	0
0	11101		RZ		RX		00	

32 位指令	
操作	$RZ \leftarrow \text{zero\_extend}(RX[7:0]);$
语法	<code>zextb32 rz, rx</code>
说明	将 RX 的低字节（RX[7:0]）零扩展至 32 位，结果存在 RZ 中。 注意，该指令是 <code>zext32 rz, rx, 0x7, 0x0</code> 的伪指令。
影响标志位	无影响
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001	00000			RX	010101		00111				RZ

14.225 ZEXTH——半字提取并无符号扩展指令

统一化指令	
语法	<code>zexth rz, rx</code>
操作	$RZ \leftarrow \text{zero\_extend}(RX[15:0]);$
编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (x<16) and (z<16), then <code>zexth16 rz, rx;</code> else <code>zexth32 rz, rx</code>
说明	将 RX 的低半字（RX[15:0]）零扩展至 32 位，结果存在 RZ 中。
影响标志位	无影响
异常	无

16 位指令	
操作	$RZ \leftarrow \text{zero\_extend}(RX[15:0]);$
语法	<code>zexth16 rz, rx</code>
说明	将 RX 的低半字（RX[15:0]）零扩展至 32 位，结果存在 RZ 中。
影响标志位	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

15	14	10	9	6	5	2	1	0
0	11101	RZ			RX		01	

32 位指令	
操作	$RZ \leftarrow \text{zero\_extend}(RX[15:0]);$
语法	<code>zexth32 rz, rx</code>
说明	将 RX 的低半字（RX[15:0]）零扩展至 32 位，结果存在 RZ 中。 注意，该指令是 <code>zext32 rz, rx, 0x15, 0x0</code> 的伪指令。
影响标志位	无影响
异常	无

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	10001	00000			RX		010101		01111		RZ	

第十五章 附录 B 矢量浮点指令术语表

以下是每条 C860 实现的玄铁 V2 浮点矢量指令的具体描述，下面根据指令英文字母顺序对每条指令进行详细说明。

15.1 VABS.T——矢量浮点绝对值指令

统一化指令	
语法	vabs.T vrz, vrx (T=f32)
操作	vrz[31:0] = vrx[31:0] ; vrz[63:32] = vrx[63:32] ; vrz[95:64] = vrx[95:64] ; vrz[127:96] = vrx[127:96] ;
编译结果	仅存在 32 位指令。 vabs.T vrz, vrx
数据类型	单精度
说明	将 vrx[31:0] 中的浮点数取绝对值，保存至 vrz[31:0] 中。 将 vrx[63:32] 中的浮点数取绝对值，保存至 vrz[63:32] 中。 将 vrx[95:64] 中的浮点数取绝对值，保存至 vrz[95:64] 中。 将 vrx[127:96] 中的浮点数取绝对值，保存至 vrz[95:64] 中。
影响标志位	无
限制	寄存器范围为 vr0-vr15 中。
异常	无

32 位指令	
操作	$vrz[31:0] = vrx[31:0] ;$ $vrz[63:32] = vrx[63:32] ;$ $vrz[95:64] = vrx[95:64] ;$ $vrz[127:96] = vrx[127:96] ;$
语法	Vabs.T vrz, vrx
数据类型	单精度
说明	将 vrx[31:0] 中的浮点数取绝对值，保存至 vrz[31:0] 中。 将 vrx[63:32] 中的浮点数取绝对值，保存至 vrz[63:32] 中。 将 vrx[95:64] 中的浮点数取绝对值，保存至 vrz[95:64] 中。 将 vrx[127:96] 中的浮点数取绝对值，保存至 vrz[95:64] 中。
影响标志位	无
限制	寄存器范围为 vr0-vr15 中。
异常	无

32 位指令格式：

vabs.f32

31	26	25	24	21	20	16	15	5	4	0
111101	0		0000		VRX			10101010001		VRZ

15.2 VADD.T——矢量浮点加法指令

统一化指令	
语法	vadd.T vrz, vrx,vry (T=f32)
操作	$vrz[31:0] = vrx[31:0] + vry[31:0];$ $vrz[63:32] = vrx[63:32] + vry[63:32];$ $vrz[95:64] = vrx[95:64] + vry[95:64];$ $vrz[127:96] = vrx[127:96] + vry[127:96];$
编译结果	仅存在 32 位指令。 vadd.T vrz, vrx,vry
数据类型	单精度
说明	将 vrx[31:0] 和 vry[31:0] 中的浮点数相加，保存至 vrz[31:0] 中。 将 vrx[63:32] 和 v ry[63:32] 中的浮点数相加，保存至 vrz[63:32] 中。 将 vrx[95:64] 和 v ry[95:64] 中的浮点数相加，保存至 vrz[95:64] 中。 将 vrx[127:96] 和 vry [127:96] 中的浮点数相加，保存至 vrz[127:96] 中。
影响标志位	无
限制	寄存器范围为 vr0-vr15 中。
异常	ID,IO,IX,UF,OF

32 位指令	
操作	$vrz[31:0] = vrz[31:0] + vry[31:0];$ $vrz[63:32] = vrz[63:32] + vry[63:32];$ $vrz[95:64] = vrz[95:64] + vry[95:64];$ $vrz[127:96] = vrz[127:96] + vry[127:96];$
语法	vadd.T vrz, vrz,vry
数据类型	单精度
说明	将 vrz[31:0] 和 vry[31:0] 中的浮点数相加，保存至 vrz[31:0] 中。 将 vrz[63:32] 和 vry[63:32] 中的浮点数相加，保存至 vrz[63:32] 中。 将 vrz[95:64] 和 vry[95:64] 中的浮点数相加，保存至 vrz[95:64] 中。 将 vrz[127:96] 和 vry[127:96] 中的浮点数相加，保存至 vrz[127:96] 中
影响标志位	无
限制	寄存器范围为 vr0-vr15 中。
异常	ID,IO,IX,UF,OF

32 位指令格式：

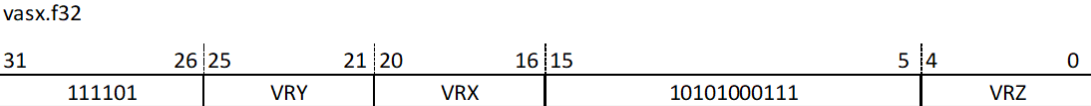
vadd.f32									
31	26	25	21	20	16	15	5	4	0
111101		VRY		VRX		10101000001		VRZ	

15.3 VASX.T——矢量浮点交叉加减法指令

统一化指令	
语法	vasx.T vrz, vrz,vry (T=f32)
操作	$vrz[31:0] = vrz[31:0] - vry[63:32];$ $vrz[63:32] = vrz[63:32] + vry[31:0];$ $vrz[95:64] = vrz[95:64] - vry[127:96];$ $vrz[127:96] = vrz[127:96] + vry[95:64];$
编译结果	仅存在 32 位指令。 vasx.T vrz, vrz,vry
数据类型	单精度
说明	将 vrz[31:0] 和 vry[63:32] 中的浮点数相减，保存至 vrz[31:0] 中。 将 vrz[63:32] 和 vry[31:0] 中的浮点数相加，保存至 vrz[63:32] 中。 将 vrz[95:64] 和 vry[127:96] 中的浮点数相减，保存至 vrz[95:64] 中。 将 vrz[127:96] 和 vry[95:64] 中的浮点数相加，保存至 vrz[127:96] 中。
影响标志位	无
限制	寄存器范围为 vr0-vr15 中。
异常	ID,IO,IX,UF,OF

32 位指令	
操作	vrz[31:0]= vrx[31:0]-vry[63:32]; vrz[63:32]= vrx[63:32]+vry[31:0]; vrz[95:64]= vrx[95:64]-vry[127:96]; vrz[127:96]= vrx[127:96]+vry[95:64];
语法	vasx.T vrz, vrx,vry
数据类型	单精度
说明	将 vrx[31 :0] 和 vry[63:32] 中的浮点数相减，保存至 vrz[31:0] 中。 将 vrx[63: 32] 和 vry[31:0] 中的浮点数相加，保存至 vrz[63:32] 中。 将 vrx[95:64] 和 vry[127:96] 中的浮点数相减，保存至 vrz[95:64] 中。 将 vrx[127:96] 和 vry[95:64] 中的浮点数相加，保存至 vrz[127:96] 中。
影响标志位	无
限制	寄存器范围为 vr0-vr15 中。
异常	ID,IO,IX,UF,OF

32 位指令格式：



15.4 VCMPHS.T——矢量浮点大于等于比较指令

统一化指令	
语法	<code>vcmp_{hs}.T vrz, vr_x, vr_y</code> (T=F32)
操作	<pre>w={32}; for(i=0; i< 128/w; i++) { if(vr_x[i]>=vr_y[i]) { vrz[i] = AllOnes; } Else { vrz[i] = AllZeros; } }</pre>
编译结果	仅存在 32 位指令。 <code>vcmp_{hs}.T vrz, vr_x, vr_y</code>
数据类型	单精度
说明	比较 VRX 和 VRY 对应元素，如果 VRX 大于等于 VRY，则将 VRZ 的对应元素置全 1；否则，置全 0。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr31。
异常	IO、ID

32 位指令	
操作	W={32}; for(i=0; i< 128/w; i++) { if(vrx[i]>=vry[i]) { vrz[i] = AllOnes; } Else { vrz[i] = AllZeros; } }
语法	vcmphs.T vrz, vrx, vry
数据类型	单精度
说明	比较 VRX 和 VRY 对应元素，如果 VRX 大于等于 VRY，则将 VRZ 的对应元素置全 1； 否则，置全 0。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr31。
异常	IO、ID

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11101		VRY		VRX		101100		00011		VRZ	

15.5 VCMPHSZ.T——矢量浮点大于等于零比较指令

统一化指令	
语法	vcmphsz.T vrz,vrx (T=F32)
操作	w={32}; for(i=0; i< 128/w; i++) { if(vrx[i]>=0) { vrz[i] = AllOnes; } Else { vrz[i] = AllZeros; } }
编译结果	仅存在 32 位指令。 vcmphsz.T vrz, vrx
数据类型	单精度
说明	比较 VRX 元素和 0，如果 VRX 的元素大于等于 0，则将 VRZ 的对应元素置全 1；否则，置全 0。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr31。
异常	IO、ID

32 位指令	
操作	<pre>w={32}; for(i=0; i< 128/w; i++) { if(vrx[i]>=0]) { vrz[i] = AllOnes; } Else { vrz[i] = AllZeros; } }</pre>
语法	vcmphsz.T vrz, vrx
数据类型	单精度
说明	比较 VRX 元素和 0，如果 VRX 的元素大于等于 0，则将 VRZ 的对应元素置全 1；否则，置全 0。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr31。
异常	IO、ID

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11101	00000		VRX		101101		00011		VRZ		

15.6 VCMPHZ.T——矢量浮点大于零比较指令

统一化指令	
语法	<code>vcmphz.T vrz,vrx</code> (T=F32)
操作	<pre>w={32}; for(i=0; i< 128/w; i++) { if(vrx[i]>0) { vrz[i] = AllOnes; } Else { vrz[i] = AllZeros; } }</pre>
编译结果	仅存在 32 位指令。 <code>vcmphz.T vrz, vrx</code>
数据类型	单精度
说明	比较 VRX 元素和 0，如果 VRX 的元素大于 0，则将 VRZ 的对应元素置全 1；否则，置全 0。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr31。
异常	IO、ID

32 位指令	
操作	w={32} for(i=0; i< 128/w; i++) { if(vrx[i]>0]) { vrz[i] = AllOnes; } Else { vrz[i] = AllZeros; } }
语法	vcmphz.T vrz, vrx
数据类型	单精度
说明	比较 VRX 元素和 0，如果 VRX 的元素大于 0，则将 VRZ 的对应元素置全 1；否则，置全 0。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr31。
异常	IO、ID

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11101	00000		VRX		101101		00111		VRZ		

15.7 VCMPLSZ.T——矢量浮点小于等于零比较指令

统一化指令	
语法	vcmplsz.T vrz,vrx (T=F32)
操作	w={32} for(i=0; i< 128/w; i++) { if(vrx[i]<=0) { vrz[i] = AllOnes; } Else { vrz[i] = AllZeros; } }
编译结果	仅存在 32 位指令。 vcmplsz.T vrz, vrz
数据类型	单精度
说明	比较 VRX 元素和 0，如果 VRX 的元素小于等于 0，则将 VRZ 的对应元素置全 1；否则，置全 0。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr31。
异常	IO、ID

32 位指令	
操作	<pre>w={32}; for(i=0; i< 128/i; i++) { if(vrx[i]<=0]) { vrz[i] = AllOnes; } Else { vrz[i] = AllZeros; } }</pre>
语法	vcmplsz.T vrz, vrx
数据类型	单精度
说明	比较 VRX 元素和 0，如果 VRX 的元素小于等于 0，则将 VRZ 的对应元素置全 1；否则，置全 0。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr31。
异常	IO、ID

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11101	00000		VRX		101101		01001		VRZ		

15.8 VCMPLT.T——矢量浮点小于比较指令

统一化指令	
语法	vcmplt.T vrz, vrx, vry (T=F32)
操作	w={32}; for(i=0; i<128/w; i++) { if(vrx[i]<vry[i]) { vrz[i] = AllOnes; } Else { vrz[i] = AllZeros; } }
编译结果	仅存在 32 位指令。 vcmplt.T vrz, vrx, vry
数据类型	单精度
说明	比较 VRX 和 VRY 对应元素，如果 VRX 小于 VRY，则将 VRZ 的对应元素置全 1；否则，置全 0。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr31。
异常	IO、ID

32 位指令	
操作	<pre>w={32}; for(i=0; i< 128/i; i++) { if(vrx[i]<vry[i]) { vrz[i] = AllOnes; } Else { vrz[i] = AllZeros; } }</pre>
语法	vcmplt.T vrz, vrx, vry
数据类型	单精度
说明	比较 VRX 和 VRY 对应元素，如果 VRX 小于 VRY，则将 VRZ 的对应元素置全 1；否则，置全 0。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr31。
异常	IO、ID

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11101		VRY		VRX		101100		00101		VRZ	

15.9 VCMPLTZ.T——矢量浮点小于零比较指令

统一化指令	
语法	vcmpltz.T vrz,vrx (T=F32)
操作	w={32}; for(i=0; i< 128/w; i++) { if(vrx[i]<0) { vrz[i] = AllOnes; } Else { vrz[i] = AllZeros; } }
编译结果	仅存在 32 位指令。 vcmpltz.T vrz, vrz
数据类型	单精度
说明	比较 VRX 元素和 0，如果 VRX 的元素小于 0，则将 VRZ 的对应元素置全 1；否则，置全 0。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr31。
异常	IO、ID

32 位指令	
操作	<pre>w={32}; for(i=0; i< 128/w; i++) { if(vrx[i]<0]) { vrz[i] = AllOnes; } Else { vrz[i] = AllZeros; } }</pre>
语法	vcmpltz.T vrz, vrx
数据类型	单精度
说明	比较 VRX 元素和 0，如果 VRX 的元素小于 0，则将 VRZ 的对应元素置全 1；否则，置全 0。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr31。
异常	IO、ID

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11101	00000		VRX		101101		00101		VRZ		

15.10 VCOMPNE.T——矢量浮点不等于比较指令

统一化指令	
语法	vcmpne.T vrz, vrx, vry (T=F32)
操作	w={32}; for(i=0; i< 128/w; i++) { if(vrx[i]!=vry[i]) { vrz[i] = AllOnes; } Else { vrz[i] = AllZeros; } }
编译结果	仅存在 32 位指令。 vcmpne.T vrz, vrx, vry
数据类型	单精度
说明	比较 VRX 和 VRY 对应元素，如果 V RX 不等于 VRY，则将 VRZ 的对应元素置全 1； 否则，置全 0。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr31。
异常	IO、ID

32 位指令	
操作	<pre>w={32}; for(i=0; i< 128/w; i++) { if(vrx[i]!=vry[i]) { vrz[i] = AllOnes; } Else { vrz[i] = AllZeros; } }</pre>
语法	vcmpne.T vrz, vr _x , vr _y
数据类型	单精度
说明	比较 VR _X 和 VR _Y 对应元素，如果 V R _X 不等于 VR _Y ，则将 VR _Z 的对应元素置全 1；否则，置全 0。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr31。
异常	IO、ID

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11101		VR _Y		VR _X		101100		00001		VR _Z	

15.11 VCOMPNEZ.T——矢量浮点不等于零比较指令

统一化指令	
语法	vcmpnez.T vrz,vrx (T=F32)
操作	w={32}; for(i=0; i< 128/w; i++) { if(vrx[i]!=0) { vrz[i] = AllOnes; } Else { vrz[i] = AllZeros; } }
编译结果	仅存在 32 位指令。 vcmpnez.T vrz, vrz
数据类型	单精度
说明	比较 VRX 元素和 0，如果 VRX 的元素不等于 0，则将 VRZ 的对应元素置全 1；否则，置全 0。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr31。
异常	IO、ID

32 位指令	
操作	<pre>w={32}; for(i=0; i< 128/w; i++) { if(vrx[i]!=0) { vrz[i] = AllOnes; } Else { vrz[i] = AllZeros; } }</pre>
语法	vcmpnez.T vrz, vrx
数据类型	单精度
说明	比较 VRX 元素和 0，如果 VRX 的元素小于 0，则将 VRZ 的对应元素置全 1；否则，置全 0。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr31。
异常	IO、ID

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11101	00000		VRX		101101		00001		VRZ		

15.12 VDTOS——矢量双精度浮点单精度浮点转换指令

统一化指令	
语法	vdtos vrz, vrx
操作	vrz[31:0]=(float)vrx[63:0]; vrz[63:32]=(float)vrx[127:64];
编译结果	仅存在 32 位指令。 vdtos vrz, vrx
数据类型	双精度浮点数
说明	将 vrx[63:0] 中的双精度浮点数转换成单精度浮点数，保存至 vrz[31:0] 中。 将 vrx[127:64] 中的双精度浮点数转换成单精度浮点数，保存至 vrz[63:32] 中。vrz[127:64] 为 0。
影响标志位	无
限制	寄存器范围为 vr0-vr15。vrz[127:64] 为 0。
异常	ID,IO,UF,OF,IX

32 位指令	
操作	vrz[31:0]=(float)vrx[63:0]; vrz[63:32]=(float)vrx[127:64];
语法	vdtos vrz, vrx
数据类型	双精度浮点数
说明	将 vrx[63:0] 中的双精度浮点数转换成单精度浮点数，保存至 vrz[31:0] 中。 将 vrx[127:64] 中的双精度浮点数转换成单精度浮点数，保存至 vrz[63:32] 中。vrz[127:64] 为 0。
影响标志位	无
限制	寄存器范围为 vr0-vr15。vrz[127:64] 为 0。
异常	ID,IO,UF,OF,IX

32 位指令格式：

31	26	25	24	21	20	16	15	5	4	0
111101			0	0000		VRX		11010010000		VRZ

15.13 VEXPE.T——矢量浮点倒数逼近指令

统一化指令	
语法	vexpe.t vrz, vrx (t=32)
操作	vrz(i) = float_to_int(1064807268.539173 + 12102203.161561 * vrx(i)) i=0 : 128/32 - 1 其中，Vx(i) 的范围为 [-85, 85]，若 Vx(i)>85, 则将 Vx(i) 置为 85，若 Vx(i)<-85, 则将 Vx(i) 置为-85。 1064807268.539173 = 0x4E7DDEAD 12102203.161561 = 0x4B38AA3B
编译结果	仅存在 32 位指令 vexpe.t vrz, vrx
数据类型:	单精度
说明	将 vrx 中的每个元素都与 12102203.161561 相乘到的乘积结果保留全部精度，再加上 1064807268.539173，最终的结果放入 vrz 对应的元素中。
影响标志位	无影响
异常	IO、ID、IX

32 位指令	
操作	vrz(i) = float_to_int(1064807268.539173 + 12102203.161561 * vrx(i)) i=0 : 128/32 - 1 其中，Vx(i) 的范围为 [-85, 85]，若 Vx(i)> 85, 则将 Vx(i) 置为 85，若 Vx(i)<-85, 则将 Vx(i) 置为-85。 1064807268.539173 = 0x4E7DDEAD 12102203.161561 = 0x4B38AA3B
数据类型:	单精度
语法	vexpe.t vrz, vrx
说明	将 vrx 中的每个元素都与 12102203.161561 相乘，得到的乘积结果保留全部精度，再加上 1064807268.539173，最终的结果放入 vrz 对应的元素中
影响标志位	无影响
异常	IO、ID、IX

32 位指令格式：

vexpe.f32

31	30	26	25	21	20	16	15	10	9	5	4	0
1		11101		X		VRX		101110		01001		VRZ

15.14 VFCMUL.T——矢量浮点复数乘法指令

统一化指令	
语法	vfcmul.t vrz, vrx, vry (t=f32)
操作	$tmp(2i+1) = vrz(2i) * vry(2i+1)$ $tmp(2i) = vrz(2i) * vry(2i)$ $vrz(2i+1) = tmp(2i+1) + vrz(2i+1) * vry(2i)$ $vrz(2i) = tmp(2i) - vrz(2i+1) * vry(2i+1)$ $i = 0 : 128/32/2 - 1$ 注：tmp(i) 乘法结果做一次舍入饱和操作，vrz 中乘累加结果再做一次舍入饱和操作。
编译结果	仅存在 32 位指令 vfcmul.t vrz, vrx, vry
数据类型:	单精度
说明	复数乘法计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；该指令会拆分成两部分，先计算最终结果的实数和虚数的部分积，再通过乘累加得到两个复数相乘后的结果。该指令会完成两次舍入操作，乘法和乘累加计算完毕后各执行一次舍入操作
影响标志位	无影响
异常	IO、ID、IX、UF、OF

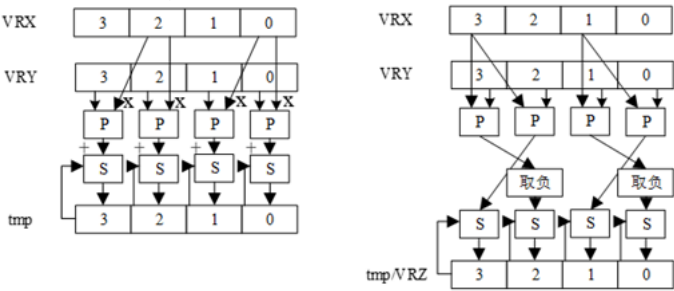
32 位指令	
操作	tmp(2i+1) = vr_x(2i) * vr_y(2i+1) tmp(2i) = vr_x(2i) * vr_y(2i) vr_z(2i+1) = tmp(2i+1) + vr_x(2i+1) * vr_y(2i) vr_z(2i) = tmp(2i) - vr_x(2i+1) * vr_y(2i+1) i = 0 : 128/32/2 -1 注：tmp(i) 乘法结果做一次舍入饱和操作，vr_z 中乘累加结果再做一次舍入饱和操作
数据类型:	单精度
语法	vfcmul.t vr _z , vr _x , vr _y
说明	复数乘法计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；该指令会拆分成两部分，先计算最终结果的实数和虚数的部分积，再通过乘累加得到两个复数相乘后的结果。该指令会完成两次舍入操作，乘法和乘累加计算完毕后各执行一次舍入操作
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令格式：

vfcmul.f32

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11101	VRY				VRX				101001	00101	VRZ

15.15 VFCMULA.T——矢量浮点复数乘累加指令

统一化指令	
语法	vfcmula.t vrz, vrx, vry (t=f32)
操作	$tmp(2i+1) = vrz(2i+1) + vrx(2i) * vry(2i+1)$ $tmp(2i) = vrz(2i) + vrx(2i) * vry(2i)$ $vrz(2i+1) = tmp(2i+1) + vrx(2i+1) * vry(2i)$ $vrz(2i) = tmp(2i) - vrx(2i+1) * vry(2i+1)$ $i = 0 : 128/32/2 - 1$ 注：tmp(i) 乘法结果做一次舍入饱和操作，vrz 中乘累加结果再做一次舍入饱和操作
编译结果	仅存在 32 位指令 vfcmula.t vrz, vrx, vry
数据类型:	单精度
说明	复数乘累加计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；该指令会拆分成两部分，先通过乘累加计算最终结果的实数和虚数的部分结果，再通过一次乘累加得到两个复数相乘再加上一个复数的结果。该指令会完成两次舍入操作，乘法和乘累加计算完毕后各执行一次舍入操作 
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令	
操作	tmp(2i+1) = vrz(2i+1) + vrz(2i) * vry(2i+1) tmp(2i) = vrz(2i) + vrz(2i) * vry(2i) vrz(2i+1) = tmp(2i+1) + vrz(2i+1) * vry(2i) vrz(2i) = tmp(2i) - vrz(2i+1) * vry(2i+1) 注：tmp(i) 乘法结果做一次舍入饱和操作，vrz 中乘累加结果再做一次舍入饱和操作。
数据类型:	单精度
语法	vfcmla.t vrz, vrz, vry
说明	复数乘累加计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；该指令会拆分成两部分，先通过乘累加计算最终结果的实数和虚数的部分结果，再通过一次乘累加得到两个复数相乘再加上一个复数的结果。该指令会完成两次舍入操作，乘法和乘累加计算完毕后各执行一次舍入操作。
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令格式:

vfcmla.f32

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11101	VRY	VRX	101001	01101	VRZ						

15.16 VFCMULC.T——矢量浮点复数共轭乘法指令

统一化指令																									
语法	<code>vfcmulc.t vrz, vrz, vry</code> (t=f32)																								
操作	$tmp(2i+1) = vrz(2i) * vry(2i+1)$ $tmp(2i) = vrz(2i) * vry(2i)$ $vrz(2i+1) = tmp(2i+1) - vrz(2i+1) * vry(2i)$ $vrz(2i) = tmp(2i) + vrz(2i+1) * vry(2i+1)$ $i = 0 : 128/32/2 - 1$ 注：tmp(i) 乘法结果做一次舍入饱和操作，vrz 中乘累加结果再做一次舍入饱和操作																								
编译结果	仅存在 32 位指令 <code>vfcmulc.t vrz, vrz, vry</code>																								
数据类型:	单精度																								
说明	复数共轭乘法计算 $conj(x) * y$，一个矢量操作数包含两个复数，实部在低位，虚部在高位；该指令会拆分成两部分，先计算最终结果的实数和虚数的部分积，再通过乘累加得到两个复数相乘后的结果。该指令会完成两次舍入操作，乘法和乘累加计算完毕后各执行一次舍入操作。 VRX <table><tr><td>3</td><td>2</td><td>1</td><td>0</td></tr></table> VRZ <table><tr><td>3</td><td>2</td><td>1</td><td>0</td></tr></table> tmp <table><tr><td>3</td><td>2</td><td>1</td><td>0</td></tr></table> VRX <table><tr><td>3</td><td>2</td><td>1</td><td>0</td></tr></table> VRZ <table><tr><td>3</td><td>2</td><td>1</td><td>0</td></tr></table> tmp^VRZ <table><tr><td>3</td><td>2</td><td>1</td><td>0</td></tr></table>	3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0
3	2	1	0																						
3	2	1	0																						
3	2	1	0																						
3	2	1	0																						
3	2	1	0																						
3	2	1	0																						
影响标志位	无影响																								
异常	IO、ID、IX、UF、OF																								

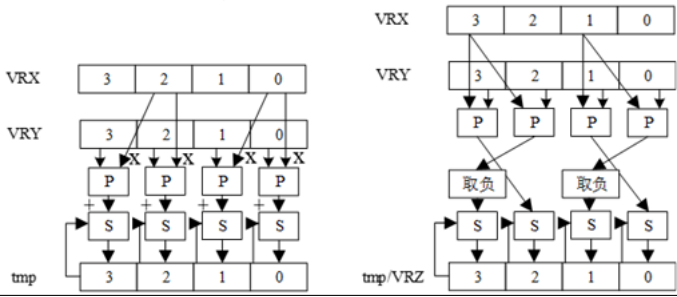
32 位指令	
操作	tmp(2i+1) = vr_x(2i) * vr_y(2i+1) tmp(2i) = vr_x(2i) * vr_y(2i) vr_z(2i+1) = tmp(2i+1) - vr_x(2i+1) * vr_y(2i) vr_z(2i) = tmp(2i) + vr_x(2i+1) * vr_y(2i+1) i = 0 : 128/32/2 -1 注：tmp(i) 乘法结果做一次舍入饱和操作，vr_z 中乘累加结果再做一次舍入饱和操作。
数据类型:	单精度
语法	vfcmlc.t vr _z , vr _x , vr _y
说明	复数共轭乘法计算 conj(x) * y，一个矢量操作数包含两个复数，实部在低位，虚部在高位；该指令会拆分成两部分，先计算最终结果的实数和虚数的部分积，再通过乘累加得到两个复数相乘后的结果。该指令会完成两次舍入操作，乘法和乘累加计算完毕后各执行一次舍入操作。
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令格式：

vfcmlc.f32

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11101	VRY	VRX	101001	00111	VRZ						

15.17 VFCMULCA.T——矢量浮点复数共轭乘累加指令

统一化指令	
语法	vfcmulca.t vrz, vrz, vrz, vry (t=f32)
操作	$tmp(2i+1) = vrz(2i+1) + vrz(2i) * vry(2i+1)$ $tmp(2i) = vrz(2i) + vrz(2i) * vry(2i)$ $vrz(2i+1) = tmp(2i+1) - vrz(2i+1) * vry(2i)$ $vrz(2i) = tmp(2i) + vrz(2i+1) * vry(2i+1)$ $i = 0 : 128/32/2 - 1$ 注：tmp(i) 乘法结果做一次舍入饱和操作，vrz 中乘累加结果再做一次舍入饱和操作
编译结果	仅存在 32 位指令 vfcmulca.t vrz, vrz, vrz, vry
数据类型:	单精度
说明	复数共轭乘累加计算 $z + conj(x) * y$，一个矢量操作数包含两个复数，实部在低位，虚部在高位；该指令会拆分成两部分，先通过乘累加计算最终结果的实数和虚数的部分结果，再通过一次乘累加得到两个复数相乘再加上一个复数的结果。该指令会完成两次舍入操作，乘法和乘累加计算完毕后各执行一次舍入操作。 
影响标志位	无影响
异常	IO、ID、IX、UF、OF

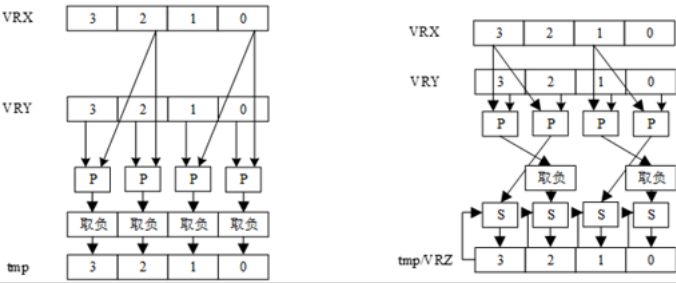
32 位指令	
操作	$\text{tmp}(2i+1) = \text{vrz}(2i+1) + \text{vrx}(2i) * \text{vry}(2i+1)$ $\text{tmp}(2i) = \text{vrz}(2i) + \text{vrx}(2i) * \text{vry}(2i)$ $\text{vrz}(2i+1) = \text{tmp}(2i+1) - \text{vrx}(2i+1) * \text{vry}(2i)$ $\text{vrz}(2i) = \text{tmp}(2i) + \text{vrx}(2i+1) * \text{vry}(2i+1)$ $i = 0 : 128/32/2 - 1$ 注：tmp(i) 乘法结果做一次舍入饱和操作，vrz 中乘累加结果再做一次舍入饱和操作。
数据类型:	单精度
语法	vfcmulca.t vrz, vrx, vry
说明	复数共轭乘累加计算 $z + \text{conj}(x) * y$，一个矢量操作数包含两个复数，实部在低位，虚部在高位；该指令会拆分成两部分，先通过乘累加计算最终结果的实数和虚数的部分结果，再通过一次乘累加得到两个复数相乘再加上一个复数的结果。该指令会完成两次舍入操作，乘法和乘累加计算完毕后各执行一次舍入操作。 Diagram illustrating the first stage of the instruction, showing the calculation of intermediate results for the real and imaginary parts of the product. Diagram illustrating the second stage of the instruction, showing the calculation of intermediate results for the real and imaginary parts of the product.
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令格式:

vfcmulca.f32

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11101				VRX		101001		01111		VRZ	

15.18 VFCMULCN.T——矢量浮点复数共轭取负乘法指令

统一化指令	
语法	vfcmulcn.t vrz, vrz, vry (t=f32)
操作	$tmp(2i+1) = -vrz(2i) * vry(2i+1)$ $tmp(2i) = -vrz(2i) * vry(2i)$ $vrz(2i+1) = tmp(2i+1) + vrz(2i+1) * vry(2i)$ $vrz(2i) = tmp(2i) - vrz(2i+1) * vry(2i+1)$ $i = 0 : 128/32/2 - 1$ 注：tmp(i) 乘法结果做一次舍入饱和操作，vrz 中乘累加结果再做一次舍入饱和操作。
编译结果	仅存在 32 位指令 vfcmulcn.t vrz, vrz, vry
数据类型:	单精度
说明	复数共轭乘法计算-$conj(x) * y$，一个矢量操作数包含两个复数，实部在低位，虚部在高位；该指令会拆分成两部分，先计算最终结果的实数和虚数的部分积并进行取负操作后，再通过乘累加得到两个复数相乘后的结果。该指令会完成两次舍入操作，乘法和乘累加计算完毕后各执行一次舍入操作。 
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令	
操作	tmp(2i+1) = vr_x(2i) * vr_y(2i+1) tmp(2i) = vr_x(2i) * vr_y(2i) vr_z(2i+1) = tmp(2i+1) - vr_x(2i+1) * vr_y(2i) vr_z(2i) = tmp(2i) + vr_x(2i+1) * vr_y(2i+1) i = 0 : 128/32/2 -1 注：tmp(i) 乘法结果做一次舍入饱和操作，vr_z 中乘累加结果再做一次舍入饱和操作。
数据类型:	单精度
语法	vfcmulcn.t vr _z , vr _x , vr _y
说明	复数共轭乘法计算-conj(x) * y，一个矢量操作数包含两个复数，实部在低位，虚部在高位；该指令会拆分成两部分，先计算最终结果的实数和虚数的部分积并进行取负操作后，再通过乘累加得到两个复数相乘后的结果。该指令会完成两次舍入操作，乘法和乘累加计算完毕后各执行一次舍入操作。
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令格式:

vfcmulcn.f32

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11101		VRY		VRX		101001		01011		VRZ	

15.19 VFCMULCNA.T——矢量浮点复数共轭乘累加指令

统一化指令																																									
语法	<code>vfcmulcna.t vrz, vrz, vry</code> (t=f32)																																								
操作	$\text{tmp}(2i+1) = \text{vrz}(2i+1) - \text{vrz}(2i) * \text{vry}(2i+1)$ $\text{tmp}(2i) = \text{vrz}(2i) - \text{vrz}(2i) * \text{vry}(2i)$ $\text{vrz}(2i+1) = \text{tmp}(2i+1) + \text{vrz}(2i+1) * \text{vry}(2i)$ $\text{vrz}(2i) = \text{tmp}(2i) - \text{vrz}(2i+1) * \text{vry}(2i+1)$ $i = 0 : 128/32/2 - 1$ 注：tmp(i) 乘法结果做一次舍入饱和操作，vrz 中乘累加结果再做一次舍入饱和操作。																																								
编译结果	仅存在 32 位指令 <code>vfcmulcna.t vrz, vrz, vry</code>																																								
数据类型:	单精度																																								
说明	复数共轭乘累加计算 $z + (-\text{conj}(x) * y)$，一个矢量操作数包含两个复数，实部在低位，虚部在高位；该指令会拆分成两部分，先通过乘累加计算最终结果的实数和虚数的部分结果，再通过一次乘累加得到两个复数相乘再加上一个复数的结果。该指令会完成两次舍入操作，乘法和乘累加计算完毕后各执行一次舍入操作。 VRX <table><tr><td>3</td><td>2</td><td>1</td><td>0</td></tr></table> VRZ <table><tr><td>3</td><td>2</td><td>1</td><td>0</td></tr></table> 取负 <table><tr><td>3</td><td>2</td><td>1</td><td>0</td></tr></table> S <table><tr><td>3</td><td>2</td><td>1</td><td>0</td></tr></table> tmp <table><tr><td>3</td><td>2</td><td>1</td><td>0</td></tr></table> VRX <table><tr><td>3</td><td>2</td><td>1</td><td>0</td></tr></table> VRZ <table><tr><td>3</td><td>2</td><td>1</td><td>0</td></tr></table> 取负 <table><tr><td>3</td><td>2</td><td>1</td><td>0</td></tr></table> S <table><tr><td>3</td><td>2</td><td>1</td><td>0</td></tr></table> tmp/VRZ <table><tr><td>3</td><td>2</td><td>1</td><td>0</td></tr></table>	3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0
3	2	1	0																																						
3	2	1	0																																						
3	2	1	0																																						
3	2	1	0																																						
3	2	1	0																																						
3	2	1	0																																						
3	2	1	0																																						
3	2	1	0																																						
3	2	1	0																																						
3	2	1	0																																						
影响标志位	无影响																																								
异常	IO、ID、IX、UF、OF																																								

32 位指令	
操作	tmp(2i+1) = vrz(2i+1) - vr_x(2i) * vr_y(2i+1) tmp(2i) = vrz(2i) - vr_x(2i) * vr_y(2i) vrz(2i+1) = tmp(2i+1) + vr_x(2i+1) * vr_y(2i) vrz(2i) = tmp(2i) - vr_x(2i+1) * vr_y(2i+1) i = 0 : 128/32/2 - 1 注：tmp(i) 乘法结果做一次舍入饱和操作，vrz 中乘累加结果再做一次舍入饱和操作
数据类型:	单精度
语法	vfcmlcna.t vrz, vr _x , vr _y
说明	复数共轭乘累加计算 $z + (-\text{conj}(x) * y)$，一个矢量操作数包含两个复数，实部在低位，虚部在高位；该指令会拆分成两部分，先通过乘累加计算最终结果的实数和虚数的部分结果，再通过一次乘累加得到两个复数相乘再加上一个复数的结果。该指令会完成两次舍入操作，乘法和乘累加计算完毕后各执行一次舍入操作。
影响标志位	无影响
异常	IO、ID、IX、UF、OF

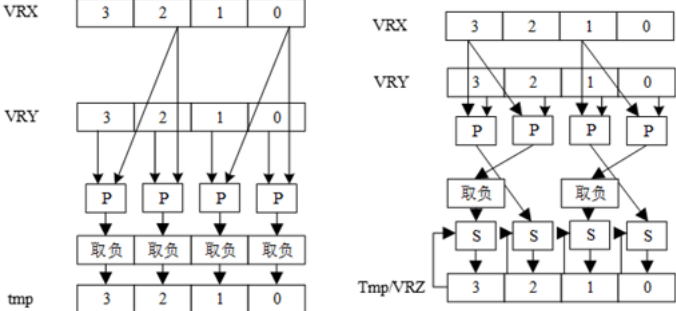
32 位指令格式:

vfcmlcna.f32

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11101	VRY	VRX	101001	10011	VRZ						

15.20 VFCMULN.T——矢量浮点复数取负乘法指令

统一化指令																									
语法	vfcmuln.t vrz, vrx, vry (t=f32)																								
操作	$tmp(2i+1) = -vrx(2i) * vry(2i+1)$ $tmp(2i) = -vrx(2i) * vry(2i)$ $vrz(2i+1) = tmp(2i+1) - vrx(2i+1) * vry(2i)$ $vrz(2i) = tmp(2i) + vrx(2i+1) * vry(2i+1)$ $i = 0 : 128/32/2 - 1$ 注：tmp(i) 乘法结果做一次舍入饱和操作，vrz 中乘累加结果再做一次舍入饱和操作																								
编译结果	仅存在 32 位指令 vfcmuln.t vrz, vrx, vry																								
数据类型:	单精度																								
说明	复数取负乘法计算 $(-x) * y$，一个矢量操作数包含两个复数，实部在低位，虚部在高位；该指令会拆分成两部分，先计算最终结果的实数和虚数的部分积并进行取负操作，再通过乘累加得到两个复数相乘后的结果。该指令会完成两次舍入操作，乘法和乘累加计算完毕后各执行一次舍入操作。 VRX <table><tr><td>3</td><td>2</td><td>1</td><td>0</td></tr></table> VRy <table><tr><td>3</td><td>2</td><td>1</td><td>0</td></tr></table> tmp <table><tr><td>3</td><td>2</td><td>1</td><td>0</td></tr></table> VRX <table><tr><td>3</td><td>2</td><td>1</td><td>0</td></tr></table> VRy <table><tr><td>3</td><td>2</td><td>1</td><td>0</td></tr></table> Tmp^VRZ <table><tr><td>3</td><td>2</td><td>1</td><td>0</td></tr></table>	3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0
3	2	1	0																						
3	2	1	0																						
3	2	1	0																						
3	2	1	0																						
3	2	1	0																						
3	2	1	0																						
影响标志位	无影响																								
异常	IO、ID、IX、UF、OF																								

32 位指令	
操作	tmp(2i+1) = -vrx(2i) * vry(2i+1) tmp(2i) = -vrx(2i) * vry(2i) vrz(2i+1) = tmp(2i+1) - vrx(2i+1) * vry(2i) vrz(2i) = tmp(2i) + vrx(2i+1) * vry(2i+1) i = 0 : 128/32/2 -1 注：tmp(i) 乘法结果做一次舍入饱和操作，vrz 中乘累加结果再做一次舍入饱和操作
数据类型:	单精度
语法	vfcmuln.t vrz, vrx, vry
说明	复数取负乘法计算 $(-x) * y$，一个矢量操作数包含两个复数，实部在低位，虚部在高位；该指令会拆分成两部分，先计算最终结果的实数和虚数的部分积并进行取负操作，再通过乘累加得到两个复数相乘后的结果。该指令会完成两次舍入操作，乘法和乘累加计算完毕后各执行一次舍入操作。 
影响标志位	无影响
异常	IO、ID、IX、UF、OF

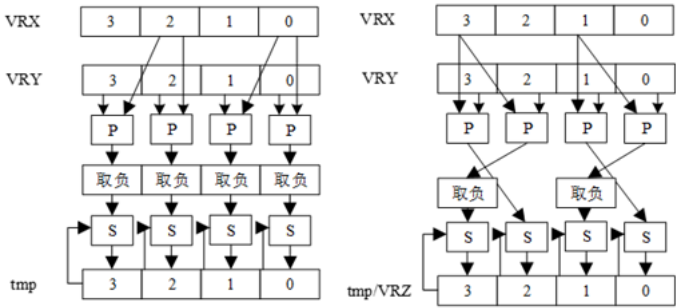
32 位指令格式：

vfcmuln.f32

31	30	26	25	21	20	16	15	10	9	5	4	0
1		11101		VRY		VRX		101001		01001		VRZ

15.21 VFCMULNA.T——矢量浮点复数取负乘累加指令

统一化指令	
语法	<code>vfcmulna.t vrz, vrx, vry</code> (<code>t=f32</code>)
操作	$tmp(2i+1) = vrz(2i+1) - vrz(2i) * vry(2i+1)$ $tmp(2i) = vrz(2i) - vrz(2i) * vry(2i)$ $vrz(2i+1) = tmp(2i+1) - vrz(2i+1) * vry(2i)$ $vrz(2i) = tmp(2i) + vrz(2i+1) * vry(2i+1)$ $i = 0 : 128/32/2 - 1$ 注：tmp(i) 乘法结果做一次舍入饱和操作，vrz 中乘累加结果再做一次舍入饱和操作
编译结果	仅存在 32 位指令 <code>vfcmulna.t vrz, vrx, vry</code>
数据类型:	单精度
说明	复数取负乘累加计算 $z + (-x) * y$，一个矢量操作数包含两个复数，实部在低位，虚部在高位；该指令会拆分成两部分，先通过乘累加计算最终结果的实数和虚数的部分结果，再通过一次乘累加得到两个复数相乘再加上一个复数的结果。该指令会完成两次舍入操作，乘法和乘累加计算完毕后各执行一次舍入操作。 VRX 3210 VRY 3210 P P P P 取负 取负 取负 取负 S S S S tmp 3210 VRX 3210 VRY 3210 P P P P 取负 取负 S S S S tmp/VRZ 3210
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令	
操作	tmp(2i+1) = vrz(2i+1) - vrx(2i) * vry(2i+1) tmp(2i) = vrz(2i) - vrx(2i) * vry(2i) vrz(2i+1) = tmp(2i+1) - vrx(2i+1) * vry(2i) vrz(2i) = tmp(2i) + vrx(2i+1) * vry(2i+1) i = 0 : 128/32/2 -1 注：tmp(i) 乘法结果做一次舍入饱和操作，vrz 中乘累加结果再做一次舍入饱和操作
数据类型:	单精度
语法	vfcmulna.t vrz, vrx, vry
说明	复数取负乘累加计算 $z + (-x) * y$，一个矢量操作数包含两个复数，实部在低位，虚部在高位；该指令会拆分成两部分，先通过乘累加计算最终结果的实数和虚数的部分结果，再通过一次乘累加得到两个复数相乘再加上一个复数的结果。该指令会完成两次舍入操作，乘法和乘累加计算完毕后各执行一次舍入操作。 
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令格式：

vfcmulna.f32

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11101	VRY	VRX	101001	10001	VRZ						

15.22 VFMULA.T——矢量浮点融合乘累加指令

统一化指令	
语法	vfimula.t vrz, vr _x , vr _y (t=f32)
操作	vrz(i) = vrz(i) + vr _x (i) * vr _y (i) i = 0 : 128/32 -1
编译结果	仅存在 32 位指令 vfimula.t vrz, vr _x , vr _y
数据类型:	单精度
说明	将 vr_x 和 vr_y 中的对应元素相乘，保留得到的乘积的所有精度，再与 vrz 中对应的元素相加，最终结果写回 vrz 中。 <pre> graph TD VRX[VRX: 3, 2, 1, 0] --> M1[x] VRX --> M2[x] VRX --> M3[x] VRX --> M4[x] VRY[VRY: 3, 2, 1, 0] --> M1 VRY --> M2 VRY --> M3 VRY --> M4 M1 --> A1[+] M2 --> A2[+] M3 --> A3[+] M4 --> A4[+] VRZ[VRZ: 3, 2, 1, 0] --> A1 VRZ --> A2 VRZ --> A3 VRZ --> A4 A1 --> VRZ A2 --> VRZ A3 --> VRZ A4 --> VRZ </pre>
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令	
操作	$vrz(i) = vrz(i) + vrx(i) * vry(i)$ $i = 0 : 128/32 - 1$
语法	vformula.t vrx, vry
数据类型:	单精度
说明	将 <code>vrx</code> 和 <code>vry</code> 中的对应元素相乘，保留的到的乘积的所有精度，再与 <code>vrz</code> 中对应的元素相加，最终结果写回 <code>vrz</code> 中。 <pre> graph TD VRX[VRX: 3, 2, 1, 0] --> M1[x] VRX --> M2[x] VRX --> M3[x] VRX --> M4[x] VRY[VRY: 3, 2, 1, 0] --> M1 VRY --> M2 VRY --> M3 VRY --> M4 M1 --> A1[+] M2 --> A2[+] M3 --> A3[+] M4 --> A4[+] VRZ[VRZ: 3, 2, 1, 0] --> A1 VRZ --> A2 VRZ --> A3 VRZ --> A4 A1 --> VRZ A2 --> VRZ A3 --> VRZ A4 --> VRZ </pre>
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令格式:

vfmula.f32

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11101			VR _Y		VR _X		101000		11100		VR _Z

15.23 VFMULS.T——矢量浮点乘取负累加指令

统一化指令	
语法	vfmls.t vrz, vrz, vry (t=f32)
操作	$vrz(i) = vrz(i) - vrz(i) * vry(i)$ $i = 0 : 128/32 - 1$
编译结果	仅存在 32 位指令 vfmls.t vrz, vrz, vry
数据类型:	单精度
说明	将 <code>vrz</code> 和 <code>vry</code> 中的对应元素相乘，保留得到的乘积的所有精度并进行取负操作，再与 <code>vrz</code> 中对应的元素相加，最终结果写回 <code>vrz</code> 中。
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令	
操作	$vrz(i) = vrz(i) - vrz(i) * vry(i)$ $i = 0 : 128/32 - 1$
语法	<code>vfmuls.t vrz, vrz, vry</code>
数据类型:	单精度
说明	将 <code>vrz</code> 和 <code>vry</code> 中的对应元素相乘，保留得到的乘积的所有精度并进行取负操作，再与 <code>vrz</code> 中对应的元素相加，最终结果写回 <code>vrz</code> 中。 <pre> graph TD VRX[VRX: 3, 2, 1, 0] --> VRY[VRY: 3, 2, 1, 0] VRY --> X[x] X --> Neg[取负] Neg --> Add[+] Add --> VRZ[VRZ: 3, 2, 1, 0] VRZ --> Add Add --> VRZ </pre>
影响标志位	无影响
异常	IO、ID、IX、UF、OF

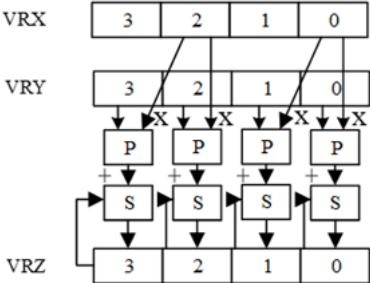
32 位指令格式:

vfmuls.f32

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11101			VR _Y		VR _X		101001		00000		VR _Z

15.24 VFMULXAA.T——矢量浮点复数乘累加实部虚部部分计算指令

统一化指令																					
语法	<code>vfmulxaa.t vrz, vrz, vry</code> (t=f32)																				
操作	$vrz(2i+1) = vrz(2i+1) + vrz(2i) * vry(2i+1)$ $vrz(2i) = vrz(2i) + vrz(2i) * vry(2i)$ $i = 0 : 128/32/2 - 1$																				
编译结果	仅存在 32 位指令 <code>vfmulxaa.t vrz, vrz, vry</code>																				
数据类型:	单精度																				
说明	复数的实部和虚部部分计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；将 <code>vrz</code> 和 <code>vry</code> 的对应元素交织相乘后，保留所得到乘积的所有精度后，与 <code>vrz</code> 的对应元素相加，结果写回到 <code>vrz</code> 的对应元素中。 VRX <table><tr><td>3</td><td>2</td><td>1</td><td>0</td></tr></table> VRY <table><tr><td>3</td><td>2</td><td>1</td><td>0</td></tr></table> X <table><tr><td>P</td><td>P</td><td>P</td><td>P</td></tr></table> + <table><tr><td>S</td><td>S</td><td>S</td><td>S</td></tr></table> VRZ <table><tr><td>3</td><td>2</td><td>1</td><td>0</td></tr></table>	3	2	1	0	3	2	1	0	P	P	P	P	S	S	S	S	3	2	1	0
3	2	1	0																		
3	2	1	0																		
P	P	P	P																		
S	S	S	S																		
3	2	1	0																		
影响标志位	无影响																				
异常	IO、ID、IX、UF、OF																				

32 位指令	
操作	$vrz(2i+1) = vrz(2i+1) + vrx(2i) * vry(2i+1)$ $vrz(2i) = vrz(2i) + vrx(2i) * vry(2i)$ $i = 0 : 128/32/2 - 1$
语法	<code>vfmulxaa.t vrz, vrx, vry</code>
数据类型:	单精度
说明	复数的实部和虚部部分计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；将 <code>vrx</code> 和 <code>vry</code> 的对应元素交织相乘后，保留所得到乘积的所有精度后，与 <code>vrz</code> 的对应元素相加，结果写回到 <code>vrz</code> 的对应元素中。 
影响标志位	无影响
异常	IO、ID、IX、UF、OF

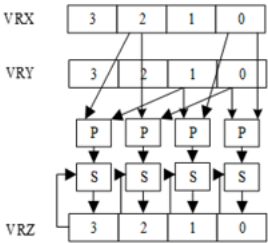
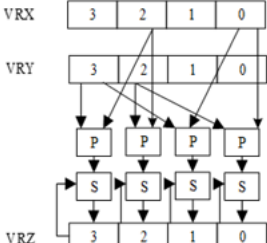
32 位指令格式：

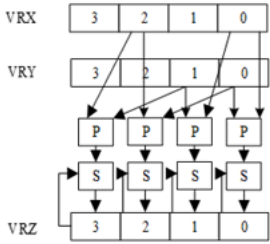
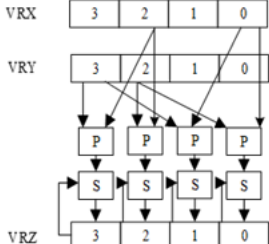
vfmulxaa.f32

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11101	VRY	VRX	101000	01011	VRZ						

15.25

VFMULXAAI.T——矢量浮点索引复数乘累加实部虚部部分计算指令

统一化指令	
语法	vfmulxaai.t vrz, vrx, vry, index (t=f32)
操作	$vrz(2i+1) = vrz(2i+1) + vrx(2i) * vry(2index+1)$ $vrz(2i) = vrz(2i) + vrx(2i) * vry(2index)$ $i = 0 : 128/32/2 - 1$ $index = 0 \sim 128 / (32 * 2) - 1$
编译结果	仅存在 32 位指令 vfmulxaai.t vrz, vrx, vry, index
数据类型:	单精度
说明	复数的实部和虚部部分计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；将 vrx 对应元素和 vry 中由 index 指定的元素交织相乘后，保留所得乘积的所有精度后，与 vrz 的对应元素相加，结果写回到 vrz 的对应元素中。 index = 0  index = 1 
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令	
操作	$vrz(2i+1) = vrz(2i+1) + vrx(2i) * vry(2index+1)$ $vrz(2i) = vrz(2i) + vrx(2i) * vry(2index)$ $i = 0 : 128/32/2 - 1$ $index = 0 \sim 128 / (32 * 2) - 1$
语法	<code>vfmulxaai.t vrz, vrx, vry, index</code>
数据类型:	单精度
说明	复数的实部和虚部部分计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；将 <code>vrx</code> 对应元素和 <code>vry</code> 中由 <code>index</code> 指定的元素交织相乘后，保留所得到乘积的所有精度后，与 <code>vrz</code> 的对应元素相加，结果写回到 <code>vrz</code> 的对应元素中。 index = 0  index = 1 
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令格式：

vfmulxaai.f32

31	30	26	25	21	20	16	15	9	8	7	6	5	4	0
1	11101	VRY	VRX	1000101	IMM2	0	1	VRZ						

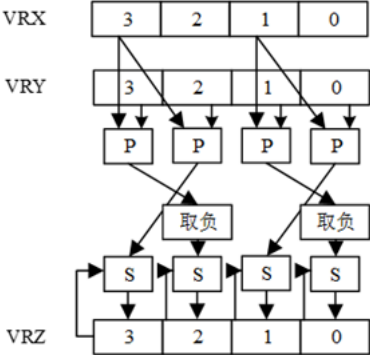
注：IMM2 的取值为：2’ b00 ~ 2’ b01

index = 0, IMM2 = 2’ b00

index = 1, IMM2 = 2’ b01

15.26 VFMULXAS.T——矢量浮点复数乘累加实部虚部部分计算指令

统一化指令	
语法	<code>vfmulxas.t vrz, vrz, vrz, vary</code> (t=f32)
操作	$vrz(2i+1) = vrz(2i+1) + vrz(2i+1) * vary(2i)$ $vrz(2i) = vrz(2i) - vrz(2i+1) * vary(2i+1)$ $i = 0 : 128/32/2 - 1$
编译结果	仅存在 32 位指令 <code>vfmulxaa.t vrz, vrz, vary</code>
数据类型:	单精度
说明	复数的实部和虚部部分计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；将 <code>vrz</code> 和 <code>vary</code> 的对应元素交织相乘后，虚数部分保留所得到乘积的所有精度后，与 <code>vrz</code> 的虚数部分对应元素相加；实数部分保留所得到乘积的所有精度并取负后，与 <code>vrz</code> 的实数部分相加，结果写回到 <code>vrz</code> 的对应元素中 <pre>graph TD VRX[VRX: 3, 2, 1, 0] --> P1[P] VRX --> P2[P] VRX --> P3[P] VRX --> P4[P] VRY[VRY: 3, 2, 1, 0] --> P1 VRY --> P2 VRY --> P3 VRY --> P4 P1 --> Neg1[取负] P2 --> Neg2[取负] P3 --> Neg3[取负] P4 --> Neg4[取负] Neg1 --> S1[S] Neg2 --> S2[S] Neg3 --> S3[S] Neg4 --> S4[S] VRZ[VRZ: 3, 2, 1, 0] --> S1 VRZ --> S2 VRZ --> S3 VRZ --> S4 S1 --> VRZ S2 --> VRZ S3 --> VRZ S4 --> VRZ</pre>
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令	
操作	$vrz(2i+1) = vrz(2i+1) + vrx(2i+1) * vry(2i)$ $vrz(2i) = vrz(2i) - vrx(2i+1) * vry(2i+1)$ $i = 0 : 128/32/2 - 1$
语法	<code>vfmulxas.t vrz, vrx, vry</code>
数据类型:	单精度
说明	复数的实部和虚部部分计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；将 <code>vrx</code> 和 <code>vry</code> 的对应元素交织相乘后，虚数部分保留所得到乘积的所有精度后，与 <code>vrz</code> 的虚数部分对应元素相加；实数部分保留所得到乘积的所有精度并取负后，与 <code>vrz</code> 的实数部分相加，结果写回到 <code>vrz</code> 的对应元素中。 
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令格式:

vfmulxas.f32

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11101	VRY	VRX	101000	01101	VRZ						

15.27 VFMULXASI.T——矢量浮点复数乘累加实部虚部部分计算指令

统一化指令	
语法	<code>vfmulxasi.t vrz, vrz, vry, index</code> (t=f32)
操作	$vrz(2i+1) = vrz(2i+1) + vrz(2i+1) * vry(2index)$ $vrz(2i) = vrz(2i) - vrz(2i+1) * vry(2index+1)$ $i = 0 : 128/32/2 - 1$ $index = 0 \sim 128 / (32*2) - 1$
编译结果	仅存在 32 位指令 <code>vfmulxasi.t vrz, vrz, vry, index</code>
数据类型:	单精度
说明	复数的实部和虚部部分计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；将 <code>vrz</code> 和 <code>vry</code> 的对应元素交织相乘后，虚数部分保留所得到乘积的所有精度后，与 <code>vrz</code> 的虚数部分对应元素相加；实数部分保留所得到乘积的所有精度并取负后，与 <code>vrz</code> 的实数部分相加，结果写回到 <code>vrz</code> 的对应元素中。 index = 0 VRX: 3 2 1 0 VRY: 3 2 1 0 VRZ: 3 2 1 0 index = 1 VRX: 3 2 1 0 VRY: 3 2 1 0 VRZ: 3 2 1 0
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令	
操作	$vrz(2i+1) = vrz(2i+1) + vrx(2i+1) * vry(2index)$ $vrz(2i) = vrz(2i) - vrx(2i+1) * vry(2index+1)$ $i = 0 : 128/32/2 - 1$ $index = 0 \sim 128 / (32*2) - 1$
语法	vfmulxasi.t vrx, vry, index
数据类型:	单精度
说明	复数的实部和虚部部分计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；将 vrx 和 vry 的对应元素交织相乘后，虚数部分保留所得乘积的所有精度后，与 vrz 的虚数部分对应元素相加；实数部分保留所得乘积的所有精度并取负后，与 vrz 的实数部分相加，结果写回到 vrz 的对应元素中。 index = 0 index = 1
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令格式:

vfmulxasi.f32

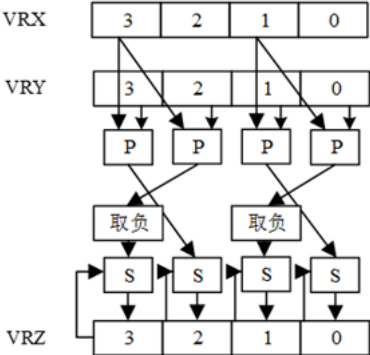
31	30	26	25	21	20	16	15	9	8	7	6	5	4	0
1	11101	VRY	VRX	1000110	IMM2	0	1	VRZ						

注：IMM2 的取值为：2' b00 ~ 2' b01

index = 0, IMM2 = 2' b00

index = 1, IMM2 = 2' b01

15.28 VFMULXSA.T——矢量浮点复数乘累加实部虚部部分计算指令

统一化指令	
语法	<code>vfmulxsa.t vrz, vrz, vrz, vrz</code> (t=f32)
操作	$vrz(2i+1) = vrz(2i+1) - vrz(2i+1) * vrz(2i)$ $vrz(2i) = vrz(2i) + vrz(2i+1) * vrz(2i+1)$ $i = 0 : 128/32/2 -1$
编译结果	仅存在 32 位指令 <code>vfmulxsa.t vrz, vrz, vrz, vrz</code>
数据类型:	单精度
说明	复数的实部和虚部部分计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；将 <code>vrz</code> 和 <code>vrz</code> 的对应元素交织相乘后，实数部分保留所得到乘积的所有精度后，与 <code>vrz</code> 的实数部分对应元素相加；虚数部分保留所得到乘积的所有精度并取负后，与 <code>vrz</code> 的虚数部分相加，结果写回到 <code>vrz</code> 的对应元素中。  The diagram illustrates the VFMULXSA.T instruction operation. It shows four 4-element registers: VRX, VRY, and VRZ. VRX and VRY are inputs. VRZ is the destination. The diagram shows element-wise multiplication (P) between VRX and VRY. The results are then processed by '取负' (Negate) and 'S' (Shift/Sign) blocks before being added to the corresponding elements of VRZ.
影响标志位	无影响
异常	IO、ID、IX、UF、OF

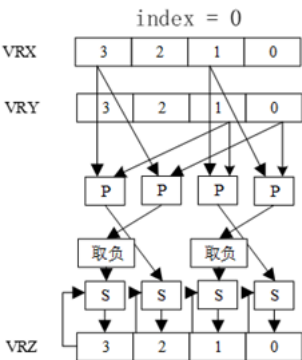
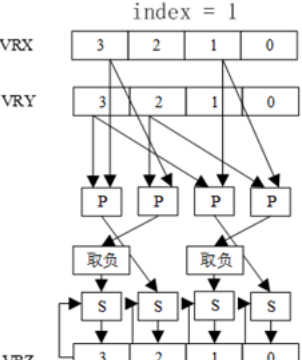
32 位指令	
操作	$vrz(2i+1) = vrz(2i+1) - vrx(2i+1) * vry(2i)$ $vrz(2i) = vrz(2i) + vrx(2i+1) * vry(2i+1)$ $i = 0 : 128/32/2 - 1$
语法	<code>vfmulxsa.t vrz, vrx, vry</code>
数据类型:	单精度
说明	复数的实部和虚部部分计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；将 <code>vrx</code> 和 <code>vry</code> 的对应元素交织相乘后，实数部分保留所得到乘积的所有精度后，与 <code>vrz</code> 的实数部分对应元素相加；虚数部分保留所得到乘积的所有精度并取负后，与 <code>vrz</code> 的虚数部分相加，结果写回到 <code>vrz</code> 的对应元素中。
影响标志位	无影响
异常	IO、ID、IX、UF、OF

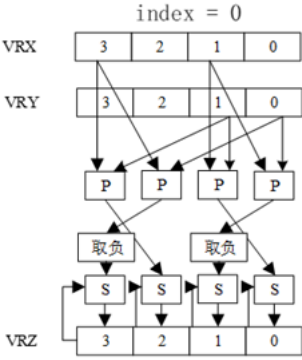
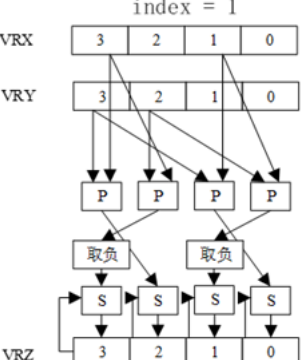
32 位指令格式:

vfmulxsa.f32

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11101	VRY	VRX	101000	10001	VRZ						

15.29 VFMULXSAT——矢量浮点复数乘累加实部虚部部分计算指令

统一化指令	
语法	vfmulxsai.t vrz, vrz, vry, index (t=f32)
操作	$vrz(2i+1) = vrz(2i+1) - vrz(2i+1) * vry(2index)$ $vrz(2i) = vrz(2i) + vrz(2i+1) * vry(2index+1)$ $i = 0 : 128/32/2 - 1$ $index = 0 \sim 128 / (32*2) - 1$
编译结果	仅存在 32 位指令 vfmulxsai.t vrz, vrz, vry, index
数据类型:	单精度
说明	复数的实部和虚部部分计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；将 vrz 和 vry 的对应元素交织相乘后，实数部分保留所得到乘积的所有精度后，与 vrz 的实数部分对应元素相加；虚数部分保留所得到乘积的所有精度并取负后，与 vrz 的虚数部分相加，结果写回到 vrz 的对应元素中。 index = 0  index = 1 
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令	
操作	$vrz(2i+1) = vrz(2i+1) - vrx(2i+1) * vry(2index)$ $vrz(2i) = vrz(2i) + vrx(2i+1) * vry(2index+1)$ $i = 0 : 128/32/2 - 1$ $index = 0 \sim 128 / (32*2) - 1$
语法	<code>vfmulxsai.t vrz, vrx, vry, index</code>
数据类型:	单精度
说明	复数的实部和虚部部分计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；将 <code>vrx</code> 和 <code>vry</code> 的对应元素交织相乘后，实数部分保留所得到乘积的所有精度后，与 <code>vrz</code> 的实数部分对应元素相加；虚数部分保留所得到乘积的所有精度并取负后，与 <code>vrz</code> 的虚数部分相加，结果写回到 <code>vrz</code> 的对应元素中。 index = 0  index = 1 
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令格式:

vfmulxsai.f32

31	30	26	25	21	20	16	15	9	8	7	6	5	4	0
1	11101		VRY		VRX		1001000		IMM2	0	1		VRZ	

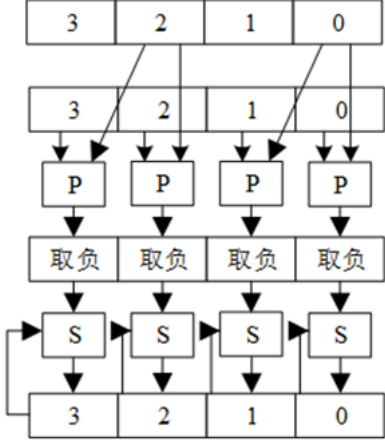
注：IMM2 的取值为：2’ b00 ~ 2’ b01

index = 0, IMM2 = 2’ b00

index = 1, IMM2 = 2’ b01

15.30 VFMULXSS.T——矢量浮点复数乘累加实部虚部部分计算指令

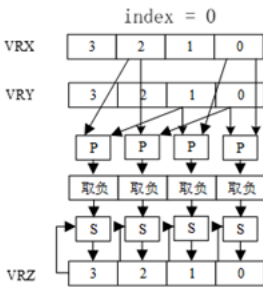
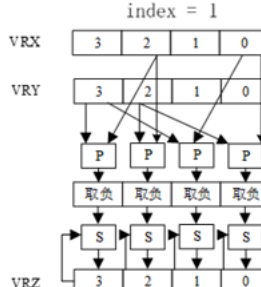
统一化指令	
语法	<code>vfmulxss.t vrz, vrz, vrz, vry</code> (t=f32)
操作	$vrz(2i+1) = vrz(2i+1) - vrz(2i) * vry(2i+1)$ $vrz(2i) = vrz(2i) - vrz(2i) * vry(2i)$ $i = 0 : 128/32/2 - 1$
编译结果	仅存在 32 位指令 <code>vfmulxss.t vrz, vrz, vrz, vry</code>
数据类型:	单精度
说明	复数的实部和虚部部分计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；将 <code>vrz</code> 和 <code>vry</code> 的对应元素交织相乘后，保留所得到乘积的所有精度并取负后，与 <code>vrz</code> 的对应元素相加，结果写回到 <code>vrz</code> 的对应元素中。 <pre>graph TD Input[3 2 1 0] --> P1[P] Input --> P2[P] Input --> P3[P] Input --> P4[P] P1 --> Neg1[取负] P2 --> Neg2[取负] P3 --> Neg3[取负] P4 --> Neg4[取负] Neg1 --> S1[S] Neg2 --> S2[S] Neg3 --> S3[S] Neg4 --> S4[S] S1 --> Output[3 2 1 0] S2 --> Output S3 --> Output S4 --> Output</pre>
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令	
操作	$vrz(2i+1) = vrz(2i+1) - vrx(2i) * vry(2i+1)$ $vrz(2i) = vrz(2i) - vrx(2i) * vry(2i)$ $i = 0 : 128/32/2 - 1$
语法	<code>vfmulxss.t vrz, vrx, vry</code>
数据类型:	单精度
说明	复数的实部和虚部部分计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；将 <code>vrx</code> 和 <code>vry</code> 的对应元素交织相乘后，保留所得乘积的所有精度并取负后，与 <code>vrz</code> 的对应元素相加，结果写回到 <code>vrz</code> 的对应元素中。 
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令格式:

vfmulxss.f32															
31	30	26	25	21	20	16	15	10	9	5	4	0			
1	11101			VRY			VRX			101000			01111		VRZ

15.31 VFMULXSSI.T——矢量浮点索引复数乘累加实部虚部计算指令

统一化指令	
语法	vfmulxssi.t vrz, vrx, vry, index (t=f32)
操作	$vrz(2i+1) = vrz(2i+1) - vrz(2i) * vry(2index+1)$ $vrz(2i) = vrz(2i) - vrz(2i) * vry(2index)$ $i = 0 : 128/32/2 - 1$ $index = 0 \sim 128 / (32 * 2) - 1$
编译结果	仅存在 32 位指令 vfmulxssi.t vrz, vrx, vry, index
数据类型:	单精度
说明	复数的实部和虚部部分计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；将 vrz 对应元素和 vry 中由 index 指定的元素交织相乘后，保留所得乘积的所有精度并取负后，与 vrz 的对应元素相加，结果写回到 vrz 的对应元素中。 index = 0  index = 1 
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令	
操作	$vrz(2i+1) = vrz(2i+1) - vrx(2i) * vry(2index+1)$ $vrz(2i) = vrz(2i) - vrx(2i) * vry(2index)$ $i = 0 : 128/32/2 - 1$ $index = 0 \sim 128 / (32 * 2) - 1$
语法	vfmulxssi.t vrx, vry, index
数据类型:	单精度
说明	复数的实部和虚部部分计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；将 vrx 对应元素和 vry 中由 index 指定的元素交织相乘后，保留所得到乘积的所有精度并取负后，与 vrz 的对应元素相加，结果写回到 vrz 的对应元素中。 index = 0 index = 1
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令格式:

vfmulxssi.f32

31	30	26	25	21	20	16	15	9	8	7	6	5	4	0
1	11101			VR _Y		VR _X		1000111		IMM2		0	1	VR _Z

注：IMM2 的取值为：2' b00 ~ 2' b01

index = 0, IMM2 = 2' b00

index = 1, IMM2 = 2' b01

15.32 VFNMULA.T——矢量浮点融合乘取负累减指令

统一化指令	
语法	vfmfmula.t vrz, vr _x , vr _y (t=f32)
操作	vrz(i) = -vrz(i) - vr _x (i) * vr _y (i) i = 0 : 128/32 -1
编译结果	仅存在 32 位指令 vfmfmula.t vrz, vr _x , vr _y
数据类型:	单精度
说明	将 vr_x 和 vr_y 中的对应元素相乘，保留得到的乘积的所有精度并进行取负操作，再与 vrz 中对应的元素相减，最终结果写回 vrz 中。 <pre> graph TD VRX[VRX: 3, 2, 1, 0] --> M1[x] VRX --> M2[x] VRX --> M3[x] VRX --> M4[x] VRY[VRY: 3, 2, 1, 0] --> M1 VRY --> M2 VRY --> M3 VRY --> M4 M1 --> N1[取负] M2 --> N2[取负] M3 --> N3[取负] M4 --> N4[取负] VRZ[VRZ: 3, 2, 1, 0] --> S1[-] VRZ --> S2[-] VRZ --> S3[-] VRZ --> S4[-] N1 --> S1 N2 --> S2 N3 --> S3 N4 --> S4 S1 --> VRZ_3[3] S2 --> VRZ_2[2] S3 --> VRZ_1[1] S4 --> VRZ_0[0] </pre>
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令	
操作	$vrz(i) = -vrz(i) - vrz(i) * vry(i)$ $i = 0 : 128/32 - 1$
语法	vfmmla.t vrz, vrz, vry
数据类型:	单精度
说明	将 <code>vrz</code> 和 <code>vry</code> 中的对应元素相乘，保留得到的乘积的所有精度并进行取负操作，再与 <code>vrz</code> 中对应的元素相加，最终结果写回 <code>vrz</code> 中。 <pre> graph TD VRX[VRX: 3, 2, 1, 0] --> M1((x)) VRY[VRY: 3, 2, 1, 0] --> M1 M1 --> N1[取负] VRX --> M2((x)) VRY --> M2 M2 --> N2[取负] VRX --> M3((x)) VRY --> M3 M3 --> N3[取负] VRX --> M4((x)) VRY --> M4 M4 --> N4[取负] N1 --> A1[-] VRZ[VRZ: 3, 2, 1, 0] --> A1 A1 --> VRZ N2 --> A2[-] VRZ --> A2 A2 --> VRZ N3 --> A3[-] VRZ --> A3 A3 --> VRZ N4 --> A4[-] VRZ --> A4 A4 --> VRZ </pre>
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令格式:

vfnmula.f32

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11101			VR _Y		VR _X		101001		00010		VR _Z

15.33 VFNMULS.T——矢量浮点乘累减指令

统一化指令	
语法	vfnmuls.t vrz, vrz, vry (t=f32)
操作	$vrz(i) = -vrz(i) + vrz(i) * vry(i)$ $i = 0 : 128/32 - 1$
编译结果	仅存在 32 位指令 vfnmuls.t vrz, vrz, vry
数据类型:	单精度
说明	将 <code>vrz</code> 和 <code>vry</code> 中的对应元素相乘，保留得到的乘积的所有精度，再与 <code>vrz</code> 中对应的元素相减，最终结果写回 <code>vrz</code> 中。 <pre> graph TD VRX[VRX: 3, 2, 1, 0] --> M1[x] VRY[VRY: 3, 2, 1, 0] --> M1 M1 --> N1[取负] N1 --> A1[-] VRZ[VRZ: 3, 2, 1, 0] --> A1 A1 --> VRZ M2[x] --> N2[取负] N2 --> A2[-] VRZ --> A2 A2 --> VRZ M3[x] --> N3[取负] N3 --> A3[-] VRZ --> A3 A3 --> VRZ M4[x] --> N4[取负] N4 --> A4[-] VRZ --> A4 A4 --> VRZ </pre>
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令	
操作	$vrz(i) = vrz(i) - vrz(i) * vry(i)$ $i = 0 : 128/32 - 1$
语法	<code>vfnmuls.t vrz, vrz, vry</code>
数据类型:	单精度
说明	将 <code>vrz</code> 和 <code>vry</code> 中的对应元素相乘，保留得到的乘积的所有精度，再与 <code>vrz</code> 中对应的元素相减，最终结果写回 <code>vrz</code> 中。 <pre> graph TD VRX[VRX: 3, 2, 1, 0] VRY[VRY: 3, 2, 1, 0] subgraph Units U3[x, 取负, -] U2[x, 取负, -] U1[x, 取负, -] U0[x, 取负, -] end VRZ[VRZ: 3, 2, 1, 0] VRX --> U3 VRX --> U2 VRX --> U1 VRX --> U0 VRY --> U3 VRY --> U2 VRY --> U1 VRY --> U0 U3 --> VRZ U2 --> VRZ U1 --> VRZ U0 --> VRZ </pre>
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令格式:

vfnmuls.f32

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11101			VR _Y		VR _X		101000		11110		VR _Z

15.34 VFTOFL.T.RM——矢量带舍入模式的浮点转换为整型浮点指令

统一化指令	
语法	vftofi.T.rm vrz, vrx (T=f32)
操作	vrz[31:0]=(float32/double)vrx[31:0]; vrz[63:32]=(float32/double)vrx[63:32]; vrz[95:64]=(float32/double)vrx[95:64]; vrz[127:96]=(float32/double)vrz[127:96];
编译结果	仅存在 32 位指令 vftofi.T.rm vrz, vrx
数据类型	单精度
说明	将 vrx[31:0] 中的浮点舍入为整数结果再用相同类型浮点数表示，保存至 vrz[31:0] 中。 将 vrx[63:32] 中的浮点舍入为整数结果再用相同类型浮点数表示，保存至 vrz[63:32] 中。 将 vrx[95:64] 中的浮点舍入为整数结果再用相同类型浮点数表示，保存至 vrz[95:64] 中。 将 vrx[127:96] 中的浮点舍入为整数结果再用相同类型浮点数表示，保存至 vrz[127:96] 中。 rm 为舍入方式。 RM 表征： 2’ b00: 就近舍入模式，对应的汇编指令为 fftoi.T.rm 2’ b01: 向 0 舍入模式，对应的汇编指令为 fftoi.T.rz 2’ b10: 向正无穷舍入模式，对应的汇编指令为 fftoi.T.rpi 2’ b11: 向负无穷舍入模式，对应的汇编指令为 fftoi.T.rni
影响标志位	无
限制	寄存器范围为 vr0-vr15。
异常	ID,IO,IX

32 位指令	
操作	vrz[31:0]=(float32/double)vrx[31:0]; vrz[63:32]=(float32/double)vrx[63:32]; vrz[95:64]=(float32/double)vrx[95:64]; vrz[127:96]=(float32/double)vrz[127:96];
语法	vftofi.T.rm vrz, vrx
数据类型	单精度
说明	将 vrx[31:0] 中的浮点舍入为整数结果再用相同类型浮点数表示，保存至 vrz[31:0] 中。 将 vrx[63:32] 中的浮点舍入为整数结果再用相同类型浮点数表示，保存至 vrz[63:32] 中。 将 vrx[95:64] 中的浮点舍入为整数结果再用相同类型浮点数表示，保存至 vrz[95:64] 中。 将 vrx[127:96] 中的浮点舍入为整数结果再用相同类型浮点数表示，保存至 vrz[127:96] 中。 rm 为舍入方式。 RM 表征： 2’ b00: 就近舍入模式，对应的汇编指令为 fftoi.T.rm 2’ b01: 向 0 舍入模式，对应的汇编指令为 fftoi.T.rz 2’ b10: 向正无穷舍入模式，对应的汇编指令为 fftoi.T.rpi 2’ b11: 向负无穷舍入模式，对应的汇编指令为 fftoi.T.rni
影响标志位	无
限制	寄存器范围为 vr0-vr15。
异常	ID,IO,IX

32 位指令格式：

vftofi.f32.rm

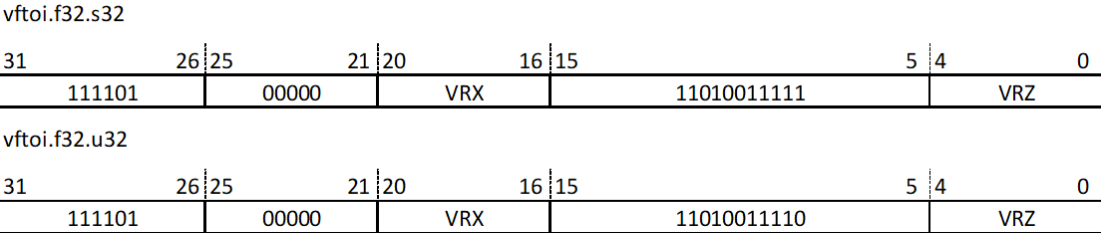
31	26	25	23	22	21	20	16	15	5	4	0
111101				000		RM	VRX		11011011110		VRZ

15.35 VFTOI.T1.T2——矢量浮点整数转换指令

统一化指令	
语法	vftoi.T1.T2 vrz, vrx (T1=f32) (T2=u32/s32)
操作	vrz[31:0]=(int)vrx[31:0]; vrz[63:32]=(int)vrx[63:32]; vrz[95:64]=(int)vrx[95:64]; vrz[127:96]=(int)vrz[127:96];
编译结果	仅存在 32 位指令。 vftoi.T1.T2 vrz, vrx
数据类型	单精度
说明	将 vrx[31:0] 中的浮点数转换成 16bit 有符号整数，保存至 vrz[31:0] 中。 将 vrx[63:32] 中的浮点数转换成 16bit 有符号整数，保存至 vrz[63:32] 中。 将 vrx[95:64] 中的浮点数转换成 16bit 有符号整数，保存至 vrz[95:64] 中。 将 vrx[127:96] 中的浮点数转换成 16bit 有符号整数，保存至 vrz[127:96] 中。
影响标志位	无
限制	寄存器范围为 vr0-vr15。
异常	ID,IO,IX

32 位指令	
操作	vrz[31:0]=(int)vrx[31:0]; vrz[63:32]=(int)vrx[63:32]; vrz[95:64]=(int)vrx[95:64]; vrz[127:96]=(int)vrz[127:96];
语法	vftoi.f16.s16 vrz, vrx
数据类型	单精度
说明	将 vrx[31:0] 中的浮点数转换成 16bit 有符号整数，保存至 vrz[31:0] 中。 将 vrx[63:32] 中的浮点数转换成 16bit 有符号整数，保存至 vrz[63:32] 中。 将 vrx[95:64] 中的浮点数转换成 16bit 有符号整数，保存至 vrz[95:64] 中。 将 vrx[127:96] 中的浮点数转换成 16bit 有符号整数，保存至 vrz[127:96] 中。
影响标志位	无
限制	寄存器范围为 vr0-vr15。
异常	ID,IO,IX

32 位指令格式：



15.36 VFTOI.T1.T2.RM——矢量带舍入模式的浮点整数转换指令

统一化指令	
语法	vftoi.T1.T2.rm vrz, vrx (T=f32) (T=u32/s32)
操作	vrz[31:0]=(int)vrx[31:0]; vrz[63:32]=(int)vrx[63:32]; vrz[95:64]=(int)vrx[95:64]; vrz[127:96]=(int)vrz[127:96];
编译结果	仅存在 32 位指令。 vftoi.T1.T2.rm vrz, vrx
数据类型	单精度
说明	将 vrz[31:0] 中的浮点数转换成 16bit 有符号整数，保存至 vrz[31:0] 中。 将 vrz[63:32] 中的浮点数转换成 16bit 有符号整数，保存至 vrz[63:32] 中。 将 vrz[95:64] 中的浮点数转换成 16bit 有符号整数，保存至 vrz[95:64] 中。 将 vrz[127:96] 中的浮点数转换成 16bit 有符号整数，保存至 vrz[127:96] 中。 RM 表征： 2’ b00: 就近舍入模式，对应的汇编指令为 vftoi.f16.s32.rn 2’ b01: 向 0 舍入模式，对应的汇编指令为 vftoi.f16.s32.rz 2’ b10: 向正无穷舍入模式，对应的汇编指令为 vftoi.f16.s32.rpi 2’ b11: 向负无穷舍入模式，对应的汇编指令为 vftoi.f16.s32.rni
影响标志位	无
限制	寄存器范围为 vr0-vr15。
异常	ID,IO,IX

32 位指令	
操作	vrz[31:0]=(int)vrx[31:0]; vrz[63:32]=(int)vrx[63:32]; vrz[95:64]=(int)vrx[95:64]; vrz[127:96]=(int)vrz[127:96];
语法	vftoi.T1.T2.rm vrz, vrx
数据类型	单精度
说明	将 vrx[31:0] 中的浮点数转换成 16bit 有符号整数，保存至 vrz[31:0] 中。 将 vrx[63:32] 中的浮点数转换成 16bit 有符号整数，保存至 vrz[63:32] 中。 将 vrx[95:64] 中的浮点数转换成 16bit 有符号整数，保存至 vrz[95:64] 中。 将 vrx[127:96] 中的浮点数转换成 16bit 有符号整数，保存至 vrz[127:96] 中。 RM 表征： 2’ b00: 就近舍入模式，对应的汇编指令为 vftoi.f16.s32.rn 2’ b01: 向 0 舍入模式，对应的汇编指令为 vftoi.f16.s32.rz 2’ b10: 向正无穷舍入模式，对应的汇编指令为 vftoi.f16.s32.rpi 2’ b11: 向负无穷舍入模式，对应的汇编指令为 vftoi.f16.s32.rni
影响标志位	无
限制	寄存器范围为 vr0-vr15。
异常	ID,IO,IX

32 位指令格式：

vftoi.f32.s32.rm											
31	26	25	23	22	21	20	16	15	5	4	0
111101			000		RM		VRX		110111010111		
									VRZ		

vftoi.f32.u32.rm											
31	26	25	23	22	21	20	16	15	5	4	0
111101			000		RM		VRX		11011010110		
									VRZ		

15.37 VFTOX.T1.T2——矢量浮点定点转换指令

统一化指令	
语法	vftox.T1.T2 vrz, vrx (T1=f32) (T2=u32/s32)
操作	vrz[31:0]=(fix)vrx[31:0]; vrz[63:32]=(fix)vrx[63:32]; vrz[95:64]=(fix)vrx[95:64]; vrz[127:96]=(fix)vrz[127:96];
编译结果	仅存在 32 位指令。 vftox.T1.T2 vrz, vrx
数据类型	单精度
说明	将 vr x[31:0] 中的浮点数转换成定点数，保存至 vrz[31:0] 中。 将 vrx[63:32] 中的浮点数转换成定点数，保存至 vrz[63:32] 中。 将 vrx[95:64] 中的浮点数转换成定点数，保存至 vrz[95:64] 中。 将 vrx[12 7:96] 中的浮点数转换成定点数，保存至 vrz[127:96] 中。 FCR[20:16].frpos[4:0] 指定定点数的小数点位置。 16bit 定点数采用 frpos[3:0] 指示 1~16 位的小数部分。frpos[3:0]=0 时，说明有 1b it 小数部分。frpos[3:0]=15 时，说明有 16bit 小数部分。 32bit 定点数采用 frpos[4:0] 指示 1~32 位的小数部分。frpos[4:0]=0 时，说明有 1b it 小数部分。frpos[4:0]=31 时，说明有 32bit 小数部分。
影响标志位	无
限制	寄存器范围为 vr0-vr15。
异常	ID,IO,IX

32 位指令	
操作	vrz[31:0] = (fix)vrx[31:0]; vrz[63:32] = (fix)vrx[63:32]; vrz[95:64] = (fix)vrx[95:64]; vrz[127:96] = (fix)vrz[127:96];
语法	vftox.f16.s16 vrz, vrx
数据类型	单精度
说明	将 vr x[31:0] 中的浮点数转换成定点数，保存至 vrz[31:0] 中。 将 vrx[63:32] 中的浮点数转换成定点数，保存至 vrz[63:32] 中。 将 vrx[95:64] 中的浮点数转换成定点数，保存至 vrz[95:64] 中。 将 vrx[12 7:96] 中的浮点数转换成定点数，保存至 vrz[127:96] 中。 FCR[20:16].frpos[4:0] 指定定点数的小数点位置。 16bit 定点数采用 frpos[3:0] 指示 1~16 位的小数部分。frpos[3:0]=0 时，说明有 1b it 小数部分。frpos[3:0]=15 时，说明有 16bit 小数部分。 32bit 定点数采用 frpos[4:0] 指示 1~32 位的小数部分。frpos[4:0]=0 时，说明有 1b it 小数部分。frpos[4:0]=31 时，说明有 32bit 小数部分。
影响标志位	无
限制	寄存器范围为 vr0-vr15。
异常	ID,IO,IX

32 位指令格式：

vftox.f32.s32															
31	26	25	21	20	16	15	5	4	0						
111101				00000				VRX				11011000111			
												VRZ			

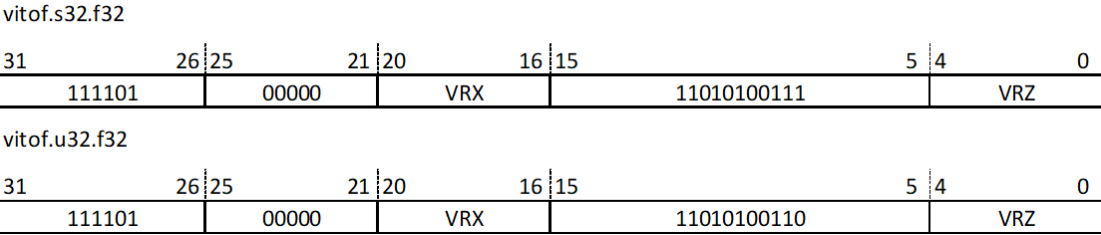
vftox.f32.u32															
31	26	25	21	20	16	15	5	4	0						
111101				00000				VRX				11011000110			
												VRZ			

15.38 VITOF.T1.T2——矢量整数浮点转换指令

统一化指令	
语法	Vitof.T1.T2 vrz, vrx (T1=s32/u32) (T2=f32)
操作	vrz[31:0]=(float32)vrx[31:0]; vrz[63:32]=(float32)vrx[63:32]; vrz[95:64]=(float32)vrx[95:64]; vrz[127:96]=(float32)vrz[127:96];
编译结果	仅存在 32 位指令。 vitof.T1.T2 vrz, vrx
数据类型	32bit 有符号整数
说明	将 vrx[31:0] 中的整数转换成浮点数，保存至 vrz[31:0] 中。 将 vrx[63:32] 中的整数转换成浮点数，保存至 vrz[63:32] 中。 将 vrx[95:64] 中的整数转换成浮点数，保存至 vrz[95:64] 中。 将 vrx[127:96] 中的整数转换成浮点数，保存至 vrz[127:96] 中。
影响标志位	无
限制	寄存器范围为 vr0-vr15。
异常	IX

32 位指令	
操作	vrz[31:0]=(float32)vrx[31:0]; vrz[63:32]=(float32)vrx[63:32]; vrz[95:64]=(float32)vrx[95:64]; vrz[127:96]=(float32)vrz[127:96];
语法	Vitof.T1.T2 vrz, vrx
数据类型	32bit 有符号整数
说明	将 vrx[31:0] 中的整数转换成浮点数，保存至 vrz[31:0] 中。 将 vrx[63:32] 中的整数转换成浮点数，保存至 vrz[63:32] 中。 将 vrx[95:64] 中的整数转换成浮点数，保存至 vrz[95:64] 中。 将 vrx[127:96] 中的整数转换成浮点数，保存至 vrz[127:96] 中。
影响标志位	无
限制	寄存器范围为 vr0-vr15。
异常	IX

32 位指令格式：



15.39 VMAX.T——矢量浮点最大值选择指令

统一化指令	
语法	vmax.T vrz,vrx,vry (T=F32)
操作	w={32}; for(i=0; i< 128/w; i++) { if(vrx[i]>=vry[i]) { vrz[i] = vrz[i]; } Else { vrz[i] = vry[i]; } } }
编译结果	仅存在 32 位指令。 vmax.T vrz, vrz,vry
数据格式:	单精度
说明	比较 VRX 元素和 VRY 元素，如果 VRX 的元素大于等于 VRY 的元素，则 VRZ 的对 应元素取 VRX 元素；否则，取 VRY 元素。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr31。
异常	IO、ID

32 位指令	
操作	<pre>w={32}; for(i=0; i< 128/w; i++) { if(vrx[i]>=vry[i]) { vrz[i] = vrx[i]; } Else { vrz[i] = vry[i]; } }</pre>
语法	vmax.T vrz, vrx,vry
数据格式:	单精度
说明	比较 VRX 元素和 VRY 元素，如果 VRX 的元素大于等于 VRY 的元素，则 VRZ 的对应元素取 VRX 元素；否则，取 VRY 元素。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr31。
异常	IO、ID

32 位指令格式:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11101		VRY		VRX		101100		01001		VRZ	

15.40 VMAXNM.T——矢量浮点规范最大值选择指令

统一化指令	
语法	vmaxnm.T vrz,vrx,vry (T=F32)
操作	W={32}; for(i=0; i< 128/w; i++) { if(vrx[i]>=vry[i]) { vrz[i] = vr[x]; } Else { vrz[i] = vr[y]; } }
编译结果	仅存在 32 位指令。 vmaxnm.T vrz, vr[x],vry
数据格式:	单精度
说明	比较 VRX 元素和 VR Y 元素，如果 VRX 的元素大于等于 VRY 的元素，则 VRZ 的对应元素取 VRX 元素；否则，取 VRY 元素。特别的，当比较的两个元素一个是 qNaN，另一个是规范数时，结果取规范数。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr31。
异常	IO、ID

32 位指令	
操作	<pre>w={32} for(i=0; i< 128/w; i++) { if(vrx[i]>=vry[i]) { vrz[i] = vrx[i]; } Else { vrz[i] = vry[i]; } }</pre>
语法	vmaxnm.T vrz, vrx,vry
数据格式:	单精度
说明	比较 VRX 元素和 VRY 元素，如果 VRX 的元素大于等于 VRY 的元素，则 VRZ 的对应元素取 VRX 元素；否则，取 VRY 元素。特别的，当比较的两个元素一个是 qNaN，另一个是规范数时，结果取规范数。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr31。
异常	IO、ID

32 位指令格式:

31	30	26	25	21	20	16	15	10	9	5	4	0				
1	11101				VRY				101100				10001		VRZ	

15.41 VMIN.T——矢量浮点最小值选择指令

统一化指令	
语法	vmin.T vrz,vrx,vry (T=F32)
操作	w={32}; for(i=0; i< 128/w; i++) { if(vrx[i]<=vry[i]) { vrz[i] = vr[x]; } Else { vrz[i] = vr[y]; } }
编译结果	仅存在 32 位指令。 vmin.T vrz, vrx,vry
数据格式:	单精度
说明	比较 VRX 元素和 VRY 元素，如果 VRX 的元素小于等于 VRY 的元素，则 VRZ 的对应元素取 VRX 元素；否则，取 VRY 元素。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr31。
异常	IO、ID

32 位指令	
操作	W={32}; for(i=0; i< 128/w; i++) { if(vrx[i]<=vry[i]) { vrz[i] = vrx[i]; } Else { vrz[i] = vry[i]; } }
语法	vmin.T vrz, vrx,vry
数据格式:	单精度
说明	比较 VRX 元素和 VRY 元素，如果 VRX 的元素小于等于 VRY 的元素，则 VRZ 的对应元素取 VRX 元素；否则，取 VRY 元素。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr31。
异常	IO、ID

32 位指令格式:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11101		VRY		VRX		101100		01011		VRZ	

15.42 VMINNM.T——矢量浮点规范最小值选择指令

统一化指令	
语法	<code>vminnm.T vrz,vrx,vry</code> (T=F32)
操作	<pre>w={32}; for(i=0; i< 128/w; i++) { if(vrx[i]<=vry[i]) { vrz[i] = vr[x]; } Else { vrz[i] = vr[y]; } }</pre>
编译结果	仅存在 32 位指令。 <code>vminnm.T vrz, vrz,vry</code>
数据格式:	单精度
说明	比较 VRX 元素和 VRY 元素，如果 VRX 的元素小于等于 VRY 的元素，则 VRZ 的对应元素取 VRX 元素；否则，取 VRY 元素。特别的，当比较的两个元素一个是 qNaN，另一个是规范数时，结果取规范数。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr31。
异常	IO、ID

32 位指令	
操作	<pre>w={32}; for(i=0; i< 128/i; i++) { if(vrx[i]<=vry[i]) { vrz[i] = vrx[i]; } Else { vrz[i] = vry[i]; } }</pre>
语法	vminnm.T vrz, vrx,vry
数据格式:	单精度
说明	比较 VRX 元素和 VRY 元素，如果 VRX 的元素小于等于 VRY 的元素，则 VRZ 的对应元素取 VRX 元素；否则，取 VRY 元素。特别的，当比较的两个元素一个是 qNaN，另一个是规范数时，结果取规范数。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr31。
异常	IO、ID

32 位指令格式:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11101		VRY		VRX		101100		10011		VRZ	

15.43 VMUL.T——矢量乘法指令

统一化指令	
语法	vmul.t vrz, vrx, vry (t=f32)
操作	vrz(i) = vrx(i) * vry(i) i=0 : 128/32 - 1
编译结果	仅存在 32 位指令 vmul.t vrz, vrx, vry
数据类型:	单精度
说明	将 vrx 与 vry 寄存器中的对应元素相乘，得到的乘积结果放入的 vrz 寄存器对应元素中。 VRX 3 2 1 0 X X X X VRZ 3 2 1 0 VRZ 3 2 1 0
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令	
操作	vrz(i) = vrx(i) * vry(i) i=0 : 128/32 - 1
语法	vmul.t vrz, vrx, vry
数据类型:	单精度
说明	将 vrx 与 vry 寄存器中的对应元素相乘，得到的乘积结果放入的 vrz 寄存器对应元素中。 VRX 3 2 1 0 X X X X VRZ 3 2 1 0 VRZ 3 2 1 0
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令格式:

vmul.f32

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11101	VRY	VRX	101000	00001	VRZ						

15.44 VMULA.T——矢量浮点乘累加指令

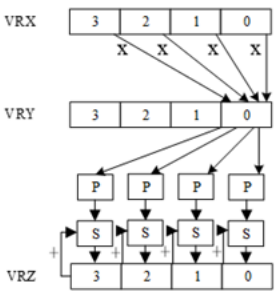
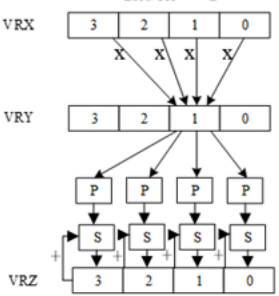
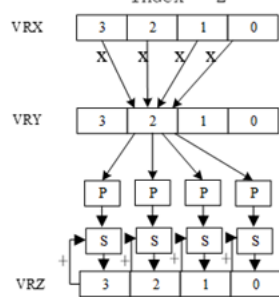
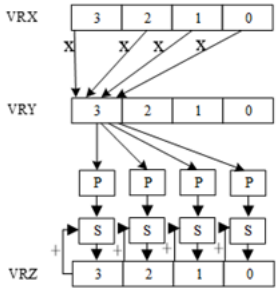
统一化指令	
语法	vmula.t vrz, vr _x , vr _y (t=f32)
操作	vrz(i) = vrz(i) + vr _x (i) * vr _y (i) i = 0 : 128/32 -1
编译结果	仅存在 32 位指令 vmula.t vrz, vr _x , vr _y
数据类型:	单精度
说明	将 vr_x 和 vr_y 中的对应元素相乘, 对得到的乘积进行舍入操作后, 再与 vrz 中对应的元素相加, 最终结果写回 vrz 中。 <pre> graph TD VRX[VRX: 3, 2, 1, 0] --> M1[x] VRY[VRY: 3, 2, 1, 0] --> M1 M1 --> A1[+] VRX --> M2[x] VRY --> M2 M2 --> A2[+] VRX --> M3[x] VRY --> M3 M3 --> A3[+] VRX --> M4[x] VRY --> M4 M4 --> A4[+] A1 --> VRZ[VRZ: 3, 2, 1, 0] A2 --> VRZ A3 --> VRZ A4 --> VRZ </pre>
影响标志位	无影响
异常	IO、ID、IX、UF、OF

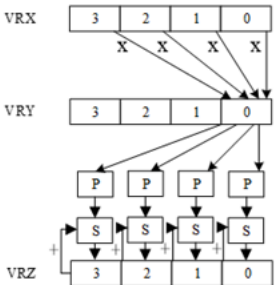
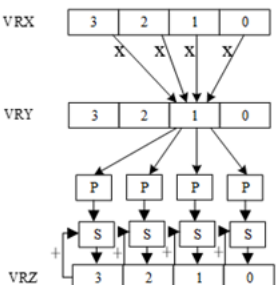
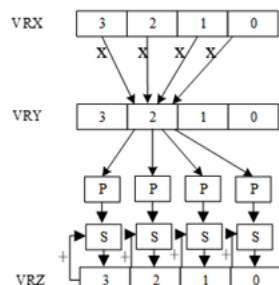
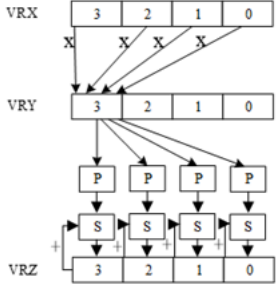
32 位指令	
操作	$vrz(i) = vrz(i) + vrx(i) * vry(i)$ $i = 0 : 128/32 - 1$
语法	vmula.t vrx, vry
数据类型:	单精度
说明	将 <code>vrx</code> 和 <code>vry</code> 中的对应元素相乘，对得到的乘积进行舍入操作后，再与 <code>vrz</code> 中对应的元素相加，最终结果写回 <code>vrz</code> 中。 <pre> graph TD VRX[VRX: 3, 2, 1, 0] --> M1[x] VRX --> M2[x] VRX --> M3[x] VRX --> M4[x] VRY[VRY: 3, 2, 1, 0] --> M1 VRY --> M2 VRY --> M3 VRY --> M4 M1 --> A1[+] M2 --> A2[+] M3 --> A3[+] M4 --> A4[+] VRZ[VRZ: 3, 2, 1, 0] --> A1 VRZ --> A2 VRZ --> A3 VRZ --> A4 A1 --> VRZ A2 --> VRZ A3 --> VRZ A4 --> VRZ </pre>
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令格式：

vmula.f32												
31	30	26	25	21	20	16	15	10	9	5	4	0
1	11101			VRY		VRX		101000		00011		VRZ

15.45 VMULAI.T——矢量浮点索引乘累加指令

统一化指令	
语法	vmulai.t vrz, vrx, vry, index (t=f32)
操作	$vrz(i) = vrz(i) + vrx(i) * vry(index)$ $i = 0 : 128/32 - 1$ $index = 0 \sim 128/32 - 1$
编译结果	仅存在 32 位指令 vmulai.t vrz, vrx, vry,index
数据类型:	单精度
说明	将 vrx 的各元素与 vry 中由 index 选定的元素相乘，对得到的乘积进行舍入操作后，再与 vrz 中对应的元素相加，最终结果写回 vrz 中。 index = 0  index = 1  index = 2  index = 3 
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令	
操作	$vrz(i) = vrz(i) + vrz(i) * vry(index)$ $i = 0 : 128/32 - 1$ $index = 0 \sim 128/32 - 1$
语法	<code>vmulai.t vrz, vrz, vry, index</code>
数据类型:	单精度
说明	将 <code>vrz</code> 的各元素与 <code>vry</code> 中由 <code>index</code> 选定的元素相乘，对得到的乘积进行舍入操作后，再与 <code>vrz</code> 中对应的元素相加，最终结果写回 <code>vrz</code> 中。 index = 0  index = 1  index = 2  index = 3 
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令格式:

`vmulai.f32`

31	30	26	25	21	20	16	15	9	8	6	5	4	0
1	11101	VRY	VRX	1000001	IMM3	1	VRZ						

注: IMM3 的取值为 3' b000~3' b110

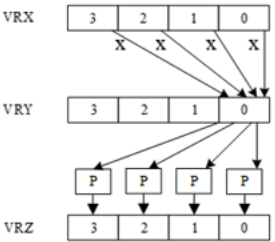
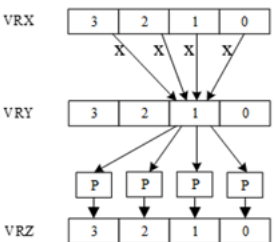
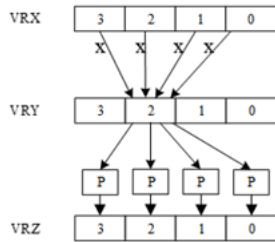
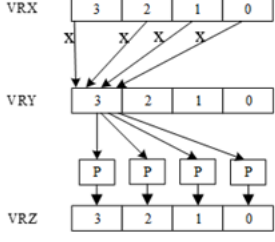
index = 0, IMM3 = 3' b000

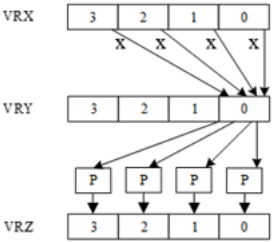
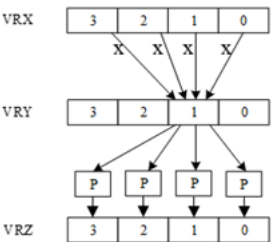
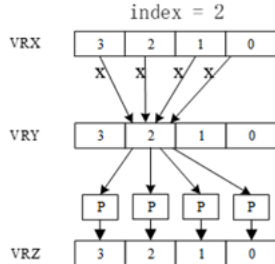
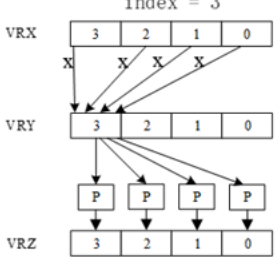
index = 1, IMM3 = 3' b010

index = 2, IMM3 = 3' b100

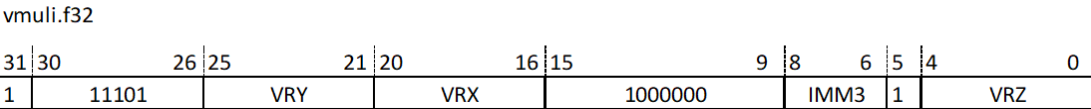
index = 3, IMM3 = 3' b110

15.46 VMULI.T——矢量索引乘法指令

统一化指令	
语法	vmuli.t vrz, vrx, vry, index (t=f32)
操作	vrz(i) = vrx(i) * vry(index) i=0 : 128/32 - 1 index = 0 ~ 128/32 - 1
编译结果	仅存在 32 位指令 vmuli.t vrz, vrx, vry, index
数据类型:	单精度
说明	将 vrx 的各元素与 index 选定的 vry 中的元素相乘，将乘积写入 vrz 的对应元素中。 index = 0  index = 1  index = 2  index = 3 
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令	
操作	$vrz(i) = vrx(i) * vry(index)$ $i=0 : 128/32 - 1$ $index = 0 \sim 128/32 - 1$
语法	<code>vmuli.t vrz, vrx, vry, index</code>
数据类型:	单精度
说明	将 <code>vrx</code> 的各元素与 <code>index</code> 选定的 <code>vry</code> 中的元素相乘，将乘积写入 <code>vrz</code> 的对应元素中。 index = 0  index = 1  index = 2  index = 3 
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令格式：



注：IMM3 的取值为 3' b000~3' b110

index = 0, IMM3 = 3' b000

index = 1, IMM3 = 3' b010

index = 2, IMM3 = 3' b100

index = 3, IMM3 = 3' b110

15.47 VMULS.T——矢量浮点乘取负累加指令

统一化指令	
语法	vmuls.t vrz, vrx, vry (t=f32)
操作	vrz(i) = vrz(i) - vrx(i) * vry(i) i = 0 : 128/32 -1
编译结果	仅存在 32 位指令 vmuls.t vrz, vrx, vry
数据类型:	单精度
说明	将 vrx 和 vry 中的对应元素相乘, 对得到的乘积进行舍入操作并取负后, 再与 vrz 中对应的元素相加, 最终结果写回 vrz 中。 <pre> graph TD VRX[VRX: 3, 2, 1, 0] VRY[VRY: 3, 2, 1, 0] VRZ[VRZ: 3, 2, 1, 0] subgraph Path3 V3[3] --> M3[x] V3 --> R3[取负] V3 --> A3[+] V3 --> R3 V3 --> A3 end subgraph Path2 V2[2] --> M2[x] V2 --> R2[取负] V2 --> A2[+] V2 --> R2 V2 --> A2 end subgraph Path1 V1[1] --> M1[x] V1 --> R1[取负] V1 --> A1[+] V1 --> R1 V1 --> A1 end subgraph Path0 V0[0] --> M0[x] V0 --> R0[取负] V0 --> A0[+] V0 --> R0 V0 --> A0 end VRX --> V3 VRX --> V2 VRX --> V1 VRX --> V0 VRY --> M3 VRY --> M2 VRY --> M1 VRY --> M0 VRZ --> A3 VRZ --> A2 VRZ --> A1 VRZ --> A0 A3 --> VRZ A2 --> VRZ A1 --> VRZ A0 --> VRZ </pre>
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令																									
操作	$vrz(i) = vrz(i) - vrz(i) * vry(i)$ $i = 0 : 128/32 - 1$																								
语法	<code>vmuls.t vrz, vrz, vry</code>																								
数据类型:	单精度																								
说明	将 <code>vrz</code> 和 <code>vry</code> 中的对应元素相乘，对得到的乘积进行舍入操作并取负后，再与 <code>vrz</code> 中对应的元素相加，最终结果写回 <code>vrz</code> 中。 VRX <table><tr><td>3</td><td>2</td><td>1</td><td>0</td></tr></table> VRY <table><tr><td>3</td><td>2</td><td>1</td><td>0</td></tr></table> x <table><tr><td>x</td><td>x</td><td>x</td><td>x</td></tr></table> 取负 <table><tr><td>取负</td><td>取负</td><td>取负</td><td>取负</td></tr></table> + <table><tr><td>+</td><td>+</td><td>+</td><td>+</td></tr></table> VRZ <table><tr><td>3</td><td>2</td><td>1</td><td>0</td></tr></table>	3	2	1	0	3	2	1	0	x	x	x	x	取负	取负	取负	取负	+	+	+	+	3	2	1	0
3	2	1	0																						
3	2	1	0																						
x	x	x	x																						
取负	取负	取负	取负																						
+	+	+	+																						
3	2	1	0																						
影响标志位	无影响																								
异常	IO、ID、IX、UF、OF																								

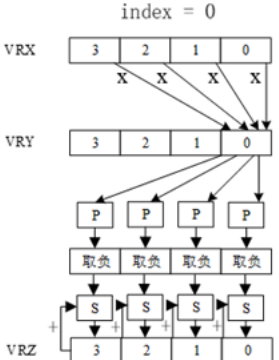
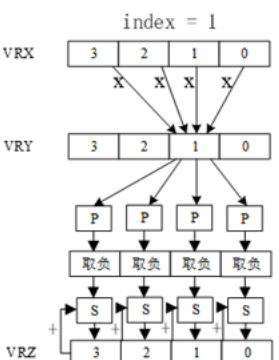
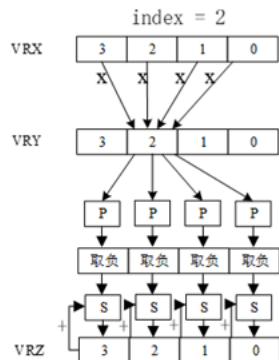
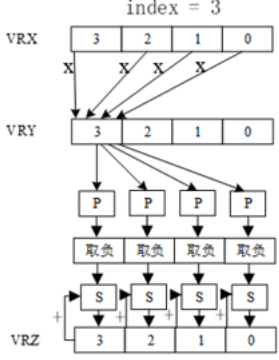
32 位指令格式:

vmuls.f32

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11101	VRY	VRX	101000	00101	VRZ						

15.48 VMULSI.T——矢量浮点索引乘取负累加指令

统一化指令	
语法	vmulsi.t vrz, vrx, vry,index (t=f32)
操作	$vrz(i) = vrz(i) - vrx(i) * vry(index)$ $i = 0 : 128/32 - 1$ $index = 0 \sim 128/32 - 1$
编译结果	仅存在 32 位指令 vmulsi.t vrz, vrx, vry, index
数据类型:	单精度
说明	将 vrx 的各元素与 vry 中由 index 选定的元素相乘，对得到的乘积进行舍入操作并取负后，再与 vrz 中对应的元素相加，最终结果写回 vrz 中。 index = 0 index = 1 index = 2 index = 3
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令	
操作	$vrz(i) = vrz(i) - vrz(i) * vry(index)$ $i = 0 : 128/32 - 1$ $index = 0 \sim 128/32 - 1$
语法	<code>vmulsi.t vrz, vrz, vry, index</code>
数据类型:	单精度
说明	将 <code>vrz</code> 的各元素与 <code>vry</code> 中由 <code>index</code> 选定的元素相乘，对得到的乘积进行舍入操作并取负后，再与 <code>vrz</code> 中对应的元素相加，最终结果写回 <code>vrz</code> 中。 index = 0  index = 1  index = 2  index = 3 
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令格式:

vmulsi.f32

31	30	26	25	21	20	16	15	9	8	6	5	4	0
1	11101	VRY	VRX	1000010	IMM3	1	VRZ						

注: IMM3 的取值为 3' b000~3' b110

index = 0, IMM3 = 3' b000

index = 1, IMM3 = 3' b010

index = 2, IMM3 = 3' b100

index = 3, IMM3 = 3' b110

15.49 VNEG.T——矢量浮点取反指令

统一化指令	
语法	vneg.T vrz, vrx (T=f32)
操作	vrz[31:0]=- vrx[31:0]; vrz[63:32]=-vrx[63:32]; vrz[95:64]= -vrx[95:64]; vrz[127:96]= -vrz[127:96];
编译结果	仅存在 32 位指令。 vneg.T vrz, vrx
数据类型	单精度
说明	将 vrx[31:0] 中的浮点数取反，保存至 vrz[31:0] 中。 将 vrx[63:32] 中的浮点数取反，保存至 vrz[63:32] 中。 将 vrx[95:64] 中的浮点数取反，保存至 vrz[95:64] 中。 将 vrx[127:96] 中的浮点数取反，保存至 vrz[127:96] 中。
影响标志位	无
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	vrz[31:0]=- vrx[31:0]; vrz[63:32]=-vrx[63:32]; vrz[95:64]= -vrx[95:64]; vrz[127:96]= -vrz[127:96];
语法	vneg.T vrz, vrx
数据类型	单精度
说明	将 vrx[31:0] 中的浮点数取反，保存至 vrz[31:0] 中。 将 vrx[63:32] 中的浮点数取反，保存至 vrz[63:32] 中。 将 vrx[95:64] 中的浮点数取反，保存至 vrz[95:64] 中。 将 vrx[127:96] 中的浮点数取反，保存至 vrz[127:96] 中。
影响标志位	无
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

vneg.f32

31	26	25	24	21	20	16	15	5	4	0
111101			0	0000		VRX		10101010011		VRZ

15.50 VPADD.T——矢量浮点耦合加法指令

统一化指令	
语法	vpadd.T vrz, vrx,vry (T=f32)
操作	vrz[31:0]= vrx[31:0]+vrx[63:32]; vrz[63:32]= vrx[95:64]+vrx[127:96]; vrz[95:64]= vry[31:0]+vry[63:32]; vrz[127:96]= vry[95:64]+vry[127:96];
编译结果	仅存在 32 位指令。 vpadd.T vrz, vrx,vry
数据类型	单精度
说明	将 vrx[31 :0] 和 vrx[63:32] 中的浮点数相加，保存至 vrz[31:0] 中。 将 vrx[95:64] 和 vrx[127:96] 中的浮点数相加，保存至 vrz[63:32] 中。 将 vry[31: 0] 和 vry[63:32] 中的浮点数相加，保存至 vrz[95:64] 中。 将 vry[95:64] 和 vry[127:96] 中的浮点数相加，保存至 vrz[127:96] 中。
影响标志位	无
限制	寄存器范围为 vr0-vr15 中。
异常	ID,IO,IX,UF,OF

32 位指令	
操作	vrz[31:0]= vrx[31:0]+vrx[63:32]; vrz[63:32]= vrx[95:64]+vrx[127:96]; vrz[95:64]= vry[31:0]+vry[63:32]; vrz[127:96]= vry[95:64]+vry[127:96];
语法	vadd.f16 vrz, vrx,vry
数据类型	单精度
说明	将 vrx[31 :0] 和 vrx[63:32] 中的浮点数相加，保存至 vrz[31:0] 中。 将 vrx[95:64] 和 vrx[127:96] 中的浮点数相加，保存至 vrz[63:32] 中。 将 vry[31: 0] 和 vry[63:32] 中的浮点数相加，保存至 vrz[95:64] 中。 将 vry[95:64] 和 vry[127:96] 中的浮点数相加，保存至 vrz[127:96] 中。
影响标志位	无
限制	寄存器范围为 vr0-vr15 中。
异常	ID,IO,IX,UF,OF

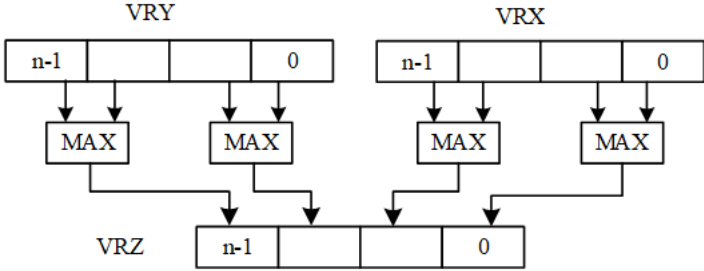
32 位指令格式：

vpadd.f32

31	26	25	21	20	16	15	5	4	0
111101			VRY		VRX		10101000101		VRZ

15.51 VPMAX.T——矢量浮点相邻最大值选择指令

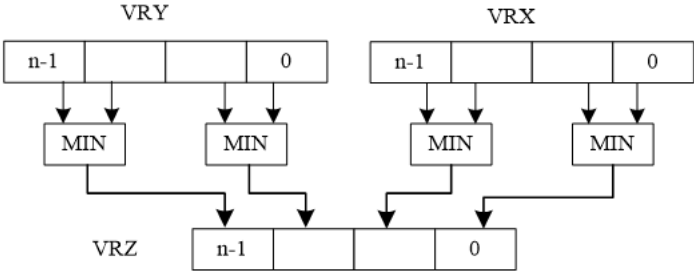
统一化指令	
语法	vpmax.T vrz,vrx,vry (T=F32)
操作	w={32}; for(i=0;i<128/2w;i++) { if(vrx[2i]>=vrx[2i+1]) vrz[i] = vrx[2i]; else vrz[i] = vrx[2i+1]; if(vry[2i]>=vry[2i+1]) vrz[i+128/2w] = vry[2i]; else vrz[i+128/2w] = vry[2i+1]; }
编译结果	仅存在 32 位指令。 vpmax.T vrz, vrx,vry
数据类型:	单精度
说明	选择 VRX 和 VRY 相邻元素的较大值存入 VRZ 的对应元素中。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr31。
异常	IO、ID

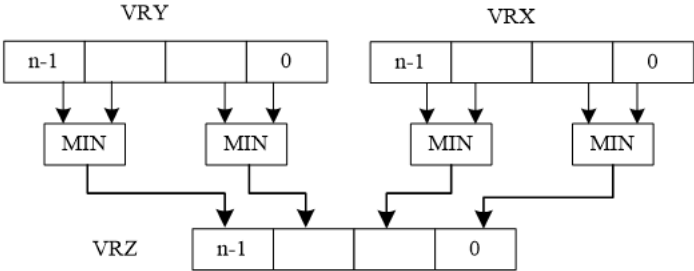
32 位指令	
操作	<pre>w={32}; for(i=0;i<128/2w;i++) { if(vrx[2i]>=vrx[2i+1]) vrz[i] = vrx[2i]; else vrz[i] = vrx[2i+1]; if(vry[2i]>=vry[2i+1]) vrz[i+128/2w] = vry[2i]; else vrz[i+128/2w] = vry[2i+1]; }</pre>
语法	vpmax.T vrz, vrx,vry
数据类型	单精度
说明	选择 VRX 和 VRY 相邻元素的较大值存入 VRZ 的对应元素中。  The diagram shows two input registers, VRY and VRX, each with elements indexed from n-1 to 0. Each element is compared with its neighbor using a MAX operation. The results of these comparisons are then stored in the VRZ register, also indexed from n-1 to 0.
影响标志位	无影响
限制	寄存器的范围为 vr0-vr31。
异常	IO、ID

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11101	VRY	VRX	101100	11001	VRZ						

15.52 VPMIN.T——矢量浮点相邻最小值选择指令

统一化指令	
语法	vpmin.T vrz,vrx,vry (T=F32)
操作	w={32}; for(i=0;i<128/2w;i++) { if(vrx[2i]>=vrx[2i+1]) vrz[i] = vrx[2i+1]; else vrz[i] = vrx[2i]; if(vry[2i]>=vry[2i+1]) vrz[i+128/2w] = vry[2i+1]; else vrz[i+128/2w] = vry[2i]; }
编译结果	仅存在 32 位指令。 vpmin.T vrz, vrx,vry
数据格式:	单精度
说明	选择 VRX 和 VRY 相邻元素的较小值存入 VRZ 的对应元素中。 
影响标志位	无影响
限制	寄存器的范围为 vr0-vr31。
异常	IO、ID

32 位指令	
操作	<pre>w={32}; for(i=0;i<128/2w;i++) { if(vrx[2i]>=vrx[2i+1]) vrz[i] = vrx[2i+1]; else vrz[i] = vrx[2i]; if(vry[2i]>=vry[2i+1]) vrz[i+128/2w] = vry[2i+1]; else vrz[i+128/2w] = vry[2i]; }</pre>
语法	vpmin.T vrz, vrx,vry
数据格式:	单精度
说明	选择 VRX 和 VRY 相邻元素的较小值存入 VRZ 的对应元素中。 
影响标志位	无影响
限制	寄存器的范围为 vr0-vr31。
异常	IO、ID

32 位指令格式:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11101		VRY		VRX		101100		11011		VRZ	

15.53 VRECPE.T——矢量倒数指令

统一化指令	
语法	vrecpe.T vrz, vrx (T=F32)
操作	Vz(i) 1/(Vx(i)) i=0:(number-1) (快速计算 Vx(i) 的倒数值) 快速算法为: Vz(i) = MAGIC -Vx(i) (将所有操作数看做有符号整数来进行运算) MAGIC=0x7EF0A3D7
编译结果	仅存在 32 位指令。 vrecpe.T vrz, vrx
数据类型	F32
说明	对于浮点倒数运算，采用快速算法：Vz= MAGIC -Vx
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Vz(i) 1/(Vx(i)) i=0:(number-1) (快速计算 Vx(i) 的倒数值) 快速算法为: Vz(i) = MAGIC -Vx(i) (将所有操作数看做有符号整数来进行运算) MAGIC=0x7EF0A3D7
语法	vrecpe.T vrz, vrx
数据类型	F32
说明	对于浮点倒数运算，采用快速算法：Vz= MAGIC -Vx
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	26	25	21	20	16	15	5	4	0
111101			X		VRX		10111000001		VRZ

15.54 VRECPS.T——矢量浮点倒数逼近指令

统一化指令	
语法	vrecps.t vrz, vrx, vry (t=f32)
操作	$vrz(i) = 2 - vrx(i) * vry(i)$ $i=0 : 128/32 - 1$
编译结果	仅存在 32 位指令 vrecps.t vrz, vrx, vry
数据类型:	单精度
说明	将 vrx 与 vry 寄存器中的对应元素相乘，得到的乘积结果保留全部精度并进行取负操作后，加上 2，最终的结果放入 vrz 对应的元素中。
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令	
操作	$vrz(i) = 2 - vrx(i) * vry(i)$ $i=0 : 128/32 - 1$
语法	vrecps.t vrz, vrx, vry
数据类型:	单精度
说明	将 vrx 与 vry 寄存器中的对应元素相乘，得到的乘积结果保留全部精度并进行取负操作后，加上 2，最终的结果放入 vrz 对应的元素中。
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	11101		VRY		VRX		101110		00011		VRZ	

15.55 VRSQRTE.T——矢量浮点倒数开方指令

统一化指令	
语法	vrsqrte.t vrz, vrx (t=f32)
操作	$V_z(i) = 1/((V_x(i))^{(1/2)})$ (快速计算 $V_x(i)$ 的倒数开方值) 快速算法为: $V_z(i) = \text{MAGIC} - V_x(i) \gg 1$ (将所有操作数看做有符号整数来进行运算, 右移为带符号移位) $\text{MAGIC}=0x5F37642F$
编译结果	仅存在 32 位指令 vrsqrte.t vrz, vrx
数据类型:	单精度
说明	对于浮点倒数开方运算, 采用快速算法: $V_z = \text{MAGIC} - V_x \gg 1$
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令	
操作	$V_z(i) = 1/((V_x(i))^{(1/2)})$ (快速计算 $V_x(i)$ 的倒数开方值) 快速算法为: $V_z(i) = \text{MAGIC} - V_x(i) \gg 1$ (将所有操作数看做有符号整数来进行运算, 右移为带符号移位) $\text{MAGIC}=0x5F37642F$
语法	vrsqrte.t vrz, vrx, vry
数据类型:	单精度
说明	对于浮点倒数开方运算, 采用快速算法: $V_z = \text{MAGIC} - V_x \gg 1$
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令格式:

vrsqrte.f32															
31	30	26	25	21	20	16	15	10	9	5	4	0			
1	11101			X		VRX		101110			10110		VRZ		

15.56 VRSQRTS.T——矢量浮点开方逼近指令

统一化指令	
语法	vrsqrts.t vrz, vrx, vry (t=f32)
操作	$vrz(i) = (3 - vrz(i) * vry(i))/2$ $i=0 : 128/32 - 1$
编译结果	仅存在 32 位指令 vrsqrts.t vrz, vrx, vry
数据类型:	单精度
说明	将 vrx 与 vry 寄存器中的对应元素相乘，得到的乘积结果保留全部精度并进行取负操作后，指数部分左移 1 位，再加上 1.5，最终的结果放入 vrz 对应的元素中。
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令	
操作	$vrz(i) = (3 - vrz(i) * vry(i)) / 2$ $i=0 : 128/32 - 1$
语法	vrsqrts.t vrz, vrx, vry
数据类型:	单精度
说明	将 vrx 与 vry 寄存器中的对应元素相乘，得到的乘积结果保留全部精度并进行取负操作后，指数部分左移 1 位，再加上 1.5，最终的结果放入 vrz 对应的元素中。
影响标志位	无影响
异常	IO、ID、IX、UF、OF

32 位指令格式：

vrsqrts.f32

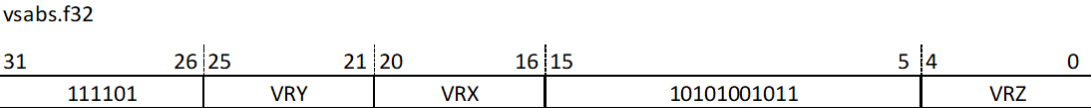
31	30	26	25	21	20	16	15	10	9	5	4	0
1		11101		VRY		VRX		101110		00111		VRZ

15.57 VSABS.T——矢量浮点减法绝对值指令

统一化指令	
语法	vsabs.T vrz, vrx,vry (T=f32)
操作	vrz[31:0]= vrx[31:0]-vry[31:0] ; vrz[63:32]= vrx[63:32]-vry[63:32] ; vrz[95:64]= vrx[95:64]-vry[95:64] ; vrz[127:96]= vrx[127:96]-vry[127:96] ;
编译结果	仅存在 32 位指令。 vsabs.T vrz, vrx,vry
数据类型	单精度
说明	将 vrx[31:0] 和 vry[31:0] 中的浮点数相减后取绝对值，保存至 vrz[31:0] 中。 将 vrx[63:32] 和 vry[63 :32] 中的浮点数相减后取绝对值，保存至 vrz[63:32] 中。 将 vrx[95:64] 和 vry[95 :64] 中的浮点数相减后取绝对值，保存至 vrz[95:64] 中。 将 vrx[127:96] 和 vry[127: 96] 中的浮点数相减后取绝对值，保存至 vrz[127:96] 中。
影响标志位	无
限制	寄存器范围为 vr0-vr15 中。
异常	ID,IO,IX,UF,OF

32 位指令	
操作	vrz[31:0]= vrx[31:0]-vry[31:0] vrz[63:32]= vrx[63:32]-vry[63:32] vrz[95:64]= vrx[95:64]-vry[95:64] vrz[127:96]= vrx[127:96]-vry[127:96] ;
语法	vsabs.fl6 vrz, vrx,vry
数据类型	单精度
说明	将 vrx[31:0] 和 vry[31:0] 中的浮点数相减后取绝对值，保存至 vrz[31:0] 中。 将 vrx[63:32] 和 vry[63 :32] 中的浮点数相减后取绝对值，保存至 vrz[63:32] 中。 将 vrx[95:64] 和 vry[95 :64] 中的浮点数相减后取绝对值，保存至 vrz[95:64] 中。 将 vrx[127:96] 和 vry[127: 96] 中的浮点数相减后取绝对值，保存至 vrz[127:96] 中。
影响标志位	无
限制	寄存器范围为 vr0-vr15 中。
异常	ID,IO,IX,UF,OF

32 位指令格式：



15.58 VSAX.T——矢量浮点交叉减加法指令

统一化指令	
语法	vsax.T vrz, vrx,vry (T=f32)
操作	vrz[31:0]= vrx[31:0]+vry[63:32]; vrz[63:32]= vrx[63:32]-vry[31:0]; vrz[95:64]= vrx[95:64]+vry[127:96]; vrz[127:96]= vrx[127:96]-vry[95:64];
编译结果	仅存在 32 位指令。 vsax.T vrz, vrx,vry
数据类型	单精度
说明	将 vrx[31 :0] 和 vry[63:32] 中的浮点数相加，保存至 vrz[31:0] 中。 将 vrx[63: 32] 和 vry[31:0] 中的浮点数相减，保存至 vrz[63:32] 中。 将 vrx[95:64] 和 vry[127:96] 中的浮点数相加，保存至 vrz[95:64] 中。 将 vrx[127:96] 和 vry[95:64] 中的浮点数相减，保存至 vrz[127:96] 中。
影响标志位	无
限制	寄存器范围为 vr0-vr15 中。
异常	ID,IO,IX,UF,OF

32 位指令	
操作	vrz[31:0]= vrx[31:0]+vry[63:32]; vrz[63:32]= vrx[63:32]-vry[31:0]; vrz[95:64]= vrx[95:64]+vry[127:96]; vrz[127:96]= vrx[127:96]-vry[95:64];
语法	vsax.T vrz, vrx,vry
数据类型	单精度
说明	将 vrx[31 :0] 和 vry[63:32] 中的浮点数相加，保存至 vrz[31:0] 中。 将 vrx[63: 32] 和 vry[31:0] 中的浮点数相减，保存至 vrz[63:32] 中。 将 vrx[95:64] 和 vry[127:96] 中的浮点数相加，保存至 vrz[95:64] 中。 将 vrx[127:96] 和 vry[95:64] 中的浮点数相减，保存至 vrz[127:96] 中。
影响标志位	无
限制	寄存器范围为 vr0-vr15 中。
异常	ID,IO,IX,UF,OF

32 位指令格式：

vsax.f32

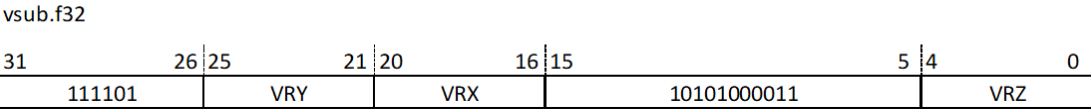
31	26	25	21	20	16	15	5	4	0
111101		VRY		VRX		10101001001		VRZ	

15.59 VSUB.T——矢量单精度浮点减法指令

统一化指令	
语法	vsub.T vrz, vrz,vry (T=f32)
操作	vrz[31:0]= vrz[31:0]-vry[31:0]; vrz[63:32]= vrz[63:32]-vry[63:32]; vrz[95:64]= vrz[95:64]-vry[95:64]; vrz[127:96]= vrz[127:96]-vry[127:96];
编译结果	仅存在 32 位指令。 vsub.T vrz, vrz,vry
数据类型	单精度
说明	将 vrz[3 1:0] 和 vry[31:0] 中的浮点数相减，保存至 vrz[31:0] 中。 将 vrz[63:3 2] 和 vry[63:32] 中的浮点数相减，保存至 vrz[63:32] 中。 将 vrz[95:6 4] 和 vry[95:64] 中的浮点数相减，保存至 vrz[95:64] 中。 将 vrz[127:9 6] 和 vry[127:96] 中浮点数相减，保存至 vrz[127:96] 中。
影响标志位	无
限制	寄存器范围为 vr0-vr15 中。
异常	ID,IO,IX,UF,OF

32 位指令	
操作	vrz[31:0]= vrz[31:0]-vry[31:0]; vrz[63:32]= vrz[63:32]-vry[63:32]; vrz[95:64]= vrz[95:64]-vry[95:64]; vrz[127:96]= vrz[127:96]-vry[127:96];
语法	vsub.T vrz, vrz,vry
数据类型	单精度
说明	将 vrz[3 1:0] 和 vry[31:0] 中的浮点数相减，保存至 vrz[31:0] 中。 将 vrz[63:3 2] 和 vry[63:32] 中的浮点数相减，保存至 vrz[63:32] 中。 将 vrz[95:6 4] 和 vry[95:64] 中的浮点数相减，保存至 vrz[95:64] 中。 将 vrz[127:9 6] 和 vry[127:96] 中浮点数相减，保存至 vrz[127:96] 中。
影响标志位	无
限制	寄存器范围为 vr0-vr15 中。
异常	ID,IO,IX,UF,OF

32 位指令格式：



15.60 VXTOF.T1.T2——矢量定点浮点转换指令

统一化指令	
语法	vxtof.T1.T2 vrz, vrx (T1=s32/u32) (T2=f32)
操作	vrz[31:0]=(float16/32)vrx[31:0]; vrz[63:32]=(float16/32)vrx[63:32]; vrz[95:64]=(float16/32)vrx[95:64]; vrz[127:96]=(float16/32)vrz[127:96];
编译结果	仅存在 32 位指令。 vxtof.T1.T2 vrz, vrx
数据类型	32bit 有符号定点数
说明	将 vr x[31:0] 中的定点数转换成浮点数，保存至 vrz[31:0] 中。 将 vrx[63:32] 中的定点数转换成浮点数，保存至 vrz[63:32] 中。 将 vrx[95:64] 中的定点数转换成浮点数，保存至 vrz[95:64] 中。 将 vrx[12 7:96] 中的定点数转换成浮点数，保存至 vrz[127:96] 中。 FCR[20:16].frpos[4:0] 指定定点数的小数点位置。 16bit 定点数采用 frpos[3:0] 指示 1~16 位的小数部分。frpos[3:0]=0 时，说明有 1b it 小数部分。frpos[3:0]=15 时，说明有 16bit 小数部分。 32bit 定点数采用 frpos[4:0] 指示 1~32 位的小数部分。frpos[4:0]=0 时，说明有 1b it 小数部分。frpos[4:0]=31 时，说明有 32bit 小数部分。
影响标志位	无
限制	寄存器范围为 vr0-vr15。
异常	ID,IO,IX

32 位指令	
操作	vrz[31:0]=(float16/32)vrx[31:0]; vrz[63:32]=(float16/32)vrx[63:32]; vrz[95:64]=(float16/32)vrx[95:64]; vrz[127:96]=(float16/32)vrz[127:96];
语法	vxtof.T1.T2 vrz, vrx
数据类型	32bit 有符号定点数
说明	将 vr x[31:0] 中的定点数转换成浮点数，保存至 vrz[31:0] 中。 将 vrx[63:32] 中的定点数转换成浮点数，保存至 vrz[63:32] 中。 将 vrx[95:64] 中的定点数转换成浮点数，保存至 vrz[95:64] 中。 将 vrx[12 7:96] 中的定点数转换成浮点数，保存至 vrz[127:96] 中。 FCR[20:16].frpos[4:0] 指定定点数的小数点位置。 16bit 定点数采用 frpos[3:0] 指示 1~16 位的小数部分。frpos[3:0]=0 时，说明有 1b it 小数部分。frpos[3:0]=15 时，说明有 16bit 小数部分。 32bit 定点数采用 frpos[4:0] 指示 1~32 位的小数部分。frpos[4:0]=0 时，说明有 1b it 小数部分。frpos[4:0]=31 时，说明有 32bit 小数部分。
影响标志位	无
限制	寄存器范围为 vr0-vr15。
异常	ID,IO,IX

32 位指令格式：

vxtof.s32.f32									
31	26	25	21	20	16	15	5	4	0
111101		00000			VRX		11011001111		VRZ
vxtof.u32.f32									
31	26	25	21	20	16	15	5	4	0
111101		00000			VRX		11011001110		VRZ

第十六章 附录 C 矢量多媒体指令术语表

以下是 C860 矢量多媒体指令的具体描述，下面根据指令英文字母顺序对每条指令进行详细说明。

表 16.1: 指令术语表中特殊运算说明表

运算符号	操作说明	示例
Lo[]	取低一半结果	a=0x7f; b=0x7f; c=a*b=0x3f01; d=Lo[c]=0x01.
High[]	取高一半结果	a=0x7f; b=0x7f; c=a*b=0x3f01; d=High[c]=0x3f.
Extend[]	根据符号位扩展成位宽的两倍	a=0x7f; b=0x8000; If(unsigned) { c=Extend[a] =0x007f; d=Extend[b] =0x00008000; } Else { c=Extend[a] =0x007f; d=Extend[b] =0xffff8000; }

16.1 VABS.T——矢量取绝对值指令

统一化指令	
语法	vabs.T vrz, vrx (T=S8/S16/S32)
操作	size = {8,16,32}; Number=128/size for(i=0; i< Number; i++) { Vz(i)=abs(Vx(i)); }
编译结果	仅存在 32 位指令。 vabs.T vrz, vrx
数据类型	S8/S16/S32
说明	将源矢量寄存器 VRX 的每个有符号元素求绝对值后，结果存放于目的矢量寄存器 VRZ 的对应元素中。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr15
异常	无

32 位指令	
操作	size = {8,16,32}; Number=128/size for(i=0; i< Number; i++) { Vz(i)=abs(Vx(i)); }
语法	vabs.T vrz, vrx
数据类型	S8/S16/S32
说明	将源矢量寄存器 VRX 的每个有符号元素求绝对值后，结果存放于目的矢量寄存器 VRZ 的对应元素中。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr15
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1	0		w0	VRX				00010000000			
											1	VRZ	

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

编码表中 VRX,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.2 VABS.T.S——矢量饱和取绝对值指令

统一化指令	
语法	vabs.T.s vrz, vrx (T=S8/S16/S32)
操作	size = {8,16,32}; Number=128/size for(i=0; i< Number; i++) { If Vx(i)=-2^(size-1) Vz(i)= 2^(size-1)-1; Else Vz(i)=abs(Vx(i)); }
编译结果	仅存在 32 位指令。 vabs.T.s vrz, vrx
数据类型	S8/S16/S32
说明	将源矢量寄存器 VRX 的每个有符号元素求绝对值并饱和后，结果存放于目的矢量寄存器 VRZ 的对应元素中。 VRX n-1 0 abs abs abs abs sat sat sat sat n-1 0 VRZ
影响标志位	无影响
限制	寄存器的范围为 vr0-vr15
异常	无

32 位指令	
操作	<pre>size = {8,16,32}; Number=128/size for(i=0; i< Number; i++) { If Vx(i)=-2^(size-1) Vz(i)= 2^(size-1)-1; Else Vz(i)=abs(Vx(i)); }</pre>
语法	vabs.T.s vrz, vrx
数据类型	S8/S16/S32
说明	将源矢量寄存器 VRX 的每个有符号元素求绝对值并饱和后，结果存放于目的矢量寄存器 VRZ 的对应元素中。 VRX VRZ
影响标志位	无影响
限制	寄存器的范围为 vr0-vr15
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1	0		w0	VRX		00010000010			1	VRZ	

注：

编码表中 w1，w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

编码表中 VRX,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.3 VADD.T——矢量加法指令

统一化指令	
语法	vadd.T vrz, vrx, vry (T=S8/S16/S32/S64/U8/U16/U32/U64)
操作	<pre> size = {8, 16, 32, 64}; Number=128/size; for(i=0; i< Number; i++) { Vz(i)=Vx(i)+Vy(i); } </pre>
编译结果	仅存在 32 位指令。 vadd.T vrz, vrx, vry
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	将 VRX 与 VRY 的对应元素相加，结果存放于 VRZ 中。 <pre> VRX n-1 0 VRY n-1 0 +-----+-----+ + + + + +-----+-----+ VRZ n-1 0 </pre>
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	<pre> size = {8, 16, 32, 64}; Number=128/size; for(i=0; i< Number; i++) { Vz(i)=Vx(i)+Vy(i); } </pre>
语法:	vadd.T vrz, vrx, vry
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	将 VRX 与 VRY 的对应元素相加，结果存放于 VRZ 中。 <pre> graph TD VRX[n-1 0] --> P1[+] VRX --> P2[+] VRX --> P3[+] VRX --> P4[+] VRY[n-1 0] --> P1 VRY --> P2 VRY --> P3 VRY --> P4 P1 --> VRZ[n-1 0] P2 --> VRZ P3 --> VRZ P4 --> VRZ </pre>
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式:

31	op	26	25	24	21	20	19	16	15			5	4	3	0	
111110		w1		VRy		w0		VRx		00000000000				S	VRz	

注：

编码表中 w1, w0 用来表征指令位宽类型, 其中:

$[w1, w0] = 2'$ b00 表示 size 为 8bit 指令;

$[w1, w0] = 2'$ b01 表示 size 为 16bit 指令;

$[w1, w0] = 2'$ b10 表示 size 为 32bit 指令;

$[w1, w0] = 2'$ b11 表示 size 为 64bit 指令;

编码表中 S 表征带符号运算还是无符号运算: 1 表示带符号, 0 表示无符号;

编码表中 VRX, VRY, VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.4 VADD.T.E——矢量扩展加法指令

统一化指令	
语法	vadd.T.e vrz, vrx, vry (T= S8/S16/S32 /U8/U16/U32)
操作	size = {8,16,32}; Number=128/size for(i=0; i< Nmuber/2; i++) { Vz(2i+1:2i)=extend(Vx(i))+extend(Vy(i)) Vz+1(2i+12i)=extend(Vx(number/2+i)) + extend(Vy(number/2+i)); } extend 根据 U/S 将值零扩展或者符号扩展至元素位宽的 2 倍
编译结果	仅存在 32 位指令。 vadd.T.e vrz, vrx, vry
数据类型	U8/U16/U32 S8/S16/S32
说明	将 VRX 与 VRY 的对应元素扩展，并相加，运算结果存放于 VRZ 中。 VRX n-1 0 VRX n-1 0 VRX' n-1 0 VRX' n-1 0 + + + + VRZ n-1 0
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32}; Number=128/size for(i=0; i< Nmuber/2; i++) { Vz(2i+1:2i)=extend(Vx(i))+extend(Vy(i)) Vz+1(2i+12i)=extend(Vx(number/2+i))+extend(Vy(number/2+i)); } extend 根据 U/S 将值零扩展或者符号扩展至元素位宽的 2 倍 i=0:number/2-1
语法	vadd.T.e vrz, vrx, vry
数据类型	U8/U16/U32 S8/S16/S32
说明	将 VRX 与 VRY 的对应元素扩展，并相加，运算结果存放于 VRZ 中。 VRX VRX VRX VRX VRX VRX VRX VRX VRX VRX VRX VRX VRX VRX VRX VRX VRX VRX VRX VRX VRX VRX VRX VRX VRX VRX VRX VRX VRX VRX VRX VRX VRX VRX VRX VRX

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
	111110	w1		VRY	w0		VRX		00000110000	S		VRZ	

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

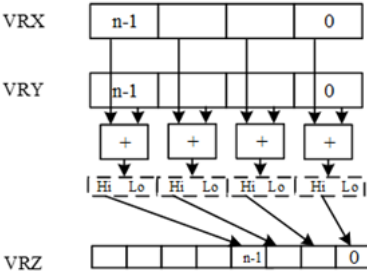
[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

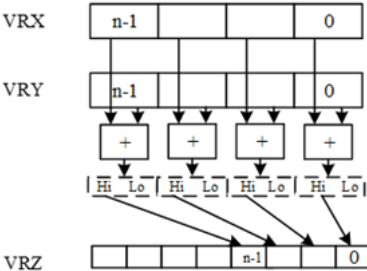
[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.5 VADD.T.H——矢量加法取高半指令

统一化指令	
语法	vadd.T.h vrz, vrx, vry (T=S16/S32/S64 /U16/U32/U64)
操作	size = {16,32,64}; Number=128/size for(i=0; i< Number; i++) { Tmp(i)=(Vx(i)+Vy(i))[size-1:size/2]; i=0:(number-1) } for(i=0; i< Number/2; i++) { Vz(i)={Tmp(2i+1), Tmp(2i)}; }
编译结果	仅存在 32 位指令。 vadd.T.h vrz, vrx, vry
数据类型	U16/U32/U64 S16/S32/S64
说明	VRX 与 VRY 的对应元素相加，然后取加法结果的高半部分结果按序放至目的寄存器 VRZ 的低半部分。 
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {16,32,64}; Number=128/size for(i=0; i< Number; i++) { Tmp(i)=(Vx(i)+Vy(i))[size-1:size/2]; i=0:(number-1) } for(i=0; i< Number/2; i++) { Vz(i)={Tmp(2i+1), Tmp(2i)}; }
语法	vadd.T.h vrz, vrx, vry
数据类型	U16/U32/U64 S16/S32/S64
说明	VRX 与 VRY 的对应元素相加，然后取加法结果的高半部分按序放至目的寄存器 VRZ 的低半部分。 
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

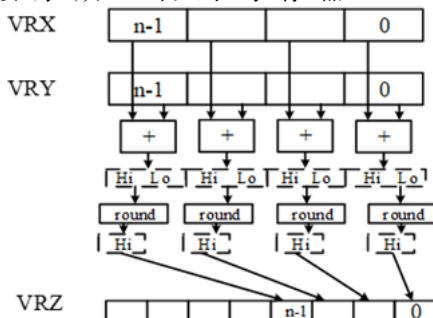
31	op	26	25	24	21	20	19	16	15	5	4	3	0
	111110	w1		VRY	w0		VRX		00000000011	S		VRZ	

注：

- 编码表中 w1, w0 用来表征指令位宽类型，其中：
- [w1,w0] = 2’ b01 表示 size 为 16bit 指令；
- [w1,w0] = 2’ b10 表示 size 为 32bit 指令；
- [w1,w0] = 2’ b11 表示 size 为 64bit 指令；
- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.6 VADD.T.RH——矢量加法取高半指令

统一化指令	
语法	vadd.T.rh vrz, vrx, vry (T=S16/S32/S64/ U16/U32/U64)
操作	size = {16,32,64}; Number=128/size round=1<<(size/2-1); for(i=0; i< Number; i++) { Tmp(i)=(Vx(i)+Vy(i)+round)[size-1:size/2]; } for(i=0; i< Number/2; i++) { Vz(i)={Tmp(2i+1),Tmp(2i)}; }
编译结果	仅存在 32 位指令。 vadd.T.rh vrz, vrx, vry
数据类型	U16/U32/U64 S16/S32/S64
说明	VRX 与 VRY 的对应元素相加，然后取加法结果的高半部分，舍入后按序放至目的寄存器 VRZ 的低半部分。 VRX VRY + Hi Lo Hi Lo Hi Lo Hi Lo round Hi VRZ
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	<pre>size = {16,32,64}; Number=128/size round=1<<(size/2-1); for(i=0; i< Number; i++) { Tmp(i)=(Vx(i)+Vy(i)+round) [size-1:size/2]; } for(i=0; i< Number/2; i++) { Vz(i)={Tmp(2i+1),Tmp(2i)}; }</pre>
语法	vadd.T.rh vrz, vrx, vry
数据类型	U16/U32/U64 S16/S32/S64
说明	VRX 与 VRY 的对应元素相加，然后取加法结果的高半部分，舍入后按序放至目的寄存器 VRZ 的低半部分。 
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0	
111110		w1	VRY			w0	VRX			00000000100			S	VRZ

注：

- 编码表中 w1, w0 用来表征指令位宽类型，其中：
- [w1,w0] = 2’ b01 表示 size 为 16bit 指令；
 - [w1,w0] = 2’ b10 表示 size 为 32bit 指令；
 - [w1,w0] = 2’ b11 表示 size 为 64bit 指令；
- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号;

16.7 VADD.T.S——矢量加法饱和指令

统一化指令	
语法	vadd.T.s vrz, vrx, vry (T=S8/S16/S32/S64/U8/U16/U32/U64)
操作	<pre> size = {8, 16, 32, 64}; Number=128/size signed=(T==S); (根据元素 U/S 类型) Max=signed? 2^(size-1)-1: 2^(size)-1; Min=signed? -2^(size-1):0; for(i=0; i< Number; i++) { If Vx(i)+Vy(i)>Max Vz(i)=Max; Else if Vx(i)+Vy(i)<Min Vz(i)=Min; Else Vz(i)= Vx(i)+Vy(i); } </pre>
编译结果	仅存在 32 位指令。 vadd.T.s vrz, vrx, vry
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	将 VRX 与 VRY 的对应元素相加，并对加法结果做饱和运算，运算结果存放于 VRZ 中。其中，当加法结果大于目的元素可表示的最大值时，取目的元素最大值作为饱和运算结果，当加法结果小于目的元素可表示的最小值时，取目的元素最小值作为饱和运算结果，当加法结果处于目的元素可表示范围时，取加法结果作为饱和运算值。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8, 16, 32, 64}; Number=128/size signed=(T==S); (根据元素 U/S 类型选择) Max=signed? 2^(size-1)-1: 2^(size)-1; Min=signed? -2^(size-1):0; for(i=0; i< Number; i++) { If Vx(i)+Vy(i)>Max Vz(i)=Max; Else if Vx(i)+Vy(i)<Min Vz(i)=Min; Else Vz(i)= Vx(i)+Vy(i); }
语法	vadd.T.s vrz, vrx, vry
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	将 VRX 与 VRY 的对应元素相加，并对加法结果做饱和运算，运算结果存放于 VRZ 中。其中，当加法结果大于目的元素可表示的最大值时，取目的元素最大值作为饱和运算结果，当加法结果小于目的元素可表示的最小值时，取目的元素最小值作为饱和运算结果，当加法结果处于目的元素可表示范围时，取加法结果作为饱和运算值。 VRX n-1 0 VRY n-1 0 + + + + sat sat sat sat VRZ n-1 0
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1		VRY		w0		VRX		0000000001		S	VRZ

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

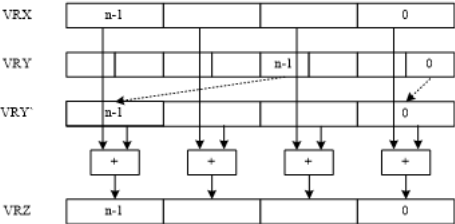
[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

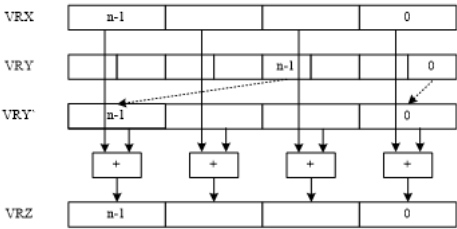
[w1,w0] = 2’ b11 表示 size 为 64bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX, VRY, VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.8 VADD.T.X——矢量非对齐加法指令

统一化指令	
语法	vadd.T.x vrz, vrx, vry (T=S8/S16/S32 /U8/U16/U32)
操作	size = {8,16,32}; Number=128/size for(i=0; i< Number/2; i++) { Vz(2i+1:2i)=Vx(2i+1:2i)+extend(Vy(i)); Vz+1(2i+1:2i)=Vx+1(2i+1:2i) + extend(Vy(number/2+i)); } extend 根据 U/S 将值零扩展或者符号扩展至元素位宽的 2 倍
编译结果	仅存在 32 位指令。 vadd.T.x vrz, vrx, vry
数据类型	U8/U16/U32 S8/S16/S32
说明	将 VRY 的元素扩展，并与 VRX 的对应元素相加，运算结果存放于 VRZ 中。 
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32}; Number=128/size for(i=0; i< Number/2; i++) { Vz(2i+1:2i)=Vx(2i+1:2i)+extend(Vy(i)); V z+1(2i+1:2i)=Vx+1(2i+1:2i)+extend(Vy(number/2+i)); } extend 根据 U/S 将值零扩展或者符号扩展至元素位宽的 2 倍
语法	vadd.T.x vrz, vrx, vry
数据类型	U8/U16/U32 S8/S16/S32
说明	将 VRY 的元素扩展，并与 VRX 的对应元素相加，运算结果存放于 VRZ 中。 
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15		5	4	3	0
111110			w1	VRY		w0	VRX		0000000010			S	VRZ	

注：

- 编码表中 w1, w0 用来表征指令位宽类型，其中：
- [w1,w0] = 2’ b00 表示 size 为 8bit 指令；
- [w1,w0] = 2’ b01 表示 size 为 16bit 指令；
- [w1,w0] = 2’ b10 表示 size 为 32bit 指令；
- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.9 VADDH.T——矢量加法平均指令

统一化指令	
语法	vaddh.T vrz, vrx, vry (T=S8/S16/S32 /U8/U16/U32)
操作	size = {8,16,32}; Number=128/size for(i=0; i< Number; i++) { Vz(i)=(Vx(i)+Vy(i))>>1; } 对于 U, 右移为逻辑右移, 对于 S, 右移为算术右移
编译结果	仅存在 32 位指令。 vaddh.T vrz, vrx, vry
数据类型	U8/U16/U32 S8/S16/S32
说明	VRX 与 VRY 的对应元素相加, 然后对加法结果取平均值, 结果存放于 VRZ 中。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32}; Number=128/size for(i=0; i< Number; i++) { Vz(i)=(Vx(i)+Vy(i))>>1; } 对于 U, 右移为逻辑右移, 对于 S, 右移为算术右移
语法	vaddh.T vrz, vrx, vry
数据类型	U8/U16/U32 S8/S16/S32
说明	VRX 与 VRY 的对应元素相加, 然后对加法结果取平均值, 结果存放于 VRZ 中。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式:

31	op	26	25	24	21	20	19	16	15		5	4	3	0
	111110	w1		VRY	w0		VRX		00000000101		S		VRZ	

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.10 VADDH.T.R——矢量加法平均并舍入指令

统一化指令	
语法	vaddh.T.r vrz, vrx, vry (T=S8/S16/S32 /U8/U16/U32)
操作	size = {8,16,32}; Number=128/size for(i=0; i< Number; i++) { Vz(i)=(Vx(i)+Vy(i)+1)>>1; } 对于 U, 右移为逻辑右移，对于 S, 右移为算术右移
编译结果	仅存在 32 位指令。 vaddh.T.r vrz, vrx, vry
数据类型	U8/U16/U32 S8/S16/S32
说明	VRX 与 VRY 的对应元素相加，然后对加法结果取平均值并做舍入操作，舍入结果存放于 VRZ 中。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32}; Number=128/size for(i=0; i< Number; i++) { Vz(i)=(Vx(i)+Vy(i)+1)>>1; } 对于 U, 右移为逻辑右移, 对于 S, 右移为算术右移
语法	vaddh.T.r vrz, vrx, vry
数据类型	U8/U16/U32 S8/S16/S32
说明	VRX 与 VRY 的对应元素相加, 然后对加法结果取平均值并做舍入操作, 舍入结果存放于 VRZ 中。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式:

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1	VRY		w0	VRX		00000001101			S	VRZ	

注:

编码表中 w1, w0 用来表征指令位宽类型, 其中:

[w1,w0] = 2' b00 表示 size 为 8bit 指令;

[w1,w0] = 2' b01 表示 size 为 16bit 指令;

[w1,w0] = 2' b10 表示 size 为 32bit 指令;

编码表中 S 表征带符号运算还是无符号运算: 1 表示带符号, 0 表示无符号;

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号;

16.11 VAND.T——矢量与指令

统一化指令	
语法	vand.T vrz,vrx,vry (T=8/16/32/64)
操作	for j=0:127 Vz[j]=Vx[j] & Vy[j] vrz[j],vry[j],vrz[j] 指操作数比特位 j 的数据;
编译结果	仅存在 32 位指令。 vand.T vrz, vrz,vry
数据类型	8/16/32/64
说明	将 Vrx,Vry 相应比特位做与操作并写入 Vrz 相应的比特位之中。 注：T 在此并无实际意义，仅为用户使用方便。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	for j=0:127 Vz[j]=Vx[j] & Vy[j] vrz[j],vry[j],vrz[j] 指相应比特位的数据;
语法	vand.T vrz, vrz,vry
数据类型	8/16/32/64
说明	将 Vrx,Vry 相应比特位做与操作并写入 Vrz 相应的比特位之中。 注：T 在此并无实际意义，仅为用户使用方便。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15					4	3	0
	111110	x		VRY	x		VRX						000110000000			VRZ

注：

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号

16.12 VANDN.T——矢量非与指令

统一化指令	
语法	vandn.T vrz,vrx,vry (T=8/16/32/64)
操作	for j=0:127 Vz[j]=Vx[j] & !Vy[j] vrz[j],vry[j],vrz[j] 指操作数比特位 j 的数据;
编译结果	仅存在 32 位指令。 vandn.T vrz, vrz,vry
数据类型	8/16/32/64
说明	将 Vry 数据取反与 Vrx 相应比特位做与操作并写入 Vrz 相应的比特位之中。注：T 在此并无实际意义，仅为用户使用方便。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	for j=0:127 Vz[j]=Vx[j] & !Vy[j] vrz[j],vry[j],vrz[j] 指操作数比特位 j 的数据;
语法	vandn.T vrz, vrz,vry
数据类型	8/16/32/64
说明	将 Vry 数据取反与 Vrx 相应比特位做与操作并写入 Vrz 相应的比特位之中。注：T 在此并无实际意义，仅为用户使用方便。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

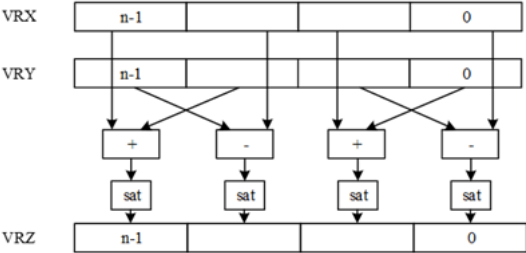
31	op	26	25	24	21	20	19	16	15		4	3	0
	111110	x		VRY	x		VRX			000110000001			VRZ

注：

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号

16.13 VASX.T.S——矢量错位加减指令

统一化指令	
语法	vasx.T.s vrz, vrx (T=S8/S16/S32 /U8/U16/U32)
操作	size = {8,16,32}; Number=128/size signed=(T==S); (根据元素 U/S 类型选择) Max=signed? 2^{^(size-1)-1}: 2^{^(size)-1}; Min=signed? -2^{^(size-1)}:0; for(i=0; i< Number; i++) { If (Vx(2i+1) +Vy(2i))>Max Vz(2i+1)=Max; Else if (Vx(2i+1) +Vy(2i))<Min Vz(2i+1)=Min; Else Vz(2i+1) = Vx(2i+1)+Vy(2i); If (Vx(2i)-Vy(2i+1))>Max Vz(2i)=Max; Else if (Vx(2i)-Vy(2i+1))<Min Vz(2i)=Min; Else Vz(2i)= Vx(2i)-Vy(2i+1); }
编译结果	仅存在 32 位指令。 vasx.T.s vrz, vrx
数据类型	U8/U16/U32 S8/S16/S32
说明	VRX 中的奇数元素与 VRZ 中的偶数元素相加，并对加法结果做饱和运算，运算结果放入 VRZ 的奇数元素中；VRX 中的偶数元素与 VRZ 中的奇数元素相减，并对减法结果做饱和运算，运算结果放入 VRZ 的偶数元素中。其中，当运算结果大于目的元素可表示的最大值时，取目的元素最大值作为饱和运算结果，当运算结果小于目的元素可表示的最小值时，取目的元素最小值作为饱和运算结果，当运算结果处于目的元素可表示范围时，取运算结果作为饱和运算值。
影响	不影响
标志位	
寄存器	寄存器基因为 0, 15

32 位指令	
操作	size = {8,16,32}; Number=128/size signed=(T==S); (根据元素 U/S 类型选择) Max=signed? 2^(size-1)-1: 2^(size)-1; Min=signed? -2^(size-1):0; for(i=0; i< Number; i++) { If (Vx(2i+1) +Vy(2i))>Max Vz(2i+1)=Max; Else if (Vx(2i+1) +Vy(2i))<Min Vz(2i+1)=Min; Else Vz(2i+1) = Vx(2i+1)+Vy(2i); If (Vx(2i)-Vy(2i+1))>Max Vz(2i)=Max; Else if (Vx(2i)-Vy(2i+1))<Min Vz(2i)=Min; Else Vz(2i)= Vx(2i)-Vy(2i+1); } }
语法	vasx.T.s vrz, vrx
数据类型	U8/U16/U32 S8/S16/S32
说明	VRX 中的奇数元素与 VRY 中的偶数元素相加，并对加法结果做饱和运算，运算结果放入 VRZ 的奇数元素中；VRX 中的偶数元素与 VRY 中的奇数元素相减，并对减法结果做饱和运算，运算结果放入 VRZ 的偶数元素中。其中，当运算结果大于目的元素可表示的最大值时，取目的元素最大值作为饱和运算结果，当运算结果小于目的元素可表示的最小值时，取目的元素最小值作为饱和运算结果，当运算结果处于目的元素可表示范围时，取运算结果作为饱和运算值。 
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0			
111110		w1		VRY		w0		VRX		00000010000				S	VRZ	

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

$[w1,w0] = 2'b00$ 表示 size 为 8bit 指令；

$[w1,w0] = 2'b01$ 表示 size 为 16bit 指令；

$[w1,w0] = 2'b10$ 表示 size 为 32bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.14 VASXH.T——矢量错位加减平均指令

统一化指令	
语法	vasxh.T vrz, vrx (T=S8/S16/S32 /U8/U16/U32)
操作	size = {8,16,32}; Number=128/size for(i=0; i< Number; i++) { Vz(2i+1)=(Vx(2i+1)+Vy(2i))>>1; Vz(2i)=(Vx(2i)-Vy(2i+1))>>1; } 对于 U, 右移为逻辑右移, 对于 S, 右移为算术右移。
编译结果	仅存在 32 位指令。 vasxh.T vrz, vrx
数据类型	U8/U16/U32 S8/S16/S32
说明	VRX 中的奇数元素与 VRY 中的偶数元素相加, 然后对加法结果取平均值, 运算结果放入 VRZ 的奇数元素中; VRX 中的偶数元素与 VRY 中的奇数元素相减, 然后对减法结果取平均值, 运算结果放入 VRZ 的偶数元素中。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32}; Number=128/size for(i=0; i< Number; i++) { Vz(2i+1)=(Vx(2i+1)+Vy(2i))>>1; Vz(2i)=(Vx(2i)-Vy(2i+1))>>1; } 对于 U, 右移为逻辑右移, 对于 S, 右移为算术右移
语法	vasxh.T vrz, vrx
数据类型	U8/U16/U32 S8/S16/S32
说明	VRX 中的奇数元素与 VRY 中的偶数元素相加, 然后对加法结果取平均值, 运算结果放入 VRZ 的奇数元素中; VRX 中的偶数元素与 VRY 中的奇数元素相减, 然后对减法结果取平均值, 运算结果放入 VRZ 的偶数元素中。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式:

31	op		26	25	24	21	20	19	16	15				5	4	3	0
111110			w1		VRY		w0		VRX		00000010010			S		VRZ	

注:

- 编码表中 w1, w0 用来表征指令位宽类型, 其中:
- [w1,w0] = 2' b00 表示 size 为 8bit 指令;
- [w1,w0] = 2' b01 表示 size 为 16bit 指令;
- [w1,w0] = 2' b10 表示 size 为 32bit 指令;
- 编码表中 S 表征带符号运算还是无符号运算: 1 表示带符号, 0 表示无符号;
- 编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号;

16.15 VCLIP.T——矢量裁剪饱和指令

统一化指令	
语法	vclip.T vrz,vrx,imm6 (T=U8/U16/U32/U64/S8/S16/S32/S64)
操作	size = {8,16,32,64}; number = 128/size; U:Max=2^(imm6)-1, Min=0; S:Max=2^(oimm6-1)-1, Min=-2^(oimm6-1); 无论 T 为 U/S, 将 Vx(i) 始终看做有符号数 If Vx(i)>Max Vz(i)=Max; else if Vx(i)<Min Vz(i)=Min; else Vz(i)=Vx(i); i=0:number-1 imm6 的范围是 0 ~ (size-1) oimm6 的范围是 1 ~ (size)
编译结果	仅存在 32 位指令。 vclip.T vrz, vrx,imm6
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	用立即数截取 vrz 中的数据并判断截取后数据是否饱和，进行饱和处理。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32,64}; number = 128/size; U:Max=2^(imm6)-1, Min=0; S: Max=2^(oimm6-1)-1, Min=-2^(oimm6-1); 无论 T 为 U/S, 将 Vx(i) 始终看做有符号数 If Vx(i)>Max Vz(i)=Max; else if Vx(i)<Min Vz(i)=Min; else Vz(i)=Vx(i); i=0:number-1 imm6 的范围是 0 ~ (size-1) oimm6 的范围是 1 ~ (size)
语法	vclip.T vrz, vrx,imm6
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	用立即数截取 vrx 中的数据的高低位并判断截取后数据是否饱和，进行饱和处理。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	20	19	16	15	6	5	4	3	0
111110		IMM6			VRX		0001001111			I	S	VRZ

注：

编码表中 Imm6,I 组合在一起用来表征指令位宽类型和 Index，其中：

IMM[6:0]={IMM6,I}; if(IMM[6:4]=010) size=8

IMM[3:0] 表示移位范围 0 到 7(高位为 0); if(IMM[6:4]=011) size=16

IMM[3:0] 表示移位范围 0 到 15; if(IMM[6:4]=001) size=32

IMM[3:0] 表示移位范围 0 到 15; if(IMM[6:4]=100) size=32

IMM[3:0] 表示移位范围 16 到 31; if(IMM[6:4]=000) size=64

IMM[3:0] 表示移位范围 0 到 15; if(IMM[6:4]=111) size=64

IMM[3:0] 表示移位范围 16 到 31; if(IMM[6:4]=110) size=64

IMM[3:0] 表示移位范围 32 到 47; if(IMM[6:4]=101) size=64

IMM[3:0] 表示移位范围 48 到 63;

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.16 VCLS.T——矢量符号位连续相同指令

统一化指令	
语法	vcls.T vrz,vrx (T=S8/S16/S32/S64)
操作	size = {8,16,32,64}; number = 128/size; 从 MSB 开始与元素符号位相同的连续位数，符号位不计入计数 Vz(i)=count_leading_sign_bit(Vx(i)); i=0:(number-1) vrx[i],vrz[i] 指相应 element 的数据;
编译结果	仅存在 32 位指令。 vcls.T vrz, vrx
数据类型	S8/S16/S32/S64
说明	从 MSB 开始与元素符号位相同的连续位数，符号位不计入计数
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32,64}; number = 128/size; 从 MSB 开始与元素符号位相同的连续位数，符号位不计入计数 Vz(i)=count_leading_sign_bit(Vx(i)); i=0:(number-1) vrx[i],vrz[i] 指相应 element 的数据;
语法	vcls.T vrz, vrx
数据类型	S8/S16/S32/S64
说明	从 MSB 开始与元素符号位相同的连续位数，符号位不计入计数
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式:

31	op	26	25	24	21	20	19	16	15		4	3	0
111110		x	VR _Y		x	VR _X		000110001100				VR _Z	

注：

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号

16.17 VCLZ.T——矢量最高位连续 0 个数指令

统一化指令	
语法	vclz.T vrz,vrx (T=8/16/32/64)
操作	size = {8,16,32,64}; number = 128/size; 从 MSB 开始连续为 0 位数 Vz(i)=count_leading_zero(Vx(i)); i=0:(number-1) vrz[i],vrz[i] 指相应 element 的数据;
编译结果	仅存在 32 位指令。 vclz.T vrz, vrz
数据类型	U8/U16/U32/U64
说明	从 MSB 开始连续为 0 的位数
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32,64}; number = 128/size; 从 MSB 开始与元素符号位相同的连续位数，符号位不计入计数 Vz(i)=count_leading_zero(Vx(i)); i=0:(number-1) vrz[i],vrz[i] 指相应 element 的数据;
语法	vclz.T vrz, vrz
数据类型	U8/U16/U32/U64
说明	从 MSB 开始连续为 0 的位数
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	4	3	0
111110		x	VRY		x	VRX		000110001110			VRZ	

注：

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号

16.18 VCMPHS.T——矢量大于等于比较指令

统一化指令	
语法	<code>vcmp_{hs}.T vrz, vr_x, vr_y</code> (T=S8/S16/S32/ U8/U16/U32)
操作	<code>size = {8, 16, 32};</code> <code>Number = 128 / size;</code> <code>for(i=0; i< Number; i++)</code> <code>{</code> <code>if(V_x(i) >= V_y(i))</code> <code>{</code> <code>V_z(i) = AllOnes;</code> <code>}</code> <code>Else</code> <code>{</code> <code>V_z(i) = AllZeros;</code> <code>}</code> <code>}</code>
编译结果	仅存在 32 位指令。 <code>vcmp_{hs}.T vrz, vr_x, vr_y</code>
数据类型	U8/U16/U32 S8/S16/S32
说明	比较 VRX 和 VRY 对应元素，如果 VRX 大于等于 VRY，则将 VRZ 的对应元素置全 1； 否则，置全 0。
影响标志位	不影响
限制	寄存器的范围为 vr0-vr15。
异常	无

32 位指令	
操作	<pre>size = {8, 16, 32}; Number = 128 / size; for(i=0; i< Number; i++) { if(Vx(i) >= Vy(i)) { Vz(i) = AllOnes; } Else { Vz(i) = AllZeros; } }</pre>
语法	vcmps.T vrz, vrx, vry
数据类型	U8/U16/U32 S8/S16/S32
说明	比较 VRX 和 VRY 对应元素，如果 VRX 大于等于 VRY，则将 VRZ 的对应元素置全 1；否则，置全 0。
影响标志位	不影响
限制	寄存器的范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1		VRY		w0		VRX		00000100000		S	VRZ

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.19 VCMPHSZ.T——矢量大于等于 0 比较指令

统一化指令	
语法	vcmphsz.T vrz, vrx (T=S8/S16/S32)
操作	size = {8, 16, 32}; Number = 128 / size; for(i=0; i< Number; i++) { if(Vx(i) >= 0) { Vz(i) = AllOnes; } Else { Vz(i) = AllZeros; } }
编译结果	仅存在 32 位指令。 vcmphsz.T vrz, vrx
数据类型	S8/S16/S32
说明	比较 VRX 元素和 0，如果 VRX 的元素大于等于 0 则将 VRZ 的对应元素，置全 1；否则，置全 0。
影响标志位	不影响
限制	寄存器的范围为 vr0-vr15。
异常	无

32 位指令	
操作	<pre>size = {8, 16, 32}; Number = 128 / size; for(i=0; i< Number; i++) { if(Vx(i) >= 0) { Vz(i) = AllOnes; } Else { Vz(i) = AllZeros; } }</pre>
语法	vcmphsz.T vrz, vrx
数据类型	S8/S16/S32
说明	比较 VRX 元素和 0，如果 VRX 的元素大于等于 0 则将 VRZ 的对应元素，置全 1；否则，置全 0。
影响标志位	不影响
限制	寄存器的范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op		26	25	24	21	20	19	16	15											5	4	3	0
111110			w1		0		w0		VRX		00000111100										1		VRZ	

注：

- 编码表中 w1, w0 用来表征指令位宽类型，其中：
- [w1,w0] = 2’ b00 表示 size 为 8bit 指令；
 - [w1,w0] = 2’ b01 表示 size 为 16bit 指令；
 - [w1,w0] = 2’ b10 表示 size 为 32bit 指令；
- 编码表中 VRX,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.20 VCMPHZ.T——矢量大于零比较指令

统一化指令	
语法	vcmphz.T vrz, vrx (T=S8/S16/S32)
操作	size = {8, 16, 32}; Number = 128 / size; for(i=0; i< Number; i++) { if(Vx(i) > 0) { Vz(i) = AllOnes; } Else { Vz(i) = AllZeros; } }
编译结果	仅存在 32 位指令。 vcmphz.T vrz, vrx
数据类型	S8/S16/S32
说明	将 VRX 的每个元素元素与 0 比较，如果 VRX 大于 0，则将 VRZ 的对应元素置全 1；否则，置全 0。
影响标志位	不影响
限制	寄存器的范围为 vr0-vr15。
异常	无

32 位指令	
操作	<pre>size = {8, 16, 32}; Number = 128 / size; for(i=0; i< Number; i++) { if(Vx(i) > 0) { Vz(i) = AllOnes; } Else { Vz(i) = AllZeros; } }</pre>
语法	vcmphz.T vrz, vrx
数据类型	S8/S16/S32
说明	将 VRX 的每个元素元素与 0 比较，如果 VRX 大于 0，则将 VRZ 的对应元素置全 1；否则，置全 0。
影响标志位	不影响
限制	寄存器的范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1	0		w0	VRX				0000011110			
											1	VRZ	

注：

- 编码表中 w1, w0 用来表征指令位宽类型，其中：
- [w1,w0] = 2’ b00 表示 size 为 8bit 指令；
 - [w1,w0] = 2’ b01 表示 size 为 16bit 指令；
 - [w1,w0] = 2’ b10 表示 size 为 32bit 指令；
- 编码表中 VRX,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.21 VCMPLSZ.T——矢量小于等于零比较指令

统一化指令	
语法	vcmplsz.T vrz, vrx (T=S8/S16/S32)
操作	size = {8, 16, 32}; Number = 128 / size; for(i=0; i< Number; i++) { if(Vx(i) <= 0) { Vz(i) = AllOnes; } Else { Vz(i) = AllZeros; } }
编译结果	仅存在 32 位指令。 vcmplsz.T vrz, vrx
数据类型	S8/S16/S32
说明	将 VRX 的每个元素元素与 0 比较，如果 VRX 小于等于 0，则将 VRZ 的对应元素置全 1；否则，置全 0。
影响标志位	不影响
限制	寄存器的范围为 vr0-vr15。
异常	无

32 位指令	
操作	<pre>size = {8, 16, 32}; Number = 128 / size; for(i=0; i< Number; i++) { if(Vx(i) <= 0) { Vz(i) = AllOnes; } Else { Vz(i) = AllZeros; } }</pre>
语法	vcmplsz.T vrz, vrx
数据类型	U8/U16/U32 S8/S16/S32
说明	将 VRX 的每个元素元素与 0 比较，如果 VRX 小于等于 0，则将 VRZ 的对应元素置全 1；否则，置全 0。
影响标志位	不影响
限制	寄存器的范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15		5	4	3	0
111110		w1	0		w0	VRX			0000011111				1	VRZ

注：编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

编码表中 VRX,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.22 VCMPLT.T——矢量小于比较指令

统一化指令	
语法	<code>vcmplt.T vrz, vrx, vry</code> (T=S8/S16/S32 /U8/U16/U32)
操作	<code>size = {8, 16, 32}; Number = 128 / size; for(i=0; i< Number; i++) { if(Vx(i) < Vy(i)) { Vz(i) = AllOnes; } Else { Vz(i) = AllZeros; } }</code>
编译结果	仅存在 32 位指令。 <code>vcmplt.T vrz, vrx, vry</code>
数据类型	U8/U16/U32 S8/S16/S32
说明	比较 VRX 和 VRY 的对应元素，如果 VRX 的元素小于 VRY 的对应元素，则将 VRZ 的对应元素置全 1；否则，置全 0。
影响标志位	不影响
限制	寄存器的范围为 vr0-vr15。
异常	无

32 位指令	
操作	<pre>size = {8, 16, 32}; Number = 128 / size; for(i=0; i< Number; i++) { if(Vx(i) < Vy(i)) { Vz(i) = AllOnes; } Else { Vz(i) = AllZeros; } }</pre>
语法	vcmlt.T vrz, vrx, vry
数据类型	U8/U16/U32 S8/S16/S32
说明	比较 VRX 和 VRY 的对应元素，如果 VRX 的元素小于 VRY 的对应元素，则将 VRZ 的对应元素置全 1；否则，置全 0。
影响标志位	不影响
限制	寄存器的范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1		VRY		w0		VRX		00000100001		S	VRZ

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.23 VCMPLTZ.T——矢量小于 0 比较指令

统一化指令	
语法	vcmpltz.T vrz, vrx (T=S8/S16/S32)
操作	size = {8, 16, 32}; Number = 128 / size; for(i=0; i< Number; i++) { if(Vx(i) < 0) { Vz(i) = AllOnes; } Else { Vz(i) = AllZeros; } }
编译结果	仅存在 32 位指令。 vcmpltz.T vrz, vrx
数据类型	S8/S16/S32
说明	比较 VRX 元素和 0，如果 VRX 的元素小于 0，则将 VRZ 的对应元素置全 1；否则，置全 0。
影响标志位	不影响
限制	寄存器的范围为 vr0-vr15。
异常	无

32 位指令	
操作	<pre>size = {8, 16, 32}; Number = 128 / size; for(i=0; i< Number; i++) { if(Vx(i) < 0) { Vz(i) = AllOnes; } Else { Vz(i) = AllZeros; } }</pre>
语法	vcmpltz.T vrz, vrx
数据类型	S8/S16/S32
说明	比较 VRX 元素和 0，如果 VRX 的元素小于 0，则将 VRZ 的对应元素置全 1；否则，置全 0。
影响标志位	不影响
限制	寄存器的范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op		26	25	24	21	20	19	16	15					5	4	3	0
111110			w1		0		w0		VRX		00000111101				1		VRZ	

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

编码表中 VRX,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.24 VCMPNE.T——矢量不等于比较指令

统一化指令	
语法	<code>vcmpne.T vrz, vrx, vry</code> (T=S8/S16/S32 /U8/U16/U32)
操作	<code>size = {8, 16, 32}; Number = 128 / size; for(i=0; i< Number; i++) { if(Vx(i) != Vy(i)) { Vz(i) = AllOnes; } Else { Vz(i) = AllZeros; } }</code>
编译结果	仅存在 32 位指令。 <code>vcmpne.T vrz, vrx, vry</code>
数据类型	U8/U16/U32 S8/S16/S32
说明	比较 VRX 和 VRY 的对应元素，如果 VRX 和 VRY 对应元素不相等，则将 VRZ 的相应元素置全 1；否则，置全 0。
影响标志位	不影响
限制	寄存器的范围为 vr0-vr15。
异常	无

32 位指令	
操作	<pre>size = {8, 16, 32}; Number = 128 / size; for(i=0; i< Number; i++) { if(Vx(i) != Vy(i)) { Vz(i) = AllOnes; } Else { Vz(i) = AllZeros; } }</pre>
语法	vcmpne.T vrz, vrx, vry
数据类型	U8/U16/U32 S8/S16/S32
说明	比较 VRX 和 VRY 的对应元素，如果 VRX 和 VRY 对应元素不相等，则将 VRZ 的相应元素置全 1；否则，置全 0。
影响标志位	不影响
限制	寄存器的范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15		5	4	3	0	
111110		w1		VRY		w0		VRX		00000100010				S	VRZ

注：

- 编码表中 w1, w0 用来表征指令位宽类型，其中：
- [w1,w0] = 2’ b00 表示 size 为 8bit 指令；
 - [w1,w0] = 2’ b01 表示 size 为 16bit 指令；
 - [w1,w0] = 2’ b10 表示 size 为 32bit 指令；
- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.25 VCOMPNEZ.T——矢量不等于 0 比较指令

统一化指令	
语法	<code>vcmpnez.T vrz, vrx</code> (T=S8/S16/S32 /U8/U16/U32)
操作	<code>size = {8, 16, 32};</code> <code>Number = 128 / size;</code> <code>for(i=0; i< Number; i++)</code> <code>{</code> <code>if(Vx(i) != 0)</code> <code>{</code> <code>Vz(i) = AllOnes;</code> <code>}</code> <code>Else</code> <code>{</code> <code>Vz(i) = AllZeros;</code> <code>}</code> <code>}</code>
编译结果	仅存在 32 位指令。 <code>vcmpnez.T vrz, vrx</code>
数据类型	U8/U16/U32 S8/S16/S32
说明	比较 VRX 元素和 0，如果 VRX 元素不等于 0，则将 VRZ 的对应元素置全 1；否则，置全 0。
影响标志位	不影响
限制	寄存器的范围为 vr0-vr15。
异常	无

32 位指令	
操作	<pre>size = {8, 16, 32}; Number = 128 / size; for(i=0; i< Number; i++) { if(Vx(i) != 0) { Vz(i) = AllOnes; } Else { Vz(i) = AllZeros; } }</pre>
语法	vcmpnez.T vrz, vrx
数据类型	U8/U16/U32 S8/S16/S32
说明	比较 VRX 元素和 0，如果 VRX 元素不等于 0，则将 VRZ 的对应元素置全 1；否则，置全 0。
影响标志位	不影响
限制	寄存器的范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1	0		w0	VRX			00000100011			S	VRZ

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.26 VCNT1.T——矢量数据 1 个数指令

统一化指令	
语法	vcnt1.T vrz,vrx (T=8/16/32/64)
操作	size = {8,16,32,64}; number = 128/size; element 内 1 的个数 Vz(i)=count_one(Vx(i)); i=0:(number-1) vrz[i],vrz[i] 指相应 element 的数据;
编译结果	仅存在 32 位指令。 vcnt1.T vrz, vrz
数据类型	8/16/32/64
说明	相应 element 中 1 的位数
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32,64}; number = 128/size; element 内 1 的个数 Vz(i)=count_one(Vx(i)); i=0:(number-1) vrz[i],vrz[i] 指相应 element 的数据;
语法	vcnt1.T vrz, vrz
数据类型	8/16/32/64
说明	相应 element 中 1 的位数
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15		4	3	0
	111110	x		VRY	x		VRX			000110001111			VRZ

注：

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号

16.27 VDTL.T.2——二元去交织指令

统一化指令																																					
语法	vdtl.T.2 vrz, vrx (T=8/16/32)																																				
操作	size= {8,16,32}; number = 128 / size * 2; T(2*number-1:0)={Vx+1,Vx}; Vz+1(i)=T(2i+1);Vz=T(2i); i=0:(number-1)																																				
编译结果	仅存在 32 位指令。 vdtl.T.2 vrz, vrx																																				
数据类型	8/16/32																																				
说明	将源矢量寄存器 VRX 和 VRX+1 进行中的元素进行去交织处理，并放入 VRZ 和 VRZ+1 中。下面以元素大小为半字举例： <table><tr><td>VRX</td><td>A7</td><td>A6</td><td>A5</td><td>A4</td><td>A3</td><td>A2</td><td>A1</td><td>A0</td></tr><tr><td>VRX1</td><td>B7</td><td>B6</td><td>B5</td><td>B4</td><td>B3</td><td>B2</td><td>B1</td><td>B0</td></tr></table> 交叉以后得到如下结果 <table><tr><td>VRZ</td><td>B6</td><td>A4</td><td>B2</td><td>B0</td><td>A6</td><td>A4</td><td>A2</td><td>A0</td></tr><tr><td>VRZ1</td><td>B7</td><td>B5</td><td>B3</td><td>B1</td><td>A7</td><td>A5</td><td>A3</td><td>A1</td></tr></table>	VRX	A7	A6	A5	A4	A3	A2	A1	A0	VRX1	B7	B6	B5	B4	B3	B2	B1	B0	VRZ	B6	A4	B2	B0	A6	A4	A2	A0	VRZ1	B7	B5	B3	B1	A7	A5	A3	A1
VRX	A7	A6	A5	A4	A3	A2	A1	A0																													
VRX1	B7	B6	B5	B4	B3	B2	B1	B0																													
VRZ	B6	A4	B2	B0	A6	A4	A2	A0																													
VRZ1	B7	B5	B3	B1	A7	A5	A3	A1																													
影响标志位	无影响																																				
限制	寄存器的范围为 vr0-vr15。																																				
异常	无																																				

32 位指令																																					
操作	size = {8,16,32}; number = 128 / size * 2; T(2*number-1:0)={Vx+1,Vx}; Vz+1(i)=T(2i+1);Vz=T(2i); i=0:(number-1)																																				
语法	vdtl.T.2 vrz, vrx																																				
数据类型	8/16/32																																				
说明	将源矢量寄存器 VRX 和 VRX+1 进行中的元素进行去交织处理，并放入 VRZ 和 VRZ+1 中。下面以元素大小为半字举例： <table><tr><td>VRX</td><td>A7</td><td>A6</td><td>A5</td><td>A4</td><td>A3</td><td>A2</td><td>A1</td><td>A0</td></tr><tr><td>VRX1</td><td>B7</td><td>B6</td><td>B5</td><td>B4</td><td>B3</td><td>B2</td><td>B1</td><td>B0</td></tr></table> 交叉以后得到如下结果 <table><tr><td>VRZ</td><td>B6</td><td>A4</td><td>B2</td><td>B0</td><td>A6</td><td>A4</td><td>A2</td><td>A0</td></tr><tr><td>VRZ1</td><td>B7</td><td>B5</td><td>B3</td><td>B1</td><td>A7</td><td>A5</td><td>A3</td><td>A1</td></tr></table>	VRX	A7	A6	A5	A4	A3	A2	A1	A0	VRX1	B7	B6	B5	B4	B3	B2	B1	B0	VRZ	B6	A4	B2	B0	A6	A4	A2	A0	VRZ1	B7	B5	B3	B1	A7	A5	A3	A1
VRX	A7	A6	A5	A4	A3	A2	A1	A0																													
VRX1	B7	B6	B5	B4	B3	B2	B1	B0																													
VRZ	B6	A4	B2	B0	A6	A4	A2	A0																													
VRZ1	B7	B5	B3	B1	A7	A5	A3	A1																													
影响标志位	无影响																																				
限制	寄存器的范围为 vr0-vr15。																																				
异常	无																																				

32 位指令格式：

31	op	26	25	24	21	20	19	16	15		4	3	0
	111110	w1		Dontcare	w0		VRX			000110110001			VRZ

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2' b00 表示 size 为 8bit 指令；

[w1,w0] = 2' b01 表示 size 为 16bit 指令；

[w1,w0] = 2' b10 表示 size 为 32bit 指令；

编码表中 VRX,VRZ,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.28 VDUP.T.1——一元矢量源目的寄存器之间的拷贝指令

统一化指令	
语法	
操作	size = {8, 16, 32}; Number = 128 / size; for(i=0; i< number ; i++) { Vz(i) = Vx(index); }
编译结果	仅存在 32 位指令。 vdup.T.1 vrz, vrz[index]
数据类型	8/16/32
说明	通过 index 来选择 VRX 中的元素并拷贝到 VRZ 的每个元素中，Index 小于 16。 VRX n-1 0 VRZ n-1 0
影响标志位	不影响
限制	寄存器的范围为 vr0-vr15
异常	无

32 位指令	
操作	<pre>size = {8, 16, 32}; Number = 128 / size; for(i=0; i< number ; i++) { Vz(i) = Vx(index); }</pre>
语法	vdup.T.1 vrz, vrx[index]
数据类型	8/16/32
说明	通过 index 来选择 VRX 中的元素并拷贝到 VRZ 的每个元素中，Index 小于 16。 VRX n-1 0 VRZ n-1 0
影响标志位	不影响
限制	寄存器的范围为 vr0-vr15
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15		5	4	3	0
111110		w1	INDEX4			w0	VRX			00011010010			0	VRZ

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

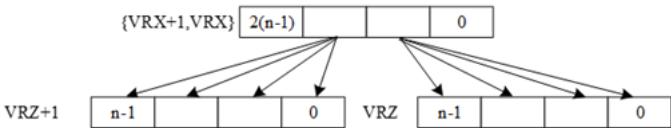
[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

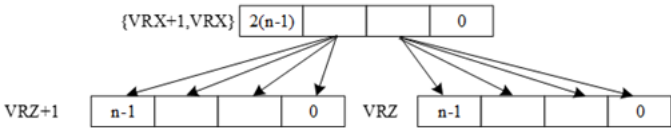
[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

编码表中 VRX,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.29 VDUP.T.2——二元矢量源目的寄存器之间的拷贝指令

统一化指令	
语法	vdup.T.2 vrz, vrx[index] (T=8/16/32)
操作	size = {8, 16, 32}; Number = 128 / size; T={Vx+1,Vx}; for(i=0; i< number ; i++) { Vz(i) = T(index); Vz+1(i) = T(index+1); }
编译结果	仅存在 32 位指令。 vdup.T.2 vrz, vrx[index]
数据类型	8/16/32
说明	把 VRX 和 VRX+1 组合起来，通过 index 来选择 {VRX+1, VRX} 中的元素并拷贝到 VRZ 的每个元素中，通过 index+1 来选择 VRX 中的元素并拷贝到 VRZ+1 的每个元素中，Index 小于 16。 
影响标志位	不影响
限制	寄存器的范围为 vr0-vr15
异常	无

32 位指令	
操作	size = {8, 16, 32}; Number = 128 / size; T={V_{x+1},V_x}; for(i=0; i< number ; i++) { V_z(i) = T(index); V_{z+1}(i) = T(index+1); }
语法	vdup.T.2 vrz, vr _x [index]
数据类型	8/16/32
说明	把 VRX 和 VRX+1 组合起来，通过 index 来选择 {VRX+1, VRX} 中的元素并拷贝到 VRZ 的每个元素中，通过 index+1 来选择 V RX 中的元素并拷贝到 VRZ+1 的每个元素中，Index 小于 16。 
影响标志位	不影响
限制	寄存器的范围为 vr0-vr15
异常	无

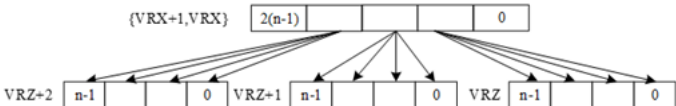
32 位指令格式：

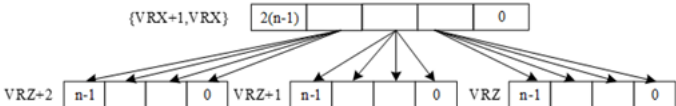
31	op	26	25	24	21	20	19	16	15		5	4	3	0
111110		w1	INDEX4			w0	VRX			00011010010			1	VRZ

注：

- 编码表中 w1, w0 用来表征指令位宽类型，其中：
- [w1,w0] = 2’ b00 表示 size 为 8bit 指令；
- [w1,w0] = 2’ b01 表示 size 为 16bit 指令；
- [w1,w0] = 2’ b10 表示 size 为 32bit 指令；
- 编码表中 VRX,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.30 VDUP.T.3——三元矢量源目的寄存器之间的拷贝指令

统一化指令	
语法	vdup.T.3 vrz, vrx[index] (T=8/16/32)
操作	size = {8, 16, 32}; Number = 128 / size; T={V_x+1,V_x}; for(i=0; i< number ; i++) { V_z(i) = T(index); V_z+1(i) = T(index+1); V_z+2(i) = T(index+2); }
编译结果	仅存在 32 位指令。 vdup.T.3 vrz, vrx[index]
数据类型	8/16/32
说明	把 VRX 和 VRX+1 组合起来，通过 index 来选择 {VRX+1, VRX} 中的元素并拷贝到 VRZ 的每个元素中，通过 index+1 来选择 {VRX+1, VRX} 中的元素并拷贝到 VRZ+1 的每个元素中，index+2 来选择 {VRX+1, V RX} 中的元素并拷贝到 VRZ+2 的每个元素中,Index 小于 16。 
影响标志位	不影响
限制	寄存器的范围为 vr0-vr15
异常	无

32 位指令	
操作	size = {8, 16, 32}; Number = 128 / size; T={V_{x+1},V_x}; for(i=0; i< number ; i++) { V_z(i) = T(index); V_{z+1}(i) = T(index+1); V_{z+2}(i) = T(index+2); }
语法	vdup.T.3 vrz, vr _x [index]
数据类型	8/16/32
说明	把 VR_x 和 VR_x+1 组合起来, 通过 index 来选择 {VR_x+1, VR_x} 中的元素并拷贝到 VR_z 的每个元素中, 通过 index+1 来选择 {VR_x+1, VR_x} 中的元素并拷贝到 VR_z+1 的每个元素中, index+2 来选择 {VR_x+1, VR_x} 中的元素并拷贝到 VR_z+2 的每个元素中,Index 小于 16。 
影响标志位	不影响
限制	寄存器的范围为 vr0-vr15
异常	无

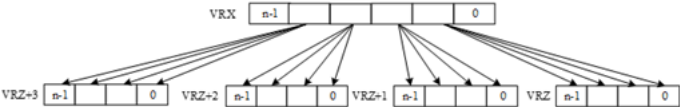
32 位指令格式：

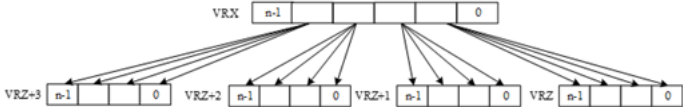
31	op	26	25	24	21	20	19	16	15				5	4	3	0
111110			w1	INDEX4			w0	VR _x			00011010011			0	VR _z	

注：

- 编码表中 w1, w0 用来表征指令位宽类型，其中：
- [w1,w0] = 2’ b00 表示 size 为 8bit 指令；
- [w1,w0] = 2’ b01 表示 size 为 16bit 指令；
- [w1,w0] = 2’ b10 表示 size 为 32bit 指令；
- 编码表中 VR_x,VR_z 分别表征这条指令各个矢量寄存器的寄存器号；

16.31 VDUP.T.4——四元矢量源目的寄存器之间的拷贝指令

统一化指令	
语法	vdup.T.4 vrz, vrx[index] (T=8/16/32)
操作	size = {8, 16, 32}; Number = 128 / size; T={Vx+1,Vx}; for(i=0; i< number ; i++) { Vz(i) = T(index); Vz+1(i) = T(index+1); Vz+2(i) = T(index+2); Vz+3(i) = T(index+3); }
编译结果	仅存在 32 位指令。 vdup.T.4 vrz, vrx[index]
数据类型	8/16/32
说明	把 VRX 和 VRX+1 组合起来，通过 index 来选择 {VRX+1, VRX} 中的元素并拷贝到 VRZ 的每个元素中，通过 index+1 来选择 {VRX+1, VRX} 中的元素并拷贝到 VRZ+1 的每个元素中，通过 index+2 来选择 {VRX+1, VRX} 中的元素并拷贝到 VRZ+2 的每个元素中，通过 index+3 来选择 {VRX+1, V RX} 中的元素并拷贝到 VRZ+2 的每个元素中,Index 小于 16。 
影响标志位	不影响
限制	寄存器的范围为 vr0-vr15
异常	无

32 位指令	
操作	<pre>size = {8, 16, 32}; Number = 128 / size; T={Vx+1,Vx}; for(i=0; i< number ; i++) { Vz(i) = T(index); Vz+1(i) = T(index+1); Vz+2(i) = T(index+2); Vz+3(i) = T(index+3); }</pre>
语法	vdup.T.4 vrz, vrx[index]
数据类型	8/16/32
说明	把 VRX 和 VRX+1 组合起来，通过 index 来选择 {VRX+1, VRX} 中的元素并拷贝到 VRZ 的每个元素中，通过 index+1 来选择 {VRX+1, VRX} 中的元素并拷贝到 VRZ+1 的每个元素中，通过 index+2 来选择 {VRX+1, VRX} 中的元素并拷贝到 VRZ+2 的每个元素中，通过 index+3 来选择 {VRX+1, VRX} 中的元素并拷贝到 VRZ+3 的每个元素中，Index 小于 16。 
影响标志位	不影响
限制	寄存器的范围为 vr0-vr15
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0	
111110		w1	INDEX4			w0	VRX			00011010011			1	VRZ

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

编码表中 VRX,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.32 VDUPG.T——通用源矢量目的寄存器拷贝指令

统一化指令	
语法	vdupg.T vrz, rx (T=8/16/32)
操作	size = {8,16,32}; number = 128 / size; for(i=0; i< number ; i++) { Vz(i)=Rx[size-1:0]; }
编译结果	仅存在 32 位指令。 vdupg.T vrz, rx
数据类型	8/16/32
说明	将通用寄存器表示数据按 size 拷贝至矢量寄存器的每一个元素中去。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32}; number = 128 / size; for(i=0; i< number ; i++) {Vz(i)=Rx[size-1:0];}
语法	vdupg.T vrz, rx
数据类型	8/16/32
说明	将通用寄存器表示数据按 size 拷贝至矢量寄存器的每一个元素中去。
影响标志位	无影响
限制	矢量寄存器的范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	16	15			5	4	3	0
111110			w1		INDEX4		RX		00011100010			w0		VRZ

注：

编码表中 w1，w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

$[w1,w0] = 2'b10$ 表示 size 为 32bit 指令；

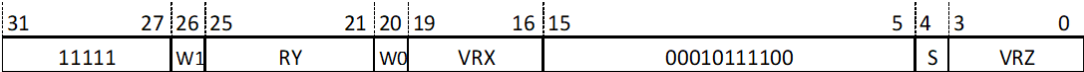
编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.33 VEXH.T——矢量高位数据插入指令

统一化指令	
语法	vexh.T vrz,vrx,ry (T=8/16/32/64)
操作	size = {8,16,32,64}; number = 128/size; imm1=ry[5:0]; imm2=ry[11:6]; Vz(i)={Vx(i)[imm2:imm1],Vz(i)[size+imm1-imm2-2:0]}; i=0:(number-1) size > imm2 imm1 0
编译结果	仅存在 32 位指令。 vexh.T vrz, vrz,ry
数据类型	8/16/32/64
说明	用 ry 中存储 0~5 位和 6~11 位作为立即数区间截取 vrz 中的数据放入 Vz 的高位，Vz 其余位保持不变。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32,64}; number = 128/size; imm1=ry[5:0]; imm2=ry[11:6]; Vz(i)={Vx(i)[imm2:imm1],Vz(i)[size+imm1-imm2-2:0]}; i=0:(number-1) size > imm2 imm1 0
语法	vexh.T vrz, vrz,ry
数据类型	8/16/32/64
说明	用 ry 中存储 0~5 位和 6~11 位作为立即数区间截取 vrz 中的数据放入 Vz 的高位，Vz 其余位保持不变。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：



注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

[w1,w0] = 2’ b11 表示 size 为 64bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,RY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.34 VEXL.T——矢量低位数据插入指令

统一化指令	
语法	vexl.T vrz,vrx,ry (T=8/16/32/64)
操作	size = {8,16,32,64}; number = 128/size; imm1=ry[5:0]; imm2=ry[11:6]; Vz(i)={Vz(i)[size-1:imm2-imm1+1], Vx(i)[imm2:imm1]}; i=0:(number-1) size > imm2 imm1 0
编译结果	仅存在 32 位指令。 vexl.T vrz, vrz,ry
数据类型	8/16/32/64
说明	用 ry 中存储 0~5 位和 6~11 位作为立即数区间截取 vrz 中的数据放入 Vz 的低位，Vz 其余位保持不变。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32,64}; number = 128/size; imm1=ry[5:0]; imm2=ry[11:6]; Vz(i)={Vz(i)[size-1:imm2-imm1+1], Vx(i)[imm2:imm1]}; i=0:(number-1) size > imm2 imm1 0
语法	vexl.T vrz, vrx,ry
数据类型	8/16/32/64
说明	用 ry 中存储 0~5 位和 6~11 位作为立即数区间截取 vrx 中的数据放入 Vz 的低位，Vz 其余位保持不变。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	21	20	19	16	15	5	4	3	0
11111W1		RY		W0		VRX		00010111101		S		VRZ

注：

- 编码表中 w1, w0 用来表征指令位宽类型，其中：
- [w1,w0] = 2’ b00 表示 size 为 8bit 指令；
 - [w1,w0] = 2’ b01 表示 size 为 16bit 指令；
 - [w1,w0] = 2’ b10 表示 size 为 32bit 指令；
 - [w1,w0] = 2’ b11 表示 size 为 64bit 指令；
- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 VRX,RY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.35 VEXPE.T——矢量 e 指数指令

统一化指令	
语法	vexpe.T vz, vx (T=S8/S16/S32 /U8/U16/U32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] T 表征指令类型 T={u8,u16,u32,s8,s16,s32} e 指数快速算法公式： Vz(i) = 1.442695 * Vx(i) << 1 for(i=0; i< Number; i++) { If(Vx(i)<0.5){ Vx(i)=0.5 ; } Elseif(Vx(i)>1){ Vx(i)=1; } Else{ Vx(i)=Vx(i); } } for(i=0; i< Number; i++) { If(T=s8)Vz(i) = 0x2E *Vx(i)<<1 (操作数、结果均为有符号数) If(T=s16) Vz(i) = 0x2E2A *Vx(i)<<1 (操作数、结果均为有符号数) If(T=s32)Vz(i)=0x2E2A8EB4*Vx(i)<<1 (操作数、结果均为有符号数) If(T=u8) Vz(i) = 0x5C *Vx(i)<<1 (操作数、结果均为无符号数) If(T=u16) Vz(i) = 0x5C55 *Vx(i)<<1 (操作数、结果均为无符号数) If(T=u32)Vz(i)=0x5C551D68*Vx(i)<<1 (操作数、结果均为无符号数) } }
编译结果	仅存在 32 位指令。 vexpe.T vz, vx
数据类型	U8/U16/U32 S8/S16/S32
说明	Vx(i) 值取值范围是 [0.5, 1], 若超过该范围，则在运算前强制将 Vx(i) 置为边界值，根据 e 指数快速算法公式：Vz(i) = 1.88 - Vx(i)，将 magic 值转化为相应有无符号，size 大小的定点数，在判断 VRX 中元素边界值后，将 VRX 中的元素代入公式计算出的结果存入 VRZ
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。625© 【杭州中天微系统有限公司】版权所有
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] T 表征指令类型 T={u8,u16,u32,s8,s16,s32} e 指数快速算法公式： $V_z(i) = 1.442695 * V_x(i) \ll 1$ for(i=0; i< Number; i++) { If($V_x(i)<0.5$){ $V_x(i)=0.5$; } Elseif($V_x(i)>1$){ $V_x(i)=1$; } Else{ $V_x(i)=V_x(i)$; } } for(i=0; i< Number; i++) { If(T=s8)$V_z(i) = 0x2E * V_x(i) \ll 1$ (操作数、结果均为有符号数) If(T=s16) $V_z(i) = 0x2E2A * V_x(i) \ll 1$ (操作数、结果均为有符号数) If(T=s32)$V_z(i)=0x2E2A8EB4*V_x(i) \ll 1$ (操作数、结果均为有符号数) If(T=u8) $V_z(i) = 0x5C * V_x(i) \ll 1$ (操作数、结果均为无符号数) If(T=u16) $V_z(i) = 0x5C55 * V_x(i) \ll 1$ (操作数、结果均为无符号数) If(T=u32)$V_z(i)=0x5C551D68*V_x(i) \ll 1$ (操作数、结果均为无符号数) }
语法:	vexpe.T vz, vx
数据类型	U8/U16/U32 S8/S16/S32
说明	$V_x(i)$ 值取值范围是 [0.5, 1], 若超过该范围，则在运算前强制将 $V_x(i)$ 置为边界值，根据 e 指数快速算法公式：$V_z(i) = 1.88 - V_x(i)$，将 magic 值转化为相应有无符号，size 大小的定点数，在判断 VRX 中元素边界值后，将 VRX 中的元素代入公式计算出的结果存入 VRZ
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15		5	4	3	0
	111110	w1		X		w0	VRX			00011101100		S		VRZ

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.36 VEXT.T——寄存器矢量数据拼接指令

统一化指令	
语法	vext.T vrz,vrx,vry,rk (T=8)
操作	size = 8; number = 128/size; imm=rk[5:0]; If imm[5]==0, Vz(imm[3:0]:0)=Vx(imm[3:0]:0); Else Vz(imm[3:0]:0)=Vx(15:15-imm[3:0]); If imm[4]==0, Vz(15:imm[3:0]+1) =Vy(15-imm[3:0]-1:0); Else Vz (15:imm[3:0]+1)= Vy(15:imm[3:0]+1); 其中 imm[3:0] 的范围为 0~14; 根据 rk 的低 6 位数据把 Vrx 和 Vry 的元素进行拼接放入 VrZ 之中 vrX[i] 指相应 element 的数据;
编译结果	仅存在 32 位指令。 vext.T vrz, vrX,vry,rk
数据类型	8
说明	将源矢量寄存器 VRX，VRY 的元素进行拼接放入 VRZ 之中。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = 8; number = 128/size; imm=rk[5:0]; vr _x [i],vr _y [i],vr _z [i] 指相应位置 element 的数据; If imm[5]==0, Vz(imm[3:0]:0)=V _x (imm[3:0]:0); (将 V _x 的低位若干个元素拷贝到 V _z 的低位若干个元素) Else Vz(imm[3:0]:0)=V _x (15:15-imm[3:0]); (将 V _x 的高位若干个元素拷贝到 V _z 的低位若干个元素) If imm[4]==0, Vz(15:imm[3:0]+1)=V _y (15-imm[3:0]-1:0); (将 V _y 的低位若干个元素拷贝到 V _z 的高位若干个元素) Else Vz (15:imm[3:0]+1)=V _y (15:imm[3:0]+1); (将 V _y 的高位若干个元素拷贝到 V _z 的高位若干个元素) 其中 imm[3:0] 的范围为 0~14; 根据 rk 低 6 位数据把 V _{rx} 和 V _{ry} 的元素进行拼接放入 V _{rz} 之中
语法	vext.8 vr _z , vr _x ,vr _y ,rk
数据类型	U8
说明	将源矢量寄存器 VR _X , VR _Y 的元素进行拼接放入 VR _Z 中。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	7	6	4	3	0
111110		k4		VRY		k3		VRX		000101011		K2K1K0		VRZ

注：

k4,k3,k2,k1 拼起来作为 rk；

编码表中 VR_X,VR_Y,VR_Z 分别表征这条指令各个矢量寄存器的寄存器号

16.37 VEXTI.T——立即数矢量数据拼接指令

统一化指令	
语法	vexti.T vrz,vrx,vry, imm6 (T=8)
操作	size = 8; number = 128/size; If imm6[5]==0, Vz(imm[3:0]:0)=Vx(imm[3:0]:0); Else Vz(imm[3:0]:0)=Vx(15:15-imm[3:0]); If imm6[4]==0, Vz(15:imm[3:0]+1)=Vy(15-imm[3:0]-1:0); Else Vz (15:imm[3:0]+1)=Vy(15:imm[3:0]+1); 其中 imm[3:0] 的范围为 0~14; 根据 imm6 数据把 Vrx 和 Vry 的元素进行拼接放入 Vrз 之中 vrз[i] 指相应 element 的数据;
编译结果	仅存在 32 位指令。 vexti.T vrz, vrз,vry, imm6
数据类型	8
说明	将源矢量寄存器 VRX，VRY 的元素进行拼接放入 VRZ 之中。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = 8; number = 128/size; vr _x [i],vr _y [i],vr _z [i] 指相应位置 element 的数据; If imm6[5]==0, V _z (imm[3:0]:0)=V _x (imm[3:0]:0); (将 V _x 的低位若干个元素拷贝到 V _z 的低位若干个元素) Else V _z (imm[3:0]:0)=V _x (15:15-imm[3:0]); (将 V _x 的高位若干个元素拷贝到 V _z 的低位若干个元素) If imm6[4]==0, V _z (15:imm[3:0]+1)=V _y (15-imm[3:0]-1:0); (将 V _y 的低位若干个元素拷贝到 V _z 的高位若干个元素) Else V _z (15:imm[3:0]+1)=V _y (15:imm[3:0]+1); (将 V _y 的高位若干个元素拷贝到 V _z 的高位若干个元素) 其中 imm[3:0] 的范围为 0~14; 根据 imm6 数据把 V _{rx} 和 V _{ry} 的元素进行拼接放入 V _{rz} 之中
语法	vexti.8 vr _z , vr _x ,vr _y ,imm6
数据类型	U8
说明	将源矢量寄存器 VR _X , VR _Y 的元素进行拼接放入 VR _Z 中。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	7	6	4	3	0
111110		I1	VR _Y		I0	VR _X		00010110			IMM4		VR _Z	

注：

Imm6={I1,I0,imm4}, 作为输入；
编码表中 VR_X,VR_Y,VR_Z 分别表征这条指令各个矢量寄存器的寄存器号

16.38 VINS.T.1——一元矢量插入指令

统一化指令	
语法	vins.T.1 vrz, vrx,index1,index2 (T=8/16/32)
操作	size = {8,16,32}; number = 128 / size; Vz(index2) = Vx(index1);
编译结果	仅存在 32 位指令。 vins.T.1 vrz, vrx,index1,index2
数据类型	8/16/32
说明	将源矢量寄存器 VRX 的某个元素（由 index1 来决定是哪个元素）插入目的矢量寄存器 VRZ 的指定元素（由 index2 来决定是哪个元素）中，VRZ 的其余元素保持不变。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32}; number = 128 / size; Vz(index2) = Vx(index1);
语法	vins.T.1 vrz, vrx,index1,index2
数据类型	8/16/32
说明	将源矢量寄存器 VRX 的某个元素（由 index1 来决定是哪个元素）插入目的矢量寄存器 VRZ 的指定元素（由 index2 来决定是哪个元素）中，VRZ 的其余元素保持不变。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15		7	6	5	4	3	0										
111110				I1	INDEX1				I0	VRX				00001000				0	INDEX2				VRZ			

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

编码表中 I1, I0 与 Imm3 组合在一起用来表征指令位宽类型和 Index，其中：

$IMM[4:0] = \{I1,I0,IMM3\}$ ；

if($IMM[4:3]=2'b01$) size=16, IMM[2:0] 表示 index0 到 7；

if($IMM[4:3]=2'b10$) size=32, IMM[2:0] 表示 index0 到 3（高位为 0）

if($IMM[4:3]=2'b00$) size=8, IMM[2:0] 表示 index0 到 7；

if($IMM[4:3]=2'b11$) size=8, IMM[2:0] 表示 index8 到 15；

16.39 VINS.T.2——二元矢量插入指令

统一化指令	
语法	vins.T.2 vrz, vrx, index1,index2 (T=8/16/32)
操作	size = {8,16,32}; number = 128 / size; T={Vx+1,Vx}; Vz(index2) = T(index1); Vz+1(index2) = T(index1+1);
编译结果	仅存在 32 位指令。 vins.T.2 vrz, vrx, index1,index2
数据类型	8/16/32
说明	把 VRX+1,VRX 组合，将源矢量寄存器 {VRX+1, VRX} 的某个元素（由 index1 来决定是哪个元素）插入目的矢量寄存器 VRZ 的指定元素（由 index2 来决定是哪个元素）中 VRZ 的其余元素保持不变； 将源矢量寄存器 {VRX+1, VRX} 的某个元素（由 index1+1 来决定是哪个元素）插入目的矢量寄存器 VRZ+1 的指定元素（由 index2 来决定是哪个元素）中，VRZ+1 的其余元素保持不变。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32}; number = 128 / size; T={V _x +1,V _x }; V _z (index2) = T(index1); V _z +1(index2) = T(index1+1);
语法	vins.T.2 vrz, vr _x ,index1,index2
数据类型	8/16/32
说明	把 VR _x +1,VR _x 组合, 将源矢量寄存器 {VR _x +1, VR _x } 的某个元素 (由 index1 来决定是哪个元素) 插入目的矢量寄存器 VR _z 的指定元素 (由 index2 来决定是哪个元素) 中, VR _z 的其余元素保持不变; 将源矢量寄存器 {VR _x +1, VR _x } 的某个元素 (由 index1+1 来决定是哪个元素) 插入目的矢量寄存器 VR _z +1 的指定元素 (由 index2 来决定是哪个元素) 中, VR _z +1 的其余元素保持不变。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr15。
异常	无

32 位指令格式:

31	op	26	25	24	21	20	19	16	15	7	6	5	4	3	0
111110		I1	INDEX1		I0	VRX		00001000			1	INDEX2		VRZ	

注:

编码表中 w1, w0 用来表征指令位宽类型, 其中:

[w1,w0] = 2' b00 表示 size 为 8bit 指令;

[w1,w0] = 2' b01 表示 size 为 16bit 指令;

[w1,w0] = 2' b10 表示 size 为 32bit 指令;

编码表中 VR_x,VR_y,VR_z 分别表征这条指令各个矢量寄存器的寄存器号;

编码表中 I1, I0 与 Imm3 组合在一起用来表征指令位宽类型和 Index, 其中:

IMM[4:0] = {I1,I0,IMM3};

if(IMM[4:3]=2' b01) size=16, IMM[2:0] 表示 index0 到 7;

if(IMM[4:3]=2' b10) size=32, IMM[2:0] 表示 index0 到 3 (高位为 0)

if(IMM[4:3]=2' b00) size=8, IMM[2:0] 表示 index0 到 7;

if(IMM[4:3]=2' b11) size=8, IMM[2:0] 表示 index8 到 15;

16.40 VINS.T.3——三元矢量插入指令

统一化指令	
语法	vins.T.3 vrz, vrx, index1,index2 (T=8/16/32)
操作	size = {8,16,32}; number = 128 / size; T={Vx+1,Vx}; Vz(index2) = T(index1); Vz+1(index2) = T(index1+1); Vz+2(index2) = T(index1+2);
编译结果	仅存在 32 位指令。 vins.T.3 vrz, vrx, index1,index2
数据类型	8/16/32
说明	把 VRX+1,VRX 组合，将源矢量寄存器 {VRX+1, VRX} 的某个元素（由 index1 来决定是哪个元素）插入目的矢量寄存器 VRZ 的指定元素（由 index2 来决定是哪个元素）中，VRZ 的其余元素保持不变； 将源矢量寄存器 {VRX+1, VRX} 的某个元素（由 index1+1 来决定是哪个元素）插入目的矢量寄存器 VRZ+1 的指定元素（由 index2 来决定是哪个元素）中，VRZ+1 的其余元素保持不变； 将源矢量寄存器 {VRX+1, VRX} 的某个元素（由 index1+2 来决定是哪个元素）插入目的矢量寄存器 VRZ+2 的指定元素（由 index2 来决定是哪个元素）中，VRZ+2 的其余元素保持不变。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32}; number = 128 / size; T={V _x +1,V _x }; V _z (index2) = T(index1); V _z +1(index2) = T(index1+1); V _z +2(index2) = T(index1+2);
语法	vins.T.3 vrz, vr _x ,index1,index2
数据类型	8/16/32
说明	把 VR _x +1,VR _x 组合, 将源矢量寄存器 {VR _x +1, VR _x } 的某个元素 (由 index1 来决定是哪个元素) 插入目的矢量寄存器 VR _z 的指定元素 (由 index2 来决定是哪个元素) 中, VR _z 的其余元素保持不变; 将源矢量寄存器 {VR _x +1, VR _x } 的某个元素 (由 index1+1 来决定是哪个元素) 插入目的矢量寄存器 VR _z +1 的指定元素 (由 index2 来决定是哪个元素) 中, VR _z +1 的其余元素保持不变; 将源矢量寄存器 {VR _x +1, VR _x } 的某个元素 (由 index1+2 来决定是哪个元素) 插入目的矢量寄存器 VR _z +2 的指定元素 (由 index2 来决定是哪个元素) 中, VR _z +2 的其余元素保持不变。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	7	6	5	4	3	0
111110		I1		INDEX1		I0		VRX		00001001		0	INDEX2		VRZ

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

编码表中 VR_x,VR_y,VR_z 分别表征这条指令各个矢量寄存器的寄存器号；

编码表中 I1, I0 与 Imm3 组合在一起用来表征指令位宽类型和 Index，其中：

IMM[4:0] = {I1,I0,IMM3}；

if(IMM[4:3]=2’ b01) size=16, IMM[2:0] 表示 index0 到 7；

if(IMM[4:3]=2’ b10) size=32, IMM[2:0] 表示 index0 到 3（高位为 0）

if(IMM[4:3]=2’ b00) size=8, IMM[2:0] 表示 index0 到 7；

if(IMM[4:3]=2' b11) size=8, IMM[2:0] 表示 index8 到 15;

16.41 VINS.T.4——四元矢量插入指令

统一化指令	
语法	vins.T.4 vrz, vrx, index1,index2 (T=8/16/32)
操作	size = {8,16,32}; number = 128 / size; T={Vx+1,Vx}; Vz(index2) = T(index1); Vz+1(index2) = T(index1+1); Vz+2(index2) = T(index1+2); Vz+3(index2) = T(index1+3);
编译结果	仅存在 32 位指令。 vins.T.4 vrz, vrx, index1,index2
数据类型	8/16/32
说明	把 VRX+1,VRX 组合, 将源矢量寄存器 {VRX+1, VRX} 的某个元素（由 index1 来决定是哪个元素）插入目的矢量寄存器 VRZ 的指定元素（由 index2 来决定是哪个元素）中, VRZ 的其余元素保持不变; 将源矢量寄存器 {VRX+1, VRX} 的某个元素（由 index1+1 来决定是哪个元素）插入目的矢量寄存器 VRZ+1 的指定元素（由 index2 来决定是哪个元素）中, VRZ+1 的其余元素保持不变; 将源矢量寄存器 {VRX+1, VRX} 的某个元素（由 index1+2 来决定是哪个元素）插入目的矢量寄存器 VRZ+2 的指定元素（由 index2 来决定是哪个元素）中, VRZ+2 的其余元素保持不变; 将源矢量寄存器 {VRX+1, VRX} 的某个元素（由 index1+3 来决定是哪个元素）插入目的矢量寄存器 VRZ+3 的指定元素（由 index2 来决定是哪个元素）中, VRZ+3 的其余元素保持不变。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32}; number = 128 / size; T={Vx+1,Vx}; Vz(index2) = T(index1); Vz+1(index2) = T(index1+1); Vz+2(index2) = T(index1+2); Vz+3(index2) = T(index1+3);
语法	vins.T.4 vrz, vrx,index1,index2
数据类型	8/16/32
说明	把 VRX+1,VRX 组合, 将源矢量寄存器 {VRX+1, VRX} 的某个元素（由 index1 来决定是哪个元素）插入目的矢量寄存器 VRZ 的指定元素（由 index2 来决定是哪个元素）中, VRZ 的其余元素保持不变; 将源矢量寄存器 {VRX+1, VRX} 的某个元素（由 index1+1 来决定是哪个元素）插入目的矢量寄存器 VRZ+1 的指定元素（由 index2 来决定是哪个元素）中, VRZ+1 的其余元素保持不变; 将源矢量寄存器 {VRX+1, VRX} 的某个元素（由 index1+2 来决定是哪个元素）插入目的矢量寄存器 VRZ+2 的指定元素（由 index2 来决定是哪个元素）中, VRZ+2 的其余元素保持不变; 将源矢量寄存器 {VRX+1, VRX} 的某个元素（由 index1+3 来决定是哪个元素）插入目的矢量寄存器 VRZ+3 的指定元素（由 index2 来决定是哪个元素）中, VRZ+3 的其余元素保持不变。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op				26	25	24	21		20	19	16		15	7				6	5	4	3	0	
111110					I1	INDEX1			I0	VRX			00001001					1	INDEX2			VRZ		

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

编码表中 I1, I0 与 Imm3 组合在一起用来表征指令位宽类型和 Index，其中：

IMM[4:0] = {I1,I0,IMM3}；

if(IMM[4:3]=2’ b01) size=16, IMM[2:0] 表示 index0 到 7;

if(IMM[4:3]=2’ b10) size=32, IMM[2:0] 表示 index0 到 3 (高位为 0)

if(IMM[4:3]=2’ b00) size=8, IMM[2:0] 表示 index0 到 7;

if(IMM[4:3]=2’ b11) size=8, IMM[2:0] 表示 index8 到 15;

16.42 VITL.T.2——二元交织指令

统一化指令																														
语法	vitl.T.2 vrz, vrx (T=8/16/32)																													
操作	size = {8,16,32}; number = 128 / size; T(2i+1,2i)={Vx+1(i),Vx(i)}; i=0:(number-1) Vz=T(number-1:0); Vz+1=T((2*number-1):number);																													
编译结果	仅存在 32 位指令。 vitl.T.2 vrz, vrx																													
数据类型	8/16/32																													
说明	将源矢量寄存器 VRX 和 VRX+1 进行中的元素进行交织处理，并放入 VRZ 和 VRZ+1 中。下面以元素大小为半字举例：																													
	<table><tr><td>VRX</td><td>A7</td><td>A6</td><td>A5</td><td>A4</td><td>A3</td><td>A2</td><td>A1</td><td>A0</td></tr><tr><td>VRX1</td><td>B7</td><td>B6</td><td>B5</td><td>B4</td><td>B3</td><td>B2</td><td>B1</td><td>B0</td></tr></table>								VRX	A7	A6	A5	A4	A3	A2	A1	A0	VRX1	B7	B6	B5	B4	B3	B2	B1	B0				
	VRX	A7	A6	A5	A4	A3	A2	A1	A0																					
	VRX1	B7	B6	B5	B4	B3	B2	B1	B0																					
交叉以后得到如下结果								<table><tr><td>VRZ</td><td>B3</td><td>A3</td><td>B2</td><td>A2</td><td>B1</td><td>A1</td><td>B0</td><td>A0</td></tr><tr><td>VRZ1</td><td>B7</td><td>A7</td><td>B6</td><td>A6</td><td>B5</td><td>A5</td><td>B4</td><td>A4</td></tr></table>	VRZ	B3	A3	B2	A2	B1	A1	B0	A0	VRZ1	B7	A7	B6	A6	B5	A5	B4	A4				
VRZ	B3	A3	B2	A2	B1	A1	B0	A0																						
VRZ1	B7	A7	B6	A6	B5	A5	B4	A4																						
影响标志位	无影响																													
限制	寄存器的范围为 vr0-vr15。																													
异常	无																													

32 位指令																																					
操作	size = {8,16,32}; number = 128 / size; T(2i+1,2i)={Vx+1(i),Vx(i)}; i=0:(number-1) Vz=T(number-1:0); Vz+1=T((2*number-1):number);																																				
语法	vitl.T.2 vrz, vrx																																				
数据类型	8/16/32																																				
说明	将源矢量寄存器 VRX 和 VRX+1 进行中的元素进行去交织处理，并放入 VRZ 和 VRZ+1 中。下面以元素大小为半字举例： <table><tr><td>VRX</td><td>A7</td><td>A6</td><td>A5</td><td>A4</td><td>A3</td><td>A2</td><td>A1</td><td>A0</td></tr><tr><td>VRX1</td><td>B7</td><td>B6</td><td>B5</td><td>B4</td><td>B3</td><td>B2</td><td>B1</td><td>B0</td></tr></table> 交叉以后得到如下结果 <table><tr><td>VRZ</td><td>B3</td><td>A3</td><td>B2</td><td>A2</td><td>B1</td><td>A1</td><td>B0</td><td>A0</td></tr><tr><td>VRZ1</td><td>B7</td><td>A7</td><td>B6</td><td>A6</td><td>B5</td><td>A5</td><td>B4</td><td>A4</td></tr></table>	VRX	A7	A6	A5	A4	A3	A2	A1	A0	VRX1	B7	B6	B5	B4	B3	B2	B1	B0	VRZ	B3	A3	B2	A2	B1	A1	B0	A0	VRZ1	B7	A7	B6	A6	B5	A5	B4	A4
VRX	A7	A6	A5	A4	A3	A2	A1	A0																													
VRX1	B7	B6	B5	B4	B3	B2	B1	B0																													
VRZ	B3	A3	B2	A2	B1	A1	B0	A0																													
VRZ1	B7	A7	B6	A6	B5	A5	B4	A4																													
影响标志位	无影响																																				
限制	寄存器的范围为 vr0-vr15。																																				
异常	无																																				

32 位指令格式：

31	op	26	25	24	21	20	19	16	15		4	3	0		
111110				w1	Dont care		w0	VRX		000110101001				VRZ	

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

编码表中 VRX,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.43 VLD.T.N——固定长度矢量加载指令

统一化指令	
语法	vld.T.N vrz,(rx, offset) (T=8/16/32 N=1/2/3/4)
操作	从存储器加载固定多个字节/半字/字到矢量寄存器 VRZ 中 ** size = {8,16,32} num = {1,2,3,4} offset= imm7<< log2(size/8)) VRZ[size*num-1:0] ← MEM[RX + zero_extend(offset)]; VRZ[127:size*num] ← 0;
编译结果	仅存在 32 位指令。 vld.T.N vrz,(rx, offset)
说明	从存储器加载多个字节 /半字/字到矢量寄存器 VRZ 中，加载的个数由指令决定。如加载的数据不足 128 位，则 VRZ 的高位填充为 0。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移的 7 位相对偏移量无符号扩展到 32 位后的值得到（如加载类型为字节，则不左移，如加载类型为半字，则左移 1 位，如加载类型为字，则左移 2 位）。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	从存储器加载固定多个字节/半字/字到矢量寄存器 VRZ 中 size = {8,16,32} num = {1,2,3,4} offset= imm7<< log2(size/8)) VRZ[size*num-1:0] ← MEM[RX + zero_extend(offset)]; VRZ[127:size*num] ← 0;
语法	vld.T.N vrz,(rx, offset)
说明	从存储器加载多个字节/半字/字到矢量寄存器 VRZ 中，加载的个数由指令决定。如加载的数据不足 128 位，则 VRZ 的高位填充为 0。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移的 7 位相对偏移量无符号扩展到 32 位后的值得到（如加载类型为字节，则不左移，如加载类型为半字，则左移 1 位，如加载类型为字，则左移 2 位）。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	8	7	6	5	4	3	0
1		11110		IMM7H		RX		001100		IMM7L		NUM		SIZE		VRZ

16.44 VLDM.T——连续矢量加载指令

统一化指令	
语法	vldm.T vry-vrz, (rx) (T=8/16/32)
操作	从存储器加载连续的多个字节、半字、字到一片连续的矢量寄存器堆中 ** dst ← Y; addr ← RX; for (n = 0; n <= (Z-Y); n++){ VRdst[127:0] ← MEM[addr]; dst ← dst + 1; addr ← addr + 16; }
编译结果	仅存在 32 位指令。 vldm.T vry-vrz, (rx);
说明	从存储器依次加载连续的多个字节/半字/字到矢量寄存器 VRY 开始的一片连续寄存器堆中，即将存储器指定地址开始的第一个 128 位宽的多个字节/半字/字加载到矢量寄存器 VRY 中，第二个 128 位宽的多个字节/半字/字加载到寄存器 VR(Y+1) 中，依次类推，最后一个 128 位宽的多个字节/半字/字加载到寄存器 VRZ 中。存储器的有效地址由基址寄存器 RX 的内容决定。
影响标志位	无影响
限制	VRZ 应当大于等于 VRY。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常 **

32 位指令	
操作	从存储器加载连续的多个字节、半字、字到一片连续的矢量寄存器堆中 ** dst ← Y; addr ← RX; for (n = 0; n <= (Z-Y); n++){ VRdst[127:0] ← MEM[addr]; dst ← dst + 1; addr ← addr + 16; }
语法	vldm.T vry-vrz, (rx)
说明	从存储器依次加载连续的多个字节/半字/字到矢量寄存器 VR _Y 开始的一片连续寄存器堆中，即将存储器指定地址开始的第一个 128 位宽的多个字节/半字/字加载到矢量寄存器 VR _Y 中，第二个 128 位宽的多个字节/半字/字加载到寄存器 VR _(Y+1) 中，依次类推，最后一个 128 位宽的多个字节/半字/字加载到寄存器 VR _Z 中。存储器的有效地址由基址寄存器 RX 的内容决定。
影响标志位	无影响
限制	VR _Z 应当大于等于 VR _Y 。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	8	7	6	5	4	3	0
1	11110				RX		001000		11	IMM2	SIZE		VRZ			

注：

IMM2 域——指定目标寄存器的个数，IMM2 = Z - Y。

00——1 个目的寄存器

01——2 个目的寄存器

10——3 个目的寄存器

11——4 个目的寄存器

16.45 VLDMRU.T——跳跃矢量加载基址更新指令

统一化指令	
语法	vldmu.T vry-vrz, (rx),ry (T=8/16/32)
操作	从存储器加载连续的多个字节、半字、字到一片连续的矢量寄存器堆中，并更新基址。 dst ← Y; addr ← RX; for (n = 0; n <= (Z-Y); n++){ VRdst[127:0] ← MEM[addr]; dst ← dst + 1; addr ← addr + RY; } RX←addr
编译结果	仅存在 32 位指令。 vldmu.T vry-vrz, (rx), ry;
说明	从存储器依次加载跳跃的多个字节/半字/字到矢量寄存器 VR _Y 开始的一片连续寄存器堆中，即将存储器指定地址开始的第一个 128 位宽的多个字节/半字/字加载到矢量寄存器 VR _Y 中，基址加上偏移寄存器 RY 后的 128 位宽的多个字节/半字/字加载到寄存器 VR _(Y+1) 中，依次类推，最后一个 128 位宽的多个字节/半字/字加载到寄存器 VR _Z 中。存储器的有效地址由基址寄存器 RX 与每次的偏移寄存器的 RY 的内容决定。最后，更新基址寄存器 RX 为最后所加载地址再次加偏移寄存器 RY 的结果。
影响标志位	无影响
限制	VR _Z 应当大于等于 VR _Y 。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	从存储器加载连续的多个字节、半字、字到一片连续的矢量寄存器堆中，并更新基址。 dst ← Y; addr ← RX; for (n = 0; n <= (Z-Y); n++){ VRdst[127:0] ← MEM[addr]; dst ← dst + 1; addr ← addr + RY; } RX←addr
语法	vldmru.T vry-vrz, (rx), ry
说明	从存储器依次加载跳跃的多个字节/半字/字到矢量寄存器 VR _Y 开始的一片连续寄存器堆中，即将存储器指定地址开始的第一个 128 位宽的多个字节/半字/字加载到矢量寄存器 VR _Y 中，基址加上偏移寄存器 RY 后的 128 位宽的多个字节/半字/字加载到寄存器 VR _(Y+1) 中，依次类推，最后一个 128 位宽的多个字节/半字/字加载到寄存器 VR _Z 中。存储器的有效地址由基址寄存器 RX 与每次的偏移寄存器的 RY 的内容决定。最后，更新基址寄存器 RX 为最后所加载地址再次加偏移寄存器 RY 的结果。
影响标志位	无影响
限制	VR _Z 应当大于等于 VR _Y 。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常。

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	8	7	6	5	4	3	0
1	11110	00000		RX		001001		10	IMM2	SIZE					VRZ	

注：

IMM2 域——指定目标寄存器的个数，IMM2 = Z - Y。

00——1 个目的寄存器

01——2 个目的寄存器

10——3 个目的寄存器

11——4 个目的寄存器

16.46 VLDMU.T——连续矢量加载基址更新指令

统一化指令	
语法	vldmu.T vry-vrz, (rx) (T=8/16/32)
操作	从存储器加载连续的多个字节、半字、字到一片连续的矢量寄存器堆中，并更新基址 ** dst ← Y; addr ← RX; for (n = 0; n <= (Z-Y); n++){ VRdst[127:0] ← MEM[addr]; dst ← dst + 1; addr ← addr + 16; } RX←addr
编译结果	仅存在 32 位指令。 vldmu.T vry-vrz, (rx);
说明	从存储器依次加载连续的多个字节/半字/字到矢量寄存器 VRY 开始的一片连续寄存器堆中，即将存储器指定地址开始的第一个 128 位宽的多个字节/半字/字加载到矢量寄存器 VRY 中，第二个 128 位宽的多个字节/半字/字加载到寄存器 VR(Y+1) 中，依次类推，最后一个 128 位宽的多个字节/半字/字加载到寄存器 V RZ 中。存储器的有效地址由基址寄存器 RX 的内容决定。最后，更新基址寄存器 RX 为所有所加载数据之后的地址。
影响标志位	无影响
限制	VRZ 应当大于等于 VRY。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	从存储器加载连续的多个字节、半字、字到一片连续的矢量寄存器堆中，并更新基址 $dst \leftarrow Y; addr \leftarrow RX;$ for ($n = 0; n \leq (Z-Y); n++$){ $VRdst[127:0] \leftarrow MEM[addr];$ $dst \leftarrow dst + 1;$ $addr \leftarrow addr + 16;$ } $RX \leftarrow addr$
语法	$vldmu.T \ vry-vrz, (rx)$
说明	从存储器依次加载连续的多个字节/半字/字到矢量寄存器 VR _Y 开始的一片连续寄存器堆中，即将存储器指定地址开始的第一个 128 位宽的多个字节/半字/字加载到矢量寄存器 VR _Y 中，第二个 128 位宽的多个字节/半字/字加载到寄存器 VR _(Y+1) 中，依次类推，最后一个 128 位宽的多个字节/半字/字加载到寄存器 VR _Z 中。存储器的有效地址由基址寄存器 RX 的内容决定。最后，更新基址寄存器 RX 为所有所加载数据之后的地址。
影响标志位	无影响
限制	VR _Z 应当大于等于 VR _Y 。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	8	7	6	5	4	3	0
1	11110	00000		RX		001000		10	IMM2	SIZE					VRZ	

- 注：
- IMM2 域——指定目标寄存器的个数， $IMM2 = Z - Y$ 。
- 00——1 个目的寄存器
- 01——2 个目的寄存器
- 10——3 个目的寄存器
- 11——4 个目的寄存器

16.47 VLDRU.T.N——矢量加载基址跳跃更新指令

统一化指令	
语法	vldru.T.N vrz,(rx),ry (T=8/16/32 N=1/2/3/4)
操作	从存储器加载固定多个字节/半字/字到 VRZ 中并更新基址 ** size = {8,16,32} num = {1,2,3,4} VRZ[size*num-1:0] ← MEM[RX]; RX ← RX + RY
编译结果	仅存在 32 位指令。 vldru.T.N vrz,(rx),ry
说明	从存储器加载多个字节/半字/字到矢量寄存器 VRZ 中并更新基址，加载的个数由指令决定。如加载的数据不足 128 位，则 VRZ 的高位填充为 0。采用寄存器寻址方式。存储器的有效地址由基址寄存器 RX 得到。最后，更新基址寄存器 RX 为寄存器 RX 与 RY 的和。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	从存储器加载固定多个字节/半字/字到 VRZ 中并更新基址 ** size = {8,16,32} num = {1,2,3,4} VRZ[size*num-1:0] ← MEM[RX]; RX ← RX + RY
语法	vldru.T.N vrz,(rx),ry
说明	从存储器加载多个字节/半字/字到矢量寄存器 VRZ 中并更新基址，加载的个数由指令决定。如加载的数据不足 128 位，则 VRZ 的高位填充为 0。采用寄存器寻址方式。存储器的有效地址由基址寄存器 RX 得到。最后，更新基址寄存器 RX 为寄存器 RX 与 RY 的和。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	8	7	6	5	4	3	0
1	11110			RY		RX		001001		00		NUM	SIZE	VRZ		

16.48 VLDU.T.N——矢量加载基址更新指令

统一化指令	
语法	vldu.T.N vrz,(rx) (T=8/16/32 N=1/2/3/4)
操作	从存储器加载固定多个字节/半字/字到矢量寄存器 VRZ 中并更新基址 size = {8,16,32} num = {1,2,3,4} VRZ[size*num-1:0] ← MEM[RX]; RX ← RX + num*size/8
编译结果	仅存在 32 位指令。 vldu.T.N vrz,(rx)
说明	从存储器加载多个字节/半字/字到矢量寄存器 VRZ 中并更新基址，加载的个数由指令决定。如加载的数据不足 128 位，则 VRZ 的高位填充为 0。采用寄存器寻址方式。存储器的有效地址由基址寄存器 RX 得到。最后，更新基址寄存器 RX 为所加载数据之后的地址。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	从存储器加载固定多个字节/半字/字到 VRZ 中并更新基址 size = {8,16,32} num = {1,2,3,4} VRZ[size*num-1:0] ← MEM[RX]; RX ← RX + num*size/8
语法	vldu.T.N vrz,(rx)
说明	从存储器加载多个字节/半字/字到矢量寄存器 VRZ 中并更新基址，加载的个数由指令决定。如加载的数据不足 128 位，则 VRZ 的高位填充为 0。采用寄存器寻址方式。存储器的有效地址由基址寄存器 RX 得到。最后，更新基址寄存器 RX 为所加载数据之后的地址。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	8	7	6	5	4	3	0
1	11110			00000			RX	001000			00	NUM	SIZE	VRZ		

16.49 VLDX.T——可变长度矢量加载指令

统一化指令	
语法	vldx.T vrz,(rx),ry (T=8/16/32)
操作	从存储器加载多个字节/半字/字到矢量寄存器 VRZ 中 size = {8,16,32} num = mod(ry,128/size) if(num==0) then num =128/size VRZ[size*num-1:0] ← MEM[RX]; VRZ[127:size*num] ← 0;
编译结果	仅存在 32 位指令。 vldx.T vrz,(rx),ry
说明	从存储器加载多个字节/半字/字到矢量寄存器 VRZ 中，加载的个数由 RY 的后几位决定（如加载类型为字节，则由后 4 位决定；如加载类型为半字，则由后 3 位决定；如加载类型为字，则由后 2 位决定），如果 RY 的后几位为 0，则共加载 128 位数据。如加载的数据不足 128 位，则 VRZ 的高位填充为 0。采用寄存器寻址方式。存储器的有效地址由基址寄存器 RX 得到。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	从存储器加载多个字节/半字/字到矢量寄存器 VRZ 中 size = {8,16,32} num = mod(ry,128/size) if(num==0) then num =128/size VRZ[size*num-1:0] ← MEM[RX]; VRZ[127:size*num] ← 0;
语法	vldx.T vrz,(rx),ry
说明	从存储器加载多个字节/半字/字到矢量寄存器 VRZ 中，加载的个数由 RY 的后几位决定（如加载类型为字节，则由后 4 位决定；如加载类型为半字，则由后 3 位决定；如加载类型为字，则由后 2 位决定），如果 RY 的后几位为 0，则共加载 128 位数据。如加载的数据不足 128 位，则 VRZ 的高位填充为 0。采用寄存器寻址方式。存储器的有效地址由基址寄存器 RX 得到。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	8	7	6	5	4	3	0
1		11110		RY		RX		001001		01		00	SIZE		VRZ	

16.50 VLRW.32.N——矢量存储器读入指令

统一化指令	
语法	vlrw.32.1 vrz,imm32 vlrw.32.2 vrz,imm64 vlrw.32.3 vrz,imm96 vlrw.32.4 vrz,imm128
操作	从存储器加载多个字到矢量寄存器 $VRZ[len-1:0] \leftarrow MEM[(PC + zero\_extend(offset \ll 2)) \& 0xffffffff]**$ $VRZ[127:len] \leftarrow 0$
编译结果	仅存在 32 位指令。 vlrw.32.1 vrz,imm32 vlrw.32.2 vrz,imm64 vlrw.32.3 vrz,imm96 vlrw.32.4 vrz,imm128
说明	加载多个 32 位立即数至矢量寄存器 VRZ，如果立即数不足 128 位，则高位补 0。存储器地址根据 PC 加左移两位的相对偏移量，并无符号扩展到 32 位后，再经最低两位强制清零得到。VLRW 指令的加载范围是全部 4GB 地址空间。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	从存储器加载多个字到矢量寄存器 $VRZ[len-1:0] \leftarrow MEM[(PC + zero\_extend(offset \ll 2)) \& 0xffffffffc]$ $VRZ[127:len] \leftarrow 0$
语法	vlrw.32.1 vrz,imm32 vlrw.32.2 vrz,imm64 vlrw.32.3 vrz,imm96 vlrw.32.4 vrz,imm128
说明	加载多个 32 位立即数至矢量寄存器 VRZ，如果立即数不足 128 位，则高位补 0。存储器地址根据 PC 加左移两位的相对偏移量，并无符号扩展到 32 位后，再经最低两位强制清零得到。VLRW 指令的加载范围是全部 4GB 地址空间。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

31	30	26	25	16	15	10	9	8	7	6	5	4	3	0
1	11110	IMM12H				001101	IMM12L		NUM	10	VRZ			

16.51

VMASKI.8.L——矢量低半部分立即数扩展传输指令（按字节元素）

统一化指令	
语法	vmaski.8.l vrz, imm8
操作	size = 8; number = 128/size; Vz(i)= {8{imm8[i]}}; i=0:(number/2-1); Vz 其余元素置 0 vrz[i] 指相应 element 的数据;
编译结果	仅存在 32 位指令。 vmaski.8.l vrz, imm8
数据类型	8
说明	将 IMM8 每位扩展 8 存入 VRZ 低 64 位的每个 element 中。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = 8; number = 128/size; Vz(i)= {8{imm8[i]}}; i=0:(number/2-1); Vz 其余元素置 0 vrz[i] 指相应 element 的数据;
语法	vmaski.8.l vrz, imm8
数据类型	8
说明	将 IMM8 每位扩展 8 存入 VRZ 低 64 位的每个 element 中。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15		4	3	0	
111110		0	IMM4			0	IMM4			000011011010				VRZ

注：

编码表中 VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

两个 IMM4 拼成 1 个 IMM8 作为输入；

16.52 VMASKI.8.H——矢量高半部分立即数扩展传输指令（按字节元素）

统一化指令	
语法	vmaski.8.h vrz, imm8
操作	size = 8; number = 128/size; $Vz(i+8)=\{8\{imm8[i]\}\}; i=0:(number/2-1);$ Vz 其余元素保持不变 vrz[i] 指相应 element 的数据;
编译结果	仅存在 32 位指令。 vmaski.8.h vrz, imm8
数据类型	8
说明	将 IMM8 每位扩展 8 存入 VRZ 高 64 位的每个 element 中，其余位不变。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = 8; number = 128/size; $Vz(i+8)=\{8\{imm8[i]\}\}; i=0:(number/2-1);$ Vz 其余元素置 0 vrz[i] 指相应 element 的数据;
语法	vmaski.8.h vrz, imm8
数据类型	8
说明	将 IMM8 每位扩展 8 存入 VRZ 高 64 位的每个 element 中，其余位不变。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	4	3	0
	111110		0	IMM4		0	IMM4		000011011011			VRZ

注：

编码表中 VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

两个 IMM4 拼成 1 个 IMM8 作为输入；

16.53

VMASKI.16——矢量立即数扩展传输指令（按半字元素）

统一化指令	
语法	vmaski.16 vrz, imm8
操作	size = 16; number = 128/size; $V_z(i) = \{16\{imm8[i]\}\}; i=0:(number-1);$ vrz[i] 指相应 element 的数据;
编译结果	仅存在 32 位指令。 vmaski.16 vrz, imm8
数据类型	16
说明	将 IMM8 每位扩展 16 存入 VRZ 每个 element 中。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = 16; number = 128/size; $V_z(i) = \{16\{imm8[i]\}\}; i=0:(number-1);$ vrz[i] 指相应 element 的数据;
语法	vmaski.16 vrz, imm8
数据类型	16
说明	将 IMM8 每位扩展 16 存入 VRZ 每个 element。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	4	3	0
111110		0	IMM4		0	IMM4		000011011011			VRZ	

注：

编码表中 VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

两个 IMM4 拼成 1 个 IMM8 作为输入；

16.54 VMAX.T——矢量最大值选择指令

统一化指令	
语法	vmax.T vrz, vrx, vry (T=S8/S16/S32 /U8/U16/U32)
操作	Size = {8, 16, 32}; Number = 128 / size; for(i=0; i< number; i++) Vz(i) = max(Vx(i), Vy(i));
编译结果	仅存在 32 位指令。 vmax.T vrz, vrx, vry
数据类型	U8/U16/U32 S8/S16/S32
说明	选择 VRX 和 VRY 相应元素的较大值存入 VRZ 的对应元素中。
影响标志位	不影响
限制	寄存器的范围为 vr0-vr15。
异常	无

32 位指令	
操作	Size = {8, 16, 32}; Number = 128 / size; for(i=0; i< number; i++) Vz(i) = max(Vx(i), Vy(i));
语法	vmax.T vrz, vrx, vry
数据类型	U8/U16/U32 S8/S16/S32
说明	选择 VRX 和 VRY 相应元素的较大值存入 VRZ 的对应元素中。
影响标志位	不影响
限制	寄存器的范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15			5	4	3	0			
111110				w1		VRY		w0		VRX		00000100100				S	VRZ	

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2' b01 表示 size 为 16bit 指令；

[w1,w0] = 2' b10 表示 size 为 32bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.55 Vmfvr.T——矢量写传送指令

统一化指令	
语法	vmfvr.T rz, vr _x , index (T=S8/S16/S32 /U8/U16/U32)
操作	size = {8,16,32} if(unsigned){ rz = zero_extend(vr _x [(index+1) *size-1 : index*size]); }else{ rz = sign_extend(vr _x [(index+1) *size-1 : index*size]); }
编译结果	仅存在 32 位指令。 vmfvr.T rz, vr _x , index
数据类型	U8/U16/U32 S8/S16/S32
说明	将源矢量寄存器 VRX 的某个元素（由 index 来决定是哪个元素）扩展后传送到主处理器的目的寄存器 RZ 中，如果传输的是无符号数则高位扩展 0，如果是有符号数，则扩展符号位。如果传输的是 32 位，则无需进行扩展，直接传输给 RZ。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32}; if(unsigned){ rz = zero_extend(vrx[(index+1)*size-1 : index*size]); }else{ rz = sign_extend(vrx[(index+1)*size-1 : index*size]); } number = 128/ size; Index 范围为 [0,number-1]
语法	vmfvr.T rz, vrX, index
数据类型	U8/U16/U32 S8/S16/S32
说明	将源矢量寄存器 VRX 的某个元素（由 index 来决定是哪个元素）扩展后传送到主处理器的目的寄存器 RZ 中，如果传输的是无符号数则高位扩展 0，如果是有符号数，则扩展符号位。如果传输的是 32 位，则无需进行扩展，直接传输给 RZ。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15			6	5	4	0							
111110				w1	IMM4				w0	VRX				0001110010				S	VRZ			

注：

- 编码表中 w1, w0 用来表征指令位宽类型，其中：
- [w1,w0] = 2’ b00 表示 size 为 8bit 指令；
- [w1,w0] = 2’ b01 表示 size 为 16bit 指令；
- [w1,w0] = 2’ b10 表示 size 为 32bit 指令；
- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 VRX,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.56 VMIN.T——矢量最小值选择指令

统一化指令	
语法	vmin.T vrz, vrx, vry (T=S8/S16/S32 /U8/U16/U32)
操作	Size = {8, 16, 32}; Number = 128 / size; for(i=0; i< number; i++) Vz(i) = min(Vx(i), Vy(i));
编译结果	仅存在 32 位指令。 vmin.T vrz, vrx, vry
数据类型	U8/U16/U32 S8/S16/S32
说明	选择 VRX 和 VRY 相应元素的较小值存入 VRZ 的对应元素中。
影响标志位	不影响
限制	寄存器的范围为 vr0-vr15。
异常	无

32 位指令	
操作	Size = {8, 16, 32}; Number = 128 / size; for(i=0; i< number; i++) Vz(i) = min(Vx(i), Vy(i));
语法	vmin.T vrz, vrx, vry
数据类型	U8/U16/U32 S8/S16/S32
说明	选择 VRX 和 VRY 相应元素的较小值存入 VRZ 的对应元素中。
影响标志位	不影响
限制	寄存器的范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1		VRY		w0		VRX		00000100101		S	VRZ

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

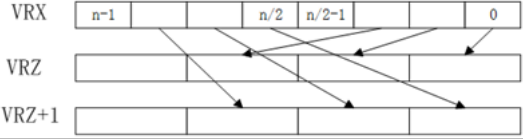
[w1,w0] = 2' b01 表示 size 为 16bit 指令；

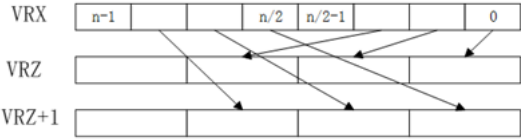
[w1,w0] = 2' b10 表示 size 为 32bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.57 VMOV.T.E——矢量扩展传输指令

统一化指令	
语法	vmov.T.e vrz, vrx (T=S8/S16/S32 /U8/U16/U32)
操作	size = {8,16,32} number = 128/2*size; i = 0:number-1; if(unsigned){ vrz[2i+1:2i] = zero_extend(vrx[i]); vrz+1[2i+1:2i]= zero_extend(vrx[i+number]); }else{ vrz[2i+1:2i] = sign_extend[i]; vrz+1[2i+1:2i]= sign_extend(vrx[i+number]); } vrx[i] 指相应 element 的数据; extend 根据 U/S 将值零扩展或者符号扩展为元素位宽的 2 倍。
编译结果	仅存在 32 位指令。 vmov.T.e vrz, vrx
数据类型	U8/U16/U32 S8/S16/S32
说明	将源矢量寄存器 VRX 的元素扩展为原位宽 2 倍之后传送到的目的寄存器 VRZ 和 VRZ+1 中，扩展后元素的位宽变为原来的 2 倍。 
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32} number = 128/2*size; i = 0:number-1; vr_x[i] 指相应位置 element 的数据; if(unsigned){ vr_z[2i+1:2i] = zero_extend(vr_x[i]); vr_z+1[2i+1:2i] = zero_extend(vr_x[i+number]); }else{ vr_z[2i+1:2i] = sign_extend[i]; vr_z+1[2i+1:2i] = sign_extend(vr_x[i+number]); } extend 根据 U/S 将值零扩展或者符号扩展为元素位宽的 2 倍。
语法	vmov.T.e vr _z , vr _x
数据类型	U8/U16/U32 S8/S16/S32
说明	将源矢量寄存器 VR_X 的元素扩展为原位宽 2 倍之后传送到的目的寄存器 VR_Z 和 VR_Z+1 中，扩展后元素的位宽变为原来的 2 倍。 
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15			5	4	3	0
111110		w1		X		w0		VRX				00001111010		S	VRZ

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2' b00 表示 size 为 8bit 指令；

[w1,w0] = 2' b01 表示 size 为 16bit 指令；

[w1,w0] = 2' b10 表示 size 为 32bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VR_X,VR_Y,VR_Z 分别表征这条指令各个矢量寄存器的寄存器号；

16.58 VMOV.T.H——矢量高位传输指令

统一化指令	
语法	<code>vmov.T.h vrz, vrz,vry</code> (T=16/32/64)
操作	<code>size = {16,32,64};</code> <code>number = 128/size;</code> <code>Vz(i)={Vx(2i+1)[size-1,size/2],Vx(2i)[size-1,size/2]};</code> <code>i=0:(number/2-1)</code> <code>Vz(number/2+i)={Vy(2i+1)[size-1,size/2],Vy(2i)[size-1,size/2]};</code> <code>i=0:(number/2-1)</code> 取每个元素的高半部分放入 VZ 中 <code>vrz[i]</code> 指相应 element 的数据;
编译结果	仅存在 32 位指令。 <code>vmov.T.h vrz, vrz,vry</code>
数据类型	16/32/64
说明	将源矢量寄存器 VRX 和 VRY 的元素分别取元素的高半部分存入 VRZ 中。 VRX VRZ+1 VRY
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {16,32, 64} number = 128/size; vr _x [i] 指相应位置 element 的数据; V _z (i)={V _x (2i+1)[size-1,size/2],V _x (2i)[size-1,size/2]}; i=0:(number/2-1) V _z (number/2+i)={V _y (2i+1)[size-1,size/2],V _y (2i)[size-1,size/2]}; i=0:(number/2-1) 取每个元素的高半部分
语法	vmov.T.h vr _z , vr _x ,vr _y
数据类型	16/32/64
说明	将源矢量寄存器 V R _X 和 V R _Y 的元素取高一半之后传送到的目的寄存器 V R _Z 中。 VR_X VR_Y VR_Z+1
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15		4	3	0
111110		w1	VRY		w0	VRX		000011000010				VRZ	

注：

- 编码表中 w1, w0 用来表征指令位宽类型，其中：
- [w1,w0] = 2’ b01 表示 size 为 16bit 指令；
- [w1,w0] = 2’ b10 表示 size 为 32bit 指令；
- [w1,w0] = 2’ b11 表示 size 为 64bit 指令；
- 编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.59 VMOV.T.L——矢量低位传输指令

统一化指令	
语法	vmov.T.l vrz, vrx,vry (T=16/32/64)
操作	size = {16,32,64}; number = 128/size; Vz(i)={Vx(2i+1)[size/2-1:0],Vx(2i)[size/2-1:0]}; i=0:(number/2-1) Vz(number/2+i)={Vy(2i+1)[size/2-1:0],Vy(2i)[size/2-1:0]}; i=0:(number/2-1) 取元素的低半部分 vrz[i] 指相应 element 的数据;
编译结果	仅存在 32 位指令。 vmov.T.l vrz, vrx,vry
数据类型	16/32/64
说明	将源矢量寄存器 VRX 和 VRY 的元素分别取元素的低半部分存入 VRZ 中。 VRX hi lo hi lo hi lo hi lo VRY hi lo hi lo hi lo hi lo VRZ+1 hi lo hi lo hi lo hi lo
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	$size = \{16, 32, 64\}$ $number = 128 / size;$ $vr_x[i]$ 指相应位置 element 的数据; $V_z(i) = \{V_x(2i+1)[size/2-1:0], V_x(2i)[size/2-1:0]\};$ $i = 0:(number/2-1)$ $V_z(number/2+i) = \{V_y(2i+1)[size/2-1:0], V_y(2i)[size/2-1:0]\};$ $i = 0:(number/2-1)$ 取元素的低半部分
语法	<code>vmov.T.l vrz, vr_x, vr_y</code>
数据类型	16/32/64
说明	将源矢量寄存器 VRX 和 VRY 的元素取低一半之后传送到的目的寄存器 VRZ 中。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式:

31	op	26	25	24	21	20	19	16	15			4	3	0
111110		w1		VRX		VRX		000011000000				VRZ		

注：

编码表中 w1, w0 用来表征指令位宽类型, 其中:

$[w1, w0] = 2'$ b01 表示 size 为 16bit 指令;

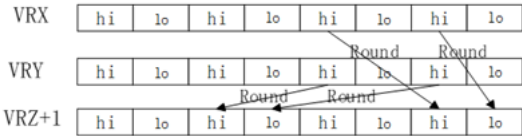
$[w1, w0] = 2'$ b10 表示 size 为 32bit 指令;

$[w1, w0] = 2'$ b11 表示 size 为 64bit 指令;

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号;

16.60 VMOV.T.RH——矢量高位 round 传输指令

统一化指令	
语法	vmov.T.rh vrz, vrx,vry (T=S16/S32/S64/U16/U32/U64)
操作	size = {16,32,64}; number = 128/size; round=1<<(size/2-1); Tmp1(i)=Vx(i)[size-1:size/2]+round; i=0:(number-1) Tmp2(i)=Vy(i)[size-1:size/2]+round; i=0:(number-1) Vz(i)={Tmp1(2i+1),Tmp1(2i)}; i=0:(number/2-1) Vz(i+number/2)={Tmp2(2i+1),Tmp2(2i)}; i=0:(number/2-1) 取每个元素的高半部分进行 round 操作之后放入 VZ 中 vrz[i] 指相应 element 的数据;
编译结果	仅存在 32 位指令。 vmov.T.rh vrz, vrx,vry
数据类型	U16/U32/U64 S16/S32/S64
说明	将源矢量寄存器 VRX 和 VRY 的元素分别取元素的高半部分然后进行 round 操作之后存入 VRZ 中。 VRX VRZ+1 VRY hi lo hi lo hi lo hi lo hi lo hi lo hi lo hi lo hi lo hi lo hi lo hi lo Round Round Round Round
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {16,32, 64} number = 128/size; vr_x[i],vr_y[i] 指相应位置 element 的数据; number = 128/size; round=1<<(size/2-1); Tmp1(i)=V_x(i)[size-1:size/2]+round; i=0:(number-1) Tmp2(i)=V_y(i)[size-1:size/2]+round; i=0:(number-1) V_z(i)={Tmp1(2i+1),Tmp1(2i)}; i=0:(number/2-1) V_z(i+number/2)={Tmp2(2i+1),Tmp2(2i)}; i=0:(number/2-1) 取每个元素高半部分之后进行 round 操作之后放入 VRZ 中
语法	vmov.T.rh vrz, vr _x ,vr _y
数据类型	U16/U32/U64 S16/S32/S64
说明	将源矢量寄存器 VRX 和 VRY 的元素取高一半 round 操作之后传送到的目的寄存器 VRZ 中。 
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15		5	4	3	0
111110		w1	VRY		w0	VRX		00001100011			s	VRZ		

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

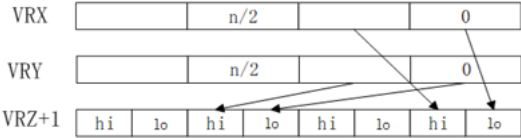
[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

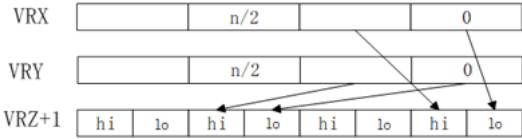
[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

[w1,w0] = 2’ b11 表示 size 为 64bit 指令；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.61 VMOV.T.SL——矢量低位饱和传输指令

统一化指令	
语法	vmov.T.sl vrz, vrx,vry (T=S16/S32/S64 /U16/U32/U64)
操作	size = {16,32,64} number = 128/size; 元素饱和至 size 的一半 signed=(T==S); Max=signed? 2^{^(size/2-1)}-1: 2^{^(size/2)}-1; Min=signed? -2^{^(size/2-1)}: 0; If Vx(i)>Max Tmp1(i)=Max; Else if Vx(i)<Min Tmp1(i)=Min; Else Tmp1(i)=Vx(i)[size/2-1:0]; End i=0:(number-1) If Vy(i)>Max Tmp2(i)=Max; Else if Vy(i)<Min Tmp2(i)=Min; Else Tmp2(i)=Vy(i)[size/2-1:0]; End i=0:(number-1) Vz(i)={Tmp1(2i+1),Tmp1(2i)}; i=0:(number/2-1) Vz(i+number/2)={Tmp2(2i+1),Tmp2(2i)}; i=0:(number/2-1) vrz[i] 指相应 element 的数据;
编译结果	仅存在 32 位指令。 vmov.T.sl vrz, vrx,vry
数据类型	U16/U32/U64 S16/S32/S64
说明	将源矢量寄存器 VRX 和 VRY 的元素分别取元素的低半部分存入 VRZ 中。 
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {16,32, 64} number = 128/size; vr_x[i] 指相应位置 element 的数据; signed=(T==S); Max=signed? 2^{^(size/2-1)}-1: 2^{^(size/2)}-1; Min=signed? -2^{^(size/2-1)}: 0; If V_x(i)>Max Tmp1(i)=Max; Else if V_x(i)<Min Tmp1(i)=Min; Else Tmp1(i)=V_x(i)[size/2-1:0]; End i=0:(number-1) If V_y(i)>Max Tmp2(i)=Max; Else if V_y(i)<Min Tmp2(i)=Min; Else Tmp2(i)=V_y(i)[size/2-1:0]; End i=0:(number-1) V_z(i)={Tmp1(2i+1),Tmp1(2i)}; i=0:(number/2-1) V_z(i+number/2)={Tmp2(2i+1),Tmp2(2i)}; i=0:(number/2-1) 元素饱和至 size 的一半
语法	vmov.T.sl vr _z , vr _x , vr _y
数据类型	U16/U32/U64 S16/S32/S64
说明	将源矢量寄存器 VR_X 和 VR_Y 的元素饱和至 size 的一半之后传送到的目的寄存器 VR_Z 中。 
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1	VR _Y		w0	VR _X		00001100010			S	VR _Z	

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

[w1,w0] = 2’ b11 表示 size 为 64bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX, VRY, VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.62

VMOVI.8——矢量立即数传输指令（按字节元素）

统一化指令	
语法	vmovi.8. vrz, imm8
操作	size = 8; number = 128/size; IMM8= imm8[7:0]; Vz(i)= IMM8; i=0:(number-1) 将立即数 imm8 放入 VZ 所有 element 中 vrz[i] 指相应 element 的数据;
编译结果	仅存在 32 位指令。 vmovi.8 vrz, imm8
数据类型	U8
说明	将立即数 imm8 放入 VZ 所有 element 中。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = 8; number = 128/size; IMM8= imm8[7:0]; Vz(i)= IMM8; i=0:(number-1) 将立即数 imm8 放入 VZ 所有 element 中 vrz[i] 指相应 element 的数据
语法	vmovi.8 vrz, imm8
数据类型	U8
说明	将立即数 imm8 放入 VZ 所有 element 中。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0	
111110		11		VRY		10		IMM4		000011011			S VRZ	

注：

编码表中 I1, I0 与 Imm 组合在一起用来表征指令位宽类型和 Index，其中：

IMM[2:0]={I1,I0,IMM} If(IMM[2:1]=10) size=8 ；

If(IMM[2:1]=11) size=16 IMM[0] 表示 index0 到 1；

If(IMM[2:1]=01) size=32 IMM[0] 表示 index0 到 1；

If(IMM[2:1]=00) size=32 IMM[0] 表示 index2 到 3；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.63 VMOVI.(U/S)16——矢量（有无符号）立即数传输指令（按半字元素）

统一化指令	
语法	vmovi.(u/s)16 vrz, imm8,index 例:vmovi.u16 vz, imm8,index 例:vmovi.s16 vz, imm8, index
操作	size = 16; number = 128/size; IMM16={8’ b0,imm8[7:0]}<<(index*8), (index= 0~1) If Type=U, Vz(i)= IMM16; i=0:(number-1) If Type=S, Vz(i)= ~IMM16; i=0:(number-1) vrz[i] 指相应 element 的数据;
编译结果	仅存在 32 位指令。 vmovi.(u/s)16 vrz, imm8,index
数据类型	U16 S16
说明	将 8 位立即数根据 index 放入 16 位全 0 立即数中，其余位保持不变，并根据符号位确定当前立即数是否需要取反，之后写入 VRZ 中。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = 16; number = 128/size; IMM16={8' b0,imm8[7:0]}<<(index*8), (index= 0~1) If Type=U, Vz(i)= IMM16; i=0:(number-1) If Type=S, Vz(i)= ~IMM16; i=0:(number-1) vr _x [i] 指相应 element 的数据;
语法	vmovi. (u/s)16 vr _z , imm8,index
数据类型	U16 S16
说明	将 8 位立即数根据 index 放入 16 位全 0 立即数中，其余位保持不变，并根据符号位确定当前立即数是否需要取反，之后写入 VRZ 中。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	8	7	6	5	4	3	0
111110			I1	VRY		I0	VRX		000011011			IMM	0	0	S	VRZ

注：

编码表中 I1，I0 与 Imm 组合在一起用来表征指令位宽类型和 Index，其中：

IMM[2:0]={I1,I0,IMM} If(IMM[2:1]=10) size=8 ；

If(IMM[2:1]=11) size=16 IMM[0] 表示 index0 到 1；

If(IMM[2:1]=01) size=32 IMM[0] 表示 index0 到 1；

If(IMM[2:1]=00) size=32 IMM[0] 表示 index2 到 3；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.64 VMOVI.(U/S)32——矢量（有无符号）立即数传输指令（按字元素）

统一化指令	
语法	vmovi.(u/s)32 vrz, imm8,index 例:vmovi.u32 vz, imm8, index 例:vmovi.s32 vz, imm8, index
操作	size = 32; number = 128/size; IMM32={24' b0,imm8[7:0]}<<(index*8) (index= 0~3) If Type=U, Vz(i)= IMM32; i=0:(number-1) If Type=S, Vz(i)= ~IMM32; i=0:(number-1) vrz[i] 指相应 element 的数据;
编译结果	仅存在 32 位指令。 vmovi.(u/s)32 vrz, imm8,index
数据类型	U32 S32
说明	将 IMM8 左移 8*index 位之后根据符号确认是否取反存入 VRZ 中。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = 32; number = 128/size; IMM32={24' b0,imm8[7:0]}<<(index*8) (index= 0~3) If Type=U, Vz(i)= IMM32; i=0:(number-1) If Type=S, Vz(i)= ~IMM32; i=0:(number-1) vrz[i] 指相应 element 的数据;
语法	vmovi. (u/s)32 vrz, imm8,index
数据类型	U32 S32
说明	将 IMM8 左移 8*index 位之后根据符号确认是否取反存入 VRZ 中。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	8	7	6	5	4	3	0
111110		1		VRY	0		VRX		000011011	IMM	0	0	S		VRZ	

注：

编码表中 I1, I0 与 Imm 组合在一起用来表征指令位宽类型和 Index，其中：

IMM[2:0]={I1,I0,IMM} If(IMM[2:1]=10) size=8 ；

If(IMM[2:1]=11) size=16 IMM[0] 表示 index0 到 1；

If(IMM[2:1]=01) size=32 IMM[0] 表示 index0 到 1；

If(IMM[2:1]=00) size=64 IMM[0] 表示 index2 到 3；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.65 VMTVR.T.1——矢量写传送指令

统一化指令	
语法	vmtvr.T.1 vz, rx,index (T=8/16/32)
操作	size = {8,16,32} Vz(index)=Rx[size-1:0], 其余元素不变 Index 范围为 0~(128/size -1)
编译结果	仅存在 32 位指令。 vmtvr.T.1 vz, rx,index
数据类型	8/16/32
说明	将主流水线寄存器 RX 的数据根据 size 传送到矢量寄存器 VRZ 中的某个元素中去，选择哪个元素由 index 来决定。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32}; vrz[(index+1)* size-1 : index*size] = rx[size-1:0] number = 128/size; Index 范围为 [0,number-1]
语法	vmtvr.T.1 vz, rx,index
数据类型	8/16/32
说明	将主流水线寄存器 RX 的数据根据 size 选择低位传送到矢量寄存器 VRZ 中的某个元素中去，选择哪个元素由 index 来决定。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	16	15	5	4	3	0
111110			w1	IMM4		RX		00011100000			w0	VRZ

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.66 VMTVR.T.2——矢量写传送指令

统一化指令	
语法	vmtvr.T.2 vz, rx,index (T=8/16/32)
操作	size = {8,16,32} Vz(index)=Rx[size-1:0], Vz(index+1)=Rx+1[size-1:0], 其余元素不变 Index 范围为 0~(128/size -2)
编译结果	仅存在 32 位指令。 vmtvr.T.2 vz, rx,index
数据类型	8/16/32
说明	将主流水线寄存器 RX 和 RX+1 的数据根据 size 传送到矢量寄存器 VRZ 中的连续的两个元素中去，选择哪个两个连续的元素由 index 来决定。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32}; vrz[(index+1)* size-1 : index* size] = rx[size -1:0]; vrz[(index+2)* size-1 : (index+1)* size] = rx+1[size -1:0]; number = 128/ size; Index 范围为 [0,number-2]
语法	vmtvr.T.2 vz, rx,index
数据类型	8/16/32
说明	将主流水线寄存器 RX 和 Rx+1 的数据根据 size 选择低位传送到矢量寄存器 VRZ 中的两个连续的元素中去，选择哪个两个连续的元素由 index 来决定。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	16	15	5	4	3	0			
111110			w1	IMM4			RX			00011100001			w0	VRZ	

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

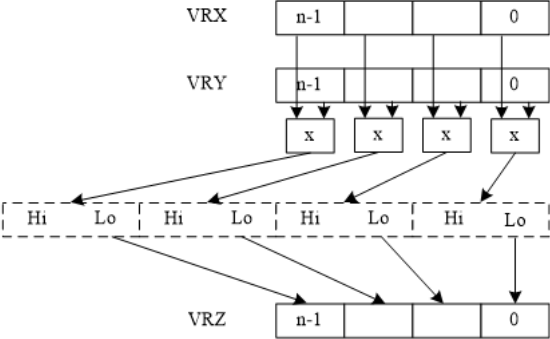
[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.67 VMUL.T——矢量乘法指令

统一化指令	
语法	vmul.T vrz, vrx, vry (T=S8/S16/S32 /U8/U16/U32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] for(i=0; i< Number; i++) { Vrz(i) = Lo[Vrx(i)*Vry(i)]; }
编译结果	仅存在 32 位指令。 vmul.T vrz, vrx, vry
数据类型	U8/U16/U32 S8/S16/S32
说明	将 VRX 与 VR Y 的对应元素相乘，取乘法结果的低半部分存放于 VRZ 中。 
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] for(i=0; i< Number; i++) { Vrz(i) = Lo[Vrx(i)*Vry(i)]; }
语法	vmul.T vrz, vrx, vry
数据类型	U8/U16/U32 S8/S16/S32
说明	将 VRX 与 VR Y 的对应元素相乘，取乘法结果的低半部分存放于 VRZ 中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

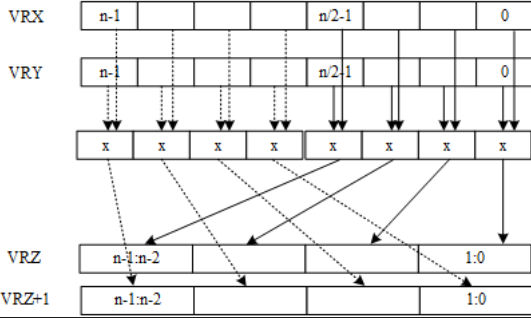
32 位指令格式：

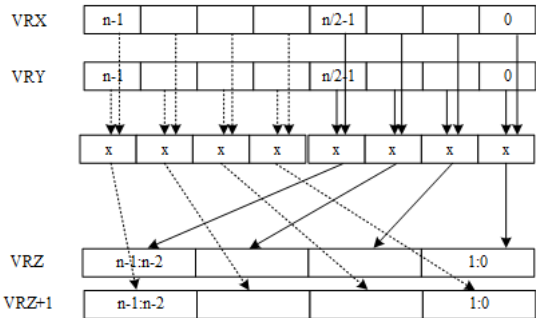
31	op	26	25	24	21	20	19	16	15	5	4	3	0			
111110				w1	VRY			w0	VRX			0111100000		S	VRZ	

注：

- 编码表中 w1, w0 用来表征指令位宽类型，其中：
- [w1,w0] = 2’ b00 表示 size 为 8bit 指令；
- [w1,w0] = 2’ b01 表示 size 为 16bit 指令；
- [w1,w0] = 2’ b10 表示 size 为 32bit 指令；
- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.68 VMUL.T.E——矢量扩展乘法指令

统一化指令	
语法	vmul.T.e vrz, vrx,vry (T=S8/S16/S32 /U8/U16/U32)
操作	Number=128/size i 表征第几个元素， 范围为 [0,Number-1] for(i=0; i< Number/2; i++) { Vz(2i+1:2i)=(Vx(i)*Vy(i)); Vz+1(2i+1:2i)= (Vx(Number/2+i)*Vy(Number/2+i)); }
编译结果	仅存在 32 位指令。 vmul.T.e vrz, vrx, vry
数据类型	U8/U16/U32 S8/S16/S32
说明	将 VRX 与 VRY 的对应元素相乘，乘法结果保留全部精度存放于 VRZ 和 VRZ+1 中。 
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] for(i=0; i< Number/2; i++) { Vz(2i+1:2i)=(Vx(i)*Vy(i)); Vz+1(2i+1:2i)= (Vx(Number/2+i)*Vy(Number/2+i)); }
语法	vmul.eT vrz, vrx, vry
数据类型	U8/U16/U32 S8/S16/S32
说明	将 VRX 与 VRY 的对应元素相乘，乘法结果保留全部精度存放于 VRZ 和 VRZ+1 中。 
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0	
111110		w1	VRY			w0	VRX			01111100000			S	VRZ

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

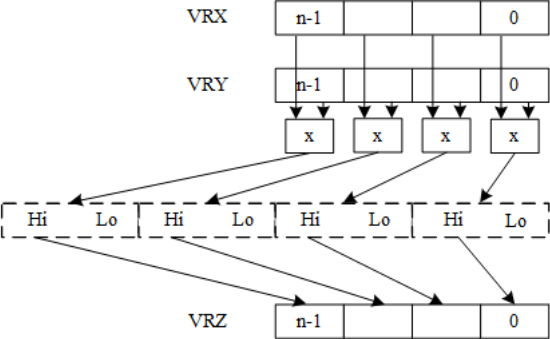
[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

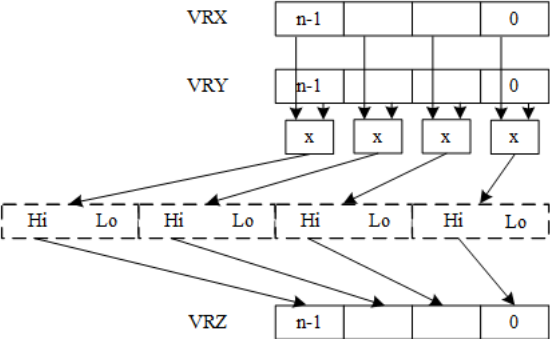
[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.69 VMUL.T.H——矢量取高半乘法指令

统一化指令	
语法	vmul.T.h vrz, vrx, vry (T=S8/S16/S32 /U8/U16/U32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] for(i=0; i< Number; i++) { Vrz(i) = High[Vrx(i)*Vry(i)]; }
编译结果	仅存在 32 位指令。 vmul.T.h vrz, vrx, vry
数据类型	U8/U16/U32 S8/S16/S32
说明	将 VRX 与 VRY 的对应元素相乘，取乘法结果的高半部分存放于 VRZ 中。 
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] for(i=0; i< Number; i++) { Vrz(i) = High[Vrx(i)*Vry(i)]; }
语法	vmul.T.h vrz, vrx, vry
数据类型	U8/U16/U32 S8/S16/S32
说明	将 VRX 与 VRY 的对应元素相乘，取乘法结果的高半部分存放于 VRZ 中。 
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

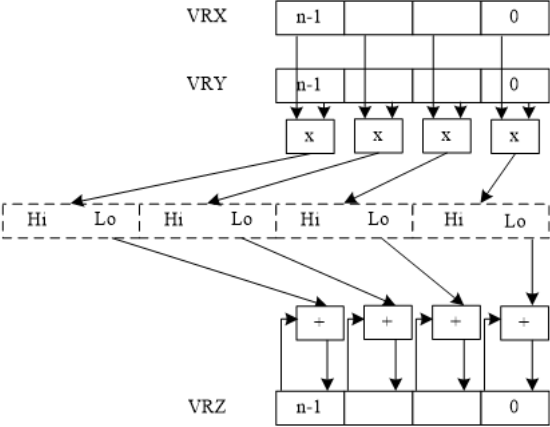
31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1	VRY			w0	VRX			01111000001			
										S	VRZ		

注：

- 编码表中 w1, w0 用来表征指令位宽类型，其中：
- [w1,w0] = 2’ b00 表示 size 为 8bit 指令；
- [w1,w0] = 2’ b01 表示 size 为 16bit 指令；
- [w1,w0] = 2’ b10 表示 size 为 32bit 指令；
- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.70 VMULA.T——矢量乘累加指令

统一化指令	
语法	vmula.T vrz, vrx, vry (T=S8/S16/S32 /U8/U16/U32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] for(i=0; i< Number; i++) { Vz(i)=Vz(i)+Lo[Vx(i)*Vy(i)]; }
编译结果	仅存在 32 位指令。 vmula.T vrz, vrx, vry
数据类型	U8/U16/U32 S8/S16/S32
说明	将 VRX 与 VRY 的对应元素相乘，取乘法结果的低半部分与 VRZ 的对应元素相加，最后结果存放于 VRZ 中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] for(i=0; i< Number; i++) { Vz(i)=Vz(i)+Lo[Vx(i)*Vy(i)]; }
语法	vmula.T vrz, vrx, vry
数据类型	U8/U16/U32 S8/S16/S32
说明	将 VRX 与 VRY 的对应元素相乘，取乘法结果的低半部分与 VRZ 的对应元素相加，最后结果存放于 VRZ 中。 
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1	VRY		w0	VRX		01111010000			S	VRZ	

注：

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

[w1,w0] = 2’ b11 表示 size 为 64bit 指令；

16.71 VMULA.T.E——矢量扩展乘累加指令

统一化指令	
语法	vmula.T.e vrz, vr_x, vr_y (T=S8/S16/S32/U8/U16/U32)
操作	Number=128/size i 表征第几个元素, 范围 [0,Number-1] for(i=0; i< Number/2; i++) { V_z(2i+1:2i)=V_z(2i+1:2i) + (V_x(i)*V_y(i)); V_z+1(2i+1:2i)=V_z+1(2i+1:2i) + (V_x(Number/2+i)*V_y(Number/2+i)); }
编译结果	仅存在 32 位指令。 vmula.T.e vrz,vr_x,vr_y
数据类型	U8/U16/U32 S8/S16/S32
说明	将 V_{RX} 与 V_{RY} 的对应元素相乘, 乘法结果保留全部精度与 V_{RZ} 和 V_{RZ}+1 的对应元素相加, 最后结果存放于 V_{RZ} 和 V_{RZ}+1 中。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] for(i=0; i< Number/2; i++) { $V_z(2i+1:2i)=V_z(2i+1:2i) + (V_x(i)*V_y(i));$ $V_{z+1}(2i+1:2i)=V_{z+1}(2i+1:2i) +$ $(V_x(\text{Number}/2+i)*V_y(\text{Number}/2+i));$ }
语法	vmula.T.e vrz, vrx, vry
数据类型	U8/U16/U32 S8/S16/S32
说明	将 V RX 与 VRY 的对应元素相乘，乘法结果保留全部精度与 VRZ 和 VRZ+1 的对应元素相加，最后结果存放于 VRZ 和 VRZ+1 中。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15		5	4	3	0
	111110	w1		VRY	w0		VRX		01111110000		S		VRZ	

注：

- 编码表中 w1, w0 用来表征指令位宽类型，其中：
- [w1,w0] = 2’ b00 表示 size 为 8bit 指令；
- [w1,w0] = 2’ b01 表示 size 为 16bit 指令；
- [w1,w0] = 2’ b10 表示 size 为 32bit 指令；
- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.72 VMULACA.T——矢量链乘累加指令

统一化指令	
语法	vmulaca.T vrz, vrx, vry (T=S8/S16/U8/U16)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] for(i=0; i< Number; i++) { Tmp(i)=(Vx(i)*Vy(i)); } for(i=0; i< Number/4; i++) { Vz(4i+3:4i)=Extend(Tmp[4i+3] +Tmp[4i+2]+Tmp[4i+1]+Tmp[4i]); }
编译结果	仅存在 32 位指令。 vmulaca.T vrz, vrx, vry
数据类型	U8/U16/ S8/S16/
说明	将 VRX 与 VRY 的对应元素相乘，乘法结果保留全部精度（源操作数元素位宽的 2 倍），再从低到高，将连续的 4 个元素相乘得到的乘法结果累加并扩展至源操作数元素位宽的 4 倍。结果保存到 VRZ 寄存器中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] for(i=0; i< Number; i++) { Tmp(i)=(Vx(i)*Vy(i)); } for(i=0; i< Number/4; i++) { Vz(4i+3: 4i)=Extend(Tmp[4i+3]+Tmp[4i+2]+Tmp[4i+1]+Tmp[4i]); }
语法	vmulaca.T vrz, vrX, vry
数据类型	U8/U16/ S8/S16/
说明	将 VRX 与 VRY 的对应元素相乘，乘法结果保留全部精度（源操作数元素位宽的 2 倍），再从低到高，将连续的 4 个元素相乘得到的乘法结果累加并扩展至源操作数元素位宽的 4 倍。结果保存到 VRZ 寄存器中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1	VRY		w0	VRX		01111000010			S	VRZ	

注：

- 编码表中 w1, w0 用来表征指令位宽类型，其中：
- [w1,w0] = 2’ b00 表示 size 为 8bit 指令；
- [w1,w0] = 2’ b01 表示 size 为 16bit 指令；
- [w1,w0] = 2’ b10 表示 size 为 32bit 指令；
- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.73 VMULACAA.T——矢量链乘累加指令

统一化指令	
语法	vmulacaa.T vrz, vrx, vry (T=S8/S16/U8/U16)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] for(i=0; i< Number; i++) { Tmp(i)=(Vx(i)*Vy(i)); } for(i=0; i< Number/4; i++) { Vz(4i+3:4i)=Vz(4i+3:4i)+ Extend(Tmp[4i+3]+Tmp[4i+2] +Tmp[4i+1]+Tmp[4i]); }
编译结果	仅存在 32 位指令。 vmulacaa.T vrz, vrx, vry
数据类型	U8/U16/ S8/S16/
说明	将 VRX 与 VRY 的对应元素相乘，乘法结果保留全部精度（源操作数元素位宽的 2 倍），再从低到高，将连续的 4 个元素相乘得到的乘法结果累加并扩展至源操作数元素位宽的 4 倍，最后将累加的结果与 VRZ 中对应的值相加，结果保存到 VRZ 寄存器中
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] for(i=0; i< Number; i++) { Tmp(i)=(Vx(i)*Vy(i)); } for(i=0; i< Number/4; i++) { Vz(4i+3:4i)=Vz(4i+3: 4i)+Extend(Tmp[4i+3]+Tmp[4i+2]+Tmp[4i+1]+Tmp[4i]); }
语法	vmulacaa.T vrz, vrx, vry
数据类型	U8/U16/ S8/S16/
说明	将 VRX 与 VRY 的对应元素相乘，乘法结果保留全部精度（源操作数元素位宽的 2 倍），再从低到高，将连续的 4 个元素相乘得到的乘法结果累加并扩展至源操作数元素位宽的 4 倍，最后将累加的结果与 VRZ 中对应的值相加，结果保存到 VRZ 寄存器中
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1	VRY		w0	VRX		01111011000			S	VRZ	

注：

- 编码表中 w1, w0 用来表征指令位宽类型，其中：
- [w1,w0] = 2’ b00 表示 size 为 8bit 指令；
- [w1,w0] = 2’ b01 表示 size 为 16bit 指令；
- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.74 VMULACAAI.T——矢量带索引链乘累加指令

统一化指令	
语法	vmulacaa.T vrz, vrx, vry,index (T=S8/S16/U8/U16)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Index 范围为 [0,Number/4-1] for(i=0; i< Number/4; i++) { Ttmp(4i)=(Vx(4i)*Vy(4*index)); Ttmp(4i+1)=(Vx(4i+1)*Vy(4*index+1)) Ttmp(4i+2)=(Vx(4i+2)*Vy(4*index+2)) Ttmp(4i+3)=(Vx(4i+3)*Vy(4*index+3)) Vz(4i+3:4i)=Vz(4i+3:4i)+Extend(Ttmp[4i+3]+Ttmp[4i+2]+Ttmp[4i+1]+Ttmp[4i]); }
编译结果	仅存在 32 位指令。 vmulacaa.T vrz, vrx, vry,index
数据类型	U8/U16/ S8/S16/
说明	将 VRX 与 index 所索引 VRY 的对应元素相乘，乘法结果保留全部精度（源操作数元素位宽的 2 倍），再从低到高，将连续的 4 个元素相乘得到的乘法结果累加并扩展至源操作数元素位宽的 4 倍，最后将累加的结果与 VRZ 中对应的值相加，结果保存到 VRZ 寄存器中
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Index 范围为 [0,Number/4-1] for(i=0; i< Number/4; i++) { Tmp(4i)=(Vx(4i)*Vy(4*index)); Tmp(4i+1)=(Vx(4i+1)*Vy(4*index+1)) Tmp(4i+2)=(Vx(4i+2)*Vy(4*index+2)) Tmp(4i+3)=(Vx(4i+3)*Vy(4*index+3)) Vz(4i+3:4i)=Vz(4i+3:4i)+Extend(Tmp[4i+3]+Tmp[4i+2]+Tmp[4i+1]+Tmp[4i]); }
语法	vmulacaai.T vrz, vrx, vry,index
数据类型	U8/U16/ S8/S16/
说明	将 VRX 与 index 所索引 VRY 的对应元素相乘，乘法结果保留全部精度（源操作数元素位宽的 2 倍），再从低到高，将连续的 4 个元素相乘得到的乘法结果累加并扩展至源操作数元素位宽的 4 倍，最后将累加的结果与 VRZ 中对应的值相加，结果保存到 VRZ 寄存器中
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	8	7	6	5	4	3	0
111110		I1	VRY		I0	VRX		01000011		IMM	0	0	S	VRZ		

注：

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

编码表中 I1, I0 与 Imm 组合在一起用来表征指令位宽类型和 Index，其中：

IMM[2:0] = {I1,I0,IMM}；

if(IMM[2:1]=2’ b01) size=16, IMM[0] 表示 index0 到 1；

if(IMM[2:1]=2’ b00) size=8, IMM[0] 表示 index0 到 1；

if(IMM[2:1]=2’ b11) size=8, IMM[0] 表示 index2 到 3；

16.75 VMULACAI.T——矢量带索引链乘累加指令

统一化指令	
语法	vmulacai.T vrz, vrx, vry,index (T=S8/S16/U8/U16)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Index 范围为 [0,Number/4-1] for(i=0; i< Number/4; i++) { Ttmp(4i)=(Vx(4i)*Vy(4*index)); Ttmp(4i+1)=(Vx(4i+1)*Vy(4*index+1)) Ttmp(4i+2)=(Vx(4i+2)*Vy(4*index+2)) Ttmp(4i+3)=(Vx(4i+3)*Vy(4*index+3)) Vz(4i+3:4i)=Extend(Ttmp[4i+3]+Ttmp[4i+2]+Ttmp[4i+1]+Ttmp[4i]); }
编译结果	仅存在 32 位指令。 vmulacai.T vrz, vrx, vry,index
数据类型	U8/U16/ S8/S16/
说明	将 VRX 与 index 所索引 VRY 的对应元素相乘，乘法结果保留全部精度（源操作数元素位宽的 2 倍），再从低到高，将连续的 4 个元素相乘得到的乘法结果累加并扩展至源操作数元素位宽的 4 倍。结果保存到 VRZ 寄存器中
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Index 范围为 [0,Number/4-1] for(i=0; i< Number/4; i++) { Tmp(4i)=(Vx(4i)*Vy(4*index)); Tmp(4i+1)=(Vx(4i+1)*Vy(4*index+1)) Tmp(4i+2)=(Vx(4i+2)*Vy(4*index+2)) Tmp(4i+3)=(Vx(4i+3)*Vy(4*index+3)) Vz(4i+3: 4i)=Extend(Tmp[4i+3]+Tmp[4i+2]+Tmp[4i+1]+Tmp[4i]); }
语法	vmulacai.T vrz, vrx, vry,index
数据类型	U8/U16/ S8/S16/
说明	将 VRX 与 index 所索引 VRY 的对应元素相乘，乘法结果保留全部精度（源操作数元素位宽的 2 倍），再从低到高，将连续的 4 个元素相乘得到的乘法结果累加并扩展至源操作数元素位宽的 4 倍。结果保存到 VRZ 寄存器中
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	8	7	6	5	4	3	0
111110			I1	VRY		I0	VRX		01000010			IMM	0	0	S	VRZ

注：

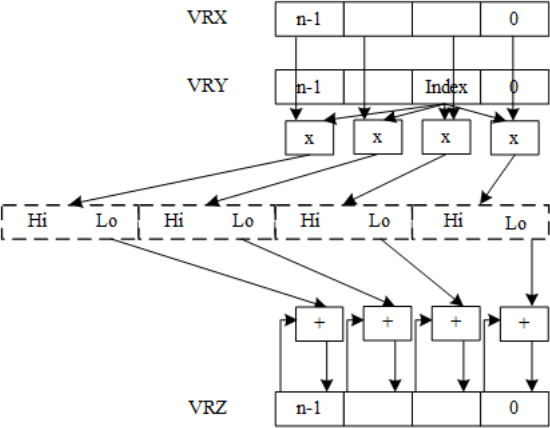
编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

编码表中 I1, I0 与 Imm 组合在一起用来表征指令位宽类型和 Index，其中：

- IMM[2:0] = {I1,I0,IMM};
- if(IMM[2:1]=2’ b01) size=16, IMM[0] 表示 index0 到 1;
- if(IMM[2:1]=2’ b00) size=8, IMM[0] 表示 index0 到 1;
- if(IMM[2:1]=2’ b11) size=8, IMM[0] 表示 index2 到 3;

16.76 VMULAI.T——矢量带索引乘累加指令

统一化指令	
语法	vmulai.T vrz, vrx, vry,index (T=S8/S16/S32 /U8/U16/U32)
操作	Number=128/size i 表征第几个元素, [0,Number-1] Index 范围为 [0,number-1] for(i=0; i< Number; i++) { Vz(i)=Vz(i)+Lo[Vx(i)*Vy(index)]; }
编译结果	仅存在 32 位指令。 vmulai.T vrz, vrx, vry,index
数据类型	U8/U16/U32 S8/S16/S32
说明	将 VRX 与 index 所索引 VRY 的对应元素相乘，取乘法结果的 的低半部分与 VRZ 的对应元素相加，最后结果存放于 VRZ 中。 
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Index 范围为 [0,number-1] for(i=0; i< Number; i++) { Vz(i)=Vz(i)+Lo[Vx(i)*Vy(index)]; }
语法	vmulai.T vrz, vrx, vry,index
数据类型	U8/U16/U32 S8/S16/S32
说明	将 VRX 与 index 所索引 VRY 的对应元素相乘，取乘法结果的低半部分与 VRZ 的对应元素相加，最后结果存放于 VRZ 中。 The diagram shows three registers: VRX, VRY, and VRZ. VRX and VRY are n-bit registers. An 'Index' value is used to select an element 'x' from VRY. This element 'x' is multiplied by the corresponding element from VRX. The result is then added to the corresponding element in VRZ. The diagram shows this process for multiple elements, with 'Hi' and 'Lo' parts of the results being combined.
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	8	7	5	4	3	0	
111110		I1		VRY		I0		VRX		01010000			IMM3		S	VRZ

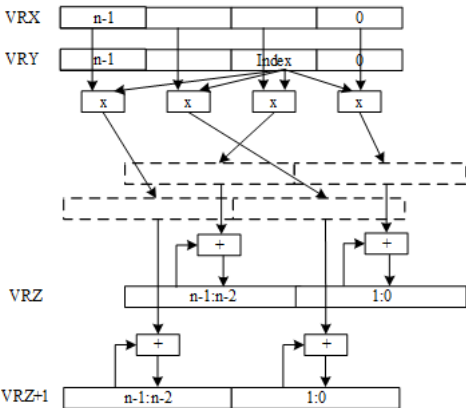
注：

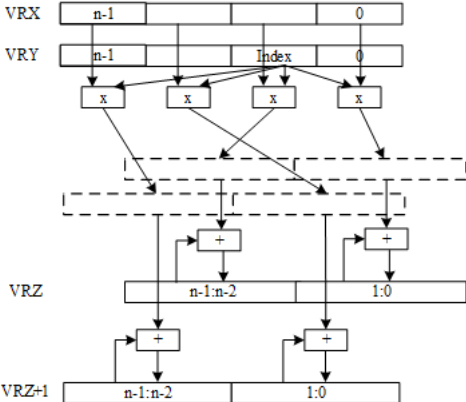
- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；
- 编码表中 I1, I0 与 Imm3 组合在一起用来表征指令位宽类型和 Index，其中：
IMM[4:0] = {I1,I0,IMM3};
if(IMM[4:3]=2’ b01) size=16, IMM[2:0] 表示 index0 到 7;
if(IMM[4:3]=2’ b10) size=32, IMM[2:0] 表示 index0 到 3（高位为 0）

if(IMM[4:3]=2' b00) size=8, IMM[2:0] 表示 index0 到 7;

if(IMM[4:3]=2' b11) size=8, IMM[2:0] 表示 index8 到 15;

16.77 VMULAI.T.E——矢量扩展带索引乘累加指令

统一化指令	
语法	vmulai.T.e vrz, vrx, vry,index (T=S8/S16/S32/U8/U16/U32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Index 范围为 [0,number-1] for(i=0; i< Number/2; i++) { Vz(2i+1:2i)=Vz(2i+1:2i)+ (Vx(i)*Vy(index)); Vz+1(2i+1:2i)=Vz+1(2i+1:2i) + (Vx(Number/2+i)*Vy(index)); }
编译结果	仅存在 32 位指令。 vmulai.T.e vrz, vrx, vry,index
数据类型	U8/U16/U32 S8/S16/S32
说明	将 VRX 与 index 所索引 VRY 的对应元素相乘，乘法结果保留全部精度与 VRZ 和 VRZ+1 的对应元素相加，最后结果存放于 VRZ 和 VRZ+1 中。 
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Index 范围为 [0,number-1] for(i=0; i< Number/2; i++) { $V_z(2i+1:2i)=V_z(2i+1:2i)+$ $(V_x(i)*V_y(index));$ $V_{z+1}(2i+1:2i)=V_{z+1}(2i+1:2i) +$ $(V_x(\text{Number}/2+i)*V_y(index));$ }
语法	vmulai.T.e vrz, vrx, vry,index
数据类型	U8/U16/U32 S8/S16/S32
说明	将 VRX 与 index 所索引 VRY 的对应元素相乘，乘法结果保留全部精度与 VRZ 和 VRZ+1 的对应元素相加，最后结果存放于 VRZ 和 VRZ+1 中。 
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	8	7	5	4	3	0
111110		I1		VRY	I0		VRX	01110010			IMM3	S	VRZ		

注：

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

编码表中 I1, I0 与 Imm3 组合在一起用来表征指令位宽类型和 Index，其中：

IMM[4:0] = {I1,I0,IMM3}；

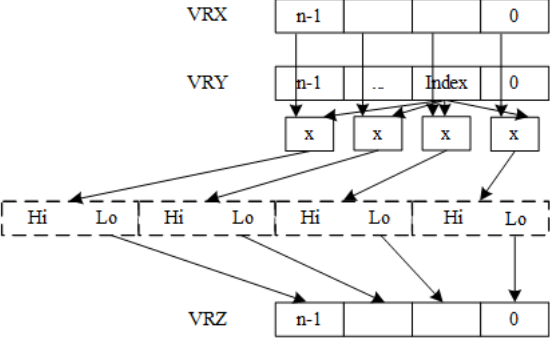
if(IMM[4:3]=2’ b01) size=16, IMM[2:0] 表示 index0 到 7;

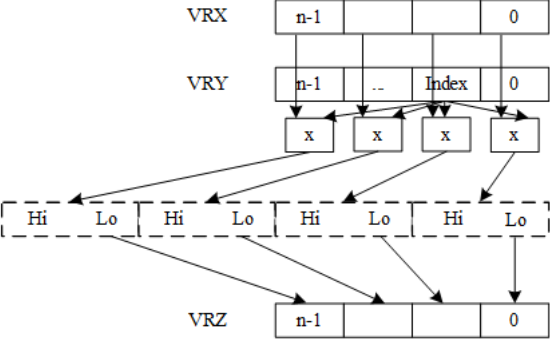
if(IMM[4:3]=2’ b10) size=32, IMM[2:0] 表示 index0 到 3 (高位为 0)

if(IMM[4:3]=2’ b00) size=8, IMM[2:0] 表示 index0 到 7;

if(IMM[4:3]=2’ b11) size=8, IMM[2:0] 表示 index8 到 15;

16.78 VMULI.T——矢量带索引乘法指令

统一化指令	
语法	vmuli.T vrz,vrx,vry,index (T=S8/S16/S32 /U8/U16/U32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Index 范围为 [0,number-1] for(i=0; i< Number; i++) { Vrz(i) = Lo[Vrx(i)*Vry(index)]; }
编译结果	仅存在 32 位指令。 vmuli.T vrz, vrx, vry,index
数据类型	U8/U16/U32 S8/S16/S32
说明	将 VRX 与 index 所索引 VRY 的对应元素相乘，取乘法结果的低半部分存放于 VRZ 中。 
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Index 范围为 [0,number-1] for(i=0; i< Number; i++) { Vrz(i) = Lo[Vrx(i)*Vry(index)]; }
语法	vmuli.T vrz, vrx, vry, index
数据类型	U8/U16/U32 S8/S16/S32
说明	将 VRX 与 index 所指向的 VRY 的对应元素相乘,取乘法结果的低半部分存放于 VRZ 中。 
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	8	7	5	4	3	0		
111110		I1		VRY		I0		VRX		01000000			IMM3		S	VRZ	

注：

- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；
- 编码表中 I1, I0 与 Imm3 组合在一起用来表征指令位宽类型和 Index，其中：

IMM[4:0] = {I1,I0,IMM3};
if(IMM[4:3]=2’ b01) size=16, IMM[2:0] 表示 index0 到 7;
if(IMM[4:3]=2’ b10) size=32, IMM[2:0] 表示 index0 到 3 (高位为 0)
if(IMM[4:3]=2’ b00) size=8, IMM[2:0] 表示 index0 到 7;
if(IMM[4:3]=2’ b11) size=8, IMM[2:0] 表示 index8 到 15;

16.79 VMULI.T.E——矢量扩展带索引乘法指令

统一化指令	
语法	vmuli.T.e vrz, vr_x, vr_y, index (T=S8/S16/S32 /U8/U16/U32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Index 范围为 [0,number-1] for(i=0; i< Number/2; i++) { V_z(2i+1:2i)=(V_x(i)*V_y(index)); V_z+1(2i+1:2i)= (V_x(Number/2+i)*V_y(index)); }
编译结果	仅存在 32 位指令。 vmuli.T.e vrz, vr_x, vr_y, index
数据类型	U8/U16/U32 S8/S16/S32
说明	将 VRX 与 Index 索引 VRY 的对应元素相乘，乘法结果保留全部精度存放于 VRZ 和 VRZ+1 中。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Index 范围为 [0,number-1] for(i=0; i< Number/2; i++) { Vz(2i+1:2i)=(Vx(i)*Vy(i)); Vz+1(2i+1:2i)= (Vx(Number/2+i)*Vy(Number/2+i)); }
语法	vmuli.T.e vrz, vrx,vry, index
数据类型	U8/U16/U32 S8/S16/S32
说明	将 VRX 与 Index 索引 VRY 的对应元素相乘，乘法结果保留全部精度存放于 VRZ 和 VRZ+1 中。 VRX VRZ VRZ+1 VRY Diagram description: The diagram illustrates the data flow for the vmuli instruction. It shows four registers: VRX, VRY, VRZ, and VRZ+1. VRX and VRY are n-bit registers (n-1 to 0). VRZ and VRZ+1 are n-bit registers (n-1:n-2 to 1:0). An 'Index' field is shown within the VRY register. Arrows indicate that elements from VRX and VRY are selected based on the Index and multiplied. The results are then stored in VRZ and VRZ+1. Specifically, VRX[i] * VRY[Index + i] results are stored in VRZ[2i+1:2i] and VRZ+1[2i+1:2i].
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	8	7	5	4	3	0
111110		I1	VRY		I0	VRX		01100000			IMM3		S	VRZ	

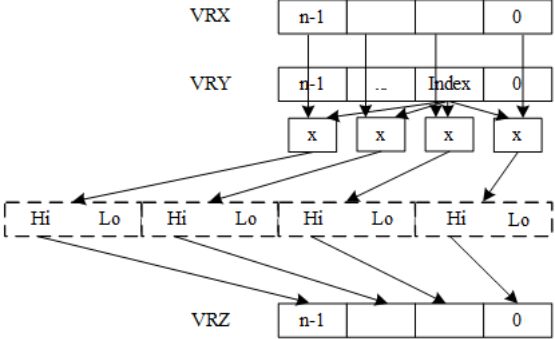
注：

- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；
- 编码表中 I1, I0 与 Imm3 组合在一起用来表征指令位宽类型和 Index，其中：
 $IMM[4:0] = \{I1,I0,IMM3\}$;
if(IMM[4:3]=2’ b01) size=16, IMM[2:0] 表示 index0 到 7;
if(IMM[4:3]=2’ b10) size=32, IMM[2:0] 表示 index0 到 3（高位为 0）
if(IMM[4:3]=2’ b00) size=8, IMM[2:0] 表示 index0 到 7;

if(IMM[4:3]=2’ b11) size=8, IMM[2:0] 表示 index8 到 15;

16.80 VMULI.T.H——矢量带索引取高半乘法指令

统一化指令	
语法	vmuli.T.h vrz,vrx,vry,index (T=S8/S16/S32 /U8/U16/U32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Index 范围为 [0,number-1] for(i=0; i< Number; i++) { Vrz(i) = High[Vrx(i)*Vry(index)]; }
编译结果	仅存在 32 位指令。 vmuli.T.h vrz, vrz, vry,index
数据类型	U8/U16/U32 S8/S16/S32
说明	将 VRX 与 index 所索引 VRY 的对应元素相乘，取乘法结果的高半部分存放于 VRZ 中。 VRX VRY VRZ
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Index 范围为 [0,number-1] for(i=0; i< Number; i++) { Vrz(i) = High[Vrx(i)*Vry(index)]; }
语法	vmuli.T.h vrz, vrx, vry, index
数据类型	U8/U16/U32 S8/S16/S32
说明	将 VRX 与 index 所指向的 VR Y 的对应元素相乘，取乘法结果的高半部分存放于 VRZ 中。 
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

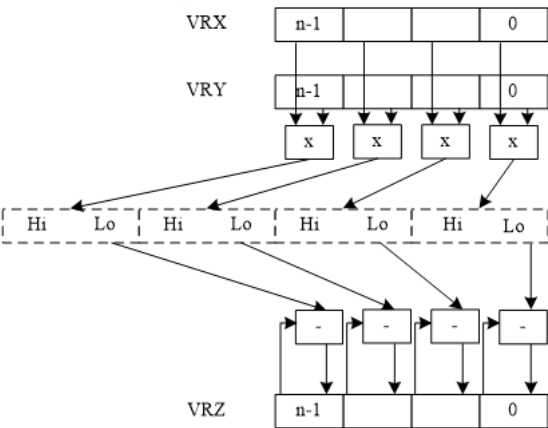
31	op	26	25	24	21	20	19	16	15	8	7	5	4	3	0	
111110		I1		VRY		I0		VRX		01000001			IMM3		S	VRZ

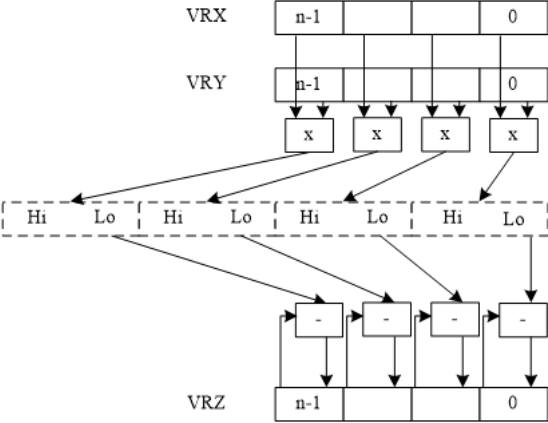
注：

- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；
- 编码表中 I1, I0 与 Imm3 组合在一起用来表征指令位宽类型和 Index，其中：

```
IMM[4:0] = {I1,I0,IMM3};  
if(IMM[4:3]=2' b01) size=16, IMM[2:0] 表示 index0 到 7;  
if(IMM[4:3]=2' b10) size=32, IMM[2:0] 表示 index0 到 3（高位为 0）  
if(IMM[4:3]=2' b00) size=8, IMM[2:0] 表示 index0 到 7;  
if(IMM[4:3]=2' b11) size=8, IMM[2:0] 表示 index8 到 15;
```

16.81 VMULS.T——矢量乘累减指令

统一化指令	
语法	vmuls.T vrz, vrx, vry (T=S8/S16/S32 /U8/U16/U32)
操作	Number=128/size i 表征第几个元素，[0,Number-1] for(i=0; i< Number; i++) { Vz(i)=Vz(i)-Lo[Vx(i)*Vy(i)]; }
编译结果	仅存在 32 位指令。 vmuls.T vrz, vrx, vry
数据类型	U8/U16/U32 S8/S16/S32
说明	将 VRX 与 VRY 的对应元素相乘，取结果的低位并与 VRZ 的对应元素相减，最后结果存放于 VRZ 中。其中，VRZ 中的元素为被减数，乘法结果作为减数。 
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素, 范围为 [0,Number-1] for(i=0; i< Number; i++) { Vz(i)=Vz(i)-Lo[Vx(i)*Vy(i)]; }
语法	vmuls.T vrz, vrx, vry
数据类型	U8/U16/U32 S8/S16/S32
说明	将 VRX 与 VRY 的对应元素相乘, 取结果的低位并与 VRZ 的对应元素相减, 最后结果存放于 VRZ 中。其中, VRZ 中的元素为被减数, 乘法结果作为减数。 
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

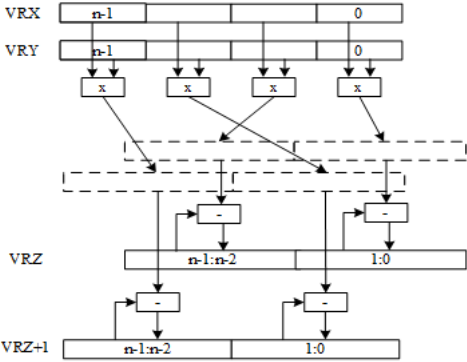
32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1	VRY		w0	VRX		01111010001			S	VRZ	

注：

- 编码表中 w1, w0 用来表征指令位宽类型, 其中:
- [w1,w0] = 2' b00 表示 size 为 8bit 指令;
- [w1,w0] = 2' b01 表示 size 为 16bit 指令;
- [w1,w0] = 2' b10 表示 size 为 32bit 指令;
- 编码表中 S 表征带符号运算还是无符号运算: 1 表示带符号, 0 表示无符号;
- 编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号;

16.82 VMULS.T.E——矢量扩展乘累减指令

统一化指令	
语法	vmuls.T.e vrz, vr_x, vr_y (T=S8/S16/S32 /U8/U16/U32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] for(i=0; i< Number/2; i++) { $V_z(2i+1:2i)=V_z(2i+1:2i)- (V_x(i)*V_y(i));$ $V_{z+1}(2i+1:2i)=V_{z+1}(2i+1:2i) -$ $(V_x(\text{Number}/2+i)*V_y(\text{Number}/2+i));$ }
编译结果	仅存在 32 位指令。 vmuls.T.e vrz, vr_x, vr_y
数据类型	U8/U16/U32 S8/S16/S32
说明	将 VR_X 与 VR_Y 的对应元素相乘，乘法结果保留全部精度与 VR_Z 和 VR_Z+1 的对应元素相减，最后结果存放于 VR_Z 和 VR_Z+1 中。其中，VR_Z 和 VR_Z+1 中的元素为被减数，乘法结果作为减数。 
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] for(i=0; i< Number/2; i++) { $V_z(2i+1:2i)=V_z(2i+1:2i)- (V_x(i)*V_y(i));$ $V_{z+1}(2i+1:2i)=V_{z+1}(2i+1:2i) -$ $(V_x(\text{Number}/2+i)*V_y(\text{Number}/2+i));$ }
语法	vmuls.T.e vrz, vrz, vrx, vry
数据类型	U8/U16/U32 S8/S16/S32
说明	将 VRX 与 VRY 的对应元素相乘，乘法结果保留全部精度与 VRZ 和 VRZ+1 的对应元素相减，最后结果存放于 VRZ 和 VRZ+1 中。其中，VRZ 和 VRZ+1 中的元素为被减数，乘法结果作为减数。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1	VRY			w0	VRX			01111110001		S	VRZ

注：

编码表中 w1，w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

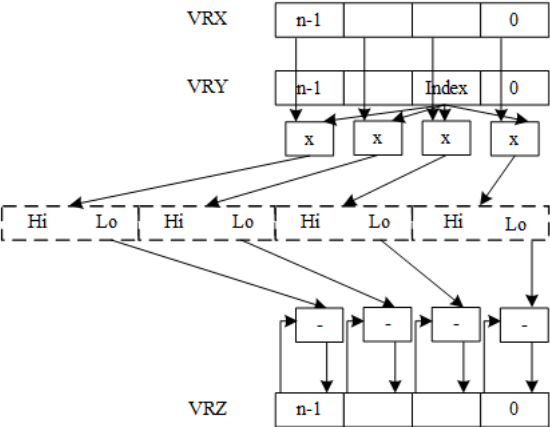
[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

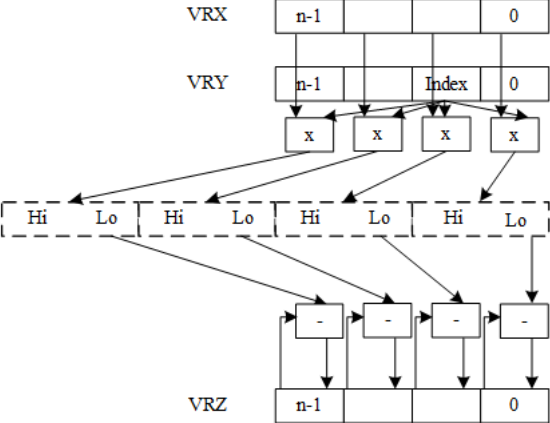
[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.83 VMULSI.T——矢量带索引乘累减指令

统一化指令	
语法	vmulsi.T vrz, vrx, vry,index (T=S8/S16/S32 /U8/U16/U32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Index 范围为 [0,number-1] for(i=0; i< Number; i++) { Vz(i)=Vz(i)-Lo[Vx(i)*Vy(index)]; }
编译结果	仅存在 32 位指令。 vmulsi.T vrz, vrx, vry,index
数据类型	U8/U16/U32 S8/S16/S32
说明	将 VRX 与 Index 所索引 VRY 的对应元素相乘，取结果的低位并与 VRZ 的对应元素相减，最后结果存放于 VRZ 中。其中，VRZ 中的元素为被减数，乘法结果作为减数。 
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Index 范围为 [0,number-1] for(i=0; i< Number; i++) { Vz(i)=Vz(i)-Lo[Vx(i)*Vy(index)]; }
语法	vmulsi.T vrz, vrx, vry,index
数据类型	U8/U16/U32 S8/S16/S32
说明	将 VRX 与 Index 所索引 VRY 的对应元素相乘，取结果的低位并与 VRZ 的对应元素相减，最后结果存放于 VRZ 中。其中，VRZ 中的元素为被减数，乘法结果作为减数。 
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	8	7	5	4	3	0
111110		I1	VRY		I0	VRX		01010001			IMM3		S	VRZ	

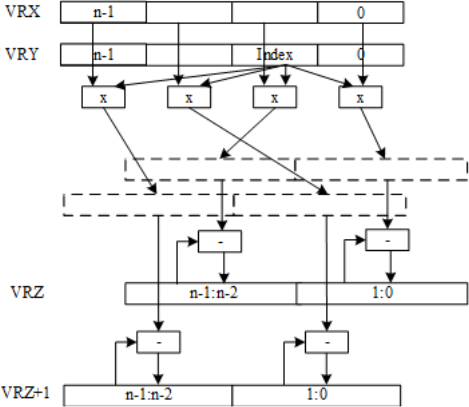
注：

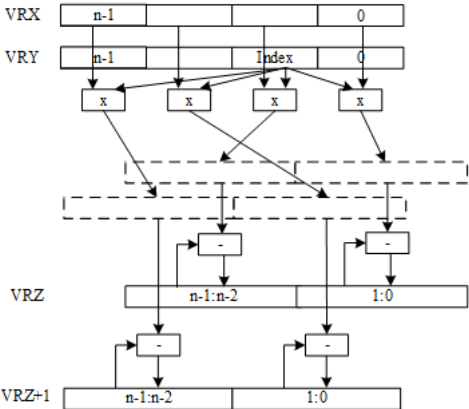
- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；
- 编码表中 I1, I0 与 Imm3 组合在一起用来表征指令位宽类型和 Index，其中：
IMM[4:0] = {I1,I0,IMM3};
if(IMM[4:3]=2’ b01) size=16, IMM[2:0] 表示 index0 到 7;
if(IMM[4:3]=2’ b10) size=32, IMM[2:0] 表示 index0 到 3（高位为 0）

if(IMM[4:3]=2' b00) size=8, IMM[2:0] 表示 index0 到 7;

if(IMM[4:3]=2' b11) size=8, IMM[2:0] 表示 index8 到 15;

16.84 VMULSI.T.E——矢量带索引扩展乘累减指令

统一化指令	
语法	vmulsi.T.e vrz, vrz, vry,index (T=S8/S16/S32 /U8/U16/U32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Index 范围为 [0,number-1] for(i=0; i< Number/2; i++) { Vz(2i+1:2i)=Vz(2i+1:2i)- (Vx(i)*Vy(index)); Vz+1(2i+1:2i)=Vz+1(2i+1:2i) - (Vx(Number/2+i)*Vy(index)); }
编译结果	仅存在 32 位指令。 vmulsi.T.e vrz, vrz, vry,index
数据类型	U8/U16/U32 S8/S16/S32
说明	将 VRX 与 Index 所索引 VRY 的对应元素相乘，乘法结果保留全部精度与 VRZ 和 VRZ+1 的对应元素相减，最后结果存放于 VRZ 和 VRZ+1 中。其中，VRZ 和 VRZ+1 中的元素为被减数，乘法结果作为减数。 
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] for(i=0; i< Number/2; i++) { $V_z(2i+1:2i)=V_z(2i+1:2i)- (V_x(i)*V_y(i));$ $V_{z+1}(2i+1:2i)=V_{z+1}(2i+1:2i) -$ $(V_x(\text{Number}/2+i)*V_y(\text{Number}/2+i));$ }
语法	vmulsi.T.e vrz, vrx, vry,index
数据类型	U8/U16/U32 S8/S16/S32
说明	将 VRX 与 Index 所索引 VRY 的对应元素相乘，乘法结果保留全部精度与 VRZ 和 VRZ+1 的对应元素相减，最后结果存放于 VRZ 和 VRZ+1 中。其中，VRZ 和 VRZ+1 中的元素为被减数，乘法结果作为减数。 
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	8	7	5	4	3	0	
111110			I1	VRY		I0	VRX		01110011			IMM3		S	VRZ	

注：

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

编码表中 I1, I0 与 Imm3 组合在一起用来表征指令位宽类型和 Index，其中：

$IMM[4:0] = \{I1,I0,IMM3\};$

if(IMM[4:3]=2' b01) size=16, IMM[2:0] 表示 index0 到 7;

if(IMM[4:3]=2' b10) size=32, IMM[2:0] 表示 index0 到 3 (高位为 0)

if(IMM[4:3]=2' b00) size=8, IMM[2:0] 表示 index0 到 7;

if(IMM[4:3]=2' b11) size=8, IMM[2:0] 表示 index8 到 15;

16.85 VNEG.T——矢量取反指令

统一化指令	
语法	vneg.T vrz, vrx (T=S8/S16/S32)
操作	size = {8,16,32}; Number = 128 / size; for(i=0; i< number; i++) { Vz(i) = ~Vx(i)+1; }
编译结果	仅存在 32 位指令。 vneg.T vrz, vrx
数据类型	S8/S16/S32
说明	将源矢量寄存器 VRX 的每个有符号元素求反后，结果存放于目的矢量寄存器 VRZ 中的对应元素。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr15
异常	无

32 位指令	
操作	size = {8,16,32}; Number = 128 / size; for(i=0; i< number; i++) { Vz(i) = ~Vx(i)+1; }
语法	vneg.T vrz, vrx
数据类型	S8/S16/S32
说明	将源矢量寄存器 VRX 的每个有符号元素求反后，结果存放于目的矢量寄存器 VRZ 中的对应元素。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr15
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0	
111110		w1	0		w0	VRX				00000111000			1	VRZ

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

编码表中 VRX,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.86 VNEG.T.S——矢量饱和取反指令

统一化指令	
语法	vneg.T.s vrz, vrx (T=S8/S16/S32)
操作	size = {8,16,32}; Number = 128 / size; for(i=0; i< number; i++) { Vz(i) = Sat(~Vx(i)+1); }
编译结果	仅存在 32 位指令。 vneg.T.s vrz, vrx
数据类型	S8/S16/S32
说明	将源矢量寄存器 VRX 的每个有符号元素求反并取饱和, 结果存放于目的矢量寄存器 VRZ 的对应元素中。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr15
异常	无

32 位指令	
操作	size = {8,16,32}; Number = 128 / size; for(i=0; i< number; i++) { Vz(i) = Sat(~Vx(i)+1); }
语法	vneg.T.s vrz, vrx
数据类型	S8/S16/S32
说明	将源矢量寄存器 VRX 的每个有符号元素求反并取饱和, 结果存放于目的矢量寄存器 VRZ 的对应元素中。
影响标志位	无影响
限制	寄存器的范围为 vr0-vr15
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1	0		w0	VRX		00000111001			1	VRZ	

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

编码表中 VRX,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.87 VNOT.T——矢量取反指令

统一化指令	
语法	vnot.T vrz,vrx (T=8/16/32/64)
操作	for j=0:127 Vz[j]=!Vx[j] vrz[j],vry[j],vrz[j] 指操作数比特位 j 的数据;
编译结果	仅存在 32 位指令。 vnot.T vrz, vrz
数据类型	8/16/32/64
说明	将 Vrx 数据按比特位取反并写入 Vrz 相应的比特位之中。 注：T 在此并无实际意义，仅为用户使用方便。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	for j=0:127 Vz[j]=!Vx[j] vrz[j],vry[j],vrz[j] 指操作数比特位 j 的数据;
语法	vnot.T vrz, vrz
数据类型	8/16/32/64
说明	将 Vrx 数据按比特位取反并写入 Vrz 相应的比特位之中。注：T 在此并无实际意义，仅为用户使用方便。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	4	3	0
111110		x	VRY		x	VRX		000110001000			VRZ	

注：

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号

16.88 VOR.T——矢量或指令

统一化指令	
语法	vor.T vrz,vrx,vry (T=8/16/32/64)
操作	for j=0:127 Vz[j]=Vx[j] Vy[j] vrz[j],vry[j],vrz[j] 指操作数比特位 j 的数据;
编译结果	仅存在 32 位指令。 vor.T vrz, vrz,vry
数据类型	8/16/32/64
说明	将 Vrx,Vry 相应比特位数据做或操作并写入 Vrz 相应的比特位之中。 注：T 在此并无实际意义，仅为用户使用方便。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	for j=0:127 Vz[j]=Vx[j] Vy[j] vrz[j],vry[j],vrz[j] 指操作数比特位 j 的数据;
语法	vor.T vrz, vrz,vry
数据类型	U8/U16/U32/U64
说明	将 Vrx,Vry 相应比特位数据做或操作并写入 Vrz 相应的比特位之中。 注：T 在此并无实际意义，仅为用户使用方便。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	4	3	0
111110		x	VRY		x	VRX		000110000011			VRZ	

注：

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号

16.89 VORN.T——矢量非或指令

统一化指令	
语法	vorn.T vrz,vrx,vry (T=8/16/32/64)
操作	for j=0:127 Vz[j]=Vx[j] !Vy[j] vrz[j],vry[j],vrz[j] 指操作数比特位 j 的数据;
编译结果	仅存在 32 位指令。 vorn.T vrz, vrz,vry
数据类型	8/16/32/64
说明	将 Vry 相应比特位取反，与 Vrx 相应比特位做或操作并写入 Vrz 相应的比特位之中。注： T 在此并无实际意义，仅为用户使用方便。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	for j=0:127 Vz[j]=Vx[j] !Vy[j] vrz[j],vry[j],vrz[j] 指操作数比特位 j 的数据;
语法	vorn.T vrz, vrz,vry
数据类型	8/16/32/64
说明	将 Vry 相应比特位取反，与 Vrx 相应比特位做或操作并写入 Vrz 相应的比特位之中。注： T 在此并无实际意义，仅为用户使用方便。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

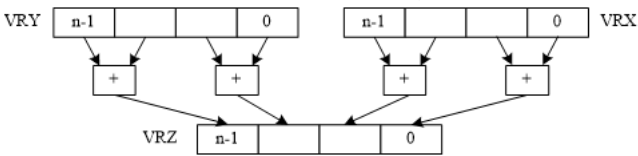
32 位指令格式：

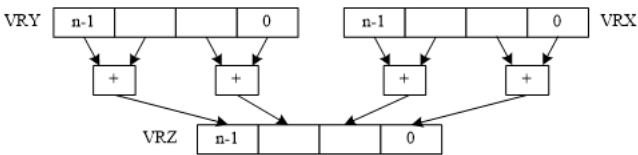
31	op	26	25	24	21	20	19	16	15	4	3	0
111110		x	VRY		x	VRX		000110000100			VRZ	

注：

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号

16.90 VPADD.T——矢量链加法指令

统一化指令	
语法	vpadd.T vrz, vrx, vry (T=S8/S16/S32 /U8/U16/U32)
操作	size = {8,16,32,64}; Number=128/size for(i=0; i< Number/2; i++) { Vz(i)=Vx(2i)+Vx(2i+1); Vz(number/2+i)=Vy(2i)+Vy(2i+1); }
编译结果	仅存在 32 位指令。 vpadd.T vrz, vrx, vry
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	VRX 与 VRY 中的相邻元素相加，结果存放于 VRZ 中。 
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32,64}; Number=128/size for(i=0; i< Number/2; i++) { Vz(i)=Vx(2i)+Vx(2i+1); Vz(number/2+i)=Vy(2i)+Vy(2i+1); }
语法	vpadd.T vrz, vrx, vry
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	VRX 与 VRY 中的相邻元素相加，结果存放于 VRZ 中。 
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

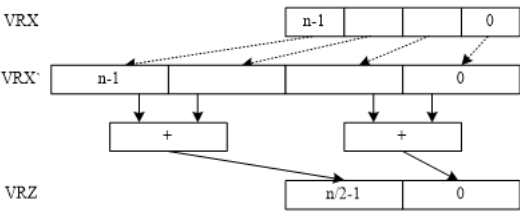
32 位指令格式：

31	op	26	25	24	21	20	19	16	15					5	4	3	0
	111110	w1		VRY	w0		VRX			00000010110				S			VRZ

注：

- 编码表中 w1, w0 用来表征指令位宽类型，其中：
- [w1,w0] = 2’ b00 表示 size 为 8bit 指令；
- [w1,w0] = 2’ b01 表示 size 为 16bit 指令；
- [w1,w0] = 2’ b10 表示 size 为 32bit 指令；
- [w1,w0] = 2’ b11 表示 size 为 64bit 指令；
- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.91 VPADD.T.E——矢量扩展链加法指令

统一化指令	
语法	vpadd.T.e vrz, vrx (T=S8/S16/S32/U8/U16/U32)
操作	size = {8,16,32}; Number=128/size for(i=0; i< Number/2; i++) { Vz(2i+1:2i) =extend(Vx(2i)) +extend(Vx(2i+1)); } extend 根据 U/S 将值零扩展或者符号扩展至元素位宽的 2 倍
编译结果	仅存在 32 位指令。 vpadd.T.e vrz, vrx
数据类型	U8/U16/U32 S8/S16/S32
说明	VRX 的元素扩展，并将相邻元素相加，结果存放于 VRZ 中。 
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32}; Number=128/size for(i=0; i< Number/2; i++) { Vz(2i+1:2i) =extend(Vx(2i)) +extend(Vx(2i+1)); } extend 根据 U/S 将值零扩展或者符号扩展至元素位宽的 2 倍
语法	vpadd.T.e vrz, vrx
数据类型	U8/U16/U32 S8/S16/S32
说明	VRX 的元素扩展，并将相邻元素相加，结果存放于 VRZ 中。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

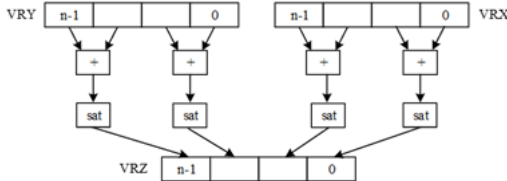
32 位指令格式：

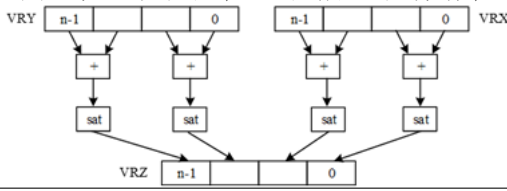
31	op	26	25	24	21	20	19	16	15	5	4	3	0			
111110		w1	0		w0	VRX				00000110100				S	VRZ	

注：

- 编码表中 w1, w0 用来表征指令位宽类型，其中：
- [w1,w0] = 2' b00 表示 size 为 8bit 指令；
- [w1,w0] = 2' b01 表示 size 为 16bit 指令；
- [w1,w0] = 2' b10 表示 size 为 32bit 指令；
- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 VRX,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.92 VPADD.T.S——矢量链加法指令

统一化指令	
语法	vpadd.T.s vrz, vr_x, vr_y (T=S8/S16/S32 /U8/U16/U32)
操作	size = {8, 16, 32, 64}; Number=128/size signed=(T==S); (根据 U/S 类型选择) Max=signed? 2^{^(size-1)}-1: 2^{^(size)}-1; Min=signed? -2^{^(size-1)}:0; for(i=0; i< Number/2; i++) { If V_x(i)+V_x(2i+1)>Max V_z(i)=Max; Else if V_x(i)+V_x(2i+1)<Min V_z(i)=Min; Else V_z(i)=V_x(2i)+V_x(2i+1); If V_y(i)+V_y(2i+1)>Max V_z(number/2+i)=Max; Else if V_y(i)+V_y(2i+1)<Min V_z(number/2+i)=Min; Else V_z(number/2+i)=V_y(2i)+V_y(2i+1); }
编译结果	仅存在 32 位指令。 vpadd.T.s vrz, vr_x, vr_y
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	VR_X 与 VR_Y 中的相邻元素相加，之后做饱和操作，结果存放于 VR_Z 中。 
影响标志位	无影响
限制:	寄存器范围为 vr0-vr15。
异常:	无

32 位指令	
操作	size = {8, 16, 32, 64}; Number=128/size signed=(T==S); (根据元素 U/S 类型选择) Max=signed? 2^{^(size-1)}-1: 2^{^(size)}-1; Min=signed? -2^{^(size-1)}:0; for(i=0; i< Number/2; i++) { If Vx(i)+Vx(2i+1)>Max Vz(i)=Max; Else if Vx(i)+Vx(2i+1)<Min Vz(i)=Min; Else Vz(i)=Vx(2i)+Vx(2i+1); If Vy(i)+Vy(2i+1)>Max Vz(number/2+i)=Max; Else if Vy(i)+Vy(2i+1)<Min Vz(number/2+i)=Min; Else Vz(number/2+i)=Vy(2i)+Vy(2i+1); }
语法	vpadd.T.s vrz, vrx, vry
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	VRX 与 VRY 中的相邻元素相加，之后做饱和操作，结果存放 VRZ 中。 
影响标志位	无影响
限制:	寄存器范围为 vr0-vr15。
异常:	无

32 位指令格式:

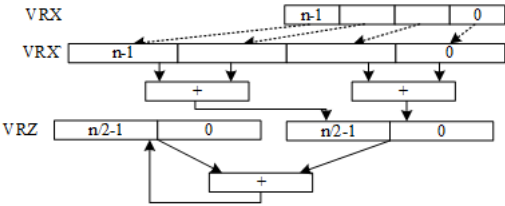
31	op	26	25	24	21	20	19	16	15					5	4	3	0
	111110	w1		VRY	w0		VRX			00000010111				S			VRZ

注:

- 编码表中 w1, w0 用来表征指令位宽类型，其中：
- [w1,w0] = 2’ b00 表示 size 为 8bit 指令；
- [w1,w0] = 2’ b01 表示 size 为 16bit 指令；
- [w1,w0] = 2’ b10 表示 size 为 32bit 指令；
- [w1,w0] = 2’ b11 表示 size 为 64bit 指令；
- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.93 VPADDA.T.E——矢量扩展链累加指令

统一化指令	
语法	vpadda.T.e vrz, vrx (T=S8/S16/S32 /U8/U16/U32)
操作	size = {8,16,32}; Number=128/size for(i=0; i< Number/2; i++) { Vz(2i+1:2i) =Vz(2i+1:2i)+extend(Vx(2i))+extend(Vx(2i+1)); } extend 根据 U/S 将值零扩展或者符号扩展至元素位宽的 2 倍
编译结果	仅存在 32 位指令。 vpadda.T.e vrz, vrx
数据类型	U8/U16/U32 S8/S16/S32
说明	VRX 的元素扩展，并将相邻元素相加，再与 VRZ 的元素相加，结果存放于 VRZ 中。 
影响标志位	不影响
限制:	寄存器范围为 vr0-vr15。
异常:	无

32 位指令	
操作	size = {8,16,32}; Number=128/size for(i=0; i< Number/2; i++) { Vz(2i+1:2i) =Vz(2i+1:2i)+extend(Vx(2i))+extend(Vx(2i+1)); } extend 根据 U/S 将值零扩展或者符号扩展至元素位宽的 2 倍
语法	vpadda.T.e vrz, vrx
数据类型	U8/U16/U32 S8/S16/S32
说明	VRX 的元素扩展，并将相邻元素相加，再与 VRZ 的元素相加，结果存放于 VRZ 中。
影响标志位	不影响
限制:	寄存器范围为 vr0-vr15。
异常:	无

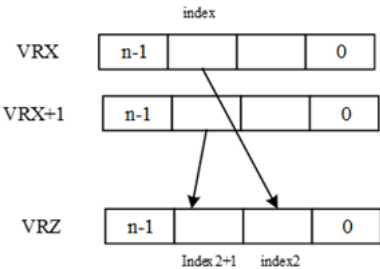
32 位指令格式:

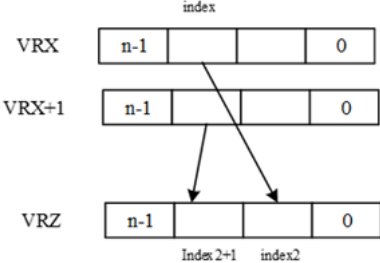
31	op	26	25	24	21	20	19	16	15	5	4	3	0		
111110		w1	0		w0	VRX				00000110101				S	VRZ

注:

- 编码表中 w1, w0 用来表征指令位宽类型，其中:
- [w1,w0] = 2' b00 表示 size 为 8bit 指令;
- [w1,w0] = 2' b01 表示 size 为 16bit 指令;
- [w1,w0] = 2' b10 表示 size 为 32bit 指令;
- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 VRX,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.94 VPKG.T.2——二元封装指令

统一化指令	
语法	vpkg.T.2 vrz, vrx index, index2 (T=8/16/32)
操作	size = {8,16,32}; number = 128 / size; bound=number; (bound 用来判断是否跨寄存器，number 即元素个数) Vz(index2)=Vx(index); if (index2+1<bound){ Vz(index2+1)=Vx+1(index); } else { Vz+1(index2+1-bound)=Vx+1(index); } 此外，Vz,Vz+1 中其余元素不变。 且 index=0 ~ (128/element_size-1); index2=0 ~ (128/element_size -1)
编译结果	仅存在 32 位指令。 vpkg.T.2 vrz, vrx index, index2
数据类型	8/16/32
说明:	将源矢量寄存器 VRX 和 VRX+1 中的第 index 个元素拿出来并依次放入 VRZ 和 VRZ+1 合并后的第 index2 和 index2+1 位。VRZ 和 VRZ+1 的其余元素保持不变。 
影响标志位:	无影响
限制:	寄存器的范围为 vr0-vr15。
异常:	无

32 位指令	
操作:	size = {8,16,32}; number = 128 / size; bound=number; (bound 用来判断是否跨寄存器, number 即元素个数) Vz(index2)=Vx(index); if (index2+1<bound){ Vz(index2+1)=Vx+1(index); } else { Vz+1(index2+1-bound)=Vx+1(index); } 此外, Vz,Vz+1 中其余元素不变。 Index =0 ~ (128/element_size-1); index2=0 ~ (128/element_size -1)
语法:	vpkg.T.2 vrz, vrx index, index2
数据类型	8/16/32
说明:	将源矢量寄存器 VRX 和 VRX+1 中的第 index 个元素拿出来并依次放入 VRZ 和 VRZ+1 合并后的第 index2 和 index2+1 位。VRZ 和 VRZ+1 的其余元素保持不变。 
影响标志位:	无影响
限制:	寄存器的范围为 vr0-vr15。
异常:	无

32 位指令格式:

31	op	26	25	24	21	20	19	16	15	7	6	4	3	0
111110				l1	IMM4		l0	VRX		000010101			IMM3	VRZ

注:

编码表中 w1, w0 用来表征指令位宽类型, 其中:

[w1,w0] = 2' b00 表示 size 为 8bit 指令;

[w1,w0] = 2' b01 表示 size 为 16bit 指令;

[w1,w0] = 2' b10 表示 size 为 32bit 指令;

编码表中 VRX,VRZ 分别表征这条指令各个矢量寄存器的寄存器号;

编码表中 I1, I0 与 Imm3 组合在一起用来表征指令位宽类型和 Index, 其中:

$IMM[4:0] = \{I1, I0, IMM3\}$;

if($IMM[4:3]=2'b01$) size=16, IMM[2:0] 表示 index0 到 7;

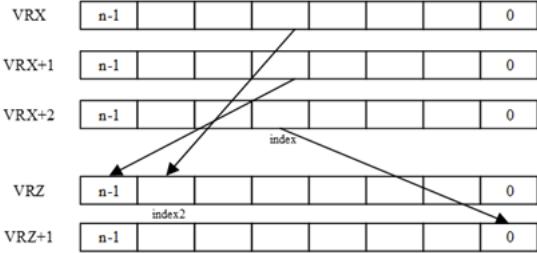
if($IMM[4:3]=2'b10$) size=32, IMM[2:0] 表示 index0 到 3 (高位为 0)

if($IMM[4:3]=2'b00$) size=8, IMM[2:0] 表示 index0 到 7;

if($IMM[4:3]=2'b11$) size=8, IMM[2:0] 表示 index8 到 15;

16.95 VPKG.T.3——三元封装指令

统一化指令	
语法	vpkg.T.3 vrz, vrx index, index2 (T=8/16/32)
操作	size = {8,16,32}; number = 128 / size; bound=number; (bound 用来判断是否跨寄存器，number 即元素个数) Vz(index2)=Vx(index); if (index2+1<bound){ Vz(index2+1)=Vx+1(index); } else { Vz+1(index2+1-bound)=Vx+1(index); } if (index2+2<bound){ Vz(index2+2)=Vx+2(index); } else { Vz+1(index2+2-bound)=Vx+2(index); } 此外，Vz,Vz+1 中其余元素不变。 且 index=0 ~ (128/element_size-1); index2=0 ~ (128/element_size -1)
编译结果	仅存在 32 位指令。 vpkg.T.3 vrz, vrx index, index2
数据类型	8/16/32
说明	将源矢量寄存器 VRX 和 VRX+1, VRX+2 中的第 index 个元素拿出来并依次放入 VRZ 和 VRZ+1 合并后的第 index2, index2+1 和 index2+2 位。VRZ 和 VRZ+1 的其余元素保持不变。 VRX VRX+1 VRX+2 VRZ VRZ+1 n-1 0 n-1 0 n-1 0 n-1 0 n-1 0 index2
影响标志位	无影响
限制	寄存器的范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32}; number = 128 / size; bound=number; (bound 用来判断是否跨寄存器，number 即元素个数) Vz(index2)=Vx(index); if (index2+1<bound){ Vz(index2+1)=Vx+1(index); } else { Vz+1(index2+1-bound)=Vx+1(index); } if (index2+2<bound){ Vz(index2+2)=Vx+2(index); } else { Vz+1(index2+2-bound)=Vx+2(index); } 此外，Vz,Vz+1 中其余元素不变。 且 index=0 ~ (128/element_size-1); index2=0 ~ (128/element_size -1)
语法	vpkg.T.3 vrz, vrx index, index2
数据类型	8/16/32
说明	将源矢量寄存器 VRX 和 VRX+1, VRX+2 中的第 index 个元素拿出来并依次放入 VRZ 和 VRZ+1 合并后的第 index2, index2+1 和 index2+2 位。VRZ 和 VRZ+1 的其余元素保持不变。 
影响标志位	无影响
限制	寄存器的范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	7	6	4	3	0
111110			I1		IMM4		I0	VRX		000010110			IMM3	VRZ

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

$[w1, w0] = 2' b00$ 表示 size 为 8bit 指令；

$[w1, w0] = 2' b01$ 表示 size 为 16bit 指令；

$[w1, w0] = 2' b10$ 表示 size 为 32bit 指令；

编码表中 VRX, VRY, VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

编码表中 I1, I0 与 Imm3 组合在一起用来表征指令位宽类型和 Index，其中：

$IMM[4:0] = \{I1, I0, IMM3\}$ ；

if($IMM[4:3] = 2' b01$) size=16, $IMM[2:0]$ 表示 index0 到 7；

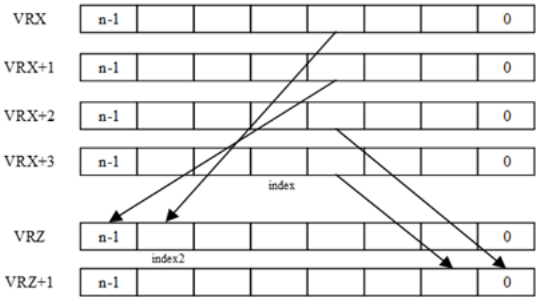
if($IMM[4:3] = 2' b10$) size=32, $IMM[2:0]$ 表示 index0 到 3（高位为 0）

if($IMM[4:3] = 2' b00$) size=8, $IMM[2:0]$ 表示 index0 到 7；

if($IMM[4:3] = 2' b11$) size=8, $IMM[2:0]$ 表示 index8 到 15。

16.96 VPKG.T.4——四元封装指令

统一化指令	
语法	vpkg.T.4 vrz, vrx index, index2 (T=8/16/32)
操作	size = {8,16,32}; number = 128 / size; bound=number; (bound 用来判断是否跨寄存器，number 即元素个数) Vz(index2)=Vx(index); if (index2+1<bound){ Vz(index2+1)=Vx+1(index); } else { Vz+1(index2+1-bound)=Vx+1(index); } if (index2+2<bound){ Vz(index2+2)=Vx+2(index); } else { Vz+1(index2+2-bound)=Vx+2(index); } if (index2+3<bound){ Vz(index2+3)=Vx+3(index); } else { Vz+1(index2+3-bound)=Vx+3(index); } 此外，Vz,Vz+1 中其余元素不变。 且 index=0 ~ (128/element_size-1); index2=0 ~ (128/element_size -1)
编译结果	仅存在 32 位指令。 vpkg.T.4 vrz, vrx index, index2
数据类型	8/16/32
说明	将源矢量寄存器 VRX, VRX+1, VRX+2 和 VRX+3 中的第 index 个元素拿出来并依次放入 VRZ 和 VRZ+1 合并后的第 index2, index2+1, index2+2 和 index2+3 位。VRZ 和 VRZ+1 的其余元素保持不变。 VRX n-1 0 VRX+1 n-1 0 VRX+2 n-1 0 VRX+3 n-1 0 index VRZ n-1 734 VRZ+1 n-1 0 index2 ©【杭州中天微系统有限公司】版权所有
影响标志位	无影响
寄存器	寄存器范围从 0 ~ 15

32 位指令	
操作	size = {8,16,32}; number = 128 / size; bound=number; (bound 用来判断是否跨寄存器, number 即元素个数) Vz(index2)=Vx(index); if (index2+1<bound){ Vz(index2+1)=Vx+1(index); } else { Vz+1(index2+1-bound)=Vx+1(index); } if (index2+2<bound){ Vz(index2+2)=Vx+2(index); } else { Vz+1(index2+2-bound)=Vx+2(index); } if (index2+3<bound){ Vz(index2+3)=Vx+3(index); } else { Vz+1(index2+3-bound)=Vx+3(index); } 此外, Vz,Vz+1 中其余元素不变。 Index =0 ~ (128/element_size-1); index2=0 ~ (128/element_size -1)
语法	vpkg.T.4 vrz, vrx index, index2
数据类型	8/16/32
说明	将源矢量寄存器 VRX, VRX+1, VRX+2 和 VRX+3 中的第 index 个元素拿出来并依次放入 VRZ 和 VRZ+1 合并后的第 index2, index2+1, index2+2 和 index2+3 位。VRZ 和 VRZ+1 的其余元素保持不变。 
影响标志位	无影响
限制	寄存器的范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	7	6	4	3	0
111110			I1	IMM4		I0	VRX		000010111			IMM3		VRZ

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

编码表中 VRX,VRZ,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

编码表中 I1, I0 与 Imm3 组合在一起用来表征指令位宽类型和 Index，其中：

IMM[4:0] = {I1,I0,IMM3}；

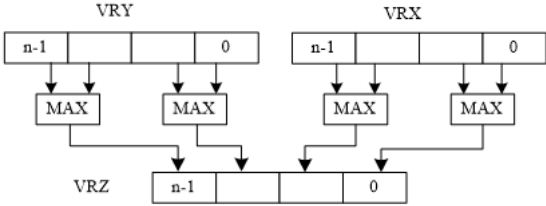
if(IMM[4:3]=2’ b01) size=16, IMM[2:0] 表示 index0 到 7；

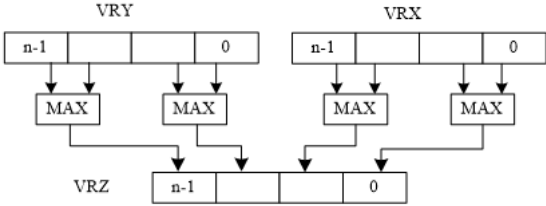
if(IMM[4:3]=2’ b10) size=32, IMM[2:0] 表示 index0 到 3（高位为 0）

if(IMM[4:3]=2’ b00) size=8, IMM[2:0] 表示 index0 到 7；

if(IMM[4:3]=2’ b11) size=8, IMM[2:0] 表示 index8 到 15；

16.97 VPMAX.T——矢量相邻最大值选择指令

统一化指令	
语法	vpmax.T vrz, vrx, vry (T=S8/S16/S32 /U8/U16/U32)
操作	size = {8, 16, 32}; Number = 128 / size; for(i=0; i< number; i++) { Vz(i) = max(Vx(i*2), Vx(i*2+1)); Vz(i+number/2)=max(Vy(i*2),Vy(i*2+1)); }
编译结果	仅存在 32 位指令。 vpmax.T vrz, vrx, vry
数据类型	U8/U16/U32 S8/S16/S32
说明	选择 VRX 和 VRY 相邻元素的较大值存入 VRZ 的对应元素中。 
影响标志位	不影响
限制	寄存器的范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8, 16, 32}; Number = 128 / size; for(i=0; i< number; i++) { Vz(i) = max(Vx(i*2), Vx(i*2+1)); Vz(i+number/2)=max(Vy(i*2),Vy(i*2+1)); }
语法	vpmax.T vrz, vrx, vry
数据类型	U8/U16/U32 S8/S16/S32
说明	选择 VRX 和 VRY 相邻元素的较大值存入 VRZ 的对应元素中。 
影响标志位	不影响
限制	寄存器的范围为 vr0-vr15。
异常	无

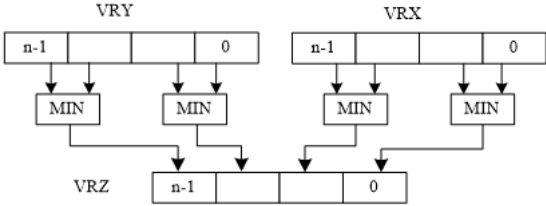
32 位指令格式：

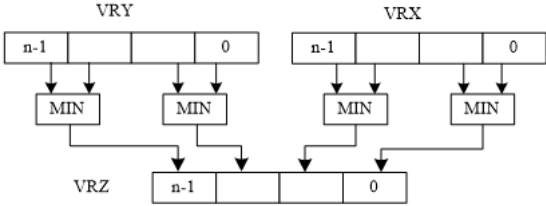
31	op	26	25	24	21	20	19	16	15	5	4	3	0	
111110		w1	VRY			w0	VRX			00000101000			S	VRZ

注：

- 编码表中 w1, w0 用来表征指令位宽类型，其中：
- [w1,w0] = 2’ b00 表示 size 为 8bit 指令；
- [w1,w0] = 2’ b01 表示 size 为 16bit 指令；
- [w1,w0] = 2’ b10 表示 size 为 32bit 指令；
- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.98 VPMIN.T——矢量相邻最小值选择指令

统一化指令	
语法	<code>vpmmin.T vrz, vr_x, vr_y</code> (T=S8/S16/S32 /U8/U16/U32)
操作	<code>size = {8, 16, 32};</code> <code>Number = 128 / size;</code> <code>for(i=0; i< number; i++)</code> <code>{</code> <code> Vz(i) = min(Vx(i*2), Vx(i*2+1));</code> <code> Vz(i+number/2)=min(Vy(i*2),Vy(i*2+1));</code> <code>}</code>
编译结果	仅存在 32 位指令。 <code>vpmmin.T vrz, vr_x, vr_y</code>
数据类型	U8/U16/U32 S8/S16/S32
说明	选择 VRX 和 VRY 相邻元素的较小值存入 VRZ 的对应元素中。 
影响标志位	不影响
限制	寄存器的范围为 vr0-vr15。
异常	无

32 位指令	
操作	<pre>size = {8, 16, 32}; Number = 128 / size; for(i=0; i< number; i++) { Vz(i) = min(Vx(i*2), Vx(i*2+1)); Vz(i+number/2)=min(Vy(i*2),Vy(i*2+1)); }</pre>
语法	vpmin.T vrz, vrx, vry
数据类型	U8/U16/U32 S8/S16/S32
说明	选择 VRX 和 VRY 相邻元素的较小值存入 VRZ 的对应元素中。 
影响标志位	不影响
限制	寄存器的范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1	VRY		w0	VRX		00000101001			S	VRZ	

注：

- 编码表中 w1, w0 用来表征指令位宽类型，其中：
- [w1,w0] = 2’ b00 表示 size 为 8bit 指令；
 - [w1,w0] = 2’ b01 表示 size 为 16bit 指令；
 - [w1,w0] = 2’ b10 表示 size 为 32bit 指令；
- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.99 VRCMUL.T.RS——矢量带饱和复数乘法取高半舍入指令

统一化指令	
语法	vrcmul.T.rs vz, vx, vy (T=S8/S16/S32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Round= 1<<(size-1) for(i=0; i< Number/2; i++) { Tmp(2i+1)=Vx(2i)*Vy(2i+1)*2 +Vx(2i+1)*Vy(2i)*2+round; Tmp(2i)= Vx(2i)*Vy(2i)*2-Vx(2i+1) *Vy(2i+1)*2+round; } 注： Tmp(i) 保留全部精度 for(i=0; i< Number; i++) If(Tmp(i)>2^(2*size-1)-1) Vz(i)= 2^(size-1)-1; Else if(Tmp(i)<-2^(2*size-1)) Vz(i)= -2^(size-1); Else Vz(i)=High[Tmp(i)]; } 注：饱和在累加减后进行
编译结果	仅存在 32 位指令。 vrcmul.T.rs vz, vx, vy
数据类型	S8/S16/S32
说明	实现复数乘法计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；VRX 和 V RY 中元素所对应的实部虚部交织相乘保留全部精度相加，对结果进行舍入饱和判断，将高半部分结果存入 VRZ。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Round= 1<<(size-1) for(i=0; i< Number/2; i++) { Tmp(2i+1)= Vx(2i)*Vy(2i+1)*2+Vx(2i+1)*Vy(2i)*2+round; Tmp(2i)=Vx(2i)*Vy(2i)*2-Vx(2i+1)*Vy(2i+1)*2+round; } 注： Tmp(i) 保留全部精度 for(i=0; i< Number; i++) If(Tmp(i)>2^(2*size-1)-1) Vz(i)= 2^(size-1)-1; Else if(Tmp(i)<-2^(2*size-1)) Vz(i)= -2^(size-1); Else Vz(i)=High[Tmp(i)]; } 注：饱和在累加减后进行
语法	vrcmul.T.rs vz, vx, vy
数据类型	S8/S16/S32
说明	实现复数乘法计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；VRX 和 V RY 中元素所对应的实部虚部交织相乘保留全部精度相加，对结果进行舍入饱和判断，将高半部分结果存入 VRZ。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1	VRY			w0	VRX			01111001000			
										S		VRZ	

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.100

VRCMULA.T.E——矢量复数乘法右移扩位累加指令

统一化指令	
语法	vrcmula.T.e vz, vx, vy (T=S8/S16/S32)
操作	Number=128/size i 表征第几个元素， 范围为 [0,Number-1] Imm4 表示立即数， 范围为 [0,15] for(i=0; i< Number/4; i++) { $Vz(4i+3:4i+2)=Vz(4i+3:4i+2)+((Vx(2i)*Vy(2i+1))>>imm4)+((Vx(2i+1)*Vy(2i))>>imm4);$ (虚部) $Vz(4i+1:4i)=Vz(4i+1:4i)+((Vx(2i)*Vy(2i))>>imm4)+((-Vx(2i+1)*Vy(2i+1))>>imm4);$ (实部) } for(i=Number/4; i<Number/2; i++) { k=i-Number/4; $Vz+1(4k+3:4k+2)=Vz+1(4k+3:4k+2)+((Vx(2i)*Vy(2i+1))>>imm4)+((Vx(2i+1)*Vy(2i))>>imm4);$ (虚部) $Vz+1(4k+1:4k)=Vz+1(4k+1:4k)+((Vx(2i)*Vy(2i))>>imm4)+((-Vx(2i+1)*Vy(2i+1))>>imm4);$ (实部) } 注：复数乘法结果保留全部精度后进行算术右移，再累加
编译结果	仅存在 32 位指令。 vrcmula.T.e vz, vx, vy
数据类型	S8/S16/S32
说明	实现复数乘法右移累加计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；VRX 和 VRY 中元素所对应的实部虚部相乘后得到的积保留全精度进行右移，计算出来的虚部和实部的结果分别与 VRZ 和 VRZ+1 对应的元素累加，最终的结果存入 VRZ 和 VRZ+1。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Imm4 表示立即数，范围为 [0,15] for(i=0; i< Number/4; i++) { $Vz(4i+3:4i+2)=Vz(4i+3:4i+2)+((Vx(2i)*Vy(2i+1))>>imm4)+((Vx(2i+1)*Vy(2i))>>imm4);$ (虚部) $Vz(4i+1:4i)=Vz(4i+1:4i)+((Vx(2i)*Vy(2i))>>imm4)+((-Vx(2i+1)*Vy(2i+1))>>imm4);$ (实部) } for(i=Number/4; i<Number/2; i++) { k=i-Number/4; $Vz+1(4k+3:4k+2)=Vz+1(4k+3:4k+2)+((Vx(2i)*Vy(2i+1))>>imm4)+((Vx(2i+1)*Vy(2i))>>imm4);$ (虚部) $Vz+1(4k+1:4k)=Vz+1(4k+1:4k)+((Vx(2i)*Vy(2i))>>imm4)+((-Vx(2i+1)*Vy(2i+1))>>imm4);$ (实部) } 注：复数乘法结果保留全部精度后进行算术右移，再累加
语法	vrcmula.T.e vz, vx, vy
数据类型	S8/S16/S32
说明	实现复数乘法右移累加计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；VRX 和 VRY 中元素所对应的实部虚部相乘后得到的积保留全精度进行右移，计算出来的虚部和实部的结果分别与 VRZ 和 VRZ+1 对应的元素累加，最终的结果存入 VRZ 和 VRZ+1。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	10	9	5	4	3	0
111110		w1	VRY		w0	VRX		0110100		IMM4		S	VRZ		

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

$[w1, w0] = 2'$ b00 表示 size 为 8bit 指令；

$[w1, w0] = 2'$ b01 表示 size 为 16bit 指令；

$[w1, w0] = 2'$ b10 表示 size 为 32bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 Imm4 表征立即数；

编码表中 VRX, VRY, VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.101 VRCMULC.T.RS——矢量带饱和复数共轭乘法取高半舍入指令

统一化指令	
语法	vrcmulc.T.rs vz, vx, vy (T=S8/S16/S32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Round= 1<<(size-1) for(i=0; i< Number/2; i++) { Tmp(2i+1)=Vx(2i)*Vy(2i+1)*2-Vx(2i+1)*Vy(2i)*2+round; Tmp(2i)=Vx(2i)*Vy(2i)*2+Vx(2i+1)*Vy(2i+1)*2+round; } 注： Tmp(i) 保留全部精度 for(i=0; i< Number; i++) If(Tmp(i)>2^(2*size-1)-1) Vz(i)= 2^(size-1)-1; Else if(Tmp(i)<-2^(2*size-1)) Vz(i)= -2^(size-1); Else Vz(i)=High[Tmp(i)]; } 注： 饱和在累加减后进行
编译结果	仅存在 32 位指令。 vrcmulc.T.rs vz, vx, vy
数据类型	S8/S16/S32
说明	实现复数共轭乘法，一个矢量操作数包含两个复数，实部在低位，虚部在高位；VRX 和 VRY 中元素所对应的实部虚部交织相乘保留全部精度相加，对结果进行舍入饱和判断，将高半部分结果存入 VRZ。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Round= 1<<(size-1) for(i=0; i< Number/2; i++) { Tmp(2i+1)=Vx(2i)*Vy(2i+1)*2-Vx(2i+1)*Vy(2i)*2+round; Tmp(2i)=Vx(2i)*Vy(2i)*2+Vx(2i+1)*Vy(2i+1)*2+round; } 注： Tmp(i) 保留全部精度 for(i=0; i< Number; i++) If(Tmp(i)>2^(2*size-1)-1) Vz(i)= 2^(size-1)-1; Else if(Tmp(i)<-2^(2*size-1)) Vz(i)= -2^(size-1); Else Vz(i)=High[Tmp(i)]; } 注： 饱和在累加减后进行
语法	vrcmulc.T.rs vz, vx, vy
数据类型	S8/S16/S32
说明	实现复数共轭乘法计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；VRX和 VRY 中元素所对应的实部虚部交织相乘保留全部精度累加，对结果进行舍入饱和判断，将高半部分结果存入 VRZ。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1	VRY		w0	VRX		01111001001			S	VRZ	

注：

- 编码表中 w1, w0 用来表征指令位宽类型，其中：
- [w1,w0] = 2’ b00 表示 size 为 8bit 指令；
- [w1,w0] = 2’ b01 表示 size 为 16bit 指令；
- [w1,w0] = 2’ b10 表示 size 为 32bit 指令；
- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.102 VRCMULCA.T.E——矢量共轭复数乘法右移扩位累加指令

统一化指令	
语法	vrcmulca.T.e vz, vx, vy (T=S8/S16/S32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Imm4 表示立即数，范围为 [0,15] for(i=0; i< Number/4; i++) { Vz(4i+3:4i+2)=Vz(4i+3:4i+2) + ((Vx(2i)*Vy(2i+1))>>imm4) + ((- Vx(2i+1)*Vy(2i))>>imm4); (虚部) Vz(4i+1:4i)=Vz(4i+1:4i) + ((Vx(2i)*Vy(2i))>>imm4) + ((Vx(2i+1)*Vy(2i+1))>>imm4); (实部) } for(i=Number/4; i<Number/2; i++) { k=i-Number/4; Vz+1(4k+3:4k+2)=Vz+1(4k+3:4k+2) + ((Vx(2i)*Vy(2i+1))>>imm4) + ((- Vx(2i+1)*Vy(2i))>>imm4); (虚部) Vz+1(4k+1:4k)=Vz+1(4k+1:4k) + ((Vx(2i)*Vy(2i))>>imm4) + ((Vx(2i+1)*Vy(2i+1))>>imm4); (实部) } 注：复数乘法结果保留全部精度后进行算术右移，再累加
编译结果	仅存在 32 位指令。 vrcmulca.T.e vz, vx, vy
数据类型	S8/S16/S32
说明	实现复数乘法右移累加计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；VRX 和 VRY 中元素所对应的实部虚部相乘后得到的积保留全精度进行右移，计算出来的虚部和实部的结果分别与 VRZ 和 VRZ+1 对应的元素累加，最终的结果存入 VRZ 和 VRZ+1。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Imm4 表示立即数，范围为 [0,15] for(i=0; i< Number/4; i++) { $Vz(4i+3:4i+2)=Vz(4i+3:4i+2) + ((Vx(2i)*Vy(2i+1))>>imm4) + ((-Vx(2i+1)*Vy(2i))>>imm4);$ (虚部) $Vz(4i+1:4i)=Vz(4i+1:4i) + ((Vx(2i)*Vy(2i))>>imm4) + ((Vx(2i+1)*Vy(2i+1))>>imm4);$ (实部) } for(i=Number/4; i<Number/2; i++) { k=i-Number/4; $Vz+1(4k+3:4k+2)=Vz+1(4k+3:4k+2) + ((Vx(2i)*Vy(2i+1))>>imm4) + ((-Vx(2i+1)*Vy(2i))>>imm4);$ (虚部) $Vz+1(4k+1:4k)=Vz+1(4k+1:4k) + ((Vx(2i)*Vy(2i))>>imm4) + ((Vx(2i+1)*Vy(2i+1))>>imm4);$ (实部) } 注：复数乘法结果保留全部精度后进行算术右移，再累加
语法	vrcmulca.T.e vz, vx, vy
数据类型	S8/S16/S32
说明	实现复数乘法右移累加计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；VRX 和 VRY 中元素所对应的实部虚部相乘后得到的积保留全精度进行右移，计算出来的虚部和实部的结果分别与 VRZ 和 VRZ+1 对应的元素累加，最终的结果存入 VRZ 和 VRZ+1。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	10	9	5	4	3	0
111110			w1	VRY		w0	VRX		0110101			IMM4		S	VRZ

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

$[w1, w0] = 2' \text{ b}00$ 表示 size 为 8bit 指令；

$[w1, w0] = 2' \text{ b}01$ 表示 size 为 16bit 指令；

$[w1, w0] = 2' \text{ b}10$ 表示 size 为 32bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 Imm4 表征立即数；

编码表中 VRX, VRY, VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.103 VRCMULN.T.RS——矢量带饱和复数取负乘法取高半舍入指令

统一化指令	
语法	vrcmuln.T.rs vz, vx, vy (T=S8/S16/S32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Round= 1<<(size-1) for(i=0; i< Number/2; i++) { Tmp(2i+1)= -Vx(2i)*Vy(2i+1)*2-Vx(2i+1)*Vy(2i)*2+round; Tmp(2i)= -Vx(2i)*Vy(2i)*2+Vx(2i+1)*Vy(2i+1)*2+round; } 注： Tmp(i) 保留全部精度 for(i=0; i< Number; i++) If(Tmp(i)>2^(2*size-1)-1) Vz(i)= 2^(size-1)-1; Else if(Tmp(i)<-2^(2*size-1)) Vz(i)= -2^(size-1); Else Vz(i)=High[Tmp(i)]; } 注：饱和在累加减后进行
编译结果	仅存在 32 位指令。 vrcmuln.T.rs vz, vx, vy
数据类型	S8/S16/S32
说明	实现复数取反乘法计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；VRX 和 VRX 中元素所对应的实部虚部交织相乘保留全部精度相加，对结果进行舍入饱和判断，将高半部分结果存入 VRZ。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Round= 1<<(size-1) for(i=0; i< Number/2; i++) { Tmp(2i+1)=-Vx(2i)*Vy(2i+1)*2-Vx(2i+1)*Vy(2i)*2+round; Tmp(2i)=-Vx(2i)*Vy(2i)*2+Vx(2i+1)*Vy(2i+1)*2+round; } 注：Tmp(i) 保留全部精度 for(i=0; i< Number; i++) If(Tmp(i)>2^(2*size-1)-1) Vz(i)= 2^(size-1)-1; Else if(Tmp(i)<-2^(2*size-1)) Vz(i)= -2^(size-1); Else Vz(i)=High[Tmp(i)]; } 注：饱和在累加减后进行
语法	vrcmuln.T.rs vz, vx, vy
数据类型	S8/S16/S32
说明	实现复数取反乘法，一个矢量操作数包含两个复数，实部在低位，虚部在高位；VRX 和 VRY 中元素所对应的实部虚部交织相乘并进行取负操作再进行舍入后，分别进行乘累加得到两个复数相乘后的结果，最后进行饱和判断，最终的结果存入 VRZ。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1	VRY		w0	VRX		01111001010			S	VRZ	

注：

- 编码表中 w1, w0 用来表征指令位宽类型，其中：
- [w1,w0] = 2’ b00 表示 size 为 8bit 指令；
- [w1,w0] = 2’ b01 表示 size 为 16bit 指令；
- [w1,w0] = 2’ b10 表示 size 为 32bit 指令；
- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.104 VRCMULNA.T.E——矢量复数取负乘法右移扩位累加指令

统一化指令	
语法	vrcmula.T.e vz, vx, vy (T=S8/S16/S32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Imm4 表示立即数，范围为 [0,15] for(i=0; i< Number/4; i++) { Vz(4i+3:4i+2)=Vz(4i+3:4i+2)+((-Vx(2i)*Vy(2i+1))>>imm4)+((-Vx(2i+1)*Vy(2i))>>imm4); (虚部) Vz(4i+1:4i)=Vz(4i+1:4i)+((-Vx(2i)*Vy(2i))>>imm4)+((Vx(2i+1)*Vy(2i+1))>>imm4); (实部) } for(i=Number/4; i<Number/2; i++) { k=i-Number/4; Vz+1(4k+3:4k+2)=Vz+1(4k+3:4k+2) + (-Vx(2i)*Vy(2i+1))>>imm4 + (-Vx(2i+1)*Vy(2i))>>imm4); (虚部) Vz+1(4k+1:4i)=Vz+1(4k+1:4i) + ((-Vx(2i)*Vy(2i))>>imm4) + ((Vx(2i+1)*Vy(2i+1))>>imm4); (实部) } 注：复数乘法结果保留全部精度后进行算术右移，再累加
编译结果	仅存在 32 位指令。 vrcmula.T.e vz, vx, vy
数据类型	S8/S16/S32
说明	实现复数取反乘法右移累加计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；VRX 和 VRY 中元素所对应的实部虚部相乘后得到的积取负后保留全精度进行右移，计算出来的虚部和实部的结果分别与 VRZ 和 VRZ+1 对应的元素累加，最终的结果存入 VRZ 和 VRZ+1。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Imm4 表示立即数，范围为 [0,15] for(i=0; i< Number/4; i++) { $Vz(4i+3:4i+2)=Vz(4i+3:4i+2)+((-Vx(2i)*Vy(2i+1))>>imm4)+((-Vx(2i+1)*Vy(2i))>>imm4);$ (虚部) $Vz(4i+1:4i)=Vz(4i+1:4i)+((-Vx(2i)*Vy(2i)>>imm4)+((Vx(2i+1)*Vy(2i+1))>>imm4);$ (实部) } for(i=Number/4; i<Number/2; i++) { k=i-Number/4; $Vz+1(4k+3:4k+2)=Vz+1(4k+3:4k+2)+(- (Vx(2i)*Vy(2i+1))>>imm4)+(- (Vx(2i+1)*Vy(2i))>>imm4);$ (虚部) $Vz+1(4k+1:4i)=Vz+1(4k+1:4i)+((-Vx(2i)*Vy(2i)>>imm4)+((Vx(2i+1)*Vy(2i+1))>>imm4);$ (实部) } 注：复数乘法结果保留全部精度后进行算术右移，再累加
语法	vrcmula.T.e vz, vx, vy
数据类型	S8/S16/S32
说明	实现复数取反乘法右移累加计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；VRX 和 VRY 中元素所对应的实部虚部相乘后得到的积取负后保留全精度进行右移，计算出来的虚部和实部的结果分别与 VRZ 和 VRZ+1 对应的元素累加，最终的结果存入 VRZ 和 VRZ+1。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	10	9	5	4	3	0
111110		w1	VRY		w0	VRX		0110101		IMM4		S	VRZ		

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

$[w1, w0] = 2'$ b00 表示 size 为 8bit 指令；

$[w1, w0] = 2'$ b01 表示 size 为 16bit 指令；

$[w1, w0] = 2'$ b10 表示 size 为 32bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 Imm4 表征立即数；

编码表中 VRX, VRY, VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.105

VRCMULCN.T.RS——

矢量带饱和复数共轭取负乘法取高半舍入指令

统一化指令	
语法	vrcmulcn.T.rs vz, vx, vy (T=S8/S16/S32)
操作	Number=128/size i 表征第几个元素, [0,Number-1] Round= 1<<(size-1) for(i=0; i< Number/2; i++) { Tmp(2i+1)= -Vx(2i)*Vy(2i+1)*2+ Vx(2i+1)*Vy(2i)*2+round; Tmp(2i)= -Vx(2i)*Vy(2i)*2- Vx(2i+1)*Vy(2i+1)*2+round; } 注： Tmp(i) 保留全部精度 for(i=0; i< Number; i++) If(Tmp(i)>2^(2*size-1)-1) Vz(i)= 2^(size-1)-1; Else if(Tmp(i)<-2^(2*size-1)) Vz(i)= -2^(size-1); Else Vz(i)=High[Tmp(i)]; } 注： 饱和在累加减后进行
编译结果	仅存在 32 位指令。 vrcmulcn.T.rs vz, vx, vy
数据类型	S8/S16/S32
说明	实现复数共轭取反乘法，一个矢量操作数包含两个复数，实部在低位，虚部在高位；VRX和 VRY 中元素所对应的实部虚部交织相乘保留全部精度累加，对结果进行舍入饱和判断，将高半部分结果存入 VRZ。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Round= 1<<(size-1) for(i=0; i< Number/2; i++) { Tmp(2i+1)= -Vx(2i)*Vy(2i+1)*2+Vx(2i+1)*Vy(2i)*2+round; Tmp(2i)= -Vx(2i)*Vy(2i)*2-Vx(2i+1)*Vy(2i+1)*2+round; } 注：Tmp(i) 保留全部精度 for(i=0; i< Number; i++) If(Tmp(i)>2^(2*size-1)-1) Vz(i)= 2^(size-1)-1; Else if(Tmp(i)<-2^(2*size-1)) Vz(i)= -2^(size-1); Else Vz(i)=High[Tmp(i)]; } 注：饱和在累加减后进行
语法	vrcmulcn.T.rs vz, vx, vy
数据类型	S8/S16/S32
说明	实现复数共轭取反乘法，一个矢量操作数包含两个复数，实部在低位，虚部在高位；VRX和 VRY 中元素所对应的实部虚部交织相乘保留全部精度累加，对结果进行舍入饱和判断，将高半部分结果存入 VRZ。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1	VRY		w0	VRX		01111001011			S	VRZ	

注：

- 编码表中 w1, w0 用来表征指令位宽类型，其中：
- [w1,w0] = 2’ b00 表示 size 为 8bit 指令；
- [w1,w0] = 2’ b01 表示 size 为 16bit 指令；
- [w1,w0] = 2’ b10 表示 size 为 32bit 指令；
- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.106 VRCMULCNA.T.E——矢量共轭复数取负乘法右移扩位累加指令

统一化指令	
语法	vrcmulcna.T.e vz, vx, vy (T=S8/S16/S32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Imm4 表示立即数，范围为 [0,15] for(i=0; i< Number/4; i++) { $Vz(4i+3:4i+2)=Vz(4i+3:4i+2) \quad + \quad ((-Vx(2i)*Vy(2i+1))>>imm4) \quad +$$((Vx(2i+1)*Vy(2i))>>imm4);$ (虚部) $Vz(4i+1:4i)=Vz(4i+1:4i) \quad + \quad ((-Vx(2i)*Vy(2i))>>imm4)+ \quad ((-$$Vx(2i+1)*Vy(2i+1))>>imm4);$ (实部) } for(i=Number/4; i<Number/2; i++) { k=i-Number/4; $Vz+1(4k+3:4k+2)=Vz+1(4k+3:4k+2) \quad + \quad ((-Vx(2i)*Vy(2i+1))>>imm4) \quad +$$((Vx(2i+1)*Vy(2i))>>imm4);$ (虚部) $Vz+1(4k+1:4k)=Vz+1(4k+1:4k) \quad + \quad ((-Vx(2i)*Vy(2i))>>imm4) \quad + \quad ((-$$Vx(2i+1)*Vy(2i+1))>>imm4);$ (实部) } 注：复数乘法结果保留全部精度后进行算术右移，再累加
编译结果	仅存在 32 位指令。 vrcmulcna.T.e vz, vx, vy
数据类型	S8/S16/S32
说明	实现复数共轭取反乘法右移累加计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；VRX 和 VRY 中元素所对应的实部虚部相乘后得到的积取负后保留全精度进行右移，计算出来的虚部和实部的结果分别与 VRZ 和 VRZ+1 对应的元素累加，最终的结果存入 VRZ 和 VRZ+1。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Imm4 表示立即数，范围为 [0,15] for(i=0; i< Number/4; i++) { $V_z(4i+3:4i+2)=V_z(4i+3:4i+2) + ((-V_x(2i)*V_y(2i+1))>>imm4)$ $((V_x(2i+1)*V_y(2i))>>imm4);$(虚部) $V_z(4i+1:4i)=V_z(4i+1:4i) + ((-V_x(2i)*V_y(2i))>>imm4)$ $((-V_x(2i+1)*V_y(2i+1))>>imm4);$(实部) } for(i=Number/4; i<Number/2; i++) { k=i-Number/4; $V_{z+1}(4k+3:4k+2)=V_{z+1}(4k+3:4k+2) + ((-V_x(2i)*V_y(2i+1))>>imm4)$ $((V_x(2i+1)*V_y(2i))>>imm4);$(虚部) $V_{z+1}(4k+1:4k)=V_{z+1}(4k+1:4k) + ((-V_x(2i)*V_y(2i))>>imm4)$ $((-V_x(2i+1)*V_y(2i+1))>>imm4);$(实部) } 注：复数乘法结果保留全部精度后进行算术右移，再累加
语法	vrcmulcna.T.e vz, vx, vy
数据类型	S8/S16/S32
说明	实现复数共轭取反乘法右移累加计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；VRX 和 VRY 中元素所对应的实部虚部相乘后得到的积取负后保留全精度进行右移，计算出来的虚部和实部的结果分别与 VRZ 和 VRZ+1 对应的元素累加，最终的结果存入 VRZ 和 VRZ+1
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	10	9	5	4	3	0
111110			w1	VRY			w0	VRX			0110101	IMM4		S	VRZ

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 Imm4 表征立即数；

编码表中 VRX, VRY, VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.107 VRECPE.T——矢量倒数指令

统一化指令		
语法	vrecpe.T vrz, vrx (T=S8/S16/S32 /U8/U16/U32)	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] T 表征指令类型 T={u8,u16,u32,s8,s16,s32} 倒数快速算法公式：$V_z(i) = 2.823 - V_x(i)*1.8823$ for(i=0; i< Number; i++) { If($V_x(i)<0.5$){ Vx(i)=0.5 ; } Elseif($V_x(i)>1$){ Vx(i)=1; } Else{ Vx(i)=Vx(i); } } for(i=0; i< Number; i++) { If(T=s8) Vz(i) = 0x5A + ((Vx(i)* 0xC4)>>5-bit) ; (操作数、结果均为有符号数) If(T=s16) Vz(i) = 0x5A56 • ((Vx(i)* 0xC3C5)>>13-bit); (操作数、结果均为有符号数) If(T=s32) Vz(i) = 0x5A560418 + ((Vx(i)*0xC3C432CB)>>29-bit);(操作数、结果均为有符号数) If(T=u8) Vz(i) = 0x0B4 + ((({0,Vx(i)}*0x188)>>6-bit);(8- bit 无符号数扩展为 9-bit 有符号数运算)，结果取低 8-bit If(T=u16) Vz(i) = 0x0B4AC + ((({0,Vx(i)}* 0x18789)>>14-bit) (16-bit 无符号数扩展为 17-bit 有符号数运算)，结果取低 16-bit If(T=u32) Vz(i) = 0x0B4AC0831 + ((({0,Vx(i)}* 0x187886595)>>30-bit) (32-bit 无符号数扩展为 33-bit 有符号数运算)，结果取低 32-bit }	
编译结果	仅存在 32 位指令。 vrecpe.T vrz, vrx	
数据类型	U8/U16/U32 S8/S16/S32	762 © 【杭州中天微系统有限公司】版权所有

统一化指令	
说明	Vx(i) 值取值范围是 [0.5, 1], 若超过该范围, 则在运算前强制将 Vx(i) 置为边界值, 根据倒数快速算法公式: $V_z(i) = 2.823 - V_x(i) * 1.8823$, magic 值转化为相应有无符号, size 大小的定点数, 在判断 VRX 中元素边界值后, 将 VRX 中的元素代入公式计算出的结果存入 VRZ
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] T 表征指令类型 T={u8,u16,u32,s8,s16,s32} 倒数快速算法公式：$V_z(i) = 2.823 - V_x(i) * 1.8823$ for(i=0; i< Number; i++) { If($V_x(i) < 0.5$){ $V_x(i) = 0.5$; } Elseif($V_x(i) > 1$){ $V_x(i) = 1$; } Else{ $V_x(i) = V_x(i)$; } } for(i=0; i< Number; i++) { If(T=s8) $V_z(i) = 0x5A + ((V_x(i) * 0xC4) >> 5\text{-bit})$;(操作数、结果均为有符号数) If(T=s16) $V_z(i) = 0x5A56 + ((V_x(i) * 0xC3C5) >> 13\text{-bit})$;(操作数、结果均为有符号数) If(T=s32) $V_z(i) = 0x5A560418 + ((V_x(i) * 0xC3C432CB) >> 29\text{-bit})$;(操作数、结果均为有符号数) If(T=u8) $V_z(i) = 0x0B4 + ((\{0, V_x(i)\} * 0x188) >> 6\text{-bit})$;(8-bit 无符号数扩展为 9-bit 有符号数运算)，结果取低 8-bit If(T=u16) $V_z(i) = 0x0B4AC + ((\{0, V_x(i)\} * 0x18789) >> 14\text{-bit})$ (16-bit 无符号数扩展为 17-bit 有符号数运算)，结果取低 16-bit If(T=u32) $V_z(i) = 0x0B4AC0831 + ((\{0, V_x(i)\} * 0x187886595) >> 30\text{-bit})$ (32-bit 无符号数扩展为 33-bit 有符号数运算)，结果取低 32-bit }
语法	vrecpe.T vrz, vrX
数据类型	U8/U16/U32 S8/S16/S32
说明	$V_x(i)$ 值取值范围是 [0.5, 1]，若超过该范围，则在运算前强制将 $V_x(i)$ 置为边界值，根据倒数快速算法公式：$V_z(i) = 2.823 - V_x(i) * 1.8823$，magic 值转化为相应有无符号，size 大小的定点数，在判断 VRX 中元素边界值后，将 VRX 中的元素代入公式计算出的结果存入 VRZ
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110			w1	X	w0	VRX			00011101000			S	VRZ

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.108 VRECPS.T——矢量倒数逼近指令

统一化指令	
语法	vrecps.T vz, vx, vy (T=S8/S16/S32 /U8/U16/U32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] T 表征指令类型 T={u8,u16,u32,s8,s16,s32} 倒数逼近快速算法公式： $V_z(i) = 2 - V_x(i) * V_y(i)$ for(i=0; i< Number; i++) { If(T=s8) $V_z(i) = 0x40 + ((-V_x(i) * V_y(i)) >> 5\text{-bit})$ (操作数、结果均为有符号数) If(T=s16) $V_z(i) = 0x4000 + ((-V_x(i) * V_y(i)) >> 13\text{-bit})$ (操作数、结果均为有符号数) If(T=s32) $V_z(i) = 0x40000000 + ((-V_x(i) * V_y(i)) >> 29\text{-bit})$ (操作数、结果均为有符号数) If(T=u8) $V_z(i) = 0x80 + ((-V_x(i) * V_y(i)) >> 6\text{-bit})$ (8-bit 无符号数扩展为 9-bit 有符号数运算)，结果取低 8-bit If(T=u16) $V_z(i) = 0x8000 + ((-V_x(i) * V_y(i)) >> 14\text{-bit})$ (16-bit 无符号数扩展为 17-bit 有符号数运算)，结果取低 16-bit If(T=u32) $V_z(i) = 0x80000000 + ((-V_x(i) * V_y(i)) >> 30\text{-bit})$ (32-bit 无符号数扩展为 33-bit 有符号数运算)，结果取低 32-bit }
编译结果	仅存在 32 位指令。 vrecps.T vz, vx, vy
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	根据倒数逼近快速算法公式：$V_z(i) = 2 - V_x(i) * V_y(i)$ 将 2 转化为相应有/无符号，size 大小的定点数，将 VRX 和 VRY 中的元素代入公式计算出的结果存入 VRZ。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] T 表征指令类型 T={u8,u16,u32,s8,s16,s32} 倒数逼近快速算法公式： $V_z(i) = 2 - V_x(i) * V_y(i)$ for(i=0; i< Number; i++) { If(T=s8) $V_z(i) = 0x40 + ((-V_x(i) * V_y(i)) >> 5\text{-bit})$ If(T=s16) $V_z(i) = 0x4000 + ((-V_x(i) * V_y(i)) >> 13\text{-bit})$ If(T=s32) $V_z(i) = 0x40000000 + ((-V_x(i) * V_y(i)) >> 29\text{-bit})$ If(T=u8) $V_z(i) = 0x80 + ((-V_x(i) * V_y(i)) >> 6\text{-bit})$ (8-bit 无符号数扩展为 9-bit 有符号数运算)，结果取低 8-bit If(T=u16) $V_z(i) = 0x8000 + ((-V_x(i) * V_y(i)) >> 14\text{-bit})$ (16-bit 无符号数扩展为 17-bit 有符号数运算)，结果取低 16-bit If(T=u32) $V_z(i) = 0x80000000 + ((-V_x(i) * V_y(i)) >> 30\text{-bit})$ (32-bit 无符号数扩展为 33-bit 有符号数运算)，结果取低 32-bit } }
语法:	vrecps.T vz, vx, vy
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	根据倒数逼近快速算法公式： $V_z(i) = 2 - V_x(i) * V_y(i)$ 将 2 转化为相应有无符号，size 大小的定点数，将 VRX 和 VRY 中的元素代入公式计算出的结果存入 VRZ。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

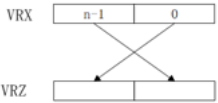
31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1		VRY		w0		VRX		00011101001		S	VRZ

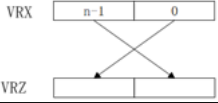
注：

- 编码表中 w1, w0 用来表征指令位宽类型，其中：
- [w1,w0] = 2’ b00 表示 size 为 8bit 指令；
- [w1,w0] = 2’ b01 表示 size 为 16bit 指令；
- [w1,w0] = 2’ b10 表示 size 为 32bit 指令；
- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.109 VREVD——矢量数据双字倒序指令

统一化指令	
语法	vrevd vrz, vrx
操作	size = 64; number = 128/size; Vz(number-1:0)={Vx(0),Vx(1)}; 把 Vrx 的元素倒序放入 Vrz 之中 vrz[i] 指相应 element 的数据;
编译结果	仅存在 32 位指令。 vrevd vrz, vrx
数据类型	64
说明	将源矢量寄存器 VRX 的元素倒序放入 VRZ 之中。 
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = 64; number = 128/size; vrz[i] 指相应位置 element 的数据; Vz(number-1:0)={Vx(0),Vx(1)}; 将 Vrx 中元素数据倒序放入 VRZ 中。
语法	vrevd vrz, vrx
数据类型	64
说明	将源矢量寄存器 VRX 的元素倒序放入 VRZ 中。 
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110	1	X	1	VRX	00001101000	0	VRZ						

注：

编码表中 VRX,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.110 VREVH——矢量数据半字倒序指令

统一化指令	
语法	vrevh vrz, vrx
操作	size = 16; number = 128/size; Vz(number-1:0)={Vx(0),Vx(1),Vx(2),Vx(6),Vx(7)}; 把 Vrx 的元素倒序放入 Vrz 之中 vrx[i] 指相应 element 的数据;
编译结果	仅存在 32 位指令。 vrevh vrz, vrx
数据类型	16
说明	将源矢量寄存器 VRX 的元素倒序放入 VRZ 之中。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = 16; number = 128/size; vr _x [i] 指相应位置 element 的数据; V _z (number-1:0)={V _x (0),V _x (1),V _x (2),V _x (6), V _x (7)}; 将 V _{rx} 中元素数据倒序放入 VRZ 中。
语法	vrevh vr _z , vr _x
数据类型	16
说明	将源矢量寄存器 VRX 的元素倒序放入 VRZ 中。
影响标志位	不影响
限制	寄存器范围为 vr ₀ -vr ₁₅ 。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
	111110	0		X	1		VRX		00001101000		0		VRZ

注：

编码表中 VRX,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.111 VREVQ——矢量数据字节倒序指令

统一化指令	
语法	vrevq vrz, vrx
操作	size = 8; number = 128/size; Vz(number-1:0)={Vx(0),Vx(1),Vx(2),Vx(14), Vx(15)}; 把 Vrx 的元素倒序放入 Vrz 之中 vrz[i] 指相应 element 的数据;
编译结果	仅存在 32 位指令。 vrevq vrz, vrx
数据类型	8
说明	将源矢量寄存器 VRX 的元素倒序放入 VRZ 之中。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = 8; number = 128/size; vrz[i] 指相应位置 element 的数据; Vz(number-1:0)={Vx(0),Vx(1),Vx(2),Vx(14), Vx(15)}; 将 Vrx 中元素数据倒序放入 VRZ 中。
语法	vrevq vrz, vrx
数据类型	8
说明	将源矢量寄存器 VRX 的元素倒序放入 VRZ 中。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

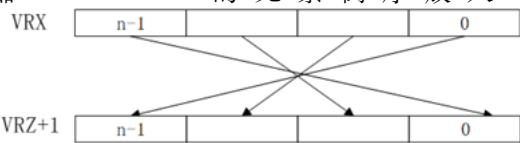
32 位指令格式：

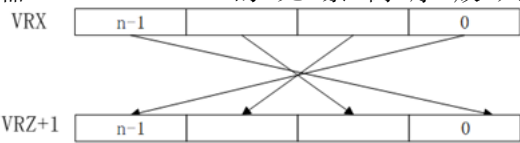
31	op	26	25	24	21	20	19	16	15	5	4	3	0	
111110		0	X		0	VRX				00001101000			0	VRZ

注：

编码表中 VRX,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.112 VREVV——矢量数据字倒序指令

统一化指令	
语法	vrevw vrz, vrx
操作	size = 32; number = 128/size; Vz(number-1:0)={Vx(0),Vx(1),Vx(2),V(3)}; 把 Vrx 的元素倒序放入 Vrz 之中 vrz[i] 指相应 element 的数据;
编译结果	仅存在 32 位指令。 vrevw vrz, vrx
数据类型	32
说明	将源矢量寄存器 VRX 的元素倒序放入 VRZ 之中。 
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = 32; number = 128/size; vrz[i] 指相应位置 element 的数据; Vz(number-1:0)={Vx(0),Vx(1),Vx(2),⋯⋯Vx(3)}; 将 Vrx 中元素数据倒序放入 VRZ 中。
语法	vrevw vrz, vrx
数据类型	32
说明	将源矢量寄存器 VRX 的元素倒序放入 VRZ 中。 
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110	1	X	0	VRX	00001101000	0	VRZ						

注：

编码表中 VRX,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.113 VRMUL.T.SE——矢量扩展带饱和和小数乘法指令

统一化指令	
语法	vrmul.T.se vrz, vrx, vry (T=S8/S16/S32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] for(i=0; i< Number; i++) { If(Vx(i)=-2^(size-1))&&(Vy(i)=-2^(size-1)) Tmp(i)= 2^(2*size-1)-1; Else Tmp(i)= Vx(i)*Vy(i)*2; (乘法结果取全部精度, 即元素位宽的 2 倍) End } for(i=0; i< Number/2; i++) { Vz(2i+1,2i)=Tmp(i); Vz+1(2i+1,2i)=Tmp(Number/2+i); }
编译结果	仅存在 32 位指令。 vrmul.T.se vrz, vrx, vry
数据类型	S8/S16/S32
说明	将 VRX 与 VRY 的对应元素相乘再乘 2 取饱和，乘法结果保留全部精度，低一半元素的相乘结果存放于 VRZ 中，高一半元素的相乘结果存放于 VRZ+1 中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] for(i=0; i< Number; i++) { If (Vx(i)== -2^(size-1)) && (Vy(i)== -2^(size-1)) Tmp(i)= 2^(2*size-1)-1; Else Tmp(i)= Vx(i)*Vy(i)*2; (乘法结果取全部精度，即元素位宽的 2 倍) End i=0:(number-1) } for(i=0; i< Number/2; i++) { Vz(2i+1,2i)=Tmp(i); Vz+1(2i+1,2i)=Tmp(Number/2+i); }
语法	vrmul.T.se vrz, vrx, vry
数据类型	S8/S16/S32
说明	将 VRX 与 VRY 的对应元素相乘再乘 2 取饱和，乘法结果保留全部精度，低一半元素的相乘结果存放于 VRZ 中，高一半元素的相乘结果存放于 VRZ+1 中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1	VRY		w0	VRX		01111100001			S	VRZ	

注：

- 编码表中 w1, w0 用来表征指令位宽类型，其中：
- [w1,w0] = 2’ b00 表示 size 为 8bit 指令；
- [w1,w0] = 2’ b01 表示 size 为 16bit 指令；
- [w1,w0] = 2’ b10 表示 size 为 32bit 指令；
- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.114

VRMULH.T.RS——矢量带饱和取高半舍入小数乘法指令

统一化指令	
语法	vrmulh.T.rs vrz, vr_x, vr_y (T=S8/S16/S32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Round= 1<<(size-1) for(i=0; i< Number; i++) { If (V_x(i)== -2^(size-1)) && (V_y(i)==-2^(size-1)) V_z(i)= 2^(2*size-1)-1; Else V_z(i)= High[V_x(i)*V_y(i))*2+Round]; }
编译结果	仅存在 32 位指令。 vrmulh.T.rs vrz, vr_x, vr_y
数据类型	S8/S16/S32
说明	将 VR _X 与 VR _Y 的对应元素相乘再乘 2 进行舍入操作再取饱和，取乘法结果的高半部分存放于 VR _Z 中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Round= 1<<(size-1) for(i=0; i< Number; i++) { If (Vx(i)== -2^(size-1)) && (Vy(i)== -2^(size-1)) Vz(i)= 2^(2*size-1)-1; Else Vz(i)= High[Vx(i)*Vy(i)]*2+Round];
语法	vrmulh.T.rs vrz, vrx, vry
数据类型	S8/S16/S32
说明	将 VRX 与 VRY 的对应元素相乘再乘 2 进行舍入操作再取饱和，取乘法结果的高半部分存放于 VRZ 中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1	VRY		w0	VRX		01111000100			S	VRZ	

注：

- 编码表中 w1, w0 用来表征指令位宽类型，其中：
- [w1,w0] = 2’ b00 表示 size 为 8bit 指令；
- [w1,w0] = 2’ b01 表示 size 为 16bit 指令；
- [w1,w0] = 2’ b10 表示 size 为 32bit 指令；
- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.115 VRMULH.T.S——矢量带饱和取高半小数乘法指令

统一化指令	
语法	vrmulh.T.s vrz, vrx, vry (T=S8/S16/S32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] for(i=0; i< Number; i++) { If (Vx(i)== -2^(size-1)) && (Vy(i)== -2^(size-1)) Vz(i)= 2^(2*size-1)-1; Else Vz(i)= High[Vx(i)*Vy(i)]*2; } }
编译结果	仅存在 32 位指令。 vrmulh.T.s vrz, vrx, vry
数据类型	S8/S16/S32
说明	将 VRX 与 VRY 的对应元素相乘再乘 2 取饱和，取乘法结果的高半部分存放于 VRZ 中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] for(i=0; i< Number; i++) { If (Vx(i)== -2^(size-1)) && (Vy(i)== -2^(size-1)) Vz(i)= 2^(2*size-1)-1; Else Vz(i)= High[Vx(i)*Vy(i)]*2; } }
语法	vrmulh.T.s vrz, vrx, vry
数据类型	S8/S16/S32
说明	将 VRX 与 VRY 的对应元素相乘再乘 2 取饱和，取乘法结果的高半部分存放于 VRZ 中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15		5	4	3	0
	111110	w1		VRY	w0		VRX		01111000011		S		VRZ	

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2' b00 表示 size 为 8bit 指令；

[w1,w0] = 2' b01 表示 size 为 16bit 指令；

[w1,w0] = 2' b10 表示 size 为 32bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.116 VRMULHA.T.RS——矢量带饱和取高半舍入小数乘累加指令

统一化指令	
语法	vrmulha.T.rs vrz, vrx, vry (T=S8/S16/S32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Round= 1<<(size-1) for(i=0; i< Number; i++) { Tmp(i)=(Vz(i) << size)+Vx(i)*Vy(i))*2+Round; If(Tmp(i)>2^(2*size-1)-1) Vz(i)= 2^(size-1)-1; Else if(Tmp(i)<-2^(2*size-1)) Vz(i)= -2^(size-1); Else Vz(i)=High[Tmp(i)]; }
编译结果	仅存在 32 位指令。 vrmulha.T.rs vrz, vrx, vry
数据类型	S8/S16/S32
说明	将 VRX 与 VRY 的对应元素相乘再乘 2 进行舍入操作，其结果保留全精度，再与 VRZ 左移一个元素位宽大小的值累加，最后取饱和，取其结果的高半部分存放于 VRZ 中
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Round= 1<<(size-1) for(i=0; i< Number; i++) { Tmp(i)=(Vz(i) << size)+Vx(i)*Vy(i))*2+Round; If(Tmp(i)>2^(2*size-1)-1) Vz(i)= 2^(size-1)-1; Else if(Tmp(i)<-2^(2*size-1)) Vz(i)= -2^(size-1); Else Vz(i)=High[Tmp(i)]; }
语法	vrmulha.T.rs vrz, vrx, vry
数据类型	S8/S16/S32
说明	将 VRX 与 VRY 的对应元素相乘再乘 2 进行舍入操作，其结果保留全精度，再与 VRZ 左移一个元素位宽大小的值累加，最后取饱和，取其结果的高半部分存放于 VRZ 中
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1	VRY		w0	VRX		01111010010			S	VRZ	

注：

- 编码表中 w1, w0 用来表征指令位宽类型，其中：
- [w1,w0] = 2’ b00 表示 size 为 8bit 指令；
- [w1,w0] = 2’ b01 表示 size 为 16bit 指令；
- [w1,w0] = 2’ b10 表示 size 为 32bit 指令；
- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.117 VRMULHAI.T.RS——矢量带索引带饱和取高半舍入小数乘累加指令

统一化指令	
语法	vrmulhai.T.rs vrz, vrx, vry,index (T=S8/S16/S32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Index 范围为 [0,Number-1] Round= 1<<(size-1) for(i=0; i< Number; i++) { Tmp(i)=(Vz(i)<<size)+Vx(i)*Vy(index))*2+Round; If(Tmp(i)>2^(2*size-1)-1) Vz(i)= 2^(size-1)-1; Else if(Tmp(i)<-2^(2*size-1)) Vz(i)= -2^(size-1); Else Vz(i)=High[Tmp(i)]; }
编译结果	仅存在 32 位指令。 vrmulhai.T.rs vrz, vrx, vry,index
数据类型	S8/S16/S32
说明	将 VRX 与 index 所索引 VRY 的对应元素相乘再乘 2 进行舍入操作，其结果保留全精度，再与 VRZ 左移一个元素位宽大小的值累加，最后取饱和，取其结果的高半部分存放于 VRZ 中
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Index 范围为 [0,Number-1] Round= 1<<(size-1) for(i=0; i< Number; i++) { Tmp(i)=(Vz(i)<<size)+Vx(i)*Vy(index))*2+Round; If(Tmp(i)>2^(2*size-1)-1) Vz(i)= 2^(size-1)-1; Else if(Tmp(i)<-2^(2*size-1)) Vz(i)= -2^(size-1); Else Vz(i)=High[Tmp(i)]; }
语法	vrmulhai.T.rs vrz, vrx, vry,index
数据类型	S8/S16/S32
说明	将 VRX 与 index 所索引 VRY 的对应元素相乘再乘 2 进行舍入操作，其结果保留全精度，再与 VRZ 左移一个元素位宽大小的值累加，最后取饱和，取其结果的高半部分存放于 VRZ 中
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	8	7	5	4	3	0
111110		w1	VRY		w0	VRX		01010100			IMM3		S	VRZ	

注：

- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；
- 编码表中 I1, I0 与 Imm3 组合在一起用来表征指令位宽类型和 Index，其中：
- IMM[4:0] = {I1,I0,IMM3};
- if(IMM[4:3]=2’ b01) size=16, IMM[2:0] 表示 index0 到 7；
- if(IMM[4:3]=2’ b10) size=32, IMM[2:0] 表示 index0 到 3（高位为 0）
- if(IMM[4:3]=2’ b00) size=8, IMM[2:0] 表示 index0 到 7；
- if(IMM[4:3]=2’ b11) size=8, IMM[2:0] 表示 index8 到 15。

16.118 VRMULHI.T.RS——矢量带索引带饱和取高半舍入小数乘法指令

统一化指令	
语法	vrmulhi.T.rs vrz, vrx, vry,index (T=S8/S16/S32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Round= 1<<(size-1) Index 范围为 [0,Number-1] for(i=0; i< Number; i++) { If (Vx(i)== -2^(size-1)) && (Vy(index)==-2^(size-1)) Vz(i)= 2^(2*size-1)-1; Else Vz(i)= High[Vx(i)*Vy(index)]*2+Round]; }
编译结果	仅存在 32 位指令。 vrmulhi.T.rs vrz, vrx, vry,index
数据类型	S8/S16/S32
说明	将 VRX 与 index 所索引 VRY 的对应元素相乘再乘 2 进行舍入操作再取饱和，取乘法结果的高半部分存放于 VRZ 中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Round= 1<<(size-1) Index 范围为 [0,Number-1] for(i=0; i< Number; i++) { If (Vx(i)== -2^(size-1)) && (Vy(index)==-2^(size-1)) Vz(i)= 2^(2*size-1)-1; Else Vz(i)= High[Vx(i)*Vy(index)]*2+Round];
语法	vrmulhi.T.rs vrz, vrx, vry,index
数据类型	S8/S16/S32
说明	将 VRX 与 index 所索引 VRY 的对应元素相乘再乘 2 进行舍入操作再取饱和，取乘法结果的高半部分存放于 VRZ 中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	8	7	5	4	3	0	
111110			I1	VRY		I0	VRX		01010011			IMM3		S	VRZ	

注：

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

编码表中 I1, I0 与 Imm3 组合在一起用来表征指令位宽类型和 Index，其中：

IMM[4:0] = {I1,I0,IMM3}；

if(IMM[4:3]=2’ b01) size=16, IMM[2:0] 表示 index0 到 7；

if(IMM[4:3]=2’ b10) size=32, IMM[2:0] 表示 index0 到 3（高位为 0）

if(IMM[4:3]=2’ b00) size=8, IMM[2:0] 表示 index0 到 7；

if(IMM[4:3]=2’ b11) size=8, IMM[2:0] 表示 index8 到 15。

16.119 VRMULHI.T.S——矢量带索引带饱和取高半小数乘法指令

统一化指令	
语法	vrmulhi.T.s vrz, vrx, vry,index (T=S8/S16/S32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Index 范围为 [0,Number-1] for(i=0; i< Number; i++) { If (Vx(i)== -2^(size-1)) && (Vy(index)==-2^(size-1)) Vz(i)= 2^(2*size-1)-1; Else Vz(i)= High[Vx(i)*Vy(index))*2]; }
编译结果	仅存在 32 位指令。 vrmulhi.T.s vrz, vrx, vry,index
数据类型	S8/S16/S32
说明	将 VRX 与 index 所索引 VRY 的对应元素相乘再乘 2 取饱和，取乘法结果的高半部分存放于 VRZ 中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] for(i=0; i< Number; i++) { If (Vx(i)== -2^(size-1)) && (Vy(index)==-2^(size-1)) Vz(i)= 2^(2*size-1)-1; Else Vz(i)= High[Vx(i)*Vy(index))*2]; }
语法	vrmulhi.T.s vrz, vrx, vry,index
数据类型	S8/S16/S32
说明	将 VRX 与 index 所索引 VRY 的对应元素相乘再乘 2 取饱和，取乘法结果的高半部分存放于 VRZ 中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	8	7	5	4	3	0	
111110			I1	VRY		I0	VRX		01010010			IMM3		S	VRZ	

注：

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

编码表中 I1, I0 与 Imm3 组合在一起用来表征指令位宽类型和 Index，其中：

$IMM[4:0] = \{I1,I0,IMM3\}$;

if($IMM[4:3]=2'b01$) size=16, IMM[2:0] 表示 index0 到 7;

if($IMM[4:3]=2'b10$) size=32, IMM[2:0] 表示 index0 到 3（高位为 0）

if($IMM[4:3]=2'b00$) size=8, IMM[2:0] 表示 index0 到 7;

if($IMM[4:3]=2'b11$) size=8, IMM[2:0] 表示 index8 到 15。

16.120 VRMULHS.T.RS——矢量带饱和取高半舍入小数乘累减指令

统一化指令	
语法	vrmulhs.T.rs vrz, vrx, vry (T=S8/S16/S32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Round= 1<<(size-1) for(i=0; i< Number; i++) { Tmp(i)=(Vz(i) << size)-Vx(i)*Vy(i))*2+Round; If(Tmp(i)>2^(2*size-1)-1) Vz(i)= 2^(size-1)-1; Else if(Tmp(i)<-2^(2*size-1)) Vz(i)= -2^(size-1); Else Vz(i)=High[Tmp(i)]; }
编译结果	仅存在 32 位指令。 vrmulhs.T.rs vrz, vrx, vry
数据类型	S8/S16/S32
说明	将 VRX 与 VRY 的对应元素相乘再乘 2 进行舍入操作，其结果保留全精度，再与 VRZ 左移一个元素位宽大小的值累减，最后取饱和，取其结果的高半部分存放于 VRZ 中
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Round= 1<<(size-1) for(i=0; i< Number; i++) { Tmp(i)=(Vz(i) << size)-Vx(i)*Vy(i))*2+Round; If(Tmp(i)>2^(2*size-1)-1) Vz(i)= 2^(size-1)-1; Else if(Tmp(i)<-2^(2*size-1)) Vz(i)= -2^(size-1); Else Vz(i)=High[Tmp(i)]; }
语法	vrmulhs.T.rs vrz, vrx, vry
数据类型	S8/S16/S32
说明	将 VRX 与 VRY 的对应元素相乘再乘 2 进行舍入操作，其结果保留全精度，再与 VRZ 左移一个元素位宽大小的值累减，最后取饱和，取其结果的高半部分存放于 VRZ 中
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1	VRY		w0	VRX		01111010011			S	VRZ	

注：

- 编码表中 w1, w0 用来表征指令位宽类型，其中：
- [w1,w0] = 2’ b00 表示 size 为 8bit 指令；
 - [w1,w0] = 2’ b01 表示 size 为 16bit 指令；
 - [w1,w0] = 2’ b10 表示 size 为 32bit 指令；
- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.121

VRMULHSI.T.RS——矢量带索引带饱和取高半舍入小数乘累减指令

统一化指令	
语法	vrmulhsi.T.rs vrz, vrx, vry,index (T=S8/S16/S32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Index 范围为 [0,Number-1] Round= 1<<(size-1) for(i=0; i< Number; i++) { Tmp(i)= (Vz(i)<<size)-Vx(i)*Vy(index))*2+Round; If(Tmp(i)>2^(2*size-1)-1) Vz(i)= 2^(size-1)-1; Else if(Tmp(i)<-2^(2*size-1)) Vz(i)= -2^(size-1); Else Vz(i)=High[Tmp(i)]; }
编译结果	仅存在 32 位指令。 vrmulhsi.T.rs vrz, vrx, vry,index
数据类型	S8/S16/S32
说明	将 VRX 与 index 所索引 VRY 的对应元素相乘再乘 2 进行舍入操作，其结果保留全精度，再与 VRZ 左移一个元素位宽大小的值累减，最后取饱和，取其结果的高半部分存放于 VRZ 中
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Index 范围为 [0,Number-1] Round= 1<<(size-1) for(i=0; i< Number; i++) { Tmp(i)= (Vz(i)<<size)-Vx(i)*Vy(index))*2+Round; If(Tmp(i)>2^(2*size-1)-1) Vz(i)= 2^(size-1)-1; Else if(Tmp(i)<-2^(2*size-1)) Vz(i)= -2^(size-1); Else Vz(i)=High[Tmp(i)]; }
语法	vrmlhsl.T.rs vrz, vrx, vry,index
数据类型	S8/S16/S32
说明	将 VRX 与 index 所索引 VRY 的对应元素相乘再乘 2 进行舍入操作，其结果保留全精度，再与 VRZ 左移一个元素位宽大小的值累减，最后取饱和，取其结果的高半部分存放于 VRZ 中
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	8	7	5	4	3	0
111110		w1	VRY		w0	VRX		01010101			IMM3		S	VRZ	

注：

- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；
- 编码表中 I1, I0 与 Imm3 组合在一起用来表征指令位宽类型和 Index，其中：
IMM[4:0] = {I1,I0,IMM3};
if(IMM[4:3]=2’ b01) size=16, IMM[2:0] 表示 index0 到 7;
if(IMM[4:3]=2’ b10) size=32, IMM[2:0] 表示 index0 到 3（高位为 0）
if(IMM[4:3]=2’ b00) size=8, IMM[2:0] 表示 index0 到 7;
if(IMM[4:3]=2’ b11) size=8, IMM[2:0]。

16.122 VRMULI.T.SE——矢量带索引扩展带饱和小数乘法指令

统一化指令	
语法	vrmulti.T.se vrz, vrx, vry,index (T=S8/S16/S32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Index 范围为 [0,Number-1] for(i=0; i< Number; i++) { If (Vx(i)== -2^(size-1)) && (Vy(index)==-2^(size-1)) Tmp(i)= 2^(2*size-1)-1; Else Tmp(i)= Vx(i)*Vy(index))*2; (乘法结果取全部精度, 即元素位宽的 2 倍) End } for(i=0; i< Number/2; i++) { Vz(2i+1,2i)=Tmp(i); Vz+1(2i+1,2i)=Tmp(Number/2+i); } }
编译结果	仅存在 32 位指令。 vrmulti.T.se vrz, vrx, vry,index
数据类型	S8/S16/S32
说明	将 VRX 与 Index 所索引 VRY 的对应元素相乘再乘 2 取饱和，乘法结果保留全部精度，低一半元素的相乘结果存放于 VRZ 中，高一半元素的相乘结果存放于 VRZ+1 中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Index 范围为 [0,Number-1] for(i=0; i< Number; i++) { If (Vx(i)== -2^(size-1)) && (Vy(index)==-2^(size-1)) Tmp(i)= 2^(2*size-1)-1; Else Tmp(i)= Vx(i)*Vy(index))*2; (乘法结果取全部精度, 即元素位宽的 2 倍) End i=0:(number-1) } for(i=0; i< Number/2; i++) { Vz(2i+1,2i)=Tmp(i); Vz+1(2i+1,2i)=Tmp(Number/2+i); } }
语法	vrmuli.T.se vrz, vrx, vry,index
数据类型	S8/S16/S32
说明	将 VRX 与 Index 所索引 VRY 的对应元素相乘再乘 2 取饱和，乘法结果保留全部精度，低一半元素的相乘结果存放于 VRZ 中，高一半元素的相乘结果存放于 VRZ+1 中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	8	7	5	4	3	0
111110		I1	VRY		I0	VRX		01100001			IMM3		S	VRZ	

注：

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

编码表中 I1, I0 与 Imm3 组合在一起用来表征指令位宽类型和 Index，其中：

IMM[4:0] = {I1,I0,IMM3}；

if(IMM[4:3]=2’ b01) size=16, IMM[2:0] 表示 index0 到 7；

if(IMM[4:3]=2’ b10) size=32, IMM[2:0] 表示 index0 到 3（高位为 0）

if(IMM[4:3]=2’ b00) size=8, IMM[2:0] 表示 index0 到 7；

if(IMM[4:3]=2’ b11) size=8, IMM[2:0] 表示 index8 到 15。

16.123 VRMULSA.T.E——矢量扩展带移位小数乘累加指令

统一化指令	
语法	vrmulsa.T.e vrz, vrx, vry,imm4 (T=S8/S16/S32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Imm4 表示立即数，范围为 [0,15] for(i=0; i< Number/2; i++) { Vz(2i+1:2i)= Vz(2i+1:2i)+(Vx(i)*Vy(i))>>imm; Vz+1(2i+1:2i)= Vz+1(2i+1:2i)+(Vx(Number/2+i)*Vy(Number/2+i))>>imm; } 乘法结果保留全部精度后进行算数右移
编译结果	仅存在 32 位指令。 Vrmulsa.T.e vrz, vrx, vry,imm4
数据类型	S8/S16/S32
说明	将 VRX 与 VRY 的对应元素相乘，乘法结果保留全部精度，低一半元素的相乘结果进行算数右移后与 VRZ 对应元素累加，结果存放于 VRZ 中，高一半元素的相乘结果进行算数右移后与 VRZ+1 对应元素累加，结果存放于 VRZ+1 中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Imm4 表示立即数，范围为 [0,15] for(i=0; i< Number/2; i++) { Vz(2i+1:2i)= Vz(2i+1:2i)+(Vx(i)*Vy(i))>>imm; Vz+1(2i+1:2i)= Vz+1(2i+1:2i)+(Vx(Number/2+i)*Vy(Number/2+i))>>imm; } 乘法结果保留全部精度后进行算数右移
语法	vrmulsa.T.e vrz, vrx, vry,imm4
数据类型	S8/S16/S32
说明	将 VRX 与 VRY 的对应元素相乘，乘法结果保留全部精度，低一半元素的相乘结果进行算数右移后与 VRZ 对应元素累加，结果存放于 VRZ 中，高一半元素的相乘结果进行算数右移后与 VRZ+1 对应元素累加，结果存放于 VRZ+1 中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op		26	25	24	21	20	19	16	15	5		8	5		4	3	0	
111110				w1		VRY		w0		VRX		0111000			IMM4		S	VRZ	

注：

- 编码表中 w1, w0 用来表征指令位宽类型，其中：
- [w1,w0] = 2’ b00 表示 size 为 8bit 指令；
- [w1,w0] = 2’ b01 表示 size 为 16bit 指令；
- [w1,w0] = 2’ b10 表示 size 为 32bit 指令；
- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 IMM4 表征立即数；
- 编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.124 VRMULSAI.T.E——矢量带索引扩展带移位小数乘累加指令

统一化指令	
语法	vrmul sai.T.e vrz, vrx, imm4,index (T=S8/S16/S32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Index 范围为 [0,Number-1] Imm4 表示立即数，范围为 [0,15] for(i=0; i< Number/2; i++) { Vz(2i+1:2i)=Vz(2i+1:2i)+(Vx(i)*Vx+1(index))>>imm; Vz+1(2i+1:2i)=Vz(2i+1:2i)+(Vx(Number/2+i)*Vx+1(index))>>imm; } 乘法结果保留全部精度后进行算数右移
编译结果	仅存在 32 位指令。 vrmul sai.T.e vrz,vry,imm4,index
数据类型	S8/S16/S32
说明	将 VRX 与 Index 所索引 VRX+1 的对应元素相乘，乘法结果保留全部精度，低一半元素的相乘结果进行算数右移后与 VRZ 对应元素累加，结果存放于 VRZ 中，高一半元素的相乘结果进行算数右移后与 VRZ+1 对应元素累加，结果存放于 VRZ+1 中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Index 范围为 [0,Number-1] Imm4 表示立即数，范围为 [0,15] for(i=0; i< Number/2; i++) { Vz(2i+1:2i)=Vz(2i+1:2i)+(Vx(i)*Vx+1(index))>>imm; Vz+1(2i+1:2i)=Vz(2i+1:2i)+Vx(Number/2+i)*Vx+1(index)>>imm; } 乘法结果保留全部精度后进行算数右移
语法	vrmul sai.T.e vrz, vrx, imm4,index
数据类型	S8/S16/S32
说明	将 VRX 与 Index 所索引 VRX+1 的对应元素相乘，乘法结果保留全部精度，低一半元素的相乘结果进行算数右移后与 VRZ 对应元素累加，结果存放于 VRZ 中，高一半元素的相乘结果进行算数右移后与 VRZ+1 对应元素累加，结果存放于 VRZ+1 中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	8	7	5	4	3	0
111110		I1		IMM4		I0		VRX		01110101		IMM3		S	VRZ

注：

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 IMM4 表征立即数；

编码表中 VRX,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

编码表中 I1, I0 与 Imm3 组合在一起用来表征指令位宽类型和 Index，其中：

IMM[4:0] = {I1,I0,IMM3}；

if(IMM[4:3]=2’ b01) size=16, IMM[2:0] 表示 index0 到 7；

if(IMM[4:3]=2’ b10) size=32, IMM[2:0] 表示 index0 到 3（高位为 0）

if(IMM[4:3]=2’ b00) size=8, IMM[2:0] 表示 index0 到 7；

if(IMM[4:3]=2’ b11) size=8, IMM[2:0] 表示 index8 到 15。

16.125 VRMULSHR.T.E——矢量扩展带移位小数乘法指令

统一化指令	
语法	vrmulshr.T.e vrz, vrx, vry,imm4 (T=S8/S16/S32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Imm4 表示立即数，范围为 [0,15] for(i=0; i< Number/2; i++) { Vz(2i+1:2i)= (Vx(i)*Vy(i))>>imm; Vz+1(2i+1:2i)=Vx(Number/2+i)*Vy(Number/2+i)>>imm; } 乘法结果保留全部精度后进行算数右移
编译结果	仅存在 32 位指令。 vrmulshr.T.e vrz, vrx, vry,imm4
数据类型	S8/S16/S32
说明	将 VRX 与 VRY 的对应元素相乘，乘法结果保留全部精度，低一半元素的相乘结果进行算数右移后结果存放于 VRZ 中，高一半元素的相乘结果进行算数右移后结果存放于 VRZ+1 中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Imm4 表示立即数，范围为 [0,15] for(i=0; i< Number/2; i++) { Vz(2i+1:2i)= (Vx(i)*Vy(i))>>imm; Vz+1(2i+1:2i)=Vx(Number/2+i)*Vy(Number/2+i))>>imm; } 乘法结果保留全部精度后进行算数右移
语法	vrmulshr.T.e vrz, vrx, vry,imm4
数据类型	S8/S16/S32
说明	将 VRX 与 VRY 的对应元素相乘，乘法结果保留全部精度，低一半元素的相乘结果进行算数右移后结果存放于 VRZ 中，高一半元素的相乘结果进行算数右移后结果存放于 VRZ+1 中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	8	5	4	3	0
111110		w1	VRY			w0	VRX			0110001		IMM4		S	VRZ

注：

- 编码表中 w1, w0 用来表征指令位宽类型，其中：
- [w1,w0] = 2’ b00 表示 size 为 8bit 指令；
- [w1,w0] = 2’ b01 表示 size 为 16bit 指令；
- [w1,w0] = 2’ b10 表示 size 为 32bit 指令；
- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 IMM4 表征立即数；
- 编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.126 VRMULSHRI.T.E——矢量带索引扩展带移位小数乘法指令

统一化指令	
语法	vrmulshri.T.e vrz, vrx, imm4,index (T=S8/S16/S32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Index 范围为 [0,Number-1] Imm4 表示立即数，范围为 [0,15] for(i=0; i< Number/2; i++) { Vz(2i+1:2i)= (Vx(i)*Vx+1(index))>>imm; Vz+1(2i+1:2i)= Vx(Number/2+i)*Vx+1(index))>>imm; } 乘法结果保留全部精度后进行算数右移
编译结果	仅存在 32 位指令。 vrmulshri.T.e vrz,vry,imm4,index
数据类型	S8/S16/S32
说明	将 VRX 与 Index 所索引 VRX+1 的对应元素相乘，乘法结果保留全部精度，低一半元素的相乘结果进行算数右移后结果存放于 VRZ 中，高一半元素的相乘结果进行算数右移后结果存放于 VRZ+1 中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Index 范围为 [0,Number-1] Imm4 表示立即数，范围为 [0,15] for(i=0; i< Number/2; i++) { Vz(2i+1:2i)= (Vx(i)*Vx+1(index))>>imm; Vz+1(2i+1:2i)=Vx(Number/2+i)*Vx+1(index)>>imm; } 乘法结果保留全部精度后进行算数右移
语法	vrmulshri.T.e vrz, vrx, imm4,index
数据类型	S8/S16/S32
说明	将 VRX 与 Index 所索引 VRX+1 的对应元素相乘，乘法结果保留全部精度，低一半元素的相乘结果进行算数右移后结果存放于 VRZ 中，高一半元素的相乘结果进行算数右移后结果存放于 VRZ+1 中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	8	7	5	4	3	0
111110		I1		IMM4		I0		VRX		01100100		IMM3		S	VRZ

注：

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 IMM4 表征立即数；

编码表中 VRX,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

编码表中 I1, I0 与 Imm3 组合在一起用来表征指令位宽类型和 Index，其中：

IMM[4:0] = {I1,I0,IMM3}；

if(IMM[4:3]=2’ b01) size=16, IMM[2:0] 表示 index0 到 7；

if(IMM[4:3]=2’ b10) size=32, IMM[2:0] 表示 index0 到 3（高位为 0）

if(IMM[4:3]=2’ b00) size=8, IMM[2:0] 表示 index0 到 7；

if(IMM[4:3]=2’ b11) size=8, IMM[2:0] 表示 index8 到 15。

16.127 VRMULSS.T.E——矢量扩展带移位小数乘负累加指令

统一化指令	
语法	vrmulss.T.e vrz, vrx, vry,imm4 (T=S8/S16/S32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Imm4 表示立即数，范围为 [0,15] for(i=0; i< Number/2; i++) { $V_z(2i+1:2i)=V_z(2i+1:2i)+(-V_x(i)*V_y(i))>>imm;$ $V_{z+1}(2i+1:2i)=$ $V_{z+1}(2i+1:2i)+(-V_x(Number/2+i)*V_y(Number/2+i))>>imm;$ } 乘法结果保留全部精度后进行算数右移。
编译结果	仅存在 32 位指令。 Vrmulss.T.e vrz, vrx, vry,imm4
数据类型	S8/S16/S32
说明	将 VRX 与 VRY 的对应元素相乘取负，乘法结果保留全部精度，低一半元素的相乘结果进行算数右移后与 VRZ 对应元素累加，结果存放于 VRZ 中，高一半元素的相乘结果进行算数右移后与 VRZ+1 对应元素累加，结果存放于 VRZ+1 中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Imm4 表示立即数，范围为 [0,15] for(i=0; i< Number/2; i++) { Vz(2i+1:2i)= Vz(2i+1:2i)+(-Vx(i)*Vy(i))>>imm; Vz+1(2i+1:2i)=Vz+1(2i+1:2i)+(-Vx(Number/2+i)*Vy(Number/2+i))>>imm; } 乘法结果保留全部精度后进行算数右移
语法	vrmulss.T.e vrz, vrx, vry,imm4
数据类型	S8/S16/S32
说明	将 VRX 与 VRY 的对应元素相乘取负，乘法结果保留全部精度，低一半元素的相乘结果进行算数右移后与 VRZ 对应元素累加，结果存放于 VRZ 中，高一半元素的相乘结果进行算数右移后与 VRZ+1 对应元素累加，结果存放于 VRZ+1 中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	8	5	4	3	0
111110		w1		VRY		w0		VRX		0111011		IMM4		S	VRZ

注：

- 编码表中 w1, w0 用来表征指令位宽类型，其中：
- [w1,w0] = 2’ b00 表示 size 为 8bit 指令；
- [w1,w0] = 2’ b01 表示 size 为 16bit 指令；
- [w1,w0] = 2’ b10 表示 size 为 32bit 指令；
- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 IMM4 表征立即数；
- 编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.128 VRMULSSI.T.E——矢量带索引扩展带移位小数乘负累加指令

统一化指令	
语法	vrmulssi.T.e vrz, vrx, imm4,index (T=S8/S16/S32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Index 范围为 [0,Number-1] Imm4 表示立即数，范围为 [0,15] for(i=0; i< Number/2; i++) { Vz(2i+1:2i)=Vz(2i+1:2i)+(-Vx(i)*Vx+1(index))>>imm; Vz+1(2i+1:2i)=Vz(2i+1:2i)+(-Vx(Number/2+i)*Vx+1(index))>>imm; } 乘法结果保留全部精度后进行算数右移
编译结果	仅存在 32 位指令。 vrmulssi.T.e vrz,vry,imm4,index
数据类型	S8/S16/S32
说明	将 VRX 与 Index 所索引 VRX+1 的对应元素相乘取负，乘法结果保留全部精度，低一半元素的相乘结果进行算数右移后与 VRZ 对应元素累加，结果存放于 VRZ 中，高一半元素的相乘结果进行算数右移后与 VRZ+1 对应元素累加，结果存放于 VRZ+1 中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Index 范围为 [0,Number-1] Imm4 表示立即数，范围为 [0,15] for(i=0; i< Number/2; i++) { Vz(2i+1:2i)=Vz(2i+1:2i)+(-Vx(i)*Vx+1(index))>>imm; Vz+1(2i+1:2i)=Vz(2i+1:2i)+(-Vx(Number/2+i)*Vx+1(index))>>imm; } 乘法结果保留全部精度后进行算数右移
语法	vrmulssi.T.e vrz, vrx, imm4,index
数据类型	S8/S16/S32
说明	将 VRX 与 Index 所索引 VRX+1 的对应元素相乘取负，乘法结果保留全部精度，低一半元素的相乘结果进行算数右移后与 VRZ 对应元素累加，结果存放于 VRZ 中，高一半元素的相乘结果进行算数右移后与 VRZ+1 对应元素累加，结果存放于 VRZ+1 中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	8	7	5	4	3	0
111110		I1	IMM4		I0	VRX		01110100			IMM3		S	VRZ	

注：

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 IMM4 表征立即数；

编码表中 VRX,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

编码表中 I1, I0 与 Imm3 组合在一起用来表征指令位宽类型和 Index，其中：

IMM[4:0] = {I1,I0,IMM3}；

if(IMM[4:3]=2’ b01) size=16, IMM[2:0] 表示 index0 到 7；

if(IMM[4:3]=2’ b10) size=32, IMM[2:0] 表示 index0 到 3（高位为 0）

if(IMM[4:3]=2’ b00) size=8, IMM[2:0] 表示 index0 到 7；

if(IMM[4:3]=2’ b11) size=8, IMM[2:0] 表示 index8 到 15。

16.129 VRMULXAA.T.RS——矢量带饱和复数实部虚部交叉相乘累加累加取高半舍入指令

统一化指令	
语法	vrmulxaa.T.rs vz, vx, vy (T=S8/S16/S32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Round= 1<<(size-1) for(i=0; i< Number/2; i++) { Tmp(2i+1)=Vz(2i+1)<<size+Vx(2i)*Vy(2i+1)*2+round; Tmp(2i)=Vz(2i)<<size+ Vx(2i)*Vy(2i)*2+round; } 注： Tmp(i) 保留全部精度 for(i=0; i< Number; i++) If(Tmp(i)>2^(2*size-1)-1) Vz(i)= 2^(size-1)-1; Else if(Tmp(i)<-2^(2*size-1)) Vz(i)= -2^(size-1); Else Vz(i)=High[Tmp(i)]; } 注： 饱和在累加后进行
编译结果	仅存在 32 位指令。 vrmulxaa.T.rs vz, vx, vy
数据类型	S8/S16/S32
说明	复数的实部和虚部部分计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；将 VRX 和 VRY 的对应元素交织相乘后再乘 2** 进行舍入操作，保留所得乘积的所有精度后，与 VRZ 的对应元素相加，最后取饱和，结果写回到 VRZ 的对应元素中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Round= 1<<(size-1) for(i=0; i< Number/2; i++) { Tmp(2i+1)=Vz(2i+1)<<size+Vx(2i)*Vy(2i+1)*2+round; Tmp(2i)=Vz(2i)<<size+ Vx(2i)*Vy(2i)*2+round; } 注：Tmp(i) 保留全部精度 for(i=0; i< Number; i++) If(Tmp(i)>2^(2*size-1)-1) Vz(i)= 2^(size-1)-1; Else if(Tmp(i)<-2^(2*size-1)) Vz(i)= -2^(size-1); Else Vz(i)=High[Tmp(i)]; } 注：饱和在累加后进行
语法	vrmlxaa.T.rs vz, vx, vy
数据类型	S8/S16/S32
说明	复数的实部和虚部部分计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；将 VRX 和 VRY 的对应元素交织相乘后再乘 2** 进行舍入操作，保留所得乘积的所有精度后，与 VRZ 的对应元素相加，最后取饱和，结果写回到 VRZ 的对应元素中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op		26	25	24	21	20	19	16	15	5			4	3	0	
111110			w1		VRY		w0		VRX		01111010100			S		VRZ	

注：

- 编码表中 w1, w0 用来表征指令位宽类型，其中：
- [w1,w0] = 2’ b00 表示 size 为 8bit 指令；
- [w1,w0] = 2’ b01 表示 size 为 16bit 指令；
- [w1,w0] = 2’ b10 表示 size 为 32bit 指令；
- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.130

VRMULXAAI.T.RS——

矢量带索引带饱和复数实部虚部交叉相乘累加累加取高半舍入指令

统一化指令	
语法	vrmulxaai.T.rs vz, vx, vy,index (T=S8/S16/S32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Index 范围为 [0,Number/2-1] Round= 1<<(size-1) for(i=0; i< Number/2; i++) { Tmp(2i+1)=Vz(2i+1)<<size+Vx(2i)*Vy(2index+1)*2+round; Tmp(2i)=Vz(2i)<<size+ Vx(2i)*Vy(2index)*2+round; } 注：Tmp(i) 保留全部精度 for(i=0; i< Number; i++) If(Tmp(i)>2^(2*size-1)-1) Vz(i)= 2^(size-1)-1; Else if(Tmp(i)<-2^(2*size-1)) Vz(i)= -2^(size-1); Else Vz(i)=High[Tmp(i)]; } 注：饱和在累加后进行
编译结果	仅存在 32 位指令。 vrmulxaai.T.rs vz, vx, vy,index
数据类型	S8/S16/S32
说	复数的实部和虚部部分计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；将 VRX 和 index 索引 VRY 的对应元素交织相乘后再乘 2 进行舍入操作，保留所得到乘积的所有精度后，与 VRZ 的对应元素相加，最后取饱和，结果写回到 VRZ 的对应元素中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Index 范围为 [0,Number/2-1] Round= 1<<(size-1) for(i=0; i< Number/2; i++) { Tmp(2i+1)=Vz(2i+1)<<size+Vx(2i)*Vy(2index+1)*2+round; Tmp(2i)=Vz(2i)<<size+ Vx(2i)*Vy(2index)*2+round; } 注：Tmp(i) 保留全部精度 for(i=0; i< Number; i++) If(Tmp(i)>2^(2*size-1)-1) Vz(i)= 2^(size-1)-1; Else if(Tmp(i)<-2^(2*size-1)) Vz(i)= -2^(size-1); Else Vz(i)=High[Tmp(i)]; } 注：饱和在累加后进行
语法	vrmlxaai.Trs vz, vx, vy,index
数据类型	S8/S16/S32
说明	复数的实部和虚部部分计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；将 VRX 和 index 索引 VRY 的对应元素交织相乘后再乘 2 进行舍入操作，保留所得到乘积的所有精度后，与 VRZ 的对应元素相加，最后取饱和，结果写回到 VRZ 的对应元素中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	8	7	6	5	4	3	0
111110		I1	VRY		I0	VRX		01010110			IMM2	0	S	VRZ		

注：

IMM[3:0] = {I1,I0,IMM2};

if(IMM[3:2]=2’ b01) size=16, IMM[1:0] 表示 index0 到 3;

if(IMM[3:2]=2’ b10) size=32, IMM[1:0] 表示 index0 到 1(高位为 0);

if(IMM[3:2]=2’ b00) size=8, IMM[1:0] 表示 index0 到 3;

if(IMM[3:2]=2’ b11) size=8, IMM[1:0] 表示 index4 到 7;

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.131

VRMULXAS.T.RS——矢量带饱和复数实部虚部交叉相乘累加
累减取高半舍入指令

统一化指令	
语法	vrmulxas.T.rs vz, vx, vy (T=S8/S16/S32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Round= 1<<(size-1) for(i=0; i< Number/2; i++) { Ttmp(2i+1)=Vz(2i+1)<<size+Vx(2i+1)*Vy(2i)*2+round; Ttmp(2i)=Vz(2i)<<size-Vx(2i+1)*Vy(2i+1)*2+round; } 注：Ttmp(i) 保留全部精度 for(i=0; i< Number; i++) If(Ttmp(i)>2^(2*size-1)-1) Vz(i)= 2^(size-1)-1; Else if(Ttmp(i)<-2^(2*size-1)) Vz(i)= -2^(size-1); Else Vz(i)=High[Ttmp(i)]; } 注：饱和在累加累减后进行
编译结果	仅存在 32 位指令。 vrmulxas.T.rs vz, vx, vy
数据类型	S8/S16/S32
说明	复数的实部和虚部部分计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位； 将 VRX 和 VRY 的对应元素交织相乘后再乘 2 进行舍入操作，虚数部分保留所得到乘积的所有精度后，与 VRZ 的虚数部分对应元素相加，最后再取饱和；实数部分保留所得到乘积的所有精度并取负后，与 VRZ 的实数部分相加再取饱和，结果写回到 VRZ 的对应元素中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Round= 1<<(size-1) for(i=0; i< Number/2; i++) { Tmp(2i+1)=Vz(2i+1)<<size+Vx(2i)*Vy(2i+1)*2+round; Tmp(2i)=Vz(2i)<<size+ Vx(2i)*Vy(2i)*2+round; } 注：Tmp(i) 保留全部精度 for(i=0; i< Number; i++) If(Tmp(i)>2^(2*size-1)-1) Vz(i)= 2^(size-1)-1; Else if(Tmp(i)<-2^(2*size-1)) Vz(i)= -2^(size-1); Else Vz(i)=High[Tmp(i)]; } 注：饱和在累加后进行
语法	vrmulxas.T.rs vz, vx, vy
数据类型	S8/S16/S32
说明	复数的实部和虚部部分计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；将 VRX 和 VRY 的对应元素交织相乘后再乘 2 进行舍入操作，虚数部分保留所得到乘积的所有精度后，与 VRZ 的虚数部分对应元素相加，最后再取饱和；实数部分保留所得到乘积的所有精度并取负后，与 VRZ 的实数部分相加再取饱和，结果写回到 VRZ 的对应元素中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1	VRY		w0	VRX		01111010100			S	VRZ	

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX, VRY, VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.132

VRMULXASI.T.RS——

矢量带索引带饱和复数实部虚部交叉相乘累加累减取高半舍入指令

统一化指令	
语法	vrmulxasi.T.rs vz, vx, vy, index (T=S8/S16/S32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Round= 1<<(size-1) Index 范围为 [0,Number/2-1] for(i=0; i< Number/2; i++) { Tmp(2i+1)=Vz(2i+1)<<size+Vx(2index+1)*Vy(2i)*2+round; Tmp(2i)=Vz(2i)<<size-Vx(2i+1)*Vy(2index+1)*2+round; } 注：Tmp(i) 保留全部精度 for(i=0; i< Number; i++) If(Tmp(i)>2^(2*size-1)-1) Vz(i)= 2^(size-1)-1; Else if(Tmp(i)<-2^(2*size-1)) Vz(i)= -2^(size-1); Else Vz(i)=High[Tmp(i)]; } 注：饱和在累加累减后进行
编译结果	仅存在 32 位指令。 vrmulxasi.T.rs vz, vx, vy, index
数据类型	S8/S16/S32
说明	复数的实部和虚部部分计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；将 VRX 和 index 索引 VRY 的对应元素交织相乘后再乘 2 进行舍入操作，虚数部分保留所得到乘积的所有精度后，与 VRZ 的虚数部分对应元素相加，最后再取饱和；实数部分保留所得到乘积的所有精度并取负后，与 VRZ 的实数部分相加再取饱和，结果写回到 VRZ 的对应元素中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Round= 1<<(size-1) Index 范围为 [0,Number/2-1] for(i=0; i< Number/2; i++) { Tmp(2i+1)=Vz(2i+1)<<size+Vx(2index+1)*Vy(2i)*2+round; Tmp(2i)=Vz(2i)<<size-Vx(2i+1)*Vy(2index+1)*2+round; } 注： Tmp(i) 保留全部精度 for(i=0; i< Number; i++) If(Tmp(i)>2^(2*size-1)-1) Vz(i)= 2^(size-1)-1; Else if(Tmp(i)<-2^(2*size-1)) Vz(i)= -2^(size-1); Else Vz(i)=High[Tmp(i)]; } 注：饱和在累加累减后进行
语法	vrmulxasi.T.rs vz, vx, vy,index
数据类型	S8/S16/S32
说明	复数的实部和虚部部分计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；将 VRX 和 index 索引 VRY 的对应元素交织相乘后再乘 2 进行舍入操作，虚数部分保留所得到乘积的所有精度后，与 VRZ 的虚数部分对应元素相加，最后再取饱和；实数部分保留所得到乘积的所有精度并取负后，与 VRZ 的实数部分相加再取饱和，结果写回到 VRZ 的对应元素中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	8	7	6	5	4	3	0
111110		I1	VRY		I0	VRX		01010111			IMM2	0	S	VRZ		

注：

编码表中 I1, I0 与 Imm2 组合在一起用来表征指令位宽类型和 Index，其中：

IMM[3:0] = {I1,I0,IMM2};

if(IMM[3:2]=2’ b01) size=16, IMM[1:0] 表示 index0 到 3;

if(IMM[3:2]=2' b10) size=32, IMM[1:0] 表示 index0 到 1(高位为 0);

if(IMM[3:2]=2' b00) size=8, IMM[1:0] 表示 index0 到 3;

if(IMM[3:2]=2' b11) size=8, IMM[1:0] 表示 index4 到 7;

编码表中 S 表征带符号运算还是无符号运算: 1 表示带符号, 0 表示无符号;

编码表中 VRX, VRY, VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.133 VRMULXSA.T.RS——矢量带饱和复数实部虚部交叉相乘累减
累加取高半舍入指令

统一化指令	
语法	vrmulxsa.T.rs vz, vx, vy (T=S8/S16/S32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Round= 1<<(size-1) for(i=0; i< Number/2; i++) { Tmp(2i+1)=Vz(2i+1)<<size-Vx(2i+1)*Vy(2i)*2+round; Tmp(2i)=Vz(2i)<<size+Vx(2i+1)*Vy(2i+1)*2+round; } 注：Tmp(i) 保留全部精度 for(i=0; i< Number; i++) If(Tmp(i)>2^(2*size-1)-1) Vz(i)= 2^(size-1)-1; Else if(Tmp(i)<-2^(2*size-1)) Vz(i)= -2^(size-1); Else Vz(i)=High[Tmp(i)]; } 注：饱和在累加累减后进行
编译结果	仅存在 32 位指令。 vrmulxsa.T.rs vz, vx, vy
数据类型	S8/S16/S32
说明	复数的实部和虚部部分计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位； 将 VRX 和 VRY 的对应元素交织相乘后再乘 2 进行舍入操作，虚数部分保留所得到乘积的所有精度后取负，与 VRZ 的虚数部分对应元素相加，最后再取饱和；实数部分保留所得到乘积的所有精度，与 VRZ 的实数部分相加再取饱和，结果写回到 VRZ 的对应元素中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Round= 1<<(size-1) for(i=0; i< Number/2; i++) { Tmp(2i+1)=Vz(2i+1)<<size-Vx(2i+1)*Vy(2i)*2+round; Tmp(2i)=Vz(2i)<<size+Vx(2i+1)*Vy(2i+1)*2+round; } 注：Tmp(i) 保留全部精度 for(i=0; i< Number; i++) If(Tmp(i)>2^(2*size-1)-1) Vz(i)= 2^(size-1)-1; Else if(Tmp(i)<-2^(2*size-1)) Vz(i)= -2^(size-1); Else Vz(i)=High[Tmp(i)]; } 注：饱和在累加累减后进行
语法	vrmlksa.T.rs vz, vx, vy
数据类型	S8/S16/S32
说明	复数的实部和虚部部分计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；将 VRX 和 VRY 的对应元素交织相乘后再乘 2 进行舍入操作，虚数部分保留所得到乘积的所有精度后取负，与 VRZ 的虚数部分对应元素相加，最后再取饱和；实数部分保留所得到乘积的所有精度，与 VRZ 的实数部分相加再取饱和，结果写回到 VRZ 的对应元素中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1	VRY			w0	VRX			01111010111		S	VRZ

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX, VRY, VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.134 VRMULXSai.T.RS——矢量带索引带饱和复数实部虚部交叉相乘累加累减取高半舍入指令

统一化指令	
语法	vrmulxsai.T.rs vz, vx, vy,index (T=S8/S16/S32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Round= 1<<(size-1) Index 范围为 [0,Number/2-1] for(i=0; i< Number/2; i++) { T _{mp} (2i+1)=V _z (2i+1)<<size-V _x (2index+1)*V _y (2i)*2+round; T _{mp} (2i)=V _z (2i)<<size+V _x (2i+1)*V _y (2index+1)*2+round; } 注：T _{mp} (i) 保留全部精度 for(i=0; i< Number; i++) If(T _{mp} (i)>2 ^(2*size-1) -1) V _z (i)= 2 ^(size-1) -1; Else if(T _{mp} (i)<-2 ^(2*size-1)) V _z (i)= -2 ^(size-1) ; Else V _z (i)=High[T _{mp} (i)]; } 注：饱和在累加累减后进行
编译结果	仅存在 32 位指令。 vrmulxsai.T.rs vz, vx, vy,index
数据类型	S8/S16/S32
说明	复数的实部和虚部部分计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；将 VRX 和 index 索引 VRY 的对应元素交织相乘后再乘 2 进行舍入操作，虚数部分保留所得到乘积的所有精度后取负，与 VRZ 的虚数部分对应元素相加，最后再取饱和；实数部分保留所得到乘积的所有精度后，与 VRZ 的实数部分相加再取饱和，结果写回到 VRZ 的对应元素中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Round= 1<<(size-1) Index 范围为 [0,Number/2-1] for(i=0; i< Number/2; i++) { Tmp (2i+1)=Vz(2i+1)<<size-Vx(2index+1)*Vy(2i)*2+round; T mp(2i)=Vz(2i)<<size+Vx(2i+1)*Vy(2index+1)*2+round; } 注： Tmp(i) 保留全部精度 for(i=0; i< Number; i++) If(Tmp(i)>2^(2*size-1)-1) Vz(i)= 2^(size-1)-1; Else if(Tmp(i)<-2^(2*size-1)) Vz(i)= -2^(size-1); Else Vz(i)=High[Tmp(i)]; } 注：饱和在累加累减后进行
语法	vrmulxasi.T.rs vz, vx, vy,index
数据类型	S8/S16/S32
说明	复数的实部和虚部部分计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；将 VRX 和 index 索引 VRY 的对应元素交织相乘后再乘 2 进行舍入操作，虚数部分保留所得到乘积的所有精度后取负，与 VRZ 的虚数部分对应元素相加，最后再取饱和；实数部分保留所得到乘积的所有精度后，与 VRZ 的实数部分相加再取饱和，结果写回到 VRZ 的对应元素中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	8	7	6	5	4	3	0
111110		I1	VRY			I0	VRX			01110101		IMM2	0	S	VRZ	

注：

编码表中 I1, I0 与 Imm2 组合在一起用来表征指令位宽类型和 Index，其中：

IMM[3:0] = {I1,I0,IMM2}；

if(IMM[3:2]=2’ b01) size=16, IMM[1:0] 表示 index0 到 3；

if(IMM[3:2]=2’ b10) size=32, IMM[1:0] 表示 index0 到 1(高位为 0)；

if(IMM[3:2]=2' b00) size=8, IMM[1:0] 表示 index0 到 3;

if(IMM[3:2]=2' b11) size=8, IMM[1:0] 表示 index4 到 7;

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.135

VRMULXSS.T.RS——矢量带饱和复数实部虚部交叉相乘累减累减取高半舍入指令

统一化指令	
语法	vrmulxss.T.rs vz, vx, vy (T=S8/S16/S32)
操作	Number=128/size i 表征第几个元素, [0,Number-1] Round= 1<<(size-1) for(i=0; i< Number/2; i++) { Ttmp(2i+1)=Vz(2i+1)<<size-Vx(2i)*Vy(2i+1)*2+round; Ttmp(2i)=Vz(2i)<<size- Vx(2i)*Vy(2i)*2+round; } 注： Tmp(i) 保留全部精度 for(i=0; i< Number; i++) If(Tmp(i)>2^(2*size-1)-1) Vz(i)= 2^(size-1)-1; Else if(Tmp(i)<-2^(2*size-1)) Vz(i)= -2^(size-1); Else Vz(i)=High[Tmp(i)]; } 注： 饱和在累加后进行
编译结果	仅存在 32 位指令。 vrmulxss.T.rs vz, vx, vy
数据类型	S8/S16/S32
说明	复数的实部和虚部部分计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位； 将 VRX 和 VRY 的对应元素交织相乘后再乘 2 进行舍入操作，保留所得到乘积的所有精度后取负，与 VRZ 的对应元素相加，最后取饱和，结果写回到 VRZ 的对应元素中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Round= 1<<(size-1) for(i=0; i< Number/2; i++) { Tmp(2i+1)=Vz(2i+1)<<size-Vx(2i)*Vy(2i+1)*2+round; Tmp(2i)=Vz(2i)<<size- Vx(2i)*Vy(2i)*2+round; } 注：Tmp(i) 保留全部精度 for(i=0; i< Number; i++) If(Tmp(i)>2^(2*size-1)-1) Vz(i)= 2^(size-1)-1; Else if(Tmp(i)<-2^(2*size-1)) Vz(i)= -2^(size-1); Else Vz(i)=High[Tmp(i)]; } 注：饱和在累加后进行
语法	vrmulxss.T.rs vz, vx, vy
数据类型	S8/S16/S32
说明	复数的实部和虚部部分计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；将 VRX 和 VRY 的对应元素交织相乘后再乘 2 进行舍入操作，保留所得到乘积的所有精度后取负，与 VRZ 的对应元素相加，最后取饱和，结果写回到 VRZ 的对应元素中 **
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op		26	25	24	21	20	19	16	15	5			4	3	0	
111110			w1		VRY		w0		VRX		01111010110			S		VRZ	

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.136

VRMULXSSI.T.RS——

矢量带索引带饱和复数实部虚部交叉相乘累减累减取高半舍入指令

统一化指令	
语法	vrmulxssi.T.rs vz, vx, vy,index (T=S8/S16/S32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Index 范围为 [0,Number/2-1] Round= 1<<(size-1) for(i=0; i< Number/2; i++) { Tmp(2i+1)=Vz(2i+1)<<size-Vx(2i)*Vy(2index+1)*2+round; Tmp(2i)=Vz(2i)<<size- Vx(2i)*Vy(2index)*2+round; } 注： Tmp(i) 保留全部精度 for(i=0; i< Number; i++) If(Tmp(i)>2^(2*size-1)-1) Vz(i)= 2^(size-1)-1; Else if(Tmp(i)<-2^(2*size-1)) Vz(i)= -2^(size-1); Else Vz(i)=High[Tmp(i)]; } 注：饱和在累加减后进行
编译结果	仅存在 32 位指令。 vrmulxssi.T.rs vz, vx, vy,index
数据类型	S8/S16/S32
说明	复数的实部和虚部部分计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；将 VRX 和 index 索引 VRY 的对应元素交织相乘后再乘 2 进行舍入操作，保留所得到乘积的所有精度后取负，与 VRZ 的对应元素相加，最后取饱和，结果写回到 VRZ 的对应元素中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] Index 范围为 [0,Number/2-1] Round= 1<<(size-1) for(i=0; i< Number/2; i++) { Tmp(2i+1)=Vz(2i+1)<<size-Vx(2i)*Vy(2index+1)*2+round; Tmp(2i)=Vz(2i)<<size- Vx(2i)*Vy(2index)*2+round; } 注：Tmp(i) 保留全部精度 for(i=0; i< Number; i++) If(Tmp(i)>2^(2*size-1)-1) Vz(i)= 2^(size-1)-1; Else if(Tmp(i)<-2^(2*size-1)) Vz(i)= -2^(size-1); Else Vz(i)=High[Tmp(i)]; } 注：饱和在累加减后进行
语法	vrmulxssi.T.rs vz, vx, vy,index
数据类型	S8/S16/S32
说明	复数的实部和虚部部分计算，一个矢量操作数包含两个复数，实部在低位，虚部在高位；将 VRX 和 index 索引 VRY 的对应元素交织相乘后再乘 2 进行舍入操作，保留所得到乘积的所有精度后取负，与 VRZ 的对应元素相加，最后取饱和，结果写回到 VRZ 的对应元素中 **
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	8	7	6	5	4	3	0
111110			I1		VRY		I0	VRX		01011000			IMM2	0	S	VRZ

注：

编码表中 I1, I0 与 Imm2 组合在一起用来表征指令位宽类型和 Index，其中：

IMM[3:0] = {I1,I0,IMM2};

if(IMM[3:2]=2’ b01) size=16, IMM[1:0] 表示 index0 到 3;

if(IMM[3:2]=2’ b10) size=32, IMM[1:0] 表示 index0 到 1(高位为 0);

if(IMM[3:2]=2' b00) size=8, IMM[1:0] 表示 index0 到 3;

if(IMM[3:2]=2' b11) size=8, IMM[1:0] 表示 index4 到 7;

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX, VRY, VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.137 VRSQRTE.T——矢量倒数开方指令

统一化指令	
语法	vrsqrte.T vz, vx (T=S8/S16/S32/U8/U16/U32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] T 表征指令类型 T={u8,u16,u32,s8,s16,s32} 倒数开方快速算法公式： Vz(i) = 1.88 - Vx(i) for(i=0; i< Number; i++) { If(Vx(i)<0.5){ Vx(i)=0.5 ; } Elseif(Vx(i)>1){ Vx(i)=1; } Else{ Vx(i)=Vx(i); } } for(i=0; i< Number; i++) { If(T=s8) Vz(i) = 0x3C- Vx(i) (操作数、结果均为有符号数) If(T=s16) Vz(i) = 0x3C28 -Vx(i) (操作数、结果均为有符号数) If(T=s32) Vz(i) = 0x3C28F5C2 -Vx(i) (操作数、结果均为有符号数) If(T=u8) Vz(i) = 0x78 -Vx(i) (操作数、结果均为无符号数) If(T=u16) Vz(i) = 0x7851 -Vx(i) (操作数、结果均为无符号数) If(T=u32)Vz(i) = 0x7851EB85 -Vx(i) (操作数、结果均为无符号数) }
编译结果	仅存在 32 位指令。 vrsqrte.T vz, vx
数据类型	U8/U16/U32 S8/S16/S32
说明	Vx(i) 值取值范围是 [0.5, 1], 若超过该范围，则在运算前强制将 Vx(i) 置为边界值，根据倒数开方快速算法公式：Vz(i) = 1.88 - Vx(i)，将 magic 值转化为相应有/无符号，size 大小的定点数，在判断 VRX 中元素边界值后，将 VRX 中的元素代入公式计算出的结果存入 VRZ
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无 823 © 【杭州中天微系统有限公司】版权所有

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] T 表征指令类型 T={u8,u16,u32,s8,s16,s32} 倒数开方快速算法公式： $V_z(i) = 1.88 - V_x(i)$ for(i=0; i< Number; i++) { If($V_x(i)<0.5$){ $V_x(i)=0.5$; } Elseif($V_x(i)>1$){ $V_x(i)=1$; } Else{ $V_x(i)=V_x(i)$; } } for(i=0; i< Number; i++) { If(T=s8) $V_z(i) = 0x3C - V_x(i)$ (操作数、结果均为有符号数) If(T=s16) $V_z(i) = 0x3C28 - V_x(i)$ (操作数、结果均为有符号数) If(T=s32) $V_z(i) = 0x3C28F5C2 - V_x(i)$ (操作数、结果均为有符号数) If(T=u8) $V_z(i) = 0x78 - V_x(i)$ (操作数、结果均为无符号数) If(T=u16) $V_z(i) = 0x7851 - V_x(i)$ (操作数、结果均为无符号数) If(T=u32) $V_z(i) = 0x7851EB85 - V_x(i)$ (操作数、结果均为无符号数) }
语法:	vrsqrte.T vz, vx
数据类型	U8/U16/U32 S8/S16/S32
说明	Vx(i) 值取值范围是 [0.5, 1], 若超过该范围，则在运算前强制将 Vx(i) 置为边界值，根据倒数开方快速算法公式： $V_z(i) = 1.88 - V_x(i)$ ，将 magic 值转化为相应有无符号，size 大小的定点数，在判断 VRX 中元素边界值后，将 VRX 中的元素代入公式计算出的结果存入 VRZ
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110	w1	X	w0	VRX	00011101010	S	VRZ						

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.138 VRSQRTS.T——矢量倒数开方逼近指令

统一化指令	
语法	vrsqrts.T vrz, vrx, vry (T=S8/S16/S32 /U8/U16/U32)
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] T 表征指令类型 T={u8,u16,u32,s8,s16,s32} 倒数快速算法公式：$V_z(i) = 1.5 + ((-V_x(i)*V_y(i))/2)$ for(i=0; i< Number; i++) { If(T=s8) $V_z(i) = 0x30 + ((-V_x(i)*V_y(i))>>6\text{-bit})$ (操作数、结果均为有符号数) If(T=s16) $V_z(i) = 0x3000 + ((-V_x(i)*V_y(i))>>14\text{-bit})$ (操作数、结果均为有符号数) If(T=s32) $V_z(i)=0x30000000+((-V_x(i)*V_y(i))>>30\text{-bit})$ (操作数、结果均为有符号数) If(T=u8) $V_z(i) = 0x60 + ((-V_x(i)*V_y(i))>>7\text{-bit})$ (操作数从 8-bit 无符号数扩位为 9-bit 有符号数，结果取低 8 位) If(T=u16) $V_z(i) = 0x6000 + ((-V_x(i)*V_y(i))>>15\text{-bit})$ (操作数从 16-bit 无符号数扩位为 17-bit 有符号数，结果取低 16 位) If(T=u32) $V_z(i)=0x60000000+((-V_x(i)*V_y(i))>>31\text{-bit})$ (操作数从 32-bit 无符号数扩位为 33-bit 有符号数，结果取低 32 位) }
编译结果	仅存在 32 位指令。 vrsqrts.T vrz, vrx, vy
数据类型	U8/U16/U32 S8/S16/S32
说明	根据倒数逼近快速算法公式：$V_z(i) = 1.5 + ((-V_x(i)*V_y(i))/2)$ 将 1.5 转化为相应有无符号，size 大小的定点数，将 VRX 和 VRY 中的元素代入公式计算出的结果存入 VRZ
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Number=128/size i 表征第几个元素，范围为 [0,Number-1] T 表征指令类型 T={u8,u16,u32,s8,s16,s32} 倒数快速算法公式： $V_z(i) = 1.5 + ((-V_x(i)*V_y(i))/2)$ for(i=0; i< Number; i++) { If(T=s8) $V_z(i) = 0x30 + ((-V_x(i)*V_y(i))>>6\text{-bit})$ (操作数、结果均为有符号数) If(T=s16) $V_z(i) = 0x3000 + ((-V_x(i)*V_y(i))>>14\text{-bit})$ (操作数、结果均为有符号数) If(T=s32) $V_z(i)=0x30000000+((-V_x(i)*V_y(i))>>30\text{-bit})$ (操作数、结果均为有符号数) If(T=u8) $V_z(i) = 0x60 + ((-V_x(i)*V_y(i))>>7\text{-bit})$ (操作数从 8-bit 无符号数扩位为 9-bit 有符号数，结果取低 8 位) If(T=u16) $V_z(i) = 0x6000 + ((-V_x(i)*V_y(i))>>15\text{-bit})$ (操作数从 16-bit 无符号数扩位为 17-bit 有符号数，结果取低 16 位) If(T=u32) $V_z(i)=0x60000000+((-V_x(i)*V_y(i))>>31\text{-bit})$ (操作数从 32-bit 无符号数扩位为 33-bit 有符号数，结果取低 32 位) }
语法	vrsqrts.T vrz, vrx, vy
数据类型	U8/U16/U32 S8/S16/S32
说明	根据倒数逼近快速算法公式： $V_z(i) = 1.5 + ((-V_x(i)*V_y(i))/2)$ 将 1.5 转化为相应有无符号，size 大小的定点数，将 VRX 和 VRY 中的元素代入公式计算出的结果存入 VRZ
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

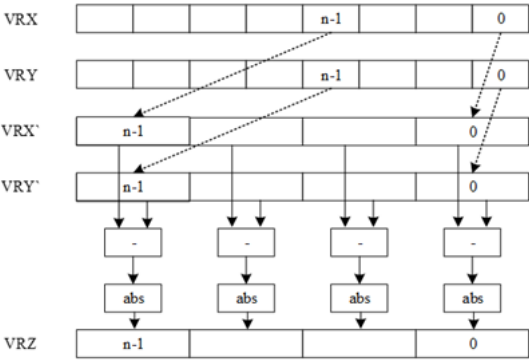
31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1	VRY		w0	VRX		00011101011			S	VRZ	

注：

- 编码表中 w1, w0 用来表征指令位宽类型，其中：
- [w1,w0] = 2’ b00 表示 size 为 8bit 指令；
- [w1,w0] = 2’ b01 表示 size 为 16bit 指令；
- [w1,w0] = 2’ b10 表示 size 为 32bit 指令；
- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 VRX,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.139 VSABS.T.E——矢量取绝对值指令

统一化指令	
语法	vsabs.T.e vrz, vrx (T=S8/S16/S32/U8/U16/U32)
操作	size = {8,16,32}; Number=128/size; for(i=0; i<Number; i++) { Vz(2i+1:2i)=abs(extend(Vx(i))-extend(Vy(i))); Vz+1(2i+1:2i)=abs(extend(Vx(Number/2+i))-extend(Vy(Number/2+i))); } extend 根据 U/S 将值零扩展或者符号扩展至元素位宽的 2 倍
编译结果	仅存在 32 位指令。 vsabs.T.e vrz, vrx
数据类型	U8/U16/U32 S8/S16/S32
说明	将 VRX 与 VRY 的对应元素扩展，相减后取绝对值，运算结果存放于 VRZ 中。 The diagram illustrates the operation of the VSABS.T.E instruction. It shows the flow of data from input registers VRX and VRY to intermediate registers VRX' and VRY', then through subtraction and absolute value operations to the final result register VRZ. VRX and VRY are registers of size n. VRX' and VRY' are registers of size 2n. VRZ is a register of size 2n. The operation is performed in four parallel lanes. In each lane, the corresponding elements of VRX and VRY are extended to VRX' and VRY' respectively. The elements of VRX' and VRY' are then subtracted (indicated by '-' boxes). The result of the subtraction is then passed through an absolute value operation (indicated by 'abs' boxes). The final results are stored in VRZ.
影响标志位	无影响
限制	寄存器的范围为 vr0-vr15
异常	无

32 位指令	
操作	size = {8,16,32}; Number=128/size; for(i=0; i<Number; i++) { Vz(2i+1:2i)=abs(extend(Vx(i))-extend(Vy(i))); Vz+1(2i+1:2i)=abs(extend(Vx(Number/2+i))-extend(Vy(Number/2+i))); } extend 根据 U/S 将值零扩展或者符号扩展至元素位宽的 2 倍
语法	vsabs.T.e vrz, vrx
数据类型	U8/U16/U32 S8/S16/S32
说明	将 VRX 与 VRY 的对应元素扩展，相减后取绝对值，运算结果存放于 VRZ 中。 
影响标志位	无影响
限制	寄存器的范围为 vr0-vr15
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0	
111110			w1	VRY		w0	VRX		00000110010			S	VRZ	

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.140 VSABS.T.S——矢量减法取绝对值并累加指令

统一化指令	
语法	vsabs.T.s vrz, vr _x , vr _y (T=S8/S16/S32 /U8/U16/U32)
操作	size = {8,16,32}; Number=128/size; U: Max=2 ^{(size)-1} ; Min= - Max; S: Max=2 ^{(size-1)-1} , Min= - Max; for(i=0; i< Number; i++) { If (V _x (i)-V _y (i)) < Min (V _x (i)-V _y (i)) > Max V _z (i)= Max; Else V _z (i)=abs(V _x (i)-V _y (i)); }
编译结果	仅存在 32 位指令。 vsabs.T.s vrz, vr _x , vr _y
数据类型	U8/U16/U32 S8/S16/S32
说明	VR _X 与 VR _Y 的对应元素相减，然后对减法结果取绝对值。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32}; Number=128/size; U: Max=2^(size)-1; Min= - Max; S: Max=2^(size-1)-1, Min= - Max; for(i=0; i< Number; i++) { If (Vx(i)-Vy(i)) < Min (Vx(i)-Vy(i)) > Max Vz(i)= Max; Else Vz(i)=abs(Vx(i)-Vy(i)); }
语法	vsabs.T vrz, vrx, vry
数据类型	U8/U16/U32 S8/S16/S32
说明	VRX 与 VRY 的对应元素相减，然后对减法结果取绝对值。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110			w1	VRY		w0	VRX		00000101010			S	VRZ

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.141 VSABSA.T——矢量减法取绝对值并累加指令

统一化指令	
语法	vsabsa.T vrz, vrx, vry (T=S8/S16/S32 /U8/U16/U32)
操作	size = {8,16,32}; Number=128/size; for(i=0; i< Number; i++) { Vz(i)= Vz(i)+abs(Vx(i)-Vy(i)) ; }
编译结果	仅存在 32 位指令。 vsabsa.T vrz, vrx, vry
数据类型	U8/U16/U32 S8/S16/S32
说明	VRX 与 VRY 的对应元素相减，然后对减法结果取绝对值，绝对值结果与 VRZ 中的对应元素相加，最后结果存放于 VRZ 中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32}; Number=128/size; for(i=0; i< Number; i++) { Vz(i)= Vz(i)+abs(Vx(i)-Vy(i)) ; }
语法	vsabsa.T vrz, vrx, vry
数据类型	U8/U16/U32 S8/S16/S32
说明	VRX 与 VRY 的对应元素相减，然后对减法结果取绝对值，绝对值结果与 VRZ 中的对应元素相加，最后结果存放于 VRZ 中。
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110	w1			VRY	w0			VRX	00000010101	S			VRZ

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

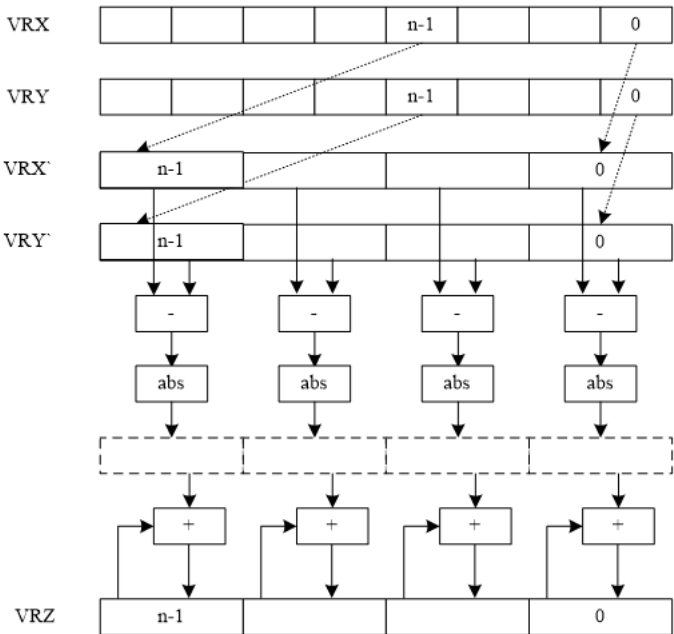
[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

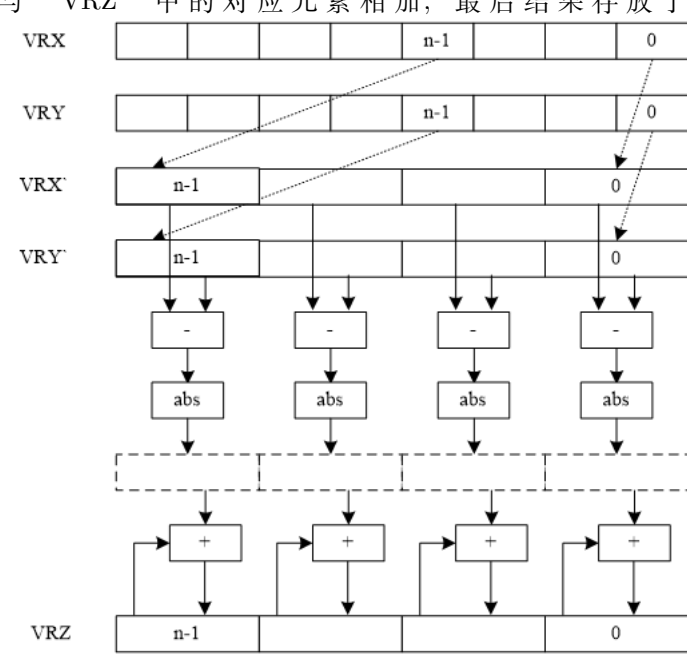
[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.142 VSABSA.T.E——矢量扩展减法取绝对值并累加指令

统一化指令	
语法	vsabsa.T.e vrz, vr_x, vr_y (T=S8/S16/S32/U8/U16/U32)
操作	size = {8,16,32}; Number=128/size; for(i=0; i< Number; i++) { V_z(2i+1:2i)=V_z(2i+1:2i) + abs(extend(V_x(i))-extend(V_y(i))); V_{z+1}(2i+1:2i)=V_{z+1}(2i+1:2i) + abs(extend(V_x(number/2+i))- extend(V_y(number/2+i))); } extend 根据 U/S 将值零扩展或者符号扩展至元素位宽的 2 倍
编译结果	仅存在 32 位指令。 vsabsa.T.e vrz, vr_x, vr_y
数据类型	U8/U16/U32 S8/S16/S32
说明	VR_X 与 VR_Y 的对应元素扩展并相减，然后对减法结果取绝对值，绝对值结果再与 VR_Z 中的对应元素相加，最后结果存放于 VR_Z 中。 
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32}; Number=128/size; for(i=0; i< Number; i++) { Vz(2i+1:2i)=Vz(2i+1:2i)+abs(extend(Vx(i))-extend(Vy(i))); Vz+1(2i+1:2i)=Vz+1(2i+1:2i)+abs(extend(Vx(number/2+i)) - extend(Vy(number/2+i))); } extend 根据 U/S 将值零扩展或者符号扩展至元素位宽的 2 倍
语法	vsabsa.T.e vrz, vrx, vry
数据类型	U8/U16/U32 S8/S16/S32
说明	VRX 与 VRY 的对应元素扩展并相减，然后对减法结果取绝对值，绝对值结果再与 VRZ 中的对应元素相加，最后结果存放于 VRZ 中。 
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1	VRY		w0	VRX		00000110011			S	VRZ	

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

$[w1, w0] = 2' b00$ 表示 size 为 8bit 指令；

$[w1, w0] = 2' b01$ 表示 size 为 16bit 指令；

$[w1, w0] = 2' b10$ 表示 size 为 32bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX, VRY, VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.143 VSAX.T.S——矢量错位减加指令

统一化指令	
语法	vsax.T.s vrz, vrx (T=S8/S16/S32 /U8/U16/U32)
操作	size = {8,16,32}; Number=128/size signed=(T==S); (根据元素 U/S 类型选择) Max=signed? 2^{^(size-1)}-1: 2^{^(size)}-1; Min=signed? -2^{^(size-1)}:0; for(i=0; i< Number; i++) { If (Vx(2i+1) -Vy(2i))>Max Vz(2i+1)=Max; Else if (Vx(2i+1) -Vy(2i))<Min Vz(2i+1)=Min; Else Vz(2i+1) = Vx(2i+1)-Vy(2i); If (Vx(2i)+Vy(2i+1))>Max Vz(2i)=Max; Else if (Vx(2i)+Vy(2i+1))<Min Vz(2i)=Min; Else Vz(2i)= Vx(2i)+Vy(2i+1); }
编译结果	仅存在 32 位指令。 vsax.T.s vrz, vrx
数据类型	U8/U16/U32 S8/S16/S32
说明	VRX 中的奇数元素与 VRZ 中的偶数元素相减，并对减法结果做饱和运算，运算结果放入 VRZ 的奇数元素中；VRX 中的偶数元素与 VRZ 中的奇数元素相加，并对加法结果做饱和运算，运算结果放入 VRZ 的偶数元素中。其中，当运算结果大于目的元素可表示的最大值时，取目的元素最大值作为饱和运算结果，当运算结果小于目的元素可表示的最小值时，取目的元素最小值作为饱和运算结果，当运算结果处于目的元素可表示范围时，取运算结果作为饱和运算值。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32}; Number=128/size signed=(T==S); (根据元素 U/S 类型选择) Max=signed? 2^(size-1)-1: 2^(size)-1; Min=signed? -2^(size-1):0; for(i=0; i< Number; i++) { If (Vx(2i+1) -Vy(2i))>Max Vz(2i+1)=Max; Else if (Vx(2i+1) -Vy(2i))<Min Vz(2i+1)=Min; Else Vz(2i+1) = Vx(2i+1)-Vy(2i); If (Vx(2i)+Vy(2i+1))>Max Vz(2i)=Max; Else if (Vx(2i)+Vy(2i+1))<Min Vz(2i)=Min; Else Vz(2i)= Vx(2i)+Vy(2i+1); }
语法	vsax.T.s vrz, vrx
数据类型	U8/U16/U32 S8/S16/S32
说明	VRX 中的奇数元素与 VRY 中的偶数元素相减，并对减法结果做饱和运算，运算结果放入 VRZ 的奇数元素中；VRX 中的偶数元素与 VRY 中的奇数元素相加，并对加法结果做饱和运算，运算结果放入 VRZ 的偶数元素中。其中，当运算结果大于目的元素可表示的最大值时，取目的元素最大值作为饱和运算结果，当运算结果小于目的元素可表示的最小值时，取目的元素最小值作为饱和运算结果，当运算结果处于目的元素可表示范围时，取运算结果作为饱和运算值。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1	VRY		w0	VRX		00000010001			S	VRZ	

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

$[w1,w0] = 2'b00$ 表示 size 为 8bit 指令；

$[w1,w0] = 2'b01$ 表示 size 为 16bit 指令；

$[w1,w0] = 2'b10$ 表示 size 为 32bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.144 VSAXH.T——矢量错位减加平均指令

统一化指令	
语法	vsaxh.T vrz, vrx (T=S8/S16/S32/U8/U16/U32)
操作	size = {8,16,32}; Number=128/size for(i=0; i< Number; i++) { Vz(2i+1)=(Vx(2i+1)-Vy(2i))>>1; Vz(2i)=(Vx(2i)+Vy(2i+1))>>1; } 对于 U, 右移为逻辑右移, 对于 S, 右移为算术右移
编译结果	仅存在 32 位指令。 vsaxh.T vrz, vrx
数据类型	U8/U16/U32 S8/S16/S32
说明	VRX 中的奇数元素与 VRY 中的偶数元素相减，然后对减法结果取平均值，运算结果放入 VRZ 的奇数元素中；VRX 中的偶数元素与 VRY 中的奇数元素相加，然后对加法结果取平均值，运算结果放入 VRZ 的偶数元素中。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32}; Number=128/size for(i=0; i< Number; i++) { Vz(2i+1)=(Vx(2i+1)-Vy(2i))>>1; Vz(2i)=(Vx(2i)+Vy(2i+1))>>1; } 对于 U, 右移为逻辑右移, 对于 S, 右移为算术右移
语法	vsaxh.T vrz, vrx
数据类型	U8/U16/U32 S8/S16/S32
说明	VRX 中的奇数元素与 VRY 中的偶数元素相减, 然后对减法结果取平均值, 运算结果放入 VRZ 的奇数元素中; VRX 中的偶数元素与 VRY 中的奇数元素相加, 然后对加法结果取平均值, 运算结果放入 VRZ 的偶数元素中。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式:

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1		VRY		w0		VRX		00000010011		S	VRZ

注:

编码表中 w1, w0 用来表征指令位宽类型, 其中:

[w1,w0] = 2' b00 表示 size 为 8bit 指令;

[w1,w0] = 2' b01 表示 size 为 16bit 指令;

[w1,w0] = 2' b10 表示 size 为 32bit 指令;

编码表中 S 表征带符号运算还是无符号运算: 1 表示带符号, 0 表示无符号;

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.145 VSEL.T——矢量位选择指令

统一化指令	
语法	vsel.T vrz,vrx,vry,vrk (T=8/16/32/64)
操作	for j=0:127 Vz[j]=Vk[j] ?Vx[j]:Vy[j]; vrx[j],vry[j],vrz[j] 指操作数比特位 j 的数据;
编译结果	仅存在 32 位指令。 vsel.T vrz, vrx,vry,vrk
数据类型	8/16/32/64
说明	根据 vrk 相应比特位为 1 或 0 选择 vrx 和 vry 相应的位，为 1 选择 vrx 的相应位，否则选择 vry 的相应位，放入 vrz 相应地比特位。 注：T 在此并无实际意义，仅为用户使用方便。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	for j=0:127 Vz[j]=Vk[j] ?Vx[j]:Vy[j]; vrx[j],vry[j],vrz[j] 指操作数比特位 j 的数据;
语法	vsel.T vrz, vrx,vry,vrk
数据类型	8/16/32/64
说明	根据 vrk 相应比特位为 1 或 0 选择 vrx 和 vry 相应的位，为 1 选择 vrx 的相应位，否则选择 vry 的相应位，放入 vrz 相应地比特位。 注：T 在此并无实际意义，仅为用户使用方便。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15		6	5	4	3	0	
111110				K3	VRY		K2	VRX		0001100100				K1	K0	VRZ

注：

k4,k3,k2,k1 拼起来作为 rk；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.146 VSEXT.T——矢量数据符号位读传送指令

统一化指令	
语法	vsext.T rz, vr_x (T=8/16/32)
操作	size = {8,16,32} If size=8 for i=0:15,Rz[i]=Vrx(i)[7]; end Rz[31:16]=0; If size=16 for i=0:7,Rz[i]=Vrx(i)[15]; end Rz[31:8]=0; If size=32 for i=0:3, Rz[i]=Vrx(i)[31]; end Rz[31:4]=0; (提取 Vrx 每个元素的符号位，将其依次放入通用寄存器 Rz 的低位)
编译结果	仅存在 32 位指令。 vsext.T rz, vr_x
数据类型	U8/U16/U32
说明	将源矢量寄存器 VRX 的每个元素的符号位依次传送到主处理器的目的寄存器 RZ 中，RZ 从低位开始存放，高位置 0。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	<pre>size = {8,16,32}; number = 128/ size; If (size==8){ for (i=0;i< number;i=i+1){ Rz[i]=Vrx(i)[7]; } Rz[31:16]=0; } else if (size==16){ for (i=0;i<number;i=i+1){ Rz[i]=Vrx(i)[15]; } Rz[31:8]=0; }else if (size==32){ for (i=0;i<=number;i=i+1){ Rz[i]=Vrx(i)[31]; } Rz[31:4]=0; }</pre>
语法	vsext.T rz, vrx
数据类型	U8/U16/U32
说明	将源矢量寄存器 VRX 的每个元素的符号位依次传送到主处理器的目的寄存器 RZ 中, RZ 从低位开始存放, 高位置 0。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	6	5	4	0		
111110				w1	X		w0	VRX		0001110011				0	RZ

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

编码表中 VRX,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.147 VSHL.T——矢量寄存器左移指令

统一化指令	
语法	vshl.T vrz,vrx,vry (T=S8/S16/S32/S64/U8/U16/U32/U64)
操作	size = {8,16,32,64}; number = 128/size; Vz(i)=Vx(i)<<Vy(i)[7:0] ; i=0:(number-1); 以 Vy 每个元素 Vy(i) 中的低 8-bit 数据 Vy(i)[7:0] 作为无符号移位索引;
编译结果	仅存在 32 位指令。 vshl.T vrz, vrx,vry
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	vrx 中 element 数据根据 vry 中 element 数据进行移位。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32,64}; number = 128/size; Vz(i)=Vx(i)<<Vy(i)[7:0] ; i=0:(number-1); 以 Vy 每个元素 Vy(i) 中的低 8-bit 数据 Vy(i)[7:0] 作为无符号移位索引;
语法	vshl.T vrz, vrx,vry
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	vrx 中 element 数据根据 vry 中 element 数据进行移位。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1	VRY		w0	VRX		00010100000			S	VRZ	

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

$[w1,w0] = 2' b00$ 表示 size 为 8bit 指令；

$[w1,w0] = 2' b01$ 表示 size 为 16bit 指令；

$[w1,w0] = 2' b10$ 表示 size 为 32bit 指令；

$[w1,w0] = 2' b11$ 表示 size 为 64bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.148 VSHL.T.S——矢量寄存器左移取饱和指令

统一化指令	
语法	vshl.T.s vrz,vrx,vry (T=S8/S16/S32/S64/U8/U16/U32/U64)
操作	size = {8,16,32,64}; number = 128/size; signed=(T==S); (根据元素 U/S 类型选择) Max=signed?2^(size-1)-1:2^(size)-1; Min=signed? -2^(size-1):0; If (Vx(i)<<Vy(i) [7:0])>Max Vz(i)=Max; Else if (Vx(i)<<Vy(i)[7:0])<Min Vz(i)=Min; Else Vz(i)= Vx(i)<<Vy(i)[7:0]; i=0:(number-1) 以 Vy 每个元素 Vy(i) 中的低 8-bit 数据 Vy(i)[7:0] 作为无符号移位索引;
编译结果	仅存在 32 位指令。 vshl.T.s vrz, vrz,vry
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	vrz 中 element 数据根据 vry 中 element 数据进行移位，并在移位后对结果进行饱和处理。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32,64}; number = 128/size; signed=(T==S); (根据元素 U/S 类型选择) Max=signed?2^(size-1)-1:2^(size)-1; Min=signed? -2^(size-1):0; If (Vx(i)<<Vy(i) [7:0])>Max Vz(i)=Max; Else if (Vx(i)<<Vy(i)[7:0])<Min Vz(i)=Min; Else Vz(i)= Vx(i)<<Vy(i)[7:0]; i=0:(number-1) 以 Vy 每个元素 Vy(i) 中的低 8-bit 数据 Vy(i)[7:0] 作为无符号移位索引;
语法	vshl.T.s vrz, vrx,vry
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	vrx 中 element 数据根据 vry 中 element 数据进行移位，并对移位结果进行饱和判断，之后进行饱和处理。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1	VRY		w0	VRX		00010100001		S	VRZ		

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

[w1,w0] = 2’ b11 表示 size 为 64bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.149 VSHLI.T——矢量立即数左移指令

统一化指令	
语法	vshli.T vrz,vrx,imm (T=S8/S16/S32/S64/U8/U16/U32/U64)
操作	size = {8,16,32,64}; number = 128/size; Vz(i)=Vx(i)<<imm; i=0:(number-1) imm 的范围是 0 ~ size-1
编译结果	仅存在 32 位指令。 vshli.T vrz, vrx,imm
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	vrx 中 element 数据根据立即数进行移位。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32,64}; number = 128/size; Vz(i)=Vx(i)<<imm; i=0:(number-1) imm 的范围是 0 ~ size-1
语法	vshli.T vrz, vrx,imm
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	vrx 中 element 数据根据立即数进行移位。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	20	19	16	15	6	5	4	3	0
111110		IMM6			VRX		0001000000			I	S	VRZ

注：

- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 VRX,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；
- 编码表中 Imm6,I 组合在一起用来表征指令位宽类型和 Index，其中：

IMM[6:0]={IMM6,I}; if(IMM[6:4]=010) size=8

IMM[3:0] 表示移位范围 0 到 7(高位为 0); if(IMM[6:4]=011) size=16

IMM[3:0] 表示移位范围 0 到 15; if(IMM[6:4]=001) size=32

IMM[3:0] 表示移位范围 0 到 15; if(IMM[6:4]=100) size=32

IMM[3:0] 表示移位范围 16 到 31; if(IMM[6:4]=000) size=64

IMM[3:0] 表示移位范围 0 到 15; if(IMM[6:4]=111) size=64

IMM[3:0] 表示移位范围 16 到 31; if(IMM[6:4]=110) size=64

IMM[3:0] 表示移位范围 32 到 47; if(IMM[6:4]=101) size=64

IMM[3:0] 表示移位范围 48 到 63;

16.150 VSHLI.T.E——矢量扩展立即数左移指令

统一化指令	
语法	vshli.T.e vrz,vrx,imm (T=S8/S16/S32 /U8/U16/U32)
操作	size = {8,16,32}; number = 128/size; Vz(2i+1,2i)=extend(Vx(i))<<imm; i=0:(number/2-1) Vz+1(2i+1,2i)=extend(Vx(number/2+i))<<imm; i=0:(number/2-1) extend 将元素根据 U/S 扩展为原始位宽的 2 倍之后进行立即数左移 imm 的范围是 0 ~ size*2-1
编译结果	仅存在 32 位指令。 vshli.T.e vrz, vrx,imm
数据类型	U8/U16/U32 S8/S16/S32
说明	vrx 中 element 数据扩展为原来位宽的 2 倍，然后根据立即数进行移位。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32}; number = 128/size; signed=(T==S); (根据元素 U/S 类型选择) Max=signed?2^(size-1)-1: 2^(size)-1; Min=signed? -2^(size-1):0; If (Vx(i)<<imm)>Max Vz(i)=Max; Else if (Vx(i)<<imm)<Min Vz(i)=Min; Else Vz(i)=Vx(i)<<imm; i=0:(number-1) imm 的范围是 0 ~ size-1
语法	vshli.T.s vrz, vrx,vry
数据类型	U8/U16/U32 S8/S16/S32
说明	vrx 中 element 数据扩展为原来位宽的 2 倍，然后根据立即数进行移位。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	20	19	16	15	6	5	4	3	0
111110		IMM6		VRX		0001000010		I	S	VRZ		

注：

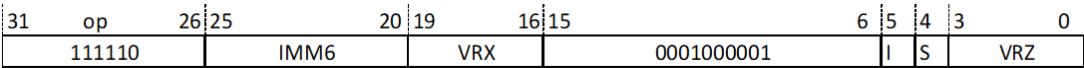
- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；
- 编码表中 Imm6,I 组合在一起用来表征指令位宽类型和 Index，其中：
- IMM[6:0]={IMM6,I}; if(IMM[6:4]=011) size=8 IMM[3:0] 表示移位范围 0 到 15；
- if(IMM[6:4]=001) size=16 IMM[3:0] 表示移位范围 0 到 15；
- if(IMM[6:4]=100) size=16 IMM[3:0] 表示移位范围 16 到 31；
- if(IMM[6:4]=000) size=32 IMM[3:0] 表示移位范围 0 到 15；
- if(IMM[6:4]=111) size=32 IMM[3:0] 表示移位范围 16 到 31；
- if(IMM[6:4]=110) size=32IMM[3:0] 表示移位范围 32 到 47；
- if(IMM[6:4]=101) size=32 IMM[3:0] 表示移位范围 48 到 63；

16.151 VSHLI.T.S——矢量立即数左移取饱和指令

统一化指令	
语法	vshli.T.s vrz,vrx,imm (T=S8/S16/S32/S64/U8/U16/U32/U64)
操作	size = {8,16,32,64}; number = 128/size; signed=(T==S); (根据元素 U/S 类型选择) Max=signed?2 ^{^(size-1)} -1: 2 ^{^(size)} -1; Min=signed? -2 ^{^(size-1)} :0; If (Vx(i)<<imm)>Max Vz(i)=Max; Else if (Vx(i)<<imm)<Min Vz(i)=Min; Else Vz(i)=Vx(i)<<imm; i=0:(number-1) imm 的范围是 0 ~ size-1
编译结果	仅存在 32 位指令。 vshli.T.s vrz, vrz,imm
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	vrz 中 element 数据根据立即数进行移位并在移位后进行饱和处理。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32,64}; number = 128/size; signed=(T==S); (根据元素 U/S 类型选择) Max=signed?2 ^{^(size-1)} -1: 2 ^{^(size)} -1; Min=signed? -2 ^{^(size-1)} :0; If (Vx(i)<<imm)>Max Vz(i)=Max; Else if (Vx(i)<<imm)<Min Vz(i)=Min; Else Vz(i)=Vx(i)<<imm; i=0:(number-1) imm 的范围是 0 ~ size-1
语法	vshli.T.s vrz, vrz,imm
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	vrz 中 element 数据根据立即数进行移位，并在移位后进行饱和处理。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：



注：

- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 VRX,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；
- 编码表中 Imm6,I 组合在一起用来表征指令位宽类型和 Index，其中：
- IMM[6:0]={IMM6,I}; if(IMM[6:4]=010) size=8
- IMM[3:0] 表示移位范围 0 到 7(高位为 0); if(IMM[6:4]=011) size=16
- IMM[3:0] 表示移位范围 0 到 15; if(IMM[6:4]=001) size=32
- IMM[3:0] 表示移位范围 0 到 15; if(IMM[6:4]=100) size=32
- IMM[3:0] 表示移位范围 16 到 31; if(IMM[6:4]=000) size=64
- IMM[3:0] 表示移位范围 0 到 15; if(IMM[6:4]=111) size=64
- IMM[3:0] 表示移位范围 16 到 31; if(IMM[6:4]=110) size=64
- IMM[3:0] 表示移位范围 32 到 47; if(IMM[6:4]=101) size=64
- IMM[3:0] 表示移位范围 48 到 63;

16.152 VSHR.T——矢量寄存器右移指令

统一化指令	
语法	vshr.T vrz,vrx,vry (T=S8/S16/S32/S64/U8/U16/U32/U64)
操作	size = {8,16,32,64}; number = 128/size; Vz(i)=Vx(i)>>Vy(i)[7:0] ; i=0:(number-1) 以 Vy 每个元素 Vy(i) 中的低 8-bit 数据 Vy(i)[7:0] 作为无符号移位索引; 对于 U, 右移为逻辑右移, 对于 S, 右移为算术右移
编译结果	仅存在 32 位指令。 vshr.T vrz, vrx,vry
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	vrx 中 element 数据根据 vry 中 element 数据进行右移。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32,64}; number = 128/size; Vz(i)=Vx(i)>>Vy(i)[7:0] ; i=0:(number-1) 以 Vy 每个元素 Vy(i) 中的低 8-bit 数据 Vy(i)[7:0] 作为无符号移位索引; 对于 U, 右移为逻辑右移, 对于 S, 右移为算术右移
语法	vshr.T vrz, vrx,vry
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	vrx 中 element 数据根据 vry 中 element 数据进行右移。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op		26	25	24	21	20	19	16	15				5	4	3	0
111110			w1		VRY		w0		VRX		00010101000			S		VRZ	

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

[w1,w0] = 2’ b11 表示 size 为 64bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.153 VSHR.T.R——矢量寄存器右移取 round 指令

统一化指令	
语法	vshr.T.r vrz,vrx,vry (T=S8/S16/S32/S64/U8/U16/U32/U64)
操作	size = {8,16,32,64}; number = 128/size; If Vy(i)[7:0]==0, round =0; else round=1<<(Vy(i)[7:0]-1); Vz(i)=(Vx(i)+round)>>Vy(i)[7:0]; i=0:(number-1) 以 Vy 每个元素 Vy(i) 中的低 8-bit 数据 Vy(i)[7:0] 作为无符号移位索引; 对于 U, 右移为逻辑右移, 对于 S, 右移为算术右移
编译结果	仅存在 32 位指令。 vshr.T.r vrz, vrz,vry
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	vrz 中 element 数据根据 vry 中 element 数据进行右移之后进行 round 操作。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32,64}; number = 128/size; If Vy(i)[7:0]==0, round =0; else round=1<<(Vy(i)[7:0]-1); Vz(i)=(Vx(i)+round)>>Vy(i)[7:0]; i=0:(number-1) 以 Vy 每个元素 Vy(i) 中的低 8-bit 数据 Vy(i)[7:0] 作为无符号移位索引; 对于 U, 右移为逻辑右移, 对于 S, 右移为算术右移
语法	vshr.T.r vrz, vrz,vry
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	vrz 中 element 数据根据 vry 中 element 数据进行右移, 之后进行 round 操作。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式:

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1	VRY		w0	VRX		00010101001			S	VRZ	

注：

- 编码表中 w1, w0 用来表征指令位宽类型，其中：
- [w1,w0] = 2' b00 表示 size 为 8bit 指令；
 - [w1,w0] = 2' b01 表示 size 为 16bit 指令；
 - [w1,w0] = 2' b10 表示 size 为 32bit 指令；
 - [w1,w0] = 2' b11 表示 size 为 64bit 指令；
- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 VRX, VRY, VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.154 VSHRI.T——矢量立即数右移指令

统一化指令	
语法	vshri.T vrz,vrx,oimm (T=S8/S16/S32/S64/U8/U16/U32/U64)
操作	size = {8,16,32,64}; number = 128/size; Vz(i)=Vx(i)>>oimm; i=0:(number-1) 对于 U, 右移为逻辑右移，对于 S, 右移为算术右移 oimm 的范围是 1 ~ size
编译结果	仅存在 32 位指令。 vshri.T vrz, vrz,oimm
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	vrz 中 element 数据根据立即数进行右移。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32,64}; number = 128/size; Vz(i)=Vx(i)>>oimm; i=0:(number-1) 对于 U, 右移为逻辑右移, 对于 S, 右移为算术右移 oimm 的范围是 1 ~ size
语法	vshri.T vrz, vrx,oimm
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	vrx 中 element 数据根据立即数进行右移。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	20	19	16	15	6	5	4	3	0
111110		IMM6			VRX		0001000100			I	S	VRZ

注：

编码表中 Imm6,I 组合在一起用来表征指令位宽类型和 Index，其中：

IMM[6:0]={IMM6,I}; if(IMM[6:4]=010) size=8

IMM[3:0] 表示移位范围 0 到 7(高位为 0); if(IMM[6:4]=011) size=16

IMM[3:0] 表示移位范围 0 到 15; if(IMM[6:4]=001) size=32

IMM[3:0] 表示移位范围 0 到 15; if(IMM[6:4]=100) size=32

IMM[3:0] 表示移位范围 16 到 31; if(IMM[6:4]=000) size=64

IMM[3:0] 表示移位范围 0 到 15; if(IMM[6:4]=111) size=64

IMM[3:0] 表示移位范围 16 到 31; if(IMM[6:4]=110) size=64

IMM[3:0] 表示移位范围 32 到 47; if(IMM[6:4]=101) size=64

IMM[3:0] 表示移位范围 48 到 63;

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.155 VSHRI.T.L——矢量立即数右移取低半指令

统一化指令	
语法	vshri.T.l vrz,vrx,oimm (T=S16/S32/S64 /U16/U32/U64)
操作	size = {16,32,64}; number = 128/size; Tmp(i)=(Vx(i)>>oimm)[size/2-1:0]; i=0:(number-1) (取结果的低半部分) Vz(i)={Tmp(2i+1),Tmp(2i)}; i=0:(number/2-1) (结果放于 Vz 的低 64bit) 对于 U, 右移为逻辑右移, 对于 S, 右移为算术右移 oimm 的范围是 1 ~ size
编译结果	仅存在 32 位指令。 vshri.T.l vrz, vrx,oimm
数据类型	U16/U32/U64 S16/S32/S64
说明	vrx 中 element 数据根据立即数进行右移并在移位后进行取结果低位操作。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {16,32,64}; number = 128/size; Tmp(i)=(Vx(i)>>oimm)[size/2-1:0]; i=0:(number-1) (取结果的低半部分) Vz(i)={Tmp(2i+1),Tmp(2i)}; i=0:(number/2-1) (结果放于 Vz 的低 64bit) 对于 U, 右移为逻辑右移, 对于 S, 右移为算术右移 oimm 的范围是 1 ~ size
语法	vshri.T.l vrz, vrx,oimm
数据类型	U16/U32/U64 S16/S32/S64
说明	vrx 中 element 数据根据立即数进行右移并在移位后进行取结果低位操作。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	20	19	16	15	6	5	4	3	0
111110		IMM6			VRX		0001000110			I	S	VRZ

注：

编码表中 Imm6,I 组合在一起用来表征指令位宽类型和 Index，其中：

IMM[6:0]={IMM6,I}; if(IMM[6:4]=010) size=8

IMM[3:0] 表示移位范围 0 到 7(高位为 0); if(IMM[6:4]=011) size=16

IMM[3:0] 表示移位范围 0 到 15; if(IMM[6:4]=001) size=32

IMM[3:0] 表示移位范围 0 到 15; if(IMM[6:4]=100) size=32

IMM[3:0] 表示移位范围 16 到 31; if(IMM[6:4]=000) size=64

IMM[3:0] 表示移位范围 0 到 15; if(IMM[6:4]=111) size=64

IMM[3:0] 表示移位范围 16 到 31; if(IMM[6:4]=110) size=64

IMM[3:0] 表示移位范围 32 到 47; if(IMM[6:4]=101) size=64

IMM[3:0] 表示移位范围 48 到 63;

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.156 VSHRI.T.LR——矢量立即数右移 Round 取低半指令

统一化指令	
语法	vshri.T.lr vrz,vrx,oimm (T=S16/S32/S64 /U16/U32/U64)
操作	size = {16,32,64}; number = 128/size; round=1<<(imm); Tmp(i)=((Vx(i)+round)>>(imm+1))[size/2-1:0]; i=0:(number-1) (取结果的低半部分) Vz(i)={Tmp(2i+1),Tmp(2i)}; i=0:(number/2-1) (结果放于 Vz 的低 64bit) 对于 U, 右移为逻辑右移, 对于 S, 右移为算术右移 oimm 的范围是 1 ~ size
编译结果	仅存在 32 位指令。 vshri.T.lr vrz, vrx,oimm
数据类型	U16/U32/U64 S16/S32/S64
说明	vrx 中 element 数据根据立即数进行右移, 之后进行 Round 操作并取结果低位。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {16,32,64}; number = 128/size; round=1<<(imm); Tmp(i)=((Vx(i)+round)>>(imm+1))[size/2-1:0]; i=0:(number-1) (取结果的低半部分) Vz(i)={Tmp(2i+1),Tmp(2i)}; i=0:(number/2-1) (结果放于 Vz 的低 64bit) 对于 U, 右移为逻辑右移, 对于 S, 右移为算术右移 oimm 的范围是 1 ~ size
语法	vshri.T.lr vrz, vrx,oimm
数据类型	U16/U32/U64 S16/S32/S64
说明	vrx 中 element 数据根据立即数进行右移之后进行 Round 操作并取结果低位。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	20	19	16	15	6	5	4	3	0
111110		IMM6		VRX		0001000111		I	S	VRZ		

注：

编码表中 Imm6,I 组合在一起用来表征指令位宽类型和 Index，其中：

IMM[6:0]={IMM6,I}; if(IMM[6:4]=010) size=8

IMM[3:0] 表示移位范围 0 到 7(高位为 0); if(IMM[6:4]=011) size=16

IMM[3:0] 表示移位范围 0 到 15; if(IMM[6:4]=001) size=32

IMM[3:0] 表示移位范围 0 到 15; if(IMM[6:4]=100) size=32

IMM[3:0] 表示移位范围 16 到 31; if(IMM[6:4]=000) size=64

IMM[3:0] 表示移位范围 0 到 15; if(IMM[6:4]=111) size=64

IMM[3:0] 表示移位范围 16 到 31; if(IMM[6:4]=110) size=64

IMM[3:0] 表示移位范围 32 到 47; if(IMM[6:4]=101) size=64

IMM[3:0] 表示移位范围 48 到 63;

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.157 VSHRI.T.LRS——矢量立即数右移 Round 取低半饱和指令

统一化指令	
语法	vshri.T.lrs vrz,vrx,oimm (T=S16/S32/S64 /U16/U32/U64)
操作	size = {16,32,64}; number = 128/size; round=1<<(oimm-1); signed=(T==S); (根据元素 U/S 类型选择) Max=signed? 2^(size/2-1)-1: 2^(size/2)-1; Min=signed? -2^(size/2-1): 0; If ((Vx(i)+round)>>oimm)>Max Tmp(i)=Max; Else if((Vx(i)+round)>>oimm)<Min Tmp(i)=Min; Else Tmp(i)=((Vx(i)+round)>>oimm)[size/2-1:0]; (取结果的低半部分) i=0:(number-1) Vz(i)={Tmp(2i+1),Tmp(2i)}; i=0:(number/2-1) ,(结果放于 Vz 的低 64bit) 对于 U, 右移为逻辑右移, 对于 S, 右移为算术右移 oimm 的范围是 1 ~ size
编译结果	仅存在 32 位指令。 vshri.T.lrs vrz, vrx,oimm
数据类型	U16/U32/U64 S16/S32/S64
说明	vrz 中 element 数据根据立即数进行右移, 之后进行 round 操作, 然后取结果低位并判断是否饱和进行饱和操作。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {16,32,64}; number = 128/size; round=1<<(oimm-1); signed=(T==S); (根据元素 U/S 类型选择) Max=signed? 2^(size/2-1)-1: 2^(size/2)-1; Min=signed? -2^(size/2-1): 0; If ((Vx(i)+round)>>oimm)>Max Tmp(i)=Max; Else if((Vx(i)+round)>>oimm)<Min Tmp(i)=Min; Else Tmp(i)=((Vx(i)+round)>>oimm)[size/2-1:0]; (取结果的低半部分) i=0:(number-1) Vz(i)={Tmp(2i+1),Tmp(2i)}; i=0:(number/2-1),(结果放于 Vz 的低 64bit) 对于 U, 右移为逻辑右移, 对于 S, 右移为算术右移 oimm 的范围是 1 ~ size
语法	vshri.T.lrs vrz, vrx,oimm
数据类型	U16/U32/U64 S16/S32/S64
说明	vrx 中 element 数据根据立即数进行右移, 之后进行 round 操作, 然后取结果低位并判断是否饱和并进行饱和操作。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式:

31	op	26	25	20	19	16	15	6	5	4	3	0
111110		IMM6			VRX		0001001001			I	S	VRZ

注:

编码表中 Imm6,I 组合在一起用来表征指令位宽类型和 Index, 其中:

IMM[6:0]={IMM6,I}; if(IMM[6:4]=010) size=8

IMM[3:0] 表示移位范围 0 到 7(高位为 0); if(IMM[6:4]=011) size=16

IMM[3:0] 表示移位范围 0 到 15; if(IMM[6:4]=001) size=32

IMM[3:0] 表示移位范围 0 到 15; if(IMM[6:4]=100) size=32

IMM[3:0] 表示移位范围 16 到 31; if(IMM[6:4]=000) size=64

IMM[3:0] 表示移位范围 0 到 15; if(IMM[6:4]=111) size=64

IMM[3:0] 表示移位范围 16 到 31; if(IMM[6:4]=110) size=64

IMM[3:0] 表示移位范围 32 到 47; if(IMM[6:4]=101) size=64

IMM[3:0] 表示移位范围 48 到 63;

编码表中 S 表征带符号运算还是无符号运算: 1 表示带符号, 0 表示无符号;

编码表中 VRX, VRY, VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.158 VSHRI.T.LS——矢量立即数右移取低半饱和指令

统一化指令	
语法	vshri.T.ls vrz,vrx,oimm (T=S16/S32/S64 /U16/U32/U64)
操作	size = {16,32,64}; number = 128/size; signed=(T==S); (根据元素 U/S 类型选择) Max=signed?2 ^{^(size/2-1)-1} :2 ^{^(size/2)-1} ; Min=signed? -2 ^{^(size/2-1)} : 0; If (Vx(i)>> oimm)>Max Tmp(i)=Max; Else if (Vx(i)>> oimm)<Min Tmp(i)=Min; Else Tmp(i)=(Vx(i)>>oimm)[size/2-1:0]; (取结果的低半部分) i=0:(number-1) Vz(i)={Tmp(2i+1),Tmp(2i)}; i=0:(number/2-1) (结果放于 Vz 的低 64bit) 对于 U, 右移为逻辑右移, 对于 S, 右移为算术右移 oimm 的范围是 1 ~ size
编译结果	仅存在 32 位指令。 vshri.T.ls vrz, vrz,oimm
数据类型	U16/U32/U64 S16/S32/S64
说明	vrz 中 element 数据根据立即数进行右移, 之后取结果低位并判断是否饱和并进行饱和操作。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {16,32,64}; number = 128/size; signed=(T==S); (根据元素 U/S 类型选择) Max=signed? 2 ^{^(size/2-1)} -1: 2 ^{^(size/2)} -1; Min=signed? -2 ^{^(size/2-1)} : 0; If (Vx(i)>> oimm)>Max Tmp(i)=Max; Else if (Vx(i)>> oimm)<Min Tmp(i)=Min; Else Tmp(i)=(Vx(i)>>oimm)[size/2-1:0]; (取结果的低半部分) i=i+(number-1) Vz(i)={Tmp(2i+1),Tmp(2i)}; i=i+(number/2-1) (结果放于 Vz 的低 64bit) 对于 U, 右移为逻辑右移, 对于 S, 右移为算术右移 oimm 的范围是 1 ~ size
语法	vshri.T.ls vrz, vrx,oimm
数据类型	U16/U32/U64 S16/S32/S64
说明	vrx 中 element 数据根据立即数进行右移, 之后取结果低位并判断是否饱和并进行饱和操作。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	20	19	16	15	6	5	4	3	0
111110		IMM6		VRX		0001001001		I	S	VRZ		

注：

编码表中 Imm6,I 组合在一起用来表征指令位宽类型和 Index，其中：

- IMM[6:0]={IMM6,I}; if(IMM[6:4]=010) size=8
- IMM[3:0] 表示移位范围 0 到 7(高位为 0); if(IMM[6:4]=011) size=16
- IMM[3:0] 表示移位范围 0 到 15; if(IMM[6:4]=001) size=32
- IMM[3:0] 表示移位范围 0 到 15; if(IMM[6:4]=100) size=32
- IMM[3:0] 表示移位范围 16 到 31; if(IMM[6:4]=000) size=64
- IMM[3:0] 表示移位范围 0 到 15; if(IMM[6:4]=111) size=64
- IMM[3:0] 表示移位范围 16 到 31; if(IMM[6:4]=110) size=64

IMM[3:0] 表示移位范围 32 到 47；if(IMM[6:4]=101) size=64

IMM[3:0] 表示移位范围 48 到 63；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.159 VSHRI.T.R——矢量立即数左移取 Round 指令

统一化指令	
语法	vshri.T.r vrz,vrx,oimm (T=S8/S16/S32/S64/U8/U16/U32/U64)
操作	size = {8,16,32,64}; round=1<<(oimm-1); Vz(i)=(Vx(i)+round)>>oimm); i=0:(number-1) 对于 U, 右移为逻辑右移，对于 S, 右移为算术右移 oimm 的范围是 1 ~ size
编译结果	仅存在 32 位指令。 vshri.T.r vrz, vrx,oimm
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	vrz 中 element 数据根据立即数进行右移并在移位后进行 Round 操作。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32,64}; number = 128/size; round=1<<(oimm-1); Vz(i)=(Vx(i)+round)>>oimm); i=0:(number-1) 对于 U, 右移为逻辑右移，对于 S, 右移为算术右移 oimm 的范围是 1 ~ size
语法	vshri.T.r vrz, vrx,oimm
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	vrz 中 element 数据根据立即数进行右移，并在移位后进行 Round 操作。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	20	19	16	15	6	5	4	3	0
111110		IMM6			VRX		0001000101			I	S	VRZ

注：

编码表中 Imm6,I 组合在一起用来表征指令位宽类型和 Index，其中：

- IMM[6:0]={IMM6,I}; if(IMM[6:4]=010) size=8
- IMM[3:0] 表示移位范围 0 到 7(高位为 0); if(IMM[6:4]=011) size=16
- IMM[3:0] 表示移位范围 0 到 15; if(IMM[6:4]=001) size=32
- IMM[3:0] 表示移位范围 0 到 15; if(IMM[6:4]=100) size=32
- IMM[3:0] 表示移位范围 16 到 31; if(IMM[6:4]=000) size=64
- IMM[3:0] 表示移位范围 0 到 15; if(IMM[6:4]=111) size=64
- IMM[3:0] 表示移位范围 16 到 31; if(IMM[6:4]=110) size=64
- IMM[3:0] 表示移位范围 32 到 47; if(IMM[6:4]=101) size=64
- IMM[3:0] 表示移位范围 48 到 63;

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.160 VSHRIA.T——矢量立即数右移累加指令

统一化指令	
语法	vshria.T vrz,vrx,oimm (T=S8/S16/S32/S64/U8/U16/U32/U64)
操作	size = {8,16,32,64}; number = 128/size; Vz(i)=Vz(i)+(Vx(i)>>oimm); i=0:(number-1) 对于 U, 右移为逻辑右移, 对于 S, 右移为算术右移 oimm 的范围是 1 ~ size
编译结果	仅存在 32 位指令。 vshria.T vrz, vrz,oimm
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	vrz 中 element 数据根据立即数进行右移, 之后和 VZ 相加得出结果。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32,64}; number = 128/size; Vz(i)=Vz(i)+(Vx(i)>>oimm); i=0:(number-1) 对于 U, 右移为逻辑右移, 对于 S, 右移为算术右移 oimm 的范围是 1 ~ size
语法	vshria.T vrz, vrx,oimm
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	vrx 中 element 数据根据立即数进行右移, 之后和 VZ 相加得出结果。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式:

31	op	26	25	20	19	16	15	6	5	4	3	0
111110		IMM6			VRX		0001001010			I	S	VRZ

注:

编码表中 Imm6,I 组合在一起用来表征指令位宽类型和 Index, 其中:

IMM[6:0]={IMM6,I}; if(IMM[6:4]=010) size=8

IMM[3:0] 表示移位范围 0 到 7(高位为 0); if(IMM[6:4]=011) size=16

IMM[3:0] 表示移位范围 0 到 15; if(IMM[6:4]=001) size=32

IMM[3:0] 表示移位范围 0 到 15; if(IMM[6:4]=100) size=32

IMM[3:0] 表示移位范围 16 到 31; if(IMM[6:4]=000) size=64

IMM[3:0] 表示移位范围 0 到 15; if(IMM[6:4]=111) size=64

IMM[3:0] 表示移位范围 16 到 31; if(IMM[6:4]=110) size=64

IMM[3:0] 表示移位范围 32 到 47; if(IMM[6:4]=101) size=64

IMM[3:0] 表示移位范围 48 到 63;

编码表中 S 表征带符号运算还是无符号运算: 1 表示带符号, 0 表示无符号;

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号;

16.161 VSHRIA.T.R——矢量立即数右移取 Round 累加指令

统一化指令	
语法	vshria.T.r vrz,vrx,oimm (T=S8/S16/S32/S64 /U8/U16/U32/U64)
操作	size = {8,16,32,64}; number = 128/size; round=1<<(oimm-1); Vz(i)=Vz(i)+((Vx(i)+round)>>oimm); i=0:(number-1) 对于 U, 右移为逻辑右移, 对于 S, 右移为算术右移 oimm 的范围是 1 ~ size
编译结果	仅存在 32 位指令。 vshria.T.r vrz, vrx,oimm
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	vrx 中 element 数据根据立即数进行右移, 取 Round 之后和 VZ 相加得出结果。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32,64}; number = 128/size; round=1<<(oimm-1); Vz(i)=Vz(i)+((Vx(i)+round)>>oimm); i=0:(number-1) 对于 U, 右移为逻辑右移, 对于 S, 右移为算术右移 oimm 的范围是 1 ~ size
语法	vshria.T.r vrz, vrx,oimm
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	vrx 中 element 数据根据立即数进行右移, 取 Round 之后和 VZ 相加得出结果。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	20	19	16	15	6	5	4	3	0
111110		IMM6			VRX		0001001011			I	S	VRZ

注：

编码表中 Imm6,I 组合在一起用来表征指令位宽类型和 Index，其中：

IMM[6:0]={IMM6,I}; if(IMM[6:4]=010) size=8

IMM[3:0] 表示移位范围 0 到 7(高位为 0); if(IMM[6:4]=011) size=16

IMM[3:0] 表示移位范围 0 到 15; if(IMM[6:4]=001) size=32

IMM[3:0] 表示移位范围 0 到 15; if(IMM[6:4]=100) size=32

IMM[3:0] 表示移位范围 16 到 31; if(IMM[6:4]=000) size=64

IMM[3:0] 表示移位范围 0 到 15; if(IMM[6:4]=111) size=64

IMM[3:0] 表示移位范围 16 到 31; if(IMM[6:4]=110) size=64

IMM[3:0] 表示移位范围 32 到 47; if(IMM[6:4]=101) size=64

IMM[3:0] 表示移位范围 48 到 63;

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

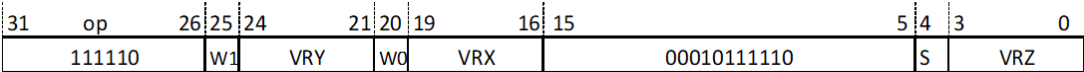
编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.162 VSHT.T——矢量移位指令

统一化指令	
语法	<code>vsht.T vrz,vrx,vry</code> (T=S8/S16/S32/S64/U8/U16/U32/U64)
操作	<code>size = {8,16,32,64};</code> <code>number = 128/size;</code> <code>if Vy(i)[7:0]>0,</code> <code>Vz(i)=Vx(i)<<Vy(i)[7:0];</code> <code>else</code> <code>Vz(i)=Vx(i)>> Vy(i)[7:0] ; i=0:(number-1)</code> 以 Vy 每个元素 Vy(i) 中的低 8-bit 数据 Vy(i)[7:0] 作为有符号移位索引; 对于 U, 右移为逻辑右移, 对于 S, 右移为算术右移;
编译结果	仅存在 32 位指令。 <code>vsht.T vrz, vrx,vry</code>
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	根据 vry 相应 element 的正负决定左移还是右移, 如果 vry 相应 element 为正则左移, 否则将 vry 相应的 element 取绝对值作为移位数进行右移。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	<code>size = {8,16,32,64};</code> <code>number = 128/size;</code> <code>if Vy(i)[7:0]>0, Vz(i)=Vx(i)<<Vy(i)[7:0];</code> <code>else Vz(i)=Vx(i)>> Vy(i)[7:0] ; i=0:(number-1)</code> 以 Vy 每个元素 Vy(i) 中的低 8-bit 数据 Vy(i)[7:0] 作为有符号移位索引; 对于 U, 右移为逻辑右移, 对于 S, 右移为算术右移;
语法	<code>vsht.T vrz, vrx,vry</code>
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	根据 vry 相应 element 的正负决定左移还是右移, 如果 vry 相应 element 为正则左移, 否则将 vry 相应的 element 取绝对值作为移位数进行右移。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式:



注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

[w1,w0] = 2’ b11 表示 size 为 64bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.163 VSHT.T.R——矢量移位 Round 指令

统一化指令	
语法	vsht.T.r vrz,vrx,vry (T=S8/S16/S32/S64/U8/U16/U32/U64)
操作	size = {8,16,32,64}; number = 128/size; If Vy(i)[7:0]==0, round=0; else round=1<<(-Vy(i) [7:0]-1); if Vy(i)[7:0]>0, Vz(i)=Vx(i)<<Vy(i)[7:0]; else Vz(i)=(Vx(i)+round)>> Vy(i)[7:0] ; i=0:(number-1) 以 Vy 每个元素 Vy(i) 中的低 8-bit 数据 Vy(i)[7:0] 作为有符号移位索引; 对于 U, 右移为逻辑右移, 对于 S, 右移为算术右移;
编译结果	仅存在 32 位指令。 vsht.T.r vrz, vrz,vry
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	根据 vry 相应 element 的正负决定左移还是右移，如果 vry 相应 element 为正则左移，否则将 vry 相应的 element 取绝对值作为移位数进行右移并对移位后的数据进行 round 操作。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32,64}; number = 128/size; If Vy(i)[7:0]==0, round=0; else round=1<<(-Vy(i) [7:0]-1); if Vy(i)[7:0]>0, Vz(i)=Vx(i)<<Vy(i)[7:0]; else Vz(i)=(Vx(i)+round)>> Vy(i)[7:0] ; i=0:(number-1) 以 Vy 每个元素 Vy(i) 中的低 8-bit 数据 Vy(i)[7:0] 作为有符号移位索引; 对于 U, 右移为逻辑右移, 对于 S, 右移为算术右移
语法	vsht.T.r vrz, vrz,vry
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	根据 vry 相应 element 的正负决定左移还是右移, 如果 vry 相应 e lement 为正则左移, 否则将 vry 相应的 element 取绝对值作为移位数进行右移, 并对移位后的数据进行 round 操作。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1	VRY		w0	VRX		00010111001		S	VRZ		

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

[w1,w0] = 2’ b11 表示 size 为 64bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.164 VSHT.T.RS——矢量移位饱和 Round 指令

统一化指令	
语法	<code>vsht.T.rs vrz,vrx,vry</code> (T=S8/S16/S32/S64/U8/U16/U32/U64)
操作	<code>size = {8,16,32,64};</code> <code>number = 128/size;</code> <code>If Vy(i)[7:0]==0, round=0;</code> <code>else round=1<<(-Vy(i) [7:0]-1); end</code> <code>if Vy(i)[7:0]>0</code> <code>Vz(i)=sat(Vx(i)<<Vy(i)[7:0]);</code> <code>else</code> <code>Vz(i)=(Vx(i)+round)>> Vy(i)[7:0] ;</code> <code>i=0:(number-1)</code> 以 Vy 每个元素 Vy(i) 中的低 8-bit 数据 Vy(i)[7:0] 作为有符号移位索引; sat 根据回写元素位宽判断左移结果是否溢出, 并根据 U/S 将溢出结果饱和为相应的最大或最小值, 对于 U, 右移为逻辑右移, 对于 S, 右移为算术右移
编译结果	仅存在 32 位指令。 <code>vsht.T.rs vrz, vrz,vry</code>
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	根据 vry 相应 element 的正负决定左移还是右移, 如果 vry 相应 element 为正则左移并判断左移后数据的饱和, 否则将 vry 相应的 element 取绝对值作为移位数进行右移并对移位后的数据进行 round 操作。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32,64}; number = 128/size; If Vy(i)[7:0]==0 round=0; else round=1<<(-Vy(i) [7:0]-1); if Vy(i)[7:0]>0, Vz(i)=sat(Vx(i)<< Vy(i)[7:0]); else Vz(i)=(Vx(i)+round)>>Vy(i)[7:0]; i=0:(number-1) 以 Vy 每个元素 Vy(i) 中的低 8-bit 数据 Vy(i)[7:0] 作为有符号移位索引; sat 根据回写元素位宽判断左移结果是否溢出, 并根据 U/S 将溢出结果饱和为相应的最大或最小值, 对于 U, 右移为逻辑右移, 对于 S, 右移为算术右移
语法	vsht.T.rs vrz, vrx,vry
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	根据 vry 相应 element 的正负决定左移还是右移, 如果 vry 相应 element 为正则左移并对移位后的数据进行饱和操作, 否则将 vry 相应的 element 取绝对值作为移位数进行右移, 并对移位后的数据进行 round 操作。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1	VRY		w0	VRX		00010111000			S	VRZ	

注：

编码表中 w1, w0 用来表征指令位宽类型, 其中:

[w1,w0] = 2' b00 表示 size 为 8bit 指令;

[w1,w0] = 2' b01 表示 size 为 16bit 指令;

[w1,w0] = 2' b10 表示 size 为 32bit 指令;

[w1,w0] = 2' b11 表示 size 为 64bit 指令;

编码表中 S 表征带符号运算还是无符号运算: 1 表示带符号, 0 表示无符号;

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.165 VSHT.T.S——矢量移位饱和指令

统一化指令	
语法	vsht.T.s vrz,vrx,vry (T=S8/S16/S32/S64/U8/U16/U32/U64)
操作	size = {8,16,32,64}; number = 128/size; if Vy(i)[7:0]>0, Vz(i)=sat(Vx(i)<<Vy(i)[7:0]); else Vz(i)=Vx(i)>> Vy(i)[7:0] ; i=0:(number-1) 以 Vy 每个元素 Vy(i) 中的低 8-bit 数据 Vy(i)[7:0] 作为有符号移位索引; sat 根据回写元素位宽判断左移结果是否溢出, 并根据 U/S 将溢出结果饱和为相应的最大或最小值; 对于 U, 右移为逻辑右移, 对于 S, 右移为算术右移
编译结果	仅存在 32 位指令。 vsht.T.s vrz, vrz,vry
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	根据 vry 相应 element 的正负决定左移还是右移, 如果 vry 相应 element 为正则左移并对数据左移后进行饱和处理, 否则将 vry 相应的 element 取绝对值作为移位数进行右移。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32,64}; number = 128/size; if Vy(i)[7:0]>0, Vz(i)=Sat(Vx(i)<<Vy(i)[7:0]); else Vz(i)=Vx(i)>> Vy(i)[7:0] ; i=0:(number-1) 以 Vy 每个元素 Vy(i) 中的低 8-bit 数据 Vy(i)[7:0] 作为有符号移位索引; sat 根据回写元素位宽判断左移结果是否溢出, 并根据 U/S 将溢出结果饱和为相应的最大或最小值; 对于 U, 右移为逻辑右移, 对于 S, 右移为算术右移
语法	vsht.T.s vrz, vrz,vry
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	根据 vry 相应 element 的正负决定左移还是右移, 如果 vry 相应 element 为正则左移并对移位后数据进行饱和处理, 否则将 vry 相应的 element 取绝对值作为移位数进行右移。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0			
111110			w1	VRY			w0	VRX			0001011111			S	VRZ	

注：

- 编码表中 w1, w0 用来表征指令位宽类型，其中：
- [w1,w0] = 2’ b00 表示 size 为 8bit 指令；
 - [w1,w0] = 2’ b01 表示 size 为 16bit 指令；
 - [w1,w0] = 2’ b10 表示 size 为 32bit 指令；
 - [w1,w0] = 2’ b11 表示 size 为 64bit 指令；
- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号；

16.166 VST.T.N——固定长度矢量存储指令

统一化指令	
语法	vst.T.N vrz,(rx, offset) (T=8/16/32 N=1/2/3/4)
操作	从矢量寄存器 VRZ 存储固定多个字节/半字/字到存储器中 size = {8,16,32} num = {1,2,3,4} offset= imm7<< log2(size/8) MEM[RX + zero_extend(offset)] ← VRZ[size*num-1:0];**
编译结果	仅存在 32 位指令。 vst.T.N vrz,(rx, imm7)
说明	从矢量寄存器 VRZ 存储多个字节/半字/字到存储器中，存储的个数由指令决定。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移的 7 位相对偏移量无符号扩展到 32 位后的值得到（如加载类型为字节，则不左移，如加载类型为半字，则左移 1 位，如加载类型为字，则左移 2 位）。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常、TLB 修改异常

32 位指令	
操作	从矢量寄存器 VRZ 存储固定多个字节/半字/字到存储器中 size = {8,16,32} num = {1,2,3,4} offset= imm7<< log2(size/8) MEM[RX + zero_extend(offset)] ← VRZ[size*num-1:0];
语法	vst.T.N vrz,(rx, offset)
说明	从矢量寄存器 VRZ 存储多个字节/半字/字到存储器中，存储的个数由指令决定。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移的 7 位相对偏移量无符号扩展到 32 位后的值得到（如加载类型为字节，则不左移，如加载类型为半字，则左移 1 位，如加载类型为字，则左移 2 位）。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常、TLB 修改异常

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	8	7	6	5	4	3	0
1		11110		IMM7H		RX		001110		IMM7L	NUM	SIZE				VRZ

16.167 VSTM.T——连续矢量存储指令

统一化指令	
语法	vstm.T vry-vrz, (rx) (T=8/16/32)
操作	从一片连续的矢量寄存器堆存储多个字节、半字、字到存储器空间中 dst ← Y; addr ← RX; for (n = 0; n <= (Z-Y); n++){ MEM[addr] ← VRdst[127:0]; dst ← dst + 1; addr ← addr + 16; }
编译结果	仅存在 32 位指令。 vstm.T vry-vrz, (rx);
说明	从 VRY 开始的一片连续矢量寄存器堆依次存储多个字节/半字/字到连续的存储器空间中，即将矢量寄存器 VRY 的 128 位宽的多个字节/半字/字存储到指定地址中，第二个寄存器 VR(Y+1) 的 128 位宽的多个字节/半字/字存储到存储器指定开始的第二个 128 位地址中，依次类推，最后一个寄存器 128 位宽的多个字节/半字/字存储到存储器指定开始的最后的 128 位地址中。存储器的有效地址由基址寄存器 RX 的内容决定。
影响标志位	无影响
限制	VRZ 应当大于等于 VRY。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常、TLB 修改异常

32 位指令	
操作	从一片连续的矢量寄存器堆存储多个字节、半字、字到存储器空间中 $dst \leftarrow Y; addr \leftarrow RX;$ for ($n = 0; n \leq (Z-Y); n++$){ $MEM[addr] \leftarrow VRdst[127:0];$ $dst \leftarrow dst + 1;$ $addr \leftarrow addr + 16;$ }
语法	vstm.T vry-vrz, (rx)
说明	从 VRY 开始的一片连续矢量寄存器堆依次存储多个字节/半字/字到连续的存储器空间中，即将矢量寄存器 VRY 的 128 位宽的多个字节/半字/字存储到指定地址中，第二个寄存器 VR(Y+1) 的 128 位宽的多个字节/半字/字存储到存储器指定开始的第二个 128 位地址中，依次类推，最后一个寄存器 128 位宽的多个字节/半字/字存储到存储器指定开始的最后的 128 位地址中。存储器的有效地址由基址寄存器 RX 的内容决定。
影响标志位	无影响
限制	VRZ 应当大于等于 VRY。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常、TLB 修改异常

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	8	7	6	5	4	3	0
1	11110	00000		RX		001010		11	IMM2	SIZE					VRZ	

- 注：
- IMM2 域——指定目标寄存器的个数， $IMM2 = Z - Y$ 。
- 00——1 个目的寄存器
- 01——2 个目的寄存器
- 10——3 个目的寄存器
- 11——4 个目的寄存器

16.168 VSTMru.T——跳跃矢量存储基址更新指令

统一化指令	
语法	vstmru.T vry-vrz, (rx) (T=8/16/32)
操作	从一片连续的矢量寄存器堆存储多个字节、半字、字到存储器空间中，并更新基址。 dst ← Y; addr ← RX; for (n = 0; n <= (Z-Y); n++){ MEM[addr] ← VRdst[127:0]; dst ← dst + 1; addr ← addr + RY; } RX←addr
编译结果	仅存在 32 位指令。 vstmru.T vry-vrz, (rx);
说明	从 VRY 开始的一片连续矢量寄存器堆依次存储多个字节/半字/ 字到连续的存储器空间中，即将矢量寄存器 VRY 的 128 位宽的多个字节/半字/字存储到指定地址中，第二个寄存器 VR (Y+1) 的 128 位宽的多个字节/半字/字存储到基址加上偏移寄存器 RY 后的指定地址中，依次类推，最后一个寄存器 128 位宽的多个字节/半字/字存储到基址加上多次偏移寄存器 RY 后的指定地址中。存储器的有效地址由基址寄存器 RX 与每次的偏移寄存器的 RY 的内容决定。最后，更新基址寄存器 RX 为最后所存储地址再次加偏移寄存器 RY 的结果。
影响标志位	无影响
限制	VRZ 应当大于等于 VRY。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常、TLB 修改异常。

32 位指令	
操作	从一片连续的矢量寄存器堆存储多个字节、半字、字到存储器空间中，并更新基址。 dst ← Y; addr ← RX; for (n = 0; n <= (Z-Y); n++){ MEM[addr] ← VRdst[127:0]; dst ← dst + 1; addr ← addr + RY; } RX←addr
语法	vstmru.T vry-vrz, (rx)
说明	从 VRY 开始的一片连续矢量寄存器堆依次存储多个字节/半字/ 字到连续的存储器空间中，即将矢量寄存器 VRY 的 128 位宽的多个字节/半字/字存储到指定地址中，第二个寄存器 VR (Y+1) 的 128 位宽的多个字节/半字/字存储到基址加上偏移寄存器 RY 后的指定地址中，依次类推，最后一个寄存器 128 位宽的多个字节/半字/字存储到基址加上多次偏移寄存器 RY 后的指定地址中。存储器的有效地址由基址寄存器 RX 与每次的偏移寄存器的 RY 的内容决定。最后，更新基址寄存器 RX 为最后所存储地址再次加偏移寄存器 RY 的结果。
影响标志位	无影响
限制	VRZ 应当大于等于 VRY。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常、TLB 修改异常。

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	8	7	6	5	4	3	0
1		11110		00000		RX		001011		10	IMM2	SIZE			VRZ	

注：

IMM2 域——指定目标寄存器的个数，IMM2 = Z - Y。

00——1 个目的寄存器

01——2 个目的寄存器

10——3 个目的寄存器

11——4 个目的寄存器

16.169 VSTMU.T——连续矢量存储基址更新指令

统一化指令	
语法	vstmu.T vry-vrz, (rx) (T=8/16/32)
操作	从一片连续的矢量寄存器堆存储多个字节、半字、字到存储器空间中，并更新基址 dst ← Y; addr ← RX; for (n = 0; n <= (Z-Y); n++){ MEM[addr] ← VRdst[127:0]; dst ← dst + 1; addr ← addr + 16; } RX←addr
编译结果	仅存在 32 位指令。 vstmu.T vry-vrz, (rx);
说明	从 VRY 开始的一片连续矢量寄存器堆依次存储多个字节/半字/字到连续的存储器空间中，即将矢量寄存器 VRY 的 128 位宽的多个字节/半字/字存储到指定地址中，第二个寄存器 VR(Y+1) 的 128 位宽的多个字节/半字/字存储到存储器指定开始的第二个 128 位地址中，依次类推，最后一个寄存器 128 位宽的多个字节/半字/字存储到存储器指定开始的最后的 128 位地址中。存储器的有效地址由基址寄存器 RX 的内容决定。最后，更新基址寄存器 RX 为所有所存储数据之后的地址。
影响标志位	无影响
限制	VRZ 应当大于等于 VRY。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常、TLB 修改异常。

32 位指令	
操作	从一片连续的矢量寄存器堆存储多个字节、半字、字到存储器空间中，并更新基址 $dst \leftarrow Y; addr \leftarrow RX;$ for ($n = 0; n \leq (Z-Y); n++$){ MEM[addr] $\leftarrow$ VRdst[127:0]; $dst \leftarrow dst + 1;$ $addr \leftarrow addr + 16;$ } $RX \leftarrow addr$
语法	vstmu.T vry-vrz, (rx)
说明	从 VR _Y 开始的一片连续矢量寄存器堆依次存储多个字节/半字/字到连续的存储器空间中，即将矢量寄存器 VR _Y 的 128 位宽的多个字节/半字/字存储到指定地址中，第二个寄存器 VR _(Y+1) 的 128 位宽的多个字节/半字/字存储到存储器指定开始的第二个 128 位地址中，依次类推，最后一个寄存器 128 位宽的多个字节/半字/字存储到存储器指定开始的最后的 128 位地址中。存储器的有效地址由基址寄存器 RX 的内容决定。最后，更新基址寄存器 RX 为所有所存储数据之后的地址。
影响标志位	无影响
限制	VR _Z 应当大于等于 VR _Y 。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常、TLB 修改异常

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	8	7	6	5	4	3	0
1	11110	00000		RX		001010		10	IMM2	SIZE		VRZ				

注：

IMM2 域——指定目标寄存器的个数，IMM2 = Z - Y。

00——1 个目的寄存器

01——2 个目的寄存器

10——3 个目的寄存器

11——4 个目的寄存器

16.170 VSTRU.T.N——矢量存储基址跳跃更新指令

统一化指令	
语法	vstru.T.N vrz,(rx),ry (T=8/16/32 N=1/2/3/4)
操作	从矢量寄存器 VRZ 存储固定多个字节/半字/字到存储器中并更新基址 size = {8,16,32} num = {1,2,3,4} MEM[RX] ← VRZ[size*num-1:0]; RX ← RX + RY
编译结果	仅存在 32 位指令。 vstru.T.N vrz,(rx),ry
说明	从矢量寄存器 VRZ 存储多个字节/半字/字到存储器中并更新基址，存储的个数由指令决定。采用寄存器寻址方式。存储器的有效地址由基址寄存器 RX 得到。最后，更新基址寄存器 RX 为寄存器 RX 与 RY 的和。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常、TLB 修改异常

32 位指令	
操作	从矢量寄存器 VRZ 存储固定多个字节/半字/字到存储器中并更新基址 size = {8,16,32} num = {1,2,3,4} MEM[RX] ← VRZ[size*num-1:0]; RX ← RX + RY
语法	vstru.T.N vrz,(rx),ry
说明	从矢量寄存器 VRZ 存储多个字节/半字/字到存储器中并更新基址，存储的个数由指令决定。采用寄存器寻址方式。存储器的有效地址由基址寄存器 RX 得到。最后，更新基址寄存器 RX 为寄存器 RX 与 RY 的和。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常、TLB 修改异常

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	8	7	6	5	4	3	0
1	11110				RY		RX		001011		00	NUM	SIZE	VRZ		

16.171 VSTU.T.N——矢量存储基址更新指令

统一化指令	
语法	vstu.T.N vrz,(rx) (T=8/16/32 N=1/2/3/4)
操作	从矢量寄存器 VRZ 存储固定多个字节/半字/字到存储器中并更新基址 size = {8,16,32} num = {1,2,3,4} MEM[RX] ← VRZ[size*num-1:0]; RX ← RX + num*size/8
编译结果	仅存在 32 位指令。 vstu.T.N vrz,(rx)
说明	从矢量寄存器 VRZ 存储多个字节/半字/字到存储器中并更新基址，存储的个数由指令决定。采用寄存器寻址方式。存储器的有效地址由基址寄存器 RX 得到。最后，更新基址寄存器 RX 为所加载数据之后的地址。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常、TLB 修改异常

32 位指令	
操作	从矢量寄存器 VRZ 存储固定多个字节/半字/字到存储器中并更新基址 ** size = {8,16,32} num = {1,2,3,4} MEM[RX] ← VRZ[size*num-1:0]; RX ← RX + num*size/8
语法	vstu.T.N vrz,(rx)
说明	从矢量寄存器 VRZ 存储多个字节/半字/字到存储器中并更新基址，存储的个数由指令决定。采用寄存器寻址方式。存储器的有效地址由基址寄存器 RX 得到。最后，更新基址寄存器 RX 为所加载数据之后的地址。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常、TLB 修改异常

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	8	7	6	5	4	3	0
1	11110			00000			RX	001010			00	NUM	SIZE	VRZ		

16.172 VSTX.T——可变长度矢量存储指令

统一化指令	
语法	vstx.T vrz,(rx),ry (T=8/16/32)
操作	从矢量寄存器 VRZ 存储多个字节/半字/字到存储器中 size = {8,16,32} num = mod(ry,128/size) if(num==0) then num =128/size MEM[RX] ← VRZ[num*size-1:0];
编译结果	仅存在 32 位指令。 vstx.T vrz,(rx),ry
说明	从矢量寄存器 VRZ 存储多个字节/半字/字到存储器中，存储的个数由 RY 的后几位决定（如加载类型为字节，则由后 4 位决定；如加载类型为半字，则由后 3 位决定；如加载类型为字，则由后 2 位决定），如果 RY 的后几位为 0，则共存储 128 位数据。采用寄存器寻址方式。存储器的有效地址由基址寄存器 RX 得到。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常、TLB 修改异常

32 位指令	
操作	从矢量寄存器 VRZ 存储多个字节/半字/字到存储器中 size = {8,16,32} num = mod(ry,128/size) if(num==0) then num =128/size MEM[RX] ← VRZ[num*size-1:0];
语法	vstx.T vrz,(rx),ry
说明	从矢量寄存器 VRZ 存储多个字节/半字/字到存储器中，存储的个数由 RY 的后几位决定（如加载类型为字节，则由后 4 位决定；如加载类型为半字，则由后 3 位决定；如加载类型为字，则由后 2 位决定），如果 RY 的后几位为 0，则共存储 128 位数据。采用寄存器寻址方式。存储器的有效地址由基址寄存器 RX 得到。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常、TLB 修改异常。

32 位指令格式：

31	30	26	25	21	20	16	15	10	9	8	7	6	5	4	3	0
1	11110			RY		RX		001011		01	00		SIZE	VRZ		

16.173 VSUB.T——矢量减法指令

统一化指令	
语法	vsb.T vrz, vr_x, vr_y (T=S8/S16/S32/S64/U8/U16/U32/U64)
操作	size = {8, 16, 32, 64}; Number=128/size; for(i=0; i< Number; i++) { V_z(i)=V_x(i)-V_y(i); }
编译结果	仅存在 32 位指令。 vsb.T vrz, vr_x, vr_y
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	将 VR_X 与 VR_Y 的对应元素相减，结果存放于 VR_Z 中。 VR_X n-1 0 VR_Y n-1 0 - - - - VR_Z n-1 0
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8, 16, 32, 64}; Number=128/size; for(i=0; i< Number; i++) { Vz(i)=Vx(i)-Vy(i); }
语法	vsub.T vrz, vrx, vry
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	将 VRX 与 VRY 的对应元素相减，结果存放于 VRZ 中。 VRX n-1 0 VRY n-1 0 - - - - VRZ n-1 0
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

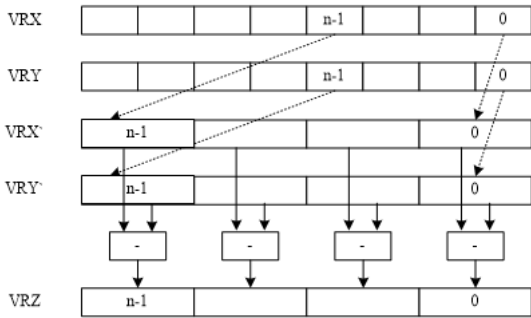
31	op	26	25	24	21	20	19	16	15	5	4	3	0			
111110		w1		VRY		w0		VRX		00000001000				S	VRZ	

注：

- 编码表中 w1, w0 用来表征指令位宽类型，其中：
- [w1,w0] = 2’ b00 表示 size 为 8bit 指令；
- [w1,w0] = 2’ b01 表示 size 为 16bit 指令；
- [w1,w0] = 2’ b10 表示 size 为 32bit 指令；
- [w1,w0] = 2’ b11 表示 size 为 64bit 指令；
- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.174 VSUB.T.E——矢量扩展减法指令

统一化指令	
语法	vs_{ub}.T.e vr_z, vr_x, vr_y (T=S8/S16/S32 /U8/U16/U32)
操作	size = {8,16,32}; Number=128/size for(i=0; i< Nmuber/2; i++) { V_z(2i+1:2i)=extend(V_x(i))+extend(V_y(i)) V_z+1(2i+12i)=extend(V_x(number/2+i)) - extend(V_y(number/2+i)); } extend 根据 U/S 将值零扩展或者符号扩展至元素位宽的 2 倍
编译结果	仅存在 32 位指令。 vs_{ub}.T.e vr_z, vr_x, vr_y
数据类型	U8/U16/U32 S8/S16/S32
说明	将 VRX 与 VRY 的对应元素扩展，并相减，运算结果存放于 VRZ 中。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32}; Number=128/size for(i=0; i< Nmuber/2; i++) { Vz(2i+1:2i)=extend(Vx(i))+extend(Vy(i)) i=0:number/2-1 Vz+1(2i+12i)=extend(Vx(number/2+i))-extend(Vy(number/2+i)); } extend 根据 U/S 将值零扩展或者符号扩展至元素位宽的 2 倍
语法	vsub.T.e vrz, vrx, vry
数据类型	U8/U16/U32 S8/S16/S32
说明	将 VRX 与 VRY 的对应元素扩展，并相减，运算结果存放于 VRZ 中。 
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0			
111110		w1	VRY			w0	VRX			00000110001				S	VRZ	

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.175 VSUB.T.H——矢量减法取高半指令

统一化指令	
语法	vsub.T.h vrz, vrx, vry (T=S16/S32/S64 /U16/U32/U64)
操作	<pre> size = {16,32,64}; Number=128/size for(i=0; i< Number; i++) { Tmp(i)=(Vx(i)-Vy(i))[size-1:size/2]; } for(i=0; i< Number/2; i++) { Vz(i)={Tmp(2i+1), Tmp(2i)}; } </pre>
编译结果	仅存在 32 位指令。 vsub.T.h vrz, vrx, vry
数据类型	U16/U32/U64 S16/S32/S64
说明	VRX 与 VRY 的对应元素相减，然后取加法结果的高半部分按序放至目的寄存器 VRZ 的低半部分。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	<pre>size = {16,32,64}; Number=128/size for(i=0; i< Number; i++) { Tmp(i)=(Vx(i)-Vy(i))[size-1:size/2]; i=i+(number-1) } for(i=0; i< Number/2; i++) { Vz(i)={Tmp(2i+1), Tmp(2i)}; }</pre>
语法	vsub.T.h vrz, vrx, vry
数据类型	U16/U32/U64 S16/S32/S64
说明	VRX 与 VRY 的对应元素相减，然后取加法结果的高半部分结果按序放至目的寄存器 VRZ 的低半部分。 The diagram shows three registers: VRX, VRY, and VRZ. VRX and VRY are n-bit registers. Arrows indicate that corresponding elements from VRX and VRY are subtracted (indicated by '-' signs). The results are then split into high (Hi) and low (Lo) halves. The high halves of the results are then stored into the low halves of the VRZ register.
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

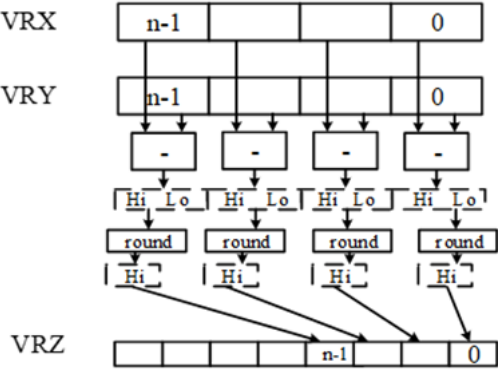
32 位指令格式：

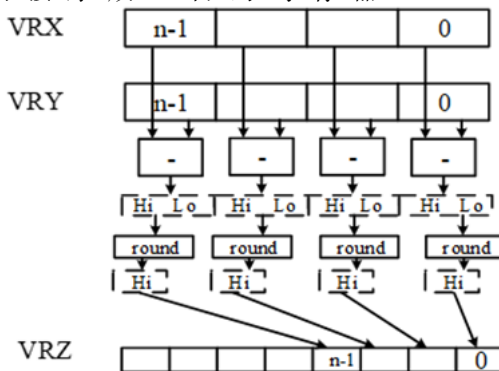
31	op	26	25	24	21	20	19	16	15		5	4	3	0
	111110	w1		VRY	w0		VRX		00000001011		S		VRZ	

注：

- 编码表中 w1, w0 用来表征指令位宽类型，其中：
- [w1,w0] = 2’ b01 表示 size 为 16bit 指令；
 - [w1,w0] = 2’ b10 表示 size 为 32bit 指令；
 - [w1,w0] = 2’ b11 表示 size 为 64bit 指令；
- 编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；
- 编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.176 VSUB.T.RH——矢量减法取高半指令

统一化指令	
语法	vsb.T.rh vrz, vr_x, vr_y (T=S16/S32/S64 /U16/U32/U64)
操作	size = {16,32,64}; Number=128/size round=1<<(size/2-1); for(i=0; i< Number; i++) { Tmp(i)=(V_x(i)-V_y(i)+round)[size-1:size/2]; } for(i=0; i< Number/2; i++) { V_z(i)={Tmp(2i+1),Tmp(2i)}; }
编译结果	仅存在 32 位指令。 vsb.T.rh vrz, vr_x, vr_y
数据类型	U16/U32/U64 S16/S32/S64
说明	VR_X 与 VR_Y 的对应元素相加，然后取加法结果的高半部分，舍入后按序放至目的寄存器 VR_Z 的低半部分。 
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。

32 位指令	
操作	<pre>size = {16,32,64}; Number=128/size round=1<<(size/2-1); for(i=0; i< Number; i++) { Tmp(i)=(Vx(i)-Vy(i)+round) [size-1:size/2]; } for(i=0; i< Number/2; i++) { Vz(i)={Tmp(2i+1),Tmp(2i)}; }</pre>
语法	vsub.T.rh vrz, vrx, vry
数据类型	U16/U32/U64 S16/S32/S64
说明	VRX 与 VRY 的对应元素相加，然后取加法结果的高半部分，舍入后按序放至目的寄存器 VRZ 的低半部分。 
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0	
111110		w1	VRY			w0	VRX			00000001100			S	VRZ

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2' b01 表示 size 为 16bit 指令；

[w1,w0] = 2' b10 表示 size 为 32bit 指令；

[w1,w0] = 2' b11 表示 size 为 64bit 指令；

编码表中 S 表征带符号运算还是无符号运算: 1 表示带符号, 0 表示无符号;

编码表中 VRX, VRY, VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.177 VSUB.T.S——矢量减法饱和指令

统一化指令	
语法	vsub.T.s vrz, vrx, vry (T=S8/S16/S32/S64 /U8/U16/U32/U64)
操作	size = {8, 16, 32, 64}; Number=128/size signed=(T==S); (根据元素 U/S 选择) Max=signed? 2^(size-1)-1: 2^(size)-1; Min=signed? -2^(size-1):0; for(i=0; i< Number; i++) { If Vx(i)-Vy(i)>Max Vz(i)=Max; Else if Vx(i)-Vy(i)<Min Vz(i)=Min; Else Vz(i)= Vx(i)-Vy(i); }
编译结果	仅存在 32 位指令。 vsub.T.s vrz, vrx, vry
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	将 VRX 与 VRY 的对应元素相减，并对减法结果做饱和运算，运算结果存放于 VRZ 中。其中，当减法结果大于目的元素可表示的最大值时，取目的元素最大值作为饱和运算结果，当减法结果小于目的元素可表示的最小值时，取目的元素最小值作为饱和运算结果，当减法结果处于目的元素可表示范围时，取减法结果作为饱和运算值。
	<pre> graph TD VRX[VRX] --> S1_1[n-1] VRX --> S1_2[] VRX --> S1_3[] VRX --> S1_4[0] VRY[VRY] --> S2_1[n-1] VRY --> S2_2[] VRY --> S2_3[] VRY --> S2_4[0] S1_1 --> Minus1[-] S1_2 --> Minus2[-] S1_3 --> Minus3[-] S1_4 --> Minus4[-] Minus1 --> Sat1[sat] Minus2 --> Sat2[sat] Minus3 --> Sat3[sat] Minus4 --> Sat4[sat] Sat1 --> VRZ[n-1] Sat2 --> VRZ[] Sat3 --> VRZ[] Sat4 --> VRZ[0] </pre>
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8, 16, 32, 64}; Number=128/size signed=(T==S); (根据元素 U/S 类型选择) Max=signed? 2^(size-1)-1: 2^(size)-1; Min=signed? -2^(size-1):0; for(i=0; i< Number; i++) { If V_x(i)-V_y(i)>Max V_z(i)=Max; Else if V_x(i)-V_y(i)<Min V_z(i)=Min; Else V_z(i)= V_x(i)-V_y(i); }
语法	vsub.T.s vrz, vr _x , vr _y
数据类型	U8/U16/U32/U64 S8/S16/S32/S64
说明	将 VR_X 与 VR_Y 的对应元素相减，并对减法结果做饱和运算，运算结果存放于 VR_Z 中。其中，当减法结果大于目的元素可表示的最大值时，取目的元素最大值作为饱和运算结果，当减法结果小于目的元素可表示的最小值时，取目的元素最小值作为饱和运算结果，当减法结果处于目的元素可表示范围时，取减法结果作为饱和运算值。 VR_X VR_Y VR_Z n-1 0 n-1 0 - - - - sat sat sat sat n-1 0
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15		5	4	3	0		
111110			w1		VR _Y		w0		VR _X		00000001001			S	VR _Z	

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

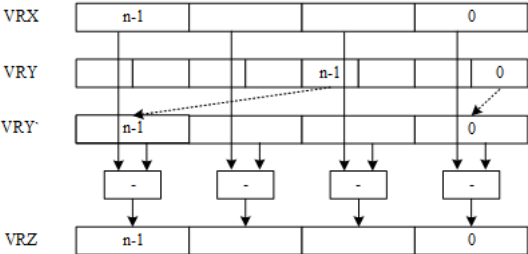
[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

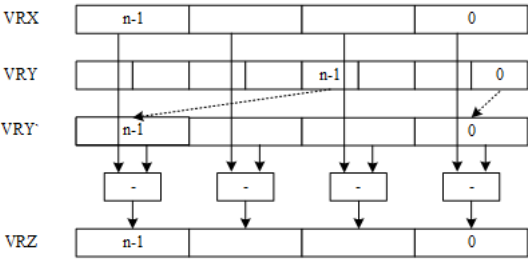
$[w1,w0] = 2'$ b11 表示 size 为 64bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.178 VSUB.T.X——矢量非对齐减法指令

统一化指令	
语法	vsusb.T.x vrz, vrx, vry (T=S8/S16/S32 /U8/U16/U32)
操作	size = {8,16,32}; Number=128/size for(i=0; i< Number/2; i++) { Vz(2i+1:2i)=Vx(2i+1:2i)-extend(Vy(i)); Vz+1(2i+1:2i)=Vx+1(2i+1:2i) - extend(Vy(number/2+i)); } extend 根据 U/S 将值零扩展或者符号扩展至元素位宽的 2 倍
编译结果	仅存在 32 位指令。 vsusb.T.x vrz, vrx, vry
数据类型	U8/U16/U32 S8/S16/S32
说明	将 VRY 的元素扩展，并与 VRX 的对应元素相减，运算结果存放于 VRZ 中。 
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32}; Number=128/size for(i=0; i< Number/2; i++) { Vz(2i+1:2i)=Vx(2i+1:2i)-extend(Vy(i)); V z+1(2i+1:2i)=Vx+1(2i+1:2i)-extend(Vy(number/2+i)); } extend 根据 U/S 将值零扩展或者符号扩展至元素位宽的 2 倍
语法	vsub.T.x vrz, vrx, vry
数据类型	U8/U16/U32 S8/S16/S32
说明	将 VRY 的元素扩展，并与 VRX 的对应元素相减，运算结果存放于 VRZ 中。 
影响标志位	无影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0			
111110			w1		VRY		w0		VRX		00000001010			S	VRZ	

注：

编码表中 w1，w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.179 VSUBH.T——矢量减法平均指令

统一化指令	
语法	<code>vsubh.T vrz, vrx, vry</code> (T=S8/S16/S32 /U8/U16/U32)
操作	<code>size = {8,16,32};</code> <code>Number=128/size</code> <code>for(i=0; i< Number; i++)</code> <code>{</code> <code> Vz(i)=(Vx(i)-Vy(i))>>1;</code> <code>}</code> 对于 U, 右移为逻辑右移, 对于 S, 右移为算术右移
编译结果	仅存在 32 位指令。 <code>vsubh.T vrz, vrx, vry</code>
数据类型	U8/U16/U32 S8/S16/S32
说明	VRX 与 VRY 的对应元素相减, 然后对减法结果取平均值, 结果存放于 VRZ 中。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	<code>size = {8,16,32};</code> <code>Number=128/size</code> <code>for(i=0; i< Number; i++)</code> <code>{</code> <code> Vz(i)=(Vx(i)-Vy(i))>>1;</code> <code>}</code> 对于 U, 右移为逻辑右移, 对于 S, 右移为算术右移
语法	<code>vsubh.T vrz, vrx, vry</code>
数据类型	U8/U16/U32 S8/S16/S32
说明	VRX 与 VRY 的对应元素相减, 然后对减法结果取平均值, 结果存放于 VRZ 中。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式:

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1		VRY	w0		VRX		00000011100	S		VRZ	

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

编码表中 S 表征带符号运算还是无符号运算：1 表示带符号，0 表示无符号；

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.180 VSUBH.T.R——矢量减法平均并舍入指令

统一化指令	
语法	vsubh.T.r vrz, vrx, vry (T=S8/S16/S32 /U8/U16/U32)
操作	size = {8,16,32}; Number=128/size for(i=0; i< Number; i++) { Vz(i)=(Vx(i)-Vy(i)+1)>>1; } 对于 U, 右移为逻辑右移，对于 S, 右移为算术右移
编译结果	仅存在 32 位指令。 vsubh.T.r vrz, vrx, vry
数据类型	U8/U16/U32 S8/S16/S32
说明	VRX 与 VRY 的对应元素相减，然后对减法结果取平均值并做饱和操作，舍入结果存放于 VRZ 中。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32}; Number=128/size for(i=0; i< Number; i++) { Vz(i)=(Vx(i)-Vy(i)+1)>>1; } 对于 U, 右移为逻辑右移, 对于 S, 右移为算术右移
语法	vsubh.T.r vrz, vrX, vry
数据类型	U8/U16/U32 S8/S16/S32
说明	VRX 与 VRY 的对应元素相减, 然后对减法结果取平均值并做饱和操作, 舍入结果存放于 VRZ 中。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式:

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1	VRY		w0	VRX		00000011101			S	VRZ	

注:

编码表中 w1, w0 用来表征指令位宽类型, 其中:

[w1,w0] = 2' b00 表示 size 为 8bit 指令;

[w1,w0] = 2' b01 表示 size 为 16bit 指令;

[w1,w0] = 2' b10 表示 size 为 32bit 指令;

编码表中 S 表征带符号运算还是无符号运算: 1 表示带符号, 0 表示无符号;

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.181 VSWP.T——矢量数据交换指令

统一化指令	
语法	vswp.T vrz, vrx (T=8/16/32/64)
操作	Vrz(i)=Vrx(i); Vrx(i)=Vrz(i); i=0:(number-1) 把 VX 和 VZ 每个元素进行交换
编译结果	仅存在 32 位指令。 vswp.T vrz, vrx
数据类型	8/16/32/64
说明	将源矢量寄存器 VRX 和 VRZ 的元素进行交换。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	Vrz(i)=Vrx(i); Vrx(i)=Vrz(i); i=0:(number-1) 将 Vrx,Vry 中每个元素数据进行交换
语法	vswp.T vrz, vrx
数据类型	8/16/32/64
说明	将源矢量寄存器 VRX 和 VRZ 的每个元素进行交换。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15		5	4	3	0
111110			w1	X		w0	VRX		00001111000			0	VRZ	

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

[w1,w0] = 2’ b11 表示 size 为 64bit 指令；

编码表中 VRX,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.182 VTBL.T——矢量数据链接指令

统一化指令	
语法	vtbl.T vrz,vrx,vry (T=8)
操作	size = 8; number = 128/size; if Vy(i)<16 Vz(i)=Vx(Vy(i)); else Vz(i)=8’ b0; i=0:(number-1) 如果 Vry[i] 的值大于 16，则写 0，否则用 Vry[i] 的值作为 index 选择 Vrx 的 element 写入 Vrz 相应的元素之中 vrz[i] 指相应 element 的数据;
编译结果	仅存在 32 位指令。 vtbl.T vrz, vrx,vry
数据类型	8
说明	如果 Vry[i] 的值大于 16，则 Vrz 相应元素写 0，否则用 Vry[i] 的值作为 index 选择 Vrx 的 element 写入 Vrz 相应的元素之中。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = 8; number = 128/size; if Vy(i)<16 Vz(i)=Vx(Vy(i)); else Vz(i)=8' b0; i=0:(number-1) 如果 Vry[i] 的值大于 16，则相应元素写 0，否则用 Vry[i] 的值作为 index 选择 Vrx 的 element 写入 Vrz 相应的元素之中
语法	vtbl.T vrz, vrx,vry
数据类型	8
说明	如果 Vry[i] 的值大于 16，则 Vrz 相应元素写 0，否则用 Vry[i] 的值作为 index 选择 Vrx 的 element 写入 Vrz 相应的元素之中。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0	
111110		0	VRY			0	VRX			00001100100		0	VRZ	

注：

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.183 VTBX.T——矢量数据链接指令

统一化指令	
语法	vtbx.T vrz,vrx,vry (T=8)
操作	size = 8; number = 128/size; if Vy(i)<16 Vz(i)=Vx(Vy(i)); else Vz(i)=Vz(i); i=0:(number-1) 如果 Vry[i] 的值大于 16，则保持原值，否则用 Vry[i] 的值作为 index 选择 Vrx 的 element 写入 Vrz 相应的元素之中 vrz[i] 指相应 element 的数据;
编译结果	仅存在 32 位指令。 vtbx.T vrz, vrx,vry
数据类型	8
说明	如果 Vry[i] 的值大于 16，则 Vrz 相应元素保持原值，否则用 Vry[i] 的值作为 index 选择 Vrx 的 element 写入 Vrz 相应的元素之中。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = 8; number = 128/size; if Vy(i)<16 Vz(i)=Vx(Vy(i)); else Vz(i)=Vz(i); i=0:(number-1) 如果 Vry[i] 的值大于 16，则相应元素保持原值，否则用 Vry[i] 的值作为 index 选择 Vrx 的 element 写入 Vrz 相应的元素之中
语法	vtbx.T vrz, vrx,vry
数据类型	8
说明	如果 Vry[i] 的值大于 16，则 Vrz 相应元素保持原值，否则用 Vry[i] 的值作为 index 选择 Vrx 的 element 写入 Vrz 相应的元素之中。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

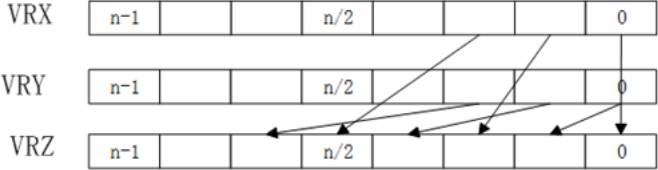
32 位指令格式：

31	op	26	25	24	21	20	19	16	15	4	3	0
111110	0	VRY	0	VRX	000011001010	VRZ						

注：

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.184 VTRN.T——矢量数据交叉传输指令

统一化指令	
语法	vtrn.T vrz, vrx,vry (T=8/16/32)
操作	size = {8,16,32}; number = 128/size; Vrz(2i+1)=Vry(2i); Vrz(2i)= Vrx(2i); i=0:number/2-1 Vrz+1(2i+1)= Vry(2i+1), Vrz+1(2i)=Vrx(2i+1); i=0:number/2-1 把 Vrx 和 Vry 的元素交叉放入 Vrz 和 Vrz+1 之中 vrx[i] 指相应 element 的数据;
编译结果	仅存在 32 位指令。 vtrn.T vrz, vrx
数据类型	8/16/32
说明	将源矢量寄存器 VRX 和 VRY 的元素交叉放入 VRZ 和 VRZ+1 之中。 
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32} number = 128/size; vr_x[i],vr_y[i] 指相应位置 element 的数据; VR_z(2i+1)=VR_y(2i); VR_z(2i)= VR_x(2i); i=0:number/2-1 VR_z+1(2i+1)= VR_y(2i+1), VR_z+1(2i)=VR_x(2i+1); i=0:number/2-1 将 VR_x,VR_y 中每个元素数据交叉放入 VR_z 和 VR_z+1 中。
语法	vtrn.T vr _z , vr _x ,vr _y
数据类型	8/16/32
说明	将源矢量寄存器 VR_x 和 VR_y 的元素交叉放入 VR_z 和 VR_z+1 中。 VR_x VR_y VR_z
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		w1		X		w0		VRX		00001111001		0	VRZ

注：

编码表中 w1, w0 用来表征指令位宽类型，其中：

[w1,w0] = 2’ b00 表示 size 为 8bit 指令；

[w1,w0] = 2’ b01 表示 size 为 16bit 指令；

[w1,w0] = 2’ b10 表示 size 为 32bit 指令；

编码表中 VRX,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.185 VTST.T——矢量按位与置位指令

统一化指令	
语法	vtst.T vrz,vrx,vry (T=8/16/32/64)
操作	size = {8,16,32,64}; number = 128/size; Vz(i)=((Vx(i)&Vy(i)))?11111:00000; i=0:(number-1)
编译结果	仅存在 32 位指令。 vtst.T vrz, vrx,vry
数据类型	8/16/32/64
说明	当 Vrx, Vry 存在相同位均为 1 时，将此 element 置为全 1，否则为 0。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令	
操作	size = {8,16,32,64}; number = 128/size; Vz(i)=((Vx(i)&Vy(i))) ? 111...11 : 000...00; i=0:(number-1)
语法	vtst.T vrz, vrx,vry
数据类型	8/16/32/64
说明	当 Vrx, Vry 存在相同位均为 1 时，将此 element 置为全 1，否则为 0。
影响标志位	不影响
限制	寄存器范围为 vr0-vr15。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		x	VRY		x	VRX		00011000010			1	VRZ	

注：

编码表中 VRX,VRY,VRZ 分别表征这条指令各个矢量寄存器的寄存器号。

16.186 VXOR.T——矢量异或指令

统一化指令	
语法	<code>vxor.T vrz,vrx,vry</code> (T=8/16/32/64)
操作	for j=0:127 $Vz[j]=Vx[j] \wedge Vy[j]$ <code>vrz[j]</code> , <code>vry[j]</code> , <code>vrz[j]</code> 指操作数比特位 j 的数据;
编译结果	仅存在 32 位指令。 <code>vxor.T vrz, vrz,vry</code>
数据类型	8/16/32/64
说明	将 <code>Vrx</code> , <code>Vry</code> 相应比特位数据做异或操作并写入 <code>Vrz</code> 相应的比特位之中。 注：T 在此并无实际意义，仅为用户使用方便。
影响标志位	不影响
限制	寄存器范围为 <code>vr0-vr15</code> 。
异常	无

32 位指令	
操作	for j=0:127 $Vz[j]=Vx[j] \wedge Vy[j]$ <code>vrz[j]</code> , <code>vry[j]</code> , <code>vrz[j]</code> 指操作数比特位 j 的数据;
语法	<code>vxor.T vrz, vrz,vry</code>
数据类型	8/16/32/64
说明	将 <code>Vrx</code> , <code>Vry</code> 相应比特位数据做异或操作并写入 <code>Vrz</code> 相应的比特位之中。 注：T 在此并无实际意义，仅为用户使用方便。
影响标志位	不影响
限制	寄存器范围为 <code>vr0-vr15</code> 。
异常	无

32 位指令格式：

31	op	26	25	24	21	20	19	16	15	5	4	3	0
111110		x	VRY		x	VRX		00011000001			0	VRZ	

注：

编码表中 `VRX`,`VRY`,`VRZ` 分别表征这条指令各个矢量寄存器的寄存器号。