

# 玄铁 E804 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](http://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](http://www.xrvm.cn)

# 版本历史

版本	描述	日期
01	第一次正式发布。	2021.07.31
02	更新模板。	2024.04.24

# 玄铁 E804 用户手册

<b>第一章 概述</b>	<b>1</b>
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 . . . . .	3
1.6.2 术语 . . . . .	3
<b>第二章 微体系结构</b>	<b>6</b>
2.1 结构框图 . . . . .	6
2.2 流水线介绍 . . . . .	7
2.3 浮点处理单元 . . . . .	8
2.4 DSP 扩展单元 . . . . .	9
2.5 可信防护技术 . . . . .	9
2.6 紧耦合 IP 架构 . . . . .	10
<b>第三章 编程模型</b>	<b>11</b>
3.1 工作模式及寄存器视图 . . . . .	11
3.2 通用寄存器 . . . . .	12
3.2.1 条件码 / 进位标志位 . . . . .	13
3.3 系统控制寄存器 . . . . .	14
3.3.1 处理器状态寄存器 (PSR, CR<0,0>) . . . . .	14
3.3.2 更新 PSR . . . . .	16
3.3.3 向量基址寄存器 (VBR, CR<1,0>) . . . . .	16
3.3.4 异常保留寄存器 (CR<2,0>, CR<4,0>) . . . . .	17
3.3.5 产品序号寄存器 (CPUIDR, CR<13,0>) . . . . .	17
3.3.6 隐式操作寄存器 (CHR, CR<31,0>) . . . . .	17
3.3.7 其它控制寄存器 . . . . .	18
3.3.8 普通用户模式通用寄存器 14 (R14(User SP), CR<14,1>) . . . . .	18
3.3.9 中断指针寄存器 ( R14(Int_SP), CR<15,1>) . . . . .	18
3.4 数据大小端 . . . . .	19

3.5	数据对齐访问 . . . . .	19
3.6	系统地址映射 . . . . .	19
3.7	内存访问顺序 . . . . .	20
<b>第四章</b>	<b>异常处理</b>	<b>22</b>
4.1	异常处理概述 . . . . .	22
4.2	异常类型 . . . . .	24
4.2.1	重启异常（向量偏移 0X0） . . . . .	24
4.2.2	未对齐访问异常（向量偏移 0X4） . . . . .	24
4.2.3	访问错误异常（向量偏移 0X8） . . . . .	25
4.2.4	除以零异常（向量偏移 0X0C） . . . . .	25
4.2.5	非法指令异常（向量偏移 0X10） . . . . .	25
4.2.6	特权违反异常（向量偏移 0X14） . . . . .	25
4.2.7	跟踪异常（向量偏移 0X18） . . . . .	25
4.2.8	断点异常（向量偏移 0X1C） . . . . .	26
4.2.9	不可恢复错误异常（向量偏移 0X20） . . . . .	26
4.2.10	IDLY 异常（异常偏移 0X24） . . . . .	26
4.2.11	陷阱指令异常（向量偏移 0X40 - 0X4C） . . . . .	26
4.2.12	TSPEND 中断（向量便宜 0X58） . . . . .	26
4.2.13	浮点异常（向量偏移 0X78） . . . . .	26
4.3	中断异常 . . . . .	26
4.4	异常优先级 . . . . .	28
4.5	异常返回 . . . . .	29
<b>第五章</b>	<b>紧耦合 IP</b>	<b>30</b>
5.1	紧耦合 IP 简介 . . . . .	30
5.2	系统统计时器 . . . . .	31
5.2.1	简介 . . . . .	31
5.2.2	寄存器定义 . . . . .	31
5.2.2.1	控制和状态寄存器 (CORET_CSR) . . . . .	32
5.2.2.2	回填值寄存器 (CORET_RVR) . . . . .	33
5.2.2.3	当前值寄存器 (CORET_CVR) . . . . .	34
5.2.2.4	校准寄存器 (CORET_CALIB) . . . . .	34
5.2.3	操作步骤 . . . . .	35
5.3	矢量中断控制器 . . . . .	36
5.3.1	简介 . . . . .	36
5.3.2	寄存器定义 . . . . .	37
5.3.2.1	中断使能设置寄存器 (VIC_ISER) . . . . .	39
5.3.2.2	中断低功耗唤醒设置寄存器 (VIC_IWER) . . . . .	40
5.3.2.3	中断使能清除寄存器 (VIC_ICER) . . . . .	40
5.3.2.4	中断低功耗唤醒清除寄存器 (VIC_IWDR) . . . . .	41
5.3.2.5	中断等待设置寄存器 (VIC_ISPR) . . . . .	41

5.3.2.6 中断等待清除寄存器 (VIC_ICPR) . . . . .	42
5.3.2.7 中断响应状态寄存器 (VIC_IABR) . . . . .	42
5.3.2.8 中断优先级设置寄存器 (VIC_IPR0 - VIC_IPR31) . . . . .	43
5.3.2.9 中断状态寄存器 (VIC_ISR) . . . . .	45
5.3.2.10 中断优先级阈值寄存器 (VIC_IPTR) . . . . .	45
5.3.2.11 Tspend 中断使能设置寄存器 (VIC_TSPEND) . . . . .	46
5.3.2.12 Tspend 中断响应状态寄存器 (VIC_TSABR) . . . . .	47
5.3.2.13 Tspend 中断优先级设置寄存器 (VIC_TSPR) . . . . .	47
5.3.3 中断处理机制 . . . . .	48
5.3.3.1 中断状态位 . . . . .	48
5.3.3.2 中断优先级 . . . . .	49
5.3.3.3 中断向量号 . . . . .	49
5.3.3.4 中断处理过程 . . . . .	50
5.3.3.5 中断嵌套 . . . . .	51
5.3.4 Tspend 中断 . . . . .	54
5.3.5 操作步骤 . . . . .	54
5.3.6 接口信号 . . . . .	54
5.3.7 中断设置示例 . . . . .	57
<b>第六章 指令集</b>	<b>59</b>
6.1 概述 . . . . .	59
6.2 32 位指令 . . . . .	59
6.2.1 32 位指令功能分类 . . . . .	59
6.2.1.1 数据运算指令 . . . . .	59
6.2.1.2 分支跳转指令 . . . . .	63
6.2.1.3 内存存取指令 . . . . .	64
6.2.1.4 特权指令 . . . . .	65
6.2.1.5 特殊功能指令 . . . . .	66
6.2.1.6 DSP 扩展指令 . . . . .	66
6.2.3 16 位指令 . . . . .	72
6.3.1 16 位指令功能分类 . . . . .	72
6.3.1.1 数据运算指令 . . . . .	72
6.3.1.2 分支跳转指令 . . . . .	74
6.3.1.3 内存存取指令 . . . . .	74
6.3.1.4 多寄存器存取指令 . . . . .	75
6.4 指令集列表 . . . . .	75
6.5 指令执行延迟 . . . . .	90
<b>第七章 内存保护</b>	<b>100</b>
7.1 内存保护单元简介 . . . . .	100
7.2 相关系统控制寄存器 . . . . .	100
7.2.1 高速缓存配置寄存器 (CCR, CR<18,0>) . . . . .	100

7.2.2 可高缓和访问权限配置寄存器 (CAPR, CR<19,0>) . . . . .	101
7.2.3 保护区控制寄存器 (PACR, CR<20,0>) . . . . .	102
7.2.4 保护区选择寄存器 (PRSR, CR<21,0>) . . . . .	103
7.3 内存访问处理 . . . . .	104
<b>第八章 片上高速缓存 . . . . .</b>	<b>105</b>
8.1 高速缓存简介 . . . . .	105
8.2 相关系统控制寄存器 . . . . .	105
8.2.1 高速缓存使能寄存器 (CER) . . . . .	106
8.2.2 高速缓存无效寄存器 (CIR) . . . . .	107
8.2.3 可高缓区配置寄存器 0~3 (CRCR) . . . . .	107
8.2.4 高速缓存性能分析控制寄存器 (CPFCR) . . . . .	109
8.2.5 高速缓存访问次数寄存器 (CPFATR) . . . . .	109
8.2.6 高速缓存缺失次数寄存器 (CPFMTR) . . . . .	109
<b>第九章 总线矩阵与总线接口 . . . . .</b>	<b>111</b>
9.1 简介 . . . . .	111
9.2 系统总线接口 . . . . .	113
9.2.1 特点 . . . . .	113
9.2.2 协议内容 . . . . .	113
9.2.2.1 支持传输类型 . . . . .	113
9.2.2.2 支持响应类型 . . . . .	113
9.2.3 不同总线响应下的行为 . . . . .	114
9.2.4 AHB 协议的接口信号 . . . . .	114
9.2.5 AHB-Lite 协议的接口信号 . . . . .	116
9.3 指令总线接口 . . . . .	117
9.3.1 特点 . . . . .	117
9.3.2 协议内容 . . . . .	117
9.3.2.1 支持传输类型 . . . . .	117
9.3.2.2 支持响应类型 . . . . .	117
9.3.3 不同总线响应下的行为 . . . . .	118
9.3.4 指令总线接口信号 . . . . .	119
9.4 数据总线接口 . . . . .	120
9.4.1 特点 . . . . .	120
9.4.2 协议内容 . . . . .	120
9.4.2.1 支持传输类型 . . . . .	120
9.4.2.2 支持响应类型 . . . . .	121
9.4.3 不同总线响应下的行为 . . . . .	121
9.4.4 数据总线接口信号 . . . . .	122
9.5 指令与数据的访问顺序 . . . . .	123
<b>第十章 调试接口 . . . . .</b>	<b>124</b>
10.1 概述 . . . . .	124

10.2 外部接口 . . . . .	125
<b>第十一章 工作模式与转换</b>	<b>127</b>
11.1 正常工作模式 . . . . .	127
11.2 低功耗模式 . . . . .	128
11.3 调试模式 . . . . .	128
11.3.1 调试模式 . . . . .	128
11.3.2 进入调试模式 . . . . .	128
11.3.3 退出调试模式 . . . . .	129
<b>第十二章 浮点处理单元</b>	<b>130</b>
12.1 概述 . . . . .	130
12.1.1 简介 . . . . .	130
12.1.2 特点 . . . . .	130
12.2 微体系结构 . . . . .	131
12.2.1 浮点单元介绍 . . . . .	131
12.3 编程模型 . . . . .	131
12.3.1 简介 . . . . .	131
12.3.2 数据格式 . . . . .	131
12.3.2.1 整形数据格式 . . . . .	131
12.3.2.2 单精度数据格式 . . . . .	132
12.3.3 浮点寄存器描述 . . . . .	132
12.3.3.1 与通用寄存器传输数据 . . . . .	132
12.3.3.2 维护寄存器精度的一致 . . . . .	132
12.3.4 系统寄存器 . . . . .	132
12.3.4.1 浮点系统寄存器的读写方法 . . . . .	134
12.3.4.2 浮点版本寄存器 (FPU ID Register, FID) . . . . .	134
12.3.4.3 浮点控制寄存器 (FPU Control Register, FCR) . . . . .	135
12.3.4.4 浮点异常状态处理器 (FPU Exception Status Register, FESR) . . . . .	137
12.3.5 数据大小端 . . . . .	138
12.4 异常处理 . . . . .	138
12.4.1 浮点异常介绍 . . . . .	139
12.4.2 非规格数输入异常 . . . . .	139
12.4.2.1 异常使能 . . . . .	139
12.4.2.2 异常不使能 . . . . .	140
12.4.3 非法操作异常 . . . . .	140
12.4.3.1 异常使能 . . . . .	140
12.4.3.2 异常不使能 . . . . .	141
12.4.4 除以零异常 . . . . .	141
12.4.4.1 异常使能 . . . . .	141
12.4.4.2 异常不使能 . . . . .	141
12.4.5 上溢异常 . . . . .	141

12.4.5.1 异常使能 . . . . .	141
12.4.5.2 异常不使能 . . . . .	142
12.4.6 下溢异常 . . . . .	142
12.4.6.1 异常使能 . . . . .	142
12.4.6.2 异常不使能 . . . . .	142
12.4.7 非精确异常 . . . . .	142
12.4.7.1 异常使能 . . . . .	143
12.4.7.2 异常不使能 . . . . .	143
12.4.8 异常优先级 . . . . .	143
12.4.9 特殊指令的异常 . . . . .	143
12.5 指令集 . . . . .	143
12.5.1 浮点指令列表 . . . . .	144
12.5.1.1 数据运算指令 . . . . .	144
12.5.1.2 传输类指令 . . . . .	145
12.5.1.3 内存存取指令 . . . . .	145
12.5.2 浮点指令执行延时 . . . . .	145
<b>第十三章 初始参考代码</b>	<b>147</b>
13.1 MPU 设置示例 . . . . .	147
13.2 高速缓存设置示例 . . . . .	149
13.3 中断使能初始化 . . . . .	150
13.4 通用寄存器初始化示例 . . . . .	150
13.5 堆栈指针初始化示例 . . . . .	151
13.6 异常和中断服务程序入口地址设置示例 . . . . .	151
13.7 FPU 初始化示例 . . . . .	152
<b>第十四章 附录 A 基本指令术语表</b>	<b>154</b>
14.1 ABS——绝对值指令 . . . . .	154
14.2 ADDC——无符号带进位加法指令 . . . . .	155
14.3 ADDI——无符号立即数加法指令 . . . . .	156
14.4 ADDI(SP)——无符号(堆栈指针)立即数加法指令 . . . . .	160
14.5 ADDU——无符号加法指令 . . . . .	162
14.6 AND——按位与指令 . . . . .	164
14.7 ANDI——立即数按位与指令 . . . . .	165
14.8 ANDN——按位非与指令 . . . . .	166
14.9 ANDNI——立即数按位非与指令 . . . . .	167
14.10 ASR——算术右移指令 . . . . .	168
14.11 ASRC——立即数算术右移至 C 位指令 . . . . .	170
14.12 ASRI——立即数算术右移指令 . . . . .	172
14.13 BCLRI——立即数位清零指令 . . . . .	173
14.14 BEZ——寄存器等于零分支指令 . . . . .	175
14.15 BF——C 为 0 分支指令 . . . . .	176

14.16 BGENI——立即数位产生指令	178
14.17 BGENR——寄存器位产生指令	179
14.18 BHSZ——寄存器大于等于零分支指令	180
14.19 BHZ——寄存器大于零分支指令	181
14.20 BKPT——断点指令	182
14.21 BLSZ——寄存器小于等于零分支指令	183
14.22 BLZ——寄存器小于零分支指令	185
14.23 BMASKI——立即数位屏蔽产生指令	186
14.24 BNEZ——寄存器不等于零分支指令	188
14.25 BNEZAD——寄存器自减大于零分支指令	189
14.26 BR——无条件跳转指令	191
14.27 BREV——位倒序指令	192
14.28 BSETI——立即数位置位指令	194
14.29 BSR——跳转到子程序指令	195
14.30 BT——C 为 1 分支指令	198
14.31 BTSTI——立即数位测试指令	199
14.32 CLRF——C 为 0 清零指令	200
14.33 CLRT——C 为 1 清零指令	201
14.34 CMPHS——无符号大于等于比较指令	202
14.35 CMPHSI——立即数无符号大于等于比较指令	204
14.36 CMPLT——有符号小于比较指令	207
14.37 CMPLTI——立即数有符号小于比较指令	209
14.38 CMPNE——不等比较指令	212
14.39 CMPNEI——立即数不等比较指令	214
14.40 DECF——C 为 0 立即数减法指令	216
14.41 DECGT——减法大于零置 C 位指令	217
14.42 DECLT——减法小于零置 C 位指令	218
14.43 DECNE——减法不等于零置 C 位指令	219
14.44 DECT——C 为 1 立即数减法指令	220
14.45 DIVS——有符号除法指令	221
14.46 DIVU——无符号除法指令	222
14.47 DOZE——进入低功耗睡眠模式指令	223
14.48 FF0——快速找 0 指令	224
14.49 FF1——快速找 1 指令	225
14.50 GRS——符号产生指令	226
14.51 IDLY——中断识别禁止指令	227
14.52 INCF——C 为 0 立即数加法指令	229
14.53 INCT——C 为 1 立即数加法指令	230
14.54 INS——位插入指令	231
14.55 IPOP——中断出栈指令	233
14.56 IPUSH——中断压栈指令	234
14.57 IXH——索引半字指令	235

14.58 IXW——索引字指令	236
14.59 IXd——索引双字指令	236
14.60 JMP——寄存器跳转指令	237
14.61 JMPI——间接跳转指令	239
14.62 JSR——寄存器跳转到子程序指令	240
14.63 JSRI——间接跳转到子程序指令	242
14.64 LD.B——无符号扩展字节加载指令	244
14.65 LD.BS——有符号扩展字节加载指令	247
14.66 LD.H——无符号扩展半字加载指令	248
14.67 LD.HS——有符号扩展半字加载指令	251
14.68 LD.W——字加载指令	252
14.69 LDM——连续多字加载指令	255
14.70 LDQ——连续四字加载指令	257
14.71 LDR.B——寄存器移位寻址无符号扩展字节加载指令	259
14.72 LDR.BS——寄存器移位寻址有符号扩展字节加载指令	260
14.73 LDR.H——寄存器移位寻址无符号扩展半字加载指令	262
14.74 LDR.HS——寄存器移位寻址有符号扩展半字加载指令	264
14.75 LDR.W——寄存器移位寻址字加载指令	266
14.76 LRS.B——字节符号加载指令	268
14.77 LRS.H——半字符串加载指令	270
14.78 LRS.W——字符串加载指令	271
14.79 LRW——存储器读入指令	273
14.80 LSL——逻辑左移指令	275
14.81 LSIC——立即数逻辑左移至 C 位指令	276
14.82 LSIL——立即数逻辑左移指令	278
14.83 LSR——逻辑右移指令	280
14.84 LSRC——立即数逻辑右移至 C 位指令	281
14.85 LSRI——立即数逻辑右移指令	283
14.86 MFCR——控制寄存器读传送指令	284
14.87 MOV——数据传送指令	285
14.88 MOVF——C 为 0 数据传送指令	286
14.89 MOVI——立即数数据传送指令	287
14.90 MOVIH——立即数高位数据传送指令	288
14.91 MOVT——C 为 1 数据传送指令	289
14.92 MTCR——控制寄存器写传送指令	290
14.93 MULA.32.I——32 位有符号乘累加取低 32 位指令	290
14.94 MULALL.S16.S——带饱和操作的 16 位有符号低半字乘累加指令	291
14.95 MULA.(U/S)32——32 位(无/有) 符号乘累加指令	293
14.96 MULT——乘法指令	294
14.97 MUL.(U/S)32——32 位(无/有) 符号乘法指令	295
14.98 MVC——C 位传送指令	296
14.99 MVCV——C 位取反传送	297

14.100 NIE——中断嵌套使能指令	298
14.101 NIR——中断嵌套返回指令	300
14.102 NOR——按位或非指令	302
14.103 NOT——按位非指令	303
14.104 OR——按位或指令	304
14.105 ORI——立即数按位或指令	305
14.106 POP——出栈指令	307
14.107 PSRCLR——PSR 位清零指令	311
14.108 PSRSET——PSR 位置位指令	313
14.109 PUSH——压栈指令	315
14.110 REVB——字节倒序指令	319
14.111 REVH——半字字节倒序指令	320
14.112 ROTL——循环左移指令	322
14.113 ROTLI——立即数循环左移指令	323
14.114 RSUB——反向减法指令	324
14.115 RTS——子程序返回指令	325
14.116 RTE——异常和普通中断返回指令	326
14.117 SCE——条件执行设置指令	327
14.118 SEXT——位提取并有符号扩展指令	330
14.119 SEXTB——字节提取并有符号扩展指令	332
14.120 SEXTH——半字提取并有符号扩展指令	333
14.121 SRS.B——字节符号存储指令	335
14.122 SRS.H——半字符串存储指令	336
14.123 SRS.W——字符串存储指令	338
14.124 ST.B——字节存储指令	339
14.125 ST.H——半字存储指令	342
14.126 ST.W——字存储指令	344
14.127 STM——连续多字存储指令	347
14.128 STOP——进入低功耗暂停模式指令	349
14.129 STQ——连续四字存储指令	350
14.130 STR.B——寄存器移位寻址字节存储指令	352
14.131 STR.H——寄存器移位寻址半字存储指令	353
14.132 STR.W——寄存器移位寻址字存储指令	355
14.133 SUBC——无符号带借位减法指令	357
14.134 SUBI——无符号立即数减法指令	359
14.135 SUBI(SP)——无符号(堆栈指针)立即数减法指令	362
14.136 SUBU——无符号减法指令	363
14.137 SYNC——CPU 同步指令	365
14.138 TRAP——操作系统陷阱指令	366
14.139 TST——零测试指令	367
14.140 TSTNBZ——无字节等于零寄存器测试指令	369
14.141 WAIT——进入低功耗等待模式指令	371

14.142 XOR——按位异或指令	372
14.143 XORI——立即数按位异或指令	373
14.144 XSR——扩展右移指令	374
14.145 XTRB0——提取字节 0 并无符号扩展指令	376
14.146 XTRB1——提取字节 1 并无符号扩展指令	377
14.147 XTRB2——提取字节 2 并无符号扩展指令	378
14.148 XTRB3——提取字节 3 并无符号扩展指令	379
14.149 ZEXT——位提取并无符号扩展指令	380
14.150 ZEXTB——字节提取并无符号扩展指令	382
14.151 ZEXTH——半字提取并无符号扩展指令	383
<b>第十五章 附录 B DSP 指令集术语表</b>	<b>385</b>
15.1 PADD.8——8 位并行加法指令	386
15.2 PADD.16——16 位并行加法指令	387
15.3 PADD.(U/S)8.S——带饱和操作的 8 位并行（无/有）符号加法指令	388
15.4 PADD.(U/S)16.S——带饱和操作的 16 位并行（无/有）符号加法指令	390
15.5 ADD.(U/S)32.S——带饱和操作的 32 位（无/有）符号加法指令	392
15.6 PSUB.8——8 位并行减法指令	394
15.7 PSUB.16——16 位并行减法指令	395
15.8 PSUB.(U/S)8.S——带饱和操作的 8 位并行（无/有）符号减法指令	396
15.9 PSUB.(U/S)16.S——带饱和操作的 16 位并行（无/有）符号减法指令	398
15.10 SUB.(U/S)32.S——带饱和操作的 32 位（无/有）符号减法指令	400
15.11 PADDH.(U/S)8——8 位并行（无/有）符号加法求平均值指令	402
15.12 PADDH.(U/S)16——16 位并行（无/有）符号加法求平均值指令	403
15.13 ADDH.(U/S)32——32 位（无/有）符号加法求平均值指令	404
15.14 PSUBH.(U/S)8——8 位并行（无/有）符号减法求平均值指令	405
15.15 PSUBH.(U/S)16——16 位并行（无/有）符号减法求平均值指令	406
15.16 SUBH.(U/S)32——32 位（无/有）符号减法求平均值指令	407
15.17 PASX.16——16 位并行交叉加减指令	407
15.18 PSAX.16——16 位并行交叉减加指令	408
15.19 PASX.(U/S)16.S——带饱和操作的 16 位并行（无/有）符号交叉加减指令	409
15.20 PSAX.(U/S)16.S——带饱和操作的 16 位并行（无/有）符号交叉减加指令	411
15.21 PASXH.(U/S)16——16 位并行（无/有）符号交叉加减求平均值指令	412
15.22 PSAXH.(U/S)16——16 位并行（无/有）符号交叉减加求平均值指令	413
15.23 ADD.64——64 位加法指令	414
15.24 ADD.(U/S)64.S——带饱和操作的 64 位（无/有）符号加法指令	416
15.25 SUB.64——64 位减法指令	417
15.26 SUB.(U/S)64.S——带饱和操作的 64 位（无/有）符号减法指令	418
15.27 ASRI.S32.R——带舍入操作的立即数算术右移指令	419
15.28 ASR.S32.R——带舍入操作的算术右移指令	420
15.29 LSRI.U32.R——带舍入操作的立即数逻辑右移指令	421
15.30 LSR.U32.R——带舍入操作的逻辑右移指令	422

15.31 LSLI.(U/S)32.S——带饱和操作的立即数（无/有）符号逻辑左移指令	423
15.32 LSL.(U/S)32.S——带饱和操作的（无/有）符号逻辑左移指令	425
15.33 PASRI.S16——16位并行立即数算术右移指令	426
15.34 PASR.S16——16位并行算术右移指令	427
15.35 PASRI.S16.R——带舍入操作的16位并行立即数算术右移指令	428
15.36 PASR.S16.R——带舍入操作的16位并行算术右移指令	430
15.37 PLSRI.U16——16位并行立即数逻辑右移指令	431
15.38 PLSR.U16——16位并行逻辑右移指令	432
15.39 PLSRI.U16.R——带舍入操作的16位并行立即数逻辑右移指令	433
15.40 PLSR.U16.R——带舍入操作的16位并行逻辑右移指令	434
15.41 PLSLI.16——16位并行立即数逻辑左移指令	436
15.42 PLSL.16——16位并行逻辑左移指令	437
15.43 PLSLI.(U/S)16.S——带饱和操作的16位并行立即数（无/有）符号逻辑左移指令	438
15.44 PLSL.(U/S)16.S——带饱和操作的16位并行（无/有）符号逻辑左移指令	440
15.45 SEL——位选择指令	441
15.46 PCMPNE.8——8位并行比较不等指令	442
15.47 PCMPNE.16——16位并行比较不等指令	443
15.48 PCMPHS.(U/S)8——8位并行（无/有）符号大于等于比较指令	445
15.49 PCMPHS.(U/S)16——16位并行（无/有）符号大于等于比较指令	446
15.50 PCMPLT.(U/S)8——8位并行（无/有）符号小于比较选择指令	447
15.51 PCMPLT.(U/S)16——16位并行（无/有）符号小于比较指令	449
15.52 PMAX.(U/S)8——8位并行（无/有）符号取大值指令	450
15.53 PMAX.(U/S)16——16位并行（无/有）符号取大值指令	451
15.54 MAX.(U/S)32——32位（无/有）符号取大值指令	452
15.55 PMIN.(U/S)8——8位并行（无/有）符号取小值指令	453
15.56 PMIN.(U/S)16——16位并行（无/有）符号取小值指令	454
15.57 MIN.(U/S)32——32位（无/有）符号取小值指令	455
15.58 DEXTI——立即数字截取指令	456
15.59 EXT——字截取指令	457
15.60 PKG——立即数移位打包指令	458
15.61 PKGLL——低半字打包指令	459
15.62 PKGHH——高半字打包指令	460
15.63 PEXT.U8.E——8位并行无符号扩展指令	461
15.64 PEXT.S8.E——8位并行有符号扩展指令	463
15.65 PEXTX.U8.E——8位并行无符号交叉扩展指令	465
15.66 PEXTX.S8.E——8位并行有符号交叉扩展指令	467
15.67 NARL——低位截取拼装指令	468
15.68 NARH——高位截取拼接指令	469
15.69 NARLX——低位交叉截取拼接指令	470
15.70 NARHX——高位交叉截取拼接指令	471
15.71 CLIPI.(U/S)32——立即数（无/有）符号裁剪饱和指令	473
15.72 CLIP.(U/S)32——（无/有）符号裁剪饱和指令	475

15.73 PCLIP.(U/S)16——16 位并行立即数（无/有）符号裁剪饱和指令	477
15.74 PCLIP.(U/S)16——16 位并行（无/有）符号裁剪饱和指令	480
15.75 PABS.S8.S——带饱和操作的 8 位并行绝对值指令	482
15.76 PABS.S16.S——带饱和操作的 16 位并行绝对值指令	483
15.77 ABS.S32.S——带饱和操作的 32 位绝对值指令	484
15.78 PNEG.S8.S——带饱和操作的 8 位并行取反指令	485
15.79 PNEG.S16.S——带饱和操作的 16 位并行取反指令	486
15.80 NEG.S32.S——带饱和操作的取反指令	487
15.81 DUP.8——8 位操作数拷贝指令	488
15.82 DUP.16——16 位操作数拷贝指令	489
15.83 MULS.(U/S)32——32 位（无/有）符号乘累减指令	490
15.84 MULA.(U/S)32.S——带饱和操作的 32 位（无/有）符号乘累加指令	491
15.85 MULS.(U/S)32.S——带饱和操作的 32 位（无/有）符号乘累减指令	493
15.86 MUL.S32.H——32 位有符号乘法取高 32 位指令	494
15.87 MUL.S32.RH——带舍入操作的 32 位有符号乘法取高 32 位指令	495
15.88 RMUL.S32.H——32 位有符号小数乘法取高 32 位指令	496
15.89 RMUL.S32.RH——带舍入操作的 32 位有符号小数乘法取高 32 位指令	497
15.90 MULA.S32.HS——带饱和操作的 32 位有符号乘累加取高 32 位指令	499
15.91 MULS.S32.HS——带饱和操作的 32 位有符号乘累减取高 32 位指令	500
15.92 MULA.S32.RHS——带舍入和饱和操作的 32 位有符号乘累加取高 32 位指令	502
15.93 MULS.S32.RHS——带舍入和饱和操作的 32 位有符号乘累减取高 32 位指令	504
15.94 MULXL.S32——32 位有符号低半字不对齐乘指令	505
15.95 MULXL.S32.R——带舍入操作的 32 位有符号低半字不对齐乘指令	506
15.96 MULXH.S32——32 位有符号高半字不对齐乘指令	507
15.97 MULXH.S32.R——带舍入操作的 32 位有符号高半字不对齐乘指令	508
15.98 RMULXL.S32——32 位有符号低半字不对齐小数乘指令	509
15.99 RMULXL.S32.R——带舍入操作的 32 位有符号低半字不对齐小数乘指令	511
15.100 RMULXH.S32——32 位有符号高半字不对齐小数乘指令	513
15.101 RMULXH.S32.R——带舍入操作的 32 位有符号高半字不对齐小数乘指令	515
15.102 MULAXL.S32.S——带饱和操作的 32 位有符号低半字不对齐乘累加指令	516
15.103 MULAXL.S32.RS——带舍入和饱和操作的 32 位有符号低半字不对齐乘累加指令	518
15.104 MULAXH.S32.S——带饱和操作的 32 位有符号高半字不对齐乘累加指令	519
15.105 MULAXH.S32.RS——带舍入和饱和操作的 32 位有符号高半字不对齐乘累加指令	521
15.106 MULLL.S16——16 位有符号低半字乘法指令	522
15.107 MULHH.S16——16 位有符号高半字乘法指令	523
15.108 MULHL.S16——16 位有符号高低半字乘法指令	524
15.109 RMULLL.S16——16 位有符号低半字小数乘法指令	525
15.110 RMULHH.S16——16 位有符号高半字小数乘法指令	526
15.111 RMULHL.S16——16 位有符号高低半字小数乘法指令	528
15.112 MULAHH.S16.S——带饱和操作的 16 位有符号高半字乘累加指令	529
15.113 MULAHL.S16.S——带饱和操作的 16 位有符号高低半字乘累加指令	531
15.114 MULALL.S16.E——带扩展操作的 16 位有符号低半字乘累加指令	532

15.115 MULAHH.S16.E——带扩展操作的 16 位有符号高半字乘累加指令	533
15.116 MULAHL.S16.E——带扩展操作的 16 位有符号高低半字乘累加指令	534
15.117 PMUL.(U/S)16——16 位并行（无/有）符号乘法指令	535
15.118 PMULX.(U/S)16——16 位并行（无/有）符号交叉乘法指令	537
15.119 PRMUL.S16——16 位并行有符号小数乘法指令	538
15.120 PRMULX.S16——16 位并行有符号交叉小数乘法指令	541
15.121 PRMUL.S16.H——16 位并行有符号小数乘法取高 16 位指令	543
15.122 PRMUL.S16.RH——带舍入操作的 16 位并行有符号小数乘法取高 16 位指令	545
15.123 PRMULX.S16.H——16 位并行有符号交叉小数乘法取高 16 位指令	548
15.124 PRMULX.S16.RH——带舍入操作的 16 位并行有符号交叉小数乘法取高 16 位指令	550
15.125 MULCA.S16.S——带饱和操作的 16 位有符号乘链加指令	553
15.126 MULCAX.S16.S——带饱和操作的 16 位有符号交叉乘链加指令	554
15.127 MULCS.S16——16 位有符号乘链减指令	555
15.128 MULCSR.S16——16 位有符号反向乘链减指令	556
15.129 MULCSX.S16——16 位有符号交叉乘链减指令	557
15.130 MULACA.S16.S——带饱和操作的 16 位有符号乘链加累加指令	558
15.131 MULACAX.S16.S——带饱和操作的 16 位有符号交叉乘链加累加指令	559
15.132 MULACS.S16.S——带饱和操作的 16 位有符号乘链减累加指令	561
15.133 MULACSR.S16.S——带饱和操作的 16 位有符号反向乘链减累加指令	562
15.134 MULACSX.S16.S——带饱和操作的 16 位有符号交叉乘链减累加指令	564
15.135 MULSCA.S16.S——带饱和操作的 16 位有符号乘链加累减指令	566
15.136 MULSCAX.S16.S——带饱和操作的 16 位有符号交叉乘链加累减指令	567
15.137 MULACA.S16.E——带扩展操作的 16 位有符号乘链加累加指令	569
15.138 MULACAX.S16.E——带扩展操作的 16 位有符号交叉乘链加累加指令	570
15.139 MULACS.S16.E——带扩展操作的 16 位有符号乘链减累加指令	571
15.140 MULACSR.S16.E——带扩展操作的 16 位有符号反向乘链减累加指令	572
15.141 MULACSRX.S16.E——带扩展操作的 16 位有符号交叉乘链减累加指令	573
15.142 MULSCA.S16.E——带扩展操作的 16 位有符号乘链加累减指令	574
15.143 MULSCAX.S16.E——带扩展操作的 16 位有符号交叉乘链加累减指令	575
15.144 PSABSA.U8——8 位并行无符号减法绝对值链加指令	576
15.145 PSABSAA.U8——8 位并行有无号减法绝对值链加累加指令	577
15.146 DIVSL——有符号长除法指令	578
15.147 DIVUL——无符号长除法指令	579
15.148 MULACA.S8——8 位并行有符号乘链加累加指令	580
15.149 BLOOP——循环体加速指令	581
15.150 LDBI.W——地址自增的字加载指令	583
15.151 LDBI.H——地址自增的无符号半字加载指令	584
15.152 LDBI.HS——地址自增的有符号半字加载指令	584
15.153 LDBI.B——地址自增的无符号字节加载指令	585
15.154 LDBI.BS——地址自增的有符号字节加载指令	586
15.155 PLDBI.D——地址自增的双字加载指令	587
15.156 STBI.W——地址自增的字存储指令	589

15.157 STBI.H——地址自增的半字存储指令	589
15.158 STBI.B——地址自增的字节存储指令	590
15.159 LDBIR.W——寄存器地址自增的字加载指令	591
15.160 LDBIR.H——寄存器地址自增的无符号半字加载指令	592
15.161 LDBIR.HS——寄存器地址自增的有符号半字加载指令	593
15.162 LDBIR.B——寄存器地址自增的无符号字节加载指令	594
15.163 LDBIR.BS——寄存器地址自增的有符号字节加载指令	594
15.164 PLDBIR.D——寄存器地址自增的双字加载指令	595
15.165 STBIR.W——寄存器地址自增的字存储指令	597
15.166 STBIR.H——寄存器地址自增的半字存储指令	597
15.167 STBIR.B——寄存器地址自增的字节存储指令	598

# 第一章 概述

## 1.1 简介

E804 是面向控制领域的 32 位高能效嵌入式 CPU 核，具有低成本、低功耗、高代码密度等多种特点。E804 采用 16/32 位混合编码指令系统，设计了精简高效的 3 级流水线。

E804 提供多种可配置功能，包括硬件浮点单元、片上高速缓存、DSP 加速单元、可信防护技术、片上紧耦合 IP 等，用户可根据应用需要进行配置。此外，E804 提供多总线接口，支持系统总线、指令总线、数据总线的灵活配置。E804 针对内存拷贝的应用做了特殊的优化，可以获得极致的内存拷贝性能。此外，E804 针对中断响应做了特殊的加速，中断响应延时仅需 13 个周期。

## 1.2 特点

E804 的体系结构和编程模型的主要特点为：

- 精简指令集处理器架构 (RISC)；
- 32 位数据，16 位/32 位混合编码指令；
- 16 个 32 位通用寄存器；
- 3 级流水线；
- 按序发射、按序执行、按序退休；
- 可配置的多总线接口；
- 可配置的高速缓存；
- 可配置的硬件浮点单元；
- 可配置的 DSP 处理单元；
- 可配置的可信防护技术；
- 可配置的内存保护单元 (0-8)；
- 可配置的硬件乘法器，支持 1 个周期快速产生乘法结果；
- 可配置的紧耦合 IP，包括矢量中断控制器与计时器；

- 支持多种处理器时钟与系统时钟比；
- 中断响应延时仅为 13 个处理器周期。

E804 微体系结构的主要特点为：

- 静态分支预测；
- 支持硬件除法；
- 支持连续内存访问；
- 支持 big endian 和 little endian。

### 1.3 可配置选项

E804 可配置选项如 表 1.1 所示。

表 1.1: E804 可配置选项

可配置单元	配置选项	详细
硬件乘法器	无/有	若配置则 1 个周期产生乘法结果，否则要 1-32 周期完成。
内存保护单元	0 到 8 个表项	可以配置为 0-8 个表项，其中 0 表示不实现内存保护单元。
高速缓存器	无/2K/4K/8K /16K	可以配置为 2KB、4KB、8KB、16KB。
浮点单元	无/有	单精度浮点硬件处理。
DSP 加速单元	无/有	音频、电机等应用的硬件加速。当配置该技术时，处理器配置 32 个 32 位通用寄存器。
可信防护技术	无/有	配置该技术，结合中天微公司的 SoC 平台技术/系统软件，将提供系统的安全防护功能。
指令总线	固定包含	仅支持 AHB-Lite，直接输出 (Non-Flop-out) 方式。
数据总线	固定包含	仅支持 AHB-Lite，直接输出 (Non-Flop-out) 方式。
系统总线	兼容 AHB/ 兼容 AHB-Lite	可以配置为兼容 AHB 协议或者兼容 AHB-Lite 协议。 仅支持直接输出 (Non-Flop-out) 方式。
矢量中断控制器	无/INT8/INT16 /INT24/INT32	支持硬件中断的嵌套处理。支持 8/16/24/32 个中断源。
系统计时器	无/有	用于计时。

## 1.4 可测性设计

E804 支持扫描链测试 (SCAN) 和内建自测试 (BIST)。其中，扫描链测试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，内建自测试用于测试高速缓存是否存在制造错误。

E804 的扫描链数目可由客户指定。

## 1.5 可调试性设计

E804 使用 JTAG 标准（2 线）设计硬件调试接口。E804 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。具体详见[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下表所示。

### 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。
- 信号，位域，控制位的表示都使用一种通用的规则。

符号	功能
+	加
-	减
*	乘
/	除
>	大于
<	小于
=	等于
$\geq$	大于或等于
$\leq$	小于或等于
$\neq$	不等于
.	与
+	或
$\oplus$	异或
NOT	取反
:	连接
$\Rightarrow$	传输
$\Leftrightarrow$	交换
$\pm$	误差
0b0011	二进制数
0x0F	十六进制数

图 1.1: 符号列表

- 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\_reset\_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

E804 结构框图如 图 2.1 所示。

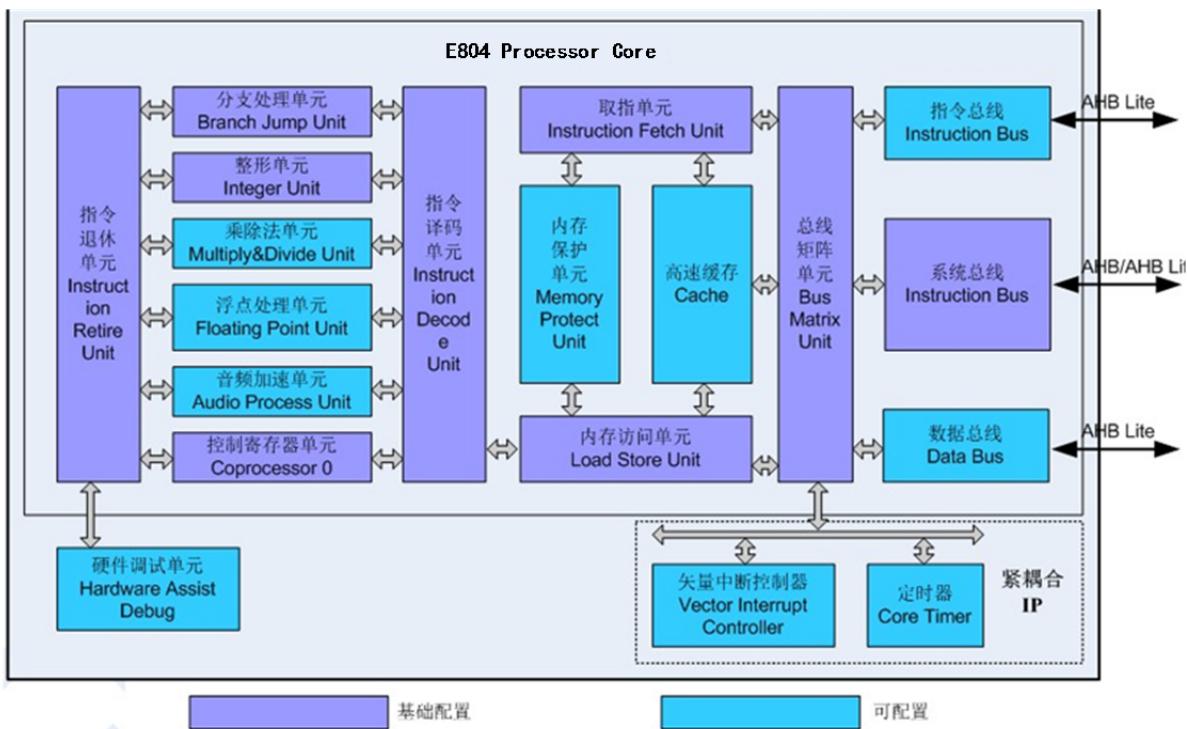


图 2.1: E804 结构框图

取指单元负责指令的访问与提取，每个时钟周期可取得 32 位数据，即每个周期可以提取 1 条 32 位指令或者 2 条 16 位指令。此外，取指单元对分支指令进行预测并对复杂指令进行拆分。

指令译码单元对指令进行译码，并访问通用寄存器以及完成指令的发射。

分支处理单元对分支预测指令进行检查并对寄存器跳转指令进行处理。整形单元负责 ALU 指令的执行。控制寄存器单元主对控制寄存器相关的指令（MTCR/MFCR）进行处理。分支指令、ALU 指令与控制寄存器指令的执行延时均为 1 个周期。

乘除法单元负责乘、乘累加以及除法指令的运算。乘法实现上，用户可配置慢速乘法与快速乘法两种方

式。其中，慢速乘法的成本开销小，但是需要 1-34 个周期产生乘法结果；快速乘法支持一个周期产生乘法结果，但是成本开销大。除法需要 3-35 个周期产生运算结果。

内存访问单元负责加载（Load）/存储指令（Store）的顺序执行，支持有符号/无符号的字节/半字/字访问。加载存储指令可连续执行，实现性能的最优化。

退休单元收集指令的执行结果并进行处理，完成结果的回写以及中断与异常的处理。此外，退休单元也负责与硬件调试单元的交互。

E804 实现了可配置的内存保护单元（MPU），用于保护敏感数据的存储器访问，支持 1-8 个表项可配置。用户通过对内存保护单元的设置可定义每个保护区的访问属性。

E804 实现了可配置的高速缓存单元，支持 2KB~32KB 可配置。高速缓存采用四路组相联的结构，支持写回、写通过两种工作模式。

E804 设计了多层次的总线接口，支持指令总线、数据总线和系统总线 3 种总线接口。指令/数据总线实现高性能的指令/数据访问，用户可通过指令/数据总线设计本地存储器子系统提高系统综合性能。指令/数据总线支持 AHB-Lite 接口，仅支持直接输出（Non-Flop-out，对接口时序有一定要求）方式。系统总线也仅支持直接输出方式，并且同时支持 AMBA2 AHB 协议和 AHB-Lite 协议可配置。CPU 时钟与系统外围总线时钟必须保持 1:1 的关系。

为了便于系统集成与开发，E804 设计了紧耦合 IP，包括矢量中断控制器和系统计时器，用户可根据应用需要进行配置。矢量中断控制器支持最多 32 个中断源，支持电平和脉冲两种中断方式，并且在硬件上支持中断嵌套。系统计时器提供了 1 个 24 位的循环递减计数器，可按照 CPU 时钟或者外部参考时钟计时，并且支持系统计时器产生中断。紧耦合 IP 的相关内容请参考[紧耦合 IP](#)。

E804 面向浮点应用，设计了可配置的浮点处理单元。浮点处理单元仅实现单精度浮点运算。

E804 面向音频、电机等应用，设计了可配置的 DSP 加速单元。DSP 加速单元实现了 DSP 加速指令子集。

E804 面向安全防护，设计了可配置的可信防护技术，配合本公司的 SoC 平台和系统软件，提供对系统的安全防护功能。具体请参考《可信防护技术手册》。

## 2.2 流水线介绍

E804 设计了精简高效的 3 级流水线，分别是取指、译码与执行。对于绝大多数整型指令（包括分支指令、整型运算类指令、加载存储指令以及 DSP 加速指令），均可在 3 级流水线内完成。对于部分的浮点指令，则需要额外的一个执行周期。E804 流水线如所示。

表 2.1: E804 流水线介绍

流水线名称	缩写	流水线作用
取指	IF	1. 发起取指令请求，处理返回的指令数据； 2. 指令预译码； 3. 静态分支预测； 4. 复杂指令拆分。
译码	ID	1. 指令译码； 2. 指令相关性分析； 3. 指令发射； 4. 执行分支跳转指令； 5. 计算加载存储指令的地址并发起访问请求。
执行	EX	1. 执行并完成整型类指令； 2. 执行浮点类指令； 3. 执行 DSP 加速类指令； 4. 完成加载存储指令； 5. 指令执行结果回写； 6. 指令退休； 7. 异常和中断处理。
执行 2	EX2	完成部分浮点类指令。

## 2.3 浮点处理单元

E804 面向浮点应用，设计了低成本、高能效的浮点处理单元，主要特征包括：

- 完全兼容 ANSI/IEEE Std 754 浮点标准（系统软件支持下）；
- 仅支持单精度浮点运算；
- 支持向零舍入、向正无穷舍入、向负无穷舍入和就近舍入四种舍入方式；
- 支持浮点异常的陷入与非陷入两种处理模式；
- 支持浮点异常的精确处理；
- 支持浮点硬件除法与开方；
- 16 个独立的单精度浮点寄存器；
- 单发射结构，每个周期处理一条浮点指令；

- 支持浮点算术指令的按序发射、按序执行、按序回写；
- 包含三条独立的执行流水线，分别是浮点 ALU、浮点乘法与浮点除法开方；
- 优化的执行延时技术，除浮点除法开方指令外，均可在 1-2 个时钟周期执行完毕。

## 2.4 DSP 扩展单元

E804 面向音频、电机等应用，设计了低成本、高能效的 DSP 处理单元。其主要设计特点包括：

- 基本涵盖现有 ARM/Andes DSP 指令子集定义的所有操作；
- 对短循环的极致加速技术；
- 数组/矩阵的数据预取技术。

主要特征包括：

- DSP 指令子集操作通用寄存器并将其扩充为 32 个；
- 支持的运算类型包括乘、乘累加、移位、加、减、数据封装、比较、存储访问等；
- 支持单指令多数据操作，单指令多数据操作支持的数据类型为字/半字，支持的运算类型为乘、乘累加、移位、加、减；
- 最大运算宽度为 64 位，支持单指令最多访问 4 个通用寄存器，回写 2 个通用寄存器；
- 部分运算支持舍入和非舍入两种取值方式；
- 部分运算支持饱和和非饱和两种取值方式。
- 部分运算支持整数和小数两种操作方式；
- 单发射结构，每个周期处理一条 DSP 扩展指令；
- 支持 DSP 扩展指令的按序发射、按序执行、按序回写；
- 优化的执行延时技术，所有指令均可在 1 个时钟周期完成。

## 2.5 可信防护技术

E804 面向安全领域，设计了可信防护技术，可用于系统的安全防护，主要特征包括：

- 基于同一物理处理器核，虚拟化出两个世界，分别为安全世界和非安全世界；
- 支持软件形式对存储器和 I/O 空间进行两个世界的空间划分；
- 支持可信中断；
- 支持可信调试；
- 支持可信引导；

更多信息请参考《可信防护技术手册》。

## 2.6 紧耦合 IP 架构

为了提高 E804 的系统集成度，方便用户集成与开发，E804 实现了一系列与 CPU 核关系密切的系统 IP，这些 IP 统称为紧耦合 IP (Titly Coupled IP, TCIP)。E804 的紧耦合 IP 包括系统计时器 CoreTim、矢量中断控制器 VIC。

矢量中断控制器的主要特征包括：

- 中断数量硬件可配置，支持 1/2/4/8/16/32 个中断源；
- 中断优先级软件可定义，可定义 4 个级别优先级；
- 支持硬件中断嵌套；
- 支持电平和脉冲两种中断源信号。

系统计时器的主要特征包括：

- 1 个 24 位的计数器；
- 支持输入时钟可选择，可以选择 CPU 时钟或外部输入时钟；
- 支持中断产生；

更多信息请参考[紧耦合 IP](#)。

# 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

E804 定义了两种处理器工作模式：普通用户模式和超级用户模式。两种工作模式对应不同的操作权限，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。

程序根据其权限来访问寄存器。普通用户程序只允许访问那些指定给普通用户模式的寄存器；工作在超级用户模式下的系统软件则可以访问所有的寄存器，并使用控制寄存器进行超级用户操作。通过对寄存器访问权限的管理，可以避免用户程序接触特权信息，操作系统则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。

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。特权指令包括 STOP, DOZE, WAIT, MFCR, MTCR, PSRSET, PSRCLR, RTE。

E804 设计了内存保护单元。操作系统可通过对内存保护单元的设置对内存的访问进行管理与控制。程序根据其权限来访问内存空间，普通用户程序只允许访问那些对普通用户模式开放的内存空间。

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，S 位被清零 0 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。在异常处理时，处理器把模式从普通用户模式切换到超级用户模式，并把当前 PSR 的值存放在异常保留处理器状态寄存器 (EPSR) 中，然后在 PSR 中设置 S 位，强制处理器进入超级用户模式。在异常处理完毕后，为返回到以前的工作模式，系统函数执行 RTE(从异常返回)，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 16 个 32 位的通用寄存器、32 位程序计数器 (PC)、条件/进位位 (C) 以及其它寄存器 (由配置决定)。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处理器控制和状态信息的 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，一个保存中断向量表的基址的寄存器 VBR 以及其他相关控制寄存器 (由配置决定)。

## 3.2 通用寄存器

E804 基础配置包括 18 个 32 位的通用寄存器：R15~R0, R14', R28，在配置 DSP 扩展单元的情况下通用寄存器的个数扩充为 33 个：R31~R0, R14'。

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。
R7	不确定。

下页继续

表 3.1 – 续上页

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 SP)	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	不确定。

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。当配置了 DSP 扩展单元时，基础指令集和 DSP 指令子集可以同时操作通用寄存器 R31~R0。

E804 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的 R14 (User SP)；在超级用户模式下，系统软件不仅可以访问系统的 R14 (Spv SP)，还可以访问普通用户的 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 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 系统控制寄存器

系统编程员用超级用户模式来设置系统操作功能, I/O 控制, 以及其他受限的操作。

超级用户模式下可操作的系统控制寄存器如下所示:

- 处理器状态寄存器 (PSR);
- 向量基址寄存器 (VBR);
- 异常保留程序计数器 (EPC);
- 异常保留处理器状态寄存器 (EPSR);
- 产品序号寄存器 (CPUID);
- 高速缓存配置寄存器 (CCR);
- 可高缓和访问权限配置寄存器 (CAPR)\*;
- 保护区控制寄存器 (PACR)\*;
- 保护区选择寄存器 (PRSR)\*;
- 隐式操作寄存器 (HINT)。

**注解:** \* 仅在特定配置时有效。

表 3.2: 超级用户编程模型控制寄存器

名称	类型	控制寄存器编号/地址	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。
EPC	读/写	CR<4,0>	异常保留程序计数器。
CPUID	读	CR<13,0>	产品序号寄存器。
CCR	读/写	CR<18,0>	高速缓存配置寄存器。
CAPR	读/写	CR<19,0>	可高缓和访问权限配置寄存器。
PACR	读/写	CR<20,0>	保护区控制寄存器。
PRSR	读/写	CR<21,0>	保护区选择寄存器。
CHR	读/写	CR<31,0>	隐式操作寄存器。
R14(User SP)	读/写	CR<14,1>	堆栈指针寄存器。

#### 3.3.1 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)。它们同样也指定了中断申请是否有效。

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

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 位在处理器重启和进入异常处理时由硬件置 1。

#### VEC[7:0]-异常向量值:

当异常出现时，VEC[7:0] 用来计算异常服务程序入口地址，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，E804 都将会进入跟踪异常服务程序；在跳转跟踪模式下，当碰到含有跳转（不管是跳转还是不跳转）的指令执行完，E804 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。下表所示为 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。

#### MM-不对齐异常掩盖位:

- 当 MM 为 0 时，加载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，加载或存储的地址不对齐异常将会被掩盖，数据访问请求被拆分为多个对齐操作；该位会被 reset 清零。

#### EE-异常有效控制位:

- 当 EE 为 0 时，异常无效，此时除了普通中断之外的任何异常一旦发生，都会被 E804 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-指令内中断控制位:

- 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 \* 的指令执行时被响应；
- 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；
- 当 IE 为 1 时，中断有效。

该位会被 reset 清零，也在进入异常服务程序时被清零。

**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 之后不确定。

**注解：** 部分多周期指令包括 LDM、STM、PUSH、POP、IPUSH、IPOP、LDQ32、STQ32，可以被中断而不等它们完成，从而缩短中断响应延时。多周期指令 NIE 不可响应中断，NIR 只在指令执行的末尾响应中断，不能被 PSR (IC) 位打断。

### 3.3.2 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 PSRSET，PSRCLR，RTE，MTCR 指令被修改，这些修改的实现有四个方面：

1. 异常响应和异常处理更新 PSR：

更新 PSR 是异常响应中的一部分，它将更新 PSR 中 S, TM, VEC, IE 以及 EE 位。

2. RTE 指令更新 PSR：

更新 PSR 作为 RTE 指令执行的一部分，会对 PSR 中的所有位都改动。

3. 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 MTCR 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。

4. 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。

### 3.3.3 向量基址寄存器 (VBR, CR<1,0>)

VBR 寄存器用来保存异常入口地址表的基址。VBR 的复位值为 0X00000000。

31		10   9	0
VECTOR BASE			Reserved
0			0

图 3.3: 基址向量寄存器

### 3.3.4 异常保留寄存器 (CR<2,0>, CR<4,0>)

EPSR, EPC, 寄存器在遇到异常情况时被用来保存当前处理器的状态。更详细的信息请参考[异常处理](#)。

### 3.3.5 产品序号寄存器 (CPUIDR, CR<13,0>)

该寄存器用于记录 CPU 的信息，包括 CPU 的型号、配置情况等。产品序号寄存器是只读的，其复位值由产品本身决定。

### 3.3.6 隐式操作寄存器 (CHR, CR<31,0>)

处理器隐式操作寄存器是处理器一些隐式操作的集合，包括软复位操作以及处理器某些加速功能的使能。

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

图 3.4: 隐式操作寄存器

#### Reset\_value-软复位使能域:

当 Reset\_Value 被写入 16' hABCD 时，将触发处理器复位操作，处理器自行复位，并通过处理器引脚 sysio\_pad\_srst 指示处理器发生软件复位，sysio\_pad\_srst 维持一个系统时钟周期。

#### HS\_EXP-硬件压栈异常指示位:

当处理器支持可信执行技术时，指示可信世界的硬件压栈操作出现异常。

#### IAE-中断响应加速控制位:

- 当 IAE 为 0 时，中断响应指令 NIE/IPUSH 将顺序执行；
- 当 IAE 为 1 时，中断响应指令 NIE/IPUSH 将投机执行，加快中断响应速度。

该位会被 reset 清零。

#### RPE-函数返回加速控制位:

- 当 RPE 为 0 时，函数返回指令 RTS 将顺序执行；
- 当 RPE 为 1 时，函数返回指令 RTS 将投机执行，加快函数返回。

该位会被 reset 置 1。

#### IPE-指令预取加速控制位:

- 当 IPE 为 0 时，指令预取加速功能不使能；
- 当 IPE 为 1 时，指令预取加速功能使能，加快指令预取效率。

该位会被 reset 清零。

#### BE-总线 Burst 传输控制位:

- 当 BE 为 0 时，总线传输类型只支持 SINGLE；
- 当 BE 为 1 时，系统总线支持 Burst 传输类型。

该位会被 reset 清零。

### 3.3.7 其它控制寄存器

E804 的其它控制寄存器还包括：

- 高速缓存配置寄存器 (CCR)\*；
- 可高缓和访问权限配置寄存器 (CAPR)\*；
- 保护区控制寄存器 (PACR)\*；
- 保护区选择寄存器 (PRSR)\*；

其中，高速缓存配置寄存器 (CCR)、可高缓和访问权限配置寄存器 (CAPR)\*、保护区控制寄存器 (PACR)\*、保护区选择寄存器 (PRSR)\* 是和内存保护单元设置相关的控制寄存器，在 CPU 配置内存保护单元时有效，具体的控制寄存器定义请参考[内存保护](#)。

### 3.3.8 普通用户模式通用寄存器 14 (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>。

### 3.3.9 中断指针寄存器 (R14(Int\_SP), CR<15,1>)

E804 可选择配置中断指针 (Int\_SP) 用于在不同线程下共享中断堆栈空间，以节省堆栈总开销。如配置且 ISE 使能，则仅在中断（不含 tspending）中，即向量号大于等于 32 时使用该指针，硬件压栈及其余时刻均使用原指针。

在非中断的超级用户态下，该寄存器映射为控制寄存器 CR<15,1>，即超级用户可通过访问 CR<15,1> 访问中断堆栈指针寄存器。

### 3.4 数据大小端

数据大小端是相对于存储器数据存储的格式提出的，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。

E804 支持小端格式和 V1 版本大端（CSKY CPU 设计了 V1 和 V2 两个版本的大小端）格式，且不支持非对齐访问。在小端模式下，无论存取指令的访问大小为字节/半字/字，始终以最低位字节数据对应于地址的最低位；在 V1 版本大端模式下，则需要根据存取指令的访问大小（字节/半字/字）确定数据。

表 3.4: 小端/大端存储访问模式

访问大小	访问地址	小端	V1 版本大端
字	A	D3D2D1D0	D3D2D1D0
半字	A	D1D0	D3D2
半字	A+2	D3D2	D1D0
字节	A	D0	D3
字节	A+1	D1	D2
字节	A+2	D2	D1
字节	A+3	D3	D0

### 3.5 数据对齐访问

数据不支持未对齐访问，当 CPU 检测到未对齐访问发生时，会进入未对齐访问异常，详情参考[未对齐访问异常（向量偏移 0X4）](#)。

E804 支持通过配置处理器状态寄存器 PSR 中的 MM 位，来屏蔽未对齐访问异常，详情参考[处理器状态寄存器（PSR, CR<0,0>）](#)。

### 3.6 系统地址映射

为了方便系统集成与开发，E804 对 4GB 的内存空间进行了地址划分与功能指定。总线矩阵单元对内存访问（指令或数据访问）地址进行仲裁并分发到不同的总线上。E804 推荐的系统地址划分及功能如下表所示。为了让 SOC 设计更加灵活，芯片厂商也可以自己定义总线址空间的划分，通过配置 pad\_bmu\_iahbl\_base，pad\_bmu\_iahbl\_mask，pad\_bmu\_dahbl\_base，pad\_bmu\_dahbl\_mask 来达到目的。具体配置方法参见[总线矩阵与总线接口](#)。

表 3.5: 地址划分图

名称	内存地址空间	功能
指令总线	0x00000000~0x1FFFFFFF	存放指令
数据总线	0x20000000~0x3FFFFFFF	存放数据
系统总线	0x40000000~0xDFFFFFFF	功能由系统开发者定义 可以存放指令、数据以及系统 IP
紧耦合 IP 总线	0xE0000000~0XEFFFFFFF	紧耦合 IP 的访问地址空间
系统总线	0xF0000000~0xFFFFFFFF	功能由系统开发者定义 可以存放指令、数据以及系统 IP

E804 的总线矩阵单元只根据内存访问地址进行仲裁，而不关心内存访问类型（指令访问/数据访问/紧耦合 IP 访问）。无论取指请求还是取数据请求都可以访问任一总线及存储器空间。编程者必须保证内存访问地址的正确性，以确保成功访问目标存储器。譬如：在配置了指令总线和数据总线的 E804 中，如果指令访问地址落在了 [0x20000000,0x40000000]，总线矩阵单元将把指令访问请求分发到数据总线上，并从数据总线存储器中访问。

E804 具有高度的可配置型，用户可以根据应用选择实现指令总线、数据总线以及紧耦合 IP 总线的任意一个或多个，总线的配置方式对内存访问的影响如下表所示。

表 3.6: 地址划分图

名称	可配置性	配置情况对内存访问的影响	
指令总线	固定包含	位于 [0x00000000-0x1FFFFFFF] 的内存访问将分发到指令总线上	
数据总线	固定包含	位于 [0x20000000-0x3FFFFFFF] 的内存访问将分发到数据总线上	
系统总线	固定包含	对系统总线空间的访问均被分发到系统总线上	
紧耦合 IP 总线	可配置	实现	位于 [0xE000 0000-0XEFFFFFFF] 的内存访问将分发到紧耦合 IP 总线上
		不实现	位于 [0xE0000000-0XEFFFF FFF] 的内存访问将分发到系统总线上
系统总线	固定包含	对系统总线空间的访问均被分发到系统总线上	

以上访问只会发生在不可高缓的区域或者高速缓存缺失的情况下。

### 3.7 内存访问顺序

E804 设计了多总线接口，系统的集成者可在总线上外接不同的存储器。由于不同总线上的存储器设备的访问延时不同，为了便于用户开发，E804 在硬件设计上保证了内存访问指令严格按照汇编指令的顺序依次完成，避免了用户以软件的方式保证内存访问的顺序。

譬如，如下两条指令序列，Ins A 访问系统总线的存储器，Ins B 访问数据总线的存储器。假设系统总线存储器的访问延时远大于数据总线存储器，为了保证指令按照程序的顺序完成，硬件保证 Ins A 指令执行完毕，才允许 Ins B 指令执行。

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

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

# 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。这些事件包括外部中断，指令执行错误和系统调用请求等等。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、特权异常和指令跟踪，TRAP 和 BKPT 指令正常执行时也会产生异常。而且，非法指令、LD 和 ST 访问的地址没有对齐还有用户模式下执行特权指令都会产生异常。异常处理利用异常向量表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 在指令退休时响应中断，并保存退出异常处理时下一条被执行的指令的地址。即使异常在取指或者译码阶段被识别，异常也要在相应的指令退休时才会被处理。E804 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，处理器保存 PSR 和 PC 到影子寄存器（EPSR 和 EPC）中。

第二步，将 PSR 中的超级用户模式设置位 S 位置 1（不管发生异常时处理器处于哪种运行模式），使处理器进入超级用户模式。

第三步，将 PSR 中的异常向量号 VEC 域更新为当前发生的异常向量号，标识异常类别以及支持共享异常服务的情况。

第四步，将 PSR 中的异常使能位 EE 位清零，禁止异常响应。在 EE 为零时发生的任何异常（除了普通中断），处理器都将其作为不可恢复错误异常处理。不可恢复的错误异常发生时，EPSR 和 EPC 也会被更新。

第五步，将 PSR 中的中断使能位 IE 位清零，禁止响应中断。

以上第二步、第三步和第四步，同时发生。

第六步，处理器首先根据 PSR 中的异常向量号计算得到异常入口地址，然后用该地址获得异常服务程序的第一条指令的地址。将异常向量乘以 4 后加上异常向量基准地址（存在向量基准地址寄存器 VBR 中，当 VBR 不存在时该值恒为零）即得到异常入口地址，以该异常入口地址从存储器中读取一个字，并将该字的 [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）。对于向量中断，异常向量由外部的中断控制器提供；对于其它的异常，处理器根据内部逻辑决定异常向量。

最后一步，处理器从异常服务程序的第一条指令处开始执行并将 CPU 的控制权转交给异常服务程序，开始异常的处理。

所有的异常向量存放在超级用户允许访问的地址空间上。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。

E804 支持最大 1024 个字节的向量表，即支持 256 个异常向量（详见 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 个向量保留，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。
9	024	Idly 异常。
10	028	普通中断。（自动向量）
11 - 15	02C - 03C	保留。
16 - 19	040 - 04C	陷阱指令异常（TRAP # 0-3）。
20 - 21	050 - 054	保留。
22	058	TSPEND 中断
23-29	05C - 074	保留
30	078	浮点运算异常
31	07C	保留。
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 E804 内部产生的异常。E804 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；
- Idly 异常；
- 普通中断；
- 陷阱指令异常；
- TSPEND 中断。

### 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) 清零以禁止中断响应。同时，VBR (向量基准寄存器) 也被清零，异常向量表的基准地址就是 0X00000000，CPU 从异常向量表中以偏移地址 0X0 为偏移地址读取异常服务程序入口地址，并把它装载到程序计数器 (PC)。

### 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。通过设置 PSR (MM)，这个异常可以被屏蔽，处理器会忽略对数据的对齐检查，而访问小于这个未对齐地址又最接近它的地址边界上。EPC 指向试图进行未对齐访问的指令。

未对齐访问异常只发生在数据访问上。

#### 4.2.3 访问错误异常（向量偏移 0X8）

如果总线访问导致了一个错误的回复，就意味着发生了访问错误异常。访问内存保护的区域出现访问错误时，也发生访问错误异常。EPC 指向发起错误访问的指令。

#### 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、RTE、STOP、WAIT、DOZE。

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，E804 对每条指令或对改变控制流指令的进行跟踪。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（包括 POP）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。

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，TRAP，STOP，WAIT，DOZE 和 BKPT。如果 EPSR (TP) 有效，跟踪异常作为 RTE 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。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 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，响应中断时这条指令已经退休，为了避免中断处理完之后跟踪异常丢失这种情况，中断处理完之后需要立即处理跟踪异常，这种情况下跟踪异常优先级是最高的仅次于重启。

#### 4.2.8 断点异常（向量偏移 0X1C）

E804 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，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) 为零时默认排除了异常事件发生的可能，在这种情况下如果 CPU 发生异常，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 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。

#### 4.2.12 TSPEND 中断（向量偏移 0X58）

当配置矢量中断控制器时，通过软件设置 VIC\_TSPEND 寄存器产生 TSPEND 中断。具体请参考紧耦合 IP。

#### 4.2.13 浮点异常（向量偏移 0X78）

浮点指令在运算过程中会发生浮点异常。

### 4.3 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。

E804 提供了紧耦合的矢量中断控制器。用户可以选用 E804 CPU 外加矢量中断控制器的方式进行系统集成，此时只需要将外部 IP 中断源信号接入矢量中断控制器即可。用户也可以选择 CPU 核，自己在系统

上集成中断控制器与其它 IP。E804 CPU 核提供了两个中断请求信号，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。

图 4.1 显示了和中断相关的处理器核的接口信号。在不配置 VIC 时，为了支持向量化的中断，在中断请求时，用 8 位的中断向量信号提供中断向量号，或设置 pad/intc\_cpu\_avec\_b 使用自动中断向量号。如果 PSR (IE) 或者 PSR (EE) 被清零了，pad/intc\_cpu\_int\_b 输入信号被屏蔽，处理器不响应中断，当 PSR (IE) 或者 PSR (EE) 同时有效时，CPU 响应中断。此时，如果 pad/intc\_cpu\_avec\_b 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 pad/intc\_cpu\_vec\_b[7:0] 上提供的向量号。cpu\_intc/pad\_int\_ack 指示 CPU 已经响应了中断。pad/intc\_cpu\_int\_b 和 pad/intc\_cpu\_avec\_b 都是低电平有效，pad/intc\_int\_ack 高电平有效。在配置有 VIC 的情况下，由控制信号生成相应的中断信号，这些信号的处理及响应与上文描述一致。

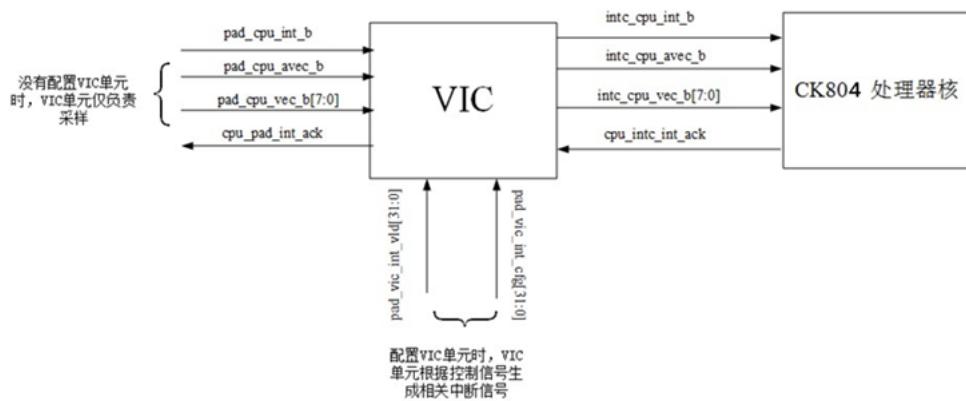


图 4.1: 中断接口信号

图 4.2 中，当中断向量号已经准备好时，拉低 pad\_cpu\_int\_b 中断信号线。

- 如果没有配置 VIC 单元，则 sys\_clk 采样 ( $T1=1\text{cyc}$ ) 后拉低 intc\_cpu\_int\_b 向 CPU 发送中断请求；
- 如果配置有 VIC 则需要用 cpu\_clk 经过两个周期进行采样和优先级的判断 ( $T1=2\text{cyc}$ )，之后才能向 CPU 发送中断请求。

该信号经 CPU 内部时钟 cpu\_clk 的上升沿采样后，CPU 内部收到中断，并根据向量信号取得中断向量。在外部系统均能在一个周期内响应的条件下，响应中断后，CPU 需要五个周期 ( $T2=5\text{cyc}$ ) 才能发出中断服务程序第一条指令的取指令请求，进入中断服务程序。在中断服务程序中，应该由软件清除外部中断源，即拉高中断有效信号，此信号亦需 CPU 及外部两个时钟依次采样后才会退出中断。

E804 可配置中断加速功能。在 CPU 响应中断后，取异常入口地址的同时开始投机执行 NIE，IPUSH 指令。如中断服务程序也首先执行这两条指令，则在外部系统均能在一个周期内响应的条件下能够节省六个周期的时间。如果发生投机预测错误，访问异常或调试请求，则中止中断加速功能。

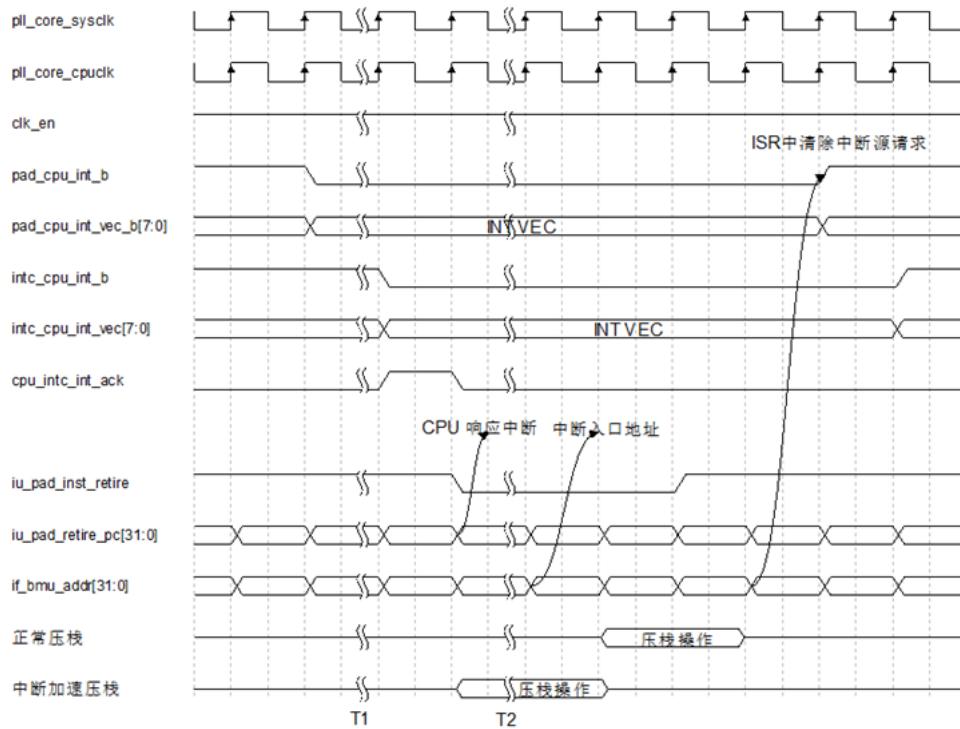


图 4.2: 处理器中断响应的时序图

如配置相应的中断控制器，则支持多个中断来源，可分别设置其对应的中断优先级并实现中断嵌套功能。更详细的中断机制及接口信号说明可参考紧耦合 IP。

#### 4.4 异常优先级

如表 4.2 所示，根据异常的特性和被处理的先后关系，E804 把优先级分为 10 级。在表 4.2 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 E804 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 4.2 中的优先级关系进行处理。

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，其余的异常可以依次重现。

表 4.2: 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 EPSR 的 TP=1，在 RTE 指令退休后，处理器处理待处理的跟踪异常。
3	IDLY 错误	在相关的指令退休后，处理器保存上下文并处理异常。
4	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	普通中断	如果 IC=0，中断在指令退休后被响应； 如果 IC=1，处理器允许中断在指令完成之前就被响应。
7.0 7.1	不可恢复错误异常 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 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。

## 4.5 异常返回

处理器通过执行 RTE 指令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 紧耦合 IP

## 5.1 紧耦合 IP 简介

为了提高 E804 的系统集成度，方便用户集成与使用，E804 实现了一系列与处理器关系密切的系统关键 IP，这些 IP 统称为紧耦合 IP (Tightly Coupled IP, TCIP)。E804 的紧耦合 IP 包括系统计时器 CoreTim、矢量中断控制器 VIC、片内高速缓存控制寄存器单元 CRU。这些紧耦合 IP 配合 E804，外加存储器等少量资源，便可以组成一款最小功能的 SoC 系统，提高了用户使用 E804 的便捷性，减少了 E804 的开发与应用成本。

E804 的紧耦合 IP 主要功能如 表 5.1，系统结构图如 图 5.1。

表 5.1: 紧耦合 IP 主要功能

IP 名	主要功能
系统计时器	完成系统的计时功能，可在低功耗时唤醒 CPU。
矢量中断控制器	完成中断的收集、仲裁、硬件嵌套以及与处理器的交互。
片内高速缓存控制寄存器单元	设置 E804 片内高速缓存，如开关 Cache，配置可高缓的区域等。

与传统 IP 不同，紧耦合 IP 通过专用的紧耦合 IP 总线接口与处理器相连，无需通过系统总线访问。其中，紧耦合 IP 总线接口直接与 E804 的总线互联单元 (Bus Matrix Unit, BMU) 相连，支持单个 CPU 时钟周期的紧耦合 IP 访问传输，不仅提高了紧耦合 IP 的访问效率，而且提高了系统集成效率。紧耦合 IP 与其它系统 IP 共享统一的内存地址空间，通过传输指令 (Load) 和存储指令 (Store) 进行寄存器访问和功能控制。紧耦合 IP 的内存地址分配如 表 5.2。

表 5.2: 紧耦合 IP 的内存地址分配

IP 名	内存地址空间
系统计时器	0xE000E010~0xE000E0FF
矢量中断控制器	0xE000E100~0xE000ECFF
片内高速缓存控制寄存器单元	0xE000F000~0xE000FFFF

紧耦合 IP 除了通过专用总线接口与内核发生通信，紧耦合 IP 还以直接相连的方式与处理器进行功能交互。其中，矢量中断控制器将仲裁之后的中断信息传送给处理器并接受处理器的中断响应信号；片内高速缓存位于内核总线互联单元 (BMU) 和指令总线接口单元 (I-AHBL) 之间。

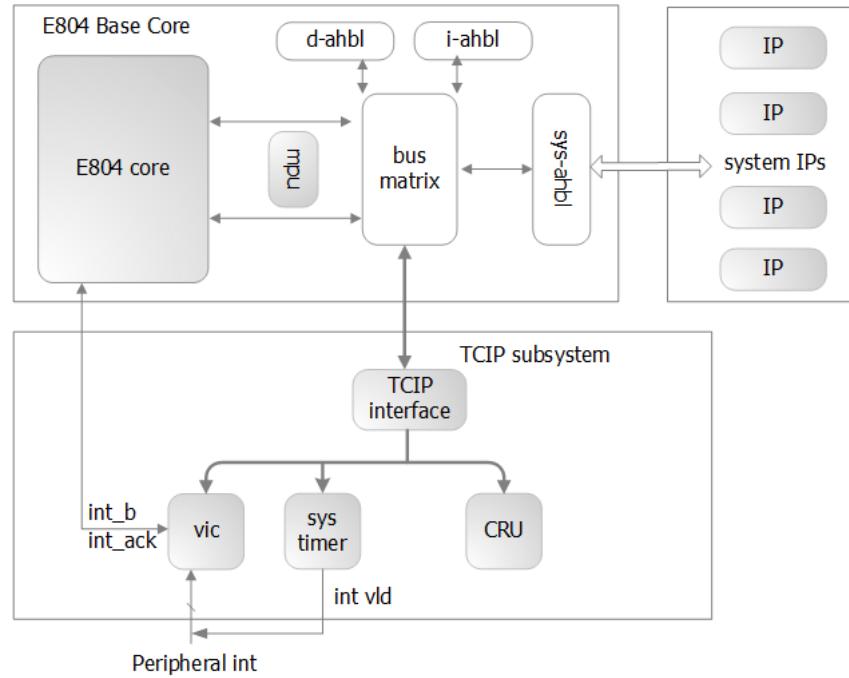


图 5.1: 紧耦合 IP 的系统结构图

紧耦合 IP 的所有控制寄存器都是 32-bit 的寄存器，因此只能通过以 word 为单位进行访问传输，任何以 half-word 或者 byte 为单位的访问都将造成不可预期的错误，并且紧耦合 IP 寄存器只支持小端格式。所有对紧耦合 IP 寄存器的访问均需在超级用户模式下才可进行，在普通用户模式下访问会产生访问错误异常。

## 5.2 系统计时器

### 5.2.1 简介

系统计时器是 E804 的一个可选模块，它主要用于计时。系统计时器提供了一个简单易用的 24 位循环递减的计数器，当系统计时器使能时，计数器开始工作。当计数器递减到 0 时，会向矢量中断控制器发起中断请求，申请获得处理器响应并处理系统计时器的事务。系统计时器的结构框图如 图 5.2 所示。

### 5.2.2 寄存器定义

系统计时器每一个寄存器宽度是 32 位，寄存器地址空间为如 表 5.3 所示。

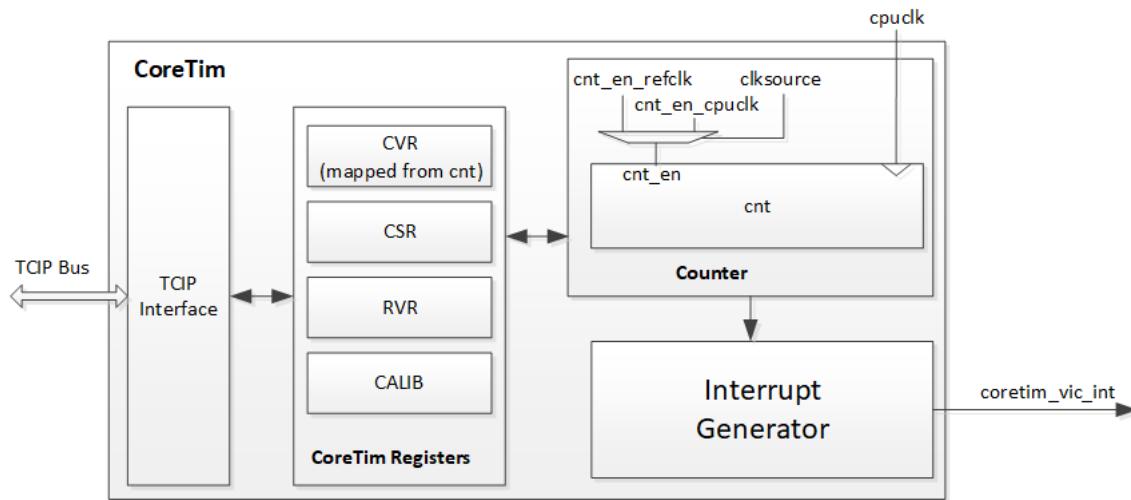


图 5.2: 系统计时器结构图

表 5.3: 系统计时器寄存器定义

地址	名称	类型	初始值	描述
0xE000E010	CORET_CSR	读/写	0x00000004	控制状态寄存器。
0xE000E014	CORET_RVR	读/写	-	回填值寄存器。
0xE000E018	CORET_CVR	读/写	-	当前值寄存器。
0xE000E01C	CORET_CALIB	只读	-	校准寄存器。
0xE000E020~0xE000E0FF	-	-	-	保留。

### 5.2.2.1 控制和状态寄存器 (CORET\_CSR)

CORET\_CSR 是系统计时器的控制和状态寄存器如 图 5.3 所示，CORET\_CSR 寄存器的位说明如 表 5.4 所示。

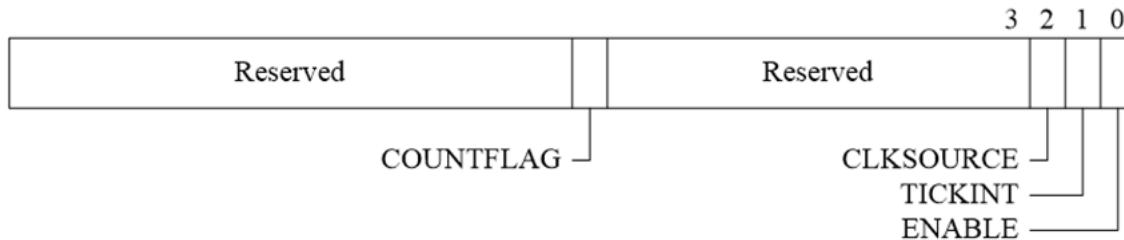


图 5.3: 系统计时器控制与状态寄存器

表 5.4: 系统计时器控制与状态寄存器域描述

位	类型	名称	描述
31:17	-	-	保留。
16	只读	COUNTFLAG	表示在上一次读此寄存器后计数器是否计数到 0: 0 计数器还没有计数到 0; 1 计数器已经计数到 0。 在计数器的值由 1 变到 0 时, COUNTFLAG 会被置位。 读 CSR 寄存器以及任何写 CVR 寄存器会使 COUNTFLAG 清零。
15:3	-	-	保留。
2	读/写	CLKSOURCE	表示系统计时器的时钟源: 0 用可选的外部参考时钟作为计数器的时钟源; 1 用内部时钟作为计数器的时钟源; 如果没有外部时钟, 读此位将返回 1, 写此位没有任何作用。 外部参考时钟频率必须小于或等于内部时钟频率的一半。
1	读/写	TICKINT	表示计数到 0 时是否会改变系统计时器的中断状态位: 0 计数到 0 时不会影响到系统计时器中断状态位; 1 计数到 0 会改变系统计时器的中断状态位。 写 CVR 寄存器会使计数器清零, 但这种方法不会导致系统计时器的中断状态位发生改变。
0	读/写	ENABLE	表示系统计时器的使能状态位: 0 计数器没有使能; 1 计数器使能。

### 5.2.2.2 回填值寄存器 (CORET\_RVR)

CORET\_RVR 寄存器用于在每一次计数循环开始时给 CORET\_CVR 寄存器赋值, CORET\_RVR 寄存器及其位说明如 图 5.4 , 表 5.5 所示。

31	24:23	0
Reserved	RELOAD	

图 5.4: 系统计时器回填值寄存器

表 5.5: 系统计时器回填值寄存器域描述

位	名称	描述
31:24	-	保留。
23:0	RELOAD	在计数器计数到 0 时, RELOAD 值会被赋给 CORET_CVR 寄存器。向 CORET_RVR 寄存器写 0 会使计数器在下一次循环时停止工作, 此后计数器的值将一直保持为 0。当使用外部参考时钟使能计数器后, 必须等到计数器正常计数开始后 (即 CORET_CVR 变为非 0 值时), 才可以将 CORET_RVR 置为 0 以让计数器在下一次循环时停止工作, 否则计数器无法开始第一次计数。

#### 如何计算 RELOAD 值:

RELOAD 的正常取值范围在 0x1-0x00FFFFFF 之间。RELOAD 可以被赋值为 0, 但它是没有任何作用的, 因为系统计时器中断以及 COUNTFLAG 位只有在计数值由 1 变为 0 时才起作用。要产生一个周期为 N 个计数时钟周期的计时器, RELOAD 的值需要被赋为 N-1。比如要在每 100 计数时钟周期时产生一个 CoreTim 中断, 需要给 RELOAD 赋值 99。

#### 5.2.2.3 当前值寄存器 (CORET\_CVR)

CORET\_CVR 包含了系统计时器的当前值, CORET\_CVR 寄存器及其位说明如 图 5.5 , 表 5.6 所示。

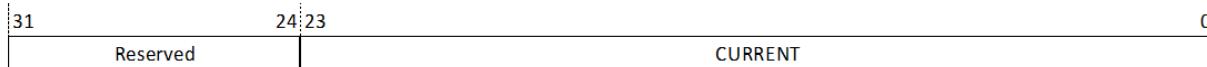


图 5.5: 系统计时器当前值寄存器

表 5.6: 系统计时器当前值寄存器域描述

位	名称	描述
31:24	-	保留。
23:0	CURRENT	它指示了计数器在被读取时的值。 写 CORET_CVR 寄存器会同时使此寄存器和 COUNTFLAG 状态位清零, 它会导致下一个时钟周期开始时, 系统计时器取出寄存器 CORET_RVR 里的值并赋给 CORET_CVR。注意写 CORET_CVR 不会导致系统计时器的中断状态位发生改变。 读 CORET_CVR 会返回访问寄存器时计数器的值。

#### 5.2.2.4 校准寄存器 (CORET\_CALIB)

CORET\_CALIB 寄存器描述了系统计时器的校准功能。它的复位值跟具体实现相关: 需要从设备提供商提供的文档里得到关于 CORET\_CALIB 位信息的含义, 以及 CORET\_CALIB 寄存器中的校验值 TENMS。

有了 TENMS 这个校准值，软件可以通过将这个值乘上一定的比例，从而得到其他不同的计数周期，当然这个计数周期必须在计数器的取值范围之内；CORET\_CALIB 寄存器及其位说明如图 5.6，表 5.7 所示。

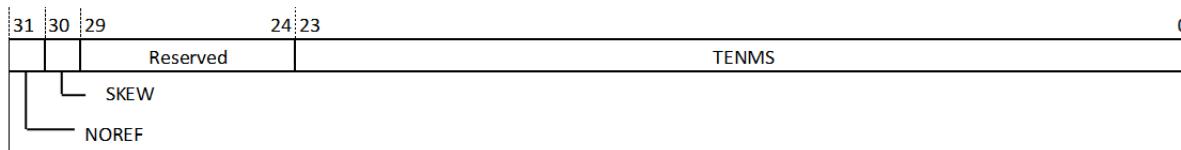


图 5.6: 系统计时器校准寄存器

表 5.7: 系统计时器校准寄存器域描述

位	名称	描述
31	NOREF	表示设备是否实现了外部参考时钟： 0 设备有外部参考时钟； 1 设备没有外部参考时钟。 当这位是 1 时，CORET_CSR 寄存器的 CLKSOURCE 位固定为 1，不能被改写。
30	SKEW	表示 10ms 校准值是否准确无误： 0 10ms 校准值准确无误； 1 10ms 校准值由于时钟频率的问题而有误差。
29:24	-	保留。
23:0	TENMS	用以表示 10ms 时间对应的回填值。根据 SKEW 具体值的不同，它可能表示完全准确的 10ms 值或者最接近 10ms 的值。 如果这个域的值是 0，表示校准值未知。这可能是因为参考时钟是一个未知的输入或者是动态变化的。

---

**注解：** 如果将 CORET\_RVR 设置为 0，那么 CoreTim 计数器将在下一回合停止工作，而不管计数器的使能位状态。

---

SYST\_CVR 寄存器的值在复位时是未知的。在使能 CoreTim 计数器之前，软件必须先将需要的计数值写入 CORET\_RVR 寄存器，然后再向 CORET\_CVR 写入任意值，后一操作会使 CORET\_CVR 的值清零。这样在使能计数器之后，计数器就可以读取 CORET\_RVR 回填寄存器里的值并从这个值开始向下计数，从而避免了从一个任意的值开始计数。

### 5.2.3 操作步骤

由于系统计时器中 CORET\_RVR 和 CORET\_CVR 两个寄存器没有复位值，在系统计时器工作之前，必须按照下列步骤进行操作：

1. 向 CORET\_RVR 寄存器里写入需要的回填值；
2. 向 CORET\_CVR 寄存器里写入任意值从而使它清零；

3. 操作 CORET\_CSR 寄存器，使能系统计时器。

## 5.3 矢量中断控制器

### 5.3.1 简介

矢量中断控制器 (VIC) 是一个与 E804 紧耦合的 IP 单元，用于中断的高效处理。矢量中断控制器最大可支持 128 个中断源 (IRQ[127:0])，每个中断源拥有独立软件可编程的中断优先级。矢量中断控制器收集来自不同中断源的中断请求，依据中断优先级对中断请求进行仲裁。最高优先级的中断将获得中断控制权并向处理器发出中断请求。当处理器响应了中断请求，处理器返回中断请求响应信号给 VIC；当处理器退出中断服务程序 (ISR)，处理器返回中断退出信号给 VIC。

矢量中断控制器支持中断嵌套。当处理器正在处理一个中断请求的同时来了一个更高优先级的中断请求，处理器将中断当前中断服务程序的处理，响应该更高优先级的中断请求。在更高优先级的中断请求处理结束时，CPU 返回被打断的中断服务程序继续执行。矢量中断控制器允许高优先级的中断请求抢占低优先级的中断请求，但不允许同级别或者低优先级的中断抢占，保证了中断响应的实时性。

矢量中断控制器的系统结构图如 图 5.7 所示。

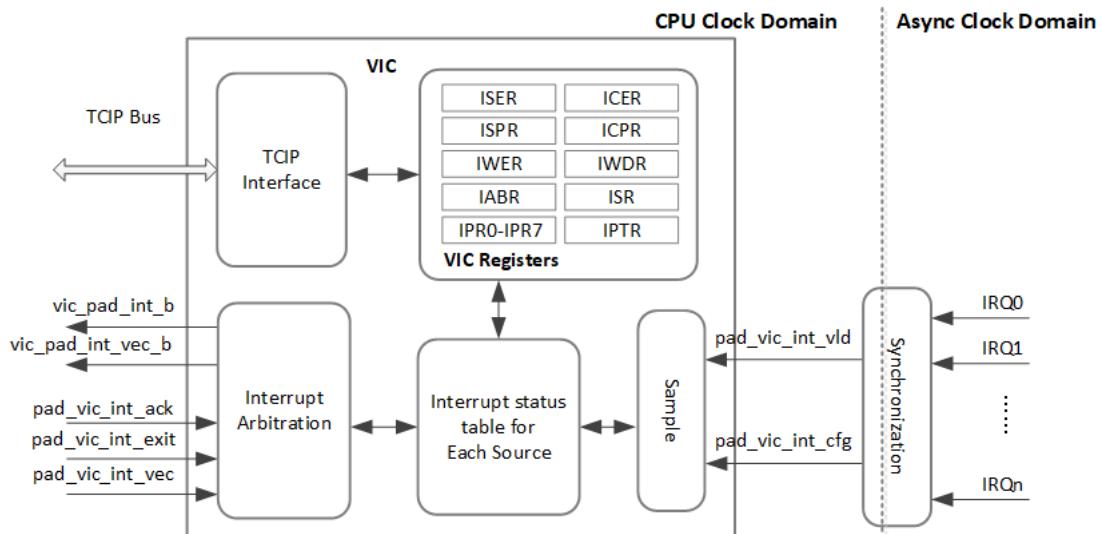


图 5.7: 矢量中断控制器系统结构图

矢量中断控制器支持以下功能：

- 中断数量硬件可配，支持 4、8、16、24、32、64、96 和 128；
- 软件通过 8 比特寄存器为每个中断配置优先级，在 E804 中若中断源数量小于或等于 32 个则仅使用 8 位中的最高 2 位有效位表征优先级高低，其余位始终保持为 0；若中断源数量为 64 个则可以使用 8 比特中的最高 3 位来表征 8 个优先级；若中断源数量为 96 或 128 个则可以使用 8 比特中的最高 4 位来表征 16 个优先级。优先级从低到高排列，0 级优先级为最高优先级，P (P=4, 8, 16) 级优先级为最低优先级；

- 支持电平和脉冲两种中断源信号；
- 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；
- 支持中断嵌套，CPU 在执行中断服务程序过程中，允许更高优先级的中断抢占。

### 5.3.2 寄存器定义

VIC 提供一组 32-bit 的寄存器，各个寄存器的地址空间如 表 5.8 所示。

表 5.8: 矢量中断控制器寄存器定义

地址	名称	类型	初始值	描述
0xE000E100	VIC_ISER0	读/写	0x00000000	中断使能设置寄存器 (0-31 号中断)
0xE000E104	VIC_ISER1	读/写	0x00000000	中断使能设置寄存器 (32-63 号中断)。
0xE000E108	VIC_ISER2	读/写	0x00000000	中断使能设置寄存器 (64-95 号中断)。
0xE000E10C	VIC_ISER3	读/写	0x00000000	中断使能设置寄存器 (96-127 号中断)。
0xE000E110~0xE000E13F	-	-	-	保留。
0xE000E140	VIC_IWER0	读/写	0x00000000	低功耗唤醒设置寄存器 (0-31 号中断)
0xE000E144	VIC_IWER1	读/写	0x00000000	低功耗唤醒设置寄存器 (32-63 号中断)。
0xE000E148	VIC_IWER2	读/写	0x00000000	低功耗唤醒设置寄存器 (64-95 号中断)。
0xE000E14C	VIC_IWER3	读/写	0x00000000	低功耗唤醒设置寄存器 (96-127 号中断)。
0xE000E150~0xE000E17F	-	-	-	保留。

下页继续

表 5.8 – 续上页

0xE000E180	VIC_ICER0	读/写	0x00000000	中断使能清除寄存器 (0-31 号中断)。
0xE000E184	VIC_ICER1	读/写	0x00000000	中断使能清除寄存器 (32-63 号中断)。
0xE000E188	VIC_ICER2	读/写	0x00000000	中断使能清除寄存器 (64-95 号中断)。
0xE000E18C	VIC_ICER3	读/写	0x00000000	中断使能清除寄存器 (96-127 号中断)。
0xE000E190~0xE000E1BF	-	-	-	保留。
0xE000E1C0	VIC_IWDR0	读/写	0x00000000	低功耗唤醒清除寄存器 (0-31 号中断)。
0xE000E1C4	VIC_IWDR1	读/写	0x00000000	低功耗唤醒清除寄存器 (32-63 号中断)。
0xE000E1C8	VIC_IWDR2	读/写	0x00000000	低功耗唤醒清除寄存器 (64-95 号中断)。
0xE000E1CC	VIC_IWDR3	读/写	0x00000000	低功耗唤醒清除寄存器 (96-127 号中断)。
0xE000E1D0~0xE000E1FF	-	-	-	保留。
0xE000E200	VIC_ISPR0	读/写	0x00000000	中断等待设置寄存器 (0-31 号中断)。
0xE000E204	VIC_ISPR1	读/写	0x00000000	中断等待设置寄存器。 (32-63 号中断)。
0xE000E208	VIC_ISPR2	读/写	0x00000000	中断等待设置寄存器 (64-95 号中断)。
0xE000E20C	VIC_ISPR3	读/写	0x00000000	中断等待设置寄存器 (96-127 号中断)。
0xE000E210~0xE000E27F	-	-	-	保留。
0xE000E280	VIC_ICPR0	读/写	0x00000000	中断等待清除寄存器 (0-31 号中断)。
0xE000E284	VIC_ICPR1	读/写	0x00000000	中断等待清除寄存器 (32-63 号中断)。
0xE000E288	VIC_ICPR2	读/写	0x00000000	中断等待清除寄存器 (64-95 号中断)。
0xE000E28C	VIC_ICPR3	读/写	0x00000000	中断等待清除寄存器 (96-127 号中断)。
0xE000E290~0xE000E2FF	-	-	-	保留。
0xE000E300	VIC_IABR0	读/写	0x00000000	中断响应状态寄存器 (0-31 号中断)。

下页继续

表 5.8 – 续上页

0xE000E304	VIC_IABR1	读/写	0x00000000	中断响应状态寄存器 (32-63 号中断)。
0xE000E308	VIC_IABR2	读/写	0x00000000	中断响应状态寄存器 (64-95 号中断)。
0xE000E30C	VIC_IABR3	读/写	0x00000000	中断响应状态寄存器 (96-127 号中断)。
0xE000E310~0xE000E3FF	-	-	-	保留。
0xE000E400~0xE000E47C	VIC_IPR0~ VIC_IPR31	读/写	0x00000000	中断优先级设置寄存器。
0xE000E480~0xE000EBFF	-	-	-	保留。
0xE000EC00	VIC_ISR	只读	0x00000000	中断状态寄存器。
0xE000EC04	VIC_IPTR	读/写	0x00000000	中断优先级阈值寄存器。
0xE000EC08	VIC_TSPE	读/写	0x00000000	Tspending 使能设置寄存器。
0xE000EC0C	VIC_TSABR	读/写	0x00000000	Tspending 响应状态寄存器。
0xE000EC10	VIC_TSPPR	读/写	0x00000000	Tspending 等待设置寄存器。
0xE000EC14~0xE000ECFF	-	-	-	保留。

### 5.3.2.1 中断使能设置寄存器 (VIC\_ISER)

VIC\_ISER 用于使能各个中断，并且反馈各个中断的使能状态。图 5.8 描述了 VIC\_ISER 的位分布，表 5.9 描述了 VIC\_ISER 的位定义。



图 5.8: 中断使能设置寄存器

表 5.9: 中断使能设置寄存器域定义

位	名称	描述
31:0	SETENA	设置使用，读取一个或者多个中断的使能状态。每一个位对应相同编号的中断源： <b>读操作：</b> 0 对应中断未使能； 1 对应中断已使能。 <b>写操作：</b> 0 无效； 1 使能对应中断。

如果一个处于等待状态的中断已使能，矢量中断控制器会根据其优先级激活该中断。如果一个中断未使能，该中断即使处于等待状态，矢量中断控制器也不会激活该中断。

### 5.3.2.2 中断低功耗唤醒设置寄存器 (VIC\_IWER)

VIC\_IWER 用于使能各个中断的低功耗唤醒功能，并且反馈各个中断低功耗唤醒的使能状态。

图 5.9 描述了 VIC\_IWER 的位分布，表 5.10 描述了 VIC\_IWER 的位定义。

31	SETENA	0
----	--------	---

图 5.9: 中断低功耗唤醒设置寄存器

表 5.10: 低功耗唤醒使能设置寄存器域定义

位	名称	描述
31:0	SETENA	<p>设置使用，读取一个或者多个中断低功耗唤醒的使能状态。每一个位对应相同编号的中断源：</p> <p><b>读操作：</b></p> <p><b>0:</b> 对应中断的低功耗唤醒功能未使能；  <b>1:</b> 对应中断的低功耗唤醒功能已使能。</p> <p><b>写操作：</b></p> <p><b>0:</b> 无效；  <b>1:</b> 使能对应中断的低功耗唤醒功能。</p>

如果一个中断的低功耗唤醒功能已使能且该中断处于等待状态，VIC 产生低功耗唤醒请求。如果一个中断的低功耗唤醒功能未使能，即使该中断处于等待状态，VIC 也不产生低功耗唤醒请求。

---

**注解：** 中断使能和中断唤醒使能分别控制中断事务和中断唤醒功能。当两者都设置时，一个处于等待状态的中断既产生中断请求又产生低功耗唤醒请求；当只有其中一个使能时，只激活对应的功能；当两者都没有使能时，即使该中断处于等待状态，也不会产生中断请求或低功耗唤醒请求。

---

### 5.3.2.3 中断使能清除寄存器 (VIC\_ICER)

VIC\_ICER 用于清除各个中断的使能，并且反馈各个中断的使能状态。图 5.10 描述了 VIC\_ICER 的位分布，表 5.11 描述了 VIC\_ICER 的位定义。

31	CLRENA	0
----	--------	---

图 5.10: 中断使能清除寄存器

表 5.11: 中断使能清除寄存器域描述

位	名称	描述
31:0	CLRENA	<p>清除使用，读取一个或者多个中断的使能状态。每一个位对应相同编号的中断源：</p> <p><b>读操作：</b></p> <p><b>0:</b> 对应中断未使能；  <b>1:</b> 对应中断已使能。</p> <p><b>写操作：</b></p> <p><b>0:</b> 无效；  <b>1:</b> 清除对应中断的使能。</p>

### 5.3.2.4 中断低功耗唤醒清除寄存器 (VIC\_IWDR)

VIC\_IWDR 用于清除各个中断的低功耗唤醒使能，并且反馈各个中断低功耗唤醒的使能状态。图 5.11 描述了 VIC\_IWDR 的位分布，表 5.12 描述了 VIC\_IWDR 的位定义。



图 5.11: 中断低功耗唤醒清除寄存器

表 5.12: 中断低功耗唤醒清除寄存器域描述

位	名称	描述
31:0	CLRENA	<p>清除使用，读取一个或者多个中断低功耗唤醒的使能状态。每一个位对应相同编号的中断源：</p> <p><b>读操作：</b></p> <p><b>0:</b> 对应中断的低功耗唤醒功能未使能；  <b>1:</b> 对应中断的低功耗唤醒功能已使能。</p> <p><b>写操作：</b></p> <p><b>0:</b> 无效；  <b>1:</b> 清除使能对应中断的低功耗唤醒功能。</p>

### 5.3.2.5 中断等待设置寄存器 (VIC\_ISPR)

VIC\_ISPR 表征设置各个中断到等待状态。描述了 VIC\_ISPR 的位分布，图 5.12 描述了 VIC\_ISPR 的位分布。表 5.13 描述了 VIC\_ISPR 的位定义。



图 5.12: 中断等待设置寄存器

表 5.13: 中断等待设置寄存器域描述

位	名称	描述
31:0	SETPEND	<p>更改一个或多个中断到等待状态。每一个位对应相同编号的中断源：</p> <p><b>读操作：</b></p> <p>0: 对应中断未处于等待状态。</p> <p>1: 对应中断处于等待状态。</p> <p><b>写操作：</b></p> <p>0: 无效。</p> <p>1: 改变对应中断到等待状态。</p>

### 5.3.2.6 中断等待清除寄存器 (VIC\_ICPR)

VIC\_ICPR 表征清除各个中断的等待状态。图 5.13 描述了 VIC\_ICPR 的位分布，表 5.14 描述了 VIC\_ICPR 的位定义。



图 5.13: 中断等待清除寄存器

表 5.14: 中断等待清除寄存器

位	名称	描述
31:0	CLRPEND	<p>清除一个或多个中断的等待状态。每一个位对应相同编号的中断源：</p> <p><b>读操作：</b></p> <p>0: 对应中断处于未等待状态；</p> <p>1: 对应中断处于等待状态。</p> <p><b>写操作：</b></p> <p>0: 无效；</p> <p>1: 清除对应中断的等待状态。</p>

### 5.3.2.7 中断响应状态寄存器 (VIC\_IABR)

VIC\_IABR 用于指示各个中断当前的 Active 状态，是一个供软件查询的寄存器，另外，软件可在初始化 VIC 时，将所有中断的 Active 状态清 0。图 5.14 描述了 VIC\_IABR 的位分布，表 5.15 描述了 VIC\_IABR 的位定义。

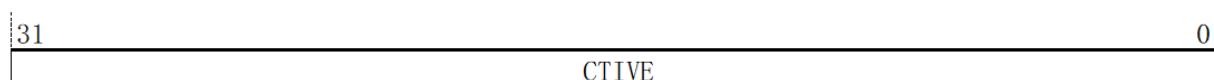


图 5.14: 中断响应状态寄存器

表 5.15: 中断响应状态寄存器域描述

位	名称	描述
31:0	Active	<p>查询位，指示该中断源是否已经被 CPU 响应但还没处理完。每一位对应相同编号的中断源。</p> <p><b>读操作：</b></p> <p>0：没有被 CPU 响应； 1：已经被 CPU 响应但还没处理完。</p> <p><b>写操作：</b></p> <p>0：清除中断的 Active 状态（软件不可对该寄存器写 1，否则会导致不可预期的错误）。</p>

### 5.3.2.8 中断优先级设置寄存器 (VIC\_IPR0 - VIC\_IPR31)

每个中断优先级设置寄存器提供 4 个中断源的优先级设置。根据应用的定义，每个中断源设置区域对应相应的中断源。对于硬件支持 128 个中断源的实现，寄存器从 IPR0 到 IPR31 如 图 5.15 所示，图 5.16 描述了中断优先级设置寄存器。

矢量中断控制器根据优先级号选择中断优先级，优先级号越小，优先级越高。如果优先级号相同，根据中断源号决定优先级顺序，号码越小，优先级越高。

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

图 5.15: 中断优先级设置寄存器整体分布

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

图 5.16: 中断优先级设置寄存器

表 5.16 和 表 5.17 描述了 VIC\_IPRn 的位定义。在这张表中，N = 4n，n 为 VIC\_IPRn 寄存器编号。如 VIC\_IPR2，n 为 2 则 N 为 8。

表 5.16: 中断优先级设置寄存器域描述

优先级阶数	位	名称	描述
2 比特 4 个优先级 (中断数小于等于 32 个)	31:30	PRI_N+3	中断号为 N+3 的优先级, 值越小优先级越高。
	29:24	-	保留。
	23:22	PRI_N+2	中断号为 N+2 的优先级, 值越小优先级越高。
	21:16	-	保留。
	15:14	PRI_N+1	中断号为 N+1 的优先级, 值越小优先级越高。
	13:8	-	保留。
	7:6	PRI_N	中断号为 N 的优先级, 值越小优先级越高。
	5:0	-	保留。
3 比特 8 个优先级 (中断数量为 64 个)	31:29	PRI_N+3	中断号为 N+3 的优先级, 值越小优先级越高。
	28:24	-	保留。
	23:21	PRI_N+2	中断号为 N+2 的优先级, 值越小优先级越高。
	20:16	-	保留。
	15:13	PRI_N+1	中断号为 N+1 的优先级, 值越小优先级越高。
	12:8	-	保留。
	7:5	PRI_N	中断号为 N 的优先级, 值越小优先级越高。
	4:0	-	保留。

表 5.17: 中断优先级设置寄存器域描述

优先级阶数	位	名称	描述
4 比特 16 个优先级 (中断数量为 96 或 128 个)	31:28	PRI_N+3	中断号为 N+3 的优先级, 值越小优先级越高。
	27:24	-	保留。
	23:20	PRI_N+2	中断号为 N+2 的优先级, 值越小优先级越高。
	19:16	-	保留。
	15:12	PRI_N+1	中断号为 N+1 的优先级, 值越小优先级越高。
	11:8	-	保留。
	7:4	PRI_N	中断号为 N 的优先级, 值越小优先级越高。
	3:0	-	保留。

### 5.3.2.9 中断状态寄存器 (VIC\_ISR)

VIC\_ISR 指示了当前 CPU 正在处理的中断向量号和处于等待的优先级最高的中断向量号, 该寄存器是一个供软件查询的只读寄存器。图 5.17 描述了 VIC\_ISR 的位分布, 表 5.18 描述了 VIC\_ISR 的位定义。

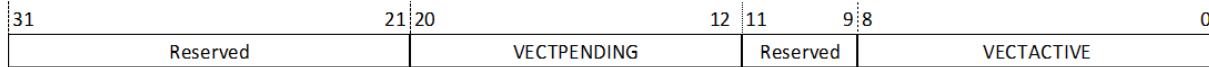


图 5.17: 中断状态寄存器

表 5.18: 中断状态寄存器域描述

位	名称	描述
31:21	Reserved	保留。
20:12	VECTPENDING	指示当前处于等待状态的优先级最高的中断向量号。
11:9	Reserved	保留。
8:0	VECTACTIVE	指示 CPU 当前正在处理的中断向量号。

### 5.3.2.10 中断优先级阈值寄存器 (VIC\_IPTR)

VIC\_IPTR 定义了当前处于等待状态的中断请求能够发起中断抢占的优先级临界值。处于等待状态的中断请求的优先级必须高于 VIC\_IPTR 定义的优先级阈值, 才能发起中断抢占请求。图 5.18 描述了 VIC\_IPTR 的位分布, 表 5.19 描述了 VIC\_IPTR 的位定义。

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

图 5.18: 中断优先级阈值寄存器

表 5.19: 中断优先级阈值寄存器域描述

位	名称	描述
31	EN	中断优先级阈值有效位： <b>0</b> : 中断抢占不需要优先级高于阈值； <b>1</b> : 中断抢占需要优先级高于阈值。
30:17	Reserved	保留。
16:8	VECTTHRESHOLD	指示优先级阈值对应的中断向量号。当 VIC 发现 CPU 从 VECTTHRESHOLD 对应的中断服务程序退出时，硬件自动清除中断优先级阈值有效位。
7:0	PRIOTHRESHOLD	指示中断抢占的优先级阈值。  注解：E804 中根据配置的中断数量决定的优先级个数可以设 [7:6],[7:5] 或 [7:4] 位来表征优先级阈值。

### 5.3.2.11 Tspend 中断使能设置寄存器 (VIC\_TSPEND)

VIC\_TSPEND 用于使能 tspend 中断，并且反馈 tspend 中断的使能状态。图 5.19 描述了 VIC\_TSPEND 的位分布，表 5.20 描述了 VIC\_TSPEND 的位定义。

31	1   0
Reserved	
SETENA	

图 5.19: Tspend 中断使能设置寄存器

表 5.20: Tspend 中断使能设置寄存器域定义

位	名称	描述
0:0	SETENA	设置使能，读取 tspend 中断的使能状态： <b>读操作：</b> <b>0</b> : 对应 tspend 中断未使能； <b>1</b> : 对应 tspend 中断已使能。 <b>写操作：</b> <b>0</b> : 无效； <b>1</b> : 使能 tspend 中断。

### 5.3.2.12 Tspend 中断响应状态寄存器 (VIC\_TSABR)

VIC\_TSABR 用于指示 tspend 中断当前的 Active 状态，是一个供软件查询的寄存器，另外，软件可在初始化 VIC 时，将所有 tspend 中断的 Active 状态清 0。图 5.20 描述了 VIC\_TSABR 的位分布，表 5.21 描述了 VIC\_TSABR 的位定义。



图 5.20: Tspend 中断响应状态寄存器

表 5.21: Tspend 中断响应状态寄存器域描述

位	名称	描述
0	Active	<p>查询位，指示 tspend 中断是否已经被 CPU 响应但还没处理完。该寄存器只有最低位有效。</p> <p><b>读操作</b> 0 没有被 CPU 响应； 1 已经被 CPU 响应但还没处理完。</p> <p><b>写操作</b> 0 清除 tspend 中断的 Active 状态。 (软件不可对该寄存器写 1，否则会导致不可预期的错误)</p>

### 5.3.2.13 Tspend 中断优先级设置寄存器 (VIC\_TSPR)

VIC\_TSPR 提供 tspend 中断的优先级设置，tspend 中断的优先级需要设置为最低。图 5.21 描述了 VIC\_TSPR 的位分布，表 5.22 描述了 VIC\_TSPR 的位定义。

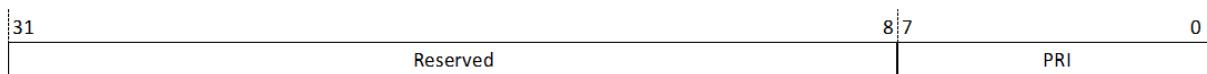


图 5.21: Tspend 中断优先级设置寄存器

表 5.22: Tspend 中断优先级设置寄存器域描述

位	名称	描述
2 比特 4 个优先级 (中断数小于等于 32 个)	31:8	- 保留。
	7:6	PRI 建议设置成最低优先级 2' b11。
	5:0	- 保留。
3 比特 8 个优先级 (中断数量为 64 个)	31:8	- 保留。
	7:5	PRI 建议设置成最低优先级 3' b111。
	5:0	- 保留。
4 比特 16 个优先级 (中断数量为 96 或 128 个)	31:8	- 保留。
	7:4	PRI 建议设置成最低优先级 4' b1111。
	5:0	- 保留。

### 5.3.3 中断处理机制

矢量中断控制器支持电平中断和脉冲中断。

对于电平中断，矢量中断控制器采样到中断有效信号的高电平后设置对应中断进入等待状态，然后请求 CPU 响应。电平中断要求中断服务程序中清除外设的中断源有效信号，否则当中断退出时中断控制器会重新向 CPU 发起中断请求。外设可以根据这一特点，常置中断信号直到不再需要中断处理程序处理。

对于脉冲中断，又称为边沿中断，矢量中断控制器采样中断有效信号的上升沿，然后设置对应中断进入等待状态，随后向 CPU 发起中断请求。为了确保矢量中断控制器检测到脉冲中断，外围需要将中断信号至少保持一个 CPU 时钟周期。在 CPU 响应该脉冲中断请求前，若脉冲中断源向矢量中断控制器发起多次中断请求，矢量中断控制器只会记录一次中断请求；在 CPU 响应该脉冲中断请求后，若脉冲中断源再次向中断控制器发起请求，矢量中断控制器会再次触发对应中断进入等待状态，该处于等待状态的中断请求在上次中断退出后才能够再次被 CPU 响应。

另外，矢量中断控制器支持软件中断。软件可通过设置中断设置等待寄存器（VIC\_ISPR）置高相应的中断等待状态位，触发该中断进入等待状态，向 CPU 发送中断请求。

当处理器响应中断请求后，矢量中断控制器会自动清除相应中断的等待状态位。也可以通过设置中断清除等待寄存器（VIC\_ICPR）清除相应中断的等待状态位。对于电平中断，如果中断有效信号持续为高，则无法通过设置 VIC\_ICPR 寄存器清除等待状态位。

#### 5.3.3.1 中断状态位

VIC 为每个中断源提供 2 比特状态位，分别为：

- Pending：表征该中断处于等待状态，即该中断请求等待 CPU 进行响应。
  - 0：表征中断尚未处于等待状态；
  - 1：表征中断已经处于等待状态。
- Active：表征该中断请求已经被 CPU 响应，但尚未处理完成。
  - 0：表征该中断请求尚未被 CPU 响应；
  - 1：表征该中断请求已经被 CPU 响应，但尚未处理完成。

Pending 位的设置条件：

1. 电平中断源，中断源有效信号置高，且 Active 为低或者 CPU 正在退出该中断服务程序；
2. 脉冲中断源，上升沿有效；
3. 软件设置 ISPR。

Pending 位的清除条件：

1. CPU 响应该中断请求；
2. 软件清除 ICPR。

Active 位的设置条件：CPU 响应该中断请求。

Active 位清除条件：CPU 退出该中断服务程序。

---

**注解：**对于电平中断源，由于需要中断服务程序中将电平中断源请求拉低，因此只需要在 Active 为低或者退出中断服务程序时采样中断源有效信号并设置 Pending 位；对于脉冲中断源，由于请求信号会自动拉低，因此需要实时采样脉冲中断信号的上升沿并设置 Pending 位。

---

### 5.3.3.2 中断优先级

VIC 通过中断优先级设置寄存器（IPR0-IPR31）为每个中断源提供优先级设置，中断个数小于等于 32 个时，硬件可以提供 4 个优先级；中断个数小于等于 64 个，硬件可以提供 8 个优先级；中断个数为 96 个或 128 个时，硬件可以提供 16 个优先级。优先级号越低，优先级越高，具体参考[中断优先级设置寄存器（VIC\\_IPR0 - VIC\\_IPR31）](#)。当软件没有设置优先级寄存器时，所有中断源优先级默认为最高优先级——0。

当多个中断处于 Pending 状态时，VIC 根据各个中断的优先级仲裁出优先级最高的中断请求提交给 CPU 处理。例如，两个中断请求 IRQ0 和 IRQ1 同时处于 Pending 状态，如果 IRQ1 的优先级号小于 IRQ0，即 IRQ1 的优先级高于 IRQ0，因此 IRQ0 先提交给 CPU 处理。

当多个 Pending 的中断拥有相同的优先级号时，根据中断号决定中断提交的顺序，中断号小的优先提交给 CPU 处理。例如，两个中断请求 IRQ0 和 IRQ1 的中断优先级相同，IRQ0 的中断源号小于 IRQ1，因此 IRQ0 优先提交给 CPU 处理。

### 5.3.3.3 中断向量号

中断向量号，是中断请求在异常向量表的位置编号。[表 5.23](#) 给出了 E804 的异常向量表，开始的 0-30 号向量是用作处理器内部识别的向量；31 号向量保留；从 32 开始的向量号预留给外部中断请求，且每个中断源对应一个中断向量号。

表 5.23: 中断向量号描述

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 0 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。
9	024	IDLY 异常。
10	028	普通中断。(自动向量)。
11-15	02C - 03C	保留。
16 - 19	040 - 04C	陷阱指令异常 (TRAP # 0-3)。
20-21	050-054	保留。
22	058	Tspend 中断。
23-29	05C-074	保留。
30	078	浮点异常。
31	07C	保留。
32	080	IRQ0。
33	084	IRQ1。
.....	.....	.....
32+n	0x80+4n	IRQn。

#### 5.3.3.4 中断处理过程

中断处理过程可分以下几个步骤进行：

- 中断源请求的同步：外部设备产生中断源请求，系统完成异步中断源请求到 CPUCLK 时钟域的同步操作，置高 pad\_vic\_int\_vld；
- 中断源请求的采样：VIC 根据中断源的类型对中断源请求进行采样；当采样到有效的中断请求时，设置 Pending 状态位，触发对应中断进入等待状态；
- 中断请求发起：在所有处于等待状态的中断中，经过优先级仲裁向 CPU 发起中断请求；
- 中断响应：CPU 在指令退休时响应中断，返回中断响应信号给 VIC，同时将 PSR 和 PC 更新到 EPSR 和 EPC，并将被响应中断的中断向量号更新 PSR.VEC，清除 PSR.EE，最后取异常入口地址；VIC 根据中断响应信号清除相应中断的 Pending 状态位，并设置其 Active 状态位；
- 中断现场保存：首先保存中断控制寄存器现场 {EPSR, EPC}，打开 PSR.EE 和 PSR.IE，以使能中断嵌套；随后保存通用寄存器现场；

- 中断事务：CPU 开始处理中断事务，对于电平中断，需要将中断源信号清除，否则在中断退出时会重入该中断；
- 中断现场恢复和中断退出：首先恢复通用寄存器现场；随后恢复中断控制寄存器现场 {EPSR, EPC}，将 EPC 和 EPSR 恢复到 PC 和 PSR，退出中断服务程序；VIC 接收中断退出信号，清除 Active 状态位。

中断现场的保存可通过在中断服务程序的起始处执行 NIE 和 IPUSH 指令完成，中断现场的恢复和退出可通过在中断服务程序的结尾处执行 IPOP 和 NIR 指令完成。

中断源请求的同步由系统完成，VIC 内部不实现。图 5.22 给出了 pad\_vic\_int\_vld 信号同步的一个示例，中断源请求信号 irq\_n 经过两级 CPUCLK 的寄存器同步到 CPU 时钟域上。另外，中断源的类型配置信号 pad\_vic\_int\_cfg 对于给定的中断源是一个固定值，0 表示电平中断源，1 表示脉冲中断源，因此不需要同步。

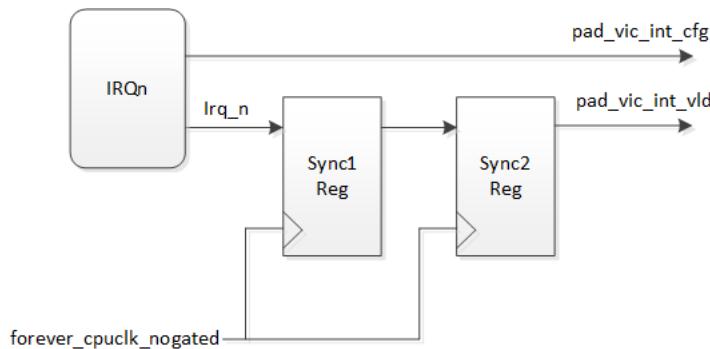


图 5.22：中断源请求同步电路示例

### 5.3.3.5 中断嵌套

VIC 支持中断嵌套功能，在中断处理的过程中允许更高优先级的中断抢占，从而提高中断响应实时性。当存在多级中断嵌套时，特定场景下可能需要改变被抢占中断的优先级，例如低优先级中断响应时间达到最大限制。此时软件可通过设置中断优先级阈值寄存器，提高中断抢占的优先级条件，使被抢占的低优先级中断能得到及时响应。

#### 中断嵌套优先级条件

中断抢占的优先级条件可分为两种，如下所示：

- 当中断优先级阈值未使能时，中断抢占的优先级必须高于当前 CPU 正在处理的中断的优先级；同级优先级不能进行抢占；
- 当中断优先级阈值使能时，中断抢占的优先级不仅要高于当前 CPU 正在处理的中断优先级，而且要高于中断优先级阈值寄存器设置的阈值。

VIC 支持中断优先级的动态调整，当被抢占中断的优先级需要调高或者调低时，在设置中断优先级设置寄存器的同时设置中断优先级阈值寄存器。

图 5.23 给出了中断抢占的示例。中断优先级设置为：IRQ0<IRQ1<IRQ2<IRQ3；中断源请求产生的顺序为：IRQ0>IRQ1>IRQ2>IRQ3。CPU 首先响应了 IRQ0，在 IRQ0 中断服务程序执行的过程中，来了更高优先级的 IRQ1，因此 IRQ0 被抢占，CPU 开始执行 IRQ1 的中断服务程序。同样，IRQ2 对 IRQ1 进行了抢占，并设置了中断优先级阈值寄存器（IPTR.VECTTHRESHOLD=IRQ0, IPTR.PRIOTHRESHOLD=0, IPTR.EN=1）。当 IRQ3 到来时，尽管优先级高于 IRQ2，但 IRQ3 的优先级没有高于 IPTR 中设置的优先级阈值所表征的中断，因此 IRQ3 无法抢占 IRQ2。IRQ3 等到 IRQ0 的中断服务程序执行结束硬件自动清除 IPTR.EN 后，才得到 CPU 响应。

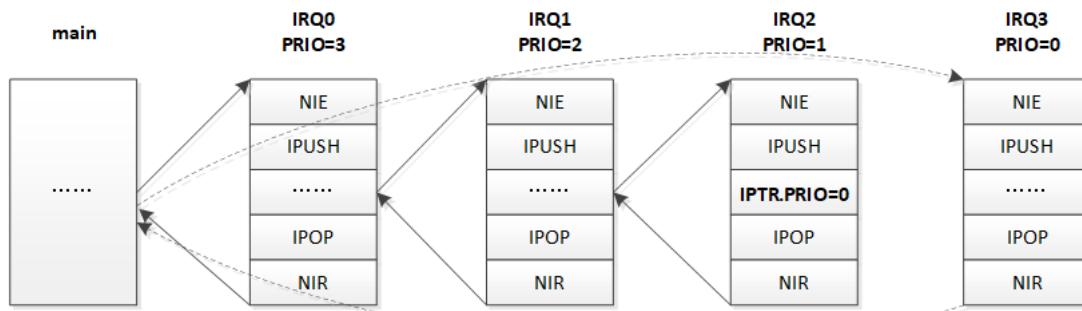


图 5.23：中断嵌套优先级示例

### 中断嵌套时机条件

中断抢占有满足优先级的条件下，还需要判断当前中断响应的阶段。中断响应的过程参考[中断处理过程](#)，和嵌套相关的主要分为以下几个阶段：

- (1) EPSR、EPC、PSR 的更新和异常入口地址的读取
- (2) NIE 指令
- (3) IPUSH 指令
- (4) 中断事务
- (5) IPOP 指令
- (6) NIR 指令。

为了保证中断嵌套现场的保存和恢复，CPU 在以下阶段不能被中断打断：

- 中断响应之后更新 EPSR、EPC、PSR 和获取异常入口地址的过程中；
- NIE 指令执行过程中，且包括指令退休；
- PSR.IC 位关闭，IPUSH 和 IPOP 指令执行过程中，不包括指令退休；
- NIR 指令执行过程中，不包括指令退休。

CPU 在以下阶段可安全可靠地响应新的中断：

- 正常程序的执行过程中，在中断响应之前；
- IPUSH、IPOP 指令退休时；

- PSR.IC 位打开，IPUSH、IPOP 指令执行过程中；
- NIR 指令退休时；
- 中断事务处理的过程中。

在 PSR.IC 位打开时，IPUSH 和 IPOP 指令在执行过程中可响应中断，因此中断返回时需要重新执行该指令。对于 IPUSH 或 IPOP 指令在退休时响应中断的情况，在中断退出后直接执行 IPUSH/IPOP 的下一条指令，不需要重新执行该 IPUSH/IPOP 指令。NIR 指令执行过程中不可被中断打断，但在退休时可响应中断，如果中断打在 NIR 指令上，CPU 在 NIR 指令退休时，直接将 NIR 的返回地址压入堆栈，在中断退出时返回 NIR 的目标地址。

图 5.24 给出了 IRQ0/IRQ1/IRQ2/IRQ3 中断嵌套的过程。在 IRQ0 被 CPU 响应后，产生了更高优先级的 IRQ1，当 PSR.IC 打开时，在执行到 IPUSH 指令时响应 IRQ1。IRQ2 在 IRQ1 处理中断事务时产生，因此可立即被 CPU 响应。IRQ3 在 IRQ2 执行 NIR 指令时产生，在 NIR 指令退休时响应 IRQ3。当 IRQ3 处理完退出中断服务程序时，直接返回到 IRQ1 被 IRQ2 打断的点。当 IRQ1 返回 IRQ0 时，需要重新执行 IPUSH 指令。

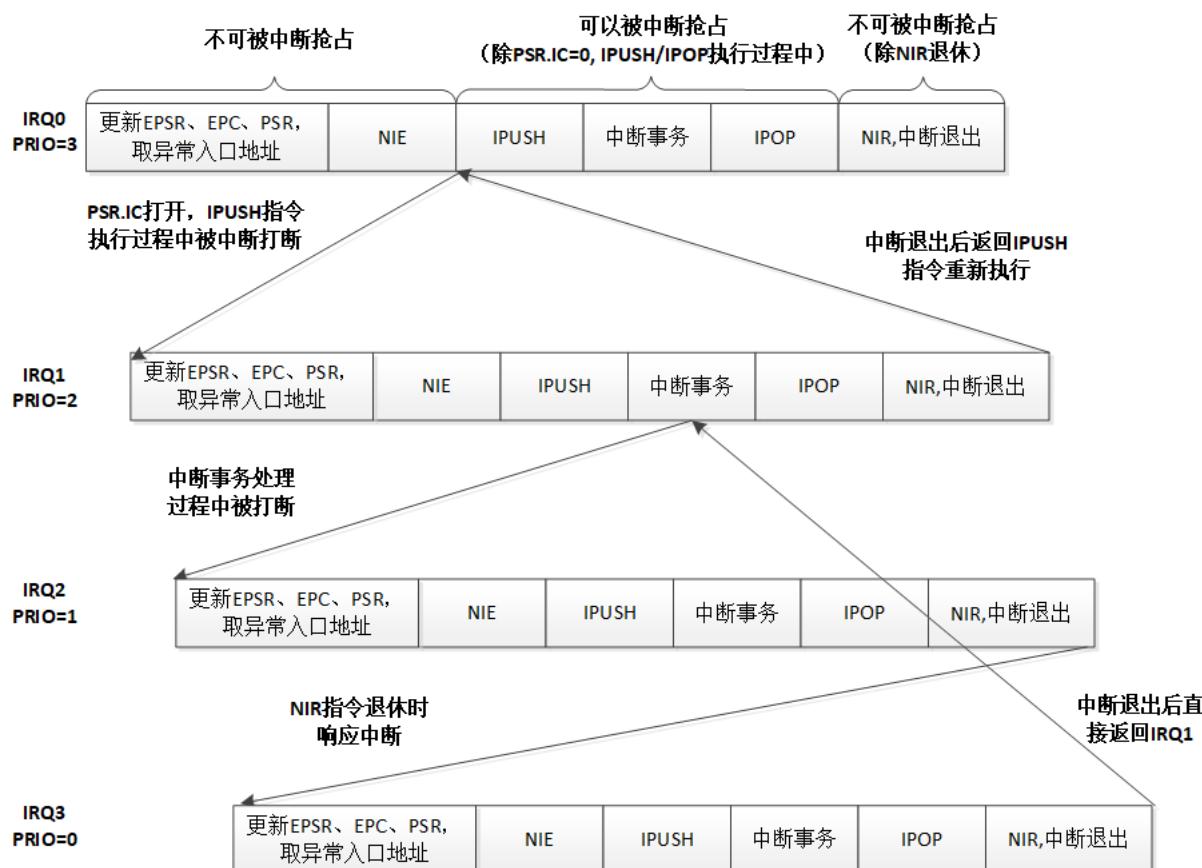


图 5.24: 中断嵌套时机条件示例

### 5.3.4 Tspend 中断

Tspend 中断控制器提供的，一个不占用外部中断向量号（32-159）的中断，Tspend 中断向量号为 22，tspend 中断功能和设置同其他中断基本相同，tspend 中断和其他中断的区别在于：

1. Tspend 中断，不需要外部设别产生中断源请求，无 pending 状态位，软件设置 VIC\_TSPEND 使能时进入等待状态。
2. Tspend 中断和其他中断优先级相同时，不以中断号大小判断中断优先级，相同中断优先级时 Tspend 中断优先级为最低；等到没有其他同级别中断处于 pending 状态时，处理 tspend 中断。

### 5.3.5 操作步骤

为保证 VIC 能够生成预期优先级的中断请求，用户需要提前设置好中断优先级并使能响应中断，步骤如下：

1. 首先设置 VIC\_IPR0-31，给每个中断配置合适的优先级；
2. 然后设置 VIC\_ISER，使能相应的中断。

另外，VIC 支持软件设置 VIC\_ISPR 产生中断请求，同样在 VIC\_ISPR 设置之前必须按上述步骤配置中断优先级和中断使能位。

---

**注解：** CPU 响应 VIC 触发的中断请求之前需要使能 PSR.IE 和 PSR.EE，以开启 CPU 的中断响应使能，否则 CPU 无法响应中断。

---

VIC 采样外部中断源产生中断低功耗唤醒之前，必须先设置 VIC\_IWER，置高相应的低功耗唤醒使能位，否则无法产生低功耗唤醒请求。

---

**注解：** 对于某个外部中断源请求，只需要相应中断的低功耗唤醒功能被使能（对应于 VIC\_IWER），就能产生低功耗唤醒请求，而不依赖于该中断自身的中断使能（对应于 VIC\_ISER）。

---

### 5.3.6 接口信号

矢量中断控制器的接口信号可分为三组，分别为：

- 矢量中断控制器与中断源的接口：VIC 接收并采样中断源的中断请求信号；
- 矢量中断控制器与 TCIP 的接口：TCIP 提供一组读写 VIC 寄存器的接口，完成 VIC 相关控制寄存器的读写，如使能中断、设置中断优先级、查询当前正在处理中断等；
- 矢量中断控制器与处理器核 E804 的接口：VIC 向 E804 发起中断请求和相应的中断向量号，E804 返回 VIC 中断响应和中断返回指示信号以及目前正在处理的中断向量号。

表 5.24: 矢量中断控制器接口信号

信号名	I/O	Reset	定义
<b>紧耦合 IP 总线接口信号:</b>			
tcipif_vic_sel	I	0	选中信号： 指示选中矢量中断控制器并进行数据传输。 1: 选中 VIC; 0: 未选中 VIC。
tcipif_vic_addr[15:0]	I	-	地址总线： 16 位地址总线（截取 32 位地址总线的低 16 位），指示访问地址。
tcipif_vic_write	I	0	读写表示信号： 指示当前 TCIP 访问是读取数据还是写数据： 1: 写访问； 0: 读访问。
tcipif_vic_wdata[31:0]	I	-	写数据总线： 32 位写数据总线。
vic_tcipif_rdata[31:0]	O	-	读数据总线： 32 位读数据总线。
vic_tcipif_cmplt	O	0	传输完成指示信号： 有效时指示当前传输已完成。
<b>处理器中断握手信号:</b>			
vic_pad_int_b	O	1	中断请求信号： 低电平时表示进行普通中断申请。
vic_pad_intraw_b	O	1	中断唤醒请求信号： 低电平时表示进行低功耗唤醒申请。
vic_pad_int_vec_b[7:0]	O	-	中断矢量序号信号： 提供 core 进行中断处理的向量号。
pad_vic_int_ack	I	0	中断响应信号： 指示 CPU 响应了当前的中断请求。
pad_vic_int_exit	I	0	中断服务程序退出信号： 指示 CPU 退出中断服务程序。
pad_vic_int_vec[7:0]	I	-	指示 CPU 正在处理的中断向量号。
pad_vic_ack_vec[7:0]	I	-	指示 CPU 响应的中断向量号。
<b>中断源信号:</b>			
pad_vic_int_cfg[127:0]	I	-	中断源类型配置信号： 0: 电平中断源； 1: 脉冲中断源。
pad_vic_int_vld[127:0]	I	0	中断有效信号： 高电平有效。

表 5.25: 矢量中断控制器接口信号

信号名	I/O	Reset	定义
<b>时钟信号:</b>			
forever_cpublk	I	-	VIC 的工作时钟： 矢量中断控制器中和低功耗唤醒无关的寄存器均工作于该时钟，低功耗模式下可以关闭该时钟信号。
forever_cpublk_nogated	I	-	VIC 中断采样电路的时钟： 矢量中断控制器中和低功耗唤醒相关的寄存器均工作于该时钟。在低功耗模式下不可以关闭该时钟信号。
<b>复位信号:</b>			
cpurst_b	I	-	矢量中断控制器复位信号： 低电平时，初始化矢量中断控制器内部。矢量中断控制器采用异步复位方式。
<b>其它信号:</b>			
pad_yy_gate_clk_en_b	I	-	门控时钟使能信号： 只有当这个信号有效时，VIC 的内部模块的门控时钟才能有效。 不用该信号时，需要接 1。
pad_yy_test_mode	I	-	进入测试模式： 使 VIC 进入测试模式，此时 VIC 时钟为测试时钟 (pad_had_jtg_tclk)。只有 VIC 进入测试模式，并且处理器输入信号 pad_yy_scan_enable 有效时，才可以通过扫描链进行测试。不用该信号时，需要接 0。

当 CPU 内部没有集成矢量中断控制器时，由外部中断控制器对中断源进行仲裁，产生中断请求发送给 CPU。CPU 对外部输入的中断请求信号进行同步并传递给核内处理。另外，CPU 也需要对外部 IP 产生低功耗唤醒信号进行同步。

表 5.26: 没有配置矢量中断控制器的接口信号

信号名	I/O	Reset	定义
<b>紧耦合 IP 总线接口信号:</b>			
pad_cpu_int_b	I	1	中断请求信号: 低电平时表示进行普通中断申请。
pad_cpu_intraw_b	I	1	低功耗唤醒请求信号: 低电平时表示进行低功耗唤醒申请。
pad_cpu_int_vec_b[7:0]	I	-	中断向量号: 指示当前中断请求对应中断向量号。
<b>时钟信号:</b>			
forever_cpubclk_nogated	I	-	VIC 同步电路时钟: 用于中断请求和低功耗唤醒信号的同步。
clk_en	I	-	系统时钟同步使能信号。

### 5.3.7 中断设置示例

```
//设置 psr 中的 ee 和 ie 位, cpu 响应中断

psrset ee, ie

//设置中断号为 32-35 的中断使能位

lrw r1, 0x0
bseti r1, 0x0 //设置 32 号中断的使能位
bseti r1, 0x1 //设置 33 号中断的使能位
bseti r1, 0x2 //设置 34 号中断的使能位
bseti r1, 0x3 //设置 35 号中断的使能位
lrw r2, 0xe000e100 //中断使能寄存器 ISER 对应的地址
st.w r1, (r2, 0x0) //使能中断号为 32-35 的 4 个中断

//其他中断的使能同上, 通过设置 ISER 寄存器的不同位使能相应的中断

//设置中断优先级寄存器 IPRO, 设置 32-35 号中断的优先级

lrw r1, 0x0

//设置 IPRO[7:6] 设置 32 号中断的优先级为 2' b11, 最低优先级

//设置 IPRO[15:14] 设置 33 号中断的优先级为 2' b00, 最高优先级
```

(下页继续)

(续上页)

```
//设置 IPRO[23:22] 设置 34 号中断的优先级为 2' b10

//设置 IPRO[31:30] 设置 35 号中断的优先级为 2' b10

bseti r1, 0x6 //通过低 [7:6] 两位设置 32 号中断优先级
bseti r1, 0x7 //设置 32 号中断的优先级为最低, IPRO[7:6]=2' b11
bclri r1, 0x14 //通过 [15:14] 两位设置 33 号中断优先级
bclri r1, 0x15 //设置 33 号中断的优先级为最高, IPRO[15:14]=2' b00
bclri r1, 0x22 //通过 [23:22] 两位设置 34 号中断优先级
bseti r1, 0x23 //设置 34 号中断的优先级为 2, IPRO[23:22]=2' b10
bclri r1, 0x30 //通过 [31:30] 两位设置 35 号中断优先级
bseti r1, 0x31 //设置 35 号中断的优先级为 2, IPRO[31:30]=2' b10

//将设置的优先级写入 IPRO

lw r2, 0xe000e400 //中断使能寄存器 IPRO 对应的地址
st.w r1, (r2, 0x0) //完成 32-35 号的中断优先级设置

//其他中断的优先级设置同上，通过设置对应的中断优先级寄存器 IPR1-IPR7 即可。

//设置中断优先级阈值寄存器 VIC_IPTR

lw r1, 0x2200 //设置中断优先级阈值寄存器的中断号, IPTR[16:8]=34
bseti r1, 0x0 //使能中断优先级阈值寄存器

//通过 IPTR[7:6] 两位设置能够抢占 34 号中断的中断优先级，

//设置能够抢占 34 号中断的先级必须优先级高于 2' b01，

//即只有优先级为 2' b00 (最高) 的中断可以抢占 34 号中断

bseti r1, 0x6
bclri r1, 0x7

//将设置的 34 号中断号对应的优先级阈值，使能和中断号写入 IPTR

lw r2, 0xe000ec04 //中断优先级阈值寄存器地址
st.w r1, (r2, 0x0) //完成 34 号中断的优先级阈值设置
```

# 第六章 指令集

## 6.1 概述

E804 的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。

E804 指令有两种宽度：16 位指令和 32 位指令，指令代码由两种指令混编而成，两种指令之间的切换没有额外的开销。

## 6.2 32 位指令

本节主要介绍 E804 实现的 CSKY V2 的 32 位指令集。

### 6.2.1 32 位指令功能分类

E804 的 32 位指令集按照指令实现的功能划分，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；
- DSP 扩展指令。

#### 6.2.1.1 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

表 6.1: 32 位加减法指令列表

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。
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。
DEC GT32	减法大于零置 C 位指令。
DEC LT32	减法小于零置 C 位指令。
DEC NE32	减法不等于零置 C 位指令。

### 逻辑操作指令

表 6.2: 32 位逻辑操作指令列表

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。

### 移位指令

表 6.3: 32 位移位指令列表

LSL32	逻辑左移指令。
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。

## 比较指令

表 6.4: 32 位比较指令列表

CMPNE32	不等比较指令。
CMPNEI32	立即数不等比较指令。
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。
TST32	零测试指令。
TSTNBZ32	无字节等于零寄存器测试指令。

## 数据传输指令

表 6.5: 32 位数据传输指令列表

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。

### 比特操作指令

表 6.6: 32 位比特操作指令列表

BSETI32	立即数位置位指令。
BTSTI32	立即数位测试指令。

### 提取插入指令

表 6.7: 32 位提取插入指令列表

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。
RE VH32	半字内字节倒序指令。

### 乘除法指令

表 6.8: 32 位乘除法指令列表

MULT32	乘法指令。
MULLL.S16 (MULSH32)*	16 位有符号乘法指令。
MUL.(U/S)32	32 位(无/有)符号乘法指令。
MULA.32.L	32 位有符号乘累加取低 32 位指令。
MULA.(U/S)32	32 位(无/有)符号乘累加指令。
MULALL.S16.S	带饱和操作的 16 位有符号低半字乘累加指令。
DIVU32	无符号除法指令。
DIVS32	有符号除法指令。

注解: \* 历史版本中的指令 MULSH32 将由 MULLL.S16 替代, 功能不变。

### 杂类运算指令

表 6.9: 32 位杂类运算指令列表

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。

#### 6.2.1.2 分支跳转指令

分支跳转指令可以进一步分为:

##### 分支指令

表 6.10: 32 位分支指令列表

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。
BHSZ32	寄存器大于等于零分支指令。
BLZ32	寄存器小于零分支指令。
BLSZ32	寄存器小于等于零分支指令。

## 跳转指令

表 6.11: 32 位跳转指令列表

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。

### 6.2.1.3 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

表 6.12: 32 位立即数偏移存取指令列表

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。
ST32.B	字节存储指令。
ST32.H	半字存储指令。
ST32.W	字存储指令。

#### 向量寄存器偏移存取指令

表 6.13: 32 位向量寄存器偏移存取指令列表

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。
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。

#### 多寄存器存取指令

表 6.14: 32 位多寄存器存取指令列表

LDQ32	连续四字加载指令。
LDM32	连续多字加载指令。
STQ32	连续四字存储指令。
STM32	连续多字存储指令。
PUSH32	压栈指令。
POP32	出栈指令。

### 符号存取指令

表 6.15: 32 位符号存取指令列表

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

#### 6.2.1.4 特权指令

特权指令可以进一步分为：

##### 控制寄存器操作指令

表 6.16: 32 位控制寄存器操作指令列表

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。

##### 低功耗指令

表 6.17: 32 位低功耗指令列表

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。

##### 异常返回指令

表 6.18: 32 位异常返回指令列表

RTE32	异常和普通中断返回指令。
-------	--------------

### 6.2.1.5 特殊功能指令

特殊功能指令如 表 6.19 所示。

表 6.19: 32 位特殊功能指令列表

SYNC32	CPU 同步指令。
BKPT32	断点指令。
SCE32	条件执行设置指令。
IDLY32	中断识别禁止指令。
TRAP32	无条件操作系统陷阱指令。

### 6.2.1.6 DSP 扩展指令

E804 在配置了 DSP 扩展单元时，会支持 DSP 扩展指令。以下以指令英文首字母为序对 DSP 指令子集进行表述。

每条 DSP 指令的助记符包含三部分，每部分以“.”分开，第一部分标识指令的基本操作，第二部分标识操作数的数据类型，第三部分标识对运算结果的处理。以“带饱和操作的 16 位并行有符号加法指令-PADD.S16.S”为例，PADD 表示这是一条并行加法指令，S16 表示操作数为有符号半字，S 表示对运算结果取饱和操作。

表 6.20: 指令助记符中常见的字母及其表示意义

助记符	英文表述	意义
第一部分	P(Parallel)	并行计算，表示单指令多数据操作。
第一部分	R(fRaction)	小数操作。
第一部分	X(Unalign)	操作数位宽不对齐，或操作数位置不对齐。
第一部分	H(Half)	取平均值。
第二部分	(S/U)/(8/16/32)	(有符号/无符号)/(字节/半字/字) 如果不带符号位指示，则表示该条指令的操作对符号位不关心。
第三部分	S(Saturate)	饱和操作。
第三部分	R(Round)	舍入操作。
第三部分	E(Extension)	扩位操作，即操作数进扩位操作后参与运算产生结果。

具体指令如 表 6.21 所示。

表 6.21: DSP 扩展指令列表

指令名称	指令描述
<b>加减法指令</b>	
PADD.8	8 位并行加法指令。
PADD.16	16 位并行加法指令。
PADD.(U/S)8.S	带饱和操作的 8 位并行 (无/有) 符号加法指令。
PADD.(U/S)16.S	带饱和操作的 16 位并行 (无/有) 符号加法指令。
ADD.(U/S)32.S	带饱和操作的 32 位 (无/有) 符号加法指令。
PSUB.8	8 位并行减法指令。
PSUB.16	16 位并行减法指令。
PSUB.(U/S)8.S	带饱和操作的 8 位并行 (无/有) 符号减法指令。
PSUB.(U/S)16.S	带饱和操作的 16 位并行 (无/有) 符号减法指令。
SUB(U/S)32.S	带饱和操作的 32 位 (无/有) 符号减法指令。
PADDH.(U/S)8	8 位并行 (无/有) 符号加法求平均值指令。
PADDH.(U/S)16	16 位并行 (无/有) 符号加法求平均值指令。
ADDH.(U/S)32	32 位 (无/有) 符号加法求平均值指令。
PSUBH.(U/S)8	8 位并行 (无/有) 符号减法求平均值指令。
PSUBH.(U/S)16	16 位并行 (无/有) 符号减法求平均值指令。
SUBH.(U/S)32	32 位 (无/有) 符号减法求平均值指令。
PASX.16	16 位并行交叉加减指令。
PSAX.16	16 位并行交叉减加指令。
PASX.(U/S)16.S	带饱和操作的 16 位并行 (无/有) 符号交叉加减指令。
PSAX.(U/S)16.S	带饱和操作的 16 位并行 (无/有) 符号交叉减加指令。
PASXH.(U/S)16	16 位并行 (无/有) 符号交叉加减求平均值指令。
PSAXH.(U/S)16	16 位并行 (无/有) 符号交叉减加求平均值指令。
ADD.64	64 位加法指令。
ADD.(U/S)64.S	带饱和操作的 64 位 (无/有) 符号加法指令。
SUB.64	64 位减法指令。
SUB.(U/S)64.S	带饱和操作的 64 位 (无/有) 符号减法指令。
<b>移位指令</b>	
ASRI.S32.R	带舍入操作的立即数算术右移指令。
ASR.S32.R	带舍入操作的算术右移指令。
LSRI.U32.R	带舍入操作的立即数逻辑右移指令。
LSR.U32.R	带舍入操作的逻辑右移指令。
LSLI.(U/S)32.S	带饱和操作的立即数 (无/有) 符号逻辑左移指令。
LSL.(U/S)32.S	带饱和操作的 (无/有) 符号逻辑左移指令。
PASRI.S16	16 位并行立即数算术右移指令。
PASR.S16	16 位并行算术右移指令。
PASRI.S16.R	带舍入操作的 16 位并行立即数算术右移指令。

下页继续

表 6.21 – 续上页

指令名称	指令描述
PASR.S16.R	带舍入操作的 16 位并行算术右移指令。
PLSRI.U16	16 位并行立即数逻辑右移指令。
PLSR.U16	16 位并行逻辑右移指令。
PLSRI.U16.R	带舍入操作的 6 位并行立即数逻辑右移指令。
PLSR.U16.R	带舍入操作的 16 位并行逻辑右移指令。
PLSLI.16	16 位并行立即数逻辑左移指令。
PLSL.16	16 位并行逻辑左移指令。
PLSLI.(U/S)16.S	带饱和操作的 16 位并行立即数（无/有）符号逻辑左移指令。
PLSL.(U/S)16.S	带饱和操作的 16 位并行（无/有）符号逻辑左移指令。
<b>控制流转数据量指令（或称为比较/选择指令）</b>	
SEL	位选择指令。
PCMPNE.8	8 位并行比较不等指令。
PCMPNE.16	16 位并行比较不等指令。
PCMPHS.(U/S)8	8 位并行（无/有）符号大于等于比较指令。
PCMPHS.(U/S)16	16 位并行（无/有）符号大于等于比较指令。
PCMPLT.(U/S)8	8 位并行（无/有）符号小于比较选择指令。
PCMPLT.(U/S)16	16 位并行（无/有）符号小于比较选择指令。
PMAX.(U/S)8	8 位并行（无/有）符号取大值指令。
PMAX.(U/S)16	16 位并行（无/有）符号取大值指令。
MAX.(U/S)32	32 位（无/有）符号取大值指令。
PMIN.(U/S)8	8 位并行（无/有）符号取小值指令。
PMIN.(U/S)16	16 位并行（无/有）符号取小值指令。
MIN.(U/S)32	32 位（无/有）符号取小值指令。
<b>数据抽取/拆封/饱和等指令</b>	
DEXTI	立即数字截取指令。
DEXT	字截取指令。
PKG	立即数移位打包指令。
PKGLL	低半字打包指令。
PKGHH	高半字打包指令。
PEXT.U8.E	8 位并行无符号扩展指令。
PEXT.S8.E	8 位并行有符号扩展指令。
PEXTX.U8.E	8 位并行无符号交叉扩展指令。
PEXTX.S8.E	8 位并行有符号交叉扩展指令。
NARL	低位截取拼装指令。
NARH	高位截取拼接指令。
NARLX	低位交叉截取拼装指令。
NARHX	高位交叉截取拼接指令。

下页继续

表 6.21 – 续上页

指令名称	指令描述
CLIP1.(U/S)32	立即数 (无/有) 符号裁剪饱和指令。
CLIP.(U/S)32	(无/有) 符号裁剪饱和指令。
PCLIP1.(U/S)16	16 位并行立即数 (无/有) 符号裁剪饱和指令。
PCLIP.(U/S)16	16 位并行 (无/有) 符号裁剪饱和指令。
PABS.S8.S	带饱和操作的 8 位并行绝对值指令。
PABS.S16.S	带饱和操作的 16 位并行绝对值指令。
ABS.S32.S	带饱和操作的 32 位绝对值指令。
PNEG.S8.S	带饱和操作的 8 位并行取反指令。
PNEGS16.S	带饱和操作的 16 位并行取反指令。
NEG.S32.S	带饱和操作的 32 位取反指令。
DUP.8	8 位操作数拷贝指令。
DUP.16	16 位操作数拷贝指令。
<b>乘/乘累加/乘累减指令</b>	
<b>32×32 取 64 位</b>	
MULS.(U/S)32	32 位 (无/有) 符号乘累减指令。
MULA.(U/S)32.S	带饱和操作的 32 位 (无/有) 符号乘累加指令。
MULS.(U/S)32.S	带饱和操作的 32 位 (无/有) 符号乘累减指令。
<b>32×32 取高字</b>	
MUL.S32.H	32 位有符号乘法取高 32 位指令。
MUL.S32.RH	带舍入操作的 32 位有符号乘法取高 32 位指令。
RMUL.S32.H	32 位有符号小数乘法取高 32 位指令。
RMUL.S32.RH	带舍入操作的 32 位有符号小数乘法取高 32 位指令。
MULA.S32.HS	带饱和操作的 32 位有符号乘累加取高 32 位指令。
MULS.S32.HS	带饱和操作的 32 位有符号乘累减取高 32 位指令。
MULA.S32.RHS	带舍入和饱和操作的 32 位有符号乘累加取高 32 位指令。
MULS.S32.RHS	带舍入和饱和操作的 32 位有符号乘累减取高 32 位指令。
<b>32×16 取高字</b>	
MULXL.S32	32 位有符号低半字不对齐乘指令。
MULXL.S32.R	带舍入操作的 32 位有符号低半字不对齐乘指令。
MULXH.S32	32 位有符号高半字不对齐乘指令。
MULXH.S32.R	带舍入操作的 32 位有符号高半字不对齐乘指令。
RMULXL.S32	32 位有符号低半字不对齐小数乘指令。
RMULXL.S32.R	带舍入操作的 32 位有符号低半字不对齐小数乘指令。
RMULXH.S32	32 位有符号高半字不对齐小数乘指令。
RMULXH.S32.R	带舍入操作的 32 位有符号高半字不对齐小数乘指令。
MULAXL.S32.S	带饱和操作的 32 位有符号低半字不对齐乘累加指令。
MULAXL.S32.RS	带舍入和饱和操作的 32 位有符号低半字不对齐乘累加指令。

下页继续

表 6.21 – 续上页

指令名称	指令描述
MULAXH.S32.S	带饱和操作的 32 位有符号高半字不对齐乘累加指令。
MULAXH.S32.RS	带舍入和饱和操作的 32 位有符号高半字不对齐乘累加指令。
<b>16×16, 非 SIMD</b>	
MULLL.S16	16 位有符号低半字乘法指令。
MULHH.S16	16 位有符号高半字乘法指令。
MULHL.S16	16 位有符号高低半字乘法指令。
RMULLL.S16	16 位有符号低半字小数乘法指令。
RMULHH.S16	16 位有符号高半字小数乘法指令。
RMULHL.S16	16 位有符号高低半字小数乘法指令。
MULAHH.S16.S	带饱和操作的 16 位有符号高半字乘累加指令。
MULAHL.S16.S	带饱和操作的 16 位有符号高低半字乘累加指令。
MULALL.S16.E	带扩展操作的 16 位有符号低半字乘累加指令。
MULAHH.S16.E	带扩展操作的 16 位有符号高半字乘累加指令。
MULAHL.S16.E	带扩展操作的 16 位有符号高低半字乘累加指令。
<b>16×16, SIMD</b>	
PMUL.(U/S)16	16 位并行 (无/有) 符号乘法指令。
PMULX.(U/S)16	16 位并行 (无/有) 符号交叉乘指令。
PRMUL.S16	16 位并行有符号小数乘法指令。
PRMULX.(U/S)16	16 位并行 (无/有) 符号交叉小数乘法指令。
PRMUL.S16.H	16 位并行有符号小数乘法取高 16 位指令。
PRMUL.S16.RH	带舍入操作的 16 位并行有符号小数乘法取高 16 位指令。
PRMULX.S16.H	16 位并行有符号交叉小数乘法取高 16 位指令。
PRMULX.S16.RH	带舍入操作的 16 位并行有符号交叉小数乘法取高 16 位指令。
<b>16×16, 链加减</b>	
MULCA.S16.S	带饱和操作的 16 位有符号乘链加指令。
MULCAX.S16.S	带饱和操作的 16 位有符号交叉乘链加指令。
MULCS.S16	16 位有符号乘链减指令。
MULCSR.S16	16 位有符号反向乘链减指令。
MULCSX.S16	16 位有符号交叉乘链减指令。
<b>16×16, 链累加减</b>	
MULACA.S16.S	带饱和操作的 16 位有符号乘链加累加指令。
MULACAX.S16.S	带饱和操作的 16 位有符号交叉乘链加累加指令。
MULACS.S16.S	带饱和操作的 16 位有符号乘链减累加指令。
MULACSR.S16.S	带饱和操作的 16 位有符号反向乘链减累加指令。
MULACSX.S16.S	带饱和操作的 16 位有符号交叉乘链减累加指令。
MULSCA.S16.S	带饱和操作的 16 位有符号乘链加累减指令。
MULSCAX.S16.S	带饱和操作的 16 位有符号交叉乘链加累减指令。

下页继续

表 6.21 – 续上页

指令名称	指令描述
MULACA.S16.E	带扩展操作的 16 位有符号乘链加累加指令。
MULACAX.S16.E	带扩展操作的 16 位有符号交叉乘链加累加指令。
MULACS.S16.E	带扩展操作的 16 位有符号乘链减累加指令。
MULACSR.S16.E	带扩展操作的 16 位有符号反向乘链减累加指令。
MULACSRX.S16.E	带扩展操作的 16 位有符号交叉乘链减累加指令。
MULSCA.S16.E	带扩展操作的 16 位有符号乘链加累减指令。
MULSCAX.S16.E	带扩展操作的 16 位有符号交叉乘链加累减指令。
<b>杂类</b>	
PSABSA.U8	8 位并行无符号减法绝对值链加指令。
PSABSAA.U8	8 位并行无符号减法绝对值链加累加指令。
DIVSL	有符号长除法指令。
DIVUL	无符号长除法指令。
MULACA.S8	8 位并行有符号乘链加累加指令。
<b>特色技术 1：循环体加速</b>	
BLOOP	循环体加速指令。
<b>特色技术 2：LD/ST</b>	
LDBI.W	地址自增的字加载指令。
LDBI.H	地址自增的无符号半字加载指令。
LDBI.HS	地址自增的有符号半字加载指令。
LDBI.B	地址自增的无符号字节加载指令。
LDBI.BS	地址自增的有符号字节加载指令。
PLDBI.D	地址自增的双字加载指令。
STBI.W	地址自增的字存储指令。
STBI.H	地址自增的半字存储指令。
STBI.B	地址自增的字节存储指令。
LDBIR.W	寄存器地址自增的字加载指令。
LDBIR.H	寄存器地址自增的无符号半字加载指令。
LDBIR.HS	寄存器地址自增的有符号半字加载指令。
LDBIR.B	寄存器地址自增的无符号字节加载指令。
LDBIR.BS	寄存器地址自增的有符号字节加载指令。
PLDBIR.D	寄存器地址自增的双字加载指令。
STBIR.W	寄存器地址自增的字存储指令。
STBIR.H	寄存器地址自增的半字存储指令。
STBIR.B	寄存器地址自增的字节存储指令。

## 6.3 16 位指令

本节主要介绍 E804 的 16 位指令集。

### 6.3.1 16 位指令功能分类

E804 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；
- 多寄存器存取指令。

#### 6.3.1.1 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

表 6.22: 16 位加减法指令列表

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。

##### 逻辑操作指令

表 6.23: 16 位逻辑操作指令列表

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。

##### 移位指令

表 6.24: 16 位移位指令列表

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。

## 比较指令

表 6.25: 16 位比较指令列表

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。

## 数据传输指令

表 6.26: 16 位数据传输指令列表

MOV16	数据传送指令。
MOVI16	立即数数据传送指令。
MVCV16	C 位传送指令。
LRW16	存储器读入指令。

## 比特操作指令

表 6.27: 16 位比特操作指令列表

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。

## 提取插入指令

表 6.28: 16 位提取插入指令列表

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。
REVH16	半字内字节倒序指令。

### 乘除法指令

表 6.29: 16 位乘法指令列表

MULT16	乘法指令。
--------	-------

#### 6.3.1.2 分支跳转指令

分支跳转指令可以进一步分为:

##### 分支指令

表 6.30: 16 位分支指令列表

BT16	C 为 1 分支指令。
BF16	C 为 0 分支指令。

##### 跳转指令

表 6.31: 16 位跳转指令列表

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。

#### 6.3.1.3 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

表 6.32: 16 位立即数偏移存取指令列表

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。

### 6.3.1.4 多寄存器存取指令

表 6.33: 16 位多寄存器存取指令列表

POP16	出栈指令。
PUSH16	压栈指令。
IPOP16	中断出栈指令。
IPUSH16	中断压栈指令。
NIE16	中断嵌套使能指令。
NIR16	中断嵌套返回指令。

---

注解: NIE16 和 NIR16 需要在超级用户模式下执行。

---

## 6.4 指令集列表

E804 的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 6.34 列出了 E804 指令集中所有 16 位和 32 位指令。

表 6.34: E804 的指令集

汇编指令	32 位	16 位	汇编格式	指令描述
ABS	O	×	ABS32 RZ, RX	绝对值指令。
ADDC	O	O	ADDC32 RZ,RX, RY ADDC16 RZ, RX	无符号带进位加法指令。
ADDI	O	O	ADDI32 RZ, RX, OIMM12 ADDI16 RZ, OIMM8	无符号立即数加法指令。

下页继续

表 6.34 – 续上页

汇编指令	32 位	16 位	汇编格式	指令描述
ADDU	O	O	ADDU32 RZ, RX, RY ADDU16 RZ, RX	无符号加法指令。
AND	O	O	AND32 RZ, RX, RY AND16 RZ, RX	按位与指令。
ANDI	O	×	ANDI32 RZ, RX, IMM12	立即数按位与指令。
ANDN	O	O	ANDN32 RZ, RZ, RX ANDN16 RZ, RX	按位非与指令。
ANDNI	O	×	ANDNI32 RZ, RX, IMM12	立即数按位非与指令。
ASR	O	O	ASR32 RZ, RX, RY ASR16 RZ, RX	算术右移指令。
ASRC	O	×	ASRC32 RZ, RX, OIMM5	立即数算术右移至 C 位指令。
ASRI	O	O	ASRI32 RZ, RX, IMM5 ASRI16 RZ, RX, IMM5	立即数算术右移指令。
BCLRI	O	O	BCLRI32 RZ, RX, IMM5 BCLRI16 RZ, IMM5	立即数位清零指令。
BEZ	O	×	BEZ32 RX, LABEL	寄存器等于零分支指令。
BF	O	O	BF32 LABEL BF16 LABEL	C 为 0 分支指令。
BGENI	O	×	BGENI32 RZ, IMM5	立即数位产生指令。
BGENR	O	×	BGENR32 RZ, RX	寄存器位产生指令。
BHSZ	O	×	BHSZ32 RX, LABEL	寄存器大于等于零分支指令。
BHZ	O	×	BHZ32 RX, LABEL	寄存器大于零分支指令。
BKPT	×	O	BKPT16	断点指令。
BLSZ	O	×	BLSZ32 RX, LABEL	寄存器小于等于零分支指令。
BLZ	O	×	BLZ32 RX, LABEL	寄存器小于零分支指令。
BMASKI	O	×	BMASKI32 RZ, OIMM5	立即数位屏蔽产生指令。
BNEZ	O	×	BNEZ32 RX, LABEL	存器不等于零分支指令。
BR	O	O	BR32 LABEL BR16 LABEL	无条件跳转指令。
BREV	O	×	BREV32 RZ, RX	位倒序指令。
BSETI	O	O	BSETI32 RZ, RX, IMM5 BSETI16 RZ, MM5	立即数位置位指令。
BSR	O	×	BSR32 LABEL	跳转到子程序指令。
BT	O	O	BT32 LABEL BT16 LABEL	C 为 1 分支指令。
BTSTI	O	O	BTSTI32 RX, IMM5 BTSTI16 RX, IMM5	立即数位测试指令。
CLRF	O	×	CLRF32 RZ	C 为 0 清零指令。

下页继续

表 6.34 – 续上页

汇编指令	32 位	16 位	汇编格式	指令描述
CLRT	O	×	CLRT32 RZ	C 为 1 清零指令。
CMPHS	O	O	CMPHS32 RX, RY CMPHS16 RX, RY	无符号大于等于比较指令。
CMPHSI	O	O	CMPHSI32 RX, OIMM16 CMPHSI16 RX, IMM5	立即数无符号大于等于比较指令。
CMPLT	O	O	CMPLT32 RX, RY CMPLT16 RX, RY	有符号小于比较指令。
CMPLTI	O	O	CMPLTI32 RX, OIMM16 CMPLTI16 RX, OIMM5	立即数有符号小于比较指令。
CMPNE	O	O	CMPNE32 RX, RY CMPNE16 RX, RY	不等比较指令。
CMPNEI	O	O	CMPNEI32 RX, IMM16 CMPNEI16 RX, IMM5	立即数不等比较指令。
DECDF	O	×	DECDF32 RZ, RX, IMM5	C 为 0 立即数减法指令。
DECGBT	O	×	DECGBT32 RZ, RX, IMM5	减法大于零置 C 位指令。
DECLT	O	×	DECLT32 RZ, RX, IMM5	减法小于零置 C 位指令。
DECNE	O	×	DECNE32 RZ, RX, IMM5	减法不等于零置 C 位指令。
DECT	O	×	DECT32 RZ, RX, IMM5	C 为 1 立即数减法指令。
DIVS	O	×	DIVS32 RZ, RX, RY	有符号除法指令。
DIVU	O	×	DIVU32 RZ, RX, RY	无符号除法指令。
DOZE	O	×	DOZE32	进入低功耗睡眠模式指令。
FF0	O	×	FF0.32 RZ, RX	快速找 0 指令。
FF1	O	×	FF1.32 RZ, RX	快速找 1 指令。
GRS	O	×	GRS32 RZ, LABEL GRS32 RZ, IMM32	符号产生指令。
IDLY	O	×	IDLY32 N	中断识别禁止指令。
INCF	O	×	INCF32 RZ, RX, IMM5	C 为 0 立即数加法指令。
INCT	O	×	INCT32 RZ, RX, IMM5	C 为 1 立即数加法指令。
INS	O	×	INS32 RZ, RX, MSB, LSB	位插入指令。
IPOP	×	O	IPOP16	中断出栈指令。
IPUSH	×	O	IPUSH16	中断压栈指令。
IXD	O	×	IXD32 RZ, RX, RY	索引双字指令。
IXH	O	×	IXH32 RZ, RX, RY	索引半字指令。
IXW	O	×	IXW32 RZ, RX, RY	索引字指令。
JMP	O	O	JMP32 RX JMP16 RX	寄存器跳转指令。
JMPI	O	×	JMPI32 LABEL	间接跳转指令。

下页继续

表 6.34 – 续上页

汇编指令	32 位	16 位	汇编格式	指令描述
JSR	O	O	JSR32 RX JSR16 RX	寄存器跳转到子程序指令。
JSRI	O	×	JSRI32 LABEL	间接跳转到子程序指令。
LD.B	O	O	LD32.B RZ, (RX, DISP) LD16.B RZ, (RX, DISP)	无符号扩展字节加载指令。
LD.BS	O	×	LD32.BS RZ, (RX, DISP)	有符号扩展字节加载指令。
LD.H	O	O	LD32.H RZ, (RX, DISP) LD16.H RZ, (RX, DISP)	无符号扩展半字加载指令。
LD.HS	O	×	LD32.HS RZ, (RX, DISP)	有符号扩展半字加载指令。
LD.W	O	O	LD32.W RZ, (RX, DISP) LD16.W RZ, (RX, DISP)	字加载指令。
LDM	O	×	LDM32 RY-RZ, (RX)	连续多字加载指令。
LDQ	O	×	LDQ32 R4-R7, (RX)	连续四字加载指令。
LDR.B	O	×	LDR32.B RZ, (RX, RY << 0) LDR32.B RZ, (RX, RY << 1) LDR32.B RZ, (RX, RY << 2) LDR32.B RZ, (RX, RY << 3)	寄存器移位寻址无符号扩展字节 加载指令。
LDR.BS	O	×	LDR32.BS RZ, (RX, RY << 0) LDR32.BS RZ, (RX, RY << 1) LDR32.BS RZ, (RX, RY << 2) LDR32.BS RZ, (RX, RY << 3)	寄存器移位寻址有符号扩展字节 加载指令。
LDR.H	O	×	LDR32.H RZ, (RX, RY << 0) LDR32.H RZ, (RX, RY << 1) LDR32.H RZ, (RX, RY << 2) LDR32.H RZ, (RX, RY << 3)	寄存器移位寻址无符号扩展半字 加载指令。
LDR.HS	O	×	LDR32.HS RZ, (RX, RY << 0) LDR32.HS RZ, (RX, RY << 1) LDR32.HS RZ, (RX, RY << 2) LDR32.HS RZ, (RX, RY << 3)	寄存器移位寻址有符号扩展半字 加载指令。

下页继续

表 6.34 – 续上页

汇编指令	32 位	16 位	汇编格式	指令描述
LDR.W	O	×	LDR32.W RZ, (RX, RY << 0) LDR32.W RZ, (RX, RY << 1) LDR32.W RZ, (RX, RY << 2) LDR32.W RZ, (RX, RY << 3)	寄存器移位寻址字加载指令。
LRS.B	O	×	LRS32.B RZ, [LABEL]	字节符号加载指令。
LRS.H	O	×	LRS32.H RZ, [LABEL]	半字符号加载指令。
LRS.W	O	×	LRS32.W RZ, [LABEL]	字符串加载指令。
LRW	O	O	LRW16 LABEL LRW16 IMM32 LRW32 LABEL LRW32 IMM32	存储器读入指令。
LSL	O	O	LSL32 RZ, RX, RY LSL16 RZ, RY	逻辑左移指令。
LSLC	O	×	LSLC32 RZ, RX, OIMM5	立即数逻辑左移至 C 位指令。
LSLI	O	O	LSLI32 RZ, RX, IMM5 LSLI16 RZ, RX, IMM5	立即数逻辑左移指令。
LSR	O	O	LSR32 RZ, RX, RY LSR16 RZ, RY	逻辑右移指令。
LSRC	O	×	LSRC32 RZ, RX, OIMM5	立即数逻辑右移至 C 位指令。
LSRI	O	O	LSRI32 RZ, RX, IMM5 LSRI16 RZ, RX, IMM5	立即数逻辑右移指令。
MFCR	O	×	MFCR32 RZ, CR <X, SEL>	控制寄存器读传送指令。
MOV	O	O	MOV16 RZ, RX	数据传送指令。
MOVF	O	×	MOVF32 RZ, RX	C 为 0 数据传送指令。
MOVI	O	O	MOVI32 RZ, IMM16 MOVI16 RZ, IMM8	立即数数据传送指令。
MOVIH	O	×	MOVIH32 RZ, IMM16	立即数高位数据传送指令。
MOVT	O	×	MOVT32 RZ, RX	C 为 1 数据传送指令。
MTCR	O	×	MTCR32 RX, CR <Z, SEL>	控制寄存器写传送指令。
MULA.32.L	O	×	MULA.32.L RZ, RX, RY	32 位有符号乘累加取低 32 位指令。
MULALL.S16.S	O	×	MULALL.S16.S RZ, RX, RY	带饱和操作的 16 位有符号低半字乘累加指令。
MULLL.S16(MULSH)	O	O	MULLL.S16 RZ, RX, RY	16 位有符号乘法指令。

下页继续

表 6.34 – 续上页

汇编指令	32 位	16 位	汇编格式	指令描述
MULT	O	O	MULT32 RZ, RX, RY MULT16 RZ, RX	乘法指令。
MUL.(U/S)32	O	×	MUL.S32 RZ, RX, RY	32 位 (无/有) 符号乘法指令。
MVC	O	×	MVC32 RZ	C 位传送指令。
MVCV	O	O	MVCV32 RZ MVCV16 RZ	C 位取反传送指令。
NIE	×	O	NIE	中断嵌套使能指令。
NIR	×	O	NIR	中断嵌套返回指令。
NOR	O	O	NOR32 RZ, RX, RY NOR16 RZ, RX	按位或非指令。
NOT	O	O	NOT32 RZ, RX NOT16 RZ	按位非指令。
OR	O	O	OR32 RZ, RX, RY OR16 RZ, RX	按位或指令。
ORI	O	×	ORI32 RZ, RX, IMM16	立即数按位或指令。
POP	O	O	POP32 REGLIST POP16 REGLIST	出栈指令。
PSRCLR	O	×	PSRCLR32 EE, IE, FE, AF	PSR 位清零指令。
PSRSET	O	×	PSRSET32 EE, IE, FE, AF	PSR 位置位指令。
PUSH	O	O	PUSH32 REGLIST PUSH16 REGLIST	压栈指令。
REVB	O	O	REVB32 RZ, RX REVB16 RZ, RX	字节倒序指令。
REVH	O	O	REVH32 RZ, RX REVH16 RZ, RX	半字内字节倒序指令。
ROTL	O	O	ROTL32 RZ, RX, RY ROTL16 RZ, RY	循环左移指令。
ROTLI	O	×	ROTLI32 RZ, RX, IMM5	立即数循环左移指令。
RSUB	O	×	RSUB32 RZ, RX, RY	反向减法指令。
RTE	O	×	RTE32	异常和普通中断返回指令。
RTS	O	O	RTS16	链接寄存器跳转指令。
SCE	O	×	SCE32 COND	条件执行设置指令。
SEXT	O	×	SEXT32 RZ, RX, MSB, LSB	位提取并有符号扩展指令。
SEXTB	O	O	SEXTB32 RZ, RX SEXTB16 RZ, RX	字节提取并有符号扩展指令。
SEXTH	O	O	SEXTH32 RZ, RX SEXTH16 RZ, RX	半字提取并有符号扩展指令。
SRS.B	O	×	SRS32.B RZ, [LABEL]	字节符号存储指令。

下页继续

表 6.34 – 续上页

汇编指令	32 位	16 位	汇编格式	指令描述
SRS.H	O	×	SRS32.H RZ, [LABEL]	半字符号存储指令。
SRS.W	O	×	SRS32.W RZ,[LABEL]	字符号存储指令。
ST.B	O	O	ST32.B RZ, (RX, DISP) ST16.B RZ,(RX, DISP)	字节存储指令。
ST.H	O	O	ST32.H RZ, (RX, DISP) ST16.H RZ,(RX, DISP)	半字存储指令。
ST.W	O	O	ST32.W RZ, (RX, DISP) ST16.W RZ,(RX, DISP)	字存储指令。
STM	O	×	STM32 RY-RZ, (RX)	连续多字存储指令。
STOP	O	×	STOP32	进入低功耗暂停模式指令。
STQ	O	×	STQ32 R4-R7, (RX)	连续四字存储指令。
STR.B	O	×	STR32.B RZ, (RX, RY << 0) STR32.B RZ,(RX, RY << 1) STR32.B RZ, (RX, RY << 2) STR32.B RZ, (RX, RY << 3)	寄存器移位寻址字节存储指令。
STR.H	O	×	STR32.H RZ, (RX, RY << 0) STR32.H RZ, (RX, RY << 1) STR32.H RZ, (RX, RY << 2) STR32.H RZ, (RX, RY << 3)	寄存器移位寻址半字存储指令。
STR.W	O	×	STR32.W RZ, (RX, RY << 0) STR32.W RZ, (RX, RY << 1) STR32.W RZ, (RX, RY << 2) STR32.W RZ, (RX, RY << 3)	寄存器移位寻址字存储指令。
SUBC	O	O	SUBC32 RZ, RX, RY SUBC16 RZ, RY	无符号带借位减法指令。
SUBI	O	O	SUBI32 RZ, RX, OIMM12 SUBI16 RZ, OIMM8	无符号立即数减法指令。
SUBU	O	O	SUBU32 RZ, RX, RY SUBU16 RZ, RY	无符号减法指令。
SYNC	O	×	SYNC32 IMM5	CPU 同步指令。
TRAP	O	×	TRAP32 0 TRAP32 1 TRAP32 2 TRAP32 3	无条件操作系统 陷阱指令。

下页继续

表 6.34 – 续上页

汇编指令	32 位	16 位	汇编格式	指令描述
TST	O	O	TST32 RX, RY TST16 RX, RY	零测试指令。
TSTNBZ	O	O	TSTNBZ32 RX TSTNBZ16 RX	无字节等于零寄存器测试指令。
WAIT	O	×	WAIT32	进入低功耗等待模式指令。
XOR	O	O	XOR32 RZ, RX, RY XOR16 RZ, RX	按位异或指令。
XORI	O	×	XORI32 RZ, RX, IMM12	立即数按位异或指令。
XSR	O	×	XSR32 RZ, RX, OIMM5	扩展右移指令。
XTRB0	O	×	XTRB0.32 RZ, RX	提取字节 0 并无符号扩展指令。
XTRB1	O	×	XTRB1.32 RZ, RX	提取字节 1 并无符号扩展指令。
XTRB2	O	×	XTRB2.32 RZ, RX	提取字节 2 并无符号扩展指令。
XTRB3	O	×	XTRB3.32 RZ, RX	提取字节 3 并无符号扩展指令。
ZEXT	O	×	ZEXT32 RZ, RX, MSB, LSB	位提取并无符号扩展指令。
ZEXTB	O	O	ZEXTB32 RZ, RX ZEXTB16 RZ, RX	字节提取并无符号扩展指令。
ZEXTH	O	O	ZEXTH32 RZ, RX ZEXTH16 RZ, RX	半字提取并无符号扩展指令。
<b>DSP 指令</b>				
PADD.8	O	×	PADD.8 RZ, RX, RY	8 位并行加法指令。
PADD.16	O	×	PADD.16 RZ, RX, RY	16 位并行加法指令。
PADD.(U/S)8.S	O	×	PADD.(U/S)8.S RZ, RX, RY	带饱和操作的 8 位并行 (无/有) 符号加法指令。
PADD.(U/S)16.S	O	×	PADD.(U/S)16.S RZ, RX, RY	带饱和操作的 16 位并行 (无/有) 符号加法指令。
ADD.(U/S)32.S	O	×	ADD.(U/S)32.S RZ, RX, RY	带饱和操作的 32 位 (无/有) 符号加法指令。
PSUB.8	O	×	PSUB.8 RZ, RX, RY	8 位并行减法指令。
PSUB.16	O	×	PSUB.16 RZ, RX, RY	16 位并行减法指令。
PSUB.(U/S)8.S	O	×	PSUB.(U/S)8.S RZ, RX, RY	带饱和操作的 8 位并行 (无/有) 符号减法指令。
PSUB.(U/S)16.S	O	×	PSUB.(U/S)16.S RZ, RX, RY	带饱和操作的 16 位并行 (无/有) 符号减法指令。
SUB(U/S)32.S	O	×	SUB(U/S)32.S RZ, RX, RY	带饱和操作的 32 位 (无/有) 符号减法指令。
PADDH.(U/S)8	O	×	PADDH.(U/S)8 RZ, RX, RY	8 位并行 (无/有) 符号加法求平均值指令。

下页继续

表 6.34 – 续上页

汇编指令	32 位	16 位	汇编格式	指令描述
PADDH.(U/S)16	O	×	PADDH.(U/S)16 RZ, RX, RY	16 位并行 (无/有) 符号加法求平均值指令。
ADDH.(U/S)32	O	×	ADDH.(U/S)32 RZ, RX, RY	32 位 (无/有) 符号加法求平均值指令。
PSUBH.(U/S)8	O	×	PSUBH.(U/S)8 RZ, RX, RY	8 位并行 (无/有) 符号减法求平均值指令。
PSUBH.(U/S)16	O	×	PSUBH.(U/S)16 RZ, RX, RY	16 位并行 (无/有) 符号减法求平均值指令。
SUBH.(U/S)32	O	×	SUBH.(U/S)32 RZ, RX, RY	32 位 (无/有) 符号减法求平均值指令。
PASX.16	O	×	PASX.16 RZ, RX, RY	16 位并行交叉加减指令。
PSAX.16	O	×	PSAX.16 RZ, RX, RY	16 位并行交叉减加指令。
PASX.(U/S)16.S	O	×	PASX.(U/S)16.S RZ, RX, RY	带饱和操作的 16 位并行 (无/有) 符号交叉加减指令。
PSAX.(U/S)16.S	O	×	PSAX.(U/S)16.S RZ, RX, RY	带饱和操作的 16 位并行 (无/有) 符号交叉减加指令。
PASXH.(U/S)16	O	×	PASXH.(U/S)16 RZ, RX, RY	16 位并行 (无/有) 符号交叉加减求平均值指令。
PSAXH.(U/S)16	O	×	PSAXH.(U/S)16 RZ, RX, RY	16 位并行 (无/有) 符号交叉减加求平均值指令。
ADD.64	O	×	ADD.64 RZ, RX, RY	64 位加法指令。
ADD.(U/S)64.S	O	×	ADD.(U/S)64.S RZ, RX, RY	带饱和操作的 64 位 (无/有) 符号加法指令。
SUB.64	O	×	SUB.64 RZ, RX, RY	64 位减法指令。
SUB.(U/S)64.S	O	×	SUB.(U/S)64.S RZ, RX, RY	带饱和操作的 64 位 (无/有) 符号减法指令。
ASRI.S32.R	O	×	ASRI.S32.R RZ,RX,OIMM5	带舍入操作的立即数算术右移指令。
ASR.S32.R	O	×	ASR.S32.R RZ, RX, RY	带舍入操作的算术右移指令。
LSRI.U32.R	O	×	LSRI.U32.R RZ,RX,OIMM5	带舍入操作的立即数逻辑右移指令。
LSR.U32.R	O	×	LSR.U32.R RZ, RX, RY	带舍入操作的逻辑右移指令。
LSLI.(U/S)32.S	O	×	LSLI.(U/S)32.S RZ,RX,OIMM5	带饱和操作的立即数 (无/有) 符号逻辑左移指令。
LSL.(U/S)32.S	O	×	LSL.(U/S)32.S RZ, RX, RY	带饱和操作的 (无/有) 符号逻辑左移指令。
PASRI.S16	O	×	PASRI.S16 RZ,RX,OIMM4	16 位并行立即数算术右移指令。
PASR.S16	O	×	PASR.S16 RZ, RX, RY	16 位并行算术右移指令。

下页继续

表 6.34 – 续上页

汇编指令	32 位	16 位	汇编格式	指令描述
PASRI.S16.R	O	×	PASRI.S16.R RZ,RX,OIMM4	带舍入操作的 16 位并行立即数算术右移指令。
PASR.S16.R	O	×	PASR.S16.R RZ, RX, RY	带舍入操作的 16 位并行算术右移指令
PLSRI.U16	O	×	PLSRI.U16 RZ,RX,OIMM4	16 位并行立即数逻辑右移指令。
PLSR.U16	O	×	PLSR.U16 RZ, RX, RY	16 位并行逻辑右移指令。
PLSRI.U16.R	O	×	PLSRI.U16.R RZ,RX,OIMM4	带舍入操作的 16 位并行立即数逻辑右移指令。
PLSR.U16.R	O	×	PLSR.U16.R RZ, RX, RY	带舍入操作的 16 位并行逻辑右移指令。
PLSLI.16	O	×	PLSLI.16 RZ,RX,OIMM4	16 位并行立即数逻辑左移指令。
PLSL.16	O	×	PLSL.16 RZ, RX, RY	16 位并行逻辑左移指令。
PLSLI.(U/S)16.S	O	×	PLSLI.(U/S)16.S RZ,RX,OIMM4	带饱和操作的 16 位并行立即数(无/有)符号逻辑左移指令。
PLSL.(U/S)16.S	O	×	PLSL.(U/S)16.S RZ, RX, RY	带饱和操作的 16 位并行(无/有)符号逻辑左移指令。
SEL	O	×	SEL RZ, RX, RY,RS	位选择指令。
PCMPNE.8	O	×	PCMPNE.8 RZ, RX, RY	8 位并行比较不等指令。
PCMPNE.16	O	×	PCMPNE.16 RZ, RX, RY	16 位并行比较不等指令。
PCMPHS.(U/S)8	O	×	PCMPHS.(U/S)8 RZ, RX, RY	8 位并行(无/有)符号大于等于比较指令。
PCMPHS.(U/S)16	O	×	PCMPHS.(U/S)16 RZ, RX, RY	16 位并行(无/有)符号大于等于比较指令。
PCMPLT.(U/S)8	O	×	PCMPLT.(U/S)8 RZ, RX, RY	8 位并行(无/有)符号小于比较选择指令。
PCMPLT.(U/S)16	O	×	PCMPLT.(U/S)16 RZ, RX, RY	16 位并行(无/有)符号小于比较选择指令。
PMAX.(U/S)8	O	×	PMAX.(U/S)8 RZ, RX, RY	8 位并行(无/有)符号取大值指令。
PMAX.(U/S)16	O	×	PMAX.(U/S)16 RZ, RX, RY	16 位并行(无/有)符号取大值指令。
MAX.(U/S)32	O	×	MAX.(U/S)32 RZ, RX, RY	32 位(无/有)符号取大值指令。
PMIN.(U/S)8	O	×	PMIN.(U/S)8 RZ, RX, RY	8 位并行(无/有)符号取小值指令。
PMIN.(U/S)16	O	×	PMIN.(U/S)16 RZ, RX, RY	16 位并行(无/有)符号取小值指令。
MIN.(U/S)32	O	×	MIN.(U/S)32 RZ, RX, RY	32 位(无/有)符号取小值指令。
DEXTI	O	×	DEXTI RZ, RX, RY,IMM5	立即数字截取指令。

下页继续

表 6.34 – 续上页

汇编指令	32 位	16 位	汇编格式	指令描述
DEXT	O	×	DEXT RZ, RX, RY,RS	字截取指令。
PKG	O	×	PKG RZ, RX, IMM4A,RY,OIMM4B	立即数移位打包指令。
PKGLL	O	×	PKGLL RZ, RX, RY	低半字打包指令。
PKGHH	O	×	PKGHH RZ, RX, RY	高半字打包指令。
PEXT.U8.E	O	×	PEXT.U8.E RZ,RX	8 位并行无符号扩展指令。
PEXT.S8.E	O	×	PEXT.S8.E RZ,RX	8 位并行有符号扩展指令。
PEXTX.U8.E	O	×	PEXTX.U8.E RZ,RX	8 位并行无符号交叉扩展指令。
PEXTX.S8.E	O	×	PEXTX.S8.E RZ,RX	8 位并行有符号交叉扩展指令。
NARL	O	×	NARL RZ, RX, RY	低位截取拼装指令。
NARH	O	×	NARH RZ, RX, RY	高位截取拼接指令。
NARLX	O	×	NARLX RZ, RX, RY	低位交叉截取拼装指令。
NARHX	O	×	NARHX RZ, RX, RY	高位交叉截取拼接指令。
CLIP.(U/S)32	O	×	CLIP.(U/S)32 RZ, RX,IMM5	立即数 (无/有) 符号裁剪饱和指令。
CLIP.(U/S)32	O	×	CLIP.(U/S)32 RZ, RX, RY	(无/有) 符号裁剪饱和指令。
PCLIP.(U/S)16	O	×	PCLIP.(U/S)16 RZ,RX,IMM4	16 位并行立即数 (无/有) 符号裁剪饱和指令。
PCLIP.(U/S)16	O	×	PCLIP.(U/S)16 RZ, RX, RY	16 位并行 (无/有) 符号裁剪饱和指令。
PABS.S8.S	O	×	PABS.S8.S RZ,RX	带饱和操作的 8 位并行绝对值指令。
PABS.S16.S	O	×	PABS.S16.S RZ,RX	带饱和操作的 16 位并行绝对值指令。
ABS.S32.S	O	×	ABS.S32.S RZ,RX	带饱和操作的 32 位绝对值指令。
PNEG.S8.S	O	×	PNEG.S8.S RZ,RX	带饱和操作的 8 位并行取反指令。
PNEGS16.S	O	×	PNEGS.S16.S RZ,RX	带饱和操作的 16 位并行取反指令。
NEG.S32.S	O	×	NEG.S32.S RZ,RX	带饱和操作的 32 位取反指令。
DUP.8	O	×	DUP.8 RZ, RX, INDEX	8 位操作数拷贝指令。
DUP.16	O	×	DUP.16 RZ, RX, INDEX	16 位操作数拷贝指令。
MULS.(U/S)32	O	×	MULS.(S/U)32 RZ, RX, RY	32 位 (无/有) 符号乘累减指令。
MULA.(U/S)32.S	O	×	MULA.(S/U)32.S RZ, RX, RY	带饱和操作的 32 位 (无/有) 符号乘累加指令。
MULS.(U/S)32.S	O	×	MULS.(S/U)32.S RZ, RX, RY	带饱和操作的 32 位 (无/有) 符号乘累减指令。
MUL.S32.H	O	×	MUL.S32.H RZ, RX, RY	32 位有符号乘法取高 32 位指令。

下页继续

表 6.34 – 续上页

汇编指令	32 位	16 位	汇编格式	指令描述
MUL.S32.RH	O	×	MUL.S32.RH RZ, RX, RY	带舍入操作的 32 位有符号乘法取高 32 位指令。
RMUL.S32.H	O	×	RMUL.S32.H RZ, RX, RY	32 位有符号小数乘法取高 32 位指令。
RMUL.S32.RH	O	×	RMUL.S32.RH RZ, RX, RY	带舍入操作的 32 位有符号小数乘法取高 32 位指令。
MULA.S32.HS	O	×	MULA.S32.HS RZ, RX, RY	带饱和操作的 32 位有符号乘累加取高 32 位指令。
MULS.S32.HS	O	×	MULS.S32.HS RZ, RX, RY	带饱和操作的 32 位有符号乘累减取高 32 位指令。
MULA.S32.RHS	O	×	MULA.S32.RHS RZ, RX, RY	带舍入和饱和操作的 32 位有符号乘累加取高 32 位指令。
MULS.S32.RHS	O	×	MULS.S32.RHS RZ, RX, RY	带舍入和饱和操作的 32 位有符号乘累减取高 32 位指令。
MULXL.S32	O	×	MULXL.S32 RZ, RX, RY	32 位有符号低半字不对齐乘指令。
MULXL.S32.R	O	×	MULXL.S32.R RZ, RX, RY	带舍入操作的 32 位有符号低半字不对齐乘指令。
MULXH.S32	O	×	MULXH.S32 RZ, RX, RY	32 位有符号高半字不对齐乘指令。
MULXH.S32.R	O	×	MULXH.S32.R RZ, RX, RY	带舍入操作的 32 位有符号高半字不对齐乘指令。
RMULXL.S32	O	×	RMULXL.S32 RZ, RX, RY	32 位有符号低半字不对齐小数乘指令。
RMULXL.S32.R	O	×	RMULXL.S32.R RZ, RX, RY	带舍入操作的 32 位有符号低半字不对齐小数乘指令。
RMULXH.S32	O	×	RMULXH.S32 RZ, RX, RY	32 位有符号高半字不对齐小数乘指令。
RMULXH.S32.R	O	×	RMULXH.S32.R RZ, RX, RY	带舍入操作的 32 位有符号高半字不对齐小数乘指令。
MULAXL.S32.S	O	×	MULAXL.S32.S RZ, RX, RY	带饱和操作的 32 位有符号低半字不对齐乘累加指令。
MULAXL.S32.RS	O	×	MULAXL.S32.RS RZ, RX, RY	带舍入和饱和操作的 32 位有符号低半字不对齐乘累加指令。
MULAXH.S32.S	O	×	MULAXH.S32.S RZ, RX, RY	带饱和操作的 32 位有符号高半字不对齐乘累加指令。
MULAXH.S32.RS	O	×	MULAXH.S32.RS RZ, RX, RY	带舍入和饱和操作的 32 位有符号高半字不对齐乘累加指令。

下页继续

表 6.34 – 续上页

汇编指令	32 位	16 位	汇编格式	指令描述
MULLL.S16	O	×	MULLL.S16 RZ, RX, RY	16 位有符号低半字乘法指令。
MULHH.S16	O	×	MULHH.S16 RZ, RX, RY	16 位有符号高半字乘法指令。
MULHL.S16	O	×	MULHL.S16 RZ, RX, RY	16 位有符号高低半字乘法指令。
RMULLL.S16	O	×	RMULLL.S16 RZ, RX, RY	16 位有符号低半字小数乘法指令。
RMULHH.S16	O	×	RMULHH.S16 RZ, RX, RY	16 位有符号高半字小数乘法指令。
RMULHL.S16	O	×	RMULHL.S16 RZ, RX, RY	16 位有符号高低半字小数乘法指令。
MULAHH.S16.S	O	×	MULAHH.S16.S RZ, RX, RY	带饱和操作的 16 位有符号高半字乘累加指令。
MULAHL.S16.S	O	×	MULAHL.S16.S RZ, RX, RY	带饱和操作的 16 位有符号高低半字乘累加指令。
MULALL.S16.E	O	×	MULALL.S16.E RZ, RX, RY	带扩展操作的 16 位有符号低半字乘累加指令。
MULAHH.S16.E	O	×	MULAHH.S16.E RZ, RX, RY	带扩展操作的 16 位有符号高半字乘累加指令。
MULAHL.S16.E	O	×	MULAHL.S16.E RZ, RX, RY	带扩展操作的 16 位有符号高低半字乘累加指令。
PMUL.(U/S)16	O	×	PMUL.(S/U)16 RZ, RX, RY	16 位并行 (无/有) 符号乘法指令。
PMULX.(U/S)16	O	×	PMULX.(S/U)16 RZ, RX, RY	16 位并行 (无/有) 符号交叉乘法指令。
PRMUL.S16	O	×	PRMUL.S16 RZ, RX, RY	16 位并行有符号小数乘法指令。
PRMULX.(U/S)16	O	×	PRMULX.S16 RZ, RX, RY	16 位并行 (无/有) 符号交叉小数乘法指令。
PRMUL.S16.H	O	×	PRMUL.S16.H RZ, RX, RY	16 位并行有符号小数乘法取高 16 位指令
PRMUL.S16.RH	O	×	PRMUL.S16.RH RZ, RX, RY	带舍入操作的 16 位并行有符号小数乘法取高 16 位指令。
PRMULX.S16.H	O	×	PRMULX.S16.H RZ, RX, RY	16 位并行有符号交叉小数乘法取高 16 位指令。
PRMULX.S16.RH	O	×	PRMULX.S16.RH RZ, RX, RY	带舍入操作的 16 位并行有符号交叉小数乘法取高 16 位指令。
MULCA.S16.S	O	×	MULCA.S16.S RZ, RX, RY	带饱和操作的 16 位有符号乘链加指令。
MULCAX.S16.S	O	×	MULCAX.S16.S RZ, RX, RY	带饱和操作的 16 位有符号交叉乘链加指令。

下页继续

表 6.34 – 续上页

汇编指令	32 位	16 位	汇编格式	指令描述
MULCS.S16	O	×	MULCS.S16 RZ, RX, RY	16 位有符号乘链减指令。
MULCSR.S16	O	×	MULCSR.S16 RZ, RX, RY	16 位有符号反向乘链减指令。
MULCSX.S16	O	×	MULCSX.S16 RZ, RX, RY	16 位有符号交叉乘链减指令。
MULACA.S16.S	O	×	MULACA.S16.S RZ, RX, RY	带饱和操作的 16 位有符号乘链加累加指令。
MULACAX.S16.S	O	×	MULACAX.S16.S RZ, RX, RY	带饱和操作的 16 位有符号交叉乘链加累加指令。
MULACS.S16.S	O	×	MULACS.S16.S RZ, RX, RY	带饱和操作的 16 位有符号乘链减累加指令。
MULACSR.S16.S	O	×	MULACSR.S16.S RZ, RX, RY	带饱和操作的 16 位有符号反向乘链减累加指令。
MULACSX.S16.S	O	×	MULACSX.S16.S RZ, RX, RY	带饱和操作的 16 位有符号交叉乘链减累加指令。
MULSCA.S16.S	O	×	MULSCA.S16.S RZ, RX, RY	带饱和操作的 16 位有符号乘链加累减指令。
MULSCAX.S16.S	O	×	MULSCAX.S16.S RZ, RX, RY	带饱和操作的 16 位有符号交叉乘链加累减指令。
MULACA.S16.E	O	×	MULACA.S16.E RZ, RX, RY	带扩展操作的 16 位有符号乘链加累加指令。
MULACAX.S16.E	O	×	MULACAX.S16.E RZ, RX, RY	带扩展操作的 16 位有符号交叉乘链加累加指令。
MULACS.S16.E	O	×	MULACS.S16.E RZ, RX, RY	带扩展操作的 16 位有符号乘链减累加指令。
MULACSR.S16.E	O	×	MULACSR.S16.E RZ, RX, RY	带扩展操作的 16 位有符号反向乘链减累加指令。
MULACSRX.S16.E	O	×	MULACSX.S16.E RZ, RX, RY	带扩展操作的 16 位有符号交叉乘链减累加指令。
MULSCA.S16.E	O	×	MULSCA.S16.E RZ, RX, RY	带扩展操作的 16 位有符号乘链加累减指令。
MULSCAX.S16.E	O	×	MULSCAX.S16.E RZ, RX, RY	带扩展操作的 16 位有符号交叉乘链加累减指令。
PSABSA.U8	O	×	PSABSA.U8 RZ, RX, RY	8 位并行无符号减法绝对值链加指令。
PSABSAA.U8	O	×	PSABSAA.U8 RZ, RX, RY	8 位并行无符号减法绝对值链加累加指令。
DIVSL	O	×	DIVUL RZ, RX, RY	有符号长除法指令。
DIVUL	O	×	DIVSL RZ, RX, RY	无符号长除法指令。
MULACA.S8	O	×	MULACA.S8 RZ, RX, RY	8 位并行有符号乘链加累加指令。

下页继续

表 6.34 – 续上页

汇编指令	32 位	16 位	汇编格式	指令描述
BLOOP	O	×	BLOOP RX, LABEL1, LABEL2	循环体加速指令。
LDBI.W	O	×	LDBI.W RZ, (RX)	地址自增的字加载指令。
LDBI.H	O	×	LDBI.H RZ, (RX)	地址自增的无符号半字加载指令。
LDBI.HS	O	×	LDBI.HS RZ, (RX)	地址自增的有符号半字加载指令。
LDBI.B	O	×	LDBI.B RZ, (RX)	地址自增的无符号字节加载指令。
LDBI.BS	O	×	LDBI.BS RZ, (RX)	地址自增的有符号字节加载指令。
PLDBI.D	O	×	PLDBI.D RZ, (RX)	地址自增的双字加载指令。
STBI.W	O	×	STBI.W RZ, (RX)	地址自增的字存储指令。
STBI.H	O	×	STBI.H RZ, (RX)	地址自增的半字存储指令。
STBI.B	O	×	STBI.B RZ, (RX)	地址自增的字节存储指令。
LDBIR.W	O	×	LDBIR.W RZ, (RX), RY	寄存器地址自增的字加载指令
LDBIR.H	O	×	LDBIR.H RZ, (RX), RY	寄存器地址自增的无符号半字加载指令。
LDBIR.HS	O	×	LDBIR.HS RZ, (RX), RY	寄存器地址自增的有符号半字加载指令。
LDBIR.B	O	×	LDBIR.B RZ, (RX), RY	寄存器地址自增的无符号字节加载指令。
LDBIR.BS	O	×	LDBIR.BS RZ, (RX), RY	寄存器地址自增的有符号字节加载指令。
PLDBIR.D	O	×	PLDBIR.D RZ, (RX), RY	寄存器地址自增的双字加载指令。
STBIR.W	O	×	STBIR.W RZ, (RX), RY	寄存器地址自增的字存储指令。
STBIR.H	O	×	STBIR.H RZ, (RX), RY	寄存器地址自增的半字存储指令。
STBIR.B	O	×	STBIR.B RZ, (RX), RY	寄存器地址自增的字节存储指令。

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。 \* 表示 DSP 扩展指令。

## 6.5 指令执行延迟

表 6.35: 指令执行延时表

指令类型	执行周期	备注
<b>基本指令</b>		
<b>加减法指令</b>		
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	-
DEC GT32	1	-
DEC LT32	1	-
DEC NE32	1	-
<b>逻辑操作指令</b>		
AND32/16	1	-
ANDI32/16	1	-
ANDN32	1	-
ANDNI32	1	-
OR32/16	1	-
ORI32	1	-
XOR32	1	-
XORI32	1	-
NOR32/16	1	-
NOT32/16	1	-
<b>移位指令</b>		
LSL32/16	1	-
LSLI32/16	1	-
LSLC32	1	-
LSR32/16	1	-

下页继续

表 6.35 – 续上页

指令类型	执行周期	备注
LSRI32/16	1	-
LSRC32	1	-
ASR32/16	1	-
ASRI32/16	1	-
ASRC32	1	-
ROTL32/16	1	-
ROTLI32	1	-
XSR32	1	-
<b>比较指令</b>		
CMPNE32/16	1	-
CMPNEI32/16	1	-
CMPHS32/16	1	-
CMPHSI32/16	1	-
CMPLT32/16	1	-
CMPLTI32/16	1	-
TST32/16	1	-
TSTNBZ32/16	1	-
<b>数据传输指令</b>		
MOV32/16	1	-
MOVF32	1	-
MOVT32	1	-
MOVI32/16	1	-
MOVIH32	1	-
MVCV32/16	1	-
MVC32	1	-
CLRF32	1	-
CLRT32	1	-
LRW32/16*		-
GRS32	1	-
<b>比特操作指令</b>		
BCLRI32/16	1	-
BSETI32/16	1	-
BTSTI32/16	1	-
<b>提取插入指令</b>		
ZEXT32/16	1	-
SEXT32/16	1	-
INS32	1	-

下页继续

表 6.35 – 续上页

指令类型	执行周期	备注
ZEXTB32	1	-
ZEXTH32	1	-
SEXTB32/16	1	-
SEXTH32/16	1	-
XTRB0.32	1	-
XTRB1.32	1	-
XTRB2.32	1	-
XTRB3.32	1	-
BREV32	1	-
REVB32/16	1	-
REVH32/16	1	-
<b>乘除法指令</b>		
MULT32/16	1/3	配置 DSP 或者浮点时为 1 个周期；基础配置下为 3 个周期
MULLLL.S16(MULSH)	1	-
MUL.(U/S)32	1/2/5	配置 DSP 时为 1 个周期；配置浮点时为 2 个周期；基础配置下为 5 个周期
MULA.32.L	1/2/4	配置 DSP 时为 1 个周期；配置浮点时为 2 个周期；基础配置下为 4 个周期
MULA.(U/S)32	1/2/5	配置 DSP 时为 1 个周期；配置浮点时为 2 个周期；基础配置下为 4 个周期
MULALL.S16.S	1-4	配置 DSP 时为 1 个周期；配置浮点时为 2 个周期；基础配置下为 2 个周期
DIVU32	5-36	-
DIVS32	5-36	-
<b>杂类运算指令</b>		
ABS32	1	-
FF0. 32	1	-
FF1. 32	1	-
BMASKI32	1	-
BGENR32	1	-
BGENI32	1	-
<b>分支指令</b>		
BT32/16	1	-
BF32/16	1	-
BEZ32	1	-
BNEZ32	1	-
BHZ32	1	-
BLSZ32	1	-

下页继续

表 6.35 – 续上页

指令类型	执行周期	备注
BLZ32	1	-
BHSZ32	1	-
<b>跳转指令</b>		
BR32/16	1	-
BSR32	1	-
JMPI32*	3	拆分为 LRW 及 JMP 指令
JSRI32*	3	拆分为 LRW 及 JSR 指令
JMP32/16	1	-
JSR32/16	1	-
RTS32/16	1	-
<b>立即数偏移存取指令</b>		
LD32/16.B*	1	-
LD32.BS*	1	-
LD32/16.H*	1	-
LD32.HS*	1	-
LD32/16.W*	1	-
ST32/16.B*	1	-
ST32/16.H*	1	-
ST32/16.W*	1	-
<b>寄存器偏移存取指令</b>		
LDR32.B*	3	-
LDR32.BS*	3	-
LDR32.H*	3	-
LDR32.HS*	3	-
LDR32.W*	3	-
STR32.B*	3	-
STR32.H*	3	-
STR32.W*	3	-
<b>多寄存器存取指令</b>		
LDQ32*	4	拆分为 4 条 LD.W 指令。
LDM32*	N	拆分为 N 条 LD.W 指令。
STQ32*	4	拆分为 4 条 ST.W 指令。
STM32*	N	拆分为 N 条 ST.W 指令。
PUSH32/16*	1+N	拆分为 SUB 和 N 条 ST.W 指令。
POP32/16*	2+N	拆分为 N 条 LD.W、ADD 和 RTS 指令。
IPOP16*	7	拆分为 7 条原子指令
IPUSH16*	7	拆分为 7 条原子指令

下页继续

表 6.35 – 续上页

指令类型	执行周期	备注
NIE16*	5	拆分为 5 条原子指令
NIR16*	5	拆分为 5 条原子指令
<b>符号存取指令</b>		
LRS32.B*	1	-
LRS32.H*	1	-
LRS32.W*	1	-
SRS32.B*	1	-
SRS32.H*	1	-
SRS32.W*	1	-
<b>控制寄存器操作指令</b>		
MFCR32	1	-
MTCR32	1	-
PSRSET32	1	-
PSRCLR32	1	-
<b>低功耗指令</b>		
WAIT32	1	-
DOZE32	1	-
STOP32	1	-
<b>异常返回指令</b>		
RTE32	1	-
<b>特殊功能指令</b>		
SYNC32	1	-
BKPT32	1	-
SCE32	1	-
IDLY32	1	-
TRAP32	1	-
<b>DSP 指令</b>		
<b>加减法指令</b>		
PADD.8	1	-
PADD.16	1	-
PADD.(U/S)8.S	1	-
PADD.(U/S)16.S	1	-
ADD.(U/S)32.S	1	-
PSUB.8	1	-
PSUB.16	1	-
PSUB.(U/S)8.S	1	-
PSUB.(U/S)16.S	1	-

下页继续

表 6.35 – 续上页

指令类型	执行周期	备注
SUB(U/S)32.S	1	-
PADDH.(U/S)8	1	-
PADDH.(U/S)16	1	-
ADDH.(U/S)32	1	-
PSUBH.(U/S)8	1	-
PSUBH.(U/S)16	1	-
SUBH.(U/S)32	1	-
PASX.16	1	-
PSAX.16	1	-
PASX.(U/S)16.S	1	-
PSAX.(U/S)16.S	1	-
PASXH.(U/S)16	1	-
PSAXH.(U/S)16	1	-
ADD.64	1	-
ADD.(U/S)64.S	1	-
SUB.64	1	-
SUB.(U/S)64.S	1	-
PABS.S8.S	1	-
PABS.S16.S	1	-
ABS.S32.S	1	-
PNEG.S8.S	1	-
PNEGS16.S	1	-
NEG.S32.S	1	-
PSABSA.U8	1	-
PSABSAA.U8	1	-
<b>移位指令</b>		
ASRL.S32.R	1	-
ASR.S32.R	1	-
LSRI.U32.R	1	-
LSR.U32.R	1	-
LSLI.(U/S)32.S	1	-
LSL.(U/S)32.S	1	-
PASRI.S16	1	-
PASR.S16	1	-
PASRI.S16.R	1	-
PASR.S16.R	1	-
PLSRI.U16	1	-

下页继续

表 6.35 – 续上页

指令类型	执行周期	备注
PLSR.U16	1	-
PLSRI.U16.R	1	-
PLSR.U16.R	1	-
PLSLI.16	1	-
PLSL.16	1	-
PLSL.(U/S)16.S	1	-
PLSL.(U/S)16.S	1	-
<b>比较指令</b>		
PCMPNE.8	1	-
PCMPNE.16	1	-
PCMPHS.(U/S)8	1	-
PCMPHS.(U/S)16	1	-
PCMPLT.(U/S)8	1	-
PCMPLT.(U/S)16	1	-
PMAX.(U/S)8	1	-
PMAX.(U/S)16	1	-
MAX.(U/S)32	1	-
PMIN.(U/S)8	1	-
PMIN.(U/S)16	1	-
MIN.(U/S)32	1	-
<b>特殊功能指令</b>		
DEXTI	1	-
DEXT	1	-
PKG	1	-
PKGLL	1	-
PKGHH	1	-
PEXT.U8.E	1	-
PEXT.S8.E	1	-
PEXTX.U8.E	1	-
PEXTX.S8.E	1	-
NARL	1	-
NARH	1	-
NARLX	1	-
NARHX	1	-
CLIP.(U/S)32	1	-
CLIP.(U/S)32	1	-
PCLIP.(U/S)16	1	-

下页继续

表 6.35 – 续上页

指令类型	执行周期	备注
PCLIP.(U/S)16	1	-
DUP.8	1	-
DUP.16	1	-
BLOOP*	1	-
SEL	1	-
<b>乘法指令</b>		
MULS.(U/S)32	2	-
MULA.(U/S)32.S	2	-
MULS.(U/S)32.S	2	-
MUL.S32.H	2	-
MUL.S32.RH	2	-
RMUL.S32.H	2	-
RMUL.S32.RH	2	-
MULA.S32.HS	2	-
MULS.S32.HS	2	-
MULA.S32.RHS	2	-
MULS.S32.RHS	2	-
MULXL.S32	2	-
MULXL.S32.R	2	-
MULXH.S32	2	-
MULXH.S32.R	2	-
RMULXL.S32	2	-
RMULXL.S32.R	2	-
RMULXH.S32	2	-
RMULXH.S32.R	2	-
MULAXL.S32.S	2	-
MULAXL.S32.RS	2	-
MULAXH.S32.S	2	-
MULAXH.S32.RS	2	-
MULLL.S16	2	-
MULHH.S16	2	-
MULHL.S16	2	-
RMULLL.S16	2	-
RMULHH.S16	2	-
RMULHL.S16	2	-
MULAHH.S16.S	2	-
MULAHL.S16.S	2	-

下页继续

表 6.35 – 续上页

指令类型	执行周期	备注
MULALL.S16.E	2	-
MULAHH.S16.E	2	-
MULAHL.S16.E	2	-
PMUL.(U/S)16	2	-
PMULX.(U/S)16	2	-
PRMUL.S16	2	-
PRMULX.(U/S)16	2	-
PRMUL.S16.H	2	-
PRMUL.S16.RH	2	-
PRMULX.S16.H	2	-
PRMULX.S16.RH	2	-
MULCA.S16.S	2	-
MULCAX.S16.S	2	-
MULCS.S16	2	-
MULCSR.S16	2	-
MULCSX.S16	2	-
MULACA.S16.S	2	-
MULACAX.S16.S	2	-
MULACS.S16.S	2	-
MULACSR.S16.S	2	-
MULACSX.S16.S	2	-
MULSCA.S16.S	2	-
MULSCAX.S16.S	2	-
MULACA.S16.E	2	-
MULACAX.S16.E	2	-
MULACS.S16.E	2	-
MULACSR.S16.E	2	-
MULACSRX.S16.E	2	-
MULSCA.S16.E	2	-
MULSCAX.S16.E	2	-
DIVUL	5-68	-
DIVSL	5-68	-
MULACA.S8	2	-
<b>内存存取指令</b>		
LDBI.W*	1	-
LDBI.H*	1	-
LDBI.HS*	1	-

下页继续

表 6.35 – 续上页

指令类型	执行周期	备注
LDBI.B*	1	-
LDBI.BS*	1	-
PLDBI.D*	1	-
STBI.W*	1	-
STBI.H*	1	-
STBI.B*	1	-
LDBIR.W*	1	-
LDBIR.H*	1	-
LDBIR.HS*	1	-
LDBIR.B*	1	-
LDBIR.BS*	1	-
PLDBIR.D*	2	-
STBIR.W*	1	-
STBIR.H*	1	-
STBIR.B*	1	-

---

**注解：**\* 表示内存存取相关指令，指令完成周期取决于总线延时或者高速缓存命中情况，表中数值所列为最快情况。

---

# 第七章 内存保护

## 7.1 内存保护单元简介

在受保护的系统中，主要有两类资源的访问需要被监视：存储器系统和外围设备。内存保护单元负责对存储器系统（包括外围设备）的访问合法性进行检查，其主要功能包括：

1. 判定当前工作模式下 CPU 是否具备对内存地址的读/写访问权限。
2. 获取该访问地址的附加属性，包括安全属性等。

内存保护单元支持 N 个表项（N 为 1-8 硬件可配置），可对 N 个区域的访问权限和属性进行设置。每个表项通过 0-7 的号码来标识和索引。内存保护单元的表项内容如 图 7.1 所示。

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

图 7.1: 内存保护单元表项

其中：

**EN:** 表示该区域是否生效；

**Base Address:** 表示该区域的起始地址；

**Size:** 表示该区域的大小；

**S:** 表示该区域的安全属性；

**NX:** 表示区域取指的可执行性；

**AP:** 表示该区域的访问权限。

具体的每个字段的属性参考相关系统控制寄存器。

## 7.2 相关系统控制寄存器

### 7.2.1 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保护使能，大小端模式，以及系统和处理器的时钟比。

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

图 7.2: 高速缓存配置寄存器

**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 power on reset 时被配置且不能在之后改变。

该域目前没有任何功能，只供软件查询。

**BE-Endian 模式:**

- 当 BE 为 0 时，小端；
- 当 BE 为 1 时，大端。

BE 在 power on reset 时被配置且不能在之后改变，CPU 上有对应引脚引出。

**MP-内存保护设置位:**

MP 用来设置 MPU 是否有效，如 表 7.1 所示。

表 7.1: E804 内存保护设置

MP	功能
00	MPU 无效。
01	MPU 有效。
10	MPU 无效。
11	MPU 有效。

**7.2.2 可高缓和访问权限配置寄存器 (CAPR, CR<19,0>)**

CAPR 的各位如 图 7.3 所示。

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

图 7.3: 可高缓和访问权限配置寄存器

**NX0~NX7-不可执行属性设置位:**

- 当 NX 为 0 时，该区为可执行区；
- 当 NX 为 1 时，该区为不可执行区。

**注解：** 当处理器取指访问到不可执行的区域时，会出现访问错误异常。

#### S0~S7-安全属性设置位：

- 当 S 为 0 时，该区为非安全区；
- 当 S 为 1 时，该区为安全区。

#### AP0~AP7-访问权限设置位：

表 7.2: 访问权限设置

AP	超级用户权限	普通用户权限
00	不可访问	不可访问
01	读写	不可访问
10	读写	只读
11	读写	读写

### 7.2.3 保护区控制寄存器 (PACR, CR<20,0>)

PACR 的各位如 图 7.4 所示。

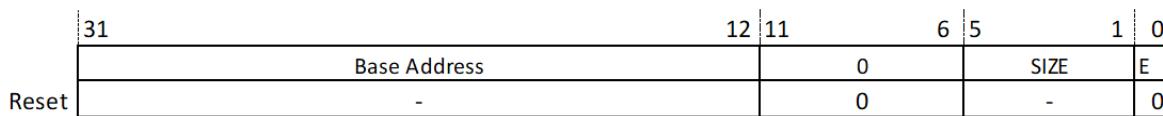


图 7.4: 保护区控制寄存器

Base Address-保护区地址的基地址：

- 该寄存器指出了保护区地址的基地址，但写入的基地址必须与设置的页面大小对齐，例如设置页面大小为 8M，CR<20,0>[22:12] 必须为 0，各页面的具体要求见 表 7.3。
- 保护区大小从 4KB 到 4GB，它可以通过公式：保护区大小 =  $2^{(Size+1)}$  计算得到。Size 可设置的范围为 01011 到 11111，其它一些值都会造成不可预测的结果。

表 7.3: 保护区大小配置和其对基址要求

Size	保护区大小	对基地址的要求
00000—01010	保留	-
01011	4KB	没有要求
01100	8KB	CR<20,0>.bit[12]=0
01101	16KB	CR<20,0>.bit[13:12] =0
01110	32KB	CR<20,0>.bit[14:12] =0
01111	64KB	CR<20,0>.bit[15:12] =0
10000	128KB	CR<20,0>.bit[16:12] =0
10001	256KB	CR<20,0>.bit[17:12] =0
10010	512KB	CR<20,0>.bit[18:12] =0
10011	1MB	CR<20,0>.bit[19:12] =0
10100	2MB	CR<20,0>.bit[20:12] =0
10101	4MB	CR<20,0>.bit[21:12] =0
10110	8MB	CR<20,0>.bit[22:12] =0
10111	16MB	CR<20,0>.bit[23:12] =0
11000	32MB	CR<20,0>.bit[24:12] =0
11001	64MB	CR<20,0>.bit[25:12] =0
11010	128MB	CR<20,0>.bit[26:12] =0
11011	256MB	CR<20,0>.bit[27:12] =0
11100	512MB	CR<20,0>.bit[28:12] =0
11101	1GB	CR<20,0>.bit[29:12] =0
11110	2GB	CR<20,0>.bit[30:12] =0
11111	4GB	CR<20,0>.bit[31:12] =0

E-保护区有效设置：

- 当 E 为 0 时，保护区无效；
- 当 E 为 1 时，保护区有效。

#### 7.2.4 保护区选择寄存器 (PRSR, CR<21,0>)

PRSR 用来选择当前操作的保护区，其各位如 图 7.5 所示：

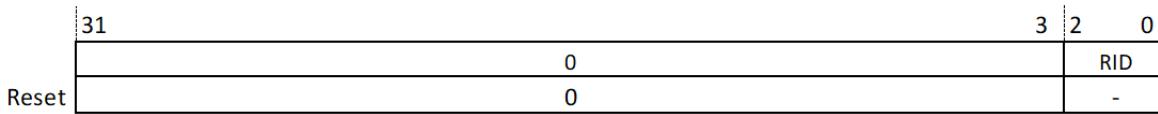


图 7.5: 保护区选择寄存器

RID-保护区索引值：

- RID 表示所选择的对应的保护区，如 000 表示第 0 保护区。

### 7.3 内存访问处理

内存保护单元对 CPU 内存访问的处理流程及结果如 表 7.4 所示。

表 7.4: 内存访问处理流程及结果

内存保护单元使能情况	内存保护单元命中情况	处理结果
不使能	不关心	所有对内存的访问都视为： 超级用户模式/普通用户模式均可读可写； 访问属性为不可缓冲。
使能	不命中	处理器将产生访问错误异常。
使能	命中	如果访问命中一个保护区，将以该保护区的访问权限与属性为准；如果访问地址命中多个保护区，则以索引号最高的保护区的访问权限与属性为准。 该地址的访问权限结合当前处理器的工作模式，判定该访问是否合法。如果访问符合该区域的访问权限，则内存保护单元允许本次内存操作，并提供该访问地址的访问属性；如果访问请求不被允许，则内存保护单元将产生一个访问错误异常并中断处理器的执行。
<hr/> <b>注解：</b> 访问地址可能命中多个保护区，此时认为命中索引号最高的保护区。 <hr/>		

# 第八章 片上高速缓存

## 8.1 高速缓存简介

E804 提供了硬件可配置的高速缓存，其主要特征如下：

- 高速缓存大小硬件可配置，支持 2KB/4KB/8KB/16KB；
- 4 路组相联，缓存行大小为 16 个字节；
- 每次访问的最大宽度为 4 字节，支持字节/半字/字访问；
- 物理地址索引，物理地址标记；
- 写策略同时支持写直模式和写回模式；
- 高速缓存采用先进先出（FIFO）的替换策略；
- 支持对整个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。

当内存访问不可高缓区域时，直接旁路片上高速缓存，通过总线访问片外存储器，加快不可高缓存区域的访问速度。当内存访问可高缓区域时，首先访问片上高速缓存；在高速缓存缺失的情况下再访问片外存储器（写直模式下的写访问在高速缓存命中时，也需要回写片外存储器）。

## 8.2 相关系统控制寄存器

Cache 提供一组 32-bit 的寄存器，包括 Cache 的使能、缓存行的无效和高速缓存区的配置等。各个寄存器地址空间如 表 8.1 所示。

表 8.1: Cache 控制寄存器定义

地址	名称	类型	初始值	描述
0xE000F000	CER	读/写	0	高速缓存使能寄存器。
0xE000F004	CIR	读/写	0	高速缓存无效寄存器。
0xE000F008	CRCR0	读/写	0	0 号可高缓区配置寄存器。
0xE000F00C	CRCR1	读/写	0	1 号可高缓区配置寄存器。
0xE000F010	CRCR2	读/写	0	2 号可高缓区配置寄存器。
0xE000F014	CRCR3	读/写	0	3 号可高缓区配置寄存器。
0xE000F018~ 0xE000FFF3	-	-	-	保留。
0xE000FFF4	CPFCR	读/写	0	高速缓存性能分析控制寄存器。
0xE000FFF8	CPFATR	读/写	0	高速缓存访问次数寄存器。
0xE000FFFC	CPFMTR	读/写	0	高速缓存缺失次数寄存器。

### 8.2.1 高速缓存使能寄存器 (CER)

高速缓存使能寄存器用于配置高速缓存 (Cache) 使能以及高速缓存的工作模式。

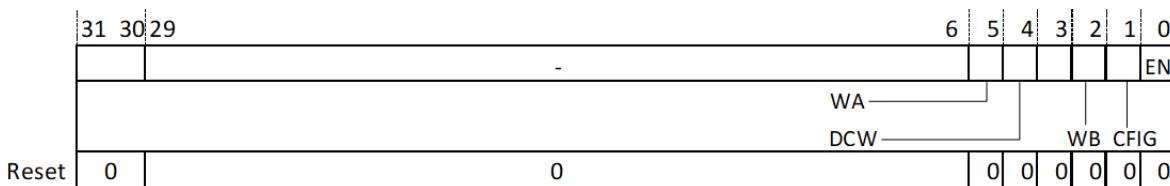


图 8.1: 高速缓存使能寄存器

WA 高速缓存写分配有效设置位:

- 0: 高速缓存为 write non-allocate 模式;
- 1: 高速缓存为 write allocate 模式;
- write allocate 模式只能在写回模式下使用, 否则结果不可预知。

DCW-Cache 可写属性配置位:

- 0: 高速缓存不可写;
- 1: 高速缓存可写。

WB-Cache 写回模式配置位:

- 0: 高速缓存为写直模式;
- 1: 高速缓存为写回模式。

CFIG-Cache 属性配置位:

- 0: 指令与数据高速缓存;

- 1: 指令高速缓存。

EN-Cache 使能位:

- 当 EN 为 0 时, 高速缓存处于关闭状态;
- 当 EN 为 1 时, 高速缓存处于工作状态。

### 8.2.2 高速缓存无效寄存器 (CIR)

高速缓存无效寄存器用于控制高速缓存的清除和无效, 可支持对特定高速缓存行操作和对整个高速缓存操作两种方式。

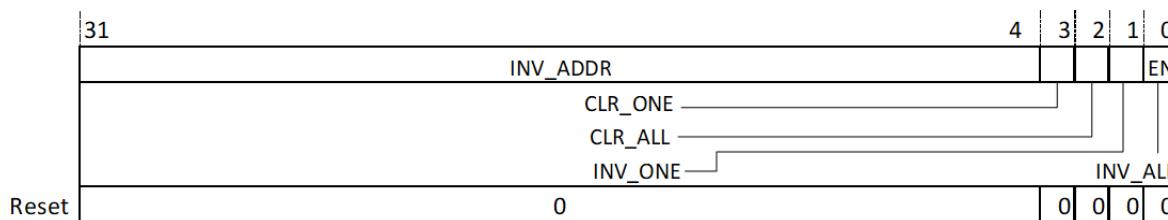


图 8.2: 高速缓存无效寄存器

INV\_ADDR-缓存行地址:

- 表征需要被无效的缓存行地址。

CLR\_ONE-单条缓存行清除设置位:

- 当 CLR\_ONE 为 1 时, 选中的缓存行若标记为脏, 将被回写到片外存储器中。

CLR\_ALL-整个高速缓存清除设置位:

- 当 CLR\_ALL 为 1 时, 标记为脏的所有缓存行, 将被回写到片外存储器中。

INV\_ONE-单条缓存行无效设置位:

- 当 INV\_ONE 为 1 时, 无效选中的缓存行。

INV\_ALL-整个高速缓存无效设置位:

- 当 INV\_ALL 为 1 时, 无效高速缓存中所有缓存行。

---

**注解:** 任何按行操作不可与任何全 cache 操作同时进行。例如: 往 CIR 中写 32' b1111, INV\_ONE 和 CLR\_ONE 操作将被屏蔽。

---

### 8.2.3 可高缓区配置寄存器 0~3 (CRCR)

可高缓区配置寄存器用于配置可高缓 (Cacheable) 的地址空间范围, 提供四个可高缓区配置寄存器, 用来配置四块不同的可高缓区域。

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

图 8.3: 可高缓区配置寄存器

Base Address-可高缓区基址地址:

- 可高缓区大小 4KB 到 4GB 可配置, 该域指出了可高缓区地址的高位, 例如设置页面大小为 8M, CRCR[22:12] 必须为 0, 不同大小的区各页面的具体要求见 表 8.2。Size-可高缓区大小:
- 可高缓区大小可以通过公式: 可高缓区大小 =  $2^{(Size+1)}$  计算得到。因此 Size 可设置的范围为 01011 到 11111, 其它一些值都会造成不可预测的结果。

表 8.2: 可高缓区大小配置和其对基址要求

Size	高速缓存区大小	对基地址的要求
00000—01010	保留	-
01011	4KB	没有要求
01100	8KB	CRCR.bit[12]=0
01101	16KB	CRCR.bit[13:12]=0
01110	32KB	CRCR.bit[14:12]=0
01111	64KB	CRCR.bit[15:12]=0
10000	128KB	CRCR.bit[16:12]=0
10001	256KB	CRCR.bit[17:12]=0
10010	512KB	CRCR.bit[18:12]=0
10011	1MB	CRCR.bit[19:12]=0
10100	2MB	CRCR.bit[20:12]=0
10101	4MB	CRCR.bit[21:12]=0
10110	8MB	CRCR.bit[22:12]=0
10111	16MB	CRCR.bit[23:12]=0
11000	32MB	CRCR.bit[24:12]=0
11001	64MB	CRCR.bit[25:12]=0
11010	128MB	CRCR.bit[26:12]=0
11011	256MB	CRCR.bit[27:12]=0
11100	512MB	CRCR.bit[28:12]=0
11101	1GB	CRCR.bit[29:12]=0
11110	2GB	CRCR.bit[30:12]=0
11111	4GB	CRCR.bit[31:12]=0

EN-可高缓区有效位:

- 当 EN 为 0 时, 可高缓区无效;
- 当 EN 为 1 时, 可高缓区有效。

### 8.2.4 高速缓存性能分析控制寄存器 (CPFCR)

高速缓存性能分析控制寄存器用于使能高速缓存性能分析功能 (Cache Profiling) 和重置性能分析相关计数器。

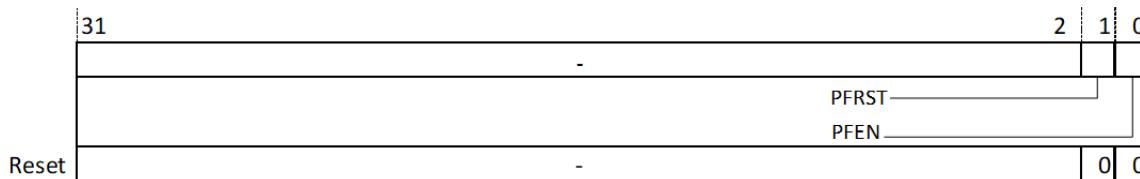


图 8.4: 高速缓存性能分析控制寄存器

PFRST-Cache profiling 复位位:

- 当 PERST 为 1 时，将 CPFATR/CPFMTR 复位为 0。

PFEN-Cache profiling 使能位:

- 当 PFEN 为 0 时，高速缓存性能分析功能处于关闭状态；
- 当 PFEN 为 1 时，高速缓存性能分析功能处于工作状态。

### 8.2.5 高速缓存访问次数寄存器 (CPFATR)

高速缓存访问次数寄存器用于记录访问高速缓存的次数。

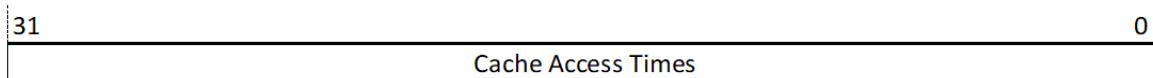


图 8.5: 高速缓存访问次数寄存器

Cache Access Times -高速缓存访问次数:

- 记录高速缓存被访问次数，在高速缓存性能分析使能情况下，每 (256) 次高速缓存访问将使寄存器数值加 1。

### 8.2.6 高速缓存缺失次数寄存器 (CPFMTR)

高速缓存缺失次数寄存器用于记录访问高速缓存中缺失的次数。

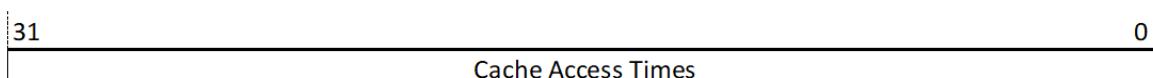


图 8.6: 高速缓存缺失次数寄存器

Cache Miss Times -访问高速缓存缺失次数:

- 记录访问高速缓存中的缺失次数，在高速缓存性能分析使能情况下，每 (256) 次高速缓存缺失将使寄存器数值加 1。

---

**注解：** 建议所有高速缓存配置操作都在其关闭情况下进行。

---

# 第九章 总线矩阵与总线接口

## 9.1 简介

E804 实现了多总线接口，分别包括系统总线、指令总线和数据总线。

总线矩阵为处理器内部请求访问外部总线接口提供了互联功能。总线矩阵与 CPU 内部请求及总线接口的连接关系如 图 9.1 所示。总线矩阵根据内存访问的地址仲裁总线接口类型，将处理器内部访问分发到系统总线、指令总线或数据总线上。

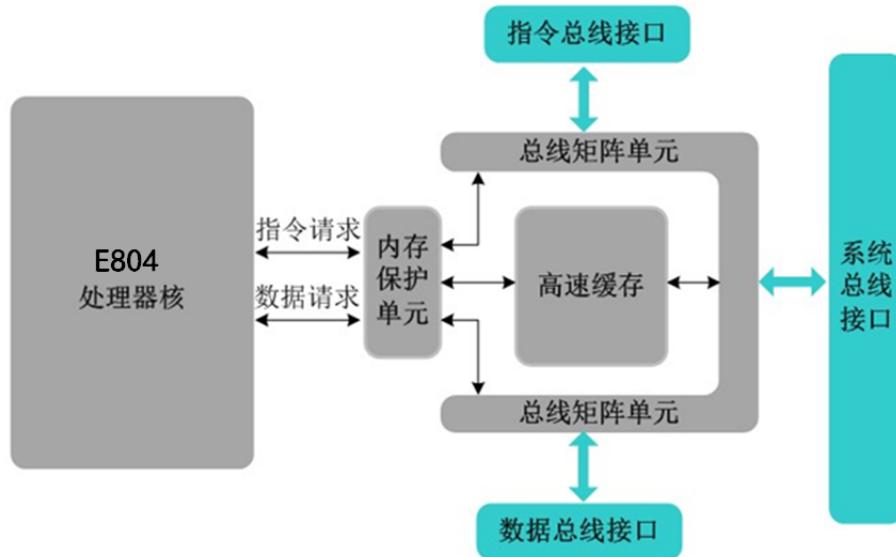


图 9.1: E804 总线矩阵

处理器内部的取指访问和数据访问拥有相同的总线访问权限，可以访问所有总线接口。为了解决同一时钟周期取指访问和数据访问竞争同一总线接口的问题，总线矩阵也负责请求的优先级判断。当取指请求和数据请求竞争同一总线接口时，数据请求拥有更高的优先级。

E804 多总线接口的基本信息和可配置性如 表 9.1 所示。

表 9.1: 多总线接口的基本信息和可配置性

总线接口	可配置性	总线协议	时序方式
系统总线	固定包含, 协议可配	AHB	直接输出
系统总线	固定包含, 协议可配	AHB-Lite	直接输出
指令总线	固定包含	AHB-Lite	直接输出
数据总线	固定包含	AHB-Lite	直接输出

另外, E804 通过提供一组接口信号 (pad\_bmu\_xahbl\_base 和 pad\_bmu\_xahbl\_mask) , 支持指令总线和数据总线的基地址和空间大小硬件可配置。其中, pad\_bmu\_xahbl\_base 指定了指令总线/数据总线的基地址。pad\_bmu\_xahbl\_base 为位宽为 20 的信号, 其表示的真实基址需要将低位 0 扩展, 即 {pad\_bmu\_xahbl\_base[19:0],12' b0}。pad\_bmu\_xahbl\_mask 指定了不同地址空间下对地址对齐的需求。指令总线或者数据总线的地址空间 4KB 到 4GB 可配置, 例如设置地址空间大小为 8M, pad\_bmu\_xahbl\_base[10:0] 必须为 11' b0, pad\_bmu\_xahbl\_mask[19:0] 必须为 20' b1111 1111 1000 0000 0000, 不同大小的地址空间具体要求见 表 9.2。

表 9.2: 指令总线和数据总线对基地址和地址对齐的要求

地址空间大小	对 pad_bmu_xahbl_base 的要求	对 pad_bmu_xahbl_mask 的要求
4KB	没有要求	bit[19:0]=20' b1111 1111 1111 1111 1111
8KB	bit[0] =1' b0	bit[19:0]=20' b1111 1111 1111 1111 1110
16KB	bit[1:0] =2' b0	bit[19:0]=20' b1111 1111 1111 1111 1100
32KB	bit[2:0] =3' b0	bit[19:0]=20' b1111 1111 1111 1111 1000
64KB	bit[3:0] =4' b0	bit[19:0]=20' b1111 1111 1111 1111 0000
128KB	bit[4:0] =5' b0	bit[19:0]=20' b1111 1111 1111 1110 0000
256KB	bit[5:0] =6' b0	bit[19:0]=20' b1111 1111 1111 1100 0000
512KB	bit[6:0] =7' b0	bit[19:0]=20' b1111 1111 1111 1000 0000
1MB	bit[7:0] =8' b0	bit[19:0]=20' b1111 1111 1111 0000 0000
2MB	bit[8:0] =9' b0	bit[19:0]=20' b1111 1111 1110 0000 0000
4MB	bit[9:0] =10' b0	bit[19:0]=20' b1111 1111 1100 0000 0000
8MB	bit[10:0] =11' b0	bit[19:0]=20' b1111 1111 1000 0000 0000
16MB	bit[11:0] =12' b0	bit[19:0]=20' b1111 1111 0000 0000 0000
32MB	bit[12:0] =13' b0	bit[19:0]=20' b1111 1110 0000 0000 0000
64MB	bit[13:0] =14' b0	bit[19:0]=20' b1111 1100 0000 0000 0000
128MB	bit[14:0] =15' b0	bit[19:0]=20' b1111 1000 0000 0000 0000
256MB	bit[15:0] =16' b0	bit[19:0]=20' b1111 0000 0000 0000 0000
512MB	bit[16:0] =17' b0	bit[19:0]=20' b1110 0000 0000 0000 0000
1GB	bit[17:0] =18' b0	bit[19:0]=20' b1100 0000 0000 0000 0000
2GB	bit[18:0] =19' b0	bit[19:0]=20' b1000 0000 0000 0000 0000
4GB	bit[19:0] =20' b0	bit[19:0]=20' b0000 0000 0000 0000 0000

## 9.2 系统总线接口

### 9.2.1 特点

E804 的系统总线接口可以配置为支持 AMBA2.0 AHB 协议（此时请参考 AMBA 2.0 规格说明—AMBA™ Specification Rev 2.0），也可以配置为支持 AMBA3.0 AHB-Lite 协议（此时请参考 AMBA 3.0 规格说明—AMBA3 AHB-Lite Protocol Specification Rev 1.0）。

系统总线接口的基本特点包括：

- 支持 AMBA2.0 AHB 或 AMBA3.0 AHB-Lite 总线协议，可由用户配置；
- 仅支持直接输出 (non-Flop-out)；
- 处理器和系统时钟频率比必须为 1: 1；

### 9.2.2 协议内容

#### 9.2.2.1 支持传输类型

考虑到 E804 的应用领域及成本，系统总线接口只实现了 AHB/AHB-Lite 协议中的部分内容。

在 AHB-Lite 协议下，作为主设备，总线接口支持的传输类型为：

- HBURST 支持 SINGLE 和 WRAP4 传输，其它突发类型均不支持；
- HTRANS 支持 IDLE 和 NONSEQ 和 SEQ，其它传输类型均不支持；
- HSIZE 支持字、字节和半字传输，其它传输大小不支持；
- HWRITE 支持读和写操作。

在 AHB 协议下，作为主设备，总线接口支持的传输类型为：

- HBURST 支持 SINGLE 和 WRAP4 传输，其它突发类型均不支持；
- HTRANS 支持 IDLE、NONSEQ 和 SEQ，其它传输类型均不支持；
- HSIZE 支持字、字节和半字传输，其它传输大小不支持；
- HWRITE 支持读和写操作。

#### 9.2.2.2 支持响应类型

在 AHB 及 AHB-Lite 协议下，总线接口接受从设备的响应类型为：

- HREADY 支持 Ready 和 Not Ready；
- HRESP 支持 OKAY 和 ERROR，其它响应类型不支持。

### 9.2.3 不同总线响应下的行为

表 9.3 列出了总线上出现不同总线响应时 CPU 的行为。

表 9.3: 总线异常处理

HREADY	HRESP	结果
不关心	ERROR	访问错误，总线结束传输并处理访问错误。
High	OKEY	传输结束。
Low	OKEY	插入等待状态。

### 9.2.4 AHB 协议的接口信号

表 9.4: AHB 协议接口信号

信号名	I/O	Reset	定义
biu_pad_haddr[31:0]	O	-	地址总线： 32 位地址总线。
biu_pad_hwdata[31:0]	O	-	写数据总线： 32 位写数据总线。
biu_pad_hburst[2:0]	O	-	突发传输指示信号： 000: SINGLE; 001: INCR; 010: WRAP4。 E804 支持 SINGLE 和 WRAP4 突发传输。
biu_pad_hsize[2:0]	O	-	传输宽度指示信号： 000: byte; 001: halfword; 010: word。
biu_pad_htrans[1:0]	O	00	传输类型表示信号： 00: IDLE; 01: BUSY; 10: NONSEQ; 11: SEQ。 E804 支持 IDLE、NONSEQ 和 SEQ 传输类型。
biu_pad_hwrite	O	0	读写表示信号： 1: 指示是写总线传输； 0: 指示是读总线传输。

下页继续

表 9.4 – 续上页

信号名	I/O	Reset	定义
biu_pad_hprot[3:0]	O	-	保护控制信号： ***0: 取指令； ***1: 数据访问； **0*: 用户访问； **1*: 超级用户访问； *0**: Not bufferable； *1**: bufferable； 0***: Not cacheable； 1***: cacheable。
biu_pad_hbusreq	O	0	总线请求信号： 指示 CPU 请求总线的使用权。
biu_pad_hlock	O	0	锁定总线信号： 当 lock 申明时，CPU 独占有总线控制权，没有其他的 busmaster 可以占有总线。
pad_biu_hrdata[31:0]	I	-	读数据总线： 32 位读数据总线。

表 9.5: AHB 协议接口信号

信号名	I/O	Reset	定义
pad_biu_hready	I	-	传输完成指示信号： 有效时指示当前传输已完成，CPU 将总线置于待命状态。
pad_biu_hgrant	I	-	总线占用指示信号： 有效时指示 CPU 当前已占用总线。
pad_biu_hresp[1:0]	I	-	传输应答信号： 00: OKAY； 01: ERROR； 10: RETRY； 11: SPLIT。 E804 仅支持 OKAY 和 ERROR 响应类型。

### 9.2.5 AHB-Lite 协议的接口信号

表 9.6: AHB-Lite 协议接口信号

信号名	I/O	Reset	定义
biu_pad_haddr[31:0]	O	-	地址总线： 32 位地址总线。
biu_pad_hwdata[31:0]	O	-	写数据总线： 32 位写数据总线。
biu_pad_hburst[2:0]	O	-	突发传输指示信号： 000: SINGLE; 001: INCR; 010: WRAP4。 E804 支持 SINGLE 和 WRAP4 突发传输。
biu_pad_hsize[2:0]	O	-	传输宽度指示信号： 000: byte; 001: halfword; 010: word。
biu_pad_htrans[1:0]	O	00	传输类型表示信号： 00: IDLE; 01: BUSY; 10: NONSEQ; 11: SEQ。 E804 支持 IDLE、NONSEQ 和 SEQ 传输类型。
biu_pad_hwrite	O	0	读写表示信号： 1: 指示是写总线传输； 0: 指示是读总线传输。
biu_pad_hprot[3:0]	O	-	保护控制信号： ***0: 取指令； ***1: 数据访问； **0*: 用户访问； **1*: 超级用户访问； *0**: Not bufferable； *1**: bufferable； 0***: Not cacheable； 1***: cacheable。
pad_biu_hrdata[31:0]	I	-	读数据总线： 32 位读数据总线。

表 9.7: AHB-Lite 协议接口信号

信号名	I/O	Reset	定义
pad_biu_hready	I	-	传输完成指示信号： 有效时指示当前传输已完成，CPU 将总线置于待命状态。
pad_biu_hresp[1:0]	I	-	传输应答信号： 00: OKAY; 01: ERROR; 10: RETRY; 11: SPLIT。 E804 仅支持 OKAY 和 ERROR 响应类型。

## 9.3 指令总线接口

### 9.3.1 特点

E804 的指令总线只支持 AMBA3.0 AHB-Lite 协议，可参考 AMBA 3.0 规格说明- AMBA3 AHB-Lite Protocol Specification Rev 1.0。

指令总线接口的基本特点有：

- 与 AMBA3.0 AHB-Lite 总线协议兼容；
- 仅支持直接输出（Non-Flop-out）方式；

### 9.3.2 协议内容

#### 9.3.2.1 支持传输类型

考虑到 E804 的应用领域及成本，指令总线接口只实现了 AHB-Lite 协议中的部分内容。

在 AHB-Lite 协议下，作为主设备，总线接口支持的传输类型为：

- HBURST 只支持 SINGLE 传输，其它突发类型均不支持；
- HTRANS 只支持 IDLE 和 NONSEQ，其它传输类型均不支持；
- HSIZE 支持字、字节和半字传输，其它传输大小不支持；
- HWRITE 支持读和写操作。

#### 9.3.2.2 支持响应类型

在 AHB-Lite 协议下，总线接口接受从设备的响应类型为：

- HREADY 支持 Ready 和 Not Ready；
- HRESP 支持 OKAY 和 ERROR，其它响应类型不支持。

### 9.3.3 不同总线响应下的行为

表 9.8 列出了总线上出现不同总线响应时 CPU 的行为。

表 9.8: 总线异常处理

HREADY	HRESP	结果
不关心	ERROR	访问错误，总线结束传输并处理访问错误。
High	OKEY	传输结束。
Low	OKEY	插入等待状态。

### 9.3.4 指令总线接口信号

表 9.9: 指令总线接口信号

信号名	I/O	Reset	定义
iahbl_pad_haddr[31:0]	O	-	地址总线： 32 位地址总线。
iahbl_pad_hburst[2:0]	O	-	突发传输指示信号： 000: SINGLE; 001: INCR; 010: WRAP4。 E804 仅支持 SINGLE 突发传输。
iahbl_pad_hprot[3:0]	O	-	保护控制信号： ***0: 取指令； ***1: 数据访问； **0*: 用户访问； **1*: 超级用户访问； *0**: Not bufferable； *1**: bufferable； 0***: Not cacheable； 1***: cacheable。
iahbl_pad_hsize[2:0]	O	-	传输宽度指示信号： 000: byte； 001: halfword； 010: word。
iahbl_pad_htrans[1:0]	O	00	传输类型表示信号： 00: IDLE； 01: BUSY； 10: NONSEQ； 11: SEQ。 E804 仅支持 IDLE 和 NONSEQ 传输类型。
iahbl_pad_hwdata[31:0]	O	-	写数据总线： 32 位写数据总线。
iahbl_pad_hwrite	O	0	读写表示信号： 1: 指示是写总线传输； 0: 指示是读总线传输。
pad_iahbl_hrdata[31:0]	I	-	读数据总线： 32 位读数据总线。

表 9.10: 指令总线接口信号

信号名	I/O	Reset	定义
pad_iahbl_hready	I	-	传输完成指示信号： 有效时指示当前传输已完成，CPU 将总线置于待命状态。
pad_iahbl_hresp	I	-	传输应答信号： 0: OKAY; 1: ERROR。
pad_bmu_iahbl_base[19:0]	I	-	IAHB-Lite 基址控制信号，上电复位之后需固定
pad_bmu_iahbl_mask[19:0]	I	-	IAHB-Lite 地址对齐控制信号，上电复位之后需固定

## 9.4 数据总线接口

### 9.4.1 特点

E804 的数据总线只支持 AMBA3.0 AHB-Lite 协议，可参考 AMBA 3.0 规格说明- AMBA3 AHB-Lite Protocol Specification Rev 1.0。

数据总线接口的基本特点有：

- 与 AMBA3.0 AHB-Lite 总线协议兼容；
- 仅支持直接输出（Non-Flop-out）方式。

### 9.4.2 协议内容

#### 9.4.2.1 支持传输类型

考虑到 E804 的应用领域及成本，数据总线接口只实现了 AHB-Lite 协议中的部分内容。

在 AHB-Lite 协议下，作为主设备，总线接口支持的传输类型为：

- HBURST 支持 SINGLE 传输，其它突发类型均不支持；
- HTRANS 只支持 IDLE 和 NONSEQ，其它传输类型均不支持；
- HSIZE 支持字、字节和半字传输，其它传输大小不支持；
- HWRITE 支持读和写操作；

#### 9.4.2.2 支持响应类型

在 AHB-Lite 协议下，总线接口接受从设备的响应类型为：

- HREADY 支持 Ready 和 Not Ready；
- HRESP 支持 OKAY 和 ERROR，其它响应类型不支持。

#### 9.4.3 不同总线响应下的行为

表 9.11 列出了总线上出现不同总线响应时 CPU 的行为：

表 9.11: 总线异常处理

HREADY	HRESP	结果
不关心	ERROR	访问错误，总线结束传输并处理访问错误。
High	OKEY	传输结束。
Low	OKEY	插入等待状态。

### 9.4.4 数据总线接口信号

表 9.12: 数据总线接口信号

信号名	I/O	Reset	定义
dahbl_pad_haddr[31:0]	O	-	地址总线： 32 位地址总线。
dahbl_pad_hburst[2:0]	O	-	突发传输指示信号： 000: SINGLE; 001: INCR; 010: WRAP4。 E804 仅支持 SINGLE 突发传输。
dahbl_pad_hprot[3:0]	O	-	保护控制信号： ***0: 取指令； ***1: 数据访问； **0*: 用户访问； **1*: 超级用户访问； *0**: Not bufferable； *1**: bufferable； 0***: Not cacheable； 1***: cacheable。
dahbl_pad_hsize[2:0]	O	-	传输宽度指示信号： 000: byte； 001: halfword； 010: word。
dahbl_pad_htrans[1:0]	O	00	传输类型表示信号： 00: IDLE； 01: BUSY； 10: NONSEQ； 11: SEQ。 E804 仅支持 IDLE 和 NONSEQ 传输类型。
dahbl_pad_hwdata[31:0]	O	-	写数据总线： 32 位写数据总线。
dahbl_pad_hwwrite	O	0	读写表示信号： 1: 指示是写总线传输； 0: 指示是读总线传输。
pad_dahbl_hrdata[31:0]	I	-	读数据总线： 32 位读数据总线。

表 9.13: 数据总线接口信号

信号名	I/O	Reset	定义
pad_dahbl_hready	I	-	传输完成指示信号： 有效时指示当前传输已完成，CPU 将总线置于待命状态。
pad_dahbl_hresp	I	-	传输应答信号： 0: OKAY; 1: ERROR。
pad_bmu_dahbl_base[19:0]	I	-	DAHB-Lite 基址控制信号，上电复位之后需固定
pad_bmu_dahbl_mask[19:0]	I	-	DAHB-Lite 地址对齐控制信号，上电复位之后需固定

## 9.5 指令与数据的访问顺序

一旦 CPU 配置了高速缓存，则所有不可高缓的数据都会绕过 cache 直接从总线矩阵访问对应的总线，如 图 9.2 中的通道 1 所示；

可高缓的数据会首先访问 cache，如果命中则直接从 cache 中取回数据，如果不命中，则会由 cache 通过总线矩阵向总线发起请求，具体如 图 9.2 中的通道 2 所示。

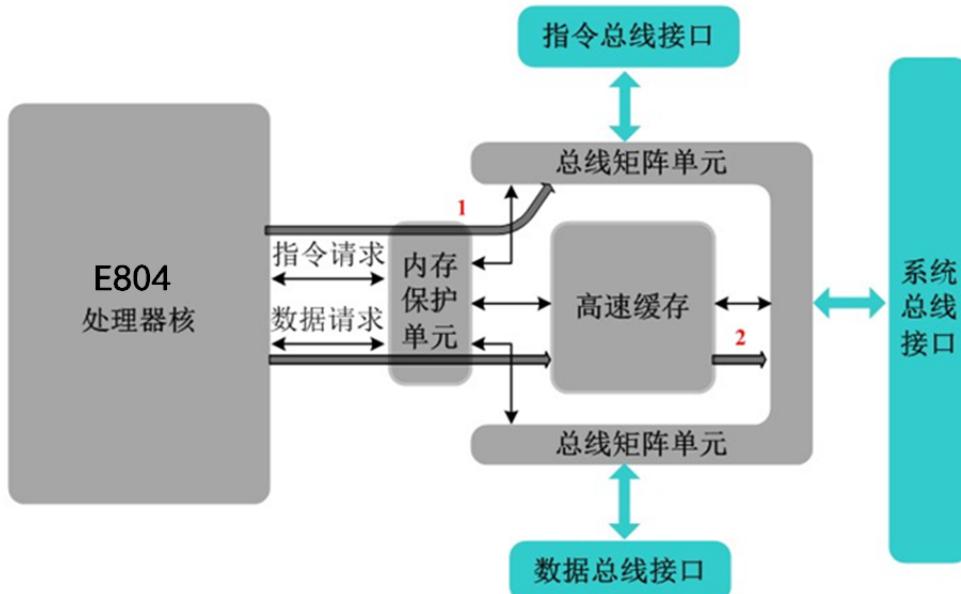


图 9.2: 访问外总线顺序

# 第十章 调试接口

## 10.1 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。

为了满足低成本的应用需求，节省 CPU 外部引脚，CSKY 定义了一套调试接口，JTAG2 接口。JTAG2 调试接口包括 JTAG2 通信协议，JTAG2 接口控制器。E804 支持 2 线制 JTAG 协议，调试接口使用 JTAG2 协议与外部的调试器通信。

调试接口的主要特性如下：

- 支持 2 线制 JTAG 接口；
- 非侵入式获取 CPU 状态；
- 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 复位之后或在普通用户模式下进入调试模式；
- 可使用 TCIP 接口访问调试寄存器资源，详见调试手册。

C-SKY 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 10.1 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。

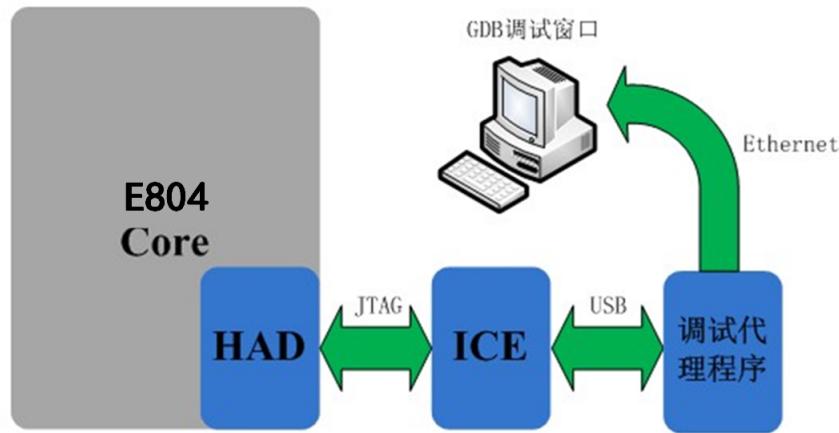


图 10.1: 调试接口在整个 CPU 调试环境中的位置

## 10.2 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 10.1 列出了与调试相关的接口信号。

表 10.1: 调试模块与外部的接口信号

信号名	方向
(sysio_pad_dbg_b)	输出
pad_had_jdb_req_b	输入
had_pad_jdb_ack_b	输出
pad_had_jtg_tap_en	输入
had_pad_jtg_tap_on	输出
had_pad_jdb_pm[1:0]	输出
pad_had_jtg_tclk	输入
pad_had_jtg_trst_b	输入
pad_had_jtg_tms_i	输入
had_pad_jtg_tms_o	输出
had_pad_htg_tms_oe	输出

### 1. biu\_pad\_dbg\_b (sysio\_pad\_dbg\_b)

低电平表示 CPU 处于调试模式中。

### 2. pad\_had\_jdb\_req\_b 与 had\_pad\_jdb\_ack\_b

pad\_had\_jdb\_req\_b 信号是让 CPU 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 CPU 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 CPU 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 CPU 进入调试状态之后，had\_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。

### 3. 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。

### 4. 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 10.2 所示。

表 10.2: had\_pad\_jdb\_pm 指示当前 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。

### 5. 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 CPU 核之间的正常工作。

### 6. 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。

## 7. 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 状态，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。

# 第十一章 工作模式与转换

E804 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是由 SOC 设计者定义的。本章将详细介绍 CPU 的工作模式以及模式之间的转换。

如 图 11.1 所示，E804 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 sysio\_pad\_jdb\_pm[1:0] 信号得到。

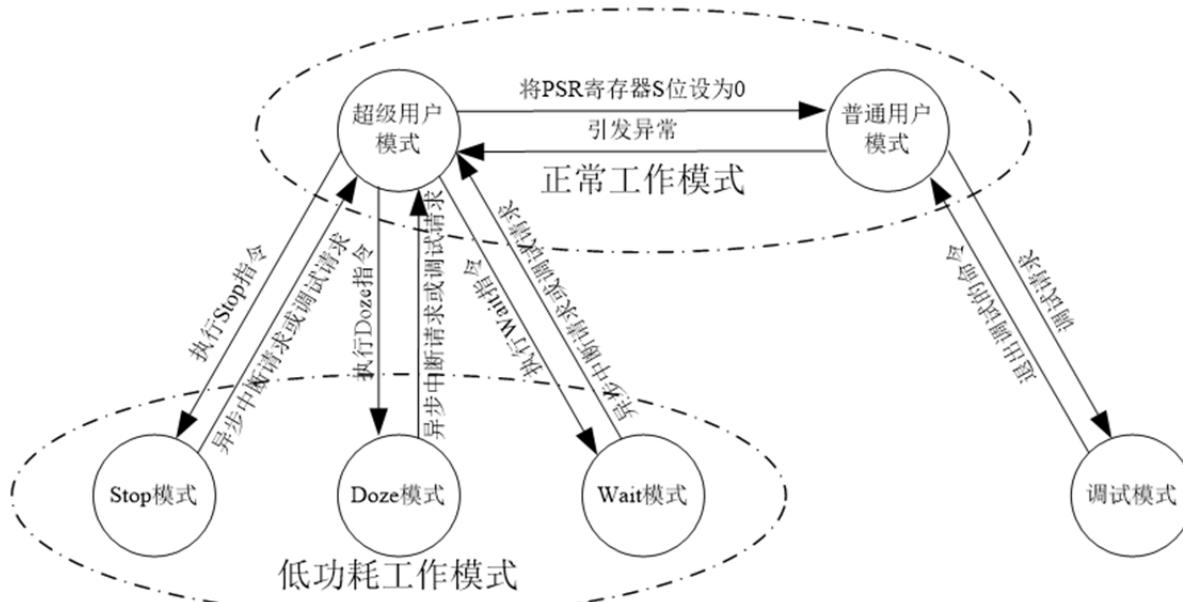


图 11.1: CPU 的各种工作状态示意图

## 11.1 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。

## 11.2 低功耗模式

当 CPU 执行完低功耗指令 (STOP、DOZE、WAIT) 之后，CPU 将进入低功耗模式。三条指令执行的过程是，CPU 执行到低功耗指令后将等待前面的所有指令执行完，然后完成低功耗指令，同时根据低功耗指令类型拉起 sysio\_pad\_lpmd\_b[1:0] 信号，停止执行指令并冻结流水线，关掉内部时钟。

只有异步中断请求 (pad\_sysio\_intraw\_b 和 pad\_sysio\_finraw\_b 信号) 或者调试请求才能使 CPU 退出低功耗模式，然后 CPU 从进入低功耗模式的低功耗指令处继续执行后续指令。当前的 CPU 工作于哪种低功耗模式可以通过查询信号线 sysio\_pad\_lpmd\_b[1:0] 来得到，每种模式具体对应的低功耗场景由 SOC 设计者决定。

## 11.3 调试模式

### 11.3.1 调试模式

CPU 进入调试模式后将停止取指和执行指令，处于一种等待状态，该状态下 CPU 只执行 HAD 输入的指令，通过 HAD 输入指令可以查询和修改 CPU 的状态，进而进行调试。

### 11.3.2 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：

- HAD 的外部输入信号 (pad\_had\_jdb\_req\_b) 可使处理器异步进入调试模式；
- 当 E804 HCR 的 IDRE 位有效时，HAD 的外部输入信号 biu\_had\_sdb\_req\_b(pad\_biu\_dbgreq\_b) 被 CPU 同步后请求处理器同步进入调试模式；
- 当 E804 HCR 的 ADR 位有效时，处理器直接进入调试模式；
- 当 E804 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 E804 CSR 的 FDB 位有效时，处理器在执行 bkpt 指令进入调试模式；
- 当 E804 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 E804 存储硬件断点进入调试模式。

处理器执行完当前的指令，保存流水线的信息，然后进入调试模式。当处理器处于低功耗模式时，通过设置 HCR 中的 ADR 以及 DR 的调试请求，可以使得处理器退出低功耗并进入调试模式。进入调试模式后，CPU 停止执行当前指令，等待用户通过调试接口输入有效的指令用于执行或退出调试模式。

### 11.3.3 退出调试模式

如果 E804 HACR 的 GO、EX 位被置为 1，同时 R/W 为 0（写操作），RS 选择的是 WBBR、PSR、PC、IR、CSR 或者 Bypass 寄存器，则执行指令时 CPU 退出调试模式，进入正常工作模式。

---

**注解：**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，上述寄存器中的值必须是刚进入调试模式时保存过的值。

---

# 第十二章 浮点处理单元

## 12.1 概述

### 12.1.1 简介

浮点单元作为 E804 的可配置硬件单元，旨在提升 E804 面向浮点应用的处理能力。E804 浮点单元提供了一种低成本、高性能的硬件浮点实现。

浮点单元支持 IEEE-754 浮点标准中的单精度浮点运算，实现了 16 个单精度浮点寄存器。在系统软件支持下，E804 可实现对双精度浮点运算的支持。

### 12.1.2 特点

浮点单元的体系结构与编程模型的主要特点如下：

- 完全兼容 ANSI/IEEE Std 754 浮点标准（系统软件支持下）；
- 仅支持单精度浮点运算；
- 支持向零舍入、向正无穷舍入、向负无穷舍入和就近舍入四种舍入方式；
- 支持浮点异常的陷入与非陷入两种处理模式；
- 支持浮点异常的精确处理；
- 支持浮点硬件除法与开方。

浮点单元的微体系结构的主要特点如下：

- 16 个独立的单精度浮点寄存器；
- 单发射结构，每个周期处理一条浮点算术指令；
- 支持浮点算术指令的按序发射、按序执行、按序回写；
- 包含三条独立的执行流水线，分别是浮点 ALU、浮点乘法与浮点除法开方；
- 优化的执行延时技术，除浮点除法开方指令外，均可在 1-2 个时钟周期执行完毕；
- 基于运算部件复用的成本优化技术；

- 基于门控时钟与数据通路隔离的功耗优化技术。

## 12.2 微体系结构

### 12.2.1 浮点单元介绍

浮点单元主要包括如下三个执行子单元：

- 算术逻辑 (FALU) 单元
- 乘法 (FMULU) 单元
- 除法和开方 (FDSU) 单元

算术逻辑单元 (FALU) 单元负责加减、比较、转换、取反、取绝对值、寄存器传输等指令的执行，运算结果均在 1-2 个周期完成。

乘法单元 (FMULU) 负责乘法、乘取反等操作，乘法结果在 2 个周期完成。

除法开方单元 (FDSU) 实现了除法、开方以及求倒数操作，每个周期产生 1 比特结果，最长需要 28 个周期完成。

## 12.3 编程模型

### 12.3.1 简介

E804 浮点单元实现了 ANSI/IEEE Std 754 浮点标准中的单精度部分

- 支持浮点立即数的加载，提高浮点立即数产生效率；

### 12.3.2 数据格式

浮点单元仅支持单精度数据格式。浮点寄存器的值可以通过浮点加载指令、浮点常数加载指令、浮点寄存器传送指令三种方式进行赋值。在浮点寄存器中的数据和内存中的数据拥有一样的格式。任何的传输或者载入都不会改变这种格式。

记录每个寄存器所存储的数据类型是由程序本身所保证。由此硬件将不对寄存器做任何有关于数据类型的判断。举例说明，即使在浮点寄存器中存储的数据是整数，一条浮点指令也只会把这个数据当成是单精度的数据来处理。也就是说硬件对数据的解释仅仅取决于执行的指令。

#### 12.3.2.1 整形数据格式

浮点单元支持有符号或者无符号的 32 位整数。在浮点寄存器中的整数格式和 E804 主流水线寄存器中或内存中一致。

### 12.3.2.2 单精度数据格式

图 12.1 表示单精度浮点数的区域分割：

31   30	23   22	0
S	Exponent	Fraction

图 12.1：单精度数据格式

单精度数据格式包括：

- 符号位 (S): bit[31];
- 偏置的指数域: The biased exponent  $e=E+bias$ , here bias=127, bits[30:23];
- 小数域: The fraction, bits[22:0]。

### 12.3.3 浮点寄存器描述

E804 浮点单元拥有独立的浮点寄存器，由 16 个 32 位单精度寄存器组成。

图 12.2 所示为包含浮点单元的 E804 寄存器资源图。相比整形流水线，浮点单元增加了浮点寄存器，可在普通模式与特权模式下被访问。增加了三个系统寄存器，仅在特权模式下被访问。

#### 12.3.3.1 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：

- FMTVRL 浮点寄存器写传送指令。
- FMFVRL 浮点寄存器读传送指令。

#### 12.3.3.2 维护寄存器精度的一致

浮点寄存器可以存储单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，可能是两种数据类型的任何一种。

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读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。

### 12.3.4 系统寄存器

浮点单元的相关系统寄存器包括 3 个：浮点版本寄存器 FID(CR<0,2>)、浮点控制寄存器 FCR(CR<1,2>)、浮点异常状态寄存器 FESR(CR<2,2>)。浮点版本寄存器记录浮点单元的实现与版本信



图 12.2: 编程模式下寄存器资源

息。浮点控制寄存器用于对浮点的异常处理与执行进行配置，而浮点异常状态寄存器则记录执行过程中的浮点异常信息。

表 12.1 列出了浮点单元的三个系统寄存器。

表 12.1: 浮点系统寄存器一览

寄存器	CR	访问方式	访问模式
浮点版本寄存器,FID (FPU ID Register)	CR<0,2>	只读	特权模式
浮点控制寄存器,FCR (FPU Control Register)	CR<1,2>	读/写	特权模式
浮点异常状态寄存器 FESR (Exception Status Register)	CR<2,2>	读/写	特权模式

#### 12.3.4.1 浮点系统寄存器的读写方法

使用 MTCSR 和 MFCSR 指令来对浮点系统寄存器进行读写操作。由于浮点系统寄存器存放在 2 号协处理器，因此 MTCSR 和 MFCSR 要对 2 号协处理器进行读写操作，具体的寄存器索引方式为：浮点版本寄存器 CR<0,2>，浮点控制寄存器 CR<1,2>，浮点异常状态寄存器 CR<2,2>。

写浮点系统寄存器的指令为 MTCSR Rx, CR<y,2>，用通用寄存器 Rx 的值改写 y 号浮点系统寄存器的值。

读浮点系统寄存器的指令为 MFCSR Rx, CR<y,2>，读 y 号浮点系统寄存器的值并改写通用寄存器 Rx。

#### 12.3.4.2 浮点版本寄存器 (FPU ID Register, FID)

浮点版本寄存器是一个只读寄存器，用来识别浮点单元的版本信息。

图 12.3 是浮点版本寄存器的位域：

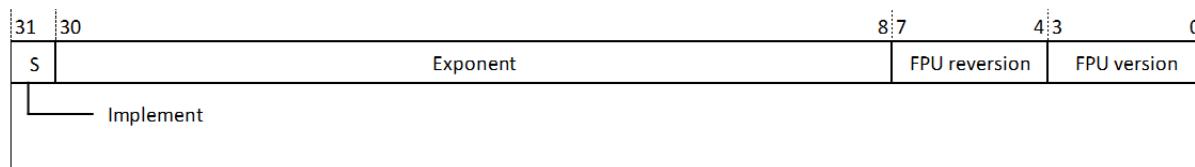


图 12.3: 浮点版本寄存器

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。

表 12.2 描述了 FID 寄存器各个位的定义：

表 12.2: 浮点版本寄存器描述

位	功能	读/写	描述
[31]	Implement	只读	浮点单元的实现方式： 1: 实现了硬件浮点单元； 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。
[30:8]	Reserved	只读	全 0。
[7:4]	FPU reversion	只读	浮点单元的升级信息。
[3:0]	FPU version	只读	浮点单元的版本信息。

#### 12.3.4.3 浮点控制寄存器 (FPU Control Register, FCR)

浮点控制寄存器可读写，负责对浮点单元的异常护理与执行方式进行配置。浮点控制寄存器只能在超级用户模式时被访问。FCR 的保留位会被初始化为 0。在希望修改其中的一些位的时候，为了保证不希望修改的位不会被错误地修改，代码中推荐使用先读取，再修改读取的值，最后写回 FCR 的方式来操作。如果没有遵从这种方式操作很可能在之后的程序中发生不可预期的错误。

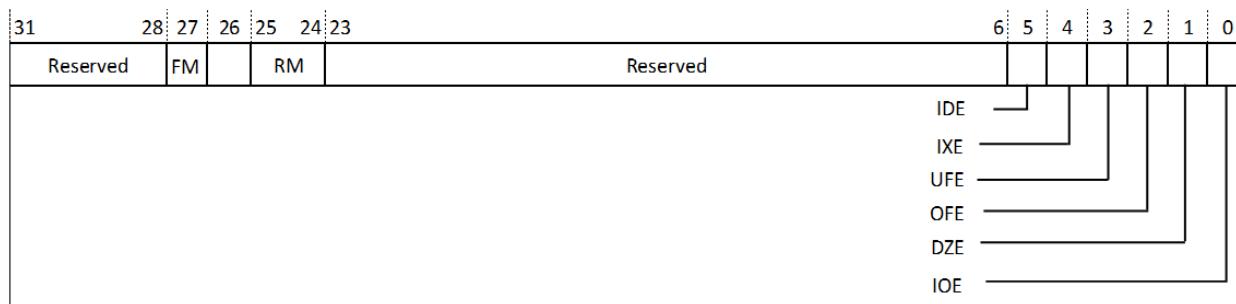


图 12.4: 浮点控制寄存器

复位值: 0x00000000。

FCR 寄存器中各个位的定义如 表 12.3 所示。

表 12.3: 浮点控制寄存器描述

位	功能	读/写	描述
[31:28]	保留	只读	全 0

下页继续

表 12.3 – 续上页

[27]	FM	读/写	<p>这个域决定在输入非规格数异常使能位/运算结果下溢出异常使能位(IDE/OFE)未被开启的情况下, 对非规格化数(Denormalized Number)的处理方式。</p> <p>1: 将非规格化数刷成有符号最小正规数(Signed Minimal Normal Number)。</p> <p>0: 将非规格化数刷成有符号零(Signed Zero)。</p>
[25:24]	RM	读/写	<p>这个域决定浮点计算的舍入模式。</p> <p>00: 舍入到最接近值(Round to the Nearest)</p> <p>01: 向0舍入(Round towards 0)</p> <p>10: 向正无穷舍入(Round towards +∞)</p> <p>11: 向负无穷舍入(Round towards -∞)</p>
[23:6]	保留	只读	全0
[5]	IDE	读/写	<p>输入非规格化数异常使能位(IDE, Input Denormalized trap Enable bit)。</p> <p>1: 使能。此时, 若遇到输入非规格数, 处理器将终止程序执行并抛出非规格数输入异常。</p> <p>0: 不使能。此时, 处理器将根据FM指示位, 对输入非规格数进行处理, 并继续运算。</p>
[4]	IXE	读/写	<p>非精确异常使能位(IXE, IneXact trap Enable bit)。</p> <p>1: 使能。此时, 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。</p> <p>0: 不使能。此时, 处理器将根据RM指示位, 对运算结果进行舍入处理, 并继续运算。</p>
[3]	UFE	读/写	<p>下溢异常使能位(UFE, UnderFlow trap Enable bit)。</p> <p>1: 使能。此时, 若运算结果下溢出, 处理器将终止程序执行并抛出下溢出异常, 不回写任何结果。</p> <p>0: 不使能。此时, 处理器将根据FM指示位, 对下溢出结果进行处理, 并继续运算。</p>
[2]	OFE	读/写	<p>上溢异常使能位(OFE, OverFlow trap Enable bit)。</p> <p>1: 使能。此时, 若运算结果上溢出, 处理器将终止程序执行并抛出上溢出异常, 不回写任何结果。</p> <p>0: 不使能。此时, 处理器将根据RM指示位, 对上溢出结果进行处理, 并继续运算。</p>
[1]	DZE	读/写	<p>除0异常使能位(DZE, Division by Zero trap Enable bit)。</p> <p>1: 使能。此时, 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。</p> <p>0: 不使能。此时, 处理器将以正/负无穷浮点数作为运算结果, 并继续运算。</p>

下页继续

表 12.3 – 续上页

[0]	IOE	读/写	非法操作异常使能位 (IOE, Invalid Operation trap Enable bit)。 1: 使能。此时, 若发生非法操作数, 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。 0: 不使能。此时, 处理器将以 QNaN 作为运算结果, 并继续运算。
-----	-----	-----	--

#### 12.3.4.4 浮点异常状态处理器 (FPU Exception Status Register, FESR)

FESR 提供浮点的异常状态信息。各个位指示出发生的浮点异常类型。FESR 只能在超级用户模式下访问。

FESR 各个区域如 图 12.5 所示:

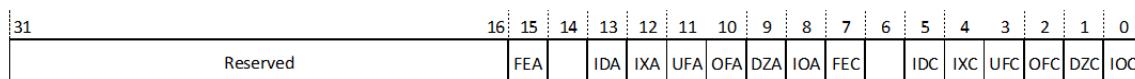


图 12.5: 浮点异常状态寄存器

表 12.4 显示浮点异常状态寄存器的位定义。

表 12.4: 浮点异常状态寄存器描述

位	功能	读/写	描述
[29:16]	保留	只读	全 0。
[15]	FEA	读/写	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。
[14]	保留	只读	全 0
[13]	IDA	读/写	输入非规格数异常累积位 (IDA, Input Denormalized Accumulative bit)。 在发生输入非规格数异常且 IDE 位为 0 时, 该位被置为 1。
[12]	IXA	读/写	非精确异常累积位 (IXA, IneXact Accumulative bit)。 在发生非精确异常且 IXE 位为 0 时, 该位被置为 1。
[11]	UFA	读/写	下溢异常累积位 (UFA, UnderFlow Accumulative bit)。 在发生下溢异常且 UFE 位为 0 时, 该位被置为 1。
[10]	OFA	读/写	上溢异常累积位。(OFA, OverFlow Accumulative bit)。 在发生上溢异常且 OFE 位为 0 时, 该位被置为 1。
[9]	DZA	读/写	除 0 异常累积位。(DZA, Division by Zero Accumulative bit)。 在发生除 0 异常且 DZE 位为 0 时, 该位被置为 1。
[8]	IOA	读/写	非法操作数异常累积位。(IOA, Invalid Operation Accumulative bit)。 在发生非法操作数异常且 IOE 位为 0 时, 该位被置为 1。
[7]	FEC	读/写	当前浮点指令发生任意浮点异常时, 这个位将被置为 1, 否则被清 0。
[6]	保留	只读	全 0.

下页继续

表 12.4 – 续上页

[5]	IDC	读/写	输入非规格数异常指示位 (IDC, Input Denormalized Current bit)。 当前浮点指令发生非规格数输入异常时，该位被置为 1，否则被清 0。
[4]	IXC	读/写	非精确异常指示位 (IXC, InExact Current bit)。 当前浮点指令发生非精确异常时，该位被置为 1，否则被清 0。
[3]	UFC	读/写	下溢出异常指示位 (UFC, UnderFlow Current bit) 当前浮点指令发生下溢异常时，该位被置为 1，否则被清 0。
[2]	OFC	读/写	上溢出异常指示位 (OFC, OverFlow Current bit) 当前浮点指令发生上溢异常时，该位被置为 1，否则被清 0。
[1]	DZC	读/写	除 0 异常指示位 (DZC, Division by Zero Current bit)。 当前浮点指令发生除 0 异常时，该位被置为 1，否则被清 0。
[0]	IOC	读/写	非法操作异常指示位 (IOC, Invalid Operation Current bit)。 当前浮点指令发生非法操作数异常时，该位被置为 1，否则被清 0。

### 12.3.5 数据大小端

数据大小端是相对于存储器数据存储的格式提出的，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。

E804 浮点的内存访问单元与整形流水线的内存访问单元一样，支持小端格式和 V1 版本大端（CSKY CPU 设计了 V1 和 V2 两个版本的大小端）格式，且不支持非对齐访问。在小端模式下，无论存取指令的访问大小为字节/半字/字，始终以最低位字节数据对应于地址的最低位；在 V1 版本大端模式下，则需要根据存取指令的访问大小（字节/半字/字）确定数据。由于浮点存取指令的访问是以字为单位，因此在小端和 V1 版本大端模式下浮点存取指令的存储访问模式是一样的。

表 12.5: 小端/大端存储访问模式

访问大小	访问地址	小端	V1 版本大端
字	A	D3D2D1D0	D3D2D1D0
半字	A	D1D0	D3D2
半字	A+2	D3D2	D1D0
字节	A	D0	D3
字节	A+1	D1	D2
字节	A+2	D2	D1
字节	A+3	D3	D0

## 12.4 异常处理

本章描述了浮点运算的异常处理。包括以下内容。

- 浮点异常介绍；

- 非规格数输入异常 (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);
- 异常优先级。

### 12.4.1 浮点异常介绍

浮点指令会产生 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：

- 非规格数输入异常 (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)。

### 12.4.2 非规格数输入异常

当一个浮点操作数是非规格数 (Denormalized Number) 时，IDC 标志位 (FESR[5]) 置 1。此时，具体的异常处理方式又分异常使能与异常不使能两种情况。

#### 12.4.2.1 异常使能

IDE 位为 1 时，非规格数输入异常使能。当输入操作数为非规格数时，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。发生异常指令的源寄存器和目的寄存器都不会被改变。

### 12.4.2.2 异常不使能

IDE 位为 0 时，非规格数输入异常不使能。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。FEA (FESR[15])、IDA (FESR[13])、FEC (FESR[7]) 和 (IDCFESR[5]) 会被置 1。

### 12.4.3 非法操作异常

当运算的一个操作数是非法时，将会触发非法操作异常。表 12.6 列出了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又分异常使能与异常不使能两种情况。

表 12.6: 非法操作异常

指令	非法操作异常
所有浮点运算指令 (FABS, FNEG, FMOV 除外)	操作数中包含 sNaN
FADDS	(+INF) + (-INF) 或 (-INF) + (+INF)
FSUBS	(+INF) - (+INF) 或 (-INF) - (-INF)
FCMPHSS FCMPLTS FCMPZHSS FCMPZLSS	任意的 NaN 操作数
FSTOSI	任意的 NaN 操作数或者结果发生上溢出
FSTOUI	任意的 NaN 操作数或者结果发生上/下溢出
FMULS FNMULS	(Zero $\times$ ±INF) 或 (±INF $\times$ Zero) :sup:a
FDIVS	Zero/zero 或 INF/INF. <sup>a</sup>
FSQRTS	操作数小于 0

---

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

---

#### 12.4.3.1 异常使能

IOE 位 (FCR[0]) 为 1 时，非法操作异常使能。此时如果发生非法操作异常，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。指令的源寄存器和目的寄存器都不会被改变。

### 12.4.3.2 异常不使能

IOE 位为 0 时，浮点运算指令（除了整型转换操作之外）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。对于格式转换指令，当发生非法操作时的默认结果如表 4-2 所示。

表 12.7: 非法转换结果的默认结果

指令	输入值	默认输出结果
FSTOSI	NaN	0x00000000
	+INF or $x \cdot 2^{31}$	0x7fffffff
	-INF or $x < -2^{31}$	0x80000000
FSTOUI	NaN	0x00000000
	+INF or $x \cdot 2^{32}$	0xffffffff
	-INF or $x < 0$	0x00000000

### 12.4.4 除以零异常

当被除数是一个确定的非零数而除数是 0 时，将产生除以零异常。当除数是一个非规格数，而 FM 位是 0，此时除数将会被刷为 0，同样也会引起除以零异常。

#### 12.4.4.1 异常使能

如果 DZE 位 (FCR[1]) 置 1，则除以零异常使能。此时如果发生除以零异常，处理器将终止程序执行并抛出 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。指令的源寄存器和目的寄存器都不会被改变。

#### 12.4.4.2 异常不使能

如果 DZE 位置 0，则 Division by Zero 异常不使能。此时如果发生该异常，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。

### 12.4.5 上溢异常

E804 的浮点单元将会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。因为发生上溢出异常时，结果将不能被准确的表示出来，因此上溢出总是和非精确异常同时产生。

#### 12.4.5.1 异常使能

如果 OFE 位 (FCR[2]) 为 1，则上溢异常使能。此时如果运算结果发生上溢异常，处理器将会终止程序执行并抛出 30 号异常，同时设置 FEC (FESR[7])、IXC (FESR[4]) 和 OFC (FESR[2]) 为 1。指令的源

寄存器和目的寄存器都不会被改变。

#### 12.4.5.2 异常不使能

如果 OFE 位为 0，则上溢异常不使能。此时如果发生该异常，浮点单元根据不同的舍入模式，取一个带符号的无穷大数或者一个最大有限数作为运算结果写入目标寄存器。FEA (FESR[15])、IXA (FESR[12])、OFA (FESR[10])、FEC (FESR[7])、IXC (FESR[4]) 和 OFC (FESR[2]) 会被置 1。

#### 12.4.6 下溢异常

E804 浮点单元将会在运算结果舍入之后检测是否发生下溢异常。当结果小于相应格式的最小有限数时，将会产生下溢异常。下溢异常总是和非精确异常同时产生。

##### 12.4.6.1 异常使能

如果 UFE 位 (FCR[3]) 为 1，则下溢异常使能。此时如果运算结果发生下溢异常，处理器将终止程序执行并抛出 30 号异常，同时置 FEC (FESR[7])、IXC (FESR[4]) 和 UFC (FESR[3]) 为 1。指令的源寄存器和目的寄存器都不会被改变。

##### 12.4.6.2 异常不使能

如果 UFE 位为 0，则下溢异常不使能。此时如果发生该异常，当 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。同时，FEA (FESR[15])、IXA (FESR[12])、UFA (FESR[11])、FEC (FESR[7])、IXC (FESR[4]) 和 UFC (FESR[3]) 将被置位以供软件查询。

#### 12.4.7 非精确异常

浮点运算结果的指数域或尾数域都会导致非精确异常，具体定义请参考浮点协议标准 (ANSI/IEEE Std 754)。以下情况均会导致非精确异常：

- 舍入前后的结果不一致；
- 发生上溢异常；
- 发生下溢异常。

---

**注解：** 非精确异常在浮点运算中会频繁出现。如果使能 IXE 位，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。

---

#### 12.4.7.1 异常使能

如果 IXE 位 (FCR[4]) 为 1，则非精确异常使能。如果浮点运算产生非精确异常，处理器将终止程序执行并抛出 30 号异常，同时置 FEC (FESR[7]) 和 IXC (FESR[4]) 为 1，以供异常服务程序查询。指令的源寄存器和目的寄存器都不会被改变。

#### 12.4.7.2 异常不使能

如果 IXE 位为 0，则非精确异常不使能。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。

### 12.4.8 异常优先级

浮点指令在运算过程中，可能会同时产生几个异常。会同时产生的异常如下：

- 非规格数输入异常与其它的 5 类异常；
- 上溢异常与非精确异常；
- 下溢异常与非精确异常。

以一条浮点乘法指令为例，其中一个操作数为非规格数，另外一个操作数为最小规格数，非规格数输入异常不使能 (IDE 为 0)，FM=1。这条指令将产生三种异常，分别是非规格数输入异常、下溢异常与非精确异常。浮点异常寄存器将记录浮点指令处理过程中发生的所有异常，供软件查询与处理。

浮点异常的优先级关系如下（推荐浮点异常程序使用）：

1. 非规格数输入异常；
2. 非法操作异常；
3. 除以零异常；
4. 上溢/下溢异常；
5. 非精确异常。

### 12.4.9 特殊指令的异常

对于乘累加/乘累减指令而言，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。

## 12.5 指令集

## 12.5.1 浮点指令列表

本节主要介绍 E804 浮点指令集。E804 的浮点指令均为 32 位指令，按照指令实现的功能划分，可以分为：

- 数据运算指令；
- 传输类指令；
- 内存存取指令。

### 12.5.1.1 数据运算指令

表 12.8: 单精度数据运算指令列表

FSTOSI	单精度浮点数转换为有符号整数
FSTOUI	单精度浮点数转换为无符号整数
FSITOS	有符号整数转换为单精度浮点数
FUITOS	无符号整数转换为单精度浮点数
FCMPZHSS	单精度浮点大于等于 0
FCMPZLSS	单精度浮点小于等于 0
FCMPZNES	单精度浮点不等于 0
FCMPZUOS	单精度浮点单操作数是否非数
FCMPHSS	单精度浮点大于等于
FCMPLTS	单精度浮点小于
FCMPNES	单精度浮点不等于
FCMPUOS	单精度浮点双操作数是否非数
FMOVIS	单精度浮点立即数生成
FMOVS	单精度浮点数在浮点寄存器间搬运
FABSS	单精度浮点数取绝对值
FNEGS	单精度浮点数取反
FADDS	单精度浮点数加法
FSUBS	单精度浮点数减法
FMULS	单精度浮点数乘法
FNMULS	单精度浮点数乘取反
FMACS	单精度浮点数乘累加
FMSCS	单精度浮点数乘累减
FNMACS	单精度浮点数乘取反累加
FNMSCS	单精度浮点数乘取反累减
FDIVS	单精度浮点数除法
FSQRTS	单精度浮点数取平方根
FRECIPS	单精度浮点数取倒数

### 12.5.1.2 传输类指令

表 12.9: 数据传输指令列表

FMTVRL	浮点寄存器字写传送指令
FMFVRL	浮点寄存器字读传送指令

### 12.5.1.3 内存存取指令

表 12.10: 内存存取指令列表

FLDS	单精度浮点加载指令
FLDRS	寄存器移位寻址单精度浮点加载指令
FLDMS	连续单精度浮点加载指令
FLRWS	单精度浮点立即数存储器读入指令
FSTS	单精度浮点存储指令
FSTRS	寄存器移位寻址单精度浮点存储指令
FSTMS	连续单精度浮点存储指令

## 12.5.2 浮点指令执行延时

E804 浮点单元对指令的执行延时进行了严格的设计，在满足低成本与低功耗设计需求同时，减少浮点指令执行延时。软件开发人员可以利用指令延时信息对软件进行调整优化，提高浮点程序执行效率。

表 12.11: 浮点指令执行延时列表

指令	执行周期	备注
<b>数据运算指令</b>		
FSTOSI	1	-
FSTOUI	1	-
FSITOS	1	-
FUITOS	1	-
FCMPZHSS	1	-
FCMPZLSS	1	-
FCMPZNES	1	-
FCMPZUOS	1	-
FCMPHSS	1	-
FCMPLTS	1	-
FCMPNES	1	-
FCMPUOS	1	-

下页继续

表 12.11 – 续上页

FMOVIS	1	-
FMOVS	1	-
FABSS	1	-
FNEGS	1	-
FADDS	2	-
FSUBS	2	-
FMULS	2	-
FNMULS	2	-
FMACS	4	拆分为 FMULS 和 FADDS 指令。
FMSCS	4	拆分为 FMULS 和 FSUBS 指令。
FNMACS	4	拆分为 FNMULS 和 FADDS 指令。
FNMSCS	4	拆分为 FNMULS 和 FSUBS 指令。
FDIVS	2-28	-
FSQRTS	2-28	-
FRECIPS	2-28	-
<b>传输指令</b>		
FMTVRL	1	-
FMFVRL	1	-
<b>内存存取指令</b>		
FLDS	1	-
FLDRS	3	拆分为 IXW 和 FLDS 指令。
FLDMS	N	拆分为 N 条 FLDS 指令。
FLRWS	1	-
FSTS	1	-
FSTRS	3	拆分为 IXW 和 FSTS 指令。
FSTMS	N	拆分为 N 条 FSTS 指令。

**注解:** 表示内存存取相关指令，指令完成周期取决于总线延时或者高速缓存命中情况，表中数值所列为最快情况。 \*

# 第十三章 初始话参考代码

## 13.1 MPU 设置示例

```
/* 设置区域 0 属性，不可执行，不可高缓，超级用户和普通用户可读写。  
  
mfcr r10,cr<19,0>  
bclri r10,0 //设置控制寄存器第 0 位为 0，区域 0 不可执行。  
bseti r10,8 //设置控制寄存器 8 和 9 位为 1，区域 0 超级用户和普通用户都可读写。  
bseti r10,9  
bclri r10,24 //设置控制寄存器第 24 位为 0，区域 0 不可高缓。  
mtcr r10,cr<19,0>  
  
/* 设置区域 1 属性，可执行，可高缓，超级用户和普通用户可读写  
  
mfcr r10,cr<19,0>  
bseti r10,1 //设置控制寄存器第 1 位为 1，区域 1 可执行  
bseti r10,10 //设置控制寄存器 10 和 11 位为 1，区域 1 超级用户和普通用户都可读//写。  
bseti r10,11  
bseti r10,25 //设置控制寄存器第 25 位为 1，区域 1 可高缓。  
mtcr r10,cr<19,0>  
  
/* 设置区域 2 属性，可执行，不可高缓，超级用户可读写，普通用户只读。  
  
mfcr r10,cr<19,0>  
bseti r10,2 //设置控制寄存器第 2 位为 1，区域 2 可执行。  
bclri r10,12 //设置 12 位为 0，13 位为 1，超级用户可读写，普通用户只读。  
bseti r10,13  
bclri r10,26 //设置控制寄存器第 26 位为 0，区域 2 不可高缓。  
mtcr r10,cr<19,0>  
  
/* 设置区域 3 属性，不可执行，可高缓，超级用户和普通用户都可读写。
```

(下页继续)

(续上页)

```
mfcr r10,cr<19,0>
bclri r10,3 //设置控制寄存器第 3 位为 0, 区域 3 不可执行。
bseti r10,14 //设置 14, 15 位为 1, 区域 3 超级用户和普通用户都可读写。
bclri r10,15
bseti r10,27 //设置控制寄存器第 27 位为 1, 区域 3 可高缓。
mtcr r10,cr<19,0>

/* 区域 4~7 属性设置和区域 0、1、2 的属性设置相同。

//区域 0~7 的可执行属性, 设置控制寄存器 CR<19,0> 的 [7:0] 位。
//区域 0~7 的访问权限, 设置控制寄存器 CR<19,0> 的 [23:12] 位。
//区域 0~7 的可缓存权限, 通过设置控制寄存器 CR<19,0> 的 [31:24] 位。


/* 设置保护区域 0, 基地址和保护区大小。

movi r10,0
mtcr r10,cr<21,0> //选择保护区域 0。
movi r10,0x0 //设置保护区基址 0x00000000。
ori r10,r10,0x3f //设置保护区大小 4G。
mtcr r10,cr<20,0> //设置保护区基址为 0, 保护区大小为 4G。

/* 设置保护区域 1, 基地址和保护区大小。

movi r10,1
mtcr r10,cr<21,0> //选择保护区域 1。
movih r10,0x2800 //设置保护区基址 0x28000000。
ori r10,r10,0x2f //设置保护区大小 16M。
mtcr r10,cr<20,0> //设置保护区地址区间为 0x28000000 ~ 0x29000000。

/* 设置保护区域 2, 基地址和保护区大小。

movi r10,2
mtcr r10,cr<21,0> //选择保护区域 2。
movih r10,0x2800 //设置保护区基址 0x28000000。
ori r10,r10,0x27 //设置保护区大小 1M。
mtcr r10,cr<20,0> //设置保护区地址区间为 0x28000000 ~0x28100000。
```

(下页继续)

(续上页)

```

/* 设置保护区域 3， 基地址和保护区大小。

movi r10,3
mtcr r10,cr<21,0> //选择保护区域 2。
movih r10,0x28f0 //设置保护区基地址 0x28f00000。
ori r10,r10,0x27 //设置保护区大小为 1M。
mtcr r10,cr<20,0> //设置保护区地址区间为 0x28f00000 ~0x29000000.

//保护区域 3~7 的基地址和保护区大小设置和区域 0、1、2 的设置相同。

//设置控制寄存器 CR<21,0> 选择保护区域。

//将保护区基地址和保护区大小，写入控制寄存器 CR<20,0>,。

/* 使能 MPU

mfcr r7, cr<18,0> //选择 MUP 使能控制寄存器。
bseti r7, 0 //设置控制寄存器 CR<18,0> 最低两位为 2' b01，使能 MPU。
bclri r7, 1
mtcr r7, cr<18,0> //将预置值写入 MPU 使能控制寄存器，开启 MPU。

```

## 13.2 高速缓存设置示例

开启高速缓存之前需要对整个高速缓存无效化，然后再根据实际应用需求配置 CER 并使能 CACHE。

```

// 高速缓存无效化。

lrw r1, 1 // 预设第 0 位 INV_ALL 为 1。
lrw r2, 0xe000f004 // 预设 CIR 所在地址。
st.w r1, (r2) // 将预设值写入 CIR，无效化操作开始。

// 设置可高速缓存区域 crcr0。

lrw r1, 0x00000039 // 预设起始地址为 0x0 的 512M 地址空间可高速缓存。
lrw r2, 0xe000f008 // 预设 CRCR0 所在地址。
st.w r1, (r2) // 将预设值写入 CIR，无效化操作开始。

// 可缓冲区域 crcr1~3 设置同 crcr0，将可高速缓冲的起始地址和 size 及使能位分别写到。

```

(下页继续)

(续上页)

```
//crcr1 地址 0xe000f00c,
//crcr2 地址 0xe000f010
//crcr3 地址 0xe000f014

// 开启高速缓存

    lrw r1, 0x0 // 预设值为 0x0000000。
    bseti r1, 0 //设置 Cache enable 打开。
    bclri r1, 1 //设置，指令和数据都可高速缓冲。
    bseti r1, 2 //设置 Cache 为 write back 模式。
    bseti r1, 4 //设置 Cache 可写。
    bclri r1, 5 //设置 Cache 关闭 write allocate。
    lrw r2, 0xe000f000 // 预设 CER 所在地址。
    st.w r1, (r2) //将预设值写入 CER, Cache 使能。

//此时 cache 可写并为写回模式，四路组相连，指令和数据 Cache，关闭 write allocate 功能。
```

### 13.3 中断使能初始化

在配置好中断控制器和中断向量表之后（具体参考紧耦合 IP），需要将中断使能位打开，具体设置如下：

```
psrset ee, ie
```

### 13.4 通用寄存器初始化示例

```
//初始化通用寄存器 r0~r15 和 r28 为 0。

    movi r0, 0 //初始化通用寄存器 0 为 0。
    movi r1, 0
    movi r2, 0
    movi r3, 0
    movi r4, 0
    movi r5, 0
    movi r6, 0
    movi r7, 0
```

(下页继续)

(续上页)

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

### 13.5 堆栈指针初始化示例

堆栈指针的设置和程序当前处在状态有关系如下所示，超级用户态下堆栈指针初始化为示例 1，用户态下堆栈指针初始化为示例 2。

#### 示例 1：

```
//超级用户态下，设置超级用户态和用户态堆栈指针  
  
//超级用户态下 r14 映射为超级用户态的堆栈指针寄存器  
  
//超级用户态下，用户态堆栈指针寄存器为 CR<14,1>  
  
lrw r14, 0x01000000 //设置超级用户态指针  
lrw r0, 0x02000000 //  
mtcr r0, cr<14,1> //设置用户态堆栈指针
```

#### 示例 2：

```
//用户态下，只能设置用户态指针：  
  
//用户态下 r14 映射为用户态的堆栈指针寄存器  
  
lrw r14, 0x02000000 //设置用户态指针
```

### 13.6 异常和中断服务程序入口地址设置示例

异常和中断服务程序的入口地址设置分两个步骤：

第一步：设置异常向量表地址寄存器 VBR，根据向量号，各个异常向量号对应异常向量表地址为，VBR + (向量号 << 2);

第一步：将异常服务程序的入口地址写到 Step1 中异常向量号对应的异常向量表地址中。

异常向量号为 2 的访问错误异常示例如下：

```
//设置异常向量表入口地址，VBR 对应控制寄存器 cr<1,0>

Lrw r2, 0
mtcr r2, cr<1,0> //设置异常向量表的地址为 0

//异常/中断服务程序入口地址设置

lrw r1, ACCERR_ERROR_BEGIN //设置异常服务程序入口地址
lrw r2, 0 //VBR 地址
movi r3, 0x2 //异常向量号
lsli r3, r3, 2
addu r2, r2, r3 //计算异常向量号对应异常向量表地址
st.w r1,(r2, 0x0) //将异常服务程序入口地址，存入相应的异常向量表地址中
br START

//用户设置的访问错误异常服务程序

label ACCERR_ERROR_BEGIN

/* 用户的异常服务程序 */

label ACCERR_ERROR_END
label START
```

## 13.7 FPU 初始化示例

浮点运算初始化时，需要对浮点运算的舍入模式，异常数据的处理等情况进行设置。

```
//清除浮点异常状态处理寄存器

lrw r1,0x0
mtcr r1, cr<2,2> //清除异常标志寄存器。

//设置浮点控制寄存器
```

(下页继续)

(续上页)

```
lrw r1,0x0
bclri r1, 27 //设置非规格数处理方式，当成 0 处理。
bclri r1, 25 //设置 25 和 24 位为 0，舍入模式 round to nearest even。
bclri r1, 24 //
bclri r1, 5 //设置非规格数，不触发异常。
bclri r1, 4 //设置结果不精确时，不进入异常，根据舍入模式处理。
bclri r1, 3 //设置结果下溢出，时不触发异常，根据舍入模式处理。
bclri r1, 2 //设置结果上溢出，时不触发异常，根据舍入模式处理。
bclri r1, 1 //设置除 0 操作时触发异常，不写回结果，跳转到除 0 异常。
bclri r1, 0 //设置若输入和输出出现 NaN，不触发异常，输出 QNaN。
mtcr r1, cr<1,2> //将预制值写到浮点控制寄存器 FCR。

//此时 FPU 的控制状态为，只有除 0 异常出现时调转到异常处理，其他异常出现时不触发异常，
//非规格数当成 0 处理，舍入模式为 round to nearest even。
```

## 第十四章 附录 A 基本指令术语表

以下是每条 E804 实现的玄铁 V2 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。

每条玄铁 V2 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

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/RY、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		0	1	

图 14.1: ADDC-1

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 ← 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

图 14.2: ADDC-2

### 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	$RZ \leftarrow RZ + \text{zero\_extend(OIMM12)}$	$RZ \leftarrow RX + \text{zero\_extend(OIMM12)}$	$RZ \leftarrow R28 + \text{zero\_extend(OIMM18)}$
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 $(\text{oimm12} < 257)$ , 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 $(\text{oimm12} < 257)$ , 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 $(\text{oimm12} < 9)$ and $(z < 8)$ and $(x < 8)$ , addi16 rz, rx, oimm3; elsif $(\text{oimm12} < 257)$ 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。		

<b>16 位指令 1</b>	
<b>操作</b>	$RZ \leftarrow RZ + \text{zero\_extend(OIMM8)}$
<b>语法</b>	addi16 rz, oimm8
<b>说明</b>	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。
<b>影响标志位</b>	无影响
<b>限制</b>	寄存器的范围为 r0-r7；立即数的范围为 1-256。
<b>异常</b>	无

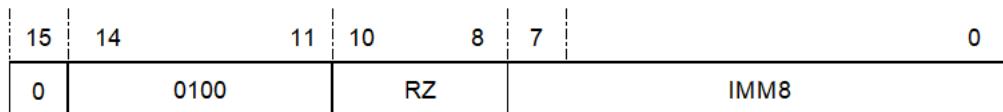
**16 位指令格式 1：**

图 14.3: ADDI-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

<b>16 位指令 2</b>	
<b>操作</b>	$RZ \leftarrow RX + \text{zero\_extend(OIMM3)}$
<b>语法</b>	addi16 rz, rx, oimm3
<b>说明</b>	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。
<b>影响标志位</b>	无影响
<b>限制</b>	寄存器的范围为 r0-r7；立即数的范围为 1-8。
<b>异常</b>	无

**16 位指令格式 2:**

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

图 14.4: ADDI-2

**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

<b>32 位指令 1</b>	
<b>操作</b>	$RZ \leftarrow RX + \text{zero\_extend}(OIMM12)$
<b>语法</b>	addi32 rz, rx, oimm12
<b>说明</b>	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。
<b>影响标</b>	无影响
<b>志位</b>	
<b>限制</b>	立即数的范围为 0x1-0x1000。
<b>异常</b>	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		

图 14.5: ADDI-3

**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。

**0000000000000:**

加 0x1

**0000000000001:**

加 0x2

.....

**1111111111111:**

加 0x1000

32 位指令 2	
<b>操作</b>	RZ $\leftarrow$ R28 + zero_extend(OIMM18)
<b>语法</b>	addi32 rz, r28, oimm18
<b>说明</b>	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。
<b>影响标</b>	无影响
<b>志位</b>	
<b>限制</b>	立即数的范围为 0x1-0x40000。
<b>异常</b>	无

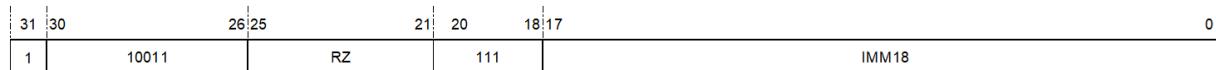
**32 位指令格式 2:**

图 14.6: ADDI-4

**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。

**000000000000000:**

加 0x1

**000000000000001:**

加 0x2

.....

**111111111111111:**

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：

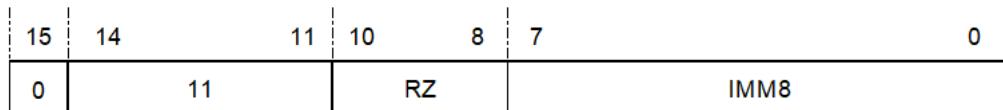


图 14.7: ADDI(SP)-1

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	$SP \leftarrow SP + \text{zero\_extend}(IMM)$
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

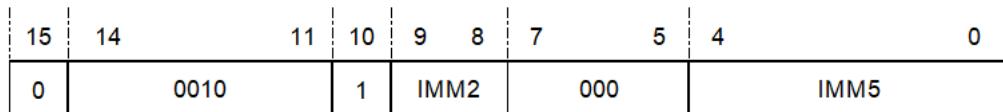
**16 位指令格式 2:**

图 14.8: ADDI(SP)-2

**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, ry; elseif ( $y == z$ ) and ( $x < 16$ ) and ( $z < 16$ ), then addu16 rz, rx; else addu32 rz, rx, ry;
说明	将 RZ/RY 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:

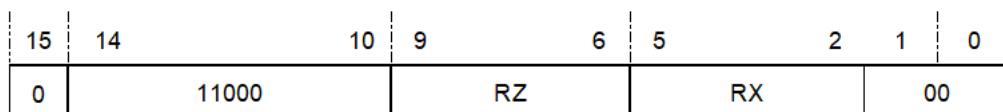


图 14.9: ADDU-1

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

图 14.10: ADDU-2

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

图 14.11: ADDU-3

## 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/RY 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：

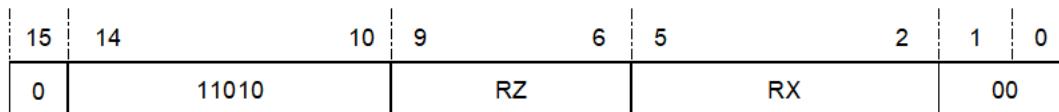


图 14.12: AND-1

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

图 14.13: AND-2

## 14.7 ANDI——立即数按位与指令

统一化指令	
<b>语法</b>	andi rz, rx, imm16
<b>操作</b>	RZ $\leftarrow$ RX and zero_extend(IMM12)
<b>编译结果</b>	仅存在 32 位指令 andi32 rz, rx, imm12
<b>说明</b>	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。
<b>影响标志位</b>	无影响
<b>限制</b>	立即数的范围为 0x0-0xFFF。
<b>异常</b>	无

32 位指令	
<b>操作</b>	RZ $\leftarrow$ RX and zero_extend(IMM12)
<b>语法</b>	andi32 rz, rx, imm12
<b>说明</b>	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。
<b>影响标志位</b>	无影响
<b>限制</b>	立即数的范围为 0x0-0xFFF。
<b>异常</b>	无

**32 位指令格式：**

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

图 14.14: ANDI

## 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	$RZ \leftarrow RZ \text{ and } (\neg RX)$	$RZ \leftarrow RX \text{ and } (\neg RY)$
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	$RZ \leftarrow RZ \text{ and } (\neg RX)$
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	

图 14.15: ANDN-1

32 位指令	
操作	$RZ \leftarrow RX \text{ and } (\neg RY)$
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

图 14.16: ANDN-2

## 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	$RZ \leftarrow RX \text{ and } \neg(\text{zero\_extend(imm16)})$
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 0x0-0xFFFF。
异常	无

32 位指令	
操作	$RZ \leftarrow RX \text{ and } \neg(\text{zero\_extend(imm16)})$
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 0x0-0xFFFF。
异常	无

**32 位指令格式:**

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

图 14.17: ANDNI

**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	$RZ \leftarrow RZ >>> RX[5:0]$	$RZ \leftarrow RX >>> RY[5:0]$
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, rx, ry;	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	

<b>16 位指令</b>	
<b>操作</b>	$RZ \leftarrow RZ >>> RX[5:0]$
<b>语法</b>	asr16 rz, rx
<b>说明</b>	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。
<b>影响标志位</b>	无影响
<b>限制</b>	寄存器的范围为 r0-r15。
<b>异常</b>	无

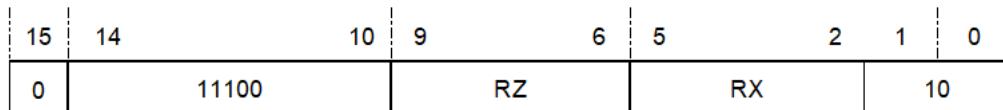
**16 位指令格式：**

图 14.18: ASR-1

<b>32 位指令</b>	
<b>操作</b>	$RZ \leftarrow RX >>> RY[5:0]$
<b>语法</b>	asr32 rz, rx, ry
<b>说明</b>	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。
<b>影响标志位</b>	无影响
<b>异常</b>	无

**32 位指令格式：**

图 14.19: ASR-2

## 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 $>>>$ 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	$RZ \leftarrow RX >>> OIMM5, C \leftarrow RX[OIMM5 - 1]$
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：

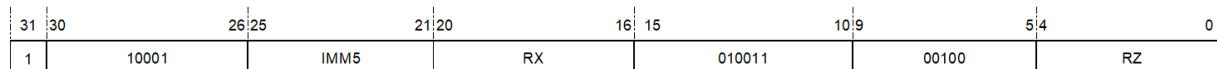


图 14.20: ASRC

#### 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	$RZ \leftarrow RX >>> IMM5$
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	$RZ \leftarrow RX >>> IMM5$
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:

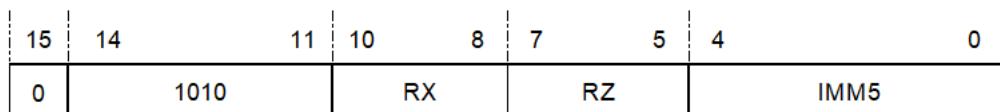


图 14.21: ASRI-1

32 位指令	
操作	RZ $\leftarrow$ 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：

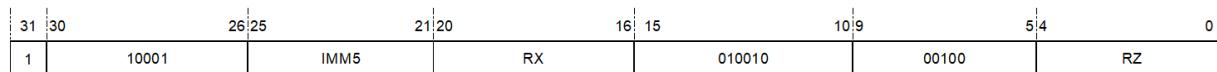


图 14.22: ASRI-2

### 14.13 BCLRI——立即数位清零指令

统一化指令	
语法	bclri rz, imm5
操作	RZ $\leftarrow$ RZ[IMM5] 清零
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;
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。

<b>16 位指令</b>	
<b>操作</b>	RZ $\leftarrow$ RZ[IMM5] 清零
<b>语法</b>	bclri16 rz, imm5
<b>说明</b>	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。
<b>影响标志位</b>	无影响
<b>限制</b>	寄存器的范围为 r0-r7; 立即数的范围为 0-31。
<b>异常</b>	无

**16 位指令格式:**

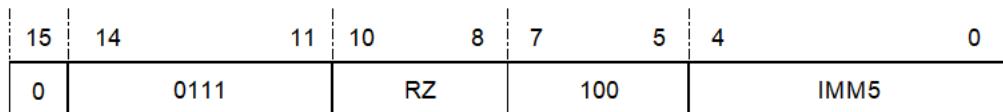


图 14.23: BCLRI-1

<b>32 位指令</b>	
<b>操作</b>	RZ $\leftarrow$ RX[IMM5] 清零
<b>语法</b>	bclri32 rz, rx, imm5
<b>说明</b>	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。
<b>影响标志位</b>	无影响
<b>限制</b>	立即数的范围为 0-31。
<b>异常</b>	无

**32 位指令格式:**

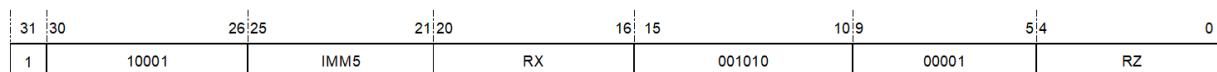


图 14.24: BCLRI-2

## 14.14 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 ±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 $PC \leftarrow PC + 4$
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 $PC \leftarrow PC + 4$ 。 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 ±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	

图 14.25: BEZ

## 14.15 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 $\leftarrow$ PC + sign_extend(offset << 1) else PC $\leftarrow$ 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 $\leftarrow$ PC + 2。 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。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：

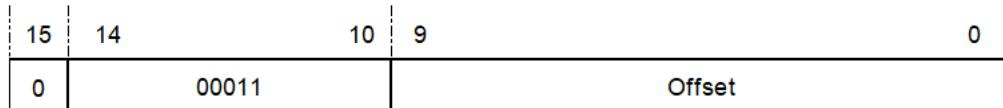


图 14.26: BF-1

32 位指令	
操作	C 等于零则程序转移 if(C == 0) PC $\leftarrow$ PC + sign_extend(offset << 1) else PC $\leftarrow$ 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 $\leftarrow$ 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。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：

31	30	26 25	21 20	16 15	0
1	11010	00010	00000	Offset	

图 14.27: BF-2

## 14.16 BGENI——立即数位产生指令

统一化指令	
语法	bgeni rz, imm5
操作	$RZ \leftarrow (2)^{IMM5}$
编译	仅存在 32 位指令。
结果	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	$(2)^{IMM5}$	

图 14.28: BGENI-1

如果 IMM5 大于等于 16:

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

图 14.29: BGENI-2

### 14.17 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 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 位指令格式:

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

图 14.30: BGENR

### 14.18 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( $RX \geq 0$ ) $PC \leftarrow PC + \text{sign\_extend}(\text{offset} \ll 1);$ else $PC \leftarrow PC + 4;$
编译结果	仅存在 32 位指令 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 $\pm 64KB$ 地址空间。
影响标志位	无影响
异常	无

32 位指令	
操作	寄存器大于等于零则程序转移 if( $RX \geq 0$ ) $PC \leftarrow PC + \text{sign\_extend}(\text{offset} \ll 1)$ else $PC \leftarrow PC + 4$
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 $PC \leftarrow PC + 4$ 。 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 $\pm 64KB$ 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	

图 14.31: BHSZ

**14.19 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( $RX > 0$ ) $PC \leftarrow PC + \text{sign\_extend}(\text{offset} \ll 1)$ else $PC \leftarrow PC + 4$
编译结果	仅存在 32 位指令 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 $PC \leftarrow PC + 4$ 。 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。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( $RX > 0$ ) $PC \leftarrow PC + sign\_extend(offset \ll 1)$ else $PC \leftarrow PC + 4$
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 $PC \leftarrow PC + 4$ 。 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。BHZ 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 位指令格式：

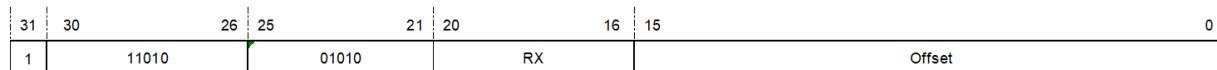


图 14.32: BHZ

## 14.20 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：

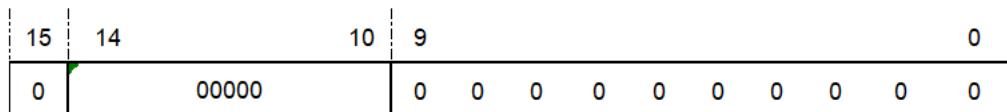


图 14.33: BKPT

## 14.21 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( $RX \leq 0$ ) $PC \leftarrow PC + \text{sign\_extend}(\text{offset} \ll 1)$ else $PC \leftarrow PC + 4$
编译结果	仅存在 32 位指令 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 $\pm 64KB$ 地址空间。
影响标志位	无影响
异常	无

32 位指令	
操作	寄存器小于等于零则程序转移 if( $RX \leq 0$ ) $PC \leftarrow PC + sign\_extend(offset \ll 1)$ else $PC \leftarrow PC + 4$
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 $PC = PC + 4$ 。 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 $\pm 64KB$ 地址空间。
影响标志位	无影响
异常	无

### 32 位指令格式：

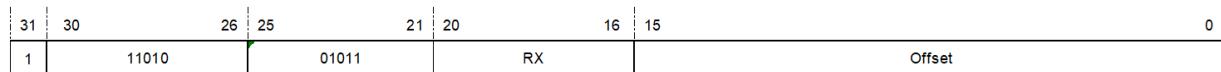


图 14.34: BLSZ

## 14.22 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( $RX < 0$ ) $PC \leftarrow PC + \text{sign\_extend}(\text{offset} \ll 1)$ else $PC \leftarrow PC + 4$
编译结果	仅存在 32 位指令 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 $PC \leftarrow PC + 4$ 。 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。BLZ 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	寄存器小于零则程序转移 if( $RX < 0$ ) $PC \leftarrow PC + \text{sign\_extend}(\text{offset} \ll 1)$ else $PC \leftarrow PC + 4$
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 $PC \leftarrow PC + 4$ 。 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。BLZ 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 位指令格式：

31	30	26	25	21	20	16	15	0
1	11010		01100		RX		Offset	

图 14.35: BLZ

## 14.23 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：

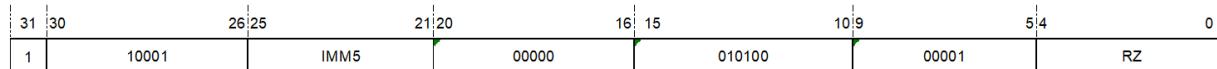


图 14.36: BMASKI

#### 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

## 14.24 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( $RX \neq 0$ ) $PC \leftarrow PC + \text{sign\_extend}(\text{offset} \ll 1)$ else $PC \leftarrow PC + 4$
编译结果	仅存在 32 位指令。 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 $PC \leftarrow PC + 4$ 。 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。BNEZ 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 if( $RX \neq 0$ ) $PC \leftarrow PC + \text{sign\_extend}(\text{offset} \ll 1)$ else $PC \leftarrow PC + 4$
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 $PC \leftarrow PC + 4$ 。 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。BNEZ 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 位指令格式：

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

图 14.37: BNEZ

## 14.25 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。 $\begin{aligned} RX &\leftarrow RX - 1 \\ \text{if}(RX > 0) \\ PC &\leftarrow PC + \text{sign\_extend}(\text{offset} \ll 1) \\ \text{else} \\ PC &\leftarrow PC + 4 \end{aligned}$
编译结果	仅存在 32 位指令。 bnez32 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

<b>32 位 指 令</b>	
<b>操 作:</b>	寄存器不等于零则程序转移 $\begin{aligned} RX &\leftarrow RX - 1 \\ & \text{if}(RX > 0) \\ & \quad PC \leftarrow PC + \text{sign\_extend}(\text{offset} \ll 1) \\ & \text{else} \\ & \quad PC \leftarrow PC + 4 \end{aligned}$
<b>语 法:</b>	bnezad32 rx, label
<b>说 明:</b>	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 $\pm 64KB$ 地址空间。
<b>影 响 标 志 位:</b>	无影响
<b>异 常:</b>	无

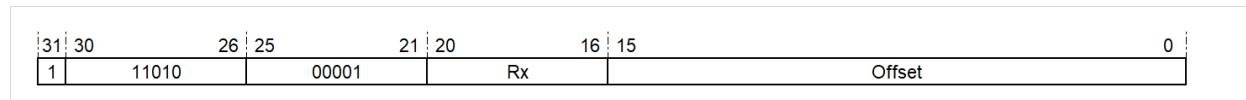
**指令格式:**

图 14.38: BNEZAD

## 14.26 BR——无条件跳转指令

统一化指令	
语法	br label
操作	$PC \leftarrow PC + sign\_extend(offset << 1)$
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	$PC \leftarrow PC + sign\_extend(offset << 1)$
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 $\pm 1KB$ 地址空间。
影响标志位	无影响
异常	无

16 位指令格式：

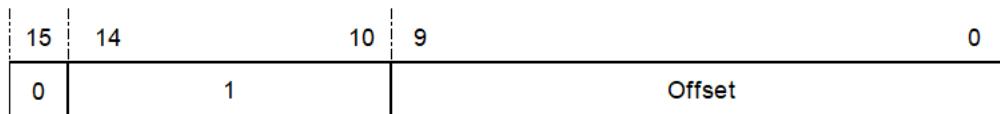


图 14.39: BR-1

32 位指令	
操作	$PC \leftarrow PC + sign\_extend(offset << 1)$
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 $\pm 64KB$ 地址空间。
影响标志位	无影响
异常	无

32 位指令格式：

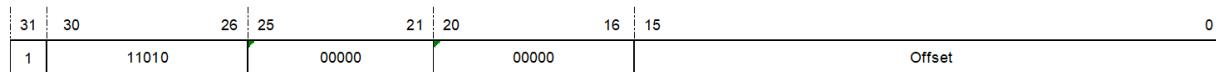


图 14.40: BR-2

## 14.27 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

<b>32 位指令</b>	
<b>操作</b>	for i=0 to 31 RZ[i] ← RX[31-i];
<b>语法</b>	brev32 rz, rx
<b>说明</b>	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”。
<b>影响标志位</b>	无影响
<b>异常</b>	无

**指令格式：**

图 14.41: BREV

## 14.28 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 $\leftarrow$ RZ[IMM5] 置位	RZ $\leftarrow$ 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 $\leftarrow$ 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:

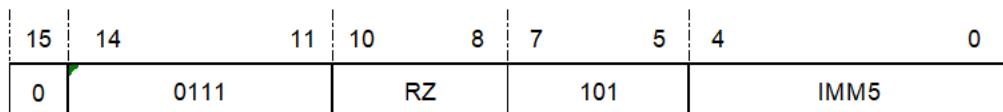


图 14.42: BSETI-1

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:

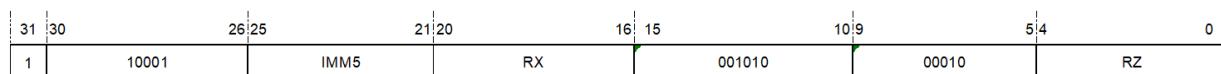


图 14.43: BSETI-2

## 14.29 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	根据跳转的范围编译为对应的 16 位或 32 位指令 if( $0 < \text{offset} < 1\text{KB}$ ), then bsr16 label; else bsr32 label;

<b>说明:</b>	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。
<b>影响标志位:</b>	无影响
<b>异常:</b>	无

<b>16 位指令</b>	
<b>操作:</b>	链接并跳转到子程序： $R15 \leftarrow PC + 2$ $PC \leftarrow PC + sign\_extend(offset \ll 1)$
<b>语法:</b>	bsr16 label
<b>说明:</b>	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 + 2）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 10 位相对偏移量有符号扩展到 32 位后的值得到。BSR16 指令的跳转范围是 ±1KB 地址空间。
<b>影响标志位:</b>	无影响
<b>限制:</b>	BSR16 指令的跳转目标不能是 BSR16 指令本身。
<b>异常:</b>	无

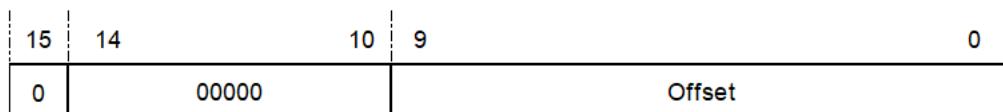
**指令格式：**

图 14.44: BSR-1

Offset 域——指定跳转的相对偏移量。

注意：跳转的相对偏移量（Offset）不能为 0x0。

<b>32 位 指 令</b>	
<b>操 作:</b>	链接并跳转到子程序: $R15 \leftarrow PC + 4$ $PC \leftarrow PC + sign\_extend(offset \ll 1)$
<b>语 法:</b>	bsr32 label
<b>说 明:</b>	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 ±64MB 地址空间。
<b>影 响 标 志 位:</b>	无影响
<b>异 常:</b>	无

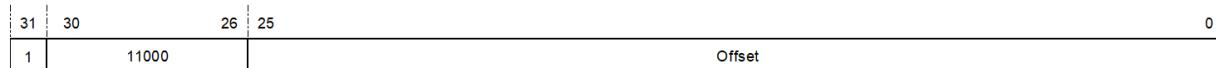
**指令格式:**

图 14.45: BSR-2

### 14.30 BT——C 为 1 分支指令

统一化指令	
语法	bt label
操作	<pre>if(C == 1)     PC ← PC + sign_extend(offset &lt;&lt; 1); else     PC ← next PC;</pre>
编译结果	<p>根据跳转的范围编译为对应的 16 位或 32 位指令。</p> <pre>if (offset&lt;1KB), then     bt16 label; else     bt32 label;</pre>
说明	<p>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。</p> <p>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。</p>
影响标志位	无影响
异常	无

16 位指令	
操作	<p>C 等于一则程序转移</p> <pre>if(C == 1)     PC ← PC + sign_extend(offset &lt;&lt; 1) else     PC ← PC + 2</pre>
语法	bt16 label
说明	<p>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 <math>PC \leftarrow PC + 2</math>。</p> <p>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。BT16 指令的转移范围是 ±1KB 地址空间。</p>
影响标志位	无影响
异常	无

16 位指令格式：

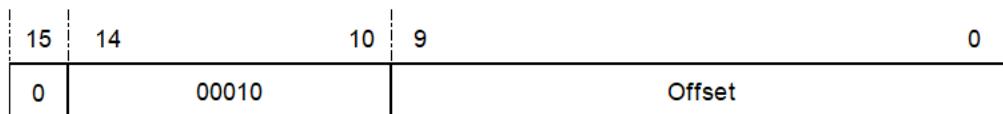


图 14.46: BT-1

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

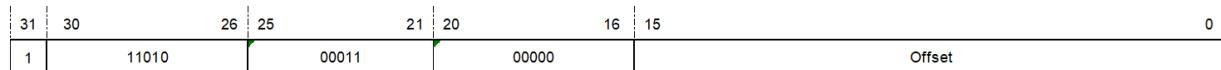
**32 位指令格式：**

图 14.47: BT-2

**14.31 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	$C \leftarrow RX[IMM5]$
语法	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	$C \leftarrow RX[IMM5]$
限制	立即数的范围为 0-31。
异常	无

32 位指令格式：

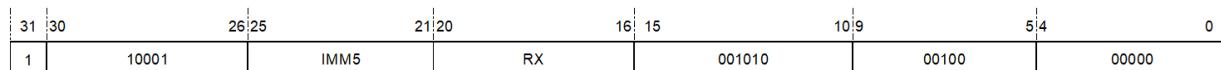


图 14.48: BTSTI

### 14.32 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

图 14.49: CLRF

**14.33 CLRT——C 为 1 清零指令**

统一化指令	
语法	clrt rz
操作	if C==1, then RZ ← 0; else RZ ← 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 ← 0; else RZ ← 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	00001	00000

图 14.50: CLRT

## 14.34 CMPHS——无符号大于等于比较指令

统一化指令	
语法	cmphs rx, ry
操作	RX 与 RY 作无符号比较。 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 cmphs16 rx, ry; else 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。cmphs 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 >= RY, then C ← 1; else C ← 0;
语法	cmphs16 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。cmphs16 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	

图 14.51: CMPHS-1

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	25	20	15	9	4	0
1	10001	RY	RX	000001	00001	00000	

图 14.52: CMPHS-2

## 14.35 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 $\geq$ 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 (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 $\geq$ zero_extend(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

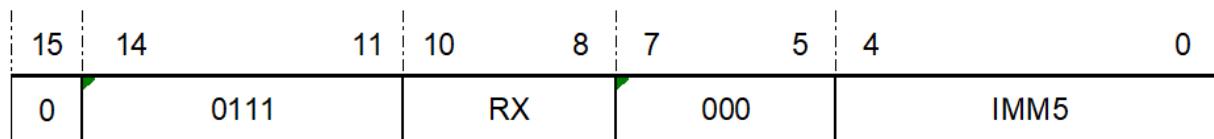
**16 位指令格式:**

图 14.53: CMPSI-1

**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 $\geq$ zero_extend(OIMM16), then C $\leftarrow$ 1; else C $\leftarrow$ 0;
语法	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。cmphsi32 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:

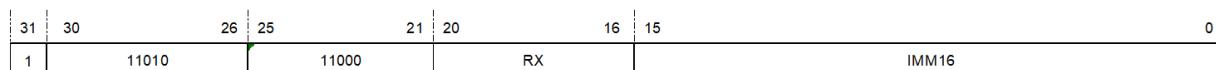


图 14.54: CMPHSI-2

### 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

### 14.36 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 ← 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 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 ← 1; else C ← 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

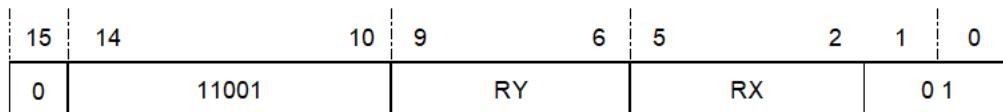
**16 位指令格式:**

图 14.55: CMPLT-1

<b>32 位指令</b>	
<b>操作</b>	RX 与 RY 作有符号比较。 If RX < RY, then C ← 1; else C ← 0;
<b>语法</b>	cmplt32 rx, ry
<b>说明</b>	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。
<b>影响标志位</b>	根据比较结果设置条件位 C
<b>异常</b>	无

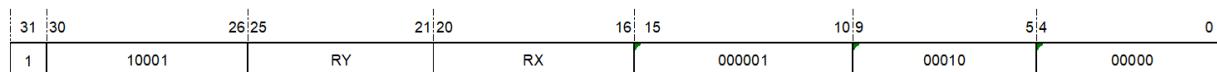
**32 位指令格式:**

图 14.56: CMPLT-2

### 14.37 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 < 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 (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 < zero_extend(OIMM5), then C ← 1; else C ← 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。cmplti16 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:

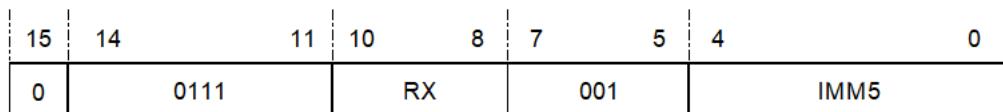


图 14.57: CMPLT-1

### 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

<b>32 位指令</b>	
<b>操作</b>	RX 与立即数作有符号比较。 If RX < zero_extend(OIMM16), then C ← 1; else C ← 0;
<b>语法</b>	cmplti32 rx, oimm16
<b>说明</b>	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。
<b>影响标志位</b>	根据比较结果设置条件位 C
<b>限制</b>	立即数的范围为 0x1-0x10000。
<b>异常</b>	无

**32 位指令格式:**

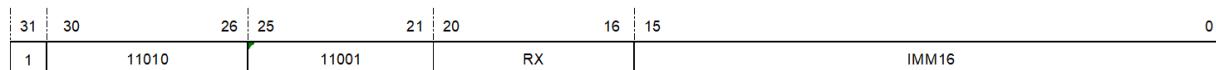


图 14.58: CMPLT-2

**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

### 14.38 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

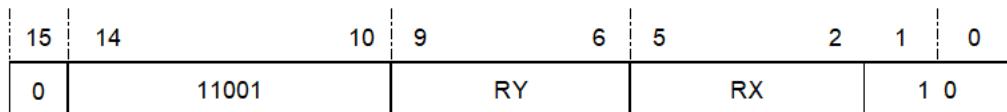
**16 位指令格式:**

图 14.59: CMPNE-1

<b>32 位指令</b>	
<b>操作</b>	RX 与 RY 作比较。 If RX != RY, then C ← 1; else C ← 0;
<b>语法</b>	cmpne32 rx, ry
<b>说明</b>	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。
<b>影响标志位</b>	根据比较结果设置条件位 C
<b>异常</b>	无

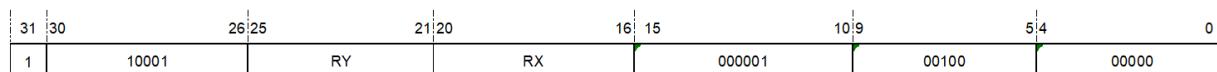
**32 位指令格式:**

图 14.60: CMPNE-2

## 14.39 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 != zero_extend(imm16), 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 (x<7) and (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 != zero_extend(IMM5), then C ← 1; else C ← 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：

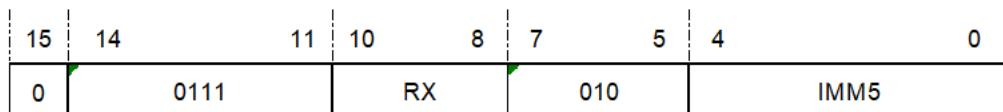


图 14.61: CMPNEI-1

32 位指令	
操作	RX 与立即数作比较。 If RX != zero_extend(imm16), then C ← 1; else C ← 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

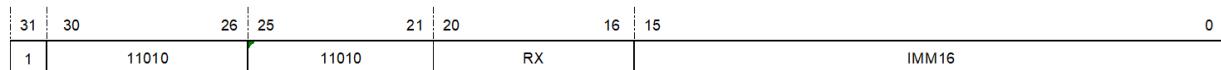
**32 位指令格式：**

图 14.62: CMPNEI-2

## 14.40 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 $\leftarrow$ RX - zero_extend(IMM5); else RZ $\leftarrow$ 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 $\leftarrow$ RX - zero_extend(IMM5); else RZ $\leftarrow$ 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:

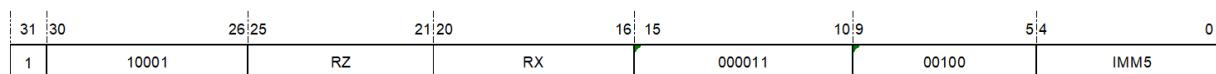


图 14.63: DECF

## 14.41 DECGT——减法大于零置 C 位指令

统一化指令	
语法	decgt rz, rx, imm5
操作	RZ $\leftarrow$ RX - 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 decgt32 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 - zero_extend(IMM5); If RZ > 0, then C $\leftarrow$ 1; else C $\leftarrow$ 0;
语法	decgt32 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：

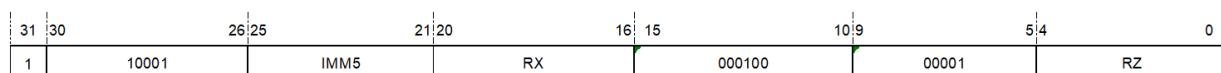


图 14.64: DECGT

## 14.42 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 - 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 - 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：

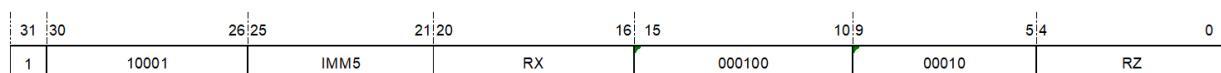


图 14.65: DECLT

### 14.43 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:

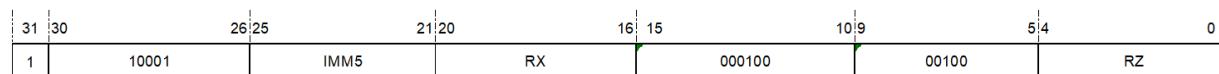


图 14.66: DECNE

## 14.44 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 $\leftarrow$ RX - zero_extend(IMM5); else RZ $\leftarrow$ 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 $\leftarrow$ RX - zero_extend(IMM5); else RZ $\leftarrow$ 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:

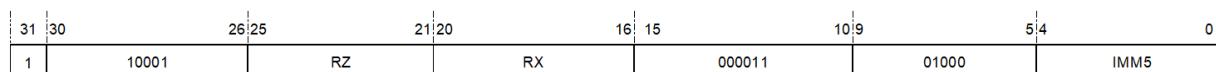


图 14.67: DECT

## 14.45 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：

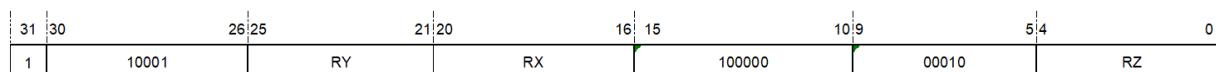


图 14.68: DIVS

## 14.46 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：

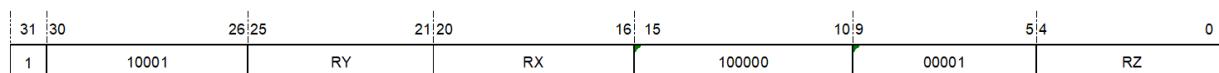


图 14.69: DIVU

### 14.47 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：

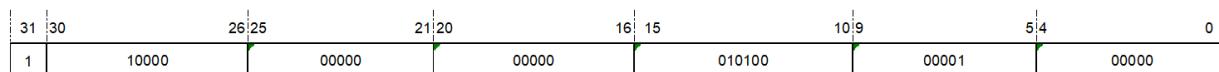


图 14.70: DOZE

## 14.48 FF0——快速找 0 指令

统一化指令	
语法	ff0 rz, rx
操作	RZ $\leftarrow$ find_first_0(RX);
编译	仅存在 32 位指令
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 $\leftarrow$ 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

指令格式：

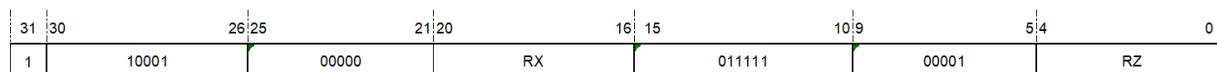


图 14.71: FF0

## 14.49 FF1——快速找 1 指令

统一化指令	
语法	ff1 rz, rx
操作	RZ $\leftarrow$ find_first_1(RX);
编译	仅存在 32 位指令
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 $\leftarrow$ 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

指令格式：

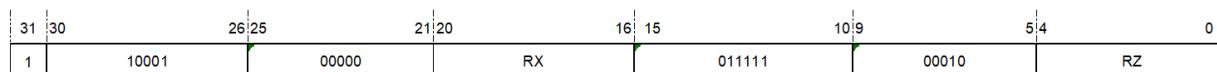


图 14.72: FF1

## 14.50 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。符号的值的有效范围是 ±256KB 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。符号的值的有效范围是 ±256KB 地址空间。
影响标志位	无影响
异常	无

指令格式：

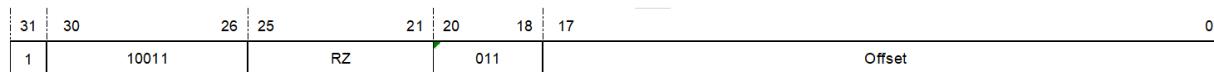


图 14.73: GRS

## 14.51 IDLY——中断识别禁止指令

统一化指令	
语 法	idle
操 作	禁止中断识别 4 个指令
编 译 结 果	仅存在 32 位指令 idle32
说 明	idle 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 idle 指令执行后被清零。如果在 idle 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	idle 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 idle 后面的指令序列中有另一个 idle 指令，那么它也将被忽略。但是如果 rte, rfi, doze, wait, stop 等指令，那么他们就会引起 idle 指令序列的中止。 idle 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 idle 计数器停在不为零的状态时，中断就会被屏蔽。如果在 idle 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。 idle 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	
操作	禁止中断识别 4 个指令 disable_int_in_following(4);
语法	idle32
说明	idle 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 idle 指令执行后被清零。如果在 idle 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	idle 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 idle 后面的指令序列中有另一个 idle 指令，那么它也将被忽略。但是如果 rte, rfi, doze, wait, stop 等指令，那么他们就会引起 idle 指令序列的中止。 idle 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 idle 计数器停在不为零的状态时，中断就会被屏蔽。如果在 idle 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。 idle 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：

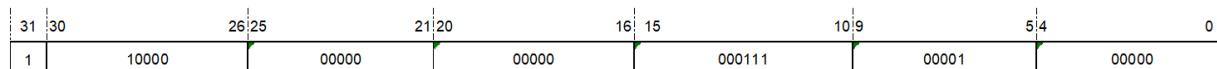


图 14.74: IDLY

## 14.52 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:

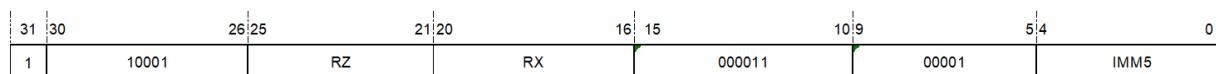


图 14.75: INC

### 14.53 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 $\leftarrow$ RX + zero_extend(IMM5); else RZ $\leftarrow$ 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 $\leftarrow$ RX + zero_extend(IMM5); else RZ $\leftarrow$ 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:

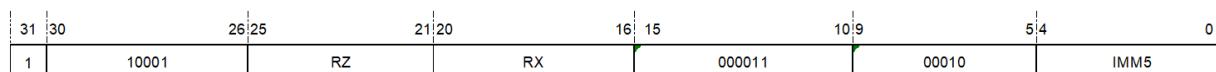


图 14.76: INCT

## 14.54 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：

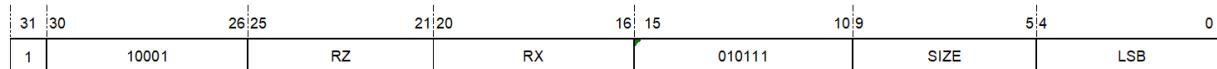


图 14.77: INS

#### 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

## 14.55 IPOP——中断出栈指令

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

<b>说明:</b>	从堆栈指针寄存器中载入中断的通用寄存器现场 {R0~R3, R12, R13 }，然后更新堆栈指针寄存器到堆栈存储器的顶端。采用堆栈指针寄存器直接寻址方式。
<b>影响标志位:</b>	无影响
<b>异常:</b>	访问错误异常、未对齐异常

<b>16 位指令</b>	
<b>操作:</b>	从堆栈指针寄存器中载入中断的通用寄存器现场 {R0~R3, R12, R13 }，然后更新堆栈指针寄存器到堆栈存储器的顶端； $\{R0\sim R3, R12, R13\} \leftarrow MEM[SP]\sim MEM[SP+20];$ $SP \leftarrow SP + 24;$
<b>语法:</b>	ipop16
<b>说明:</b>	从堆栈指针寄存器中载入中断的通用寄存器现场 {R0~R3, R12, R13 }，然后更新堆栈指针寄存器到堆栈存储器的顶端。采用堆栈指针寄存器直接寻址方式。
<b>影响标志位:</b>	无影响
<b>异常:</b>	访问错误异常、未对齐异常

**指令格式：**

15	14	10	9	8	7	5	4	0
1		00101	0	0	011		00011	

图 14.78: IPOP

## 14.56 IPUSH——中断压栈指令

<b>统一化指令</b>	
<b>语法</b>	ipush
<b>操作</b>	将中断的通用寄存器现场 {R0~R3, R12, R13} 存储到堆栈存储器中，然后更新堆栈指针寄存器到堆栈存储器的顶端； MEM[SP-4]~MEM[SP-24] ←{R13,R12,R3~R0}; SP←SP-24;
<b>编译结果</b>	仅存在 16 位指令 ipush;

<b>说明：</b>	将中断的通用寄存器现场 { R0~R3, R12, R13 } 保存到堆栈存储器中，然后更新堆栈指针寄存器到堆栈存储器的顶端。采用堆栈指针寄存器直接寻址方式。
<b>影响标志位：</b>	无影响
<b>异常：</b>	访问错误异常、未对齐异常

<b>16 位指令</b>	
<b>操作：</b>	将中断的通用寄存器现场 {R0~R3, R12, R13} 存储到堆栈存储器中，然后更新堆栈指针寄存器到堆栈存储器的顶端； MEM[SP-4]~MEM[SP-24] ←{R13,R12,R3~R0}; SP←SP-24;
<b>语法：</b>	ipush16
<b>说明：</b>	将中断的通用寄存器现场 { R0~R3, R12, R13 } 保存到堆栈存储器中，然后更新堆栈指针寄存器到堆栈存储器的顶端。采用堆栈指针寄存器直接寻址方式。
<b>影响标志位：</b>	无影响
<b>异常：</b>	访问错误异常、未对齐异常

指令格式：

15	14	10	9	8	7	5	4	0
0	00101	0	0	011		00010		

图 14.79: IPUSH

### 14.57 IXH——索引半字指令

统一化指令	
语法	ixh rz, rx, ry
操作	$RZ \leftarrow RX + (RY << 1)$
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	$RZ \leftarrow RX + (RY << 1)$
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		

图 14.80: IXH

### 14.58 IXW——索引字指令

统一化指令	
语法	ixw rz, rx, ry
操作	$RZ \leftarrow RX + (RY << 2)$
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	$RZ \leftarrow RX + (RY << 2)$
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：



图 14.81: IXW

### 14.59 IXD——索引双字指令

统一化指令	
语法	ixd rz, rx, ry
操作	$RZ \leftarrow RX + (RY << 3)$
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 << 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：

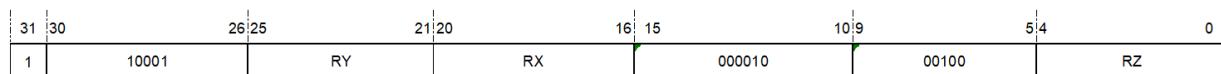


图 14.82: IxD

## 14.60 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 & 0xffffffff
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

<b>16 位指令</b>	
<b>操作</b>	跳转到寄存器指定的位置 $PC \leftarrow RX \& 0xffffffff$
<b>语法</b>	jmp16 rx
<b>说明</b>	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。
<b>影响标志位</b>	无影响
<b>异常</b>	无

**16 位指令格式:**

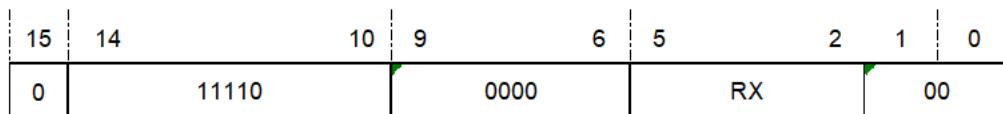


图 14.83: JMP-1

<b>32 位指令</b>	
<b>操作</b>	跳转到寄存器指定的位置 $PC \leftarrow RX \& 0xffffffff$
<b>语法</b>	jmp32 rx
<b>说明</b>	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。
<b>影响标志位</b>	无影响
<b>异常</b>	无

**32 位指令格式:**

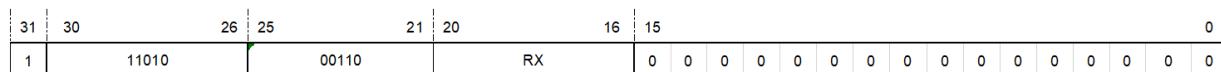


图 14.84: JMP-2

## 14.61 JMPI——间接跳转指令

统一化指令	
语法	jmpi label
操作	程序跳转到存储器指定的位置 $PC \leftarrow MEM[(PC + \text{zero\_extend}(\text{offset} \ll 2)) \& 0xfffffff]$
编译	仅存在 32 位指令。
结果	jmpi32 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 $PC \leftarrow MEM[(PC + \text{zero\_extend}(\text{offset} \ll 2)) \& 0xfffffff]$
语法	jmpi32 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:

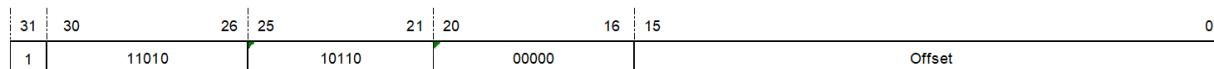


图 14.85: JMPI

## 14.62 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 $R15 \leftarrow PC + 4$ , $PC \leftarrow RX \& 0xffffffff$
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 jsr16 rx; else 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

16 位指令	
操作	链接并跳转到寄存器指定的子程序位置 $R15 \leftarrow PC + 4$ , $PC \leftarrow RX \& 0xffffffff$
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:

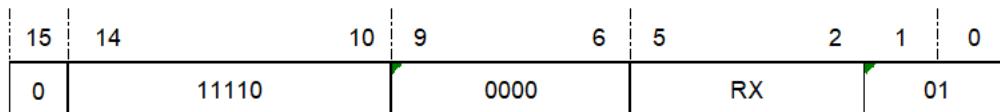


图 14.86: JSR-1

32 位指令	
操作	链接并跳转到寄存器指定的子程序位置 $R15 \leftarrow PC + 4, PC \leftarrow RX \& 0xffffffe$
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

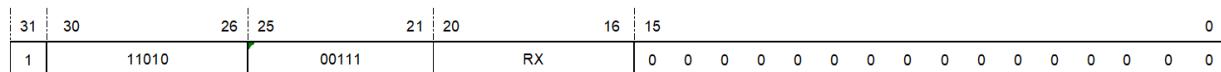
**32 位指令格式：**

图 14.87: JSR-2

## 14.63 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 $\leftarrow$ next PC, PC $\leftarrow$ MEM[ (PC + zero_extend(offset << 2)) & 0xffffffffc]
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
操作数	$R15 \leftarrow PC + 4, PC \leftarrow MEM[(PC + zero\_extend(offset \ll 2)) \& 0xffffffff]$
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：

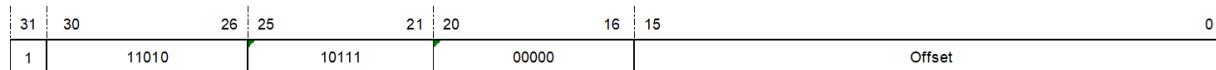


图 14.88: JSRI

## 14.64 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

<b>16 位指令</b>	
<b>操作</b>	从存储器加载字节到寄存器，无符号扩展 $RZ \leftarrow \text{zero\_extend}(\text{MEM}[RX + \text{zero\_extend}(\text{offset})])$
<b>语法</b>	ld16.b rz, (rx, disp)
<b>说明</b>	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。
<b>影响标志位</b>	无影响
<b>限制</b>	寄存器的范围为 r0-r7。
<b>异常</b>	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

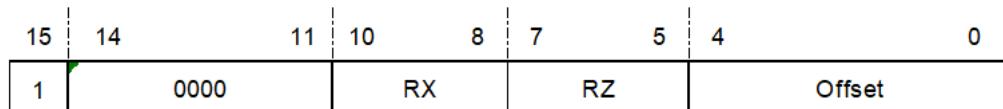
**16 位指令格式：**

图 14.89: LD.B-1

32 位指令	
操作	从存储器加载字节到寄存器，无符号扩展 $RZ \leftarrow \text{zero\_extend}(\text{MEM}[RX + \text{zero\_extend}(\text{offset})])$
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：

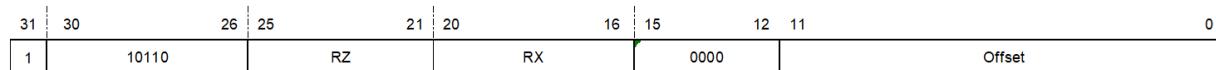


图 14.90: LD.B-2

## 14.65 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 $RZ \leftarrow \text{sign\_extend}(\text{MEM}[RX + \text{zero\_extend}(\text{offset})])$
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	

图 14.91: LD.BS

**14.66 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	$RZ \leftarrow \text{zero\_extend}(\text{MEM}[RX + \text{zero\_extend}(\text{offset} \ll 1)])$
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
操作码	RZ $\leftarrow$ zero_extend(MEM[RX + zero_extend(offset << 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：

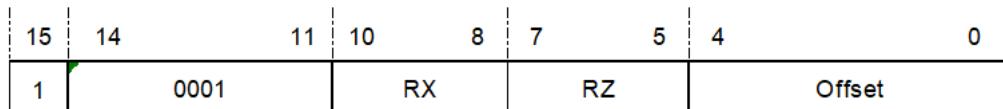


图 14.92: LD.H-1

<b>32 位指令</b>	
<b>操作</b>	从存储器加载半字到寄存器，无符号扩展
<b>操作码</b>	$RZ \leftarrow \text{zero\_extend}(\text{MEM}[RX + \text{zero\_extend}(\text{offset} << 1)])$
<b>语法</b>	ld32.h rz, (rx, disp)
<b>说明</b>	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。
<b>影响标志位</b>	无影响
<b>异常</b>	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

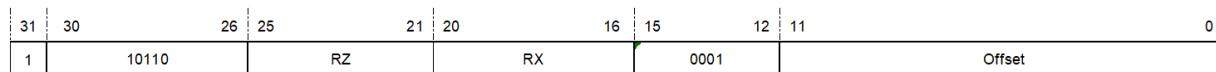
**32 位指令格式：**

图 14.93: LD.H-2

## 14.67 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	$RZ \leftarrow \text{sign\_extend}(\text{MEM}[RX + \text{zero\_extend}(\text{offset} \ll 1)])$
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
语法	$RZ \leftarrow \text{sign\_extend}(\text{MEM}[RX + \text{zero\_extend}(\text{offset} \ll 1)])$ 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	

图 14.94: LD.HS

**14.68 LD.W——字加载指令**

统一化指令	
语法	ld.w rz, (rx, disp)
操作	$RZ \leftarrow \text{MEM}[RX + \text{zero\_extend}(\text{offset} \ll 2)]$
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 ( $\text{disp} < 1024$ ), ld16.w rz, (sp, disp); else if ( $\text{disp}<128$ ) 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
作	$RZ \leftarrow MEM[RX + sign\_extend(offset << 2)]$
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：

ld16.w rz, (rx, disp)

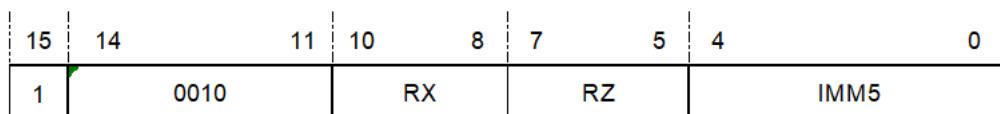


图 14.95: LD.W-1

ld16.w rz, (sp, disp)

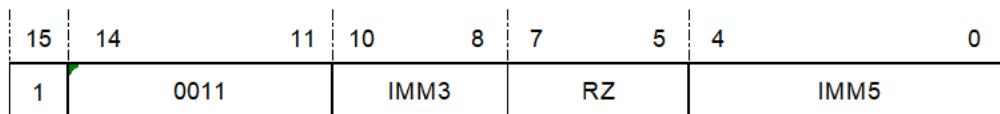


图 14.96: LD.W-2

32 位指令	
操作	从存储器加载字到寄存器
操作	$RZ \leftarrow \text{MEM}[RX + \text{zero\_extend}(\text{offset} \ll 2)]$
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：

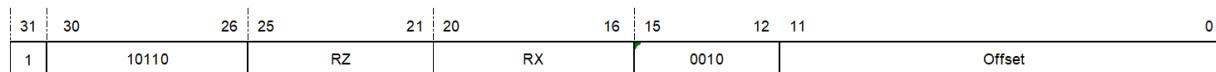


图 14.97: LD.W-3

## 14.69 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 $\begin{aligned} \text{dst} &\leftarrow Y; \text{addr} \leftarrow RX; \\ \text{for } (n = 0; n <= (Z-Y); n++)\{ \\ &\quad \text{Rdst} \leftarrow \text{MEM}[\text{addr}]; \\ &\quad \text{dst} \leftarrow \text{dst} + 1; \\ &\quad \text{addr} \leftarrow \text{addr} + 4; \\ \}\end{aligned}$
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 ( $n = 0; n <= IMM5; n++$ ) { $Rdst \leftarrow MEM[addr];$ $dst \leftarrow dst + 1;$ $addr \leftarrow addr + 4;$ }
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：

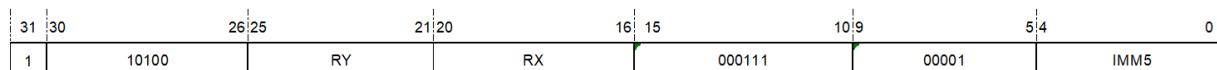


图 14.98: LDM

#### 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

## 14.70 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 $\begin{aligned} \text{dst} &\leftarrow 4; \text{addr} \leftarrow \text{RX}; \\ \text{for } (\text{n} = 0; \text{n} <= 3; \text{n}++)\{ \\ \quad \text{Rdst} &\leftarrow \text{MEM}[\text{addr}]; \\ \quad \text{dst} &\leftarrow \text{dst} + 1; \\ \quad \text{addr} &\leftarrow \text{addr} + 4; \\ \}\end{aligned}$
编译结果	仅存在 32 位指令。 <code>ldq32 r4-r7, (rx);</code>
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 $\leq$ 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：

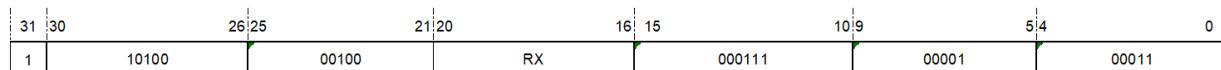


图 14.99: LDQ

## 14.71 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 $RZ \leftarrow \text{zero\_extend}(\text{MEM}[RX + RY << \text{IMM2}])$
编译	仅存在 32 位指令。
结果	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 $RZ \leftarrow \text{zero\_extend}(\text{MEM}[RX + RY << \text{IMM2}])$
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	

图 14.100: LDR.B-1

ldr32.b rz, (rx, ry &lt;&lt; 1)

	31 30	26 25	21 20	16 15	10 9	5 4	0
1	10100	RY	RX	000000	00010	RZ	

图 14.101: LDR.B-2

ldr32.b rz, (rx, ry &lt;&lt; 2)

	31 30	26 25	21 20	16 15	10 9	5 4	0
1	10100	RY	RX	000000	00100	RZ	

图 14.102: LDR.B-3

ldr32.b rz, (rx, ry &lt;&lt; 3)

## 14.72 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 $RZ \leftarrow \text{sign\_extend}(\text{MEM}[RX + RY << \text{IMM2}])$
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

31 30	26 25	21 20	16 15	10 9	5 4	0
1   10100	RY	RX	000000	01000	RZ	

图 14.103: LDR.B-4

32 位指令						
操作	从存储器加载字节到寄存器，有符号扩展 $RZ \leftarrow \text{sign\_extend}(\text{MEM}[RX + RY << \text{IMM2}])$					
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：**

ldr32.bs rz, (rx, ry &lt;&lt; 0)

31 30	26 25	21 20	16 15	10 9	5 4	0
1   10100	RY	RX	000100	00001	RZ	

图 14.104: LDR.BS-1

ldr32.bs rz, (rx, ry &lt;&lt; 1)

31 30	26 25	21 20	16 15	10 9	5 4	0
1   10100	RY	RX	000100	00010	RZ	

图 14.105: LDR.BS-2

ldr32.bs rz, (rx, ry &lt;&lt; 2)

ldr32.bs rz, (rx, ry &lt;&lt; 3)

	31 30	26 25	21 20	16 15	10 9	5 4	0
1	10100	RY	RX	000100	00100	RZ	

图 14.106: LDR.BS-3

	31 30	26 25	21 20	16 15	10 9	5 4	0
1	10100	RY	RX	000100	01000	RZ	

图 14.107: LDR.BS-4

### 14.73 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。 $RZ \leftarrow \text{zero\_extend}(\text{MEM}[RX + RY << \text{IMM2}])$
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 $RZ \leftarrow \text{zero\_extend}(\text{MEM}[RX + RY << \text{IMM2}])$
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 &lt;&lt; 0)

31	30	26 25	21 20	16 15	10 9	5 4	0
1	10100	RY	RX	000001	00001	0001	RZ

图 14.108: LDR.H-1

ldr32.h rz,(rx, ry &lt;&lt; 1)

31	30	26 25	21 20	16 15	10 9	5 4	0
1	10100	RY	RX	000001	00010	00100	RZ

图 14.109: LDR.H-2

ldr32.h rz,(rx, ry &lt;&lt; 2)

31	30	26 25	21 20	16 15	10 9	5 4	0
1	10100	RY	RX	000001	00100	00100	RZ

图 14.110: LDR.H-3

ldr32.h rz,(rx, ry &lt;&lt; 3)

31 30	26 25	21 20	16 15	10 9	5 4	0
1   10100	RY	RX	000001	01000	RZ	

图 14.111: LDR.H-4

## 14.74 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 $RZ \leftarrow \text{sign\_extend}(\text{MEM}[RX + RY << \text{IMM2}])$
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 $RZ \leftarrow \text{sign\_extend}(\text{MEM}[RX + RY << \text{IMM2}])$
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 &lt;&lt; 0)

31	30	26 25	21 20	16 15	10 9	5 4	0
1	10100	RY	RX	000101	00001	RZ	

图 14.112: LDR.HS-1

ldr32.hs rz, (rx, ry &lt;&lt; 1)

31	30	26 25	21 20	16 15	10 9	5 4	0
1	10100	RY	RX	000101	00010	RZ	

图 14.113: LDR.HS-2

ldr32.hs rz, (rx, ry &lt;&lt; 2)

31	30	26 25	21 20	16 15	10 9	5 4	0
1	10100	RY	RX	000101	00100	RZ	

图 14.114: LDR.HS-3

ldr32.hs rz, (rx, ry &lt;&lt; 3)

31 30	26 25	21 20	16 15	10 9	5 4	0
1   10100	RY	RX	000101	01000	RZ	

图 14.115: LDR.HS-4

## 14.75 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 $RZ \leftarrow MEM[RX + RY << IMM2]$
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 $RZ \leftarrow MEM[RX + RY << IMM2]$
语法	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 位指令格式:**

ldr32.w rz, (rx, ry &lt;&lt; 0)

31	30	26 25	21 20	16 15	10 9	5 4	0
1	10100	RY	RX	000010	00001	RZ	

图 14.116: LDR.W-1

ldr32.w rz, (rx, ry &lt;&lt; 1)

31	30	26 25	21 20	16 15	10 9	5 4	0
1	10100	RY	RX	000010	00010	RZ	

图 14.117: LDR.W-2

ldr32.w rz, (rx, ry &lt;&lt; 2)

31	30	26 25	21 20	16 15	10 9	5 4	0
1	10100	RY	RX	000010	00100	RZ	

图 14.118: LDR.W-3

ldr32.w rz, (rx, ry &lt;&lt; 3)

31   30	26   25	21   20	16   15	10   9	5   4	0
1   10100	RY	RX	000010	01000	RZ	

图 14.119: LDR.W-4

## 14.76 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

### 32 位指令格式：

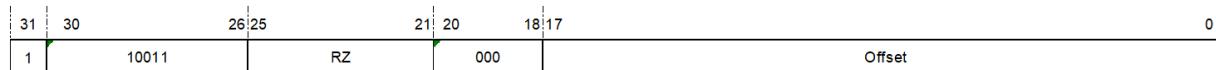


图 14.120: LRS.B

## 14.77 LRS.H——半字符号加载指令

统一化指令	
语法	lrs.h rz, [label]
操作	$RZ \leftarrow \text{zero\_extend}(\text{MEM}[R28 + \text{zero\_extend}(\text{offset} \ll 1)])$
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 R28 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号到寄存器，无符号扩展 $RZ \leftarrow \text{zero\_extend}(\text{MEM}[R28 + \text{zero\_extend}(\text{offset} \ll 1)])$
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

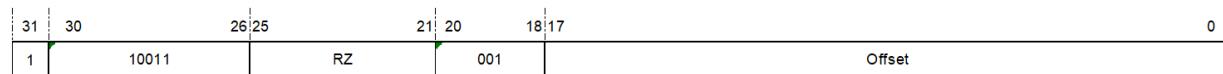
**32 位指令格式:**

图 14.121: LRS.H

**14.78 LRS.W——字符号加载指令**

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	从存储器加载字符到寄存器，无符号扩展
操作数	$RZ \leftarrow \text{zero\_extend}(\text{MEM}[R28 + \text{zero\_extend}(\text{offset} \ll 2)])$
语法	lrs32.w rz, [label]
说明	加载 label 所在位置的字符，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 位指令格式：

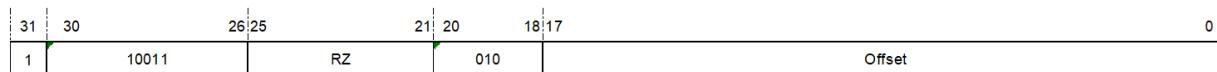


图 14.122: LRS.W

## 14.79 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 $RZ \leftarrow \text{zero\_extend}(\text{MEM}[(\text{PC} + \text{zero\_extend}(\text{offset} \ll 2)) \& 0xffffffff])$
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 $RZ \leftarrow \text{zero\_extend}(\text{MEM}[(\text{PC} + \text{zero\_extend}(\text{offset} \ll 2)) \& 0xffffffff])$
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：

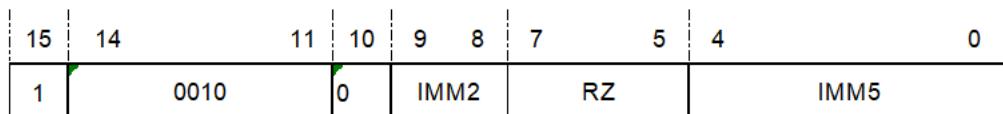


图 14.123: LRW-1

32 位指令	
操作	从存储器加载字到寄存器 $RZ \leftarrow \text{zero\_extend}(\text{MEM}[(\text{PC} + \text{zero\_extend}(\text{offset} << 2)) \& 0xfffffffffc])$
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

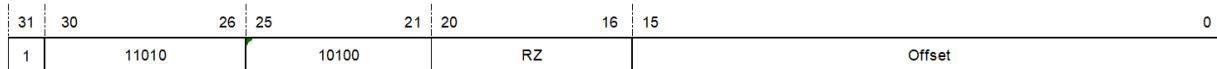
**32 位指令格式:**

图 14.124: LRW-2

## 14.80 LSL——逻辑左移指令

统一化指令		
语法	lsl rz, rx	lsl rz, rx, ry
操作	$RZ \leftarrow RZ << RX[5:0]$	$RZ \leftarrow RX << RY[5:0]$
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	$RZ \leftarrow RZ << RX[5:0]$
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:

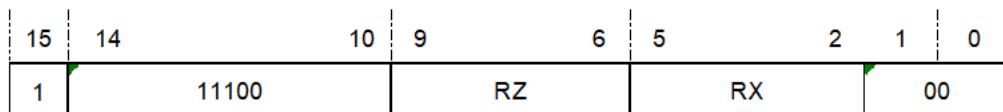


图 14.125: LSL-1

32 位指令	
操作	RZ $\leftarrow$ RX $<<$ 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：

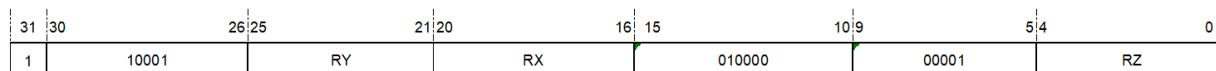


图 14.126: LSL-2

## 14.81 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 $<<$ OIMM5, C $\leftarrow$ RX[32 - OIMM5]
编译	仅存在 32 位指令。
结果	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

<b>32 位指令</b>	
<b>操作</b>	$RZ \leftarrow RX << OIMM5, C \leftarrow RX[32 - OIMM5]$
<b>语法</b>	lslc32 rz, rx, oimm5
<b>说明</b>	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。
<b>影响标志位</b>	$C \leftarrow RX[32 - OIMM5]$
<b>限制</b>	立即数的范围为 1-32。
<b>异常</b>	无

### 32 位指令格式：

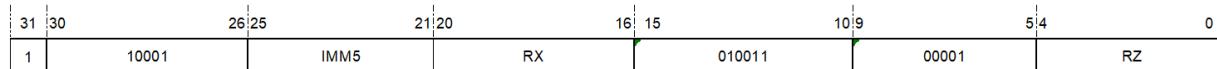


图 14.127: LSLC

#### 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

## 14.82 LSLI——立即数逻辑左移指令

统一化指令	
语法	lsli rz, rx, imm5
操作	$RZ \leftarrow RX << IMM5$
编译	根据寄存器的范围编译为对应的 16 位或 32 位指令。
结果	<pre>if (x&lt;8) and (z&lt;8), then     lsli16 rz, rx, imm5 else     lsli32 rz, rx, imm5</pre>
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	$RZ \leftarrow RX << IMM5$
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：

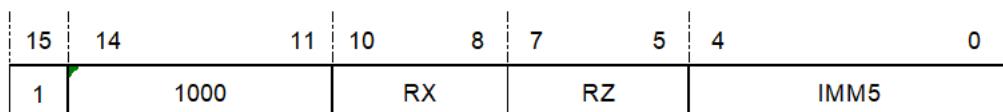


图 14.128: LSLI-1

32 位指令	
操作	RZ $\leftarrow$ RX << 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：

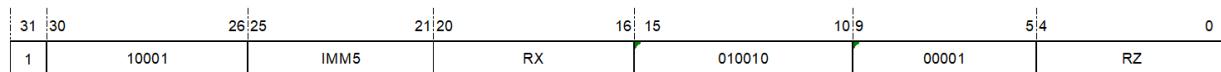


图 14.129: LSLLI-2

### 14.83 LSR——逻辑右移指令

统一化指令		
语法	lsr rz, rx	lsr rz, rx, ry
操作	RZ ← RZ >> RX[5:0]	RZ ← RX >> 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 ← RZ >> 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:

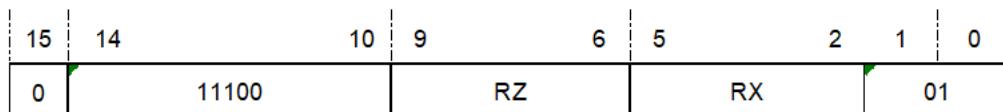


图 14.130: LSR-1

32 位指令	
操作	$RZ \leftarrow RX >> RY[5:0]$
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：

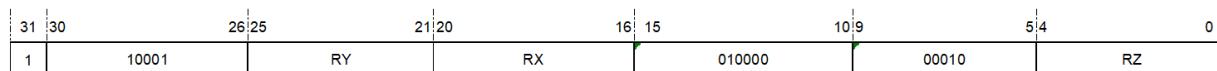


图 14.131: LSR-2

## 14.84 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	$RZ \leftarrow RX >> OIMM5,$ $C \leftarrow RX[OIMM5 - 1]$
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	$RZ \leftarrow RX >> OIMM5, C \leftarrow RX[OIMM5 - 1]$
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：

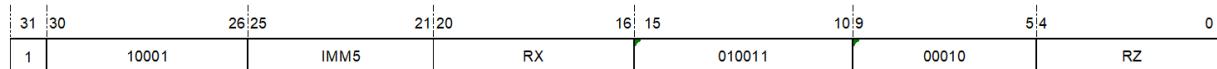


图 14.132: LSRC

#### 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

## 14.85 LSRI——立即数逻辑右移指令

统一化指令	
语法	lsri rz, rx, imm5
操作	$RZ \leftarrow RX >> IMM5$
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	$RZ \leftarrow RX >> IMM5$
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:

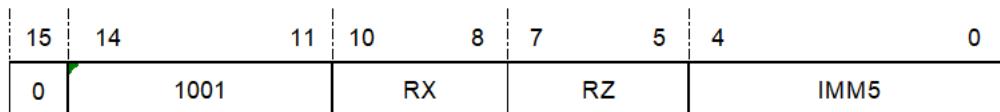


图 14.133: LSRI-1

32 位指令	
操作	RZ $\leftarrow$ RX $>>$ 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：

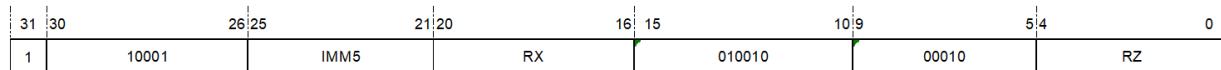


图 14.134: LSRI-2

## 14.86 MFCR——控制寄存器读传送指令

统一化指令	
语法	mfcr rz, 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。 mfcr32 rz, 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

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	mfcr32 rz, 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

**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	

图 14.135: MFCR

**14.87 MOV——数据传送指令**

统一化指令	
语法	mov rz, rx
操作	RZ $\leftarrow$ RX
编译结果	总是编译为 16 位指令。 mov16 rz, rx
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 蚁指令格式:**

15	14	10	9	6	5	2	1	0
1	11011	RZ	RX	11				

图 14.136: MOV-1

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 lsli32 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	

图 14.137: MOV-2

**14.88 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 ← RX; else RZ ← 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	

图 14.138: MOVF

## 14.89 MOVI——立即数数据传送指令

统一化指令	
语法	movi rz, imm
操作	$RZ \leftarrow \text{zero\_extend(imm)}$ ;
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	$RZ \leftarrow \text{zero\_extend(imm8)}$ ;
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		

图 14.139: MOVI-1

32 位指令	
<b>操作</b>	RZ $\leftarrow$ zero_extend(IMM16);
<b>语法</b>	movi32 rz, imm16
<b>说明</b>	将 16 位立即数零扩展至 32 位，然后传送至目的 RZ。
<b>影响标志位</b>	无影响
<b>限制</b>	立即数的范围为 0x0-0xFFFF。
<b>异常</b>	无

32 位指令格式：

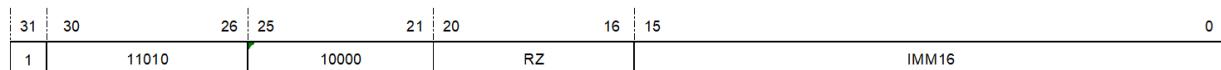


图 14.140: MOVI-2

## 14.90 MOVIH——立即数高位数据传送指令

统一化指令	
<b>语法</b>	movih rz, imm16
<b>操作</b>	RZ $\leftarrow$ zero_extend(IMM16) << 16
<b>编译结果</b>	仅存在 32 位指令。 movih32 rz, imm16
<b>说明</b>	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。
<b>影响标志位</b>	无影响
<b>限制</b>	立即数的范围为 0x0-0xFFFF。
<b>异常</b>	无

32 位指令	
<b>操作</b>	RZ $\leftarrow$ zero_extend(IMM16) << 16
<b>语法</b>	movih32 rz, imm16
<b>说明</b>	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。
<b>影响标志位</b>	无影响
<b>限制</b>	立即数的范围为 0x0-0xFFFF。
<b>异常</b>	无

**32 位指令格式:**

31	30	26	25	21	20	16	15	0
1	11010		10001	RZ			IMM16	

图 14.141: MOVIH

**14.91 MOVT——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	

图 14.142: MOVT

## 14.92 MTCR——控制寄存器写传送指令

统一化指令	
语法	mtcr 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> ← RX
编译结果	仅存在 32 位指令。 mtcr32 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

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> ← RX
语法	mtcr32 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

32 位指令格式：

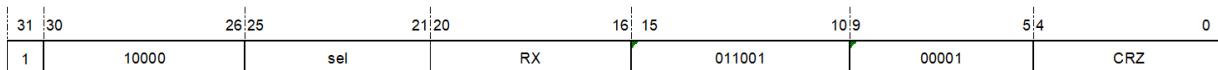


图 14.143: MTCR

## 14.93 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] <- Rz[31:0] + {Rx[31:0] X Ry[31:0]}[31:0]
编译结果	仅存在 32 位指令。 mula.32.l rz, rx, ry

<b>说明:</b>	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。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	$Rz[31:0] \leftarrow Rz[31:0] + \{Rx[31:0] \times Ry[31:0]\}[31:0]$
<b>语法:</b>	mula.32.l rz, rx, ry
<b>说明:</b>	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。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

31 30	26 25	21 20	16 15	10 9	5 4	0
1	11110	RY	RX	000010	0 0 1 0 0	Rz

图 14.144: MULA.32.l

#### 14.94 MULALL.S16.S——带饱和操作的 16 位有符号低半字乘累加指 令

<b>统一化指令</b>	
<b>语法</b>	mulall.s16.s rz, rx, ry
<b>操作</b>	$Rz[31:0] = \text{Saturate}(Rz[31:0] + Rx[15:0] \times Ry[15:0])$
<b>编译结果</b>	仅存在 32 位指令。 mulall.s16.s rz, rx, ry

<b>说 明:</b>	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 0x7FFF FFFF，结果为上饱和值，若加法结果小于下饱和值 0x8000 0000，结果为下饱和值，其它情况为加法结果本身。
<b>影响 标志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

<b>32 位指 令</b>	
<b>操 作:</b>	$Rz[31:0] = \text{Saturate}(Rz[31:0] + Rx[15:0] \times Ry[15:0])$
<b>语 法:</b>	mulall.s16.s rz, rx, ry
<b>说 明:</b>	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 0x7FFF FFFF，结果为上饱和值，若加法结果小于下饱和值 0x8000 0000，结果为下饱和值，其它情况为加法结果本身。
<b>影响 标志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

**指令格式：**

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

图 14.145: MULALL.S16.S

## 14.95 MULA.(U/S)32——32 位 (无/有) 符号乘累加指令

<b>统一化指令</b>	
<b>语法</b>	mula.u32 rz, rx, ry mula.s32 rz, rx, ry
<b>操作</b>	$\{Rz+1[31:0], Rz[31:0]\} = \{Rz+1[31:0], Rz[31:0]\} + Rx[31:0] \times Ry[31:0]$
<b>编译结果</b>	仅存在 32 位指令。 mula.u32 rz, rx, ry mula.s32 rz, rx, ry

<b>说明:</b>	Rx 与 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。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	$\{Rz+1[31:0], Rz[31:0]\} = \{Rz+1[31:0], Rz[31:0]\} + Rx[31:0] \times Ry[31:0]$
<b>语法:</b>	mula.u32 rz, rx, ry mula.s32 rz, rx, ry
<b>说明:</b>	Rx 与 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。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

31 30	26 25	21 20	16 15	10 9	5 4	0
1	11110	RY	RX	1 0 0 0 0 0	S 0 1 0 0	Rz

图 14.146: MULA

## 14.96 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 $RZ \leftarrow RX \times RY;$ $RZ \leftarrow RX \times RZ;$	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有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有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

指令格式:

15	14	10	9	6	5	2	1	0
0	11111		RZ		RX		00	

图 14.147: MULT

<b>32 位指令</b>	
<b>操作:</b>	两个数相乘，结果的低 32 位放入通用寄存器中 RZ $\leftarrow$ RX $\times$ RY;
<b>语法:</b>	mult32 rz, rx, ry
<b>说明:</b>	将通用寄存器 RX 和 RY 的内容相乘后结果的低 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有符号数还是无符号数，结果都相同。
<b>影响标志位:</b>	无影响
<b>异常:</b>	无

指令格式：

31 30	26 25	21 20	16 15	10 9	5 4	0
1   1 0 0 0 1   RY   RX   1 0 0 0 0 1   0 0 0 0 1   Rz						

图 14.148: MULT2

#### 14.97 MUL.(U/S)32——32 位 (无/有) 符号乘法指令

<b>统一化指令</b>	
<b>语法</b>	mul.u32 rz, rx, ry mul.s32 rz, rx, ry
<b>操作</b>	{Rz+1[31:0],Rz[31:0]} = Rx[31:0] X Ry[31:0]
<b>编译结果</b>	仅存在 32 位指令。 mul.u32 rz, rx, ry mul.s32 rz, rx, ry

<b>说明:</b>	Rx 与 Ry 相乘，得到 64 位结果，其中高 32 位存入 Rz+1，低 32 位存入 Rz。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	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 mul.s32 rz, rx, ry
说明:	Rx 与 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:	无
异常:	无

指令格式:

31 30	26 25	21 20	16 15	10 9	5 4	0
1   1    1    1    1    0        RY             RX        1    0    0    0    0    0   S    0    0    0    0    0        Rz						

图 14.149: MUL

## 14.98 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						

图 14.150: MVC

## 14.99 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:

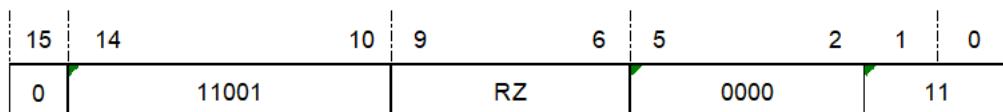


图 14.151: MVCV-1

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	

图 14.152: MVCV-2

## 14.100 NIE——中断嵌套使能指令

统一化指令	
语法	nie
操作	将中断的控制寄存器现场 {EPSR, EPC} 存储到堆栈存储器中, 然后更新堆栈指针寄存器到堆栈存储器的顶端, 打开 PSR.IE 和 PSR.EE; MEM[SP-4] ←EPC; MEM[SP-8] ←EPSR; SP←SP-8; PSR({EE,IE}) ← 1;
编译结果	仅存在 16 位指令 nie;

属性:	特权指令
说明:	将中断的控制寄存器现场 {EPSR, EPC } 保存到堆栈存储器中, 然后更新堆栈指针寄存器到堆栈指针存储器的顶端, 打开中断和异常使能位 PS R.IE 和 PSR.EE。采用堆栈指针寄存器直接寻址方式。
影响标志位:	无影响
异常:	访问错误异常、未对齐异常、特权违反异常

<b>16 位 指 令</b>	
<b>操作:</b>	将中断的控制寄存器现场 {EPSR, EPC} 存储到堆栈存储器中, 然后更新堆栈指针寄存器到堆栈存储器的顶端, 打开 PSR.IE 和 PSR.EE; $\text{MEM}[\text{SP}-4] \leftarrow \text{EPC};$ $\text{MEM}[\text{SP}-8] \leftarrow \text{EPSR};$ $\text{SP} \leftarrow \text{SP}-8;$ $\text{PSR}(\{\text{EE}, \text{IE}\}) \leftarrow 1;$
<b>语法:</b>	nie16
<b>属性:</b>	特权指令
<b>说明:</b>	将中断的控制寄存器现场 {EPSR, EPC } 保存到堆栈存储器中, 然后更新堆栈指针寄存器到堆栈指针存储器的顶端, 打开中断和异常使能位 PS R.IE 和 PSR.EE。采用堆栈指针寄存器直接寻址方式。
<b>影 响 标 志 位:</b>	无影响
<b>异常:</b>	未对齐访问异常、未对齐访问异常、访问错误异常

**指令格式:**

15	14	10	9	8	7	5	4	0
0	00101	0	0	011	00000			

图 14.153: NIE

### 14.101 NIR——中断嵌套返回指令

统一化指令	
语法	nir
操作	从堆栈存储器中载入中断的控制寄存器现场到 {EPSR, EPC } 中，然后更新堆栈指针寄存器到堆栈存储器的顶端；并中断返回 EPSR←MEM[SP]; EPC←MEM[SP+4]; SP←SP+8; PSR←EPSR; PC←EPC;
编译结果	仅存在 16 位指令 nir;

属性：	特权指令
说明：	从堆栈存储器中载入中断的现场到 {EPSR, EPC } 中，然后更新堆栈指针寄存器到堆栈存储器的顶端；PC 值恢复为控制寄存器 EPC 中的值，PSR 值恢复为 EPSR 的值，指令执行从新的 PC 地址处开始。采用堆栈指针寄存器直接寻址方式。
影响标志位：	无影响
异常：	访问错误异常、未对齐异常、特权违反异常

16 位指 令	
操作: 作:	从堆栈存储器中载入中断的控制寄存器现场到 {EPSR, EPC } 中, 然后更新堆栈指针寄存器到堆 栈存储器的顶端; 并中断返回 $\text{EPSR} \leftarrow \text{MEM}[\text{SP}]$ ; $\text{EPC} \leftarrow \text{MEM}[\text{SP}+4]$ ; $\text{SP} \leftarrow \text{SP}+8$ ; $\text{PSR} \leftarrow \text{EPSR}$ ; $\text{PC} \leftarrow \text{EPC}$ ;
语 法: :	nir16
属 性: :	特权指令
说 明: :	从堆栈存储器中载入中断的现场到 {EPSR, EPC } 中, 然后更新堆栈指针寄存器到堆栈存储器的 顶端; PC 值恢复为控制寄存器 EPC 中的值, PSR 值恢复为 EPSR 的值, 指令执行从新的 PC 地址处开始。采用堆栈指针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、未对齐访问异常、访问错误异常

指令格式:

15	14	10	9	8	7	5	4	0
0	00101	0	0	011			00001	

图 14.154: NIR

### 14.102 NOR——按位或非指令

统一化指令		
语法	nor rz, rx	or rz, rx, ry
操作	$RZ \leftarrow !(RZ   RX)$	$RZ \leftarrow !(RX   RY)$
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 ( $x < 16$ ) 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	$RZ \leftarrow !(RZ   RX)$
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：

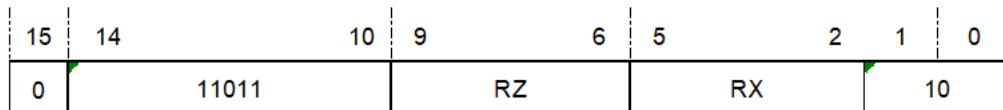


图 14.155: NOR-1

32 位指令	
操作	RZ $\leftarrow$ !(RX   RY)
语法	nor32 rz, rx, ry
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：



图 14.156: NOR-2

### 14.103 NOT——按位非指令

统一化指令		
语法	not rz	not rz, rx
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 not16 rz; else not32 rz, rz;	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 not16 rz; else not32 rz, rx;
说明	将 RZ/RX 的值按位取反，把结果存在 RZ。 注意，该指令是 nor rz, rz 和 nor rz, rx, rx 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	not16 rz
说明	将 RZ 的值按位取反，把结果存在 RZ。 注意，该指令是 nor16 rz, rz 的伪指令。
影响标志位	无影响
异常	无

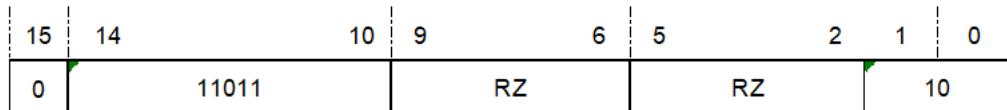
**16 位指令格式:**

图 14.157: NOT-1

32 位指令	
操作	$RZ \leftarrow !(RX)$
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

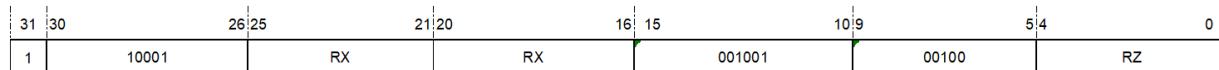
**32 位指令格式:**

图 14.158: NOT-2

**14.104 OR——按位或指令**

统一化指令		
语法	or rz, rx	or rz, rx, ry
操作	$RZ \leftarrow RZ   RX$	$RZ \leftarrow RX   RY$
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 ( $x < 16$ ) 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   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：

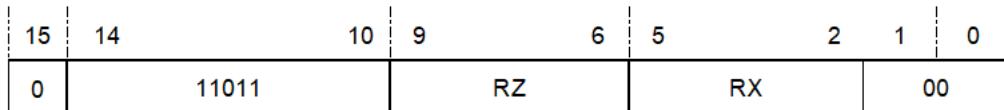


图 14.159: OR-1

32 位指令	
操作	RZ $\leftarrow$ RX   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：

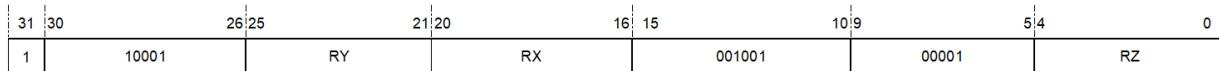


图 14.160: OR-2

## 14.105 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   zero_extend(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   zero_extend(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：

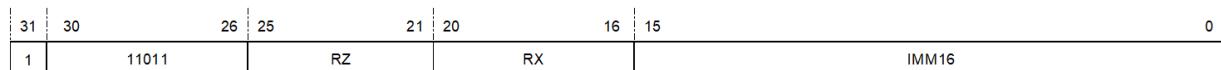


图 14.161: ORI

## 14.106 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； $\begin{aligned} \text{dst} &\leftarrow \{\text{reglist}\}; \text{addr} \leftarrow \text{SP}; \\ \text{foreach } (\text{reglist})\{ & \\ \text{Rdst} &\leftarrow \text{MEM}[\text{addr}]; \\ \text{dst} &\leftarrow \text{next } \{\text{reglist}\}; \\ \text{addr} &\leftarrow \text{addr} + 4; \\ \} & \\ \text{sp} &\leftarrow \text{addr}; \\ \text{PC} &\leftarrow \text{R15} \& 0xffffffe; \end{aligned}$
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 $\begin{aligned} \text{if } (\{\text{reglist}\} < 16), \text{then} & \\ \text{pop16 } \text{reglist}; & \\ \text{else} & \\ \text{pop32 } \text{reglist}; & \end{aligned}$
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 $\leftarrow$ {reglist}; addr $\leftarrow$ SP; foreach ( reglist ){ Rdst $\leftarrow$ MEM[addr]; dst $\leftarrow$ next {reglist}; addr $\leftarrow$ addr + 4; } sp $\leftarrow$ addr; PC $\leftarrow$ R15 & 0xffffffff;
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：

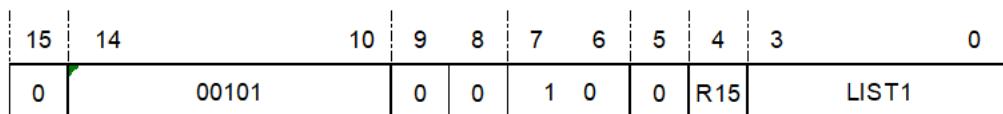


图 14.162: POP-1

### 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

<b>32 位指令</b>	
<b>操作</b>	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 & 0xffffffff;
<b>语法</b>	pop32 reglist
<b>说明</b>	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。
<b>影响</b>	无影响
<b>标志位</b>	
<b>限制</b>	寄存器的范围为 r4 - r11, r15, r16 - r17, r29。
<b>异常</b>	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:****LIST1 域:**

31	30	26	25	21	20	16	15	12	11	10	9	8	7	6	5	4	3	0
1		11010		11110		00000		0000	0	0	0	R28		LIST2	R15		LIST1	

图 14.163: POP-2

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

**R28 域:**

指定寄存器 r28 是否在寄存器列表中。

**0:**

r28 不在寄存器列表中

**1:**

r28 在寄存器列表中

## 14.107 PSRCLR——PSR 位清零指令

<b>统一化指令</b>	
<b>语法</b>	psrclr ee, ie, fe, af 或者操作数也可以为 ee、ie、fe、af 的任意组合。
<b>操作</b>	清除状态寄存器的某一位或几位。 $\text{PSR}(\{\text{EE}, \text{IE}, \text{FE}, \text{AF}\}) \leftarrow 0$
<b>编译结果</b>	仅存在 32 位指令。 <code>psrclr32 ee, ie, fe, af</code>
<b>属性：</b>	特权指令
<b>说明</b>	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
<b>影响标志位</b>	无影响
<b>异常</b>	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

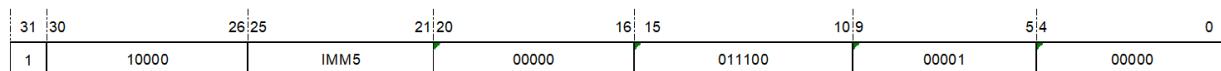
**32 位指令格式：**

图 14.164: PSRCLR

## 14.108 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 $\text{PSR}(\{\text{EE}, \text{IE}, \text{FE}, \text{AF}\}) \leftarrow 1$
编译	仅存在 32 位指令。
结果	<code>psrset32 ee, ie, fe, af</code>
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

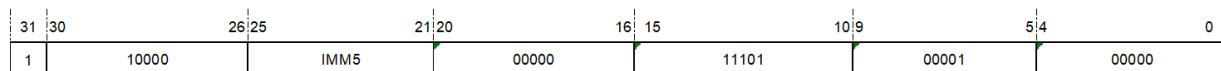
**32 位指令格式：**

图 14.165: PSRSET

## 14.109 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 $\leftarrow$ {reglist}; addr $\leftarrow$ SP; foreach ( reglist ){ MEM[addr] $\leftarrow$ Rsrc; src $\leftarrow$ next {reglist}; addr $\leftarrow$ addr - 4; } sp $\leftarrow$ 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	
<b>操作</b>	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
<b>语法</b>	push16 reglist
<b>说明</b>	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。
<b>影响标志位</b>	无影响
<b>限制</b>	寄存器的范围为 r4 – r11, r15。
<b>异常</b>	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

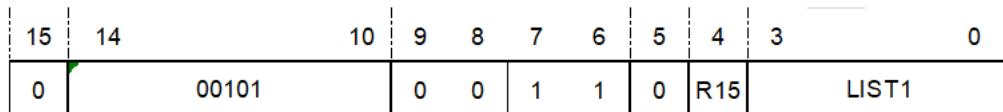
**16 位指令格式：**

图 14.166: PUSH-1

**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 $\leftarrow$ {reglist}; addr $\leftarrow$ SP; foreach ( reglist ){ MEM[addr] $\leftarrow$ Rsrc; src $\leftarrow$ next {reglist}; addr $\leftarrow$ addr - 4; } sp $\leftarrow$ 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:

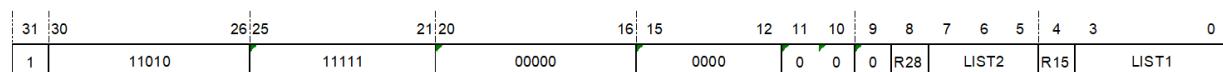


图 14.167: PUSH-2

#### 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

## 14.110 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：

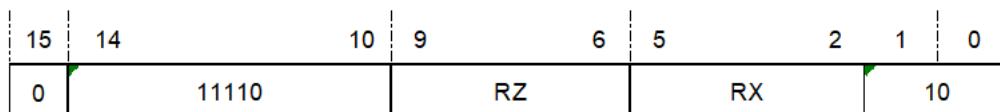


图 14.168: REV16

32 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	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：

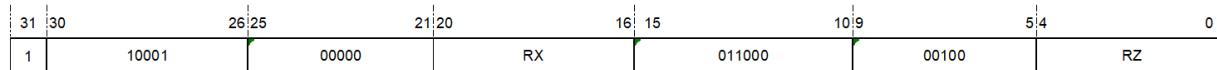


图 14.169: REVB-2

### 14.111 REVH——半字字节倒序指令

统一化指令	
语法	revh rz, rx
操作	RZ[31:24] ← RX[23:16]; RZ[23:16] ← RX[31:24]; RZ[15:8] ← RX[7:0]; RZ[7:0] ← 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	
<b>操作</b>	RZ[31:24] ← RX[23:16]; RZ[23:16] ← RX[31:24]; RZ[15:8] ← RX[7:0]; RZ[7:0] ← RX[15:8];
<b>语法</b>	revh16 rz, rx
<b>说明</b>	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。
<b>影响标志位</b>	无影响
<b>限制</b>	寄存器的范围为 r0-r15。
<b>异常</b>	无

16 位指令格式：

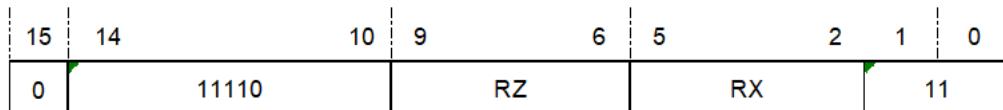


图 14.170: REVH-1

32 位指令	
<b>操作</b>	RZ[31:24] ← RX[23:16]; RZ[23:16] ← RX[31:24]; RZ[15:8] ← RX[7:0]; RZ[7:0] ← RX[15:8];
<b>语法</b>	revh32 rz, rx
<b>说明</b>	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。
<b>影响标志位</b>	无影响
<b>异常</b>	无

32 位指令格式：

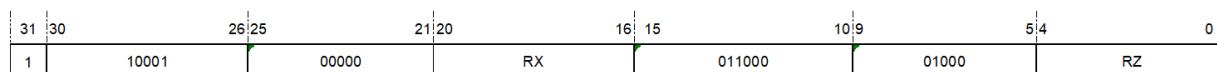


图 14.171: REVH-2

## 14.112 RRTL——循环左移指令

统一化指令		
语法	rotl rz, rx	rotl rz, rx, ry
操作	$RZ \leftarrow RZ <<< RX[5:0]$	$RZ \leftarrow RX <<< RY[5:0]$
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	$RZ \leftarrow RZ <<< RX[5:0]$
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:

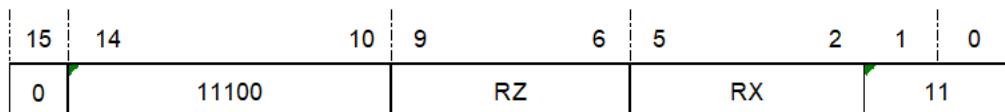


图 14.172: ROTL-1

32 位指令	
操作	$RZ \leftarrow RX <<< RY[5:0]$
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:

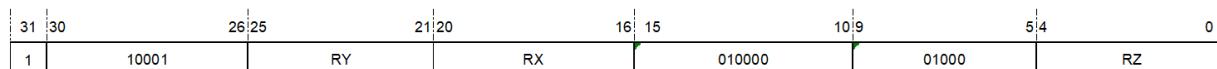


图 14.173: ROTL-2

### 14.113 ROTLI——立即数循环左移指令

统一化指令	
语法	rotli rz, rx, imm5
操作	$RZ \leftarrow RX <<< IMM5$
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 <<< 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：

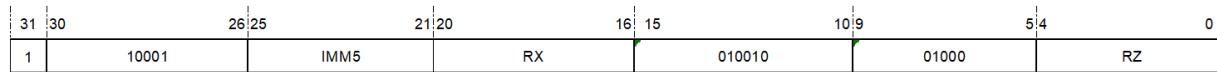


图 14.174: ROTLI

#### 14.114 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						

图 14.175: RSUB

**14.115 RTS——子程序返回指令**

<b>统一化指令</b>	
<b>语法</b>	rts
<b>操作</b>	程序跳转到链接寄存器指定的位置 $PC \leftarrow R15 \& 0xffffffe$
<b>编译结果</b>	总是编译为 16 位指令。 rts16
<b>说明</b>	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。
<b>影响标志位</b>	无影响
<b>异常</b>	无

<b>16 位指令</b>	
<b>操作</b>	程序跳转到链接寄存器指定的位置 $PC \leftarrow R15 \& 0xffffffe$
<b>语法</b>	rts16
<b>说明</b>	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。
<b>影响标志位</b>	无影响
<b>异常</b>	无

**16 位指令格式:**

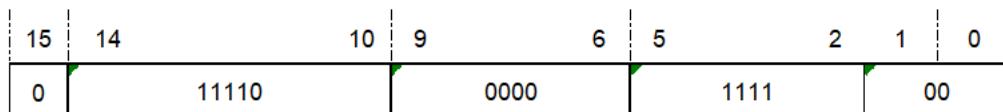


图 14.176: RTS-1

32 位指令	
操作	程序跳转到链接寄存器指定的位置 $PC \leftarrow R15 \& 0xffffffff$
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：

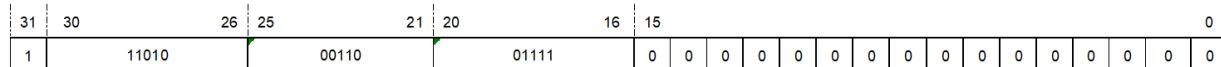


图 14.177: RTS-2

## 14.116 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:

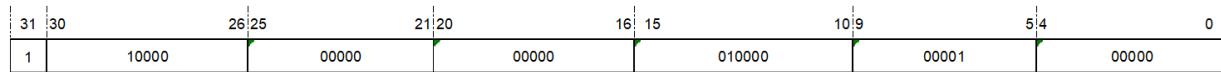


图 14.178: RTE

#### 14.117 SCE——条件执行设置指令

统一化指令	
语法	sce cond
操作	设置其后 4 条指令的条件执行位
编译结果	仅存在 32 位指令 sce32 cond

<b>说 明:</b>	sce 指令用于设置其后 4 条指令的条件执行位。操作数 COND 为 4 位二进制立即数，最低位表示 sce 指令后第一条指令的条件位，次低位表示 sce 指令之后第二条指令的条件位……以此类推。条件位为 1 表示 C 为 1 正常执行，条件位为 0 表示 C 为 0 正常执行。如果 C 位不满足条件位，条件执行指令不产生任何影响。用于判定的 C 位的值以执行 sce 指令时的为准。
<b>影 响 标 志 位:</b>	如果后续 4 条指令产生异常或中断，条件执行位会被保存在 EPSR 或 FPSR 中。
<b>限 制:</b>	sce 指令后面的指令只能是算术运算指令、乘除法指令、立即数寻址方式的字节、半字、字装载存储指令，且这些指令不能影响条件位 C。 操作数为 4 位二进制立即数。
<b>异 常:</b>	无
<b>备 注:</b>	例如指令序列为： sce 0101 mov r1, r0 mov r3, r2 mov r5, r4 mov r7, r6  sce 指令的条件位为 0101。如果执行 sce 指令时，C 位为 0，则第二条和第四条 mov 指令满足执行条件，将结果写入寄存器 3 和寄存器 7；第一条和第三条 mov 指令不满足执行条件，不将结果写入目的寄存器。

<b>32 位 指 令</b>	
<b>操 作:</b>	设置其后 4 条指令的条件执行位 <code>set_condition_execution(COND);</code>
<b>语 法:</b>	<code>sce32 cond</code>
<b>说 明:</b>	sce 指令用于设置其后 4 条指令的条件执行位。操作数 COND 为 4 位二进制立即数，最低位表示 sce 指令后第一条指令的条件位，次低位表示 sce 指令之后第二条指令的条件位……以此类推。条件位为 1 表示 C 为 1 正常执行，条件位为 0 表示 C 为 0 正常执行。如果 C 位不满足条件位，条件执行指令不产生任何影响。用于判定的 C 位的值以执行 sce 指令时的为准。
<b>影 响 标 志 位:</b>	如果后续 4 条指令产生异常或中断，条件执行位会被保存在 EPSR 或 FPSR 中。
<b>限 制:</b>	sce 指令后面的指令只能是算术运算指令、乘除法指令、立即数寻址方式的字节、半字、字装载存储指令，且这些指令不能影响条件位 C。 操作数为 4 位二进制立即数。
<b>异 常:</b>	无
<b>备 注:</b>	例如指令序列为： <code>sce32 0101</code> <code>mov32 r1, r0</code> <code>mov32 r3, r2</code> <code>mov32 r5, r4</code> <code>mov32 r7, r6</code> sce 指令的条件位为 0101。如果执行 sce 指令时，C 位为 0，则第二条和第四条 mov 指令满足执行条件，将结果写入寄存器 3 和寄存器 7；第一条和第三条 mov 指令不满足执行条件，不将结果写入目的寄存器。

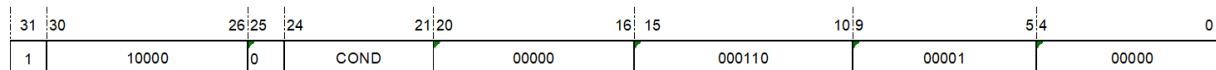
**指令格式：**

图 14.179: SCE

## 14.118 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	$RZ \leftarrow \text{sign\_extend}(RX[\text{MSB:LSB}])$
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	$RZ \leftarrow \text{sign\_extend}(RX[\text{MSB:LSB}])$
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:

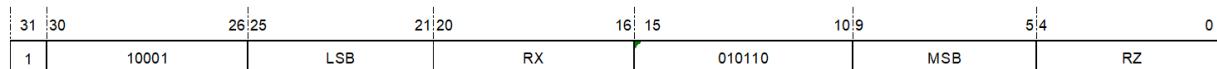


图 14.180: SEXT

#### 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

### 14.119 SEXTB——字节提取并有符号扩展指令

<b>统一化指令</b>	
<b>语法</b>	sextb rz, rx
<b>操作</b>	$RZ \leftarrow \text{sign\_extend}(RX[7:0]);$
<b>编译结果</b>	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;
<b>说明</b>	将 RX 的低字节 (RX[7:0]) 符号扩展至 32 位, 结果存在 RZ 中。
<b>影响标志位</b>	无影响
<b>异常</b>	无

<b>16 位指令</b>	
<b>操作</b>	$RZ \leftarrow \text{sign\_extend}(RX[7:0]);$
<b>语法</b>	sextb16 rz, rx
<b>说明</b>	将 RX 的低字节 (RX[7:0]) 符号扩展至 32 位, 结果存在 RZ 中。
<b>影响标志位</b>	无影响
<b>限制</b>	寄存器的范围为 r0-r15。
<b>异常</b>	无

**16 位指令格式:**

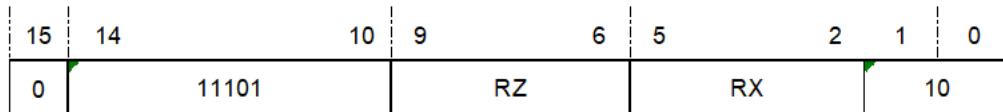


图 14.181: SEXTB-1

32 位指令	
操作	RZ $\leftarrow$ 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：

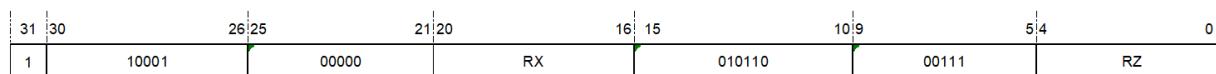


图 14.182: SEXTB-2

## 14.120 SEXTH——半字提取并有符号扩展指令

统一化指令	
语法	sexth rz, rx
操作	RZ $\leftarrow$ 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 sexth16 rz, rx; else sexth32 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 $\leftarrow$ sign_extend(RX[15:0]);
语法	sexth16 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	

图 14.183: SEXTH-1

32 位指令	
操作	$RZ \leftarrow \text{sign\_extend}(RX[15:0]);$
语法	<code>sexth32 rz, rx</code>
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 <code>sext32 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		010110		01111		RZ		

图 14.184: SEXTH-2

## 14.121 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 + zero_extend(offset)] ← 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)] ← 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

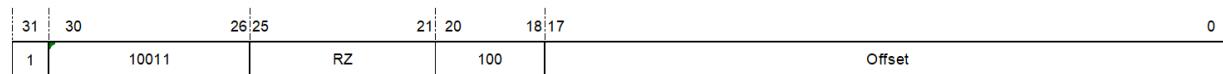
**32 位指令格式:**

图 14.185: SRS.B

**14.122 SRS.H——半字符号存储指令**

统一化指令	
语 法	srs.h rz, [label]
操 作	将寄存器中的最低半字符号存储到存储器中 $MEM[R28 + \text{zero\_extend}(\text{offset} << 1)] \leftarrow RZ[7:0]$
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 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号存储到存储器中 MEM[R28 + zero_extend(offset << 1)] ← RZ[7:0]
语法	srs32.h rz, [label]
说明	将寄存器 RZ 中的最低半字符号存储到到 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 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：

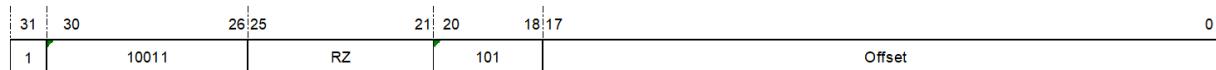


图 14.186: SRS.H

## 14.123 SRS.W——字符串存储指令

统一化指令	
语法	srs.w rz, [label]
操作	将寄存器中的最低字符串存储到存储器中 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 中的最低字符串存储到到 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串存储到存储器中 MEM[R28 + zero_extend(offset << 2)] ← RZ[7:0]
语法	srs32.w rz, [label]
说明	将寄存器 RZ 中的最低字符串存储到到 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

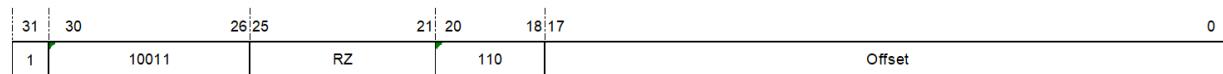
**32 位指令格式:**

图 14.187: SRS.W

**14.124 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 $MEM[RX + \text{zero\_extend}(\text{offset})] \leftarrow RZ[7:0]$
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 ( $\text{disp} < 32$ ) 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

<b>16 位指令</b>	
<b>操作</b>	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]
<b>语法</b>	st16.b rz, (rx, disp)
<b>说明</b>	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。
<b>影响标志位</b>	无影响
<b>限制</b>	寄存器的范围为 r0-r7。
<b>异常</b>	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

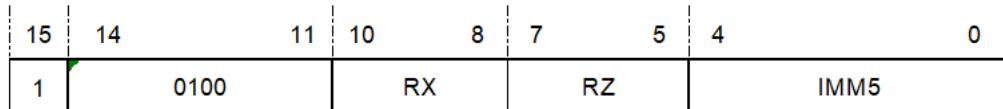
**16 位指令格式：**

图 14.188: ST.B-1

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)] ← 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

### 32 位指令格式：

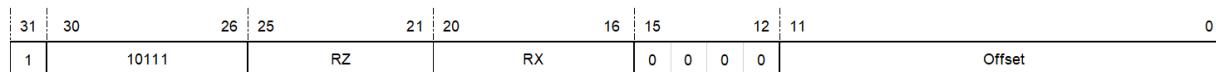


图 14.189: ST.B-2

## 14.125 ST.H——半字存储指令

<b>统一化指令</b>	
<b>语法</b>	st.h rz, (rx, disp)
<b>操作</b>	将寄存器中的最低字节存储到存储器中 MEM[RX + zero_extend(offset<<1)] ← RZ[15:0]
<b>编译结果</b>	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) ;
<b>说明</b>	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。
<b>影响标志位</b>	无影响
<b>异常</b>	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

<b>16 位指令</b>	
<b>操作</b>	将寄存器中的低半字存储到存储器中 MEM[RX + zero_extend(offset << 1)] ← RZ[15:0]
<b>语法</b>	st16.h rz, (rx, disp)
<b>说明</b>	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。
<b>影响标志位</b>	无影响
<b>限制</b>	寄存器的范围为 r0-r7。
<b>异常</b>	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：**

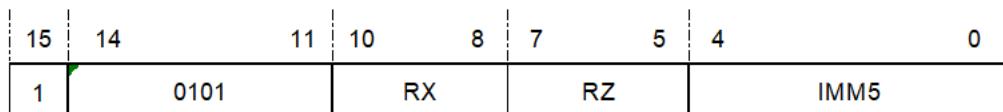


图 14.190: ST.H-1

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 << 1)] ← 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

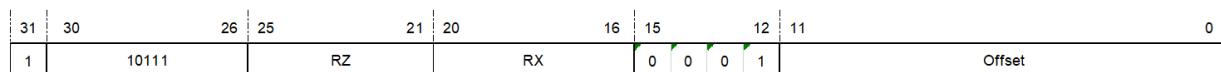
**32 位指令格式：**

图 14.191: ST.H-2

## 14.126 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
作用	$MEM[RX + \text{zero\_extend}(\text{offset} << 2)] \leftarrow RZ[31:0]$
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：

st16.w rz, (rx, disp)

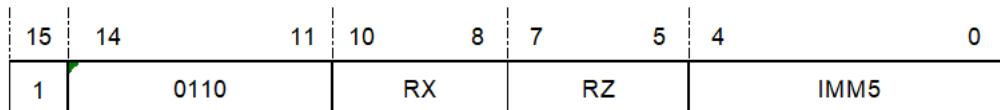


图 14.192: ST.W-1

st16.w rz, (sp, disp)

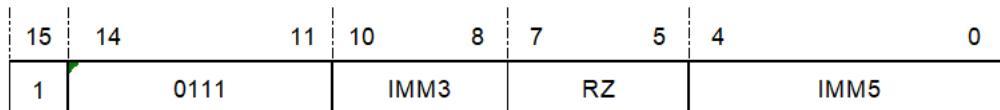


图 14.193: ST.W-2

32 位指令	
操作	将寄存器中的字存储到存储器中
操作码	MEM[RX + zero_extend(offset << 2)] ← 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：

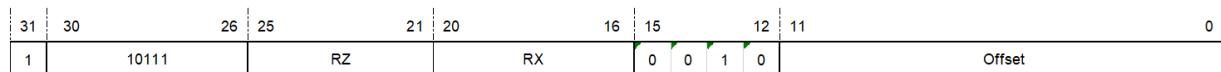


图 14.194: ST.W-3

## 14.127 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。 $\text{src} \leftarrow Y; \text{addr} \leftarrow RX;$ $\text{for } (n = 0; n <= (Z-Y); n++)\{$ $\quad \text{MEM}[\text{addr}] \leftarrow \text{Rsrc};$ $\quad \text{src} \leftarrow \text{src} + 1;$ $\quad \text{addr} \leftarrow \text{addr} + 4;$ $\}$
编译结果	仅存在 32 位指令。 $\text{stm32 ry-rz, (rx)}$
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 $\leftarrow$ Y; addr $\leftarrow$ RX; for (n = 0; n $\leq$ IMM5; n++){ MEM[addr] $\leftarrow$ Rsrc; src $\leftarrow$ src + 1; addr $\leftarrow$ 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：

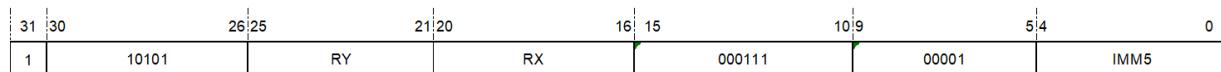


图 14.195: STM

#### 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

### 14.128 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：**

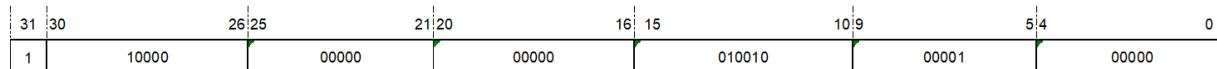


图 14.196: STOP

## 14.129 STQ——连续四字存储指令

统一化指令	
语法	stq r4-r7, (rx)
操作	<p>将寄存器 R4—R7 中的字依次存储到一片连续的存储器地址上。</p> <pre>src ← 4; addr ← RX; for (n = 0; n &lt;= 3; n++) {     MEM[addr] ← Rsrc;     src ← src + 1;     addr ← addr + 4; }</pre>
编译结果	<p>仅存在 32 位指令。</p> <p>stq32 r4-r7, (rx);</p>
说明	<p>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。</p> <p>注意，该指令是 stm r4-r7, (rx) 的伪指令。</p>
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 R4—R7 中的字依次存储到一片连续的存储器地址上。 src $\leftarrow$ 4; addr $\leftarrow$ RX; for (n = 0; n $\leq$ 3; n++) { MEM[addr] $\leftarrow$ Rsrc; src $\leftarrow$ src + 1; addr $\leftarrow$ 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：

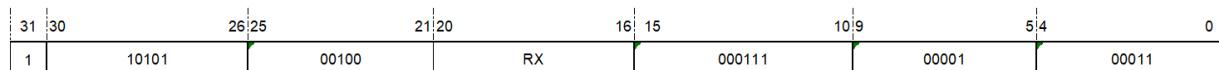


图 14.197: STQ

## 14.130 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 $MEM[RX + RY << IMM2] \leftarrow RZ[7:0]$
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 $MEM[RX + RY << IMM2] \leftarrow RZ[7:0]$
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	

图 14.198: STR.B-1

str32.b rz, (rx, ry &lt;&lt; 1)

	31 30	26 25	21 20	16 15	10 9	5 4	0
1	10101	RY	RX	000000	00010	RZ	

图 14.199: STR.B-2

str32.b rz, (rx, ry &lt;&lt; 2)

	31 30	26 25	21 20	16 15	10 9	5 4	0
1	10101	RY	RX	000000	00100	RZ	

图 14.200: STR.B-3

str32.b rz, (rx, ry &lt;&lt; 3)

### 14.131 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

31 30	26 25	21 20	16 15	10 9	5 4	0
1   10101	RY	RX	000000	01000	RZ	

图 14.201: STR.B-4

32 位指令	
<b>操作</b>	将寄存器中的低半字存储到存储器中 $MEM[RX + RY << IMM2] \leftarrow RZ[15:0]$
<b>语法</b>	str32.h rz, (rx, ry << 0) str32.h rz, (rx, ry << 1) str32.h rz, (rx, ry << 2) str32.h rz, (rx, ry << 3)
<b>说明</b>	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。
<b>影响标志位</b>	无影响
<b>异常</b>	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 &lt;&lt; 0)

31 30	26 25	21 20	16 15	10 9	5 4	0
1   10101	RY	RX	000001	00001	RZ	

图 14.202: STR.H-1

str32.h rz, (rx, ry &lt;&lt; 1)

31 30	26 25	21 20	16 15	10 9	5 4	0
1   10101	RY	RX	000001	00010	RZ	

图 14.203: STR.H-2

str32.h rz, (rx, ry &lt;&lt; 2)

str32.h rz, (rx, ry &lt;&lt; 3)

	31 30	26 25	21 20	16 15	10 9	5 4	0
1	10101	RY	RX	000001	00100	RZ	

图 14.204: STR.H-3

	31 30	26 25	21 20	16 15	10 9	5 4	0
1	10101	RY	RX	000001	01000	RZ	

图 14.205: STR.H-4

### 14.132 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 $MEM[RX + RY << IMM2] \leftarrow RZ[31:0]$
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 << IMM2] ← RZ[31:0]
语法	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 位指令格式：**

str32.w rz, (rx, ry &lt;&lt; 0)

31	30	26 25	21 20	16 15	10 9	5 4	0
1	10101	RY	RX	000010	00001	RZ	

图 14.206: STR.W-1

str32.w rz, (rx, ry &lt;&lt; 1)

31	30	26 25	21 20	16 15	10 9	5 4	0
1	10101	RY	RX	000010	00010	RZ	

图 14.207: STR.W-2

str32.w rz, (rx, ry &lt;&lt; 2)

31	30	26 25	21 20	16 15	10 9	5 4	0
1	10101	RY	RX	000010	00100	RZ	

图 14.208: STR.W-3

str32.w rz, (rx, ry &lt;&lt; 3)

31 30	26 25	21 20	16 15	10 9	5 4	0
1   10101	RY	RX	000010	01000	RZ	

图 14.209: STR.W-4

### 14.133 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 ← 借位	$RZ \leftarrow RX - RY - (!C)$ , C ← 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, rx, ry;	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 subc rz, 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 ← 借位	
异常	无	

16 位指令	
操作	$RZ \leftarrow RZ - RX - (!C)$ , C ← 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 ← 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	

图 14.210: SUBC-1

32 位指令	
操作	$RZ \leftarrow RX - RY - (!C)$ , $C \leftarrow$ 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 ← 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		00000		01000		RZ	

图 14.211: SUBC-2

## 14.134 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 ( $\text{oimm12} < 257$ ) and ( $z < 8$ ), then subi16 rz, oimm8; else subi32 rz, rx, oimm12;	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if ( $\text{oimm12} < 8$ ) and ( $z < 8$ ) and ( $x < 8$ ), then subi16 rz, rx, oimm3; elsif ( $x == z$ ) and ( $z < 8$ ) and ( $\text{oimm12} < 257$ ), 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:

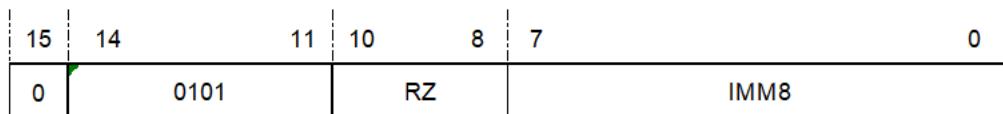


图 14.212: SUBI-1

**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

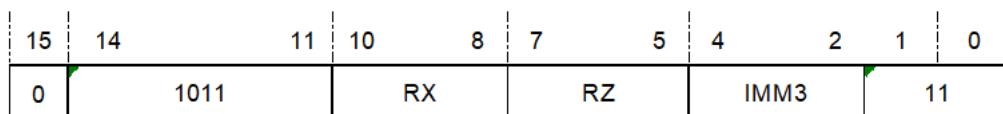
**16 位指令格式 2:**

图 14.213: SUBI-2

**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**

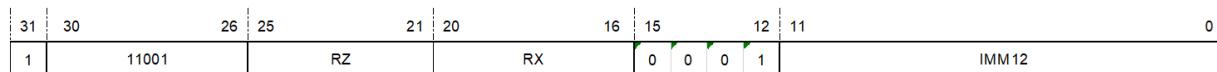
减 2

.....

**111**

减 8

<b>32 位指令</b>	
<b>操作</b>	RZ $\leftarrow$ RX - zero_extend(OIMM12)
<b>语法</b>	subi32 rz, rx, oimm12
<b>说明</b>	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。
<b>影响标</b>	无影响
<b>志位</b>	
<b>限制</b>	立即数的范围为 0x1-0x1000。
<b>异常</b>	无

**32 位指令格式：****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

### 14.135 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	$SP \leftarrow SP - \text{zero\_extend(IMM)}$
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	$SP \leftarrow SP - \text{zero\_extend(IMM)}$
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：

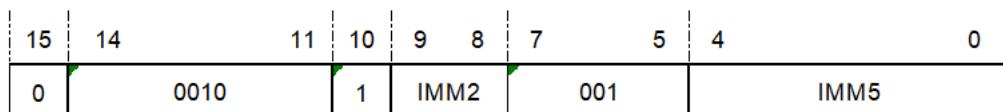


图 14.214: SUBI(SP)

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

### 14.136 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<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;	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 ← 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		10	

图 14.215: SUBU-1

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		01	

图 14.216: SUBU-2

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	25	21 20	16 15	9	5 4	0
1	10001	RY	RX	00000	00100	RZ	

图 14.217: SUBU-3

### 14.137 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：

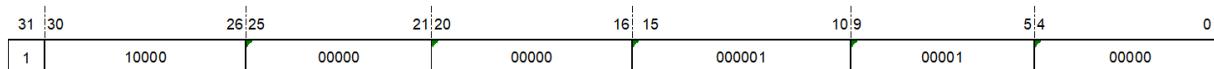


图 14.218: SYNC

### 14.138 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	仅存在 32 位指令。
说明	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

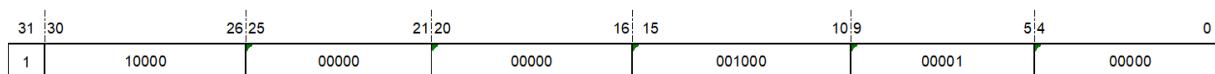


图 14.219: TRAP-1

trap32 1

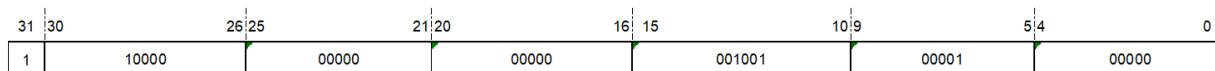


图 14.220: TRAP-2

trap32 2

	31 30	26 25	21 20	16 15	10 9	5 4	0
1	10000	00000	00000	001010	00001	00000	

图 14.221: TRAP-3

trap32 3

	31 30	26 25	21 20	16 15	10 9	5 4	0
1	10000	00000	00000	001011	00001	00000	

图 14.222: TRAP-4

### 14.139 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：

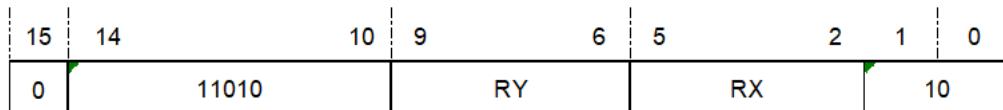


图 14.223: TST-1

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：

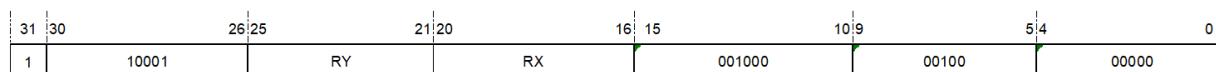


图 14.224: TST-2

## 14.140 TSTNBZ——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：

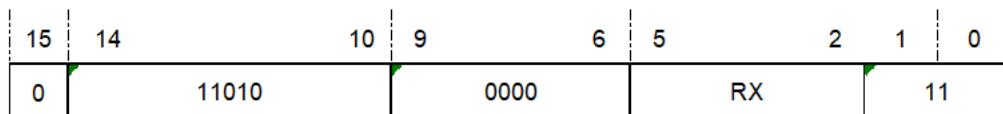


图 14.225: TSTNBZ-1

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

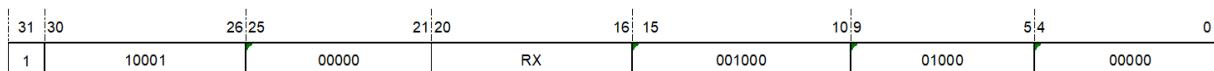
**32 位指令格式：**

图 14.226: TSTNBZ-2

### 14.141 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：

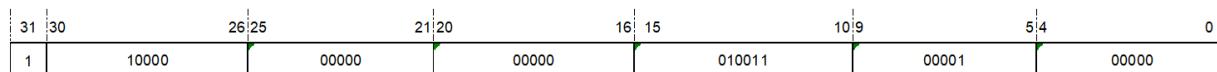


图 14.227: WAIT

### 14.142 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：

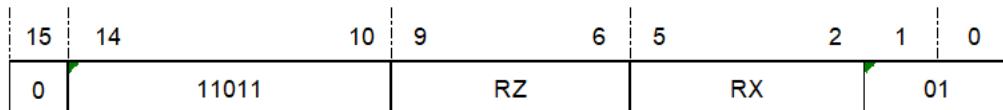


图 14.228: XOR-1

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

图 14.229: XOR-2

#### 14.143 XORI——立即数按位异或指令

统一化指令	
语法	xori rz, rx, imm16
操作	$RZ \leftarrow RX \wedge zero\_extend(IMM12)$
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	$RZ \leftarrow RX \wedge zero\_extend(IMM12)$
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		IMM 12	

图 14.230: XORI

## 14.144 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 ← 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:



图 14.231: XSR

### 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

### 14.145 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 $\leftarrow$ 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 $\leftarrow$ 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:

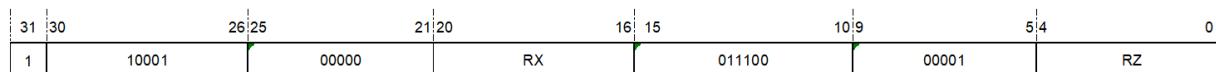


图 14.232: XTRB0

### 14.146 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 $\leftarrow$ 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 $\leftarrow$ 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:

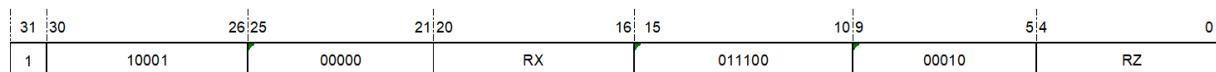


图 14.233: XTRB1

### 14.147 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 $\leftarrow$ 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 $\leftarrow$ 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:

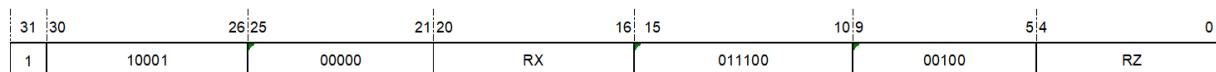


图 14.234: XTRB2

### 14.148 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 $\leftarrow$ 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 $\leftarrow$ 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:

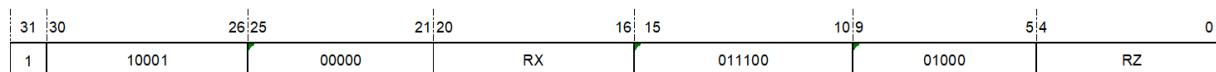


图 14.235: XTRB3

### 14.149 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	$RZ \leftarrow \text{zero\_extend}(RX[\text{MSB:LSB}])$
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 $\leftarrow$ zero_extend(RX[MSB:LSB])
语法	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 位指令格式:

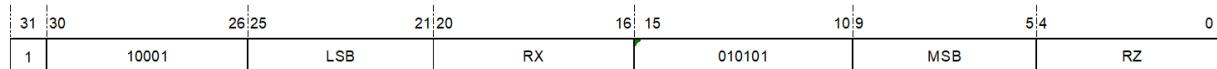


图 14.236: ZEXT

#### 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

#### 14.150 ZEXTB——字节提取并无符号扩展指令

统一化指令	
语法	<code>zextb rz, rx</code>
操作	<code>RZ ← zero_extend(RX[7:0]);</code>
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。 <code>if (x&lt;16) and (z&lt;16), then</code> <code>zextb16 rz, rx;</code> <code>else</code> <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。
影响标志位	无影响
异常	无

16 位指令	
操作	<code>RZ ← zero_extend(RX[7:0]);</code>
语法	<code>zextb16 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。
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：

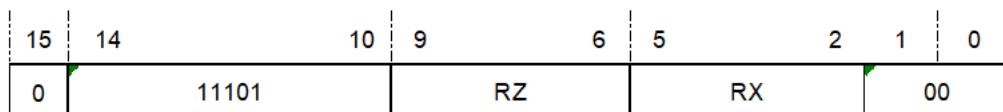


图 14.237: ZEXTB-1

32 位指令	
操作	RZ $\leftarrow$ zero_extend(RX[7:0]);
语法	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。 注意，该指令是 zext32 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：

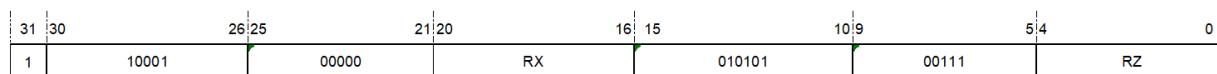


图 14.238: ZEXTB-2

### 14.151 ZEXTH——半字提取并无符号扩展指令

统一化指令	
语法	zexth rz, rx
操作	RZ $\leftarrow$ 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 zexth16 rz, rx; else zexth32 rz, rx
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 $\leftarrow$ zero_extend(RX[15:0]);
语法	zexth16 rz, rx
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

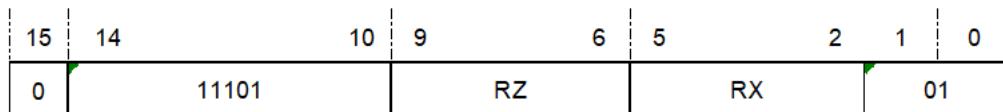
**16 位指令格式:**

图 14.239: ZEXTH-1

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:**

图 14.240: ZEXTH-2

## 第十五章 附录 B DSP 指令集术语表

以下以指令英文首字母为序对 DSP 指令子集进行表述。

每条 DSP 指令的助记符包含三部分，每部分以. 分开，第一部分标识指令的基本操作，第二部分标识操作数的数据类型，第三部分标识对运算结果的处理。以“带饱和操作的 16 位并行有符号加法指令-PADD.S16.S”为例，PADD 表示这是一条并行加法指令，S16 表示操作数为有符号半字，S 表示对运算结果取饱和操作。

指令助记符中常见的字母及其表示意义如下：

助记符	英文表述	意义
第一部 分	P(Parallel)	并行计算，表示单指令多数据操作。
第一部 分	R(fRaction)	小数操作。
第一部 分	X(Unalign)	操作数位宽不对齐，或操作数位置不对齐。
第一部 分	H(Half)	取平均值。
第二部 分	(S/U)/(8/16/32)(有符号/无符号)/(字节/半字/字) 如果不带符号位指示，则表示该条指令的操作对符号位不关心。	
第三部 分	S(Saturate)	饱和操作。
第三部 分	R(Round)	舍入操作。
第三部 分	E(Extension)	扩位操作，即操作数进扩位操作后参与运算产生结果。

## 15.1 PADD.8——8 位并行加法指令

<b>统一化 指令</b>	
<b>语法</b>	padd.8 rz, rx,ry
<b>操作</b>	$Rz[31:24] = Rx[31:24] + Ry[31:24]$ $Rz[23:16] = Rx[23:16] + Ry[23:16]$ $Rz[15:8] = Rx[15:8] + Ry[15:8]$ $Rz[7:0] = Rx[7:0] + Ry[7:0]$
<b>编译结果</b>	仅存在 32 位指令。 padd.8 rz, rx,ry

<b>说明:</b>	以字节为单位，将 Rx 的 4 个字节分别与 Ry 的 4 个字节相加，加法结果存入 Rz。
<b>影响标志位:</b>	无影响
<b>异常:</b>	无
<b>32 位指令</b>	
<b>操作:</b>	$Rz[31:24] = Rx[31:24] + Ry[31:24]$ $Rz[23:16] = Rx[23:16] + Ry[23:16]$ $Rz[15:8] = Rx[15:8] + Ry[15:8]$ $Rz[7:0] = Rx[7:0] + Ry[7:0]$
<b>语法:</b>	padd.8 rz, rx,ry
<b>说明:</b>	以字节为单位，将 Rx 的 4 个字节分别与 Ry 的 4 个字节相加，加法结果存入 Rz。
<b>影响标志位:</b>	无影响
<b>异常:</b>	无

**指令格式:**

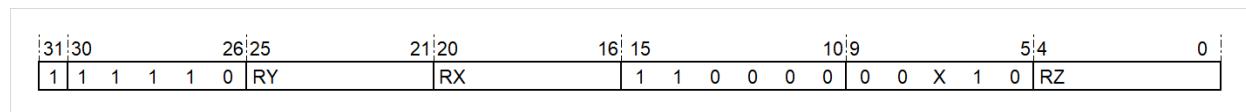


图 15.1: PADD.8

## 15.2 PADD.16——16 位并行加法指令

<b>统一化指令</b>	
<b>语法</b>	padd.16 rz, rx, ry
<b>操作</b>	$Rz[31:16] = Rx[31:16] + Ry[31:16]$ $Rz[15:0] = Rx[15:0] + Ry[15:0]$
<b>编译结果</b>	仅存在 32 位指令。 padd.16 rz, rx, ry

<b>说明:</b>	将 Rx 的高低半字分别与 Ry 的高低半字相加，加法结果存入 Rz。
<b>影响标志位:</b>	无影响
<b>异常:</b>	无
<b>32 位指令</b>	
<b>操作:</b>	$Rz[31:16] = Rx[31:16] + Ry[31:16]$ $Rz[15:0] = Rx[15:0] + Ry[15:0]$
<b>语法:</b>	padd.16 rz, rx, ry
<b>说明:</b>	将 Rx 的高低半字分别与 Ry 的高低半字相加，加法结果存入 Rz。
<b>影响标志位:</b>	无影响
<b>异常:</b>	无

**指令格式:**

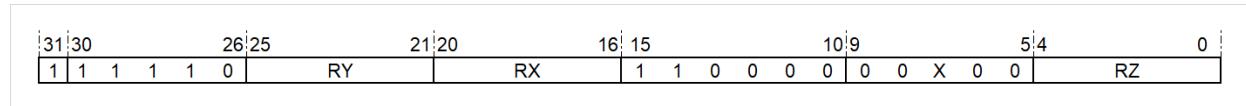


图 15.2: PADD.16

### 15.3 PADD.(U/S)8.S——带饱和操作的 8 位并行（无/有）符号加法指令

统一化指令	
语法	padd.u8.s rz, rx,ry padd.s8.s rz, rx,ry
操作	Rz[31:24] = Saturate(Rx[31:24] + Ry[31:24]) Rz[23:16] = Saturate(Rx[23:16] + Ry[23:16]) Rz[15:8] = Saturate(Rx[15:8] + Ry[15:8]) Rz[7:0] = Saturate(Rx[7:0] + Ry[7:0])
编译结果	仅存在 32 位指令。 padd.u8.s rz, rx,ry padd.s8.s rz, rx,ry

<b>说 明:</b>	以字节为单位, 将 Rx 的 4 个字节分别与 Ry 的 4 个字节相加, 加法结果做饱和处理后存入 Rz。 对于无符号数, 饱和处理的过程为, 若 8 位加法结果大于饱和值 0xFF, 结果取 0xFF, 其他情况取加法结果本身。 对于有符号数, 饱和处理的过程为, 若 8 位加法结果大于上饱和值 0x7F, 结果取 0x7F, 若 8 位加法结果小于下饱和值 0x80, 结果取 0x80, 其他情况取加法结果本身。
<b>影 响 标 志 位:</b>	无影响
<b>异 常:</b>	无
<b>32 位 指 令</b>	
<b>操 作:</b>	Rz[31:24] = Saturate(Rx[31:24] + Ry[31:24]) Rz[23:16] = Saturate(Rx[23:16] + Ry[23:16]) Rz[15:8] = Saturate(Rx[15:8] + Ry[15:8]) Rz[7:0] = Saturate(Rx[7:0] + Ry[7:0])
<b>语 法:</b>	padd.u8.s rz, rx,ry padd.s8.s rz, rx,ry
<b>说 明:</b>	以字节为单位, 将 Rx 的 4 个字节分别与 Ry 的 4 个字节相加, 加法结果做饱和处理后存入 Rz。 对于无符号数, 饱和处理的过程为, 若 8 位加法结果大于饱和值 0xFF, 结果取 0xFF, 其他情况取加法结果本身。 对于有符号数, 饱和处理的过程为, 若 8 位加法结果大于上饱和值 0x7F, 结果取 0x7F, 若 8 位加法结果小于下饱和值 0x80, 结果取 0x80, 其他情况取加法结果本身。
<b>影 响 标 志 位:</b>	无影响
<b>异 常:</b>	无

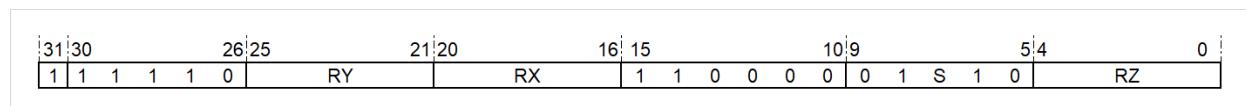
**指令格式:**

图 15.3: PADD.(US)8.S

## 15.4 PADD.(U/S)16.S——带饱和操作的 16 位并行（无/有）符号加法指令

统一化指令	
语法	padd.u16.s rz, rx,ry padd.s16.s rz, rx,ry
操作	$Rz[31:16] = \text{Saturate}(Rx[31:16] + Ry[31:16])$ $Rz[15:0] = \text{Saturate}(Rx[15:0] + Ry[15:0])$
编译结果	仅存在 32 位指令。 padd.u16.s rz, rx,ry padd.s16.s rz, rx,ry

<b>说 明:</b>	将 Rx 和 Ry 的高低半字分别相加，加法结果做饱和处理后存入 Rz。对于无符号数，饱和处理的过程为，若 16 位加法结果大于饱和值 0xFFFF，结果取 0xFFFF，其他情况取加法结果本身。对于有符号数，饱和处理的过程为，若 16 位加法结果大于上饱和值 0x7FFF，结果取 0x7FFF，若 16 位加法结果小于下饱和值 0x8000，结果取 0x8000，其他情况取加法结果本身。
<b>影 响 标 志 位:</b>	无影响
<b>异 常:</b>	无
<b>32 位 指 令</b>	
<b>操 作:</b>	Rz[31:16] = Saturate(Rx[31:16] + Ry[31:16]) Rz[15:0] = Saturate(Rx[15:0] + Ry[15:0])
<b>语 法:</b>	padd.u16.s rz, rx, ry padd.s16.s rz, rx, ry
<b>说 明:</b>	将 Rx 和 Ry 的高低半字分别相加，加法结果做饱和处理后存入 Rz。对于无符号数，饱和处理的过程为，若 16 位加法结果大于饱和值 0xFFFF，结果取 0xFFFF，其他情况取加法结果本身。对于有符号数，饱和处理的过程为，若 16 位加法结果大于上饱和值 0x7FFF，结果取 0x7FFF，若 16 位加法结果小于下饱和值 0x8000，结果取 0x8000，其他情况取加法结果本身。
<b>影 响 标 志 位:</b>	无影响
<b>异 常:</b>	无
<b>指 令 格 式:</b>	

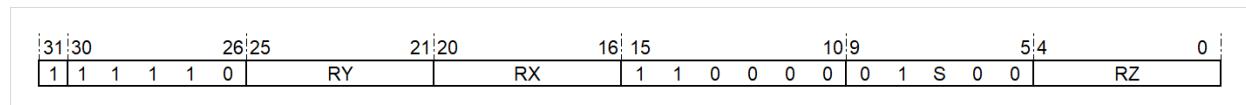


图 15.4: PADD.(US)16.S

## 15.5 ADD.(U/S)32.S——带饱和操作的 32 位（无/有）符号加法指令

统一化指令		
语法	操作	编译结果
add.u32.s rz, rx,ry add.s32.s rz, rx,ry	Rz[31:0] = Saturate(Rx[31:0] + Ry[31:0])	仅存在 32 位指令。 add.u32.s rz, rx,ry add.s32.s rz, rx,ry

<b>说 明:</b>	Rx 与 Ry 相加, 加法结果做饱和处理后存入 Rz。 对于无符号数, 饱和处理的过程为, 若 32 位加法结果大于上饱和值 0xFFFF FFFF, 结果取 0xFFFF FFFF, 其他情况取加法结果本身。 对于有符号数, 饱和处理的过程为, 若 32 位加法结果大于上饱和值 0x7FFF FFFF, 结果取 0x7FFF FFFF, 若 32 位加法结果小于下饱和值 0x8000 0000, 结果取 0x8000 0000, 其他情况取加法结果本身。
<b>影 响 标 志 位:</b>	无影响
<b>异 常:</b>	无
<b>32 位 指 令</b>	
<b>操 作:</b>	$Rz[31:0] = \text{Saturate}(Rx[31:0] + Ry[31:0])$
<b>语 法:</b>	add.u32.s rz, rx,ry add.s32.s rz, rx,ry
<b>说 明:</b>	Rx 与 Ry 相加, 加法结果做饱和处理后存入 Rz。 对于无符号数, 饱和处理的过程为, 若 32 位加法结果大于上饱和值 0xFFFF FFFF, 结果取 0xFFFF FFFF, 其他情况取加法结果本身。 对于有符号数, 饱和处理的过程为, 若 32 位加法结果大于上饱和值 0x7FFF FFFF, 结果取 0x7FFF FFFF, 若 32 位加法结果小于下饱和值 0x8000 0000, 结果取 0x8000 0000, 其他情况取加法结果本身。
<b>影 响 标 志 位:</b>	无影响
<b>异 常:</b>	无

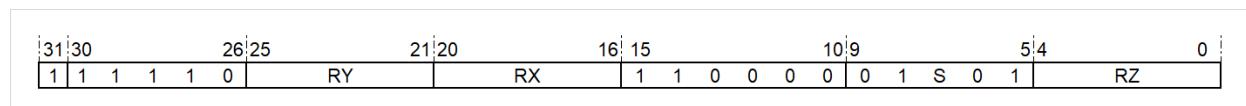
**指令格式:**

图 15.5: ADD.(US)32.S

## 15.6 PSUB.8——8 位并行减法指令

<b>统一化指令</b>	
<b>语法</b>	psub.8 rz, rx,ry
<b>操作</b>	$Rz[31:24] = Rx[31:24] - Ry[31:24]$ $Rz[23:16] = Rx[23:16] - Ry[23:16]$ $Rz[15:8] = Rx[15:8] - Ry[15:8]$ $Rz[7:0] = Rx[7:0] - Ry[7:0]$
<b>编译结果</b>	仅存在 32 位指令。 psub.8 rz, rx,ry

<b>说明:</b>	以字节为单位, 将 Rx 的 4 个字节分别与 Ry 的 4 个字节相减, 减法结果存入 Rz。
<b>影响标志位:</b>	无影响
<b>异常:</b>	无
<b>32 位指令</b>	
<b>操作:</b>	$Rz[31:24] = Rx[31:24] - Ry[31:24]$ $Rz[23:16] = Rx[23:16] - Ry[23:16]$ $Rz[15:8] = Rx[15:8] - Ry[15:8]$ $Rz[7:0] = Rx[7:0] - Ry[7:0]$
<b>语法:</b>	psub.8 rz, rx,ry
<b>说明:</b>	以字节为单位, 将 Rx 的 4 个字节分别与 Ry 的 4 个字节相减, 减法结果存入 Rz。
<b>影响标志位:</b>	无影响
<b>异常:</b>	无

**指令格式:**

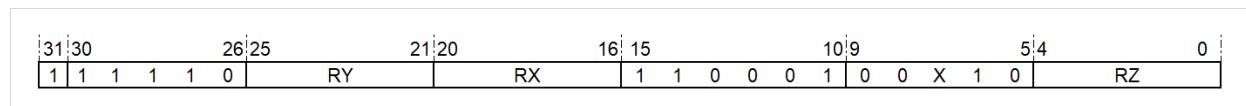


图 15.6: PSUB.8

## 15.7 PSUB.16——16 位并行减法指令

<b>统一化指令</b>	
<b>语法</b>	psub.16 rz, rx,ry
<b>操作</b>	$Rz[31:16] = Rx[31:16] - Ry[31:16]$ $Rz[15:0] = Rx[15:0] - Ry[15:0]$
<b>编译结果</b>	仅存在 32 位指令。 psub.16 rz, rx,ry

<b>说明:</b>	将 Rx 的高低半字分别与 Ry 的高低半字相减，减法结果存入 Rz。
<b>影响标志位:</b>	无影响
<b>异常:</b>	无
<b>32 位指令</b>	
<b>操作:</b>	$Rz[31:16] = Rx[31:16] - Ry[31:16]$ $Rz[15:0] = Rx[15:0] - Ry[15:0]$
<b>语法:</b>	psub.16 rz, rx,ry
<b>说明:</b>	将 Rx 的高低半字分别与 Ry 的高低半字相减，减法结果存入 Rz。
<b>影响标志位:</b>	无影响
<b>异常:</b>	无

**指令格式:**

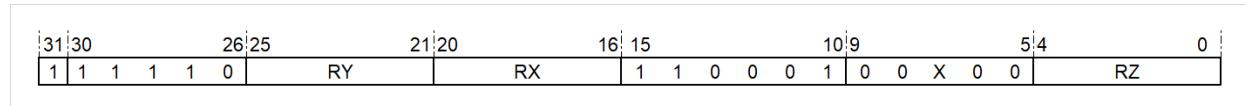


图 15.7: PSUB.16

## 15.8 PSUB.(U/S)8.S——带饱和操作的 8 位并行（无/有）符号减法指令

统一化指令	
语法	psub.u8.s rz, rx, ry psub.s8.s rz, rx, ry
操作	Rz[31:24] = Saturate(Rx[31:24] - Ry[31:24]) Rz[23:16] = Saturate(Rx[23:16] - Ry[23:16]) Rz[15:8] = Saturate(Rx[15:8] - Ry[15:8]) Rz[7:0] = Saturate(Rx[7:0] - Ry[7:0])
编译结果	仅存在 32 位指令。 psub.u8.s rz, rx, ry psub.s8.s rz, rx, ry

<b>说 明:</b>	以字节为单位, 将 Rx 的 4 个字节分别与 Ry 的 4 个字节相减, 减法结果做饱和处理后存入 Rz。 对于无符号数, 饱和处理的过程为, 若 8 位减法结果小于饱和值 0x0, 结果取 0x0, 其他情况取减法结果本身。 对于有符号数, 饱和处理的过程为, 若 8 位减法结果大于上饱和值 0x7F, 结果取 0x7F, 若 8 位减法结果小于下饱和值 0x80, 结果取 0x80, 其他情况取减法结果本身。
<b>影 响 标 志 位:</b>	无影响
<b>异 常:</b>	无
<b>32 位 指 令</b>	
<b>操 作:</b>	Rz[31:24] = Saturate(Rx[31:24] - Ry[31:24]) Rz[23:16] = Saturate(Rx[23:16] - Ry[23:16]) Rz[15:8] = Saturate(Rx[15:8] - Ry[15:8]) Rz[7:0] = Saturate(Rx[7:0] - Ry[7:0])
<b>语 法:</b>	psub.u8.s rz, rx,ry psub.s8.s rz, rx,ry
<b>说 明:</b>	以字节为单位, 将 Rx 的 4 个字节分别与 Ry 的 4 个字节相减, 减法结果做饱和处理后存入 Rz。 对于无符号数, 饱和处理的过程为, 若 8 位减法结果小于饱和值 0x0, 结果取 0x0, 其他情况取减法结果本身。 对于有符号数, 饱和处理的过程为, 若 8 位减法结果大于上饱和值 0x7F, 结果取 0x7F, 若 8 位减法结果小于下饱和值 0x80, 结果取 0x80, 其他情况取减法结果本身。
<b>影 响 标 志 位:</b>	无影响
<b>异 常:</b>	无

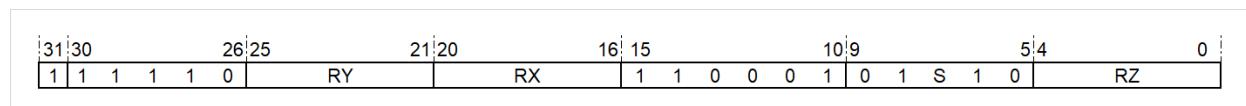
**指令格式:**

图 15.8: PSUB.(US)8.S

## 15.9 PSUB.(U/S)16.S——带饱和操作的 16 位并行（无/有）符号减法指令

统一化指令	
语法	psub.u16.s rz, rx,ry psub.s16.s rz, rx,ry
操作	$Rz[31:16] = \text{Saturate}(Rx[31:16] - Ry[31:16])$ $Rz[15:0] = \text{Saturate}(Rx[15:0] - Ry[15:0])$
编译结果	仅存在 32 位指令。 psub.u16.s rz, rx,ry psub.s16.s rz, rx,ry

<b>说 明:</b>	将 Rx 和 Ry 的高低半字分别相减，减法结果做饱和处理后存入 Rz。对于无符号数，饱和处理的过程为，若 16 位减法结果小于饱和值 0x0，结果取 0x0，其他情况取减法结果本身。 对于有符号数，饱和处理的过程为，若 16 位减法结果大于上饱和值 0x 7FFF，结果取 0x7FFF，若 16 位减法结果小于下饱和值 0x8000，结果取 0x8000，其他情况取减法结果本身。
<b>影 响 标 志 位:</b>	无影响
<b>异 常:</b>	无
<b>32 位 指 令</b>	
<b>操 作:</b>	Rz[31:16] = Saturate(Rx[31:16] - Ry[31:16]) Rz[15:0] = Saturate(Rx[15:0] - Ry[15:0])
<b>语 法:</b>	psub.u16.s rz, rx, ry psub.s16.s rz, rx, ry
<b>说 明:</b>	将 Rx 和 Ry 的高低半字分别相减，减法结果做饱和处理后存入 Rz。对于无符号数，饱和处理的过程为，若 16 位减法结果小于饱和值 0x0，结果取 0x0，其他情况取减法结果本身。 对于有符号数，饱和处理的过程为，若 16 位减法结果大于上饱和值 0x 7FFF，结果取 0x7FFF，若 16 位减法结果小于下饱和值 0x8000，结果取 0x8000，其他情况取减法结果本身。
<b>影 响 标 志 位:</b>	无影响
<b>异 常:</b>	无

**指令格式:**

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

图 15.9: PSUB.(US)16.S

## 15.10 SUB.(U/S)32.S——带饱和操作的 32 位（无/有）符号减法指令

统一化指令	
语法	sub.u32.s rz, rx,ry sub.s32.s rz, rx,ry
操作	$Rz[31:0] = \text{Saturate}(Rx[31:0] - Ry[31:0])$
编译结果	仅存在 32 位指令。 sub.u32.s rz, rx,ry sub.s32.s rz, rx,ry

说明:	Rx 与 Ry 相减，减法结果做饱和处理后存入 Rz。 对于无符号数，饱和处理的过程为，若 32 位减法结果小于饱和值 0x0，结果取 0x0，其他情况取减法结果本身。 对于有符号数，饱和处理的过程为，若 32 位减法结果大于上饱和值 0x7FFF FFFF，结果取 0x7FFF FFFF，若 32 位减法结果小于下饱和值 0x8000 0000，结果取 0x8000 0000，其他情况取减法结果本身。
影响标志位:	无影响
异常:	无

<b>32位指令</b>	
<b>操作:</b>	$Rz[31:0] = \text{Saturate}(Rx[31:0] - Ry[31:0])$
<b>语法:</b>	sub.u32.s rz, rx, ry sub.s32.s rz, rx, ry
<b>说明:</b>	Rx 与 Ry 相减，减法结果做饱和处理后存入 Rz。 对于无符号数，饱和处理的过程为，若 32 位减法结果小于饱和值 0x0，结果取 0x0，其他情况取减法结果本身。 对于有符号数，饱和处理的过程为，若 32 位减法结果大于上饱和值 0x7FFF FFFF，结果取 0x7FFF FFFF，若 32 位减法结果小于下饱和值 0x8000 0000，结果取 0x8000 0000，其他情况取减法结果本身。
<b>影响标志位:</b>	无影响
<b>异常:</b>	无

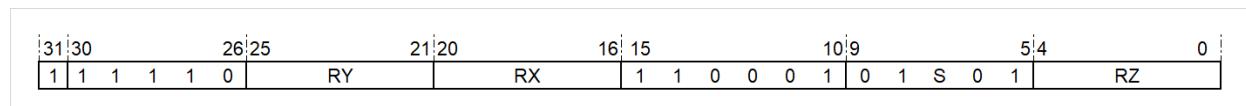
**指令格式:**

图 15.10: SUB.(US)32.S

### 15.11 PADDH.(U/S)8——8 位并行（无/有）符号加法求平均值指令

<b>统一化指令</b>	
<b>语法</b>	paddh.u8 rz, rx,ry paddh.s8 rz, rx,ry
<b>操作</b>	$Rz[31:24] = (Rx[31:24] + Ry[31:24])/2$ $Rz[23:16] = (Rx[23:16] + Ry[23:16])/2$ $Rz[15:8] = (Rx[15:8] + Ry[15:8])/2$ $Rz[7:0] = (Rx[7:0] + Ry[7:0])/2$
<b>编译结果</b>	仅存在 32 位指令。 paddh.u8 rz, rx,ry paddh.s8 rz, rx,ry

<b>说明:</b>	以字节为单位，将 Rx 的 4 个字节分别与 Ry 的 4 个字节相加，加法结果除以 2 后存入 Rz。
<b>影响标志位:</b>	无影响
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	$Rz[31:24] = (Rx[31:24] + Ry[31:24])/2$ $Rz[23:16] = (Rx[23:16] + Ry[23:16])/2$ $Rz[15:8] = (Rx[15:8] + Ry[15:8])/2$ $Rz[7:0] = (Rx[7:0] + Ry[7:0])/2$
<b>语法:</b>	paddh.u8 rz, rx,ry paddh.s8 rz, rx,ry
<b>说明:</b>	以字节为单位，将 Rx 的 4 个字节分别与 Ry 的 4 个字节相加，加法结果除以 2 后存入 Rz。
<b>影响标志位:</b>	无影响
<b>异常:</b>	无

**指令格式:**

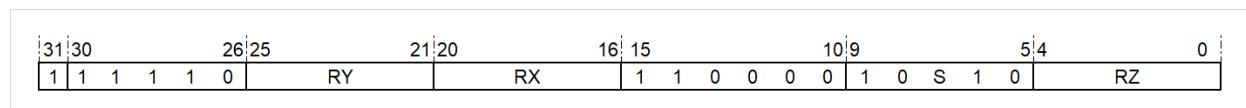


图 15.11: PADDH.(US)8

## 15.12 PADDH.(U/S)16——16 位并行（无/有）符号加法求平均值指令

统一化指令	
语法	paddh.u16 rz, rx,ry paddh.s16 rz, rx,ry
操作	$Rz[31:16] = (Rx[31:16] + Ry[31:16])/2$ $Rz[15:0] = (Rx[15:0] + Ry[15:0])/2$
编译结果	仅存在 32 位指令。 paddh.u16 rz, rx,ry paddh.s16 rz, rx,ry

说明:	将 Rx 的高低半字分别与 Ry 的高低半字相加，加法结果除以 2 后存入 Rz。
影响标志位:	无影响
异常:	无

32 位指令	
操作:	$Rz[31:16] = (Rx[31:16] + Ry[31:16])/2$ $Rz[15:0] = (Rx[15:0] + Ry[15:0])/2$
语法:	paddh.u16 rz, rx,ry paddh.s16 rz, rx,ry
说明:	将 Rx 的高低半字分别与 Ry 的高低半字相加，加法结果除以 2 后存入 Rz。
影响标志位:	无影响
异常:	无

指令格式：

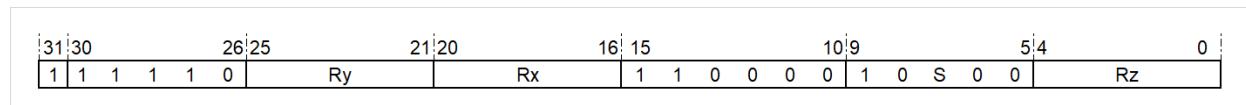


图 15.12: PADDH.(US)16

### 15.13 ADDH.(U/S)32——32 位（无/有）符号加法求平均值指令

<b>统一化指令</b>	
<b>语法</b>	addh.u32 rz, rx,ry addh.s32 rz, rx,ry
<b>操作</b>	$Rz[31:0] = (Rx[31:0] + Ry[31:0])/2$
<b>编译结果</b>	仅存在 32 位指令。 addh.u32 rz, rx,ry addh.s32 rz, rx,ry

<b>说明:</b>	Rx 与 Ry 相加，加法结果除以 2 后存入 Rz。
<b>影响标志位:</b>	无影响
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	$Rz[31:0] = (Rx[31:0] + Ry[31:0])/2$
<b>语法:</b>	addh.u32 rz, rx,ry addh.s32 rz, rx,ry
<b>说明:</b>	Rx 与 Ry 相加，加法结果除以 2 后存入 Rz。
<b>影响标志位:</b>	无影响
<b>异常:</b>	无

**指令格式：**

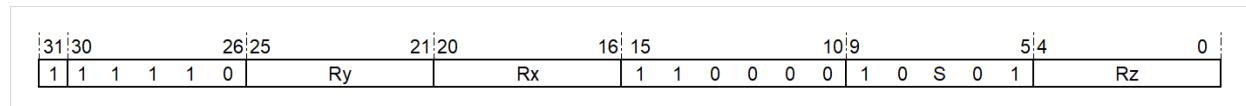


图 15.13: ADDH.(US)32

### 15.14 PSUBH.(U/S)8——8 位并行（无/有）符号减法求平均值指令

统一化 指令	
语法	psubh.u8 rz, rx, ry psubh.s8 rz, rx, ry
操作	$Rz[31:24] = (Rx[31:24] - Ry[31:24])/2$ $Rz[23:16] = (Rx[23:16] - Ry[23:16])/2$ $Rz[15:8] = (Rx[15:8] - Ry[15:8])/2$ $Rz[7:0] = (Rx[7:0] - Ry[7:0])/2$
编译结 果	仅存在 32 位指令。 psubh.u8 rz, rx, ry psubh.s8 rz, rx, ry

说明:	以字节为单位，将 Rx 的 4 个字节分别与 Ry 的 4 个字节相减，减法结果除以 2 后存入 Rz。
影响标志位:	无影响
异常:	无

32 位指 令	
操作:	$Rz[31:24] = (Rx[31:24] - Ry[31:24])/2$ $Rz[23:16] = (Rx[23:16] - Ry[23:16])/2$ $Rz[15:8] = (Rx[15:8] - Ry[15:8])/2$ $Rz[7:0] = (Rx[7:0] - Ry[7:0])/2$
语法:	psubh.u8 rz, rx, ry psubh.s8 rz, rx, ry
说明:	以字节为单位，将 Rx 的 4 个字节分别与 Ry 的 4 个字节相减，减法结果除以 2 后存入 Rz。
影响标志位:	无影响
异常:	无

指令格式：

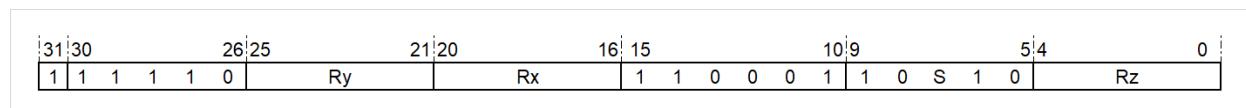


图 15.14: PSUBH.(US)8

## 15.15 PSUBH.(U/S)16——16 位并行（无/有）符号减法求平均值指令

<b>统一化指令</b>	
<b>语法</b>	psubh.u16 rz, rx,ry psubh.s16 rz, rx,ry
<b>操作</b>	$Rz[31:16] = (Rx[31:16] - Ry[31:16])/2$ $Rz[15:0] = (Rx[15:0] - Ry[15:0])/2$
<b>编译结果</b>	仅存在 32 位指令。 psubh.u16 rz, rx,ry psubh.s16 rz, rx,ry

<b>说明:</b>	将 Rx 的高低半字分别与 Ry 的高低半字相减，减法结果除以 2 后存入 Rz。
<b>影响标志位:</b>	无影响
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	$Rz[31:16] = (Rx[31:16] - Ry[31:16])/2$ $Rz[15:0] = (Rx[15:0] - Ry[15:0])/2$
<b>语法:</b>	psubh.u16 rz, rx,ry psubh.s16 rz, rx,ry
<b>说明:</b>	将 Rx 的高低半字分别与 Ry 的高低半字相减，减法结果除以 2 后存入 Rz。
<b>影响标志位:</b>	无影响
<b>异常:</b>	无

**指令格式：**

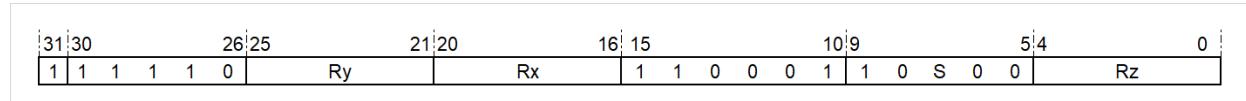


图 15.15: PSUBH.(US)16

## 15.16 SUBH.(U/S)32——32 位（无/有）符号减法求平均值指令

<b>统一化指令</b>	
<b>语法</b>	subh.u32 rz, rx,ry subh.s32 rz, rx,ry
<b>操作</b>	$Rz[31:0] = (Rx[31:0] - Ry[31:0])/2$
<b>编译结果</b>	仅存在 32 位指令。 subh.u32 rz, rx,ry subh.s32 rz, rx,ry

<b>说明:</b>	Rx 与 Ry 相减，减法结果除以 2 后存入 Rz。
<b>影响标志位:</b>	无影响
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	$Rz[31:0] = (Rx[31:0] - Ry[31:0])/2$
<b>语法:</b>	subh.u32 rz, rx,ry subh.s32 rz, rx,ry
<b>说明:</b>	Rx 与 Ry 相减，减法结果除以 2 后存入 Rz。
<b>影响标志位:</b>	无影响
<b>异常:</b>	无

指令格式：

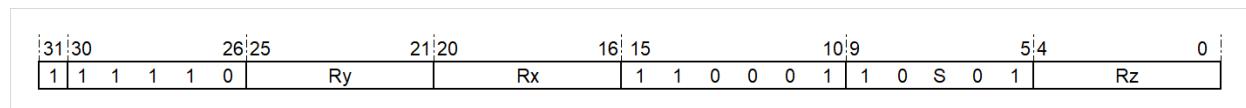


图 15.16: SUBH.(US)32

## 15.17 PASX.16——16 位并行交叉加减指令

<b>统一化指令</b>	
<b>语法</b>	pasx.16 rz, rx,ry
<b>操作</b>	$Rz[31:16] = Rx[31:16] + Ry[15:0]$ $Rz[15:0] = Rx[15:0] - Ry[31:16]$
<b>编译结果</b>	仅存在 32 位指令。 pasx.16 rz, rx,ry

<b>说明:</b>	Rx 的高 16 位与 Ry 的低 16 位相加, 结果存入 Rz 的高 16 位; Rx 的低 16 位与 Ry 的高 16 位相减, 结果存入 Rz 的低 16 位。
<b>影响标志位:</b>	无影响
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	$Rz[31:16] = Rx[31:16] + Ry[15:0]$ $Rz[15:0] = Rx[15:0] - Ry[31:16]$
<b>语法:</b>	pasx.16 rz, rx, ry
<b>说明:</b>	Rx 的高 16 位与 Ry 的低 16 位相加, 结果存入 Rz 的高 16 位; Rx 的低 16 位与 Ry 的高 16 位相减, 结果存入 Rz 的低 16 位。
<b>影响标志位:</b>	无影响
<b>异常:</b>	无

**指令格式:**

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 1 1 0	Ry	Rx	1 1 0 0 1 0	0 0 x 1 1	Rz	

图 15.17: PASX.16

## 15.18 PSAX.16——16 位并行交叉减加指令

<b>统一化指令</b>	
<b>语法</b>	psax.16 rz, rx, ry
<b>操作</b>	$Rz[31:16] = Rx[31:16] - Ry[15:0]$ $Rz[15:0] = Rx[15:0] + Ry[31:16]$
<b>编译结果</b>	仅存在 32 位指令。 psax.16 rz, rx, ry

<b>说明:</b>	Rx 的高 16 位与 Ry 的低 16 位相减, 结果存入 Rz 的高 16 位; Rx 的低 16 位与 Ry 的高 16 位相加, 结果存入 Rz 的低 16 位。
<b>影响标志位:</b>	无影响
<b>异常:</b>	无

<b>32 位指令 令</b>	
<b>操作:</b>	$Rz[31:16] = Rx[31:16] - Ry[15:0]$ $Rz[15:0] = Rx[15:0] + Ry[31:16]$
<b>语法:</b>	psax.16 rz, rx, ry
<b>说明:</b>	Rx 的高 16 位与 Ry 的低 16 位相减, 结果存入 Rz 的高 16 位; Rx 的低 16 位与 Ry 的高 16 位相加, 结果存入 Rz 的低 16 位。
<b>影响标志位:</b>	无影响
<b>异常:</b>	无

**指令格式:**

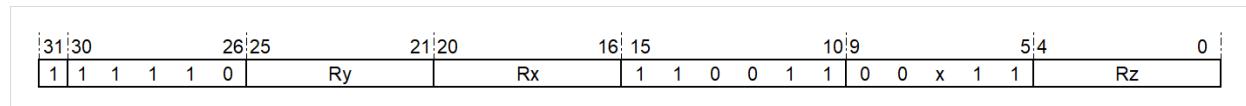


图 15.18: PSAX.16

### 15.19 PASX.(U/S)16.S——带饱和操作的 16 位并行 (无/有) 符号交叉加减指令

<b>统一化指令</b>	
<b>语法</b>	pasx.u16.s rz, rx, ry pasx.s16.s rz, rx, ry
<b>操作</b>	$Rz[31:16] = \text{saturate}(Rx[31:16] + Ry[15:0])$ $Rz[15:0] = \text{saturate}(Rx[15:0] - Ry[31:16])$
<b>编译结果</b>	仅存在 32 位指令。 pasx.u16.s rz, rx, ry pasx.s16.s rz, rx, ry

<b>说 明:</b>	Rx 的高 16 位与 Ry 的低 16 位相加, 结果做饱和处理后存入 Rz 的高 16 位; Rx 的低 16 位与 Ry 的高 16 位相减, 结果做饱和处理后存入 Rz 的低 16 位。 对于无符号数, 饱和处理的过程为, 若 16 位加/减法结果大于上饱和值 0xFFFF, 结果取 0xFFFF, 若 16 位加/减法结果小于下饱和值 0x0, 结果取 0x0, 其他情况取结果本身。 对于有符号数, 饱和处理的过程为, 若 16 位加/减法结果大于上饱和值 0x7FFF, 结果取 0x7FFF, 若 16 位加/减法结果小于下饱和值 0x8000, 结果取 0x8000, 其他情况取结果本身。
<b>影 响 标 志 位:</b>	无影响
<b>异 常:</b>	无

<b>32 位 指 令</b>	
<b>操 作:</b>	$Rz[31:16] = \text{saturate}(Rx[31:16] + Ry[15:0])$ $Rz[15:0] = \text{saturate}(Rx[15:0] - Ry[31:16])$
<b>语 法:</b>	pasx.u16.s rz, rx,ry pasx.s16.s rz, rx,ry
<b>说 明:</b>	Rx 的高 16 位与 Ry 的低 16 位相加, 结果做饱和处理后存入 Rz 的高 16 位; Rx 的低 16 位与 Ry 的高 16 位相减, 结果做饱和处理后存入 Rz 的低 16 位。 对于无符号数, 饱和处理的过程为, 若 16 位加/减法结果大于上饱和值 0xFFFF, 结果取 0xFFFF, 若 16 位加/减法结果小于下饱和值 0x0, 结果取 0x0, 其他情况取结果本身。 对于有符号数, 饱和处理的过程为, 若 16 位加/减法结果大于上饱和值 0x7FFF, 结果取 0x7FFF, 若 16 位加/减法结果小于下饱和值 0x8000, 结果取 0x8000, 其他情况取结果本身。
<b>影 响 标 志 位:</b>	无影响
<b>异 常:</b>	无

**指令格式:**

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 1 0 0 1 0	0 1 S 1 1	Rz	

图 15.19: PASX.(US)16.S

## 15.20 PSAX.(U/S)16.S——带饱和操作的 16 位并行（无/有）符号交叉减加指令

统一化指令	
语法	psax.u16.s rz, rx,ry psax.s16.s rz, rx,ry
操作	$Rz[31:16] = \text{saturate}(Rx[31:16] - Ry[15:0])$ $Rz[15:0] = \text{saturate}(Rx[15:0] + Ry[31:16])$
编译结果	仅存在 32 位指令。 psax.u16.s rz, rx,ry psax.s16.s rz, rx,ry

说明:	Rx 的高 16 位与 Ry 的低 16 位相减，结果做饱和处理后存入 Rz 的高 16 位； Rx 的低 16 位与 Ry 的高 16 位相加，结果做饱和处理后存入 Rz 的低 16 位。 对于无符号数，饱和处理的过程为，若 16 位加/减法结果大于上饱和值 0xFFFF，结果取 0xFFFF，若 16 位加/减法结果小于下饱和值 0x0，结果取 0x0，其他情况取结果本身。 对于有符号数，饱和处理的过程为，若 16 位加/减法结果大于上饱和值 0x7FFF，结果取 0x7FFF，若 16 位加/减法结果小于下饱和值 0x8000，结果取 0x8000，其他情况取结果本身。
影响标志位:	无影响
异常:	无

32 位 指 令	
操作:	Rz[31:16] = saturate(Rx[31:16] - Ry[15:0]) Rz[15:0] = saturate(Rx[15:0] + Ry[31:16])
语法:	psax.u16.s rz, rx, ry psax.s16.s rz, rx, ry
说明:	对于无符号数, 饱和处理的过程为, 若 16 位加/减法结果大于上饱和值 0xFFFF, 结果取 0xFFFF, 若 16 位加/减法结果小于下饱和值 0x0, 结果取 0x0, 其他情况取结果本身。 对于有符号数, 饱和处理的过程为, 若 16 位加/减法结果大于上饱和值 0x7FFF, 结果取 0x7FFF, 若 16 位加/减法结果小于下饱和值 0x8000, 结果取 0x8000, 其他情况取结果本身。
影响标志位:	无影响
异常:	无

指令格式:

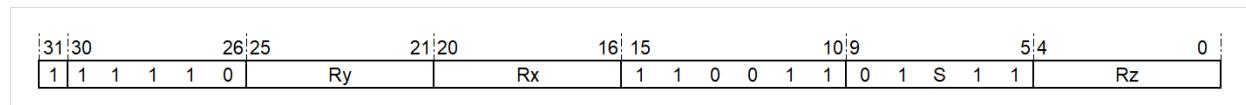


图 15.20: PSAX.(US)16.S

## 15.21 PASXH.(U/S)16——16 位并行（无/有）符号交叉加减求平均值指令

统一化指令	
语法	pasxh.u16 rz, rx, ry pasxh.s16 rz, rx, ry
操作	Rz[31:16] = (Rx[31:16] + Ry[15:0])/2 Rz[15:0] = (Rx[15:0] - Ry[31:16])/2
编译结果	仅存在 32 位指令。 pasxh.u16 rz, rx, ry pasxh.s16 rz, rx, ry

<b>说明:</b>	Rx 的高 16 位与 Ry 的低 16 位相加，结果除以 2 存入 Rz 的高 16 位； Rx 的低 16 位减 Ry 的高 16 位相减，结果除以 2 存入 Rz 的低 16 位。
<b>影响标志位:</b>	无影响
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	$Rz[31:16] = (Rx[31:16] + Ry[15:0])/2$ $Rz[15:0] = (Rx[15:0] - Ry[31:16])/2$
<b>语法:</b>	pasxh.u16 rz, rx,ry pasxh.s16 rz, rx,ry
<b>说明:</b>	Rx 的高 16 位与 Ry 的低 16 位相加，结果除以 2 存入 Rz 的高 16 位； Rx 的低 16 位减 Ry 的高 16 位相减，结果除以 2 存入 Rz 的低 16 位。
<b>影响标志位:</b>	无影响
<b>异常:</b>	无

**指令格式:**

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 1 1 1 0	Ry	Rx	1 1 0 0 1 0 1 0 S 1 1	Rz		

图 15.21: PASXH.(US)16

## 15.22 PSAXH.(U/S)16——16 位并行（无/有）符号交叉减加求平均值指令

<b>统一化指令</b>	
<b>语法</b>	psaxh.u16 rz, rx,ry psaxh.s16 rz, rx,ry
<b>操作</b>	$Rz[31:16] = (Rx[31:16] - Ry[15:0])/2$ $Rz[15:0] = (Rx[15:0] + Ry[31:16])/2$
<b>编译结果</b>	仅存在 32 位指令。 psaxh.u16 rz, rx,ry psaxh.s16 rz, rx,ry

<b>说明:</b>	Rx 的高 16 位与 Ry 的低 16 位相减, 结果除以 2 存入 Rz 的高 16 位; Rx 的低 16 位减 Ry 的高 16 位相加, 结果除以 2 存入 Rz 的低 16 位。
<b>影响标志位:</b>	无影响
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	$Rz[31:16] = (Rx[31:16] - Ry[15:0])/2$ $Rz[15:0] = (Rx[15:0] + Ry[31:16])/2$
<b>语法:</b>	psaxh.u16 rz, rx,ry psaxh.s16 rz, rx,ry
<b>说明:</b>	Rx 的高 16 位与 Ry 的低 16 位相减, 结果除以 2 存入 Rz 的高 16 位; Rx 的低 16 位减 Ry 的高 16 位相加, 结果除以 2 存入 Rz 的低 16 位。
<b>影响标志位:</b>	无影响
<b>异常:</b>	无

**指令格式:**

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 1 1 1 0	Ry	Rx	1 1 0 0 1 1 0 S 1 1	Rz		

图 15.22: PSAXH.(US)16

### 15.23 ADD.64——64 位加法指令

<b>统一化指令</b>	
<b>语法</b>	add.64 rz, rx,ry
<b>操作</b>	$\{Rz+1[31:0], Rz[31:0]\} = \{Rx+1[31:0], Rx[31:0]\} + \{Ry+1[31:0], Ry[31:0]\}$
<b>编译结果</b>	仅存在 32 位指令。 add.64 rz, rx,ry

<b>说 明:</b>	Rx+1 和 Rx 组成一个 64 位的数 (其中, Rx+1 为高 32 位, Rx 为低 32 位数据), Ry+1 和 Ry 组成一个 64 位的数 (其中, Ry+1 为高 32 位, Ry 为低 32 位数据), 执行 64 位加法, 加法结果的高 32 位存入 Rz+1, 低 32 位存入 Rz。
<b>影 响 标 志 位:</b>	无影响
<b>异 常:</b>	无

<b>32 位指 令</b>	
<b>操 作:</b>	$\{Rz+1[31:0], Rz[31:0]\} = \{Rx+1[31:0], Rx[31:0]\} + \{Ry+1[31:0], Ry[31:0]\}$
<b>语 法:</b>	add.64 rz, rx, ry
<b>说 明:</b>	Rx+1 和 Rx 组成一个 64 位的数 (其中, Rx+1 为高 32 位, Rx 为低 32 位数据), Ry+1 和 Ry 组成一个 64 位的数 (其中, Ry+1 为高 32 位, Ry 为低 32 位数据), 执行 64 位加法, 加法结果的高 32 位存入 Rz+1, 低 32 位存入 Rz。
<b>影 响 标 志 位:</b>	无影响
<b>异 常:</b>	无

**指令格式:**

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 1 0 0 0 0	0 0 x 1 1	Rz	

图 15.23: ADD.64

## 15.24 ADD.(U/S)64.S——带饱和操作的 64 位（无/有）符号加法指令

统一化指令	
语法	add.u64.s rz, rx,ry add.s64.s rz, rx,ry
操作	{Rz+1[31:0], Rz[31:0]} = Saturate({Rx+1[31:0], Rx[31:0]} + {Ry+1[31:0], Ry[31:0]})
编译结果	仅存在 32 位指令。 add.u64.s rz, rx,ry add.s64.s rz, rx,ry

32 位指令	
操作:	{Rz+1[31:0], Rz[31:0]} = Saturate({Rx+1[31:0], Rx[31:0]} + {Ry+1[31:0], Ry[31:0]})
语法:	add.u64.s rz, rx,ry add.s64.s rz, rx,ry
说明:	Rx+1 和 Rx 组成一个 64 位的数（其中，Rx+1 为高 32 位，Rx 为低 32 位数据），Ry+1 和 Ry 组成一个 64 位的数（其中，Ry+1 为高 32 位，Ry 为低 32 位数据），执行 64 位加法，加法结果做饱和处理后，高 32 位存入 Rz+1，低 32 位存入 Rz。 对于无符号数，饱和处理的过程为，若加法结果大于饱和值 0xFFFF FFFF FFFF FFFF，结果取 0xFFFF FFFF FFFF FFFF，其他情况取结果本身。 对于有符号数，饱和处理的过程为，若加法结果大于上饱和值 0x7FFF FFFF FFFF FFFF，结果取 0x7FFF FFFF FFFF FFFF，若加法结果小于下饱和值 0x8000 0000 0000 0000，结果取 0x8000 0000 0000 0000，其他情况取结果本身。
影响标志位:	无影响
异常:	无

指令格式：

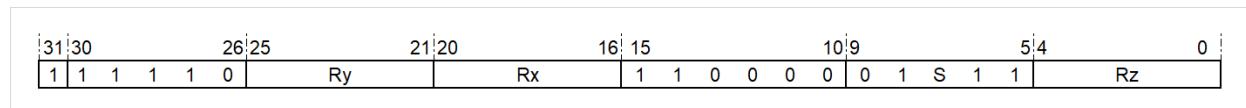


图 15.24: ADD.(US)64.S

## 15.25 SUB.64——64 位减法指令

统一化指令	
语法	sub.64 rz, rx, ry
操作	$\{Rz+1[31:0], Rz[31:0]\} = \{Rx+1[31:0], Rx[31:0]\} - \{Ry+1[31:0], Ry[31:0]\}$
编译结果	仅存在 32 位指令。 sub.64 rz, rx, ry

说 明:	Rx+1 和 Rx 组成一个 64 位的数 (其中, Rx+1 为高 32 位, Rx 为低 32 位数据), Ry+1 和 Ry 组成一个 64 位的数 (其中, Rx+1 为高 32 位, Rx 为低 32 位数据), 执行 64 位减法, 减法结果的高 32 位存入 Rz+1, 低 32 位存入 Rz。
影 响 标 志 位:	无影响
异 常:	无

32 位指 令	
操 作:	$\{Rz+1[31:0], Rz[31:0]\} = \{Rx+1[31:0], Rx[31:0]\} - \{Ry+1[31:0], Ry[31:0]\}$
语 法:	sub.64 rz, rx, ry
说 明:	Rx+1 和 Rx 组成一个 64 位的数 (其中, Rx+1 为高 32 位, Rx 为低 32 位数据), Ry+1 和 Ry 组成一个 64 位的数 (其中, Rx+1 为高 32 位, Rx 为低 32 位数据), 执行 64 位减法, 减法结果的高 32 位存入 Rz+1, 低 32 位存入 Rz。
影 响 标 志 位:	无影响
异 常:	无

指令格式：

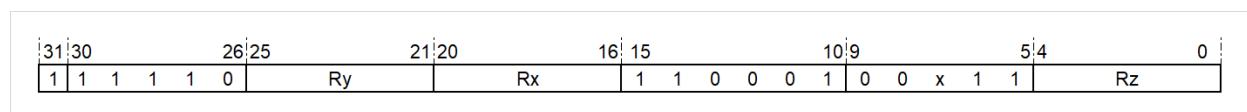


图 15.25: SUB.64

## 15.26 SUB.(U/S)64.S——带饱和操作的 64 位（无/有）符号减法指令

统一化指令	
语法	sub.u64.s rz, rx,ry sub.s64.s rz, rx,ry
操作	$\{Rz+1[31:0], Rz[31:0]\} = \text{Saturate}(\{Rx+1[31:0], Rx[31:0]\} - \{Ry+1[31:0], Ry[31:0]\})$
编译结果	仅存在 32 位指令。 sub.u64.s rz, rx,ry sub.s64.s rz, rx,ry

32 位指令	
操作:	$\{Rz+1[31:0], Rz[31:0]\} = \text{Saturate}(\{Rx+1[31:0], Rx[31:0]\} - \{Ry+1[31:0], Ry[31:0]\})$
语法:	sub.u64.s rz, rx,ry sub.s64.s rz, rx,ry
说明:	Rx+1 和 Rx 组成一个 64 位的数（其中，Rx+1 为高 32 位，Rx 为低 32 位数据），Ry+1 和 Ry 组成一个 64 位的数（其中，Ry+1 为高 32 位，Ry 为低 32 位数据），执行 64 位减法，减法结果做饱和处理后，高 32 位存入 Rz+1，低 32 位存入 Rz。 对于无符号数，饱和处理的过程为，若减法结果小于饱和值 0x0，结果取 0x0，其他情况取结果本身。 对于有符号数，饱和处理的过程为，若减法结果大于上饱和值 0x7FFF FFFF FFFF FFFF，结果取 0x7FFF FFFF FFFF FFFF，若减法结果小于下饱和值 0x8000 0000 0000 0000，结果取 0x8000 0000 0000 0000，其他情况取结果本身。
影响标志位:	无影响
异常:	无

指令格式：

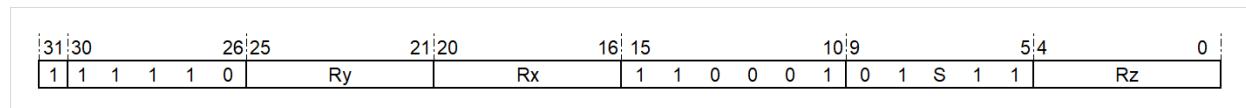


图 15.26: SUB.(US)64.S

## 15.27 ASRI.S32.R——带舍入操作的立即数算术右移指令

<b>统一化指令</b>	
<b>语法</b>	asri.s32.r rz, rx, oimm5
<b>操作</b>	Rz[31:0] <- Round(Rx[31:0] >>>oimm[4:0])
<b>编译结果</b>	仅存在 32 位指令。 asri.s32.r rz, rx, oimm5

<b>说 明:</b>	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），并做舍入操作。舍入操作过程为，若移出数据的最高位为 1，移位结果加 1，否则移位结果保持不变。右移位数由带偏置 1 的 5 位立即数（OIMM5）的值决定。
<b>影 响 标 志 位:</b>	无影响
<b>限 制:</b>	立即数的范围为 1-32。
<b>异 常:</b>	无

<b>32 位 指 令</b>	
<b>操 作:</b>	Rz[31:0] <- Round(Rx[31:0] >>>oimm[4:0])
<b>语 法:</b>	asri.s32.r rz, rx, oimm5
<b>说 明:</b>	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），并做舍入操作。舍入操作过程为，若移出数据的最高位为 1，移位结果加 1，否则移位结果保持不变。右移位数由带偏置 1 的 5 位立即数（OIMM5）的值决定。
<b>影 响 标 志 位:</b>	无影响
<b>限 制:</b>	立即数的范围为 1-32
<b>异 常:</b>	无

指令格式：

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Oimm5	Rx	1 1 0 1 0 0	0 1 1 0 1	Rz	

图 15.27: ASR.S32.R

## 15.28 ASR.S32.R——带舍入操作的算术右移指令

统一化指令	
语法	asr.s32.r rz, rx, ry
操作	$Rz[31:0] <- \text{Round}(Rx[31:0] >>> Ry[5:0])$
编译结果	仅存在 32 位指令。 asr.s32.r 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），并做舍入操作。舍入操作过程为，若移出数据的最高位为 1，移位结果加 1，否则移位结果保持不变。右移位数由 Ry[5:0] 决定。
影响标志位:	无影响
异常:	无

32 位指令	
操作:	$Rz[31:0] <- \text{Round}(Rx[31:0] >>> Ry[5:0])$
语法:	asr.s32.r 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），并做舍入操作。舍入操作过程为，若移出数据的最高位为 1，移位结果加 1，否则移位结果保持不变。右移位数由 Ry[5:0] 决定。
影响标志位:	无影响
异常:	无

指令格式:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 1 0 1 0 0	0 1 1 1 1	Rz	

图 15.28: ASR.S32.R

### 15.29 LSRI.U32.R——带舍入操作的立即数逻辑右移指令

统一化指令	
语法	lsri.u32.r rz, rx, oimm5
操作	Rz[31:0] <- Round(Rx[31:0] >> oimm[4:0])
编译结果	仅存在 32 位指令。 lsri.u32.r rz, rx, oimm5

说明:	将 Rx 的值进行逻辑右移（原值右移，左侧补 0），并做舍入操作。舍入操作过程为，若移出数据的最高位为 1，移位结果加 1，否则移位结果保持不变。右移位数由带偏置 1 的 5 位立即数 (OIMM5) 的值决定。
影响标志位:	无影响
限制:	立即数的范围为 1-32。
异常:	无

32 位指令	
操作:	Rz[31:0] <- Round(Rx[31:0] >>oimm[4:0])
语法:	lsri.u32.r rz, rx, oimm5
说明:	将 Rx 的值进行逻辑右移（原值右移，左侧补 0），并做舍入操作。舍入操作过程为，若移出数据的最高位为 1，移位结果加 1，否则移位结果保持不变。右移位数由带偏置 1 的 5 位立即数 (OIMM5) 的值决定。
影响标志位:	无影响
限制:	立即数的范围为 1-32
异常:	无

指令格式:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Oimm5	Rx	1 1 0 1 0 0	1 1 0 0 1	Rz	

图 15.29: LSRI.U32.R

### 15.30 LSR.U32.R——带舍入操作的逻辑右移指令

统一化指令	
语法	lsr.u32.r rz, rx, ry
操作	$Rz[31:0] \leftarrow \text{Round}(Rx[31:0] >> Ry[5:0])$
编译结果	仅存在 32 位指令。 lsr.u32.r rz, rx, ry

说明:	将 Rx 的值进行逻辑右移（原值右移，左侧补 0），并做舍入操作。舍入操作过程为，若移出数据的最高位为 1，移位结果加 1，否则移位结果保持不变。右移位数由 Ry[5:0] 决定。
影响标志位:	无影响
异常:	无

32 位指令	
操作:	$Rz[31:0] \leftarrow \text{Round}(Rx[31:0] >> Ry[5:0])$
语法:	lsr.u32.r rz, rx, ry
说明:	将 Rx 的值进行逻辑右移（原值右移，左侧补 0），并做舍入操作。舍入操作过程为，若移出数据的最高位为 1，移位结果加 1，否则移位结果保持不变。右移位数由 Ry[5:0] 决定。
影响标志位:	无影响
异常:	无

指令格式:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 1 0 1 0 0	1 1 0 1 1	Rz	

图 15.30: LSR.U32.R

## 15.31 LSLI.(U/S)32.S——带饱和操作的立即数（无/有）符号逻辑左移指令

统一化指令	
语法	lsli.u32.s rz, rx, oimm5 lsli.s32.s rz, rx, oimm5
操作	Rz[31:0] <- Saturate(Rx[31:0] << oimm[4:0])
编译结果	仅存在 32 位指令。 lsli.u32.s rz, rx, oimm5 lsli.s32.s rz, rx, oimm5

说明:	将 Rx 的值进行逻辑左移（原值左移，右侧补 0），并做饱和操作。左移位数由带偏置 1 的 5 位立即数（OIMM5）的值决定。 对于无符号数，饱和处理的过程为，若移位结果大于饱和值 0xFFFF FFFF，结果取 0xFFFF FFFF，其他情况取结果本身。 对于有符号数，饱和处理的过程为，若移位结果大于上饱和值 0x7FFF FFFF，结果取 0x7FFF FFFF，若移位结果小于下饱和值 0x8000 0000，结果取 0x8000 0000，其他情况取结果本身。
影响标志位:	无影响
限制:	立即数的范围为 1-32。
异常:	无

<b>32 位 指 令</b>	
<b>操 作:</b>	Rz[31:0] <- Saturate(Rx[31:0] << oimm[4:0])
<b>语 法:</b>	lsli.u32.s rz, rx, oimm5 lsli.s32.s rz, rx, oimm5
<b>说 明:</b>	将 Rx 的值进行逻辑左移（原值左移，右侧补 0），并做饱和操作。左移位数由带偏置 1 的 5 位立即数（OIMM5）的值决定。 对于无符号数，饱和处理的过程为，若移位结果大于饱和值 0xFFFF FFFF，结果取 0xFFFF FFFF，其他情况取结果本身。 对于有符号数，饱和处理的过程为，若移位结果大于上饱和值 0x7FFF FFFF，结果取 0x7FFF FFFF，若移位结果小于下饱和值 0x8000 0000，结果取 0x8000 0000，其他情况取结果本身。
<b>影 响 标 志 位:</b>	无影响
<b>限 制:</b>	立即数的范围为 1-32
<b>异 常:</b>	无

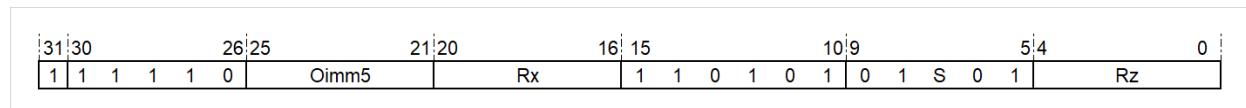
**指令格式:**

图 15.31: LSLI.(US)32.S

### 15.32 LSL.(U/S)32.S——带饱和操作的（无/有）符号逻辑左移指令

<b>统一化指令</b>	
<b>语法</b>	lsl.u32.s rz, rx, ry lsl.s32.s rz, rx, ry
<b>操作</b>	Rz[31:0] <- Saturate(Rx[31:0] << Ry[5:0])
<b>编译结果</b>	仅存在 32 位指令。 lsl.u32.s rz, rx, ry lsl.s32.s rz, rx, ry

<b>说 明:</b>	将 Rx 的值进行逻辑左移（原值左移，右侧补 0），并做饱和操作。左移位数由 Ry[5:0] 的值决定。 对于无符号数，饱和处理的过程为，若移位结果大于饱和值 0xFFFF FFFF，结果取 0xFFFF FFFF，其他情况取结果本身。 对于有符号数，饱和处理的过程为，若移位结果大于上饱和值 0x7FFF FFFF，结果取 0x7FFF FFFF，若移位结果小于下饱和值 0x8000 0000，结果取 0x8000 0000，其他情况取结果本身。
<b>影 响 标 志 位:</b>	无影响
<b>异 常:</b>	无

<b>32位指令</b>	
<b>操作:</b>	Rz[31:0] <- Saturate(Rx[31:0] << Ry[5:0])
<b>语法:</b>	lsl.u32.s rz, rx, ry lsl.s32.s rz, rx, ry
<b>说明:</b>	将 Rx 的值进行逻辑左移（原值左移，右侧补 0），并做饱和操作。左移位数由 Ry[5:0] 的值决定。 对于无符号数，饱和处理的过程为，若移位结果大于饱和值 0xFFFF FFFF，结果取 0xFFFF FFFF，其他情况取结果本身。 对于有符号数，饱和处理的过程为，若移位结果大于上饱和值 0x7FFF FFFF，结果取 0x7FFF FFFF，若移位结果小于下饱和值 0x8000 0000，结果取 0x8000 0000，其他情况取结果本身。
<b>影响标志位:</b>	无影响
<b>异常:</b>	无

**指令格式:**

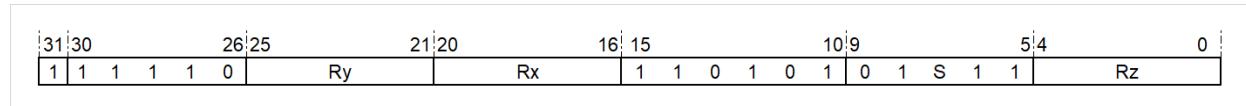


图 15.32: LSL.(US)32.S

### 15.33 PASRI.S16——16 位并行立即数算术右移指令

<b>统一化指令</b>	
<b>语法</b>	pasri.s16 rz, rx, oimm4
<b>操作</b>	Rz[31:16] <- Rx[31:16] >>> oimm[3:0] Rz[15:0] <- Rx[15:0] >>> oimm[3:0]
<b>编译结果</b>	仅存在 32 位指令。 pasri.s16 rz, rx, oimm4

<b>说明:</b>	Rx 的高低半字分别进行算术右移（原值右移，左侧移入原符号位的拷贝），移位结果分别存入 Rz 的高低半字。右移位数由带偏置 1 的 4 位立即数（OIMM4）的值决定。
<b>影响标志位:</b>	无影响
<b>限制:</b>	立即数的范围为 1-16
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	Rz[31:16] <- Rx[31:16] >>> oimm[3:0] Rz[15:0] <- Rx[15:0] >>> oimm[3:0]
<b>语法:</b>	pasri.s16 rz, rx, oimm4
<b>说明:</b>	Rx 的高低半字分别进行算术右移（原值右移，左侧移入原符号位的拷贝），移位结果分别存入 Rz 的高低半字。右移位数由带偏置 1 的 4 位立即数（OIMM4）的值决定。
<b>影响标志位:</b>	无影响
<b>限制:</b>	立即数的范围为 1-16
<b>异常:</b>	无

**指令格式:**

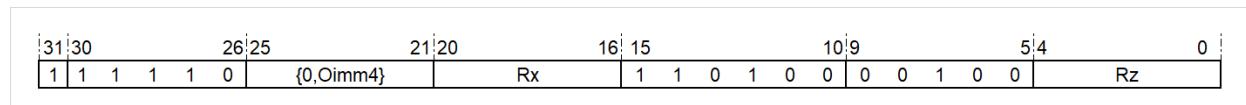


图 15.33: PASRI.S16

### 15.34 PASR.S16——16 位并行算术右移指令

<b>统一化指令</b>	
<b>语法</b>	pasr.s16 rz, rx, ry
<b>操作</b>	Rz[31:16] <- Rx[31:16] >>> Ry[4:0] Rz[15:0] <- Rx[15:0] >>> Ry[4:0]
<b>编译结果</b>	仅存在 32 位指令。 pasr.s16 rz, rx, ry

<b>说明:</b>	Rx 的高低半字分别进行算术右移（原值右移，左侧移入原符号位的拷贝），移位结果分别存入 Rz 的高低半字。右移位数由 Ry 低 5 位 (Ry[4:0]) 的值确定。
<b>影响标志:</b>	无影响
<b>志位:</b>	无
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	Rz[31:16] <- Rx[31:16] >>> Ry[4:0] Rz[15:0] <- Rx[15:0] >>> Ry[4:0]
<b>语法:</b>	pasr.s16 rz, rx, ry
<b>说明:</b>	Rx 的高低半字分别进行算术右移（原值右移，左侧移入原符号位的拷贝），移位结果分别存入 Rz 的高低半字。右移位数由 Ry 低 5 位 (Ry[4:0]) 的值确定。
<b>影响标志:</b>	无影响
<b>志位:</b>	无
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 1 1 1 0	Ry	Rx	1 1 0 1 0 0 0 0 1 1 0	Rz		

图 15.34: PASR.S16

### 15.35 PASRI.S16.R——带舍入操作的 16 位并行立即数算术右移指令

<b>统一化指令</b>	
<b>语法</b>	pasri.s16.r rz, rx, oimm4
<b>操作</b>	Rz[31:16] <- Round(Rx[31:16] >>> oimm[3:0]) Rz[15:0] <- Round(Rx[15:0] >>> oimm[3:0])
<b>编译结果</b>	仅存在 32 位指令。 pasri.s16.r rz, rx, oimm4

说 明:	Rx 的高低半字分别进行算术右移（原值右移，左侧移入原符号位的拷贝），移位结果做舍入操作后分别存入 Rz 的高低半字。舍入操作过程为，若移出数据的最高位为 1，移位结果加 1，否则移位结果保持不变。右移位数由带偏置 1 的 4 位立即数（OIMM4）的值决定。
影 响 标 志 位:	无影响
限 制:	立即数的范围为 1-16
异 常:	无

32 位 指 令	
操 作:	Rz[31:16] <- Round(Rx[31:16] >>> oimm[3:0]) Rz[15:0] <- Round(Rx[15:0] >>> oimm[3:0])
语 法:	pasri.s16.r rz, rx, oimm4
说 明:	Rx 的高低半字分别进行算术右移（原值右移，左侧移入原符号位的拷贝），移位结果做舍入操作后分别存入 Rz 的高低半字。舍入操作过程为，若移出数据的最高位为 1，移位结果加 1，否则移位结果保持不变。右移位数由带偏置 1 的 4 位立即数（OIMM4）的值决定。
影 响 标 志 位:	无影响
限 制:	立即数的范围为 1-16
异 常:	无

#### 指令格式:

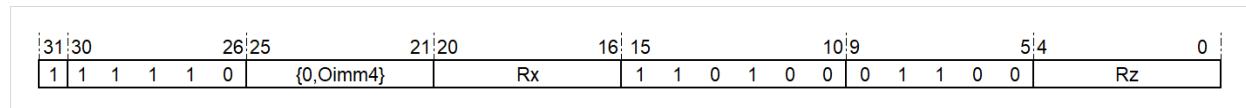


图 15.35: PASRI.S16.R

### 15.36 PASR.S16.R——带舍入操作的 16 位并行算术右移指令

<b>统一化指令</b>	
<b>语法</b>	pasr.s16.r Rz, Rx, Ry
<b>操作</b>	Rz[31:16] <- Round(Rx[31:16] >>> Ry[4:0]) Rz[15:0] <- Round(Rx[15:0] >>> Ry[4:0])
<b>编译结果</b>	仅存在 32 位指令。 pasr.s16.r rz, rx, ry

<b>说 明:</b>	Rx 的高低半字分别进行算术右移（原值右移，左侧移入原符号位的拷贝），移位结果做舍入操作后分别存入 Rz 的高低半字。舍入操作过程为，若移出数据的最高位为 1，移位结果加 1，否则移位结果保持不变。右移位数由 Ry 低 5 位 (Ry[4:0]) 的值确定。
<b>影响 标志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

<b>32 位指 令</b>	
<b>操 作:</b>	Rz[31:16] <- Round(Rx[31:16] >>> Ry[4:0]) Rz[15:0] <- Round(Rx[15:0] >>> Ry[4:0])
<b>语 法:</b>	pasr.s16.r rz, rx, ry
<b>说 明:</b>	Rx 的高低半字分别进行算术右移（原值右移，左侧移入原符号位的拷贝），移位结果做舍入操作后分别存入 Rz 的高低半字。舍入操作过程为，若移出数据的最高位为 1，移位结果加 1，否则移位结果保持不变。右移位数由 Ry 低 5 位 (Ry[4:0]) 的值确定。
<b>影响 标志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

**指令格式:**

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 1 0 1 0 0	0 1 1 1 0	Rz	

图 15.36: PASR.S16.R

### 15.37 PLSRI.U16——16 位并行立即数逻辑右移指令

统一化指令	
语法	plsri.u16 rz, rx, oimm4
操作	Rz[31:16] <- Rx[31:16] >> oimm[3:0] Rz[15:0] <- Rx[15:0] >> oimm[3:0]
编译结果	仅存在 32 位指令。 plsri.u16 rz, rx, oimm4

说明:	Rx 的高低半字分别进行逻辑右移（原值右移，左侧补 0），移位结果分别存入 Rz 的高低半字。右移位数由带偏置 1 的 4 位立即数（OIMM4）的值决定。
影响标志位:	无影响
限制:	立即数的范围为 1-16。
异常:	无

32 位指令	
操作:	Rz[31:16] <- Rx[31:16] >> oimm[3:0] Rz[15:0] <- Rx[15:0] >> oimm[3:0]
语法:	plsri.u16 rz, rx, oimm4
说明:	Rx 的高低半字分别进行逻辑右移（原值右移，左侧补 0），移位结果分别存入 Rz 的高低半字。右移位数由带偏置 1 的 4 位立即数（OIMM4）的值决定。
影响标志位:	无影响
限制:	立即数的范围为 1-16
异常:	无

指令格式:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	{0,0imm4}	Rx	1 1 0 1 0 0	1 0 0 0 0	Rz	

图 15.37: PLSRI.U16

### 15.38 PLSR.U16——16 位并行逻辑右移指令

统一化指令	
语法	plsr.u16 rz, rx, ry
操作	Rz[31:16] <- Rx[31:16] >> Ry[4:0] Rz[15:0] <- Rx[15:0] >> Ry[4:0]
编译结果	仅存在 32 位指令。 plsr.u16 rz, rx, ry

说明:	Rx 的高低半字分别进行逻辑右移 (原值右移, 左侧补 0), 移位结果分别存入 Rz 的高低半字。右移位数由 RY 低 5 位 (RY[4:0]) 的值确定。
影响标志位:	无影响
限制:	无
异常:	无

32 位指令	
操作:	Rz[31:16] <- Round(Rx[31:16] >> Ry[4:0]) Rz[15:0] <- Round(Rx[15:0] >> Ry[4:0])
语法:	plsr.u16.r rz, rx, ry
说明:	Rx 的高低半字分别进行逻辑右移 (原值右移, 左侧补 0), 移位结果做舍入操作后分别存入 Rz 的高低半字。舍入操作过程为, 若移出数据的最高位为 1, 移位结果加 1, 否则移位结果保持不变。右移位数由 RY 低 5 位 (RY[4:0]) 的值确定。
影响标志位:	无影响
限制:	无
异常:	无

**指令格式：**

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 1 0 1 0 0	1 0 0 1 0	Rz	

图 15.38: PLSR.U16

### 15.39 PLSRI.U16.R——带舍入操作的 16 位并行立即数逻辑右移指令

<b>统一化指令</b>	
<b>语法</b>	plsr.u16.r rz, rx, oimm4
<b>操作</b>	$Rz[31:16] \leftarrow Round(Rx[31:16] >> oimm[3:0])$ $Rz[15:0] \leftarrow Round(Rx[15:0] >> oimm[3:0])$
<b>编译结果</b>	仅存在 32 位指令。 plsr.u16.r rz, rx, oimm4

<b>说 明:</b>	Rx 的高低半字分别进行逻辑右移（原值右移，左侧补 0），移位结果做舍入操作后分别存入 Rz 的高低半字。舍入操作过程为，若移出数据的最高位为 1，移位结果加 1，否则移位结果保持不变。右移位数由带偏置 1 的 4 位立即数（OIMM4）的值决定。
<b>影 响</b>	无影响
<b>标 志 位:</b>	
<b>限 制:</b>	立即数的范围为 1-16。
<b>异 常:</b>	无

<b>32 位指令</b>	
<b>操作</b>	Rz[31:16] <- Round(Rx[31:16] >> oimm[3:0]) Rz[15:0] <- Round(Rx[15:0] >> oimm[3:0])
<b>语法</b>	plsri.u16.r rz, rx, oimm4
<b>说明:</b>	Rx 的高低半字分别进行逻辑右移 (原值右移, 左侧补 0), 移位结果做舍入操作后分别存入 Rz 的高低半字。舍入操作过程为, 若移出数据的最高位为 1, 移位结果加 1, 否则移位结果保持不变。右移位数由带偏置 1 的 4 位立即数 (OIMM4) 的值决定。
<b>影响标志位:</b>	无影响
<b>限制:</b>	立即数的范围为 1-16
<b>异常:</b>	无

**指令格式:**

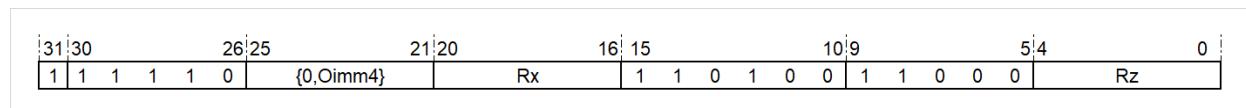


图 15.39: PLSRI.U16.R

## 15.40 PLSR.U16.R——带舍入操作的 16 位并行逻辑右移指令

<b>统一化指令</b>	
<b>语法</b>	plsr.u16.r rz, rx, ry
<b>操作</b>	Rz[31:16] <- Round(Rx[31:16] >> Ry[4:0]) Rz[15:0] <- Round(Rx[15:0] >> Ry[4:0])
<b>编译结果</b>	仅存在 32 位指令。 plsr.u16.r rz, rx, ry

<b>说 明:</b>	Rx 的高低半字分别进行逻辑右移 (原值右移, 左侧补 0), 移位结果做舍入操作后分别存入 Rz 的高低半字。舍入操作过程为, 若移出数据的最高位为 1, 移位结果加 1, 否则移位结果保持不变。右移位数由 Ry 低 5 位 (Ry[4:0]) 的值确定。
<b>影 响 标 志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

<b>32 位指 令</b>	
<b>操 作:</b>	Rz[31:16] <- Round(Rx[31:16] >> Ry[4:0]) Rz[15:0] <- Round(Rx[15:0] >> Ry[4:0])
<b>语 法:</b>	plsr.u16.r rz, rx, ry
<b>说 明:</b>	Rx 的高低半字分别进行逻辑右移 (原值右移, 左侧补 0), 移位结果做舍入操作后分别存入 Rz 的高低半字。舍入操作过程为, 若移出数据的最高位为 1, 移位结果加 1, 否则移位结果保持不变。右移位数由 Ry 低 5 位 (Ry[4:0]) 的值确定。
<b>影 响 标 志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

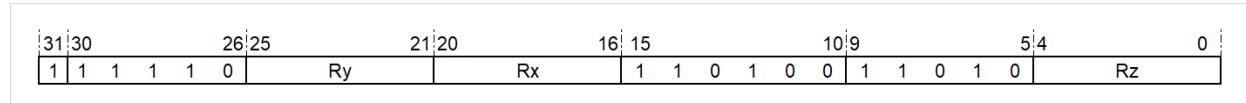
**指令格式:**

图 15.40: PLSR.U16.R

### 15.41 PLSLI.16——16 位并行立即数逻辑左移指令

<b>统一化指令</b>	
<b>语法</b>	plsl.i.16 rz, rx, oimm4
<b>操作</b>	Rz[31:16] <- Rx[31:16] << oimm[3:0] Rz[15:0] <- Rx[15:0] << oimm[3:0]
<b>编译结果</b>	仅存在 32 位指令。 plsl.i.16 rz, rx, oimm4

<b>说明:</b>	Rx 的高低半字分别进行逻辑左移（原值左移，右侧补 0），移位结果分别存入 Rz 的高低半字。右移位数由带偏置 1 的 4 位立即数（OIMM4）的值决定。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	Rz[31:16] <- Rx[31:16] << oimm[3:0] Rz[15:0] <- Rx[15:0] << oimm[3:0]
<b>语法:</b>	plsl.i.16 rz, rx, oimm4
<b>说明:</b>	Rx 的高低半字分别进行逻辑左移（原值左移，右侧补 0），移位结果分别存入 Rz 的高低半字。右移位数由带偏置 1 的 4 位立即数（OIMM4）的值决定。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

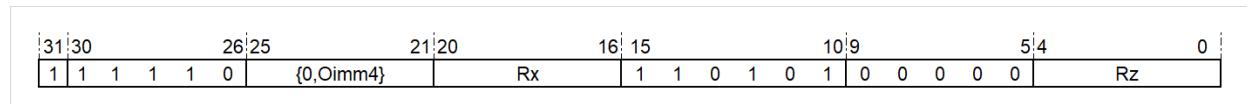


图 15.41: PLSLI.16

## 15.42 PSL.16——16 位并行逻辑左移指令

<b>统一化指令</b>	
<b>语法</b>	plsl.16 rz, rx, ry
<b>操作</b>	Rz[31:16] <- Rx[31:16] << Ry[4:0] Rz[15:0] <- Rx[15:0] << Ry[4:0]
<b>编译结果</b>	仅存在 32 位指令。 plsl.16 rz, rx, ry

<b>说明:</b>	Rx 的高低半字分别进行逻辑左移 (原值左移, 右侧补 0), 移位结果分别存入 Rz 的高低半字。右移位数由 Ry 低 5 位 (Ry[4:0]) 的值确定。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	Rz[31:16] <- Rx[31:16] << Ry[4:0] Rz[15:0] <- Rx[15:0] << Ry[4:0]
<b>语法:</b>	plsl.16 rz, rx, ry
<b>说明:</b>	Rx 的高低半字分别进行逻辑左移 (原值左移, 右侧补 0), 移位结果分别存入 Rz 的高低半字。右移位数由 Ry 低 5 位 (Ry[4:0]) 的值确定。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

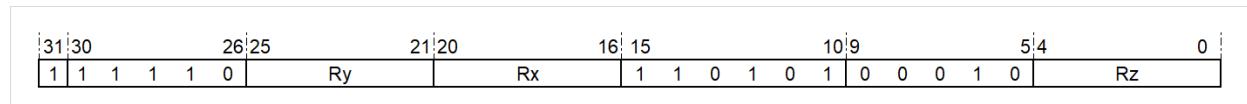


图 15.42: PSL.16

## 15.43 PLSLI.(U/S)16.S——带饱和操作的 16 位并行立即数（无/有）符号逻辑左移指令

<b>统一化指令</b>	
<b>语法</b>	plslsli.s16.s rz, rx, oimm4 plslsli.u16.s rz, rx, oimm4
<b>操作</b>	Rz[31:16] <- Saturate(Rx[31:16] << oimm[3:0]) Rz[15:0] <- Saturate(Rx[15:0] << oimm[3:0])
<b>编译结果</b>	仅存在 32 位指令。 plslsli.s16.s rz, rx, oimm4 plslsli.u16.s rz, rx, oimm4

<b>说明:</b>	将 Rx 的高低半字分别进行逻辑左移（原值左移，右侧补 0），并做饱和操作。左移位数由带偏置 1 的 5 位立即数（OIMM5）的值决定。 对于无符号数，饱和处理的过程为，若移位结果大于饱和值 0xFFFF，结果取 0xFFFF，其他情况取结果本身。 对于有符号数，饱和处理的过程为，若移位结果大于上饱和值 0x7FFF，结果取 0x7FFF，若移位结果小于下饱和值 0x8000，结果取 0x8000，其他情况取结果本身。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位 指 令</b>	
<b>操 作:</b>	Rz[31:16] <- Saturate(Rx[31:16] << oimm[3:0]) Rz[15:0] <- Saturate(Rx[15:0] << oimm[3:0])
<b>语 法:</b>	plsl.s16.s rz, rx, oimm4 plsl.u16.s rz, rx, oimm4
<b>说 明:</b>	将 Rx 的高低半字分别进行逻辑左移（原值左移，右侧补 0），并做饱和操作。左移位数由带偏置 1 的 5 位立即数（OIMM5）的值决定。 对于无符号数，饱和处理的过程为，若移位结果大于饱和值 0xFFFF，结果取 0xFFFF，其他情况取结果本身。 对于有符号数，饱和处理的过程为，若移位结果大于上饱和值 0x7FFF，结果取 0x7FFF，若移位结果小于下饱和值 0x8000，结果取 0x8000，其他情况取结果本身。
<b>影 响 标 志 位:</b>	无影响
<b>限 制:</b>	立即数的范围为 1-16
<b>异 常:</b>	无

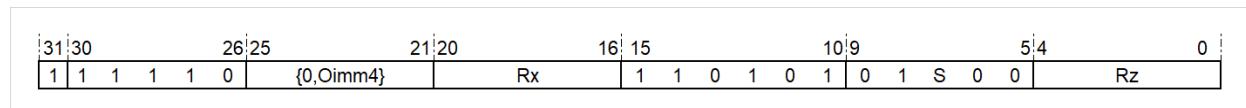
**指令格式:**

图 15.43: PLSLI.(US)16.S

## 15.44 PSL.(U/S)16.S——带饱和操作的 16 位并行（无/有）符号逻辑左移指令

统一化指令	
语法	plsl.s16.s rz, rx, ry plsl.u16.s rz, rx, ry
操作	Rz[31:16] <- Saturate(Rx[31:16] << Ry[4:0]) Rz[15:0] <- Saturate(Rx[15:0] << Ry[4:0])
编译结果	仅存在 32 位指令。 plsl.s16.s rz, rx, ry plsl.u16.s rz, rx, ry

说明:	将 Rx 的高低半字分别进行逻辑左移（原值左移，右侧补 0），并做饱和操作。左移位数由 Ry 低 5 位 (Ry[4:0]) 的值确定。 对于无符号数，饱和处理的过程为，若移位结果大于饱和值 0xFFFF，结果取 0xFFFF，其他情况取结果本身。 对于有符号数，饱和处理的过程为，若移位结果大于上饱和值 0x7FFF，结果取 0x7FFF，若移位结果小于下饱和值 0x8000，结果取 0x8000，其他情况取结果本身。
影响标志位:	无影响
限制:	无
异常:	无

<b>32 位 指 令</b>	
<b>操 作:</b>	Rz[31:16] <- Saturate(Rx[31:16] << Ry[4:0]) Rz[15:0] <- Saturate(Rx[15:0] << Ry[4:0])
<b>语 法:</b>	plsl.s16.s rz, rx, ry plsl.u16.s rz, rx, ry
<b>说 明:</b>	将 Rx 的高低半字分别进行逻辑左移（原值左移，右侧补 0），并做饱和操作。左移位数由 Ry 低 5 位 (Ry[4:0]) 的值确定。 对于无符号数，饱和处理的过程为，若移位结果大于饱和值 0xFFFF，结果取 0xFFFF，其他情况取结果本身。 对于有符号数，饱和处理的过程为，若移位结果大于上饱和值 0x7FFF，结果取 0x7FFF，若移位结果小于下饱和值 0x8000，结果取 0x8000，其他情况取结果本身。
<b>影 响 标 志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

**指令格式:**

31 30	26 25	21 20	16 15	10 9	5 4	0
1   1    1    1    1    0   Ry   Rx   1    1    0    1    0   0    1    S    1    0   Rz						

图 15.44: PSL.(US)16.S

### 15.45 SEL——位选择指令

<b>统一化指令</b>	
<b>语法</b>	sel rz, rx, ry, rs
<b>操作</b>	for(i=0;i=31;i++) Rz[i] = Rs[i] ? Rx[i] : Ry[i]
<b>编译结果</b>	仅存在 32 位指令。 sel rz, rx, ry, rs

<b>说明:</b>	根据位选择寄存器 Rs 每一位的值, 选择 Rx、Ry 中相对应的位存入 Rz。若 Rs[i]==1 选择 Rx[i] 存入 Rz[i], 否则选择 Ry[i] 存入 Rz[i]。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	for(i=0;i=31;i++) Rz[i] = Rs[i] ? Rx[i] : Ry[i]
<b>语法:</b>	sel rz, rx, ry, rs
<b>说明:</b>	根据位选择寄存器 Rs 每一位的值, 选择 Rx、Ry 中相对应的位存入 Rz。若 Rs[i]==1 选择 Rx[i] 存入 Rz[i], 否则选择 Ry[i] 存入 Rz[i]。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 1 1 1 0	Ry	Rx	1 0 0 1 0 0	Rs	Rz	

图 15.45: SEL

## 15.46 PCMPNE.8——8 位并行比较不等指令

<b>统一化指令</b>	
<b>语法</b>	pcmpne.8 rz, rx, ry
<b>操作</b>	Rz[31:24] = (Rx[31:24] != Ry[31:24]) ? 8' hFF:8' h00 Rz[23:16] = (Rx[23:16] != Ry[23:16]) ? 8' hFF:8' h00 Rz[15:8] = (Rx[15:8] != Ry[15:8]) ? 8' hFF:8' h00 Rz[7:0] = (Rx[7:0] != Ry[7:0]) ? 8' hFF:8' h00
<b>编译结果</b>	仅存在 32 位指令。 pcmpne.8 rz, rx, ry

<b>说明:</b>	以字节为单位，分别判定 Rx 的 4 个字节与 Ry 的 4 个字节是否相等。如果相等，Rz 对应字节的结果为 0xFF，否则 Rz 对应字节的结果为 0x0。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	Rz[31:24] = (Rx[31:24] != Ry[31:24]) ? 8' hFF:8' h00 Rz[23:16] = (Rx[23:16] != Ry[23:16]) ? 8' hFF:8' h00 Rz[15:8] = (Rx[15:8] != Ry[15:8]) ? 8' hFF:8' h00 Rz[7:0] = (Rx[7:0] != Ry[7:0]) ? 8' hFF:8' h00
<b>语法:</b>	pcmpne.8 rz, rx, ry
<b>说明:</b>	以字节为单位，分别判定 Rx 的 4 个字节与 Ry 的 4 个字节是否相等。如果相等，Rz 对应字节的结果为 0xFF，否则 Rz 对应字节的结果为 0x0。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 1 1 1 0	Ry	Rx	1 1 0 0 1 0 0 0 x 1 0	Rz		

图 15.46: PCMPNE.8

### 15.47 PCMPNE.16——16 位并行比较不等指令

<b>统一化指令</b>	
<b>语法</b>	pcmpne.16 rz, rx, ry
<b>操作</b>	Rz[31:16] = (Rx[31:16] != Ry[31:16]) ? 16' hFFFF:16' b0000 Rz[15:0] = (Rx[15:0] != Ry[15:0]) ? 16' hFFFF:16' b0000
<b>编译结果</b>	仅存在 32 位指令。 pcmpne.16 rz, rx, ry

<b>说明:</b>	分别判定 Rx 的高低半字与 Ry 的高低半字是否相等。如果相等，Rz 对应半字的结果为 0xFFFF，否则 Rz 对应字节的结果为 0x0。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	$Rz[31:16] = (Rx[31:16] \neq Ry[31:16]) ? 16' hFFFF:16' b0000$ $Rz[15:0] = (Rx[15:0] \neq Ry[15:0]) ? 16' hFFFF:16' b0000$
<b>语法:</b>	pcmpne.16 rz, rx, ry
<b>说明:</b>	分别判定 Rx 的高低半字与 Ry 的高低半字是否相等。如果相等，Rz 对应半字的结果为 0xFFFF，否则 Rz 对应字节的结果为 0x0。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 1 1 1 0	Ry	Rx	1 1 0 0 1 0 0 0 x 0 0		Rz	

图 15.47: PCMPNE.16

## 15.48 PCMPHS.(U/S)8——8 位并行（无/有）符号大于等于比较指令

统一化指令	
语法	pcmphs.u8 rz, rx, ry pcmphs.s8 rz, rx, ry
操作	Rz[31:24] = (Rx[31:24] >= Ry[31:24]) ? 8' hFF:8' h00 Rz[23:16] = (Rx[23:16] >= Ry[23:16]) ? 8' hFF:8' h00 Rz[15:8] = (Rx[15:8] >= Ry[15:8]) ? 8' hFF:8' h00 Rz[7:0] = (Rx[7:0] >= Ry[7:0]) ? 8' hFF:8' h00
编译结果	仅存在 32 位指令。 pcmphs.u8 rz, rx, ry pcmphs.s8 rz, rx, ry

说明:	以字节为单位，分别判定 Rx 的 4 个字节是否大于等于 Ry 的 4 个字节。如果 Rx 的字节大于等于 Ry 的字节，Rz 对应字节的结果为 0xFF，否则 Rz 对应字节的结果为 0x0。
影响标志位:	无影响
限制:	无
异常:	无

32 位指令	
操作:	Rz[31:24] = (Rx[31:24] >= Ry[31:24]) ? 8' hFF:8' h00 Rz[23:16] = (Rx[23:16] >= Ry[23:16]) ? 8' hFF:8' h00 Rz[15:8] = (Rx[15:8] >= Ry[15:8]) ? 8' hFF:8' h00 Rz[7:0] = (Rx[7:0] >= Ry[7:0]) ? 8' hFF:8' h00
语法:	pcmphs.u8 rz, rx, ry pcmphs.s8 rz, rx, ry
说明:	以字节为单位，分别判定 Rx 的 4 个字节是否大于等于 Ry 的 4 个字节。如果 Rx 的字节大于等于 Ry 的字节，Rz 对应字节的结果为 0xFF，否则 Rz 对应字节的结果为 0x0。
影响标志位:	无影响
限制:	无
异常:	无

**指令格式：**

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 1 0 0 1 0 0 1 S 1 0		Rz	

图 15.48: PCMPHS.(US)8

### 15.49 PCMPHS.(U/S)16——16 位并行（无/有）符号大于等于比较指令

<b>统一化指令</b>	
<b>语法</b>	pcmphs.u16 rz, rx, ry pcmphs.s16 rz, rx, ry
<b>操作</b>	Rz[31:16] = (Rx[31:16] >= Ry[31:16]) ? 16' hFFFF:16' b0000 Rz[15:0] = (Rx[15:0] >= Ry[15:0]) ? 16' hFFFF:16' b0000
<b>编译结果</b>	仅存在 32 位指令。 pcmphs.u16 rz, rx, ry pcmphs.s16 rz, rx, ry

<b>说明:</b>	分别判定 Rx 的高低半字是否大于等于 Ry 的高低半字。如果 Rx 的半字大于等于 Ry 的半字，Rz 对应半字的结果为 0xFFFF，否则 Rz 对应字节的结果为 0x0。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位 指令</b>	
<b>操作:</b>	$Rz[31:16] = (Rx[31:16] \geq Ry[31:16]) ? 16' hFFFF:16' b0000$ $Rz[15:0] = (Rx[15:0] \geq Ry[15:0]) ? 16' hFFFF:16' b0000$
<b>语法:</b>	pcmphy.u16 rz, rx, ry pcmphy.s16 rz, rx, ry
<b>说明:</b>	分别判定 Rx 的高低半字是否大于等于 Ry 的高低半字。如果 Rx 的半字大于等于 Ry 的半字, Rz 对应半字的结果为 0xFFFF, 否则 Rz 对应字节的结果为 0x0。
<b>影响标 志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

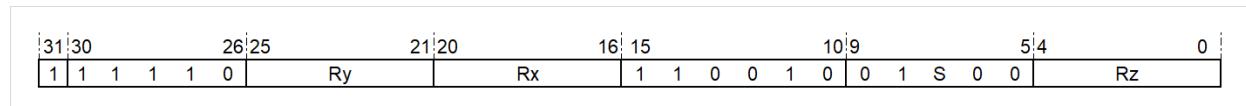
**指令格式:**

图 15.49: PCMPHS.(US)16

**15.50 PCMPLT.(U/S)8——8 位并行 (无/有) 符号小于比较选择指令**

<b>统 一 化 指 令</b>	
<b>语法</b>	pcmplt.u8 rz, rx, ry pcmplt.s8 rz, rx, ry
<b>操作</b>	$Rz[31:24] = (Rx[31:24] < Ry[31:24]) ? 8' hFF:8' h00$ $Rz[23:16] = (Rx[23:16] < Ry[23:16]) ? 8' hFF:8' h00$ $Rz[15:8] = (Rx[15:8] < Ry[15:8]) ? 8' hFF:8' h00$ $Rz[7:0] = (Rx[7:0] < Ry[7:0]) ? 8' hFF:8' h00$
<b>编 译 结 果</b>	仅存在 32 位指令。 pcmplt.u8 rz, rx, ry pcmplt.s8 rz, rx, ry

<b>说明:</b>	以字节为单位，分别判定 Rx 的 4 个字节是否小于 Ry 的 4 个字节。如果 Rx 的字节小于 Ry 的字节，Rz 对应字节的结果为 0xFF，否则 Rz 对应字节的结果为 0x0。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

32 位指令	
<b>操作:</b>	Rz[31:24] = (Rx[31:24] < Ry[31:24]) ? 8' hFF:8' h00 Rz[23:16] = (Rx[23:16] < Ry[23:16]) ? 8' hFF:8' h00 Rz[15:8] = (Rx[15:8] < Ry[15:8]) ? 8' hFF:8' h00 Rz[7:0] = (Rx[7:0] < Ry[7:0]) ? 8' hFF:8' h00
<b>语法:</b>	pcmplt.u8 rz, rx, ry pcmplt.s8 rz, rx, ry
<b>说明:</b>	以字节为单位，分别判定 Rx 的 4 个字节是否小于 Ry 的 4 个字节。如果 Rx 的字节小于 Ry 的字节，Rz 对应字节的结果为 0xFF，否则 Rz 对应字节的结果为 0x0。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

### 指令格式：

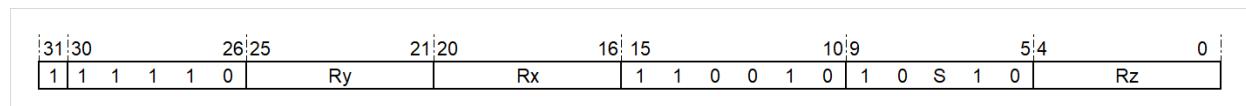


图 15.50: PCMPLT.(US)8

## 15.51 PCMPLT.(U/S)16——16 位并行（无/有）符号小于比较指令

<b>统一化指令</b>	
<b>语法</b>	pcmplt.u16 rz, rx, Ry pcmplt.s16 rz, rx, ry
<b>操作</b>	Rz[31:16] = (Rx[31:16] < Ry[31:16]) ? 16' hFFFF:16' b0000 Rz[15:0] = (Rx[15:0] < Ry[15:0]) ? 16' hFFFF:16' b0000
<b>编译结果</b>	仅存在 32 位指令。 pcmplt.u16 rz, rx, Ry pcmplt.s16 rz, rx, ry

<b>说明:</b>	分别判定 Rx 的高低半字是否小于 Ry 的高低半字。如果 Rx 的半字小于 Ry 的半字，Rz 对应半字的结果为 0xFFFF，否则 Rz 对应字节的结果为 0x0。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	Rz[31:16] = (Rx[31:16] < Ry[31:16]) ? 16' hFFFF:16' b0000 Rz[15:0] = (Rx[15:0] < Ry[15:0]) ? 16' hFFFF:16' b0000
<b>语法:</b>	pcmplt.u16 rz, rx, Ry pcmplt.s16 rz, rx, ry
<b>说明:</b>	分别判定 Rx 的高低半字是否小于 Ry 的高低半字。如果 Rx 的半字小于 Ry 的半字，Rz 对应半字的结果为 0xFFFF，否则 Rz 对应字节的结果为 0x0。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

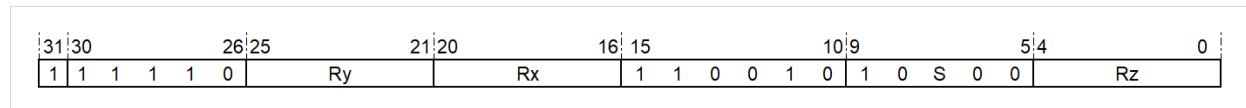


图 15.51: PCMPLT.(US)16

## 15.52 PMAX.(U/S)8——8 位并行（无/有）符号取大值指令

<b>统一化指令</b>	
<b>语法</b>	pmax.u8 rz, rx, ry pmax.s8 rz, rx, ry
<b>操作</b>	Rz[31:24] = Max(Rx[31:24], Ry[31:24]) Rz[23:16] = Max(Rx[23:16], Ry[23:16]) Rz[15:8] = Max(Rx[15:8], Ry[15:8]) Rz[7:0] = Max(Rx[7:0], Ry[7:0])
<b>编译结果</b>	仅存在 32 位指令。 pmax.u8 rz, rx, ry pmax.s8 rz, rx, ry

<b>说明:</b>	以字节为单位，分别选取 Rx 与 Ry 对应字节中的较大者，结果存入 Rz 的对应字节中。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	Rz[31:24] = Max(Rx[31:24], Ry[31:24]) Rz[23:16] = Max(Rx[23:16], Ry[23:16]) Rz[15:8] = Max(Rx[15:8], Ry[15:8]) Rz[7:0] = Max(Rx[7:0], Ry[7:0])
<b>语法:</b>	pmax.u8 rz, rx, ry pmax.s8 rz, rx, ry
<b>说明:</b>	以字节为单位，分别选取 Rx 与 Ry 对应字节中的较大者，结果存入 Rz 的对应字节中。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

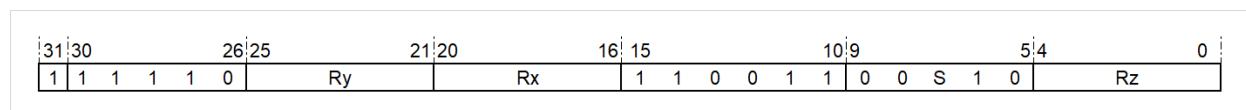


图 15.52: PMAX.(US)8

### 15.53 PMAX.(U/S)16——16 位并行（无/有）符号取大值指令

<b>统一化指令</b>	
<b>语法</b>	pmax.u16 rz, rx, ry pmax.s16 rz, rx, ry
<b>操作</b>	Rz[31:16] = Max(Rx[31:16], Ry[31:16]) Rz[15:0] = Max(Rx[15:0], Ry[15:0])
<b>编译结果</b>	仅存在 32 位指令。 pmax.u16 rz, rx, ry pmax.s16 rz, rx, ry

<b>说明:</b>	以半字为单位，分别选取 Rx 与 Ry 对应半字中的较大者，结果存入 Rz 的对应半字中。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	Rz[31:16] = Max(Rx[31:16], Ry[31:16]) Rz[15:0] = Max(Rx[15:0], Ry[15:0])
<b>语法:</b>	pmax.u16 rz, rx, ry pmax.s16 rz, rx, ry
<b>说明:</b>	以半字为单位，分别选取 Rx 与 Ry 对应半字中的较大者，结果存入 Rz 的对应半字中。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

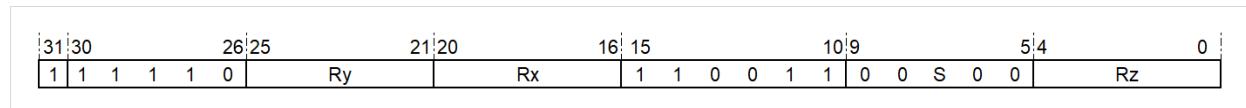


图 15.53: PMAX.(US)16

## 15.54 MAX.(U/S)32——32 位（无/有）符号取大值指令

<b>统一化指令</b>	
<b>语法</b>	max.u32 rz, rx, ry max.s32 rz, rx, ry
<b>操作</b>	Rz[31:0] = Max(Rx[31:0], Ry[31:0])
<b>编译结果</b>	仅存在 32 位指令。 max.u32 rz, rx, ry max.s32 rz, rx, ry

<b>说明:</b>	取 Rx 和 Ry 中的较大者，结果存入 Rz。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	Rz[31:0] = Max(Rx[31:0], Ry[31:0])
<b>语法:</b>	max.u32 rz, rx, ry max.s32 rz, rx, ry
<b>说明:</b>	取 Rx 和 Ry 中的较大者，结果存入 Rz。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

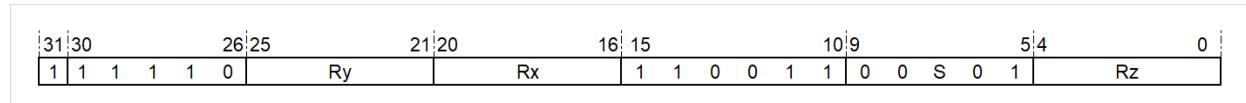


图 15.54: MAX.(US)32

## 15.55 PMIN.(U/S)8——8 位并行（无/有）符号取小值指令

<b>统一化指令</b>	
<b>语法</b>	pmin.u8 rz, rx, ry pmin.s8 rz, rx, ry
<b>操作</b>	Rz[31:24] = Min(Rx[31:24], Ry[31:24]) Rz[23:16] = Min(Rx[23:16], Ry[23:16]) Rz[15:8] = Min(Rx[15:8], Ry[15:8]) Rz[7:0] = Min(Rx[7:0], Ry[7:0])
<b>编译结果</b>	仅存在 32 位指令。 pmin.u8 rz, rx, ry pmin.s8 rz, rx, ry

<b>说明:</b>	以字节为单位，分别选取 Rx 与 Ry 对应字节中的较小者，结果存入 Rz 的对应字节中。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	Rz[31:24] = Min(Rx[31:24], Ry[31:24]) Rz[23:16] = Min(Rx[23:16], Ry[23:16]) Rz[15:8] = Min(Rx[15:8], Ry[15:8]) Rz[7:0] = Min(Rx[7:0], Ry[7:0])
<b>语法:</b>	pmin.u8 rz, rx, ry pmin.s8 rz, rx, ry
<b>说明:</b>	以字节为单位，分别选取 Rx 与 Ry 对应字节中的较小者，结果存入 Rz 的对应字节中。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

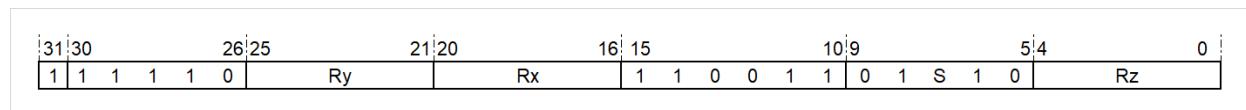


图 15.55: PMIN.(US)8

## 15.56 PMIN.(U/S)16——16 位并行（无/有）符号取小值指令

<b>统一化指令</b>	
<b>语法</b>	pmin.u16 rz, rx, ry pmin.s16 rz, rx, ry
<b>操作</b>	Rz[31:16] = Min(Rx[31:16], Ry[31:16]) Rz[15:0] = Min(Rx[15:0], Ry[15:0])
<b>编译结果</b>	仅存在 32 位指令。 pmin.u16 rz, rx, ry pmin.s16 rz, rx, ry

<b>说明:</b>	以半字为单位，分别选取 Rx 与 Ry 对应半字中的较小者，结果存入 Rz 的对应半字中。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	Rz[31:16] = Min(Rx[31:16], Ry[31:16]) Rz[15:0] = Min(Rx[15:0], Ry[15:0])
<b>语法:</b>	pmin.u16 rz, rx, ry pmin.s16 rz, rx, ry
<b>说明:</b>	以半字为单位，分别选取 Rx 与 Ry 对应半字中的较小者，结果存入 Rz 的对应半字中。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

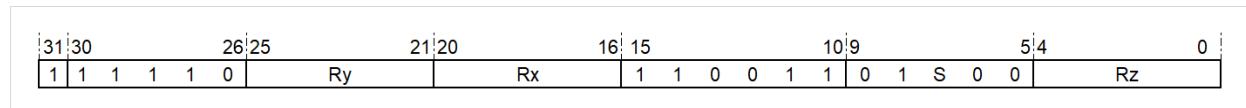


图 15.56: PMIN.(US)16

## 15.57 MIN.(U/S)32——32 位（无/有）符号取小值指令

<b>统一化指令</b>	
<b>语法</b>	min.u32 rz, rx, ry min.s32 rz, rx, ry
<b>操作</b>	$Rz[31:0] = \text{Min}(Rx[31:0], Ry[31:0])$
<b>编译结果</b>	仅存在 32 位指令。 min.u32 rz, rx, ry min.s32 rz, rx, ry

<b>说明:</b>	取 Rx 和 Ry 中的较小者，结果存入 Rz。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	$Rz[31:0] = \text{Min}(Rx[31:0], Ry[31:0])$
<b>语法:</b>	min.u32 Rz, Rx, Ry min.s32 Rz, Rx, Ry
<b>说明:</b>	取 Rx 和 Ry 中的较小者，结果存入 Rz。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

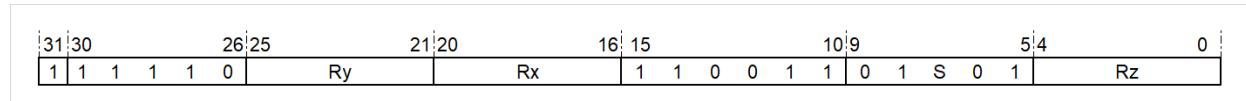


图 15.57: MIN.(US)32

## 15.58 DEXTI——立即数字截取指令

<b>统一化指令</b>	
<b>语法</b>	dexti rz, rx, ry, imm5
<b>操作</b>	$Rz[31:0] = \{Ry[31:0], Rx[31:0]\}[imm[4:0]+31:imm[4:0]]$
<b>编译结果</b>	仅存在 32 位指令。 dexti rz, rx, ry, imm5

<b>说明:</b>	Ry 和 Rx 和组成一个 64 位的数 {Ry[31:0], Rx[31:0]}, 对该数进行逻辑右移, 结果的低字存入 RZ。右移位数由 5 位立即数 (IMM5) 的值决定。
<b>影响标志位:</b>	无影响
<b>限制:</b>	立即数的范围为 0-31
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	$Rz[31:0] = \{Ry[31:0], Rx[31:0]\}[imm[4:0]+31:imm[4:0]]$
<b>语法:</b>	dexti rz, rx, ry, imm5
<b>说明:</b>	Ry 和 Rx 和组成一个 64 位的数 {Ry[31:0], Rx[31:0]}, 对该数进行逻辑右移, 结果的低字存入 RZ。右移位数由 5 位立即数 (IMM5) 的值决定。
<b>影响标志位:</b>	无影响
<b>限制:</b>	立即数的范围为 0-31
<b>异常:</b>	无

**指令格式:**

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 1 1 0	Imm5	Rz	

图 15.58: DEXTI

## 15.59 EXT——字截取指令

<b>统一化指令</b>	
<b>语法</b>	dext rz, rx, ry, rs
<b>操作</b>	$Rz[31:0] = \{Ry[31:0], Rx[31:0]\}[Rs[5:0]+31:Rs[5:0]]$
<b>编译结果</b>	仅存在 32 位指令。 dext rz, rx, ry, rs

<b>说明:</b>	Ry 和 Rx 和组成一个 64 位的数 {Ry[31:0], Rx[31:0]}, 对该数进行逻辑右移, 结果的低字存入 RZ。右移位数由 Rs [5:0] 的值决定当 Rs[5:0] 大于 32 时, 右移位数为 32。
<b>影响标志位:</b>	无影响
<b>限制:</b>	Ry[5:0] 的范围为 0-32
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	$Rz[31:0] = \{Ry[31:0], Rx[31:0]\}[Rs[5:0]+31:Rs[5:0]]$
<b>语法:</b>	dext rz, rx, ry, rs
<b>说明:</b>	Ry 和 Rx 和组成一个 64 位的数 {Ry[31:0], Rx[31:0]}, 对该数进行逻辑右移, 结果的低字存入 RZ。右移位数由 Rs [5:0] 的值决定当 Rs[5:0] 大于 32 时, 右移位数为 32。
<b>影响标志位:</b>	无影响
<b>限制:</b>	Ry[5:0] 的范围为 0-32
<b>异常:</b>	无

**指令格式:**

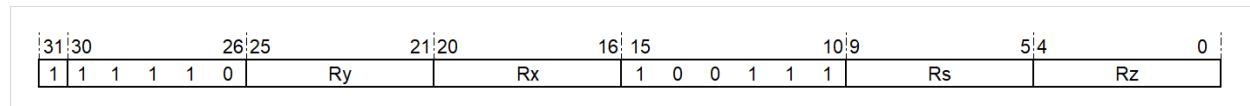


图 15.59: EXT

## 15.60 PKG——立即数移位打包指令

统一化指令	
语法	pkg rz, rx, imm4a, ry, oimm4b
操作	$Rz[31:0] = \{(Ry[31:0] >> oimm4b[3:0])[15:0], (Rx[31:0] >> imm4a[3:0])[15:0]\}$
编译结果	仅存在 32 位指令。 pkg rz, rx, imm4a, ry, oimm4b

说明:	对 Rx 逻辑右移, 截取移位结果的低半字存入 RZ 的低半字, 右移位数由立即数 (imm4a[3:0]) 的值决定; 对 RY 逻辑右移, 截取移位结果的低半字存入 RZ 的高半字, 右移位数由带偏置 1 的立即数 (oimm4b[3:0]) 的值决定。
影响标志位:	无影响
限制:	imm4a 的范围为 0~15 oimm4b 的范围为 1~16
异常:	无

32 位指令	
操作:	$Rz[31:0] = \{(Ry[31:0] >> oimm4b[3:0])[15:0], (Rx[31:0] >> imm4a[3:0])[15:0]\}$
语法:	pkg rz, rx, imm4a, ry, oimm4b
说明:	对 Rx 逻辑右移, 截取移位结果的低半字存入 RZ 的低半字, 右移位数由立即数 (imm4a[3:0]) 的值决定; 对 RY 逻辑右移, 截取移位结果的低半字存入 RZ 的高半字, 右移位数由带偏置 1 的立即数 (oimm4b[3:0]) 的值决定。
影响标志位:	无影响
限制:	imm4a 的范围为 0~15 oimm4b 的范围为 1~16
异常:	无

指令格式:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 1	oimm4b	oimm4a	Rz

图 15.60: PKG

### 15.61 PKGLL——低半字打包指令

统一化指令	
语法	pkgll rz, rx, ry
操作	$Rz[31:0] = \{Ry[15:0], Rx[15:0]\}$
编译结果	仅存在 32 位指令。 pkgll rz, rx, ry

说明:	将 Ry 的低半字存入 Rz 的高半字, Rx 的低半字存入 Rz 的低半字。
影响标志位:	无影响
限制:	无
异常:	无

32 位指令	
操作:	$Rz[31:0] = \{Ry[15:0], Rx[15:0]\}$
语法:	pkgll rz, rx, ry
说明:	将 Ry 的低半字存入 Rz 的高半字, Rx 的低半字存入 Rz 的低半字。
影响标志位:	无影响
限制:	无
异常:	无

指令格式:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 1 0 1 1 0 0 0 x 1 0		Rz	

图 15.61: PKGLL

## 15.62 PKGHH——高半字打包指令

<b>统一化指令</b>	
<b>语法</b>	pkghh rz, rx, ry
<b>操作</b>	$Rz[31:0] = \{Ry[31:16], Rx[31:16]\}$
<b>编译结果</b>	仅存在 32 位指令。 pkghh rz, rx, ry

<b>说明:</b>	将 Ry 的高半字存入 Rz 的高半字, Rx 的高半字存入 Rz 的低半字。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	$Rz[31:0] = \{Ry[31:16], Rx[31:16]\}$
<b>语法:</b>	pkghh rz, rx, ry
<b>说明:</b>	将 Ry 的高半字存入 Rz 的高半字, Rx 的高半字存入 Rz 的低半字。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

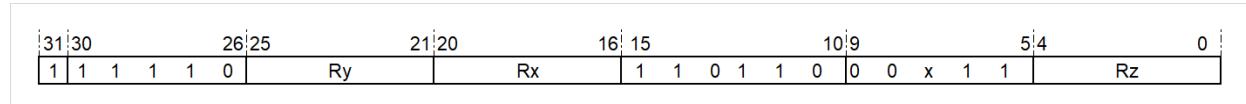


图 15.62: PKGHH

### 15.63 PEXT.U8.E——8 位并行无符号扩展指令

<b>统一化指令</b>	
<b>语法</b>	pext.u8.e rz, rx
<b>操作</b>	$Rz+1[31:16] = \text{zero\_extend}(Rx[31:24])$ $Rz+1[15:0] = \text{zero\_extend}(Rx[23:16])$ $Rz[31:16] = \text{zero\_extend}(Rx[15:8])$ $Rz[15:0] = \text{zero\_extend}(Rx[7:0])$
<b>编译结果</b>	仅存在 32 位指令。 pext.u8.e rz, rx

<b>说 明:</b>	定义 Rx[31:24]、Rx[23:16]、Rx[15:8]、Rx[7:0] 分别为字节 3、字节 2、字节 1、字节 0。依次对字节 3、字节 2、字节 1、字节 0 进行无符号扩展，扩展后的数据格式为无符号半字，结果依次存入 Rz+1 的高半字、Rz+1 的低半字、Rz 的高半字、Rz 的低半字。
<b>影 响 标 志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

<b>32 位 指 令</b>	
<b>操 作:</b>	Rz+1[31:16] = zero_extend(Rx[31:24]) Rz+1[15:0] = zero_extend(Rx[23:16]) Rz[31:16] = zero_extend(Rx[15:8]) Rz[15:0] = zero_extend(Rx[7:0])
<b>语 法:</b>	pext.u8.e rz, rx
<b>说 明:</b>	定义 Rx[31:24]、Rx[23:16]、Rx[15:8]、Rx[7:0] 分别为字节 3、字节 2、字节 1、字节 0。依次对字节 3、字节 2、字节 1、字节 0 进行无符号扩展，扩展后的数据格式为无符号半字，结果依次存入 Rz+1 的高半字、Rz+1 的低半字、Rz 的高半字、Rz 的低半字。
<b>影 响 标 志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

**指令格式:**

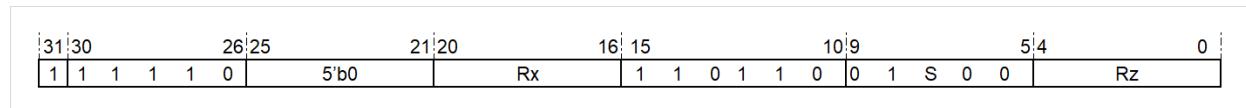


图 15.63: PEXT.U8.E

## 15.64 PEXT.S8.E——8 位并行有符号扩展指令

<b>统一化指令</b>	
<b>语法</b>	pext.s8.e rz, rx
<b>操作</b>	$Rz+1[31:16] = \text{sign\_extend}(Rx[31:24])$ $Rz+1[15:0] = \text{sign\_extend}(Rx[23:16])$ $Rz[31:16] = \text{sign\_extend}(Rx[15:8])$ $Rz[15:0] = \text{sign\_extend}(Rx[7:0])$
<b>编译结果</b>	仅存在 32 位指令。 pext.s8.e rz, rx

<b>说 明:</b>	定义 Rx[31:24]、Rx[23:16]、Rx[15:8]、Rx[7:0] 分别为字节 3、字节 2、字节 1、字节 0。依次对字节 3、字节 2、字节 1、字节 0 进行有符号扩展，扩展后的数据格式为有符号半字，结果依次存入 Rz+1 的高半字、Rz+1 的低半字、Rz 的高半字、Rz 的低半字。
<b>影 响 标 志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

<b>32 位 指 令</b>	
<b>操 作:</b>	Rz+1[31:16] = sign_extend(Rx[31:24]) Rz+1[15:0] = sign_extend(Rx[23:16]) Rz[31:16] = sign_extend(Rx[15:8]) Rz[15:0] = sign_extend(Rx[7:0])
<b>语 法:</b>	pext.s8.e rz, rx
<b>说 明:</b>	定义 Rx[31:24]、Rx[23:16]、Rx[15:8]、Rx[7:0] 分别为字节 3、字节 2、字节 1、字节 0。依次对字节 3、字节 2、字节 1、字节 0 进行有符号扩展，扩展后的数据格式为有符号半字，结果依次存入 Rz+1 的高半字、Rz+1 的低半字、Rz 的高半字、Rz 的低半字。
<b>影 响 标 志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

**指令格式:**

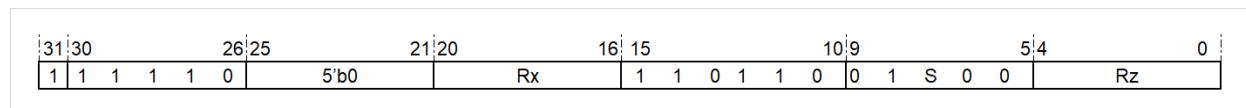


图 15.64: PEXT.S8.E

## 15.65 PEXTX.U8.E——8 位并行无符号交叉扩展指令

<b>统一化指令</b>	
<b>语法</b>	pextx.u8.e rz, rx
<b>操作</b>	Rz+1[31:16] = zero_extend(Rx[31:24]) Rz+1[15:0] = zero_extend(Rx[15:8]) Rz[31:16] = zero_extend(Rx[23:16]) Rz[15:0] = zero_extend(Rx[7:0])
<b>编译结果</b>	仅存在 32 位指令。 pextx.u8.e rz, rx

<b>说明:</b>	定义 Rx[31:24]、Rx[23:16]、Rx[15:8]、Rx[7:0] 分别为字节 3、字节 2、字节 1、字节 0。依次对字节 3、字节 1、字节 2、字节 0 进行无符号扩展，扩展后的数据格式为无符号半字，结果依次存入 Rz+1 的高半字、Rz+1 的低半字、Rz 的高半字、Rz 的低半字。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位 指 令</b>	
<b>操 作:</b>	Rz+1[31:16] = zero_extend(Rx[31:24]) Rz+1[15:0] = zero_extend(Rx[15:8]) Rz[31:16] = zero_extend(Rx[23:16]) Rz[15:0] = zero_extend(Rx[7:0])
<b>语 法:</b>	pextx.u8.e rz, rx
<b>说 明:</b>	定义 Rx[31:24]、Rx[23:16]、Rx[15:8]、Rx[7:0] 分别为字节 3、字节 2、字节 1、字节 0。依次对字节 3、字节 1、字节 2、字节 0 进行无符号扩展，扩展后的数据格式为无符号半字，结果依次存入 Rz+1 的高半字、Rz+1 的低半字、Rz 的高半字、Rz 的低半字。
<b>影 响 标 志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

**指令格式:**

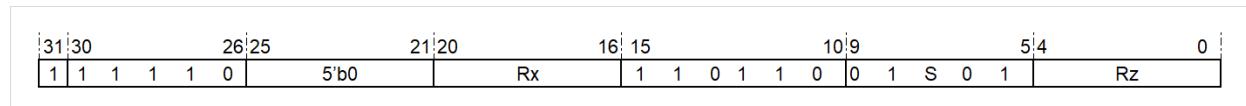


图 15.65: PEXTX.U8.E

## 15.66 PEXTX.S8.E——8 位并行有符号交叉扩展指令

<b>统一化指令</b>	
<b>语法</b>	pextx.s8.e rz, rx
<b>操作</b>	Rz+1[31:16] = sign_extend(Rx[31:24]) Rz+1[15:0] = sign_extend(Rx[15:8]) Rz[31:16] = sign_extend(Rx[23:16]) Rz[15:0] = sign_extend(Rx[7:0])
<b>编译结果</b>	仅存在 32 位指令。 pextx.s8.e rz, rx

<b>说 明:</b>	定义 Rx[31:24]、Rx[23:16]、Rx[15:8]、Rx[7:0] 分别为字节 3、字节 2、字节 1、字节 0。依次对字节 3、字节 1、字节 2、字节 0 进行有符号扩展，扩展后的数据格式为有符号半字，结果依次存入 Rz+1 的高半字、Rz+1 的低半字、Rz 的高半字、Rz 的低半字。
<b>影 响 标 志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

<b>32 位 指 令</b>	
<b>操 作:</b>	Rz+1[31:16] = sign_extend(Rx[31:24]) Rz+1[15:0] = sign_extend(Rx[15:8]) Rz[31:16] = sign_extend(Rx[23:16]) Rz[15:0] = sign_extend(Rx[7:0])
<b>语 法:</b>	pextx.s8.e rz, rx
<b>说 明:</b>	定义 Rx[31:24]、Rx[23:16]、Rx[15:8]、Rx[7:0] 分别为字节 3、字节 2、字节 1、字节 0。依次对字节 3、字节 1、字节 2、字节 0 进行有符号扩展，扩展后的数据格式为有符号半字，结果依次存入 Rz+1 的高半字、Rz+1 的低半字、Rz 的高半字、Rz 的低半字。
<b>影 响 标 志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

**指令格式:**

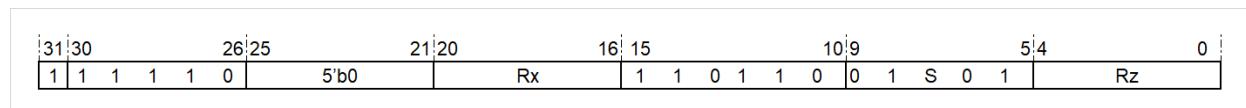


图 15.66: PEXTX.S8.E

## 15.67 NARL——低位截取拼装指令

<b>统一化指令</b>	
<b>语法</b>	narl rz, rx, ry
<b>操作</b>	Rz[31:0] = {Ry[23:16], Ry[7:0], Rx[23:16], Rx[7:0]}
<b>编译结果</b>	仅存在 32 位指令。 narl rz, rx, ry

<b>说明:</b>	定义 Ry 的高半字、低半字, Rx 的高半字、低半字分别为半字 3、2、1、0。截取半字 3、2、1、0 的低 8 位, 分别存入 Rz[31:24]、Rz[23:16]、Rz[15:8]、Rz[7:0]。
<b>影响标志:</b>	无影响
<b>定位:</b>	无
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	Rz[31:0] = { Ry[23:16], Ry[7:0], Rx[23:16], Rx[7:0] }
<b>语法:</b>	narl rz, rx, ry
<b>说明:</b>	定义 Ry 的高半字、低半字, Rx 的高半字、低半字分别为半字 3、2、1、0。截取半字 3、2、1、0 的低 8 位, 分别存入 Rz[31:24]、Rz[23:16]、Rz[15:8]、Rz[7:0]。
<b>影响标志:</b>	无影响
<b>定位:</b>	无
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 1 1 0	Ry	Rx	1 1 0 1 1 0	1 0 x 0 0	Rz	

图 15.67: NARL

## 15.68 NARH——高位截取拼接指令

<b>统一化指令</b>	
<b>语法</b>	narh rz, rx, ry
<b>操作</b>	Rz[31:0] = { Ry[31:24], Ry[15:8], Rx[31:24], Rx[15:8] }
<b>编译结果</b>	仅存在 32 位指令。 narh rz, rx, ry

<b>说明:</b>	定义 Ry 的高半字、低半字, Rx 的高半字、低半字分别为半字 3、2、1、0。截取半字 3、2、1、0 的高 8 位, 分别存入 Rz[31:24]、Rz[23:16]、Rz[15:8]、Rz[7:0]。
<b>影响标志:</b>	无影响
<b>定位:</b>	无
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	Rz[31:0] = {Ry[31:24], Ry[15:8], Rx[31:24], Rx[15:8]}
<b>语法:</b>	narh rz, rx, ry
<b>说明:</b>	定义 Ry 的高半字、低半字, Rx 的高半字、低半字分别为半字 3、2、1、0。截取半字 3、2、1、0 的高 8 位, 分别存入 Rz[31:24]、Rz[23:16]、Rz[15:8]、Rz[7:0]。
<b>影响标志:</b>	无影响
<b>定位:</b>	无
<b>限制:</b>	无
<b>异常:</b>	无

指令格式:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 1 1 0	Ry	Rx	1 1 0 1 1 0	1 0 x 0 1	Rz	

图 15.68: NARH

## 15.69 NARLX——低位交叉截取拼接指令

<b>统一化指令</b>	
<b>语法</b>	narlx rz, rx, ry
<b>操作</b>	Rz[31:0] = {Ry[23:16], Rx[23:16], Ry[7:0], Rx[7:0]}
<b>编译结果</b>	仅存在 32 位指令。 narlx rz, rx, ry

<b>说明:</b>	定义 Ry 的高半字、低半字, Rx 的高半字、低半字分别为半字 3、2、1、0。截取半字 3、1、2、0 的低 8 位, 分别存入 Rz[31:24]、Rz[23:16]、Rz[15:8]、Rz[7:0]
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	Rz[31:0] = {Ry[23:16], Rx[23:16], Ry[7:0], Rx[7:0]}
<b>语法:</b>	narlx rz, rx, ry
<b>说明:</b>	定义 Ry 的高半字、低半字, Rx 的高半字、低半字分别为半字 3、2、1、0。截取半字 3、1、2、0 的低 8 位, 分别存入 Rz[31:24]、Rz[23:16]、Rz[15:8]、Rz[7:0]
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 1 1 0	Ry	Rx	1 1 0 1 1 0	1 0 x 1 0	Rz	

图 15.69: NARLX

## 15.70 NARHX——高位交叉截取拼接指令

<b>统一化指令</b>	
<b>语法</b>	narhx rz, rx, ry
<b>操作</b>	Rz[31:0] = {Ry[31:24], Rx[31:24], Ry[15:8], Rx[15:8]}
<b>编译结果</b>	仅存在 32 位指令。 narhx rz, rx, ry

<b>说明:</b>	定义 Ry 的高半字、低半字, Rx 的高半字、低半字分别为半字 3、2、1、0。截取半字 3、1、2、0 的高 8 位, 分别存入 Rz[31:24]、Rz[23:16]、Rz[15:8]、Rz[7:0]
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	Rz[31:0] = {Ry[31:24], Rx[31:24], Ry[15:8], Rx[15:8]}
<b>语法:</b>	narhx rz, rx, ry
<b>说明:</b>	定义 Ry 的高半字、低半字, Rx 的高半字、低半字分别为半字 3、2、1、0。截取半字 3、1、2、0 的高 8 位, 分别存入 Rz[31:24]、Rz[23:16]、Rz[15:8]、Rz[7:0]
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

### 指令格式:

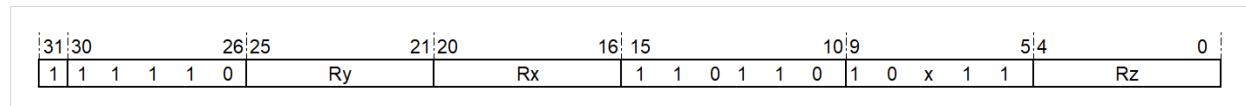


图 15.70: NARHX

## 15.71 CLIP1.(U/S)32——立即数（无/有）符号裁剪饱和指令

统 一 化 指 令	
语 法	clipi.u32 rz, rx, imm5 clipi.s32 rz, rx, oimm5
操 作	$\text{Max} = 2(\text{imm5})-1$ , $\text{Min} = 0$ //unsigned $\text{Max} = 2(\text{oimm5}-1)-1$ , $\text{Min} = -2(\text{oimm5}-1)$ //signed if( $\text{Rx}[31:0] > \text{Max}$ ) $\text{Rz}[31:0] = \text{Max}$ else if( $\text{Rx}[31:0] < \text{Min}$ ) $\text{Rz}[31:0] = \text{Min}$ else $\text{Rz}[31:0] = \text{Rx}[31:0]$
编 译  结 果	<p>仅存在 32 位指令。</p> clipi.u32 rz, rx, imm5 clipi.s32 rz, rx, oimm5

说 明:	将 Rx 缩位保存到由 IMM5/OIMM5 指定的目标数据中，其中 IMM5/OIMM5 表征了目标数据的位宽。如果 Rx 的值超过了目标数据可以表示的数值范围，则进行饱和处理。  对于无符号 CLIP 指令，上饱和值为 $2^{\text{IMM5}}-1$ ，下饱和值为 0x0。若 Rx 大于上饱和值，结果为 $2^{\text{IMM5}}-1$ ，其他情况结果为 Rx。  对于有符号 CLIP 指令，OIMM5 为带偏置 1 的立即数，上饱和值为 $2^{(\text{OIMM5}-1)}-1$ ，下饱和值为 $-2^{(\text{OIMM5}-1)}$ 。若 Rx 大于上饱和值，结果为 $2^{(\text{OIMM5}-1)}-1$ ，若 Rx 小于下饱和值，结果为 $-2^{(\text{OIMM5}-1)}$ ，其他情况结果为 Rx。
影 响 标 志 位:	无影响
限 制:	IMM5 的范围为 0~31 OIMM5 的范围为 1~32
异 常:	无

<b>32位指令</b>	
<b>操作:</b>	If(unsigned)Max = 2(imm5)-1, Min = 0 // If(signed)Max = 2(oimm5-1)-1, Min = -2(oimm5-1) // if(Rx[31:0] > Max) Rz[31:0] = Max else if(Rx[31:0] < Min) Rz[31:0] = Min else Rz[31:0] = Rx[31:0]
<b>语法:</b>	clipi.u32 rz, rx, imm5 clipi.s32 rz, rx, oimm5
<b>说明:</b>	将 Rx 缩位保存到由 IMM5/OIMM5 指定的目标数据中，其中 IMM5/OIMM5 表征了目标数据的位宽。如果 Rx 的值超过了目标数据可以表示的数值范围，则进行饱和处理。 对于无符号 CLIP 指令，上饱和值为 $2^{\text{IMM5}}-1$ ，下饱和值为 0x0。若 Rx 大于上饱和值，结果为 $2^{\text{IMM5}}-1$ ，其他情况结果为 Rx。 对于有符号 CLIP 指令，OIMM5 为带偏置 1 的立即数，上饱和值为 $2^{(\text{OIMM5}-1)}-1$ ，下饱和值为 $-2^{(\text{OIMM5}-1)}$ 。若 Rx 大于上饱和值，结果为 $2^{(\text{OIMM5}-1)}-1$ ，若 Rx 小于下饱和值，结果为 $-2^{(\text{OIMM5}-1)}$ ，其他情况结果为 Rx。
<b>影响标志位:</b>	无影响
<b>限制:</b>	IMM5 的范围为 0~31 OIMM5 的范围为 1~32
<b>异常:</b>	无

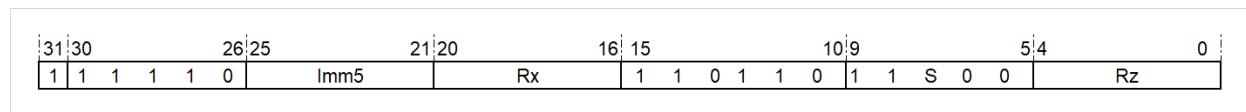
**指令格式：**

图 15.71: CLIP.(US)32

## 15.72 CLIP.(U/S)32——(无/有) 符号裁剪饱和指令

统 一 化 指 令	
语 法	clip.u32 rz, rx, ry clip.s32 rz, rx, ry
操 作	<pre>Max = 2(Ry[4:0])-1 Min = 0 //unsigned Max = 2(Ry[4:0]-1)-1 Min = -2(Ry[4:0]-1) //signed if(Rx[31:0] &gt; Max)     Rx[31:0] = Max else if(Rx[31:0] &lt; Min)     Rx[31:0] = Min else     Rx[31:0] = Rx[31:0]</pre>
编 译 结 果	仅存在 32 位指令。 clip.u32 rz, rx, ry clip.s32 rz, rx, ry

说 明:	将 Rx 缩位保存到由 Ry[4:0] 指定的目标数据中，其中 Ry[4:0] 表征了目标数据的位宽。如果 Rx 的值超过了目标数据可以表示的数值范围，则进行饱和处理。  对于无符号 CLIP 指令，上饱和值为 $2^{\lceil Ry[4:0] \rceil} - 1$ ，下饱和值为 0x0。若 Rx 大于上饱和值，结果为 $2^{\lceil Ry[4:0] \rceil} - 1$ ，其他情况结果为 Rx。  对于有符号 CLIP 指令，上饱和值为 $2^{\lceil (Ry[4:0] - 1) \rceil} - 1$ ，下饱和值为 $-2^{\lceil (Ry[4:0] - 1) \rceil}$ 。若 Rx 大于上饱和值，结果为 $2^{\lceil (Ry[4:0] - 1) \rceil} - 1$ ，若 Rx 小于下饱和值，结果为 $-2^{\lceil (Ry[4:0] - 1) \rceil}$ ，其他情况结果为 Rx。
影 响 标 志 位:	无影响
限 制:	无
异 常:	无

32 位 指 令	
操作: 作:	Max = 2(Ry[4:0])-1 Min = 0 //unsigned Max = 2(Ry[4:0]-1)-1 Min = -2(Ry[4:0]-1) //signed if(Rx[31:0] > Max) Rz[31:0] = Max else if(Rx[31:0] < Min) Rz[31:0] = Min else Rz[31:0] = Rx[31:0]
语法: 法:	clip.u32 rz, rx, ry clip.s32 rz, rx, ry
说明: 明:	将 Rx 缩位保存到由 Ry[4:0] 指定的目标数据中，其中 Ry[4:0] 表征了目标数据的位宽。如果 Rx 的值超过了目标数据可以表示的数值范围，则进行饱和处理。 对于无符号 CLIP 指令，上饱和值为 $2^{\lceil Ry[4:0] \rceil} - 1$ ，下饱和值为 0x0。若 Rx 大于上饱和值，结果为 $2^{\lceil Ry[4:0] \rceil} - 1$ ，其他情况结果为 Rx。 对于有符号 CLIP 指令，上饱和值为 $2^{\lceil (Ry[4:0]-1) \rceil} - 1$ ，下饱和值为 $-2^{\lceil (Ry[4:0]-1) \rceil}$ 。若 Rx 大于上饱和值，结果为 $2^{\lceil (Ry[4:0]-1) \rceil} - 1$ ，若 Rx 小于下饱和值，结果为 $-2^{\lceil (Ry[4:0]-1) \rceil}$ ，其他情况结果为 Rx。
影响 标志 位: 位:	无影响
限制: 制:	无
异常: 常:	无

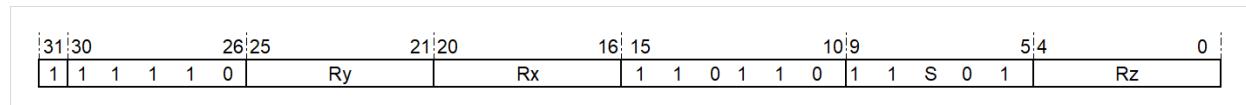
**指令格式：**

图 15.72: CLIP.(US)32

## 15.73 PCLIP1.(U/S)16——16 位并行立即数（无/有）符号裁剪饱和指令

统 一 化 指 令	
语 法	pclip.u16 rz, rx, imm4 pclip.s16 rz, rx, oimm4
操 作	<pre>Max = 2(imm4)-1, Min = 0 //unsigned Max = 2(oimm4-1)-1, Min = -2(oimm4-1) //signed if(Rx[31:16] Rx[15:0] &gt; Max)     Rz[31:16] Rz[15:0] = Max else if(Rx[31:16] Rx[15:0] &lt; Min)     Rz[31:16] Rz[15:0] = Min else     Rz[31:16] Rz[15:0] = Rx[31:16] Rx[15:0]</pre>
编 译 结 果	仅存在 32 位指令。 pclip.u16 rx, rx, imm4 pclip.s16 rx, rx, oimm4

说 明:	Rx 的高低半字作为原始数据, 分别缩位保存到由 IMM4/OIMM4 指定的目标数据中, 其中 IMM4/OIMM4 表征了目标数据的位宽。如果原始数据的值超过了目标数据可以表示的数值范围, 则进行饱和处理。 对于无符号 CLIP 指令, 上饱和值为 $2^{\text{IMM4}-1}$ , 下饱和值为 0x0。若原始数据大于上饱和值, 结果为 $2^{\text{IMM4}-1}$ , 其他情况结果为原始数据本身。 对于有符号 CLIP 指令, OIM M4 为带偏置 1 的立即数, 上饱和值为 $2^{(\text{OIMM4}-1)-1}$ , 下饱和值为 $-2^{(\text{OIMM4}-1)}$ 。若原始数据大于上饱和值, 结果为 $2^{(\text{OIMM4}-1)-1}$ , 若原始小于下饱和值, 结果为 $-2^{(\text{OIMM4}-1)}$ , 其他情况结果为原始数据本身。
影 响 标 志 位:	无影响
限 制:	Imm4 的范围为 0~15 Oimm4 的范围为 1~16
异 常:	无

<b>32位指令</b>	
<b>操作:</b>	Max = 2(imm4)-1, Min = 0 //unsigned Max = 2(oimm4-1)-1, Min = -2(oimm4-1) //signed if(Rx[31:16] Rx[15:0] > Max) Rz[31:16] Rz[15:0] = Max else if(Rx[31:16] Rx[15:0] < Min) Rz[31:16] Rz[15:0] = Min else Rz[31:16] Rz[15:0] = Rx[31:16] Rx[15:0]
<b>语法:</b>	pclip.u16 rx, rx, imm4 pclip.s16 rx, rx, oimm4
<b>说明:</b>	Rx 的高低半字作为原始数据, 分别缩位保存到由 IMM4/OIMM4 指定的目标数据中, 其中 IMM4/OIMM4 表征了目标数据的位宽。如果原始数据的值超过了目标数据可以表示的数值范围, 则进行饱和处理。 对于无符号 CLIP 指令, 上饱和值为 $2^{\text{IMM4}}-1$ , 下饱和值为 0x0。若原始数据大于上饱和值, 结果为 $2^{\text{IMM4}}-1$ , 其他情况结果为原始数据本身。 对于有符号 CLIP 指令, OIMM4 为带偏置 1 的立即数, 上饱和值为 $2^{(\text{OIMM4}-1)}-1$ , 下饱和值为 $-2^{(\text{OIMM4}-1)}$ 。若原始数据大于上饱和值, 结果为 $2^{(\text{OIMM4}-1)}-1$ , 若原始小于下饱和值, 结果为 $-2^{(\text{OIMM4}-1)}$ , 其他情况结果为原始数据本身。
<b>影响标志位:</b>	无影响
<b>限制:</b>	Imm5 的范围为 0~15 Oimm5 的范围为 1~16
<b>异常:</b>	无

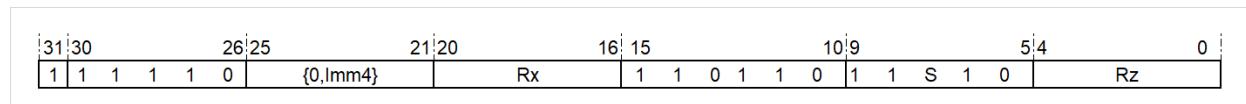
**指令格式:**

图 15.73: PCLIP1.(US)16

## 15.74 PCLIP.(U/S)16——16 位并行（无/有）符号裁剪饱和指令

统 一 化 指 令	
语 法	pclip.u16 rz, rx, ry pclip.s16 rz, rx, ry
操 作	Max = $2(Ry[3:0])-1$ , Min = 0 //unsigned Max = $2(Ry[3:0]-1)-1$ , Min = $-2(Ry[3:0]-1)$ //signed if( $Rx[31:16] Rx[15:0] > Max$ ) Rz[31:16] Rz[15:0] = Max else if( $Rx[31:16] Rx[15:0] < Min$ ) Rz[31:16] Rz[15:0] = Min else Rz[31:16] Rz[15:0] = Rx[31:16] Rx[15:0]
编 译 结 果	仅存在 32 位指令。 pclip.u16 rx, rx, ry pclip.s16 rx, rx, ry

说 明:	Rx 的高低半字作为原始数据，分别缩位保存到由 Ry[3:0] 指定的目标数据中，其中 Ry[3:0]4 表征了目标数据的位宽。如果原始数据的值超过了目标数据可以表示的数值范围，则进行饱和处理。 对于无符号 CLIP 指令，上饱和值为 $2^Ry[3:0]-1$ ，下饱和值为 0x0。若原始数据大于上饱和值，结果为 $2^Ry[3:0]-1$ ，其他情况结果为原始数据本身。 对于有符号 CLIP 指令，上饱和值为 $2^{Ry[3:0]-1}-1$ ，下饱和值为 $-2^{Ry[3:0]-1}$ 。若原始数据大于上饱和值，结果为 $2^{Ry[3:0]-1}-1$ ，若原始数据小于下饱和值，结果为 $-2^{Ry[3:0]-1}$ ，其他情况结果为原始数据本身。
影 响 标 志 位:	无影响
限 制:	无
异 常:	无

<b>32 位 指 令</b>	
<b>操 作:</b>	Max = $2(Ry[3:0])-1$ , Min = 0 //unsigned Max = $2(Ry[3:0]-1)-1$ , Min = $-2(Ry[3:0]-1)$ //signed if( $Rx[31:16] Rx[15:0] > Max$ ) $Rz[31:16] Rz[15:0] = Max$ else if( $Rx[31:16] Rx[15:0] < Min$ ) $Rz[31:16] Rz[15:0] = Min$ else $Rz[31:16] Rz[15:0] = Rx[31:16] Rx[15:0]$
<b>语 法:</b>	pclip.u16 rx, rx, ry pclip.s16 rx, rx, ry
<b>说 明:</b>	Rx 的高低半字作为原始数据，分别缩位保存到由 Ry[3:0] 指定的目标数据中，其中 Ry[3:0]4 表征了目标数据的位宽。如果原始数据的值超过了目标数据可以表示的数值范围，则进行饱和处理。 对于无符号 CLIP 指令，上饱和值为 $2^Ry[3:0]-1$ ，下饱和值为 0x0。若原始数据大于上饱和值，结果为 $2^Ry[3:0]-1$ ，其他情况结果为原始数据本身。 对于有符号 CLIP 指令，上饱和值为 $2^{(Ry[3:0]-1)}-1$ ，下饱和值为 $-2^{(Ry[3:0]-1)}$ 。若原始数据大于上饱和值，结果为 $2^{(Ry[3:0]-1)}-1$ ，若原始数据小于下饱和值，结果为 $-2^{(Ry[3:0]-1)}$ ，其他情况结果为原始数据本身。
<b>影 响 标 志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

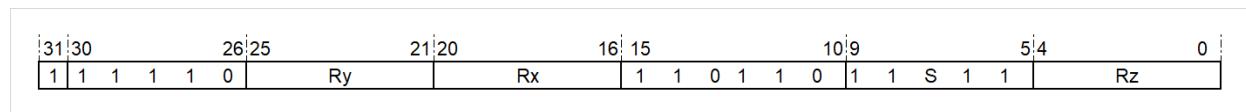
**指令格式:**

图 15.74: PCLIP.(US)16

## 15.75 PABS.S8.S——带饱和操作的 8 位并行绝对值指令

<b>统一化指令</b>	
<b>语法</b>	<b>操作</b>
pabs.s8.s rz, rx	Rz[31:24] = Saturate(abs(Rx[31:24])) Rz[23:16] = Saturate(abs(Rx[23:16])) Rz[15:8] = Saturate(abs(Rx[15:8])) Rz[7:0] = Saturate(abs(Rx[7:0]))
<b>编译结果</b>	仅存在 32 位指令。 pabs.s8.s rz, rx

<b>说明:</b>	以字节为单位, 对 Rx 的字节依次取绝对值, 并做饱和处理。当字节为 0x80 时, 绝对值饱和处理的结果为 0x7F, 其余情况与绝对值结果一样。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	Rz[31:24] = Saturate(abs(Rx[31:24])) Rz[23:16] = Saturate(abs(Rx[23:16])) Rz[15:8] = Saturate(abs(Rx[15:8])) Rz[7:0] = Saturate(abs(Rx[7:0]))
<b>语法:</b>	pabs.s8.s rz, rx
<b>说明:</b>	以字节为单位, 对 Rx 的字节依次取绝对值, 并做饱和处理。当字节为 0x8000 时, 绝对值饱和处理的结果为 0x7FFF, 其余情况与绝对值结果一样。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

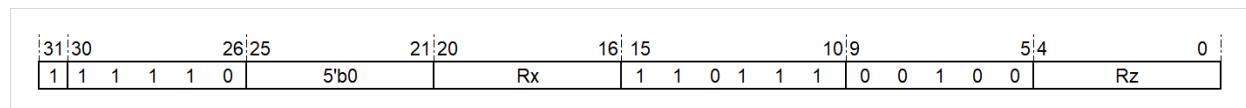


图 15.75: PABS.S8.S

## 15.76 PABS.S16.S——带饱和操作的 16 位并行绝对值指令

<b>统一化指令</b>	
<b>语法</b>	pabs.s16.s rz, rx
<b>操作</b>	Rz[31:16] = Saturate(abs(Rx[31:16])) Rz[15:0] = Saturate(abs(Rx[15:0]))
<b>编译结果</b>	仅存在 32 位指令。 pabs.s16.s rz, rx

<b>说明:</b>	对 Rx 的高低半字依次取绝对值，并做饱和处理。当半字为 0x8000 时，绝对值饱和处理的结果为 0x7FFF，其余情况与绝对值结果一样。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	Rz[31:16] = Saturate(abs(Rx[31:16])) Rz[15:0] = Saturate(abs(Rx[15:0]))
<b>语法:</b>	pabs.s16.s rz, rx
<b>说明:</b>	对 Rx 的高低半字依次取绝对值，并做饱和处理。当半字为 0x8000 时，绝对值饱和处理的结果为 0x7FFF，其余情况与绝对值结果一样。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

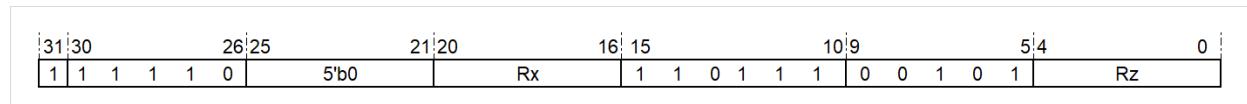


图 15.76: PABS.S16.S

## 15.77 ABS.S32.S——带饱和操作的 32 位绝对值指令

<b>统一化指令</b>	
<b>语法</b>	abs.s32.s rz, rx
<b>操作</b>	$Rz[31:0] = \text{Saturate}(\text{abs}(Rx[31:0]))$
<b>编译结果</b>	仅存在 32 位指令。 abs.s32.s rz, rx

<b>说明:</b>	对 Rx 取绝对值并做饱和处理。当 Rx 为 0x8000 0000 时，绝对值饱和处理的结果为 0x7FFF FFFF，其余情况与绝对值结果一样。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	$Rz[31:0] = \text{Saturate}(\text{abs}(Rx[31:0]))$
<b>语法:</b>	abs.s32.s rz, rx
<b>说明:</b>	对 Rx 取绝对值并做饱和处理。当 Rx 为 0x8000 0000 时，绝对值饱和处理的结果为 0x7FFF FFFF，其余情况与绝对值结果一样。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

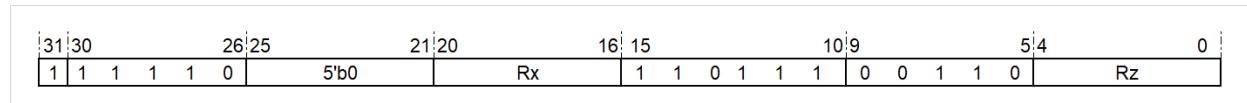


图 15.77: ABS.S32.S

## 15.78 PNEG.S8.S——带饱和操作的 8 位并行取反指令

<b>统一化指令</b>	
<b>语法</b>	pneg.s8.s rz, rx
<b>操作</b>	$Rz[31:24] = \text{Saturate}(\text{neg}(Rx[31:24]))$ $Rz[23:16] = \text{Saturate}(\text{neg}(Rx[23:16]))$ $Rz[15:8] = \text{Saturate}(\text{neg}(Rx[15:8]))$ $Rz[7:0] = \text{Saturate}(\text{neg}(Rx[7:0]))$
<b>编译结果</b>	仅存在 32 位指令。 pneg.s8.s rz, rx

<b>32 位指令</b>	
<b>操作:</b>	$Rz[31:24] = \text{Saturate}(\text{neg}(Rx[31:24]))$ $Rz[23:16] = \text{Saturate}(\text{neg}(Rx[23:16]))$ $Rz[15:8] = \text{Saturate}(\text{neg}(Rx[15:8]))$ $Rz[7:0] = \text{Saturate}(\text{neg}(Rx[7:0]))$
<b>语法:</b>	pneg.s8.s rz, rx
<b>说明:</b>	以字节为单位, 对 Rx 的字节依次取反, 并做饱和处理。当字节为 0x80 时, 取反饱和处理的结果为 0x7F, 其余情况与取反结果一样。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

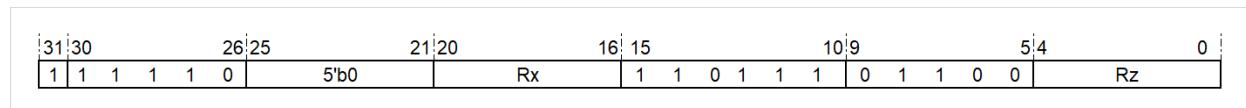


图 15.78: PNEG.S8.S

## 15.79 PNEG.S16.S——带饱和操作的 16 位并行取反指令

<b>统一化指令</b>	
<b>语法</b>	pneg.s16.s rz, rx
<b>操作</b>	Rz[31:16] = Saturate(neg(Rx[31:16])) Rz[15:0] = Saturate(neg(Rx[15:0]))
<b>编译结果</b>	仅存在 32 位指令。 pneg.s16.s rz, rx

<b>说明:</b>	对 Rx 的高低半字依次取反，并做饱和处理。当半字为 0x8000 时，取反饱和处理的结果为 0x7FFF，其余情况与取反结果一样。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	Rz[31:16] = Saturate(neg(Rx[31:16])) Rz[15:0] = Saturate(neg(Rx[15:0]))
<b>语法:</b>	pneg.s16.s rz, rx
<b>说明:</b>	对 Rx 的高低半字依次取反，并做饱和处理。当半字为 0x8000 时，取反饱和处理的结果为 0x7FFF，其余情况与取反结果一样。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

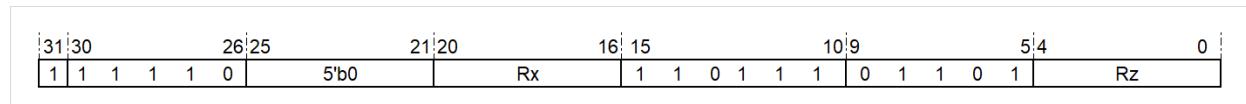


图 15.79: PNEG.S16.S

## 15.80 NEG.S32.S——带饱和操作的取反指令

<b>统一化指令</b>	
<b>语法</b>	neg.s32.s rz, rx
<b>操作</b>	$Rz[31:0] = \text{Saturate}(\text{neg}(Rx[31:0]))$
<b>编译结果</b>	仅存在 32 位指令。 neg.s32.s rz, rx

<b>说明:</b>	对 Rx 取反并做饱和处理。当 Rx 为 0x8000 0000 时，取反饱和处理的结果为 0x7FFF FFFF，其余情况与绝对值结果一样。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	$Rz[31:0] = \text{Saturate}(\text{neg}(Rx[31:0]))$
<b>语法:</b>	neg.s32.s rz, rx
<b>说明:</b>	对 Rx 取反并做饱和处理。当 Rx 为 0x8000 0000 时，取反饱和处理的结果为 0x7FFF FFFF，其余情况与绝对值结果一样。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

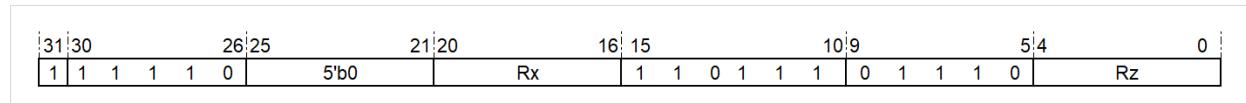


图 15.80: NEG.S32.S

## 15.81 DUP.8——8 位操作数拷贝指令

统 一 化 指 令	
语 法	dup.8 rz, rx, index
操 作	<pre>Element = Rx[7:0] //index= 0 Element = Rx[15:8] //index= 1 Element = Rx[23:16] //index= 2 Element = Rx[31:24] //index= 3 Rz[31:0] = {Element, Element, Element, Element }</pre>
编 译 结 果	仅存在 32 位指令。 dup.8 rz, rx, index

说 明:	定义 Rx[31:24]、Rx[23:16]、Rx[15 :8]、Rx[7:0] 分别为字节 3、字节 2、字节 1、字节 0。由 index 选择相应的字节，并拷贝至 Rx 的 4 个字节中。
影 响 标 志 位:	无影响
限 制:	Index 范围为 0~3
异 常:	无

32 位 指 令	
操 作:	<pre>Element = Rx[7:0] //index= 0 Element = Rx[15:8] //index= 1 Element = Rx[23:16] //index= 2 Element = Rx[31:24] //index= 3 Rz[31:0] = { Element, Element, Element, Element }</pre>
语 法:	dup.8 rz, rx, index
说 明:	定义 Rx[31:24]、Rx[23:16]、Rx[15 :8]、Rx[7:0] 分别为字节 3、字节 2、字节 1、字节 0。由 index 选择相应的字节，并拷贝至 Rx 的 4 个字节中。
影 响 标 志 位:	无影响
限 制:	Index 范围为 0~3
异 常:	无

指令格式：

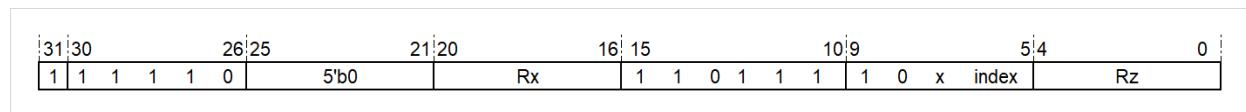


图 15.81: DUP.8

## 15.82 DUP.16——16 位操作数拷贝指令

<b>统一化指令</b>	
<b>语法</b>	dup.16 rz, rx, index
<b>操作</b>	Element = Rx[15:0] //index= 0 Element = Rx[31:16] //index= 1 Rz[31:0] = { Element, Element }
<b>编译结果</b>	仅存在 32 位指令。 dup.16 rz, rx, index

<b>说明:</b>	定义 Rx[31:16]、Rx[15:0] 分别为半字 1、半字 0。由 index 选择相应的半字，并拷贝至 Rx 的 2 个半字中。
<b>影响标志位:</b>	无影响
<b>限制:</b>	Index 范围为 0~1
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	Element = Rx[15:0] //index= 0 Element = Rx[31:16] //index= 1 Rz[31:0] = { Element, Element }
<b>语法:</b>	dup.16 rz, rx, index
<b>说明:</b>	定义 Rx[31:16]、Rx[15:0] 分别为半字 1、半字 0。由 index 选择相应的半字，并拷贝至 Rx 的 2 个半字中。
<b>影响标志位:</b>	无影响
<b>限制:</b>	Index 范围为 0~1
<b>异常:</b>	无

**指令格式:**

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	5'b0	Rx	1 1 0 1 1 1	1 1 x index	Rz	

图 15.82: DUP.16

### 15.83 MULS.(U/S)32——32 位（无/有）符号乘累减指令

统一化指令	
语法	muls.u32 rz, rx, ry muls.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。 muls.u32 rz, rx, ry muls.s32 rz, rx, ry

说明:	Rx 与 Ry 相乘，得到 64 位结果，并与 {Rz+1, Rz} 相减。减法结果的高 32 位存入 Rz+1，低 32 位存入 Rz。
影响标志位:	无影响
限制:	无
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:	muls.u32 rz, rx, ry muls.s32 rz, rx, ry
说明:	Rx 与 Ry 相乘，得到 64 位结果，并与 {Rz+1, Rz} 相减。减法结果的高 32 位存入 Rz+1，低 32 位存入 Rz。
影响标志位:	无影响
限制:	无
异常:	无

指令格式：

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 0	S 0 1 1 0	Rz	

图 15.83: MULS.(US)32

## 15.84 MULA.(U/S)32.S——带饱和操作的 32 位(无/有)符号乘累加指令

统一化指令	
语法	mula.u32.s rz, rx, ry mula.s32.s rz, rx, ry
操作	$\{Rz+1[31:0], Rz[31:0]\} = \text{Saturate}(\{Rz+1[31:0], Rz[31:0]\} + Rx[31:0] \times Ry[31:0])$
编译结果	仅存在 32 位指令。 mula.u32.s rz, rx, ry mula.s32.s rz, rx, ry

说明:	Rx 与 Ry 相乘，得到 64 位结果，并与 {Rz+1, Rz} 相加。加法结果做饱和处理后，高 32 位存入 Rz+1，低 32 位存入 Rz。 对于无符号运算，饱和处理的过程为，若加法结果大于饱和值 0xFFFF FFFF FFFF FFFF，结果为饱和值，其它情况为加法结果本身。 对于有符号运算，饱和处理的过程为，若加法结果大于上饱和值 0x7FFF FFFF FFFF FFFF，结果为上饱和值，若加法结果小于下饱和值 0x8000 0000 0000 0000，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

<b>32 位 指 令</b>	
<b>操 作:</b>	$\{Rz+1[31:0], Rz[31:0]\} = \text{Saturate}(\{Rz+1[31:0], Rz[31:0]\} + Rx[31:0] \times Ry[31:0])$
<b>语 法:</b>	mula.u32.s rz, rx, ry mula.s32.s rz, rx, ry
<b>说 明:</b>	Rx 与 Ry 相乘，得到 64 位结果，并与 {Rz+1, Rz} 相加。加法结果做饱和处理后，高 32 位存入 Rz+1，低 32 位存入 Rz。 对于无符号运算，饱和处理的过程为，若加法结果大于饱和值 0xFFFF FFFF FFFF FFFF，结果为饱和值，其它情况为加法结果本身。 对于有符号运算，饱和处理的过程为，若加法结果大于上饱和值 0x7FFF FFFF FFFF FFFF，结果为上饱和值，若加法结果小于下饱和值 0x8000 0000 0000 0000，结果为下饱和值，其它情况为加法结果本身。
<b>影 响 标 志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

**指令格式:**

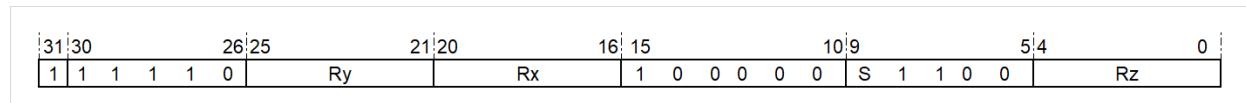


图 15.84: MULA.(US)32.S

## 15.85 MULS.(U/S)32.S——带饱和操作的 32 位（无/有）符号乘累减指令

统一化指令	
语法	muls.u32.s rz, rx, ry muls.s32.s rz, rx, ry
操作	$\{Rz+1[31:0], Rz[31:0]\} = \text{Saturate}(\{Rz+1[31:0], Rz[31:0]\} - Rx[31:0] \times Ry[31:0])$
编译结果	仅存在 32 位指令。 muls.u32.s rz, rx, ry muls.s32.s rz, rx, ry

说明:	Rx 与 Ry 相乘，得到 64 位结果，并与 {Rz+1, Rz} 相减。减法结果做饱和处理后，高 32 位存入 Rz+1，低 32 位存入 Rz。  对于无符号运算，饱和处理的过程为，若减法结果小于饱和值 0x0，结果为 0x0，其它情况为减法结果本身。  对于有符号运算，饱和处理的过程为，若减法结果大于上饱和值 0x7FFF FFFF FFFF FFFF，结果为上饱和值，若减法结果小于下饱和值 0x8000 0000 0000 0000，结果为下饱和值，其它情况为减法结果本身。
影响标志位:	无影响
限制:	无
异常:	无

<b>32 位指令</b>	
<b>操作:</b>	$\{Rz+1[31:0], Rz[31:0]\} = \text{Saturate}(\{Rz+1[31:0], Rz[31:0]\} - Rx[31:0] \times Ry[31:0])$
<b>语法:</b>	muls.u32.s rz, rx, ry muls.s32.s rz, rx, ry
<b>说明:</b>	Rx 与 Ry 相乘，得到 64 位结果，并与 {Rz+1, Rz} 相减。减法结果做饱和处理后，高 32 位存入 Rz+1，低 32 位存入 Rz。 对于无符号运算，饱和处理的过程为，若减法结果小于饱和值 0x0，结果为 0x0，其它情况为减法结果本身。 对于有符号运算，饱和处理的过程为，若减法结果大于上饱和值 0x7FFF FFFF FFFF FFFF，结果为上饱和值，若减法结果小于下饱和值 0x8000 0000 0000 0000，结果为下饱和值，其它情况为减法结果本身。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 0 0	S 1 1 1 0	Rz	

图 15.85: MULS.(US)32.S

## 15.86 MUL.S32.H——32 位有符号乘法取高 32 位指令

<b>统一化指令</b>	
<b>语法</b>	mul.s32.h rz, rx, ry
<b>操作</b>	$Rz[31:0] = \{Rx[31:0] \times Ry[31:0]\}[63:32]$
<b>编译结果</b>	仅存在 32 位指令。 mul.s32.h rz, rx, ry

<b>说明:</b>	Rx 与 Ry 相乘得到 64 位结果，其中高 32 位存入 Rz。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	$Rz[31:0] = \{Rx[31:0] \times Ry[31:0]\}[63:32]$
<b>语法:</b>	mul.s32.h rz, rx, ry
<b>说明:</b>	Rx 与 Ry 相乘得到 64 位结果，其中高 32 位存入 Rz。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

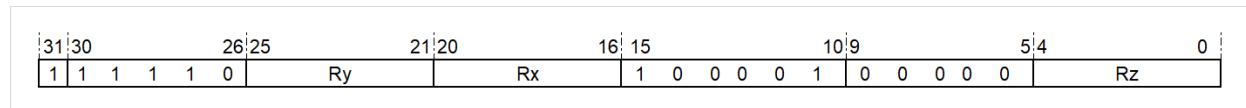


图 15.86: MUL.S32.H

### 15.87 MUL.S32.RH——带舍入操作的 32 位有符号乘法取高 32 位指令

<b>统一化指令</b>	
<b>语法</b>	mul.s32.rh rz, rx, ry
<b>操作</b>	$Rz[31:0] = \{Rx[31:0] \times Ry[31:0] + 32' h8000 0000\}[63:32]$
<b>编译结果</b>	仅存在 32 位指令。 mul.s32.rh rz, rx, ry

<b>说明:</b>	Rx 与 Ry 相乘得到的 64 位结果与 0x8000 0000 相加，截取加法结果的高 32 位存入 Rz。其中，乘法结果加上 0x8000 0000 用以实现舍入操作。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	$Rz[31:0] = \{Rx[31:0] \times Ry[31:0] + 32' h8000 0000\}[63:32]$
<b>语法:</b>	mul.s32.rh rz, rx, ry
<b>说明:</b>	Rx 与 Ry 相乘得到的 64 位结果与 0x8000 0000 相加，截取加法结果的高 32 位存入 Rz。其中，乘法结果加上 0x8000 0000 用以实现舍入操作。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

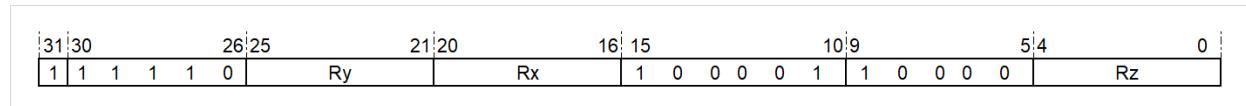


图 15.87: MUL.S32.RH

## 15.88 RMUL.S32.H——32 位有符号小数乘法取高 32 位指令

<b>统一化指令</b>	
<b>语法</b>	rmul.s32.h rz, rx, ry
<b>操作</b>	<pre>if(Rx[31:0] == 32' h8000 0000 &amp;&amp; Ry[31:0] == 32' h8000 0000)     Rz[31:0] = 32' h7FFF FFFF else     Rz[31:0] = {Rx[31:0] X Ry[31:0]}[62:31]</pre>
<b>编译结果</b>	仅存在 32 位指令。 rmul.s32.h rz, rx, ry

<b>说明:</b>	Rx、Ry 视为 32 位有符号小数，其中第 31 位为符号位、第 30~0 位为小数位。当 Rx=0x8000 0000 且 Ry=0x8000 0000 时，结果为 0x7FFF FFFF；其余情况，取 Rx 与 Ry 相乘结果的第 62 到 31 位存入 Rz。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位 指令</b>	
<b>操作:</b>	if(Rx[31:0] == 32' h8000 0000 && Ry[31:0] == 32' h8000 0000) Rz[31:0] = 32' h7FFF FFFF else Rz[31:0] = {Rx[31:0] X Ry[31:0]}[62:31]
<b>语法:</b>	rmul.s32.h rz, rx, ry
<b>说明:</b>	Rx、Ry 视为 32 位有符号小数，其中第 31 位为符号位、第 30~0 位为小数位。当 Rx=0x8000 0000 且 Ry=0x8000 0000 时，结果为 0x7FFF FFFF；其余情况，取 Rx 与 Ry 相乘结果的第 62 到 31 位存入 Rz。
<b>影响 标志 位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

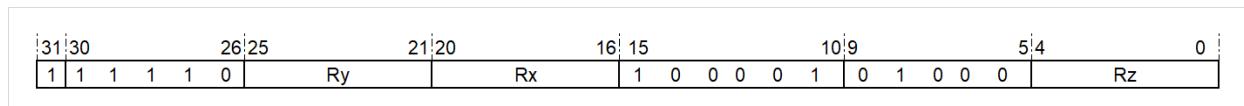


图 15.88: RMUL.S32.H

## 15.89 RMUL.S32.RH——带舍入操作的 32 位有符号小数乘法取高 32 位指令

<b>统 一 化 指 令</b>	
<b>语法</b>	rmul.s32.rh rz, rx, ry
<b>操作</b>	if(Rx[31:0] == 32' h8000 0000 && Ry[31:0] == 32' h8000 0000) Rz[31:0] = 32' h7FFF FFFF else Rz[31:0] = {Rx[31:0] X Ry[31:0] + 32' h4000 0000}[62:31]
<b>编 译 结 果</b>	仅存在 32 位指令。 rmul.s32.rh rz, rx, ry

说 明:	Rx、Ry 视为 32 位有符号小数，其中第 31 位为符号位、第 30~0 位为小数位。当 Rx=0x8000 0000 且 Ry=0x8000 0000 时，结果为 0x7FFF FF FF；其余情况，Rx 与 Ry 相乘得到的 64 位结果与 0x4000 0000 相加，截取加法结果的第 62~31 位存入 Rz。其中，乘法结果加上 0x4000 0000 用以实现小数形式的舍入操作。
影 响 标 志 位:	无影响
限 制:	无
异 常:	无

32 位 指 令	
操 作:	if(Rx[31:0] == 32' h8000 0000 && Ry[31:0] == 32' h8000 0000) Rz[31:0] = 32' h7FFF FFFF else Rz[31:0] = {Rx[31:0] X Ry[31:0] + 32' h4000 0000}[62:31]
语 法:	rmul.s32.rh rz, rx, ry
说 明:	Rx、Ry 视为 32 位有符号小数，其中第 31 位为符号位、第 30~0 位为小数位。当 Rx=0x8000 0000 且 Ry=0x8000 0000 时，结果为 0x7FFF FF FF；其余情况，Rx 与 Ry 相乘得到的 64 位结果与 0x4000 0000 相加，截取加法结果的第 62~31 位存入 Rz。其中，乘法结果加上 0x4000 0000 用以实现小数形式的舍入操作。
影 响 标 志 位:	无影响
限 制:	无
异 常:	无

指令格式：

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 0	1 1 0 0 0	Rz	

图 15.89: RMUL.S32.RH

## 15.90 MULA.S32.HS——带饱和操作的 32 位有符号乘累加取高 32 位指令

统一化指令	
语法	mula.s32.hs rz, rx, ry
操作	$Rz[31:0] = \text{Saturate}( Rz[31:0] + \{Rx[31:0] \times Ry[31:0]\}[63:32] )$
编译结果	仅存在 32 位指令。 mula.s32.hs rz, rx, ry

说 明:	Rx 与 Ry 相乘得到的 64 位结果取高 32 位，并与 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 0x7FFF FFFF，结果为上饱和值，若加法结果小于下饱和值 0x8000 0000，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

<b>32 位 指 令</b>	
<b>操 作:</b>	$Rz[31:0] = \text{Saturate}( Rz[31:0] + \{Rx[31:0] \times Ry[31:0]\}[63:32] )$
<b>语 法:</b>	mula.s32.hs rz, rx, ry
<b>说 明:</b>	Rx 与 Ry 相乘得到的 64 位结果取高 32 位，并与 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 0x7FFF FFFF，结果为上饱和值，若加法结果小于下饱和值 0x8000 0000，结果为下饱和值，其它情况为加法结果本身。
<b>影 响 标 志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

**指令格式:**

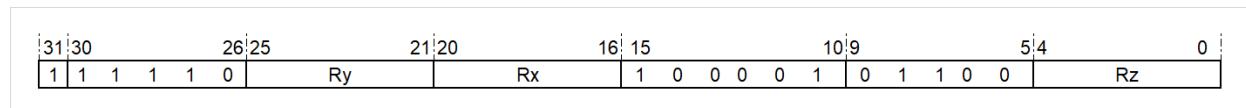


图 15.90: MULA.S32.HS

### 15.91 MULS.S32.HS——带饱和操作的 32 位有符号乘累减取高 32 位 指令

<b>统一化指令</b>	
<b>语法</b>	muls.s32.hs rz, rx, ry
<b>操作</b>	$Rz[31:0] = \text{Saturate}( Rz[31:0] - \{Rx[31:0] \times Ry[31:0]\}[63:32] )$
<b>编译结果</b>	仅存在 32 位指令。 muls.s32.hs rz, rx, ry

说 明:	Rx 与 Ry 相乘得到的 64 位结果取高 32 位，并与 Rz 相减。减法结果做饱和处理后存入 Rz。 饱和处理的过程为，若减法结果大于上饱和值 0x7FFF FFFF，结果为上饱和值，若减法结果小于下饱和值 0x8000 0000，结果为下饱和值，其它情况为减法结果本身。
影 响 标 志 位:	无影响
限 制:	无
异 常:	无

32 位 指 令	
操 作:	$Rz[31:0] = \text{Saturate}( Rz[31:0] - \{Rx[31:0] \times Ry[31:0]\}[63:32] )$
语 法:	mul.s32.hs rz, rx, ry
说 明:	Rx 与 Ry 相乘得到的 64 位结果取高 32 位，并与 Rz 相减。减法结果做饱和处理后存入 Rz。 饱和处理的过程为，若减法结果大于上饱和值 0x7FFF FFFF，结果为上饱和值，若减法结果小于下饱和值 0x8000 0000，结果为下饱和值，其它情况为减法结果本身。
影 响 标 志 位:	无影响
限 制:	无
异 常:	无

#### 指令格式:

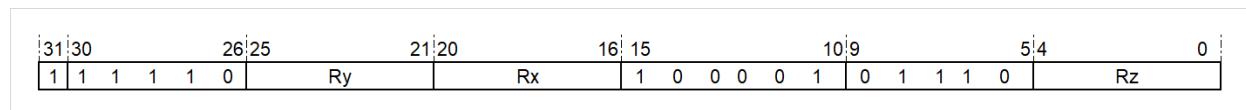


图 15.91: MULS.S32.HS

## 15.92 MULA.S32.RHS——带舍入和饱和操作的 32 位有符号乘累加取高 32 位指令

<b>统一化指令</b>	
<b>语法</b>	mula.s32.rhs rz, rx, ry
<b>操作</b>	$Rz[31:0] = \text{Saturate}(\{Rz[31:0], 32' h0000 0000\} + Rx[31:0] \times Ry[31:0] + 32' h8000 0000)[63:32]$
<b>编译结果</b>	仅存在 32 位指令。 mula.s32.rhs rz, rx, ry

<b>说明:</b>	Rx 与 Ry 相乘得到的 64 位结果，与 {Rz, 32' h0000 0000} 累加，累加结果再加上 0x8000 0000。截取加法结果的高 32 位做饱和处理后存入 Rz。其中，累加结果加上 0x8000 0000 用以实现舍入操作。 饱和处理的过程为，若加法结果大于上饱和值 0x7FFF FFFF，结果为上饱和值，若加法结果小于下饱和值 0x8000 0000，结果为下饱和值，其它情况为加法结果本身。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32位指令</b>	
<b>操作:</b>	$Rz[31:0] = \text{Saturate}(\{Rz[31:0], 32' h0000 0000\} + Rx[31:0] \times Ry[31:0] + 32' h8000 0000)[63:32]$
<b>语法:</b>	mula.s32.rhs Rz, Rx, Ry
<b>说明:</b>	Rx 与 Ry 相乘得到的 64 位结果，与 {Rz, 32' h0000 0000} 累加，累加结果再加上 0x8000 0000。 截取加法结果的高 32 位做饱和处理后存入 Rz。其中，累加结果加上 0x8000 0000 用以实现舍入操作。 饱和处理的过程为，若加法结果大于上饱和值 0x7FFF FFFF，结果为上饱和值，若加法结果小于下饱和值 0x8000 0000，结果为下饱和值，其它情况为加法结果本身。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

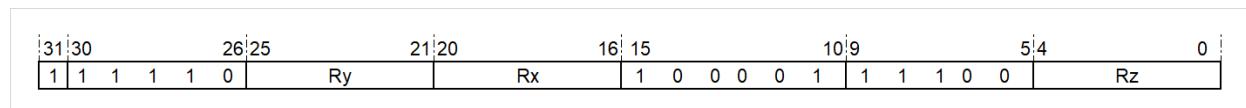


图 15.92: MULA.S32.RHS

## 15.93 MULS.S32.RHS——带舍入和饱和操作的 32 位有符号乘累减取高 32 位指令

<b>统一化指令</b>	
<b>语法</b>	muls.s32.rhs rz, rx, ry
<b>操作</b>	$Rz[31:0] = \text{Saturate}(\{(Rz[31:0], 32' h0000 0000\} - Rx[31:0] \times Ry[31:0] + 32' h8000 0000)[63:32])$
<b>编译结果</b>	仅存在 32 位指令。 muls.s32.rhs rz, rx, ry

<b>说明:</b>	Rx 与 Ry 相乘得到的 64 位结果，与 {Rz, 32' h0000 0000} 累减，累减结果再加上 0x8000 0000。截取加法结果的高 32 位做饱和处理后存入 Rz。其中，累减结果加上 0x8000 0000 用以实现舍入操作。 饱和处理的过程为，若加法结果大于上饱和值 0x7FFF FFFF，结果为上饱和值，若加法结果小于下饱和值 0x8000 0000，结果为下饱和值，其它情况为加法结果本身。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	$Rz[31:0] = \text{Saturate}(\{\{Rz[31:0], 32' h0000 0000\} - Rx[31:0] \times Ry[31:0] + 32' h8000 0000\}[63:32])$
<b>语法:</b>	muls.s32.rhs rz, rx, ry
<b>说明:</b>	Rx 与 Ry 相乘得到的 64 位结果，与 {Rz, 32' h0000 0000} 累减，累减结果再加上 0x8000 0000。截取加法结果的高 32 位做饱和处理后存入 Rz。其中，累减结果加上 0x8000 0000 用以实现舍入操作。 饱和处理的过程为，若加法结果大于上饱和值 0x7FFF FFFF，结果为上饱和值，若加法结果小于下饱和值 0x8000 0000，结果为下饱和值，其它情况为加法结果本身。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 0 1	1 1 1 1 0	Rz	

图 15.93: MUL.S32.RHS

## 15.94 MULXL.S32——32 位有符号低半字不对齐乘指令

<b>统一化指令</b>	
<b>语法</b>	mulxl.s32 Rz, Rx, Ry
<b>操作</b>	$Rz[31:0] = \{Rx[31:0] \times Ry[15:0]\}[47:16]$
<b>编译结果</b>	仅存在 32 位指令。 mulxl.s32 Rz, Rx, Ry

<b>说明:</b>	Rx 与 Ry 的低半字相乘，取结果的高 32 位存入 Rz。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	$Rz[31:0] = \{Rx[31:0] X Ry[15:0]\}[47:16]$
<b>语法:</b>	mulxl.s32 Rz, Rx, Ry
<b>说明:</b>	Rx 与 Ry 的低半字相乘，取结果的高 32 位存入 Rz。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

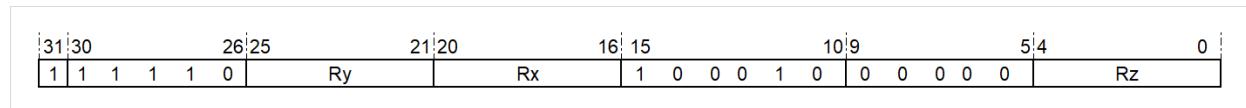


图 15.94: MULXL.S32

## 15.95 MULXL.S32.R——带舍入操作的 32 位有符号低半字不对齐乘指令

<b>统一化指令</b>	
<b>语法</b>	mulxl.s32.r rz, rx, ry
<b>操作</b>	$Rz[31:0] = \{Rx[31:0] X Ry[15:0] + 16' h8000\}[47:16]$
<b>编译结果</b>	仅存在 32 位指令。 mulxl.s32.r rz, rx, ry

<b>说明:</b>	Rx 与 Ry 的低半字相乘，并加上 0x8000，取结果的高 32 位存入 Rz。乘法结果加上 0x8000 用以实现舍入操作。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令 令</b>	
<b>操作:</b>	$Rz[31:0] = \{Rx[31:0] X Ry[15:0] + 16' h8000\}[47:16]$
<b>语法:</b>	mulxl.s32.r rz, rx, ry
<b>说明:</b>	Rx 与 Ry 的低半字相乘，并加上 0x8000，取结果的高 32 位存入 Rz。乘法结果加上 0x8000 用以实现舍入操作。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 0	1 0 0 0 0	Rz	

图 15.95: MULXL.S32.R

## 15.96 MULXH.S32——32 位有符号高半字不对齐乘指令

<b>统一化指令</b>	
<b>语法</b>	mulxh.s32 rz, rx, ry
<b>操作</b>	$Rz[31:0] = \{Rx[31:0] X Ry[31:16]\}[47:16]$
<b>编译结果</b>	仅存在 32 位指令。 mulxh.s32 rz, rx, ry

<b>说明:</b>	Rx 与 Ry 的高半字相乘，取结果的高 32 位存入 Rz。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	$Rz[31:0] = \{Rx[31:0] X Ry[31:16]\}[47:16]$
<b>语法:</b>	mulxh.s32 rz, rx, ry
<b>说明:</b>	Rx 与 Ry 的高半字相乘，取结果的高 32 位存入 Rz。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式：**

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 1	0 0 0 0 0	Rz	

图 15.96: MULXH.S32

### 15.97 MULXH.S32.R——带舍入操作的 32 位有符号高半字不对齐乘指令

<b>统一化指令</b>	
<b>语法</b>	mulxh.s32.r rz, rx, ry
<b>操作</b>	$Rz[31:0] = \{Rx[31:0] \times Ry[31:16] + 16' h8000\}[47:16]$
<b>编译结果</b>	仅存在 32 位指令。 mulxh.s32.r rz, rx, ry

<b>说明:</b>	Rx 与 Ry 的高半字相乘，并加上 0x8000，取结果的高 32 位存入 Rz。乘法结果加上 0x8000 用以实现舍入操作。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	$Rz[31:0] = \{Rx[31:0] \times Ry[31:16] + 16' h8000\}[47:16]$
<b>语法:</b>	mulxh.s32.r rz, rx, ry
<b>说明:</b>	Rx 与 Ry 的高半字相乘，并加上 0x8000，取结果的高 32 位存入 Rz。乘法结果加上 0x8000 用以实现舍入操作。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式：**

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 1	1 0 0 0 0	Rz	

图 15.97: MULXH.S32.R

## 15.98 RMULXL.S32——32 位有符号低半字不对齐小数乘指令

统一化 指令	
语法	rmulxl.s32 rz, rx, ry
操作	if(Rx[31:0] == 32' h8000 0000 && Ry[15:0] == 16' h8000) Rz[31:0] = 32' h7FFF FFFF else Rz[31:0] = {Rx[31:0] X Ry[15:0]}[46:15]
编译结 果	仅存在 32 位指令。 rmulxl.s32 rz, rx, ry

说 明:	Rx、Ry 的低半字视为有符号小数，其中 Rx 第 31 位为符号位、第 30~0 位为小数位，Ry 第 15 位为符号位、第 14~0 位为小数位。当 Rx[31:0]=0x8000 0000 且 Ry[15:0]=0x8000 时，结果为 0x7FFF FFFF；其余情况，取 Rx 与 Ry 相乘结果的第 46 到 15 位存入 Rz。
影 响 标 志 位:	无影响
限 制:	无
异 常:	无

<b>32 位 指 令</b>	
<b>操 作:</b>	if(Rx[31:0] == 32' h8000 0000 && Ry[15:0] == 16' h8000) Rz[31:0] = 32' h7FFF FFFF else Rz[31:0] = {Rx[31:0] X Ry[15:0]}[46:15]
<b>语 法:</b>	rmulxl.s32 rz, rx, ry
<b>说 明:</b>	Rx、Ry 的低半字视为有符号小数, 其中 Rx 第 31 位为符号位、第 30~0 位为小数位, Ry 第 15 位为符号位、第 14~0 位为小数位。当 Rx[31:0]=0x8000 0000 且 Ry[15:0]=0x8000 时, 结果为 0x7FFF FFFF; 其余情况, 取 Rx 与 Ry 相乘结果的第 46 到 15 位存入 Rz。
<b>影 响 标 志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

**指令格式:**

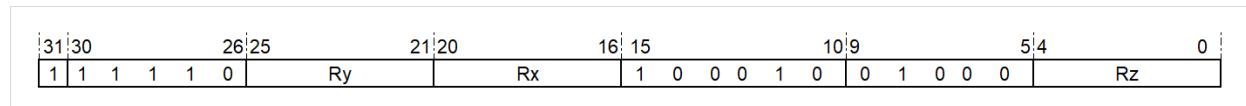


图 15.98: RMULXL.S32

## 15.99 RMULXL.S32.R——带舍入操作的 32 位有符号低半字不对齐小数乘指令

<b>统一化指令</b>	
<b>语法</b>	rmulxl.s32.r rz, rx, ry
<b>操作</b>	<pre>if(Rx[31:0] == 32' h8000 0000 &amp;&amp; Ry[15:0] == 16' h8000) Rz[31:0] = 32' h7FFF FFFF else Rz[31:0] = {Rx[31:0] X Ry[15:0] + 16' h4000}[46:15]</pre>
<b>编译结果</b>	仅存在 32 位指令。 rmulxl.s32.r rz, rx, ry

<b>说明:</b>	Rx、Ry 的低半字视为有符号小数，其中 Rx 第 31 位为符号位、第 30~0 位为小数位，Ry 第 15 位为符号位、第 14~0 位为小数位。当 Rx[31:0]=0x8000 0000 且 Ry[15:0]=0x8000 时，结果为 0x7FFF FFFF；其余情况，Rx 与 Ry 的低半字相乘，并加上 0x4000，取结果的第 46 到 15 位存入 Rz。乘法结果加上 0x4000 用以实现小数形式的舍入操作。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位 指 令</b>	
<b>操 作:</b>	if(Rx[31:0] == 32' h8000 0000 && Ry[15:0] == 16' h8000) Rz[31:0] = 32' h7FFF FFFF else Rz[31:0] = {Rx[31:0] X Ry[15:0] + 16' h4000}[46:15]
<b>语 法:</b>	rmulxl.s32.r rz, rx, ry
<b>说 明:</b>	Rx、Ry 的低半字视为有符号小数，其中 Rx 第 31 位为符号位、第 30~0 位为小数位，Ry 第 15 位为符号位、第 14~0 位为小数位。当 Rx[31:0]=0x8000 0000 且 Ry[15:0]=0x8000 时，结果为 0x7FFF FFFF；其余情况，Rx 与 Ry 的低半字相乘，并加上 0x4000，取结果的第 46 到 15 位存入 Rz。乘法结果加上 0x4000 用以实现小数形式的舍入操作。
<b>影 响 标 志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

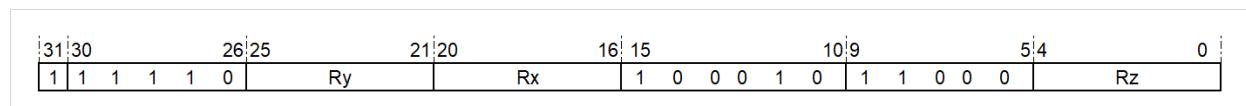
**指令格式:**

图 15.99: RMULXL.S32.R

## 15.100 RMULXH.S32——32 位有符号高半字不对齐小数乘指令

<b>统一化指令</b>	
<b>语法</b>	rmulxh.s32 rz, rx, ry
<b>操作</b>	<pre>if(Rx[31:0] == 32' h8000 0000 &amp;&amp; Ry[31:16] == 16' h8000) Rz[31:0] = 32' h7FFF FFFF else Rz[31:0] = {Rx[31:0] X Ry[31:16]}[46:15]</pre>
<b>编译结果</b>	仅存在 32 位指令。 rmulxh.s32 rz, rx, ry

<b>说 明:</b>	Rx、Ry 的低半字视为有符号小数，其中 Rx 第 31 位为符号位、第 30~0 位为小数位，Ry 第 31 位为符号位、第 30~16 位为小数位。当 Rx[31:0]=0x8000 0000 且 Ry[31:16]=0x8000 时，结果为 0x7FFF FFFF；其余情况，取 Rx 与 Ry 相乘结果的第 46 到 15 位存入 Rz。
<b>影 响 标 志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

<b>32 位 指 令</b>	
<b>操 作:</b>	if(Rx[31:0] == 32' h8000 0000 && Ry[31:16] == 16' h8000) Rz[31:0] = 32' h7FFF FFFF else Rz[31:0] = {Rx[31:0] X Ry[31:16]}[46:15]
<b>语 法:</b>	rmulxh.s32 rz, rx, ry
<b>说 明:</b>	Rx、Ry 的低半字视为有符号小数，其中 Rx 第 31 位为符号位、第 30~0 位为小数位，Ry 第 31 位为符号位、第 30~16 位为小数位。当 Rx[31:0]=0x8000 0000 且 Ry[31:16]=0x8000 时，结果为 0x7FFF FFFF；其余情况，取 Rx 与 Ry 相乘结果的第 46 到 15 位存入 Rz。
<b>影 响 标 志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

**指令格式:**

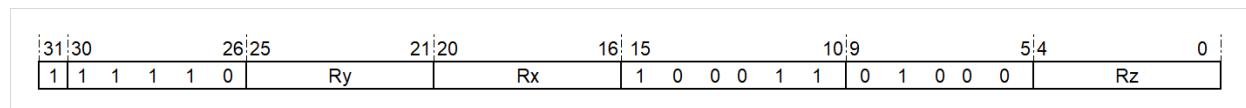


图 15.100: RMULXH.S32

## 15.101 RMULXH.S32.R——带舍入操作的 32 位有符号高半字不对齐小数乘指令

<b>统一化指令</b>	
<b>语法</b>	rmulxh.s32.r rz, rx, ry
<b>操作</b>	<pre>if(Rx[31:0] == 32' h8000 0000 &amp;&amp; Ry[31:16] == 16' h8000) Rz[31:0] = 32' h7FFF FFFF else Rz[31:0] = {Rx[31:0] X Ry[31:16] + 16' h4000}[46:15]</pre>
<b>编译结果</b>	仅存在 32 位指令。 rmulxh.s32.r rz, rx, ry

<b>说明:</b>	Rx、Ry 的低半字视为有符号小数，其中 Rx 第 31 位为符号位、第 30~0 位为小数位，Ry 第 15 位为符号位、第 14~0 位为小数位。当 Rx[31:0]=0x8000 0000 且 Ry[15:0]=0x8000 时，结果为 0x7FFF FFFF；其余情况，Rx 与 Ry 的低半字相乘，并加上 0x4000，取结果的第 46 到 15 位存入 Rz。乘法结果加上 0x4000 用以实现小数形式的舍入操作。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32位指令</b>	
<b>操作:</b>	if(Rx[31:0] == 32' h8000 0000 && Ry[31:16] == 16' h8000) Rz[31:0] = 32' h7FFF FFFF else Rz[31:0] = {Rx[31:0] X Ry[31:16] + 16' h4000}[46:15]
<b>语法:</b>	rmulxh.s32.r rz, rx, ry
<b>说明:</b>	Rx、Ry 的低半字视为有符号小数，其中 Rx 第 31 位为符号位、第 30~0 位为小数位，Ry 第 15 位为符号位、第 14~0 位为小数位。当 Rx[31:0]=0x8000 0000 且 Ry[15:0]=0x8000 时，结果为 0x7FFF FFFF；其余情况，Rx 与 Ry 的低半字相乘，并加上 0x4000，取结果的第 46 到 15 位存入 Rz。乘法结果加上 0x4000 用以实现小数形式的舍入操作。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式：**

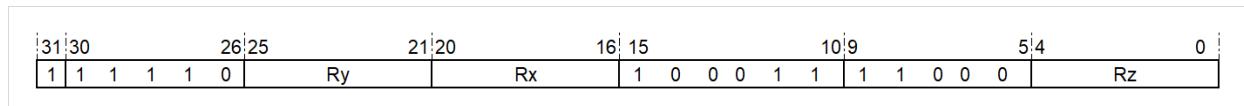


图 15.101: RMULXH.S32.R

## 15.102 MULAXL.S32.S——带饱和操作的 32 位有符号低半字不对齐乘累加指令

<b>统一化指令</b>	
<b>语法</b>	mulaxl.s32.s rz, rx, ry
<b>操作</b>	$Rz[31:0] = \text{Saturate}( Rz[31:0] + \{Rx[31:0] X Ry[15:0]\}[47:16] )$
<b>编译结果</b>	仅存在 32 位指令。 mulaxl.s32.s rz, rx, ry

<b>说 明:</b>	Rx 与 Ry 的低半字相乘并取结果的高 32 位与 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 0x7FFF FFFF，结果为上饱和值，若加法结果小于下饱和值 0x8000 0000，结果为下饱和值，其它情况为加法结果本身。
<b>影响 标志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

<b>32 位指 令</b>	
<b>操 作:</b>	$Rz[31:0] = \text{Saturate}( Rz[31:0] + \{Rx[31:0] \times Ry[15:0]\}[47:16] )$
<b>语 法:</b>	mulaxl.s32.s rz, rx, ry
<b>说 明:</b>	Rx 与 Ry 的低半字相乘并取结果的高 32 位与 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 0x7FFF FFFF，结果为上饱和值，若加法结果小于下饱和值 0x8000 0000，结果为下饱和值，其它情况为加法结果本身。
<b>影响 标志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

#### 指令格式：

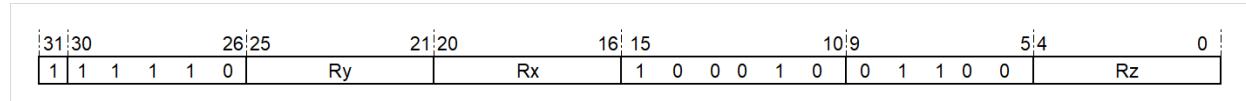


图 15.102: MULAXL.S32.S

## 15.103 MULAXL.S32.RS——带舍入和饱和操作的 32 位有符号低半字不对齐乘累加指令

<b>统一化指令</b>	
<b>语法</b>	mulaxl.s32.rs rz, rx, ry
<b>操作</b>	$Rz[31:0] = \text{Saturate}((\{Rz[31:0], 16' h0000\} + Rx[31:0] \times Ry[15:0] + 16' h8000)[47:16])$
<b>编译结果</b>	仅存在 32 位指令。 mulaxl.s32.rs rz, rx, ry

<b>说 明:</b>	Rx 与 Ry 的低半字相乘，并与 {Rz, 16' h0000} 累加，累加结果加上 0x8000。取结果的高 32 位并做饱和处理后存入 Rz 中。累加结果加上 0x8000 用以实现舍入操作。 饱和处理的过程为，若加法结果大于上饱和值 0x7FFF FFFF，结果为上饱和值，若加法结果小于下饱和值 0x8000 0000，结果为下饱和值，其它情况为加法结果本身。
<b>影 响 标 志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

32 位 指 令	
操作:	$Rz[31:0] = \text{Saturate}(\{\{Rz[31:0], 16' h0000\} + Rx[31:0] \times Ry[15:0] + 16' h8000\}[47:16])$
语法:	mulaxl.s32.rs rz, rx, ry
说明:	Rx 与 Ry 的低半字相乘，并与 {Rz, 16' h0000} 累加，累加结果加上 0x8000。取结果的高 32 位并做饱和处理后存入 Rz 中。累加结果加上 0x8000 用以实现舍入操作。 饱和处理的过程为，若加法结果大于上饱和值 0xFFFF FFFF，结果为上饱和值，若加法结果小于下饱和值 0x8000 0000，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

指令格式:

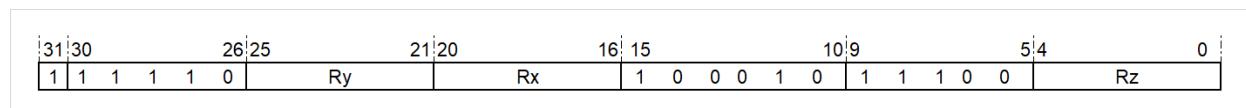


图 15.103: MULAXL.S32.RS

### 15.104 MULAXH.S32.S——带饱和操作的 32 位有符号高半字不对齐乘累加指令

统一化指令	
语法	mulaxh.s32.s rz, rx, ry
操作	$Rz[31:0] = \text{Saturate}(Rz[31:0] + \{Rx[31:0] \times Ry[31:16]\}[47:16])$
编译结果	仅存在 32 位指令。 mulaxh.s32.s rz, rx, ry

<b>说 明:</b>	Rx 与 Ry 的高半字相乘并取结果的高 32 位与 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 0x7FFF FFFF，结果为上饱和值，若加法结果小于下饱和值 0x8000 0000，结果为下饱和值，其它情况为加法结果本身。
<b>影响 标志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

<b>32 位指 令</b>	
<b>操 作:</b>	$Rz[31:0] = \text{Saturate}( Rz[31:0] + \{Rx[31:0] \times Ry[31:16]\}[47:16] )$
<b>语 法:</b>	mulaxh.s32.s rz, rx, ry
<b>说 明:</b>	Rx 与 Ry 的高半字相乘并取结果的高 32 位与 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 0x7FFF FFFF，结果为上饱和值，若加法结果小于下饱和值 0x8000 0000，结果为下饱和值，其它情况为加法结果本身。
<b>影响 标志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

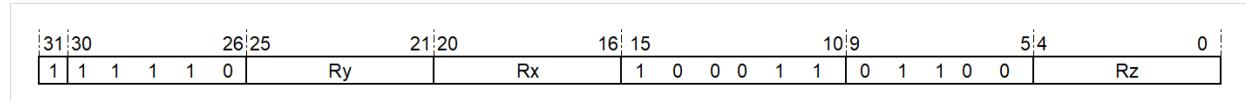
**指令格式：**

图 15.104: MULAXH.S32.S

## 15.105 MULAXH.S32.RS——带舍入和饱和操作的 32 位有符号高半字不对齐乘累加指令

<b>统一化指令</b>	
<b>语法</b>	mulaxh.s32.rs rz, rx, ry
<b>操作</b>	$Rz[31:0] = \text{Saturate}(\{\{Rz[31:0], 16' h0000\} + Rx[31:0] \times Ry[31:16] + 16' h8000\}[47:16])$
<b>编译结果</b>	仅存在 32 位指令。 mulaxh.s32.rs rz, rx, ry

<b>说明:</b>	Rx 与 Ry 的高半字相乘，并与 {Rz, 16' h0000} 累加，累加结果加上 0x8000。取结果的高 32 位并做饱和处理后存入 Rz 中。累加结果加上 0x8000 用以实现舍入操作。 饱和处理的过程为，若加法结果大于上饱和值 0x7FFF FFFF，结果为上饱和值，若加法结果小于下饱和值 0x8000 0000，结果为下饱和值，其它情况为加法结果本身。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位 指 令</b>	
<b>操 作:</b>	$Rz[31:0] = \text{Saturate}(\{\{Rz[31:0], 16' h0000\} + Rx[31:0] \times Ry[31:16] + 16' h8000\}[47:16])$
<b>语 法:</b>	mulaxh.s32.rs rz, rx, ry
<b>说 明:</b>	Rx 与 Ry 的高半字相乘，并与 {Rz, 16' h0000} 累加，累加结果加上 0x8000。取结果的高 32 位并做饱和处理后存入 Rz 中。累加结果加上 0x8000 用以实现舍入操作。 饱和处理的过程为，若加法结果大于上饱和值 0xFFFF FFFF，结果为上饱和值，若加法结果小于下饱和值 0x8000 0000，结果为下饱和值，其它情况为加法结果本身。
<b>影 响 标 志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

**指令格式:**

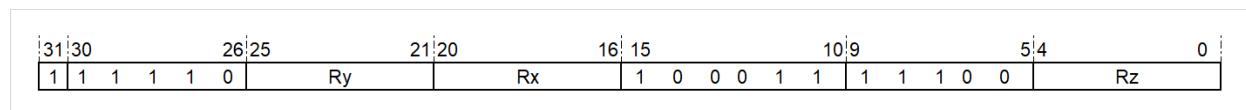


图 15.105: MULAXH.S32.RS

### 15.106 MULLL.S16——16 位有符号低半字乘法指令

<b>统一化指令</b>	
<b>语法</b>	mulll.s16 rz, rx, ry
<b>操作</b>	$Rz[31:0] = Rx[15:0] \times Ry[15:0]$
<b>编译结果</b>	仅存在 32 位指令。 mulll.s16 rz, rx, ry

<b>说明:</b>	Rx 的低半字与 Ry 的低半字相乘，结果存入 Rz。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	$Rz[31:0] = Rx[15:0] \times Ry[15:0]$
<b>语法:</b>	mulll.s16 rz, rx, ry
<b>说明:</b>	Rx 的低半字与 Ry 的低半字相乘，结果存入 Rz。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

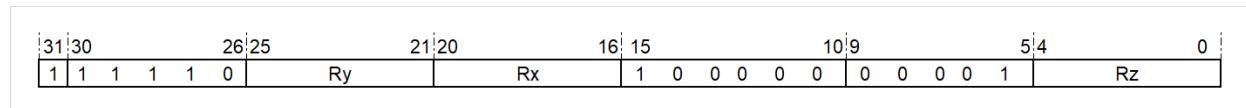


图 15.106: MULLL.S16

### 15.107 MULHH.S16——16 位有符号高半字乘法指令

<b>统一化指令</b>	
<b>语法</b>	mulhh.s16 rz, rx, ry
<b>操作</b>	$Rz[31:0] = Rx[31:16] \times Ry[31:16]$
<b>编译结果</b>	仅存在 32 位指令。 mulhh.s16 rz, rx, ry

<b>说明:</b>	Rx 的高半字与 Ry 的高半字相乘，结果存入 Rz。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

32 位指令	
<b>操作:</b>	Rz[31:0] = Rx[31:16] X Ry[31:16]
<b>语法:</b>	mulhh.s16 rz, rx, ry
<b>说明:</b>	Rx 的高半字与 Ry 的高半字相乘，结果存入 Rz。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

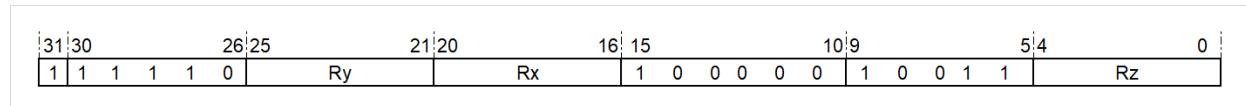


图 15.107: MULHH.S16

### 15.108 MULHL.S16——16 位有符号高低半字乘法指令

统一化指令	
<b>语法</b>	mulhl.s16 rz, rx, ry
<b>操作</b>	Rz[31:0] = Rx[31:16] X Ry[15:0]
<b>编译结果</b>	仅存在 32 位指令。 mulhl.s16 rz, rx, ry

<b>说明:</b>	Rx 的高半字与 Ry 的低半字相乘，结果存入 Rz。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

32 位指令	
<b>操作:</b>	Rz[31:0] = Rx[31:16] X Ry[15:0]
<b>语法:</b>	mulhl.s16 rz, rx, ry
<b>说明:</b>	Rx 的高半字与 Ry 的低半字相乘，结果存入 Rz。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 0	1 0 0 0 1	Rz	

图 15.108: MULHL.S16

### 15.109 RMULLL.S16——16 位有符号低半字小数乘法指令

统一化 指令	
语法	rmulll.s16 rz, rx, ry
操作	if(Rx[15:0] == 16' h8000 && Ry[15:0] == 16' h8000) Rz[31:0] = 32' h7FFF FFFF else Rz[31:0] ={Rx[15:0] X Ry[15:0]} << 1
编译结果	仅存在 32 位指令。 rmulll.s16 rz, rx, ry

说明:	Rx 与 Ry 的低半字视为有符号小数，其中第 15 位为符号位、第 14~0 位为小数位。当 Rx [15:0]=0x8000 且 Ry[15:0]=0x8000 时，结果为 0x7FFF FF FF；其余情况，Rx 与 Ry 相乘结果左移一位后存入 Rz。
影响标志位:	无影响
限制:	无
异常:	无

<b>32 位 指令</b>	
<b>操作:</b>	if(Rx[15:0] == 16' h8000 && Ry[15:0] == 16' h8000) Rz[31:0] = 32' h7FFF FFFF else Rz[31:0] ={Rx[15:0] X Ry[15:0]} << 1
<b>语法:</b>	rmulll.s16 rz, rx, ry
<b>说明:</b>	Rx 与 Ry 的低半字视为有符号小数，其中第 15 位为符号位、第 14~0 位为小数位。当 Rx [15:0]=0x8000 且 Ry[15:0]=0x8000 时，结果为 0x7FFF FF FF；其余情况，Rx 与 Ry 相乘结果左移一位后存入 Rz。
<b>影响 标志 位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

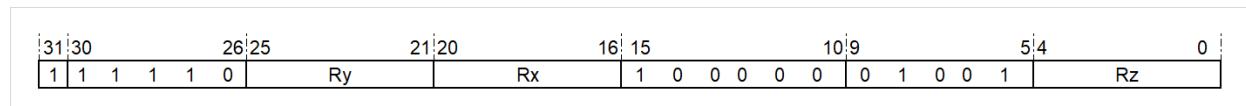


图 15.109: RMULLL.S16

### 15.110 RMULHH.S16——16 位有符号高半字小数乘法指令

<b>统一化 指令</b>	
<b>语法</b>	rmulhh.s16 rz, rx, ry
<b>操作</b>	if(Rx[31:16] == 16' h8000 && Ry[31:16] == 16' h8000) Rz[31:0] = 32' h7FFF FFFF else Rz[31:0] ={Rx[31:16] X Ry[31:16]} << 1
<b>编译结 果</b>	仅存在 32 位指令。 rmulhh.s16 rz, rx, ry

<b>说明:</b>	Rx 与 Ry 的高半字视为有符号小数，其中第 31 位为符号位、第 30~16 位为小数位。当 Rx[31:16]=0x8000 且 Ry[31:16]=0x8000 时，结果为 0x7FFF FF FF；其余情况，Rx 与 Ry 相乘结果左移一位后存入 Rz。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	if(Rx[31:16] == 16' h8000 && Ry[31:16] == 16' h8000) Rz[31:0] = 32' h7FFF FFFF else Rz[31:0] ={Rx[31:16] X Ry[31:16]} << 1
<b>语法:</b>	rmulhh.s16 rz, rx, ry
<b>说明:</b>	Rx 与 Ry 的高半字视为有符号小数，其中第 31 位为符号位、第 30~16 位为小数位。当 Rx[31:16]=0x8000 且 Ry[31:16]=0x8000 时，结果为 0x7FFF FF FF；其余情况，Rx 与 Ry 相乘结果左移一位后存入 Rz。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

31 30	26 25	21 20	16 15	10 9	5 4	0
1   1 1 1 1 0   Ry   Rx   1 0 0 0 0 0   1 1 0 1 1   Rz						

图 15.110: RMULHH.S16

### 15.111 RMULHL.S16——16 位有符号高低半字小数乘法指令

<b>统一化指令</b>	
<b>语法</b>	rmulhl.s16 rz, rx, ry
<b>操作</b>	<pre>if(Rx[31:16] == 16' h8000 &amp;&amp; Ry[15:0] == 16' h8000) Rz[31:0] = 32' h7FFF FFFF else Rz[31:0] ={Rx[31:16] X Ry[15:0]} &lt;&lt; 1</pre>
<b>编译结果</b>	仅存在 32 位指令。 rmulhl.s16 rz, rx, ry

<b>说 明:</b>	Rx 的高半字与 Ry 的低半字视为有符号小数，其中 Rx 的第 31 位为符号位、第 30~16 位为小数位，Ry 的第 15 位为符号位、第 14~0 位为小数位。当 Rx[ 31:16]=0x8000 且 Ry[15:0]=0x8000 时，结果为 0x7FFF FF FF；其余情况，Rx 与 Ry 相乘结果左移一位后存入 Rz。
<b>影 响 标 志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

<b>32 位 指 令</b>	
<b>操 作:</b>	if( $Rx[31:16] == 16' h8000 \&& Ry[15:0] == 16' h8000$ ) $Rz[31:0] = 32' h7FFF FFFF$ else $Rz[31:0] = \{Rx[31:16] X Ry[15:0]\} << 1$
<b>语 法:</b>	rmulhl.s16 rz, rx, ry
<b>说 明:</b>	Rx 的高半字与 Ry 的低半字视为有符号小数，其中 Rx 的第 31 位为符号位、第 30~16 位为小数位，Ry 的第 15 位为符号位、第 14~0 位为小数位。当 Rx[31:16]=0x8000 且 Ry[15:0]=0x8000 时，结果为 0x7FFF FF FF；其余情况，Rx 与 Ry 相乘结果左移一位后存入 Rz。
<b>影 响 标 志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

**指令格式:**

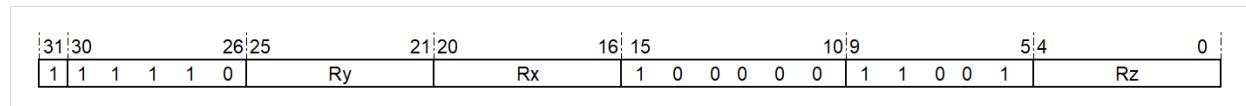


图 15.111: RMULHHL.S16

## 15.112 MULAHHL.S16.S——带饱和操作的 16 位有符号高半字乘累加指令

<b>统一化指令</b>	
<b>语法</b>	mulahh.s16.s rz, rx, ry
<b>操作</b>	$Rz[31:0] = \text{Saturate}(Rz[31:0] + Rx[31:16] X Ry[31:16])$
<b>编译结果</b>	仅存在 32 位指令。 mulahh.s16.s rz, rx, ry

<b>说 明:</b>	Rx 的高半字与 Ry 的高半字相乘，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 0x7FFF FFFF，结果为上饱和值，若加法结果小于下饱和值 0x8000 0000，结果为下饱和值，其它情况为加法结果本身。
<b>影响 标志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

<b>32 位指 令</b>	
<b>操 作:</b>	$Rz[31:0] = \text{Saturate}(Rz[31:0] + Rx[31:16] \times Ry[31:16])$
<b>语 法:</b>	mulahh.s16.s rz, rx, ry
<b>说 明:</b>	Rx 的高半字与 Ry 的高半字相乘，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 0x7FFF FFFF，结果为上饱和值，若加法结果小于下饱和值 0x8000 0000，结果为下饱和值，其它情况为加法结果本身。
<b>影响 标志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

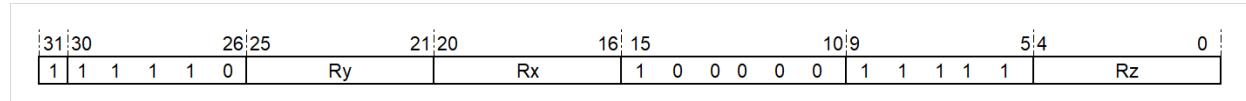
**指令格式:**

图 15.112: MULAHH.S16.S

## 15.113 MULAH.L.S16.S——带饱和操作的 16 位有符号高低半字乘累加指令

<b>统一化指令</b>	
<b>语法</b>	mulahl.s16.s rz, rx, ry
<b>操作</b>	$Rz[31:0] = \text{Saturate}(Rz[31:0] + Rx[31:16] \times Ry[15:0])$
<b>编译结果</b>	仅存在 32 位指令。 mulahl.s16.s Rz, Rx, Ry

<b>说 明:</b>	Rx 的高半字与 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 0x7FFF FFFF，结果为上饱和值，若加法结果小于下饱和值 0x8000 0000，结果为下饱和值，其它情况为加法结果本身。
<b>影 响 标 志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无
<b>指 令 格 式:</b>	

<b>32 位指令</b>	
<b>操作:</b>	$Rz[31:0] = \text{Saturate}(Rz[31:0] + Rx[31:16] \times Ry[15:0])$
<b>语法:</b>	mulahl.s16.s Rz, Rx, Ry
<b>说明:</b>	Rx 的高半字与 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 0x7FFF FFFF，结果为上饱和值，若加法结果小于下饱和值 0x8000 0000，结果为下饱和值，其它情况为加法结果本身。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 0 0	1 1 1 0 1	Rz	

图 15.113: MULAHL.S16.S

### 15.114 MULALL.S16.E——带扩展操作的 16 位有符号低半字乘累加指令

<b>统一化指令</b>	
<b>语法</b>	mulall.s16.e rz, rx, ry
<b>操作</b>	$\{Rz+1[31:0],Rz[31:0]\} = \{Rz+1[31:0],Rz[31:0]\} + Rx[15:0] \times Ry[15:0]$
<b>编译结果</b>	仅存在 32 位指令。 mulall.s16.e rz, rx, ry

<b>说明:</b>	Rx 的低半字与 Ry 的低半字相乘, 乘法结果与 {Rz+1, Rz} 相加, 结果的高 32 位存入 Rz+1, 低 32 位存入 Rz。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	$\{Rz+1[31:0], Rz[31:0]\} = \{Rz+1[31:0], Rz[31:0]\} + Rx[15:0] \times Ry[15:0]$
<b>语法:</b>	mulall.s16.e rz, rx, ry
<b>说明:</b>	Rx 的低半字与 Ry 的低半字相乘, 乘法结果与 {Rz+1, Rz} 相加, 结果的高 32 位存入 Rz+1, 低 32 位存入 Rz。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 0 0	0 0 1 0 1	Rz	

图 15.114: MULALL.S16.E

### 15.115 MULAHH.S16.E——带扩展操作的 16 位有符号高半字乘累加指令

<b>统一化指令</b>	
<b>语法</b>	mulahh.s16.e rz, rx, ry
<b>操作</b>	$\{Rz+1[31:0], Rz[31:0]\} = \{Rz+1[31:0], Rz[31:0]\} + Rx[31:16] \times Ry[31:16]$
<b>编译结果</b>	仅存在 32 位指令。 mulahh.s16.e rz, rx, ry

<b>说明:</b>	Rx 的高半字与 Ry 的高半字相乘, 乘法结果与 {Rz+1, Rz} 相加, 结果的高 32 位存入 Rz+1, 低 32 位存入 Rz。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	$\{Rz+1[31:0], Rz[31:0]\} = \{Rz+1[31:0], Rz[31:0]\} + Rx[31:16] \times Ry[31:16]$
<b>语法:</b>	mulahh.s16.e rz, rx, ry
<b>说明:</b>	Rx 的高半字与 Ry 的高半字相乘, 乘法结果与 {Rz+1, Rz} 相加, 结果的高 32 位存入 Rz+1, 低 32 位存入 Rz。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 1 1 1 0	Ry	Rx	1 0 0 0 0 0	1 0 1 1 1	Rz	

图 15.115: MULAHH.S16.E

## 15.116 MULAHL.S16.E——带扩展操作的 16 位有符号高低半字乘累加指令

<b>统一化指令</b>	
<b>语法</b>	mulahl.s16.e rz, rx, ry
<b>操作</b>	$\{Rz+1[31:0], Rz[31:0]\} = \{Rz+1[31:0], Rz[31:0]\} + Rx[31:16] \times Ry[15:0]$
<b>编译结果</b>	仅存在 32 位指令。 mulahl.s16.e rz, rx, ry

<b>说明:</b>	Rx 的高半字与 Ry 的低半字相乘, 乘法结果与 {Rz+1, Rz} 相加, 结果的高 32 位存入 Rz+1, 低 32 位存入 Rz。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	{Rz+1[31:0],Rz[31:0]} = {Rz+1[31:0],Rz[31:0]} + Rx[31:16] X Ry[15:0]
<b>语法:</b>	mulahl.s16.e rz, rx, ry
<b>说明:</b>	Rx 的高半字与 Ry 的低半字相乘, 乘法结果与 {Rz+1, Rz} 相加, 结果的高 32 位存入 Rz+1, 低 32 位存入 Rz。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 0 0	0 0 1 1 1	Rz	

图 15.116: MULAHLS16.E

### 15.117 PMUL.(U/S)16——16 位并行（无/有）符号乘法指令

<b>统一化指令</b>	
<b>语法</b>	pmul.u16 rz, rx, ry pmul.s16 rz, rx, ry
<b>操作</b>	$Rz+1[31:0] = Rx[31:16] \times Ry[31:16]$ $Rz[31:0] = Rx[15:0] \times Ry[15:0]$
<b>编译结果</b>	仅存在 32 位指令。 pmul.u16 rz, rx, ry pmul.s16 rz, rx, ry

<b>说明:</b>	Rx 的高半字与 Ry 的高半字相乘，结果存入 Rz+1； Rx 的低半字和 Ry 的低半字相乘，结果存入 Rz。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	$Rz+1[31:0] = Rx[31:16] \times Ry[31:16]$ $Rz[31:0] = Rx[15:0] \times Ry[15:0]$
<b>语法:</b>	pmul.u16 rz, rx, ry pmul.s16 rz, rx, ry
<b>说明:</b>	Rx 的高半字与 Ry 的高半字相乘，结果存入 Rz+1； Rx 的低半字和 Ry 的低半字相乘，结果存入 Rz。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

31 30	26 25	21 20	16 15	10 9	5 4	0
1   1    1    1    1    0   Ry   Rx   1    0    0    0    0   1   0    0    S    0    1   Rz						

图 15.117: PMUL.(US)16

**注意:** S 为 1 时为 unsign, S 为 0 时为 sign

### 15.118 PMULX.(U/S)16——16 位并行（无/有）符号交叉乘法指令

<b>统一化指令</b>	
<b>语法</b>	pmulx.u16 rz, rx, ry pmulx.s16 rz, rx, ry
<b>操作</b>	Rz+1[31:0] = Rx[31:16] X Ry[15:0] Rz[31:0] = Rx[15:0] X Ry[31:16]
<b>编译结果</b>	仅存在 32 位指令。 pmulx.u16 rz, rx, ry pmulx.s16 rz, rx, ry

<b>说明:</b>	Rx 的高半字和 Ry 的低半字相乘，结果存入 Rz+1； Rx 的低半字和 Ry 的高半字相乘，结果存入 Rz；
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	Rz+1[31:0] = Rx[31:16] X Ry[15:0] Rz[31:0] = Rx[15:0] X Ry[31:16]
<b>语法:</b>	pmulx.u16 rz, rx, ry pmulx.s16 rz, rx, ry
<b>说明:</b>	Rx 的高半字和 Ry 的低半字相乘，结果存入 Rz+1； Rx 的低半字和 Ry 的高半字相乘，结果存入 Rz；
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

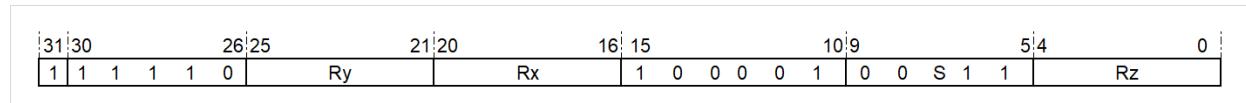


图 15.118: PMULX.(US)16

注意：S 为 1 时为 unsign，S 为 0 时为 sign

### 15.119 PRMUL.S16——16 位并行有符号小数乘法指令

统 一 化 指 令	
语 法	prmul.s16 rz, rx, ry
操 作	<pre> if(Rx[31:16] == 16' h8000 &amp;&amp; Ry[31:16] == 16' h8000) Rz+1[31:0] = 32' h7FFF FFFF else Rz+1[31:0] = {Rx[31:16] X Ry[31:16]} &lt;&lt; 1 if(Rx[15:0] == 16' h8000 &amp;&amp; Ry[15:0] == 16' h8000) Rz[31:0] = 32' h7FFF FFFF else Rz[31:0] = {Rx[15:0] X Ry[15:0]}&lt;&lt; 1 </pre>
编 译 结 果	<p>仅存在 32 位指令。</p> <pre>prmul.s16 rz, rx, ry</pre>

说 明:	Rx 与 Ry 的高半字均视为有符号小数，其中第 31 位为符号位、第 30~16 位为小数位；Rx 与 Ry 的低半字视为有符号小数，其中第 15 位为符号位、第 14~0 位为小数位。 当 Rx[31:16]=0x8000 且 Ry[31:16]=0x8000 时，结果为 0x7FFF FFFF；其余情况，Rx 高半字与 Ry 高半字相乘，乘法结果左移一位后存入 Rz+1。 当 Rx [15:0]=0x8000 且 Ry[15:0]=0x8000 时，结果为 0x7FFF FFFF；其余情况，Rx 低半字与 Ry 低半字相乘，乘法结果左移一位后存入 Rz。
影 响 标 志 位:	无影响
限 制:	无
异 常:	无

<b>32位指令</b>	
<b>操作:</b>	<pre>if(Rx[31:16] == 16' h8000 &amp;&amp; Ry[31:16] == 16' h8000) Rz+1[31:0] = 32' h7FFF FFFF else Rz+1[31:0] = {Rx[31:16] X Ry[31:16]} &lt;&lt; 1 if(Rx[15:0] == 16' h8000 &amp;&amp; Ry[15:0] == 16' h8000) Rz[31:0] = 32' h7FFF FFFF else Rz[31:0] = {Rx[15:0] X Ry[15:0]}&lt;&lt; 1</pre>
<b>语法:</b>	prmuls16 rz, rx, ry
<b>说明:</b>	<p>Rx 与 Ry 的高半字均视为有符号小数，其中第 31 位为符号位、第 30~16 位为小数位；Rx 与 Ry 的低半字视为有符号小数，其中第 15 位为符号位、第 14~0 位为小数位。</p> <p>当 Rx[31:16]=0x8000 且 Ry[31:16]=0x8000 时，结果为 0x7FFF FFFF；其余情况，Rx 高半字与 Ry 高半字相乘，乘法结果左移一位后存入 Rz+1。</p> <p>当 Rx [15:0]=0x8000 且 Ry[15:0]=0x8000 时，结果为 0x7FFF FFFF；其余情况，Rx 低半字与 Ry 低半字相乘，乘法结果左移一位后存入 Rz。</p>
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

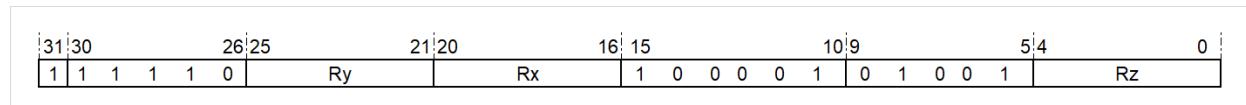
**指令格式：**

图 15.119: PRMUL.S16

## 15.120 PRMULX.S16——16 位并行有符号交叉小数乘法指令

统 一 化 指 令	
语 法	prmulx.s16 rz, rx, ry
操 作	<pre> if(Rx[31:16] == 16' h8000 &amp;&amp; Ry[15:0] == 16' h8000) Rz+1[31:0] = 32' h7FFF FFFF else Rz+1[31:0] = {Rx[31:16] X Ry[15:0]} &lt;&lt; 1 if(Rx[15:0] == 16' h8000 &amp;&amp; Ry[31:16] == 16' h8000) Rz[31:0] = 32' h7FFF FFFF else Rz[31:0] = {Rx[15:0] X Ry[31:16]} &lt;&lt; 1 </pre>
编 译 结 果	仅存在 32 位指令。 prmulx.s16 rz, rx, ry

说 明:	Rx 与 Ry 的高半字均视为有符号小数，其中第 31 位为符号位、第 30~16 位为小数位； Rx 与 Ry 的低半字视为有符号小数，其中第 15 位为符号位、第 14~0 位为小数位。 当 Rx[ 31:16]=0x8000 且 Ry[15:0]=0x8000 时，结果为 0x7FFF FFFF；其余情况，Rx 高半字与 Ry 低半字相乘，乘法结果左移一位后存入 Rz+1。 当 Rx[ 15:0]=0x8000 且 Ry[31:16]=0x8000 时，结果为 0x7FFF FFFF；其余情况，Rx 低半字与 Ry 高半字相乘，乘法结果左移一位后存入 Rz。
影 响 标 志 位:	无影响
限 制:	无
异 常:	无

<b>32位指令</b>	
<b>操作:</b>	<pre> if(Rx[31:16] == 16' h8000 &amp;&amp; Ry[15:0] == 16' h8000) Rz+1[31:0] = 32' h7FFF FFFF else Rz+1[31:0] = {Rx[31:16] X Ry[15:0]} &lt;&lt; 1 if(Rx[15:0] == 16' h8000 &amp;&amp; Ry[31:16] == 16' h8000) Rz[31:0] = 32' h7FFF FFFF else Rz[31:0] = {Rx[15:0] X Ry[31:16]} &lt;&lt; 1 </pre>
<b>语法:</b>	prmulx.s16 rz, rx, ry
<b>说明:</b>	<p>Rx 与 Ry 的高半字均视为有符号小数，其中第 31 位为符号位、第 30~16 位为小数位；Rx 与 Ry 的低半字视为有符号小数，其中第 15 位为符号位、第 14~0 位为小数位。</p> <p>当 Rx[ 31:16]=0x8000 且 Ry[15:0]=0x8000 时，结果为 0x7FFF FFFF；其余情况，Rx 高半字与 Ry 低半字相乘，乘法结果左移一位后存入 Rz+1。</p> <p>当 Rx[ 15:0]=0x8000 且 Ry[31:16]=0x8000 时，结果为 0x7FFF FFFF；其余情况，Rx 低半字与 Ry 高半字相乘，乘法结果左移一位后存入 Rz。</p>
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

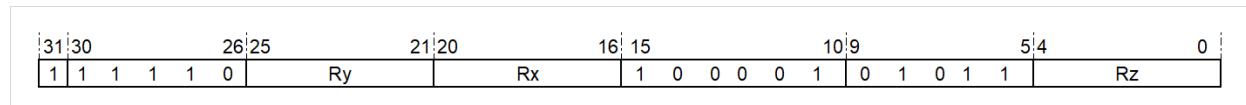
**指令格式：**

图 15.120: PRMULX.S16

## 15.121 PRMUL.S16.H——16 位并行有符号小数乘法取高 16 位指令

统 一 化 指 令	
语 法	prmul.s16.h rz, rx, ry
操 作	<pre> if(Rx[31:16] == 16' h8000 &amp;&amp; Ry[31:16] == 16' h8000) Rz[31:16] = 16' h7FFF else Rz[31:16] = {Rx[31:16] X Ry[31:16]}[30:15] if(Rx[15:0] == 16' h8000 &amp;&amp; Ry[15:0] == 16' h8000) Rz[15:0] = 16' h7FFF else Rz[15:0] = {Rx[15:0] X Ry[15:0]}[30:15]</pre>
编 译 结 果	仅存在 32 位指令。 prmul.s16.h rz, rx, ry

说 明:	Rx 与 Ry 的高半字均视为有符号小数，其中第 31 位为符号位、第 30~16 位为小数位；Rx 与 Ry 的低半字视为有符号小数，其中第 15 位为符号位、第 14~0 位为小数位。 当 Rx[31:16]=0x8000 且 Ry[31:16]=0x80 00 时，结果为 0x7FFF；其余情况，Rx 高半字与 Ry 高半字相乘，取乘法结果的第 30~15 位存入 Rz 的高半字。 当 Rx[15:0]=0x8000 且 Ry[15:0]=0x80 00 时，结果为 0x7FFF；其余情况，Rx 低半字与 Ry 低半字相乘，取乘法结果的第 30~15 位存入 Rz 的低半字。
影 响 标 志 位:	无影响
限 制:	无
异 常:	无

<b>32位指令</b>	
<b>操作:</b>	<pre>if(Rx[31:16] == 16' h8000 &amp;&amp; Ry[31:16] == 16' h8000) Rz[31:16] = 16' h7FFF else Rz[31:16] = {Rx[31:16] X Ry[31:16]}[30:15] if(Rx[15:0] == 16' h8000 &amp;&amp; Ry[15:0] == 16' h8000) Rz[15:0] = 16' h7FFF else Rz[15:0] = {Rx[15:0] X Ry[15:0]}[30:15]</pre>
<b>语法:</b>	prmuls16.h rz, rx, ry
<b>说明:</b>	<p>Rx 与 Ry 的高半字均视为有符号小数，其中第 31 位为符号位、第 30~16 位为小数位；Rx 与 Ry 的低半字视为有符号小数，其中第 15 位为符号位、第 14~0 位为小数位。</p> <p>当 Rx[31:16]=0x8000 且 Ry[31:16]=0x80 00 时，结果为 0x7FFF；其余情况，Rx 高半字与 Ry 高半字相乘，取乘法结果的第 30~15 位存入 Rz 的高半字。</p> <p>当 Rx[15:0]=0x8000 且 Ry[15:0]=0x80 00 时，结果为 0x7FFF；其余情况，Rx 低半字与 Ry 低半字相乘，取乘法结果的第 30~15 位存入 Rz 的低半字。</p>
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

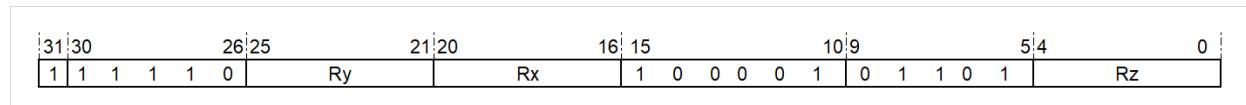
**指令格式：**

图 15.121: PRMUL.S16.H

## 15.122 PRMUL.S16.RH——带舍入操作的 16 位并行有符号小数乘法 取高 16 位指令

统 一 化 指 令	
语 法	prmul.s16.rh rz, rx, ry
操 作	<pre> if(Rx[31:16] == 16' h8000 &amp;&amp; Ry[31:16] == 16' h8000) Rz[31:16] = 16' h7FFF else Rz[31:16] = {Rx[31:16] X Ry[31:16] + 16' h4000}[30:15] if(Rx[15:0] == 16' h8000 &amp;&amp; Ry[15:0] == 16' h8000) Rz[15:0] = 16' h7FFF else Rz[15:0] = {Rx[15:0] X Ry[15:0] + 16' h4000}[30:15]</pre>
编 译 结 果	<p>仅存在 32 位指令。</p> <p>prmul.s16.rh rz, rx, ry</p>

说 明:	Rx 与 Ry 的高半字均视为有符号小数，其中第 31 位为符号位、第 30~16 位为小数位；Rx 与 Ry 的低半字视为有符号小数，其中第 15 位为符号位、第 14~0 位为小数位。 当 Rx[31:16]=0x8000 且 Ry[31:16]=0x8000 时，结果为 0x7FFF；其余情况，Rx 高半字与 Ry 高半字相乘，乘法结果加上 0x4000，取加法结果的第 30~15 位存入 Rz 的高半字。乘法结果加上 0x4000 用以实现舍入操作。 当 Rx[15:0]=0x8000 且 Ry[15:0]=0x8000 时，结果为 0x7FFF；其余情况，Rx 低半字与 Ry 低半字相乘，乘法结果加上 0x4000，取加法结果的第 30~15 位存入 Rz 的低半字。乘法结果加上 0x4000 用以实现舍入操作。
影 响 标 志 位:	无影响
限 制:	无
异 常:	无

<b>32 位 指 令</b>	
<b>操 作:</b>	<pre>if(Rx[31:16] == 16' h8000 &amp;&amp; Ry[31:16] == 16' h8000) Rz[31:16] = 16' h7FFF else Rz[31:16] = {Rx[31:16] X Ry[31:16] + 16' h4000}[30:15] if(Rx[15:0] == 16' h8000 &amp;&amp; Ry[15:0] == 16' h8000) Rz[15:0] = 16' h7FFF else Rz[15:0] = {Rx[15:0] X Ry[15:0] + 16' h4000}[30:15]</pre>
<b>语 法:</b>	prmul.s16.rh rz, rx, ry
<b>说 明:</b>	<p>Rx 与 Ry 的高半字均视为有符号小数，其中第 31 位为符号位、第 30~16 位为小数位；Rx 与 Ry 的低半字视为有符号小数，其中第 15 位为符号位、第 14~0 位为小数位。</p> <p>当 Rx[31:16]=0x8000 且 Ry[31:16]=0x8000 时，结果为 0x7FFF；其余情况，Rx 高半字与 Ry 高半字相乘，乘法结果加上 0x4000，取加法结果的第 30~15 位存入 Rz 的高半字。乘法结果加上 0x4000 用以实现舍入操作。</p> <p>当 Rx[15:0]=0x8000 且 Ry[15:0]=0x8000 时，结果为 0x7FFF；其余情况，Rx 低半字与 Ry 低半字相乘，乘法结果加上 0x4000，取加法结果的第 30~15 位存入 Rz 的低半字。乘法结果加上 0x4000 用以实现舍入操作。</p>
<b>影 响 标 志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

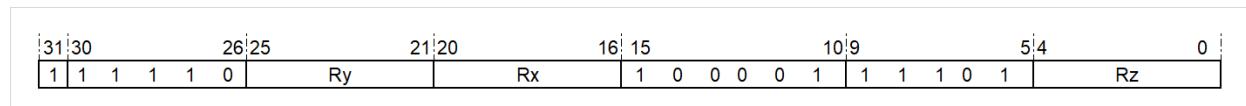
**指令格式：**

图 15.122: PRMUL.S16.RH

## 15.123 PRMULX.S16.H——16 位并行有符号交叉小数乘法取高 16 位指令

统 一 化 指 令	
语 法	prmulx.s16.h rz, rx, ry
操 作	<pre> if(Rx[31:16] == 16' h8000 &amp;&amp; Ry[15:0] == 16' h8000)     Rz[31:16] = 16' h7FFF else     Rz[31:16] = {Rx[31:16] X Ry[15:0]}[30:15] if(Rx[15:0] == 16' h8000 &amp;&amp; Ry[31:16] == 16' h8000)     Rz[15:0] = 16' h7FFF else     Rz[15:0] = {Rx[15:0] X Ry[31:16]}[30:15]</pre>
编 译 结 果	<p>仅存在 32 位指令。</p> <p>prmulx.s16.h rz, rx, ry</p>

说 明:	<p>Rx 与 Ry 的高半字均视为有符号小数，其中第 31 位为符号位、第 30~16 位为小数位；Rx 与 Ry 的低半字视为有符号小数，其中第 15 位为符号位、第 14~0 位为小数位。</p> <p>当 Rx[31:16]=0x8000 且 Ry[31:16]=0x80 00 时，结果为 0x7FFF；其余情况，Rx 高半字与 Ry 高半字相乘，取乘法结果的第 30~15 位存入 Rz 的高半字。</p> <p>当 Rx[15:0]=0x8000 且 Ry[15:0]=0x80 00 时，结果为 0x7FFF；其余情况，Rx 低半字与 Ry 低半字相乘，取乘法结果的第 30~15 位存入 Rz 的低半字。</p>
影 响 标 志 位:	无影响
限 制:	无
异 常:	无

32位指令	
操作:	<pre> if(Rx[31:16] == 16' h8000 &amp;&amp; Ry[15:0] == 16' h8000) Rz[31:16] = 16' h7FFF else Rz[31:16] = {Rx[31:16] X Ry[15:0]}[30:15] if(Rx[15:0] == 16' h8000 &amp;&amp; Ry[31:16] == 16' h8000) Rz[15:0] = 16' h7FFF else Rz[15:0] = {Rx[15:0] X Ry[31:16]}[30:15]</pre>
语法:	prmulx.s16.h rz, rx, ry
说明:	<p>Rx 与 Ry 的高半字均视为有符号小数，其中第 31 位为符号位、第 30~16 位为小数位；Rx 与 Ry 的低半字视为有符号小数，其中第 15 位为符号位、第 14~0 位为小数位。</p> <p>当 Rx[31:16]=0x8000 且 Ry[31:16]=0x80 00 时，结果为 0x7FFF；其余情况，Rx 高半字与 Ry 高半字相乘，取乘法结果的第 30~15 位存入 Rz 的高半字。</p> <p>当 Rx[15:0]=0x8000 且 Ry[15:0]=0x80 00 时，结果为 0x7FFF；其余情况，Rx 低半字与 Ry 低半字相乘，取乘法结果的第 30~15 位存入 Rz 的低半字。</p>
影响标志位:	无影响
限制:	无
异常:	无

**指令格式：**

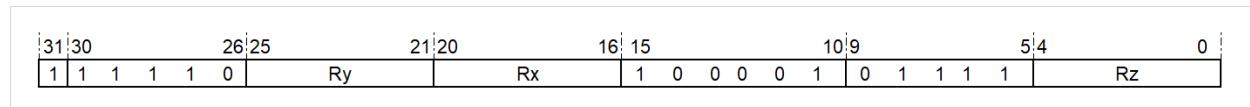


图 15.123: PRMULX.S16.H

## 15.124 PRMULX.S16.RH——带舍入操作的 16 位并行有符号交叉小数乘法取高 16 位指令

统 一 化 指 令	
语 法	prmulx.s16.rh rz, rx, ry
操 作	<pre> if(Rx[31:16] == 16' h8000 &amp;&amp; Ry[15:0] == 16' h8000) Rz[31:16] = 16' h7FFF else Rz[31:16] = {Rx[31:16] X Ry[15:0] + 16' h4000}[30:15] if(Rx[15:0] == 16' h8000 &amp;&amp; Ry[31:16] == 16' h8000) Rz[15:0] = 16' h7FFF else Rz[15:0] = {Rx[15:0] X Ry[31:16] + 16' h4000}[30:15]</pre>
编 译 结 果	<p>仅存在 32 位指令。</p> <pre>prmulx.s16.rh rz, rx, ry</pre>

说 明:	Rx 与 Ry 的高半字均视为有符号小数，其中第 31 位为符号位、第 30~16 位为小数位；Rx 与 Ry 的低半字视为有符号小数，其中第 15 位为符号位、第 14~0 位为小数位。 当 Rx[31:16]=0x8000 且 Ry[15:0]=0x8000 时，结果为 0x7FFF；其余情况，Rx 高半字与 Ry 低半字相乘，乘法结果加上 0x4000，取加法结果的第 30~15 位存入 Rz 的高半字。乘法结果加上 0x4000 用以实现舍入操作。 当 Rx[15:0]=0x8000 且 Ry[31:16]=0x8000 时，结果为 0x7FFF；其余情况，Rx 低半字与 Ry 高半字相乘，乘法结果加上 0x4000，取加法结果的第 30~15 位存入 Rz 的低半字。乘法结果加上 0x4000 用以实现舍入操作。
影 响 标 志 位:	无影响
限 制:	无
异 常:	无

<b>32位指令</b>	
<b>操作:</b>	<pre>if(Rx[31:16] == 16' h8000 &amp;&amp; Ry[15:0] == 16' h8000) Rz[31:16] = 16' h7FFF else Rz[31:16] = {Rx[31:16] X Ry[15:0] + 16' h4000}[30:15] if(Rx[15:0] == 16' h8000 &amp;&amp; Ry[31:16] == 16' h8000) Rz[15:0] = 16' h7FFF else Rz[15:0] = {Rx[15:0] X Ry[31:16] + 16' h4000}[30:15]</pre>
<b>语法:</b>	prmulp.s16.rh rz, rx, ry
<b>说明:</b>	<p>Rx 与 Ry 的高半字均视为有符号小数，其中第 31 位为符号位、第 30~16 位为小数位；Rx 与 Ry 的低半字视为有符号小数，其中第 15 位为符号位、第 14~0 位为小数位。</p> <p>当 Rx[31:16]=0x8000 且 Ry[15:0]=0x8000 时，结果为 0x7FFF；其余情况，Rx 高半字与 Ry 低半字相乘，乘法结果加上 0x4000，取加法结果的第 30~15 位存入 Rz 的高半字。乘法结果加上 0x4000 用以实现舍入操作。</p> <p>当 Rx[15:0]=0x8000 且 Ry[31:16]=0x8000 时，结果为 0x7FFF；其余情况，Rx 低半字与 Ry 高半字相乘，乘法结果加上 0x4000，取加法结果的第 30~15 位存入 Rz 的低半字。乘法结果加上 0x4000 用以实现舍入操作。</p>
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

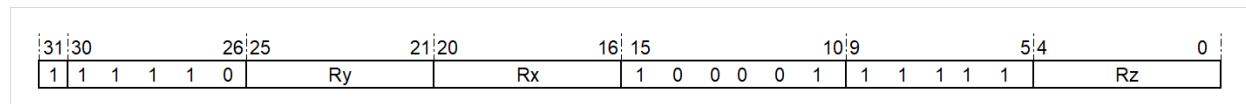
**指令格式：**

图 15.124: PRMULX.S16.RH

## 15.125 MULCA.S16.S——带饱和操作的 16 位有符号乘链加指令

<b>统一化 指令</b>	
<b>语法</b>	mulca.s16.s rz, rx, ry
<b>操作</b>	<pre>if(Rx[31:0] = 32' h80008000 &amp;&amp; Ry[31:0] = 32' h80008000) Rz[31:0] = 32' h7FFF FFFF else Rz[31:0] = Rx[31:16] X Ry[31:16] + Rx[15:0] X Ry[15:0]</pre>
<b>编译结 果</b>	仅存在 32 位指令。 mulca.s16.s rz, rx, ry

<b>说明:</b>	Rx 的低半字与 Ry 的低半字相乘，Rx 的高半字与 Ry 的高半字相乘，两个乘法结果相加。当 Rx=0x80008000 且 Ry= 0x80008000 时，乘链加结果溢出，饱和处理为 0x7FFF FFFF。
<b>影 响 标 志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

<b>32 位 指令</b>	
<b>操作:</b>	<pre>if(Rx[31:0] = 32' h80008000 &amp;&amp; Ry[31:0] = 32' h80008000) Rz[31:0] = 32' h7FFF FFFF else Rz[31:0] = Rx[31:16] X Ry[31:16] + Rx[15:0] X Ry[15:0]</pre>
<b>语法:</b>	mulca.s16.s rz, rx, ry
<b>说明:</b>	Rx 的低半字与 Ry 的低半字相乘，Rx 的高半字与 Ry 的高半字相乘，两个乘法结果相加。当 Rx=0x80008000 且 Ry= 0x80008000 时，乘链加结果溢出，饱和处理为 0x7FFF FFFF。
<b>影 响 标 志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

**指令格式:**

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 0	1 1 1 1 1	Rz	

图 15.125: MULCA.S16.S

### 15.126 MULCAX.S16.S——带饱和操作的 16 位有符号交叉乘链加指令

统一化 指令	
语法	mulcax.s16.s rz, rx, ry
操作	<pre>if(Rx[31:0] = 32' h80008000 &amp;&amp; Ry[31:0] = 32' h80008000)   Rz[31:0] = 32' h7FFF FFFF else   Rz[31:0] = Rx[31:16] X Ry[15:0] + Rx[15:0] X Ry[31:16]</pre>
编译结 果	仅存在 32 位指令。 mulcax.s16.s rz, rx, ry

说明:	Rx 的低半字与 Ry 的高半字相乘, Rx 的高半字与 Ry 的低半字相乘, 两个乘法结果相加。当 Rx=0x80008000 且 Ry= 0x80008000 时, 乘链加结果溢出, 饱和处理为 0x7FFF FFFF。
影响 标志 位:	无影响
限制:	无
异常:	无
指 令 格 式:	

<b>32 位指令</b>	
<b>操作:</b>	if( $Rx[31:0] = 32' h80008000 \&& Ry[31:0] = 32' h80008000$ ) $Rz[31:0] = 32' h7FFF FFFF$ else $Rz[31:0] = Rx[31:16] X Ry[15:0] + Rx[15:0] X Ry[31:16]$
<b>语法:</b>	mulcax.s16.s rz, rx, ry
<b>说明:</b>	Rx 的低半字与 Ry 的高半字相乘，Rx 的高半字与 Ry 的低半字相乘，两个乘法结果相加。当 Rx=0x80008000 且 Ry= 0x80008000 时，乘链加结果溢出，饱和处理为 0x7FFF FFFF。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

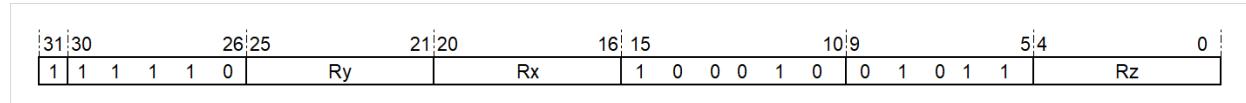


图 15.126: MULCAX.S16.S

### 15.127 MULCS.S16——16 位有符号乘链减指令

<b>统一化指令</b>	
<b>语法</b>	mulcs.s16 rz, rx, ry
<b>操作</b>	$Rz[31:0] = Rx[15:0] X Ry[15:0] - Rx[31:16] X Ry[31:16]$
<b>编译结果</b>	仅存在 32 位指令。 mulcs.s16 rz, rx, ry

<b>说明:</b>	Rx 的低半字与 Ry 的低半字相乘，Rx 的高半字与 Ry 的高半字相乘，两个乘法结果相减。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

32 位指令	
操作:	Rz[31:0] = Rx[15:0] X Ry[15:0] - Rx[31:16] X Ry[31:16]
语法:	mulcs.s16 rz, rx, ry
说明:	Rx 的低半字与 Ry 的低半字相乘, Rx 的高半字与 Ry 的高半字相乘, 两个乘法结果相减。
影响标志位:	无影响
限制:	无
异常:	无

指令格式:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 0	1 0 0 0 1	Rz	

图 15.127: MULCS.S16

### 15.128 MULCSR.S16——16 位有符号反向乘链减指令

统一化指令	
语法	mulcsr.s16 rz, rx, ry
操作	Rz[31:0] = Rx[31:16] X Ry[31:16] - Rx[15:0] X Ry[15:0]
编译结果	仅存在 32 位指令。 mulcsr.s16 rz, rx, ry

说明:	Rx 的高半字与 Ry 的高半字相乘, Rx 的低半字与 Ry 的低半字相乘, 两个乘法结果相减。
影响标志位:	无影响
限制:	无
异常:	无

32 位指令	
操作:	Rz[31:0] = Rx[31:16] X Ry[31:16] - Rx[15:0] X Ry[15:0]
语法:	mulcsr.s16 rz, rx, ry
说明:	Rx 的高半字与 Ry 的高半字相乘, Rx 的低半字与 Ry 的低半字相乘, 两个乘法结果相减。
影响标志位:	无影响
限制:	无
异常:	无

指令格式:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1	0 1 0 0 1	1	Rz

图 15.128: MULCSR.S16

### 15.129 MULCSX.S16——16 位有符号交叉乘链减指令

统一化指令	
语法	mulcsx.s16 rz, rx, ry
操作	$Rz[31:0] = Rx[15:0] \times Ry[31:16] - Rx[31:16] \times Ry[15:0]$
编译结果	仅存在 32 位指令。 mulcsx.s16 rz, rx, ry

说明:	Rx 的低字与 Ry 的高半字相乘, Rx 的高半字与 Ry 的低半字相乘, 两个乘法结果相减。
影响标志位:	无影响
限制:	无
异常:	无

32 位指令	
操作:	$Rz[31:0] = Rx[15:0] \times Ry[31:16] - Rx[31:16] \times Ry[15:0]$
语法:	mulcsx.s16 rz, rx, ry
说明:	Rx 的低字与 Ry 的高半字相乘, Rx 的高半字与 Ry 的低半字相乘, 两个乘法结果相减。
影响标志位:	无影响
限制:	无
异常:	无

指令格式:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1	1 0 0 0 0	1	Rz

图 15.129: MULCSX.S16

## 15.130 MULACA.S16.S——带饱和操作的 16 位有符号乘链加累加指令

<b>统一化指令</b>	
<b>语法</b>	mulaca.s16.s rz, rx, ry
<b>操作</b>	$Rz[31:0] = \text{Saturate}(Rz[31:0] + Rx[31:16] \times Ry[31:16] + Rx[15:0] \times Ry[15:0])$
<b>编译结果</b>	仅存在 32 位指令。 mulaca.s16.s rz, rx, ry

<b>说 明:</b>	Rx 的低半字与 Ry 的低半字相乘，Rx 的高半字与 Ry 的高半字相乘，两个乘法结果相加并与 Rz 累加。累加结果做饱和处理后存入 Rz。 饱和处理的过程为，若累加结果大于上饱和值 0x7FFF FFFF，结果为上饱和值，若累加结果小于下饱和值 0x8000 0000，结果为下饱和值，其它情况为累加结果本身。
<b>影 响 标 志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

32 位 指 令	
操作: 语 法:	$Rz[31:0] = \text{Saturate}(Rz[31:0] + Rx[31:16] \times Ry[31:16] + Rx[15:0] \times Ry[15:0])$ mulaca.s16.s rz, rx, ry
说明:	Rx 的低半字与 Ry 的低半字相乘，Rx 的高半字与 Ry 的高半字相乘，两个乘法结果相加并与 Rz 累加。累加结果做饱和处理后存入 Rz。 饱和处理的过程为，若累加结果大于上饱和值 0x7FFF FFFF，结果为上饱和值，若累加结果小于下饱和值 0x8000 0000，结果为下饱和值，其它情况为累加结果本身。
影响 标志 位:	无影响
限制:	无
异常:	无

指令格式：

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 0	0 1 1 0 1	Rz	

图 15.130: MULACA.S16.S

### 15.131 MULACAX.S16.S——带饱和操作的 16 位有符号交叉乘链加累加指令

统一化指令	
语法	mulacax.s16.s rz, rx, ry
操作	$Rz[31:0] = \text{Saturate}(Rz[31:0] + Rx[31:16] \times Ry[31:16] + Rx[15:0] \times Ry[15:0])$
编译结果	仅存在 32 位指令。 mulacax.s16.s rz, rx, ry

<b>说 明:</b>	Rx 的低半字与 Ry 的高半字相乘，Rx 的高半字与 Ry 的低半字相乘，两个乘法结果相加并与 Rz 累加。累加结果做饱和处理后存入 Rz。 饱和处理的过程为，若累加结果大于上饱和值 0x7FFF FFFF，结果为上饱和值，若累加结果小于下饱和值 0x8000 0000，结果为下饱和值，其它情况为累加结果本身。
<b>影 响 标 志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

<b>32 位 指 令</b>	
<b>操 作:</b>	$Rz[31:0] = \text{Saturate}(Rz[31:0] + Rx[31:16] \times Ry[15:0] + Rx[15:0] \times Ry[31:16])$
<b>语 法:</b>	mulacax.s16.s rz, rx, ry
<b>说 明:</b>	Rx 的低半字与 Ry 的高半字相乘，Rx 的高半字与 Ry 的低半字相乘，两个乘法结果相加并与 Rz 累加。累加结果做饱和处理后存入 Rz。 饱和处理的过程为，若累加结果大于上饱和值 0x7FFF FFFF，结果为上饱和值，若累加结果小于下饱和值 0x8000 0000，结果为下饱和值，其它情况为累加结果本身。
<b>影 响 标 志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

**指令格式:**

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0 Ry		Rx	1 0 0 0 1 0	0 1 1 1 1 Rz		

图 15.131: MULACAX.S16.S

### 15.132 MULACS.S16.S——带饱和操作的 16 位有符号乘链减累加指令

统一化指令	
语法	mulacs.s16.s rz, rx, ry
操作	$Rz[31:0] = \text{Saturate}(Rz[31:0] + Rx[15:0] \times Ry[15:0] - Rx[31:16] \times Ry[31:16])$
编译结果	仅存在 32 位指令。 mulacs.s16.s rz, rx, ry

说 明:	Rx 的低半字与 Ry 的低半字相乘，Rx 的高半字与 Ry 的高半字相乘，两个乘法结果相减并与 Rz 累加。累加结果做饱和处理后存入 Rz。 饱和处理的过程为，若累加结果大于上饱和值 0x7FFF FFFF，结果为上饱和值，若累加结果小于下饱和值 0x8000 0000，结果为下饱和值，其它情况为累加结果本身。
影 响 标 志 位:	无影响
限 制:	无
异 常:	无

32 位 指 令	
操作: 语 法:	Rz[31:0] = Saturate(Rz[31:0] + Rx[15:0] X Ry[15:0] - Rx[31:16] X Ry[31:16]) mulacs.s16.s rz, rx, ry
说明:  说 明:	Rx 的低半字与 Ry 的低半字相乘，Rx 的高半字与 Ry 的高半字相乘，两个乘法结果相减并与 Rz 累加。累加结果做饱和处理后存入 Rz。 饱和处理的过程为，若累加结果大于上饱和值 0x7FFF FFFF，结果为上饱和值，若累加结果小于下饱和值 0x8000 0000，结果为下饱和值，其它情况为累加结果本身。
影响 标志 位:	无影响
限制:	无
异常:	无

指令格式：

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 0	1 1 1 0 1	Rz	

图 15.132: MULACS.S16.S

### 15.133 MULACSR.S16.S——带饱和操作的 16 位有符号反向乘链减累加指令

统一化指令	
语法	mulacsr.s16.s rz, rx, ry
操作	Rz[31:0] = Saturate(Rz[31:0] + Rx[31:16] X Ry[31:16] - Rx[15:0] X Ry[15:0])
编译结果	仅存在 32 位指令。 mulacsr.s16.s rz, rx, ry

<b>说 明:</b>	Rx 的高半字与 Ry 的高半字相乘， Rx 的低半字与 Ry 的低半字相乘，两个乘法结果相减并与 Rz 累加。累加结果做饱和处理后存入 Rz。 饱和处理的过程为，若累加结果大于上饱和值 0x7FFF FFFF，结果为上饱和值，若累加结果小于下饱和值 0x8000 0000，结果为下饱和值，其它情况为累加结果本身。
<b>影 响 标 志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

<b>32 位 指 令</b>	
<b>操 作:</b>	$Rz[31:0] = \text{Saturate}(Rz[31:0] + Rx[31:16] \times Ry[31:16] - Rx[15:0] \times Ry[15:0])$
<b>语 法:</b>	mulacs.s16.s rz, rx, ry
<b>说 明:</b>	Rx 的高半字与 Ry 的高半字相乘， Rx 的低半字与 Ry 的低半字相乘，两个乘法结果相减并与 Rz 累加。累加结果做饱和处理后存入 Rz。 饱和处理的过程为，若累加结果大于上饱和值 0x7FFF FFFF，结果为上饱和值，若累加结果小于下饱和值 0x8000 0000，结果为下饱和值，其它情况为累加结果本身。
<b>影 响 标 志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

**指令格式:**

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 0	1 1 1 1 1	Rz	

图 15.133: MULACSR.S16.S

### 15.134 MULACSX.S16.S——带饱和操作的 16 位有符号交叉乘链减累加指令

统一化指令	
语法	mulacsx.s16.s rz, rx, ry
操作	$Rz[31:0] = \text{Saturate}(Rz[31:0] + Rx[15:0] \times Ry[31:16] - Rx[31:16] \times Ry[15:0])$
编译结果	仅存在 32 位指令。 mulacsx.s16.s rz, rx, ry

说 明:	Rx 的低半字与 Ry 的高半字相乘， Rx 的高半字与 Ry 的低半字相乘，两个乘法结果相减并与 Rz 累加。累加结果做饱和处理后存入 Rz。 饱和处理的过程为，若累加结果大于上饱和值 0x7FFF FFFF，结果为上饱和值，若累加结果小于下饱和值 0x8000 0000，结果为下饱和值，其它情况为累加结果本身。
影 响 标 志 位:	无影响
限 制:	无
异 常:	无
32 位 指 令	
操 作:	$Rz[31:0] = \text{Saturate}(Rz[31:0] + Rx[15:0] \times Ry[31:16] - Rx[31:16] \times Ry[15:0])$
语 法:	mulacsx.s16.s rz, rx, ry
说 明:	Rx 的低半字与 Ry 的高半字相乘， Rx 的高半字与 Ry 的低半字相乘，两个乘法结果相减并与 Rz 累加。累加结果做饱和处理后存入 Rz。 饱和处理的过程为，若累加结果大于上饱和值 0x7FFF FFFF，结果为上饱和值，若累加结果小于下饱和值 0x8000 0000，结果为下饱和值，其它情况为累加结果本身。
影 响 标 志 位:	无影响
限 制:	无
异 常:	无

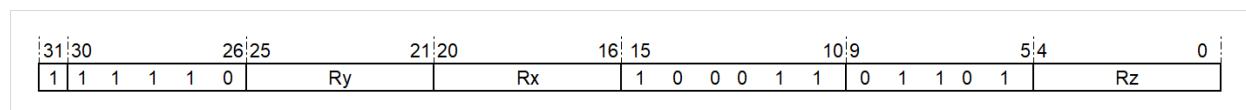
**指令格式：**

图 15.134: MULACSX.S16.S

## 15.135 MULSCA.S16.S——带饱和操作的 16 位有符号乘链加累减指令

<b>统一化指令</b>	
<b>语法</b>	mulscas16.s rz, rx, ry
<b>操作</b>	$Rz[31:0] = \text{Saturate}(Rz[31:0] - Rx[31:16] \times Ry[31:16] - Rx[15:0] \times Ry[15:0])$
<b>编译结果</b>	仅存在 32 位指令。 mulscas16.s rz, rx, ry

<b>说 明:</b>	Rx 的低半字与 Ry 的低半字相乘，Rx 的高半字与 Ry 的高半字相乘，两个乘法结果相加并与 Rz 累减。累减结果做饱和处理后存入 Rz。 饱和处理的过程为，若累减结果大于上饱和值 0x7FFF FFFF，结果为上饱和值，若累减结果小于下饱和值 0x8000 0000，结果为下饱和值，其它情况为累减结果本身。
<b>影 响 标 志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

32 位 指 令	
操作: 语 法:	Rz[31:0] = Saturate(Rz[31:0] - Rx[31:16] X Ry[31:16] - Rx[15:0] X Ry[15:0]) mulsca.s16.s rz, rx, ry
说明:  说 明:	Rx 的低半字与 Ry 的低半字相乘，Rx 的高半字与 Ry 的高半字相乘，两个乘法结果相加并与 Rz 累减。累减结果做饱和处理后存入 Rz。 饱和处理的过程为，若累减结果大于上饱和值 0x7FFF FFFF，结果为上饱和值，若累减结果小于下饱和值 0x8000 0000，结果为下饱和值，其它情况为累减结果本身。
影响 标志 位:	无影响
限制:	无
异常:	无

指令格式：

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 1	0 1 1 1 1	Rz	

图 15.135: MULSCA.S16.S

### 15.136 MULSCAX.S16.S——带饱和操作的 16 位有符号交叉乘链加累减指令

统一化指令	
语法	mulscax.s16.s rz, rx, ry
操作	Rz[31:0] = Saturate(Rz[31:0] - Rx[31:16] X Ry[15:0] - Rx[15:0] X Ry[31:16])
编译结果	仅存在 32 位指令。 mulscax.s16.s rz, rx, ry

<b>说 明:</b>	Rx 的高半字与 Ry 的低半字相乘，Rx 的低半字与 Ry 的高半字相乘，两个乘法结果相加并与 Rz 累减。累减结果做饱和处理后存入 Rz。 饱和处理的过程为，若累减结果大于上饱和值 0x7FFF FFFF，结果为上饱和值，若累减结果小于下饱和值 0x8000 0000，结果为下饱和值，其它情况为累减结果本身。
<b>影 响 标 志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

<b>32 位 指 令</b>	
<b>操 作:</b>	$Rz[31:0] = \text{Saturate}(Rz[31:0] - Rx[31:16] \times Ry[15:0] - Rx[15:0] \times Ry[31:16])$
<b>语 法:</b>	mulscax.s16.s rz, rx, ry
<b>说 明:</b>	Rx 的高半字与 Ry 的低半字相乘，Rx 的低半字与 Ry 的高半字相乘，两个乘法结果相加并与 Rz 累减。累减结果做饱和处理后存入 Rz。 饱和处理的过程为，若累减结果大于上饱和值 0x7FFF FFFF，结果为上饱和值，若累减结果小于下饱和值 0x8000 0000，结果为下饱和值，其它情况为累减结果本身。
<b>影 响 标 志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	无

**指令格式:**

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 1	1 1 1 0 1	Rz	

图 15.136: MULSCAX.S16.S

### 15.137 MULACA.S16.E——带扩展操作的 16 位有符号乘链加累加指令

统一化指令	
语法	mulaca.s16.e rz, rx, ry
操作	$\{Rz+1[31:0], Rz[31:0]\} = \{Rz+1[31:0], Rz[31:0]\} + Rx[31:16] \times Ry[31:16] + Rx[15:0] \times Ry[15:0]$
编译结果	仅存在 32 位指令。 mulaca.s16.e rz, rx, ry

说明:	Rx 的低半字与 Ry 的低半字相乘, Rx 的高半字与 Ry 的高半字相乘, 两个乘法结果相加并与 {Rz+1, Rz} 累加。累加结果的高 32 位存入 Rz+1, 低 32 位存入 Rz。
影响标志位:	无影响
限制:	无
异常:	无

32 位指令	
操作:	$\{Rz+1[31:0], Rz[31:0]\} = \{Rz+1[31:0], Rz[31:0]\} + Rx[31:16] \times Ry[31:16] + Rx[15:0] \times Ry[15:0]$
语法:	mulaca.s16.e rz, rx, ry
说明:	Rx 的低半字与 Ry 的低半字相乘, Rx 的高半字与 Ry 的高半字相乘, 两个乘法结果相加并与 {Rz+1, Rz} 累加。累加结果的高 32 位存入 Rz+1, 低 32 位存入 Rz。
影响标志位:	无影响
限制:	无
异常:	无

指令格式:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 0	0 0 1 0 1	Rz	

图 15.137: MULACA.S16.E

### 15.138 MULACAX.S16.E——带扩展操作的 16 位有符号交叉乘链加累加指令

统一化指令	
语法	mulacax.s16.e rz, rx, ry
操作	$\{Rz+1[31:0], Rz[31:0]\} = \{Rz+1[31:0], Rz[31:0]\} + Rx[31:16] \times Ry[15:0] + Rx[15:0] \times Ry[31:16]$
编译结果	仅存在 32 位指令。 mulacax.s16.e rz, rx, ry

说明:	Rx 的高半字与 Ry 的低半字相乘, Rx 的低半字与 Ry 的高半字相乘, 两个乘法结果相加并与 {Rz+1, Rz} 累加。累加结果的高 32 位存入 Rz+1, 低 32 位存入 Rz。
影响标志位:	无影响
限制:	无
异常:	无

32 位指令	
操作:	$\{Rz+1[31:0], Rz[31:0]\} = \{Rz+1[31:0], Rz[31:0]\} + Rx[31:16] \times Ry[15:0] + Rx[15:0] \times Ry[31:16]$
语法:	mulacax.s16.e rz, rx, ry
说明:	Rx 的高半字与 Ry 的低半字相乘, Rx 的低半字与 Ry 的高半字相乘, 两个乘法结果相加并与 {Rz+1, Rz} 累加。累加结果的高 32 位存入 Rz+1, 低 32 位存入 Rz。
影响标志位:	无影响
限制:	无
异常:	无

指令格式:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 0	0 0 1 1 1	Rz	

图 15.138: MULACAX.S16.E

### 15.139 MULACS.S16.E——带扩展操作的 16 位有符号乘链减累加指令

统一化指令	
语法	mulacs.s16.e rz, rx, ry
操作	$\{Rz+1[31:0], Rz[31:0]\} = \{Rz+1[31:0], Rz[31:0]\} + Rx[15:0] \times Ry[15:0] - Rx[31:16] \times Ry[31:16]$
编译结果	仅存在 32 位指令。 mulacs.s16.e rz, rx, ry

说明:	Rx 的低半字与 Ry 的低半字相乘, Rx 的高半字与 Ry 的高半字相乘, 两个乘法结果相减并与 {Rz+1, Rz} 累加。累加结果的高 32 位存入 Rz+1, 低 32 位存入 Rz。
影响标志位:	无影响
限制:	无
异常:	无

32 位指令	
操作:	$\{Rz+1[31:0], Rz[31:0]\} = \{Rz+1[31:0], Rz[31:0]\} + Rx[15:0] \times Ry[15:0] - Rx[31:16] \times Ry[31:16]$
语法:	mulacs.s16.e rz, rx, ry
说明:	Rx 的低半字与 Ry 的低半字相乘, Rx 的高半字与 Ry 的高半字相乘, 两个乘法结果相减并与 {Rz+1, Rz} 累加。累加结果的高 32 位存入 Rz+1, 低 32 位存入 Rz。
影响标志位:	无影响
限制:	无
异常:	无

指令格式:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 0	0 0 1 1 1	Rz	

图 15.139: MULACAX.S16.E

### 15.140 MULACSR.S16.E——带扩展操作的 16 位有符号反向乘链减累加指令

统一化指令	
语法	mulacsr.s16.e rz, rx, ry
操作	$\{Rz+1[31:0], Rz[31:0]\} = \{Rz+1[31:0], Rz[31:0]\} + Rx[31:16] \times Ry[31:16] - Rx[15:0] \times Ry[15:0]$
编译结果	仅存在 32 位指令。 mulacsr.s16.e rz, rx, ry

说明:	Rx 的高半字与 Ry 的高半字相乘, Rx 的低半字与 Ry 的低半字相乘, 两个乘法结果相减并与 {Rz+1, Rz} 累加。累加结果的高 32 位存入 Rz+1, 低 32 位存入 Rz。
影响标志位:	无影响
限制:	无
异常:	无

32 位指令	
操作:	$\{Rz+1[31:0], Rz[31:0]\} = \{Rz+1[31:0], Rz[31:0]\} + Rx[31:16] \times Ry[31:16] - Rx[15:0] \times Ry[15:0]$
语法:	mulacsr.s16.e rz, rx, ry
说明:	Rx 的高半字与 Ry 的高半字相乘, Rx 的低半字与 Ry 的低半字相乘, 两个乘法结果相减并与 {Rz+1, Rz} 累加。累加结果的高 32 位存入 Rz+1, 低 32 位存入 Rz。
影响标志位:	无影响
限制:	无
异常:	无

指令格式:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 0	1 0 1 1 1	Rz	

图 15.140: MULACSR.S16.E

### 15.141 MULACSRX.S16.E——带扩展操作的 16 位有符号交叉乘链减累加指令

统一化指令	
语法	mulacsx.s16.e rz, rx, ry
操作	$\{Rz+1[31:0], Rz[31:0]\} = \{Rz+1[31:0], Rz[31:0]\} + Rx[15:0] \times Ry[31:16] - Rx[31:16] \times Ry[15:0]$
编译结果	仅存在 32 位指令。 mulacsx.s16.e rz, rx, ry

说明:	Rx 的低半字与 Ry 的高半字相乘, Rx 的高半字与 Ry 的低半字相乘, 两个乘法结果相减并与 {Rz+1, Rz} 累加。累加结果的高 32 位存入 Rz+1, 低 32 位存入 Rz。
影响标志位:	无影响
限制:	无
异常:	无

32 位指令	
操作:	$\{Rz+1[31:0], Rz[31:0]\} = \{Rz+1[31:0], Rz[31:0]\} + Rx[15:0] \times Ry[31:16] - Rx[31:16] \times Ry[15:0]$
语法:	mulacsx.s16.e rz, rx, ry
说明:	Rx 的低半字与 Ry 的高半字相乘, Rx 的高半字与 Ry 的低半字相乘, 两个乘法结果相减并与 {Rz+1, Rz} 累加。累加结果的高 32 位存入 Rz+1, 低 32 位存入 Rz。
影响标志位:	无影响
限制:	无
异常:	无

指令格式:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 1	0 0 1 0 1	Rz	

图 15.141: MULACSRX.S16.E

### 15.142 MULSCA.S16.E——带扩展操作的 16 位有符号乘链加累减指令

统一化指令	
语法	mulscas16.e rz, rx, ry
操作	$\{Rz+1[31:0], Rz[31:0]\} = \{Rz+1[31:0], Rz[31:0]\} - Rx[31:16] \times Ry[31:16] - Rx[15:0] \times Ry[15:0]$
编译结果	仅存在 32 位指令。 mulscas16.e rz, rx, ry

说明:	Rx 的低半字与 Ry 的低半字相乘，Rx 的高半字与 Ry 的高半字相乘，两个乘法结果相加并与 {Rz+1, Rz} 累减。累减结果的高 32 位存入 Rz+1，低 32 位存入 Rz。
影响标志位:	无影响
限制:	无
异常:	无

32 位指令	
操作:	$\{Rz+1[31:0], Rz[31:0]\} = \{Rz+1[31:0], Rz[31:0]\} - Rx[31:16] \times Ry[31:16] - Rx[15:0] \times Ry[15:0]$
语法:	mulscas16.e rz, rx, ry
说明:	Rx 的低半字与 Ry 的低半字相乘，Rx 的高半字与 Ry 的高半字相乘，两个乘法结果相加并与 {Rz+1, Rz} 累减。累减结果的高 32 位存入 Rz+1，低 32 位存入 Rz。
影响标志位:	无影响
限制:	无
异常:	无

指令格式:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 1	0 0 1 1 1	Rz	

图 15.142: MULSCA.S16.E

### 15.143 MULSCAX.S16.E——带扩展操作的 16 位有符号交叉乘链加累减指令

统一化指令	
语法	mulscax.s16.e rz, rx, ry
操作	$\{Rz+1[31:0], Rz[31:0]\} = \{Rz+1[31:0], Rz[31:0]\} - Rx[31:16] \times Ry[15:0] - Rx[15:0] \times Ry[31:16]$
编译结果	仅存在 32 位指令。 mulscax.s16.e rz, rx, ry

说明:	Rx 的低半字与 Ry 的高半字相乘, Rx 的高半字与 Ry 的低半字相乘, 两个乘法结果相加并与 {Rz+1, Rz} 累减。累减结果的高 32 位存入 Rz+1, 低 32 位存入 Rz。
影响标志位:	无影响
限制:	无
异常:	无

32 位指令	
操作:	$\{Rz+1[31:0], Rz[31:0]\} = \{Rz+1[31:0], Rz[31:0]\} - Rx[31:16] \times Ry[15:0] - Rx[15:0] \times Ry[31:16]$
语法:	mulscax.s16.e rz, rx, ry
说明:	Rx 的低半字与 Ry 的高半字相乘, Rx 的高半字与 Ry 的低半字相乘, 两个乘法结果相加并与 {Rz+1, Rz} 累减。累减结果的高 32 位存入 Rz+1, 低 32 位存入 Rz。
影响标志位:	无影响
限制:	无
异常:	无

指令格式:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 0 0 0 1 1	1 0 1 0 1	Rz	

图 15.143: MULSCAX.S16.E

### 15.144 PSABSA.U8——8 位并行无符号减法绝对值链加指令

统一化 指令	
语法	psabsa.u8 rz, rx, ry
操作	$Rz[31:0] = \text{abs}(Rx[7:0] - Ry[7:0]) + \text{abs}(Rx[15:8] - Ry[15:8]) + \text{abs}(Rx[23:16] - Ry[23:16]) + \text{abs}(Rx[31:24] - Ry[31:24])$
编译结果	仅存在 32 位指令。 psabsa.u8 rz, rx, ry

说明:	以字节为单位, 将 Rx 的 4 个字节分别与 Ry 的 4 个字节相减, 减法结果取绝对值后链加, 链加结果存入 Rz。
影响标志位:	无影响
限制:	无
异常:	无

32 位指令	
操作:	$Rz[31:0] = \text{abs}(Rx[7:0] - Ry[7:0]) + \text{abs}(Rx[15:8] - Ry[15:8]) + \text{abs}(Rx[23:16] - Ry[23:16]) + \text{abs}(Rx[31:24] - Ry[31:24])$
语法:	psabsa.u8 rz, rx, ry
说明:	以字节为单位, 将 Rx 的 4 个字节分别与 Ry 的 4 个字节相减, 减法结果取绝对值后链加, 链加结果存入 Rz。
影响标志位:	无影响
限制:	无
异常:	无

指令格式:

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 1 1 0 0 0	0 0 x 1 0	Rz	

图 15.144: PSABSA.U8

### 15.145 PSABSAA.U8——8 位并行有无号减法绝对值链加累加指令

统一化指令	
语法	psabsaa.u8 rz, rx, ry
操作	$Rz[31:0] = Rz[31:0] + \text{abs}(Rx[7:0] - Ry[7:0]) + \text{abs}(Rx[15:8] - Ry[15:8]) + \text{abs}(Rx[23:16] - Ry[23:16]) + \text{abs}(Rx[31:24] - Ry[31:24])$
编译结果	仅存在 32 位指令。 psabsaa.u8 rz, rx, ry

说明:	以字节为单位, 将 Rx 的 4 个字节分别与 Ry 的 4 个字节相减, 减法结果取绝对值后链加, 链加结果与 Rz 累加后存入 Rz。
影响标志位:	无影响
限制:	无
异常:	无

32 位指令	
操作:	$Rz[31:0] = Rz[31:0] + \text{abs}(Rx[7:0] - Ry[7:0]) + \text{abs}(Rx[15:8] - Ry[15:8]) + \text{abs}(Rx[23:16] - Ry[23:16]) + \text{abs}(Rx[31:24] - Ry[31:24])$
语法:	psabsaa.u8 rz, rx, ry
说明:	以字节为单位, 将 Rx 的 4 个字节分别与 Ry 的 4 个字节相减, 减法结果取绝对值后链加, 链加结果与 Rz 累加后存入 Rz。
影响标志位:	无影响
限制:	无
异常:	无
指令格式:	

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 1 1 0	Ry	Rx	1 1 1 0 0 0	0 1 x 1 0	Rz	

图 15.145: PSABSAA.U8

### 15.146 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 的值均被认为是有符号数。 注意，当 0x80000000_00000000 除以 0xFFFF FFFF 时，结果没有定义。
影 响 标志 位:	无影响
限 制:	无
异 常:	除零异常

<b>32 位指令</b>	
<b>操作 作:</b>	有符号除法 $\{Rz+1[31:0], Rz[31:0]\} = \{Rx+1[31:0], Rx[31:0]\} / Ry[31:0]$
<b>语 法:</b>	divsl32 rz, rx, ry
<b>说 明:</b>	{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 的值均被认为是有符号数。 注意，当 0x80000000_00000000 除以 0xFFFF FFFF 时，结果没有定义。
<b>影响 标志 位:</b>	无影响
<b>限 制:</b>	无
<b>异 常:</b>	除零异常

**指令格式:**

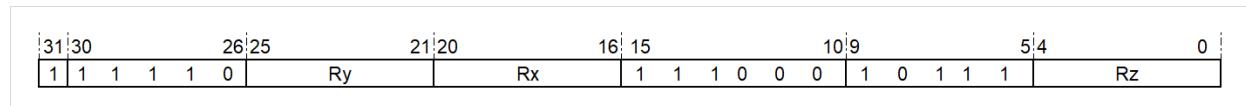


图 15.146: DIVSL

### 15.147 DIVUL——无符号长除法指令

<b>统一化指令</b>	
<b>语法</b>	divul rz, rx, ry
<b>操作</b>	无符号除法 $\{Rz+1[31:0], Rz[31:0]\} = \{Rx+1[31:0], Rx[31:0]\} / Ry[31:0]$
<b>编译结果</b>	仅存在 32 位指令 divul rz, rx, ry

<b>说明:</b>	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。
<b>影响标志:</b>	无影响
<b>志位:</b>	无
<b>限制:</b>	无
<b>异常:</b>	除零异常

<b>32 位指令</b>	
<b>操作:</b>	无符号除法 $\{Rz+1[31:0], Rz[31:0]\} = \{Rx+1[31:0], Rx[31:0]\} / Ry[31:0]$
<b>语法:</b>	divul rz, rx, ry
<b>说明:</b>	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。
<b>影响标志:</b>	无影响
<b>志位:</b>	无
<b>限制:</b>	无
<b>异常:</b>	除零异常
<b>指令格式:</b>	

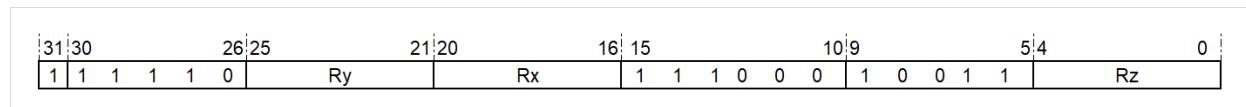


图 15.147: DIVUL

### 15.148 MULACA.S8——8 位并行有符号乘链加累加指令

<b>统一化指令</b>	
<b>语法</b>	mulaca.s8 rz, rx, ry
<b>操作</b>	$Rz = Rx[7:0] \times Ry[7:0] + Rx[15:8] \times Ry[15:8] + Rx[23:16] \times Ry[23:16] + Rx[31:24] \times Ry[31:24]$
<b>编译结果</b>	仅存在 32 位指令。 mulaca.s8 rz, rx, ry

<b>说明:</b>	以字节为单位, 将 Rx 的 4 个字节分别与 Ry 的 4 个字节相乘, 乘法结果链加厚存入 Rz。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

<b>32 位指令</b>	
<b>操作:</b>	$Rz = Rx[7:0] * Ry[7:0] + Rx[15:8] * Ry[15:8] + Rx[23:16] * Ry[23:16] + Rx[31:24] * Ry[31:24];$
<b>语法:</b>	mulaca.s8 rz, rx, ry
<b>说明:</b>	以字节为单位, 将 Rx 的 4 个字节分别与 Ry 的 4 个字节相乘, 乘法结果链加厚存入 Rz。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	无

**指令格式:**

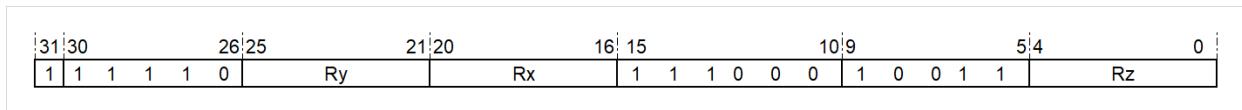


图 15.148: MULACA.S8

### 15.149 BLOOP——循环体加速指令

<b>统一化指令</b>	
<b>语法</b>	bloop rx, label1, label2
<b>操作</b>	Rx: 循环遍数寄存器 label1: 循环体的起始地址 label2: 循环体的终止地址 判定 Rx 是否等于 0, 若不等于 0, 反复执行【label1,label2】上的程序片段, 执行遍数由 Rx 的值确定。
<b>编译结果</b>	仅存在 32 位指令。 bloop rx, label1, label2

<b>说 明:</b>	典型循环体的最后两条指令为：循环遍数自减指令，比较跳转指令。BLOOP 指令用于循环体加速，作用是避免循环体中执行循环遍数自减指令与比较跳转指令。具体用法上，【label1,label2】上的程序片段不应该包括循环遍数寄存器自减指令、比较跳转指令。
<b>影响 标志 位:</b>	无
<b>限 制:</b>	【label1,label2】上的程序片段不应该修改 Rx，否则程序结果不可预期。 循环遍数 Rx 不大于 0x10000。
<b>异 常:</b>	无
<b>指 令 格 式:</b>	

<b>32 位指 令</b>	
<b>操 作:</b>	Rx：循环遍数寄存器 label1：循环体的起始地址 label2：循环体的终止地址 判定 Rx 是否等于 0，若不等于 0，反复执行【label 1,label2】上的程序片段，执行遍数由 Rx 的值确定。
<b>语 法:</b>	bloop rx, label1, label2
<b>说 明:</b>	典型循环体的最后两条指令为：循环遍数自减指令，比较跳转指令。BLOOP 指令用于循环体加速，作用是避免循环体中执行循环遍数自减指令与比较跳转指令。具体用法上，【label1,label2】上的程序片段不应该包括循环遍数寄存器自减指令、比较跳转指令。
<b>影响 标志 位:</b>	无
<b>限 制:</b>	【label1,label2】上的程序片段不应该修改 Rx，否则程序结果不可预期。 循环遍数 Rx 不大于 0x10000。
<b>异 常:</b>	无

**指令格式：**

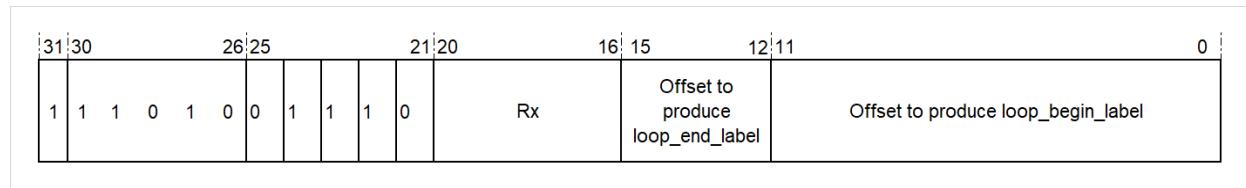


图 15.149: BLOOP

### 15.150 LDBI.W——地址自增的字加载指令

统一化指令	
语法	ldbi.w rz, (rx)
操作	Rz[31:0] <- mem(Rx) Rx[31:0] <- Rx[31:0] + 4
编译结果	仅存在 32 位指令。 ldbi.w rz, (rx)

说明:	从 Rx 对应的地址加载一个字存入 Rz，同时 Rx 加 4 后存入 Rx。
影响标志位:	无影响
限制:	无
异常:	访问错误异常

32 位指令	
操作:	Rz[31:0] <- mem(Rx) Rx[31:0] <- Rx[31:0] + 4
语法:	ldbi.w rz, (rx)
说明:	从 Rx 对应的地址加载一个字存入 Rz，同时 Rx 加 4 后存入 Rx。
影响标志位:	无影响
限制:	无
异常:	访问错误异常

指令格式：

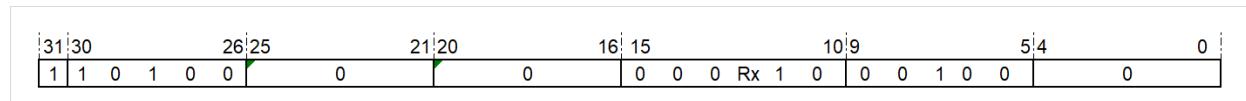


图 15.150: LDBI.W

### 15.151 LDBI.H——地址自增的无符号半字加载指令

<b>统一化指令</b>	
<b>语法</b>	ldbi.h rz, (rx)
<b>操作</b>	Rz[31:0] <- zero_extend(mem(Rx)) Rx[31:0] <- Rx[31:0] + 2
<b>编译结果</b>	仅存在 32 位指令。 ldbi.h rz, (rx)

<b>说明:</b>	从 Rx 对应的地址加载一个半字，高 16 位补 0 后存入 Rz，同时 Rx 加 2 后存入 Rx。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	访问错误异常

<b>32 位指令</b>	
<b>操作:</b>	Rz[31:0] <- zero_extend(mem(Rx)) Rx[31:0] <- Rx[31:0] + 2
<b>语法:</b>	ldbi.h rz, (rx)
<b>说明:</b>	从 Rx 对应的地址加载一个半字，高 16 位补 0 后存入 Rz，同时 Rx 加 2 后存入 Rx。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	访问错误异常

**指令格式:**

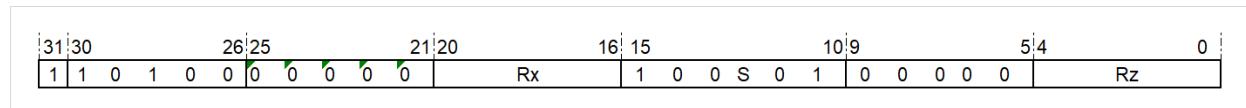


图 15.151: LDBI.H

### 15.152 LDBI.HS——地址自增的有符号半字加载指令

<b>统一化指令</b>	
<b>语法</b>	ldbi.hs rz, (rx)
<b>操作</b>	Rz[31:0] <- sign_extend(mem(Rx)) Rx[31:0] <- Rx[31:0] + 2
<b>编译结果</b>	仅存在 32 位指令。 ldbi.hs rz, (rx)

<b>说明:</b>	从 Rx 对应的地址加载一个半字，高 16 位进行符号位扩展，扩展后的结果存入 Rz，同时 Rx 加 2 后存入 Rx。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	访问错误异常

<b>32 位指令</b>	
<b>操作:</b>	Rz[31:0] <- sign_extend(mem(Rx)) Rx[31:0] <- Rx[31:0] + 2
<b>语法:</b>	ldbi.hs rz, (rx)
<b>说明:</b>	从 Rx 对应的地址加载一个半字，高 16 位进行符号位扩展，扩展后的结果存入 Rz，同时 Rx 加 2 后存入 Rx。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	访问错误异常

**指令格式:**

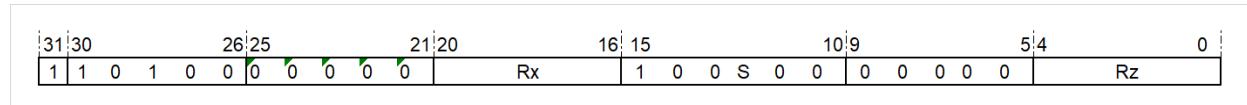


图 15.152: LDBI.HS

### 15.153 LDBI.B——地址自增的无符号字节加载指令

<b>统一化指令</b>	
<b>语法</b>	ldbi.b rz, (rx)
<b>操作</b>	Rz[31:0] <- zero_extend(mem(Rx)) Rx[31:0] <- Rx[31:0] + 1
<b>编译结果</b>	仅存在 32 位指令。 ldbi.b rz, (rx)

<b>说明:</b>	从 Rx 对应的地址加载一个字节，高 24 位补零后存入 Rz，同时 Rx 加 1 后存入 Rx。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	访问错误异常

32 位指令	
<b>操作:</b>	Rz[31:0] <- zero_extend(mem(Rx)) Rx[31:0] <- Rx[31:0] + 1
<b>语法:</b>	ldbi.b rz, (rx)
<b>说明:</b>	从 Rx 对应的地址加载一个字节，高 24 位补零后存入 Rz，同时 Rx 加 1 后存入 Rx。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	访问错误异常

**指令格式:**

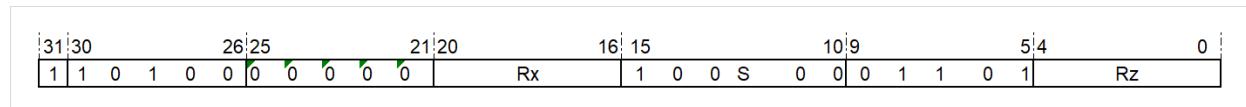


图 15.153: LDBI.B

### 15.154 LDBI.BS——地址自增的有符号字节加载指令

统一化指令	
<b>语法</b>	ldbi.bs rz, (rx)
<b>操作</b>	Rz[31:0] <- sign_extend(mem(Rx)) Rx[31:0] <- Rx[31:0] + 1
<b>编译结果</b>	仅存在 32 位指令。 ldbi.bs rz, (rx)

<b>说明:</b>	从 Rx 对应的地址加载一个字节，高 24 位进行符号位扩展，扩展后的结果存入 Rz，同时 Rx 加 1 后存入 Rx。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	访问错误异常

<b>32 位指令</b>	
<b>操作:</b>	$Rz[31:0] \leftarrow \text{sign\_extend}(\text{mem}(Rx))$ $Rx[31:0] \leftarrow Rx[31:0] + 1$
<b>语法:</b>	ldbi.bs rz, (rx)
<b>说明:</b>	从 Rx 对应的地址加载一个字节，高 24 位进行符号位扩展，扩展后的结果存入 Rz，同时 Rx 加 1 后存入 Rx。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	访问错误异常

**指令格式:**

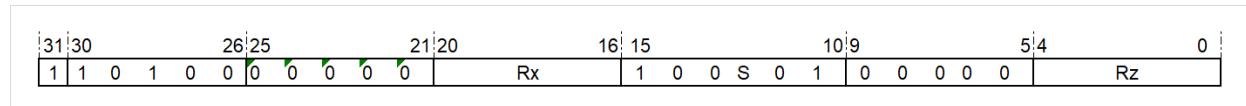


图 15.154: LDBI.BS

### 15.155 PLDBI.D——地址自增的双字加载指令

<b>统一化指令</b>	
<b>语法</b>	pldbi.d rz, (rx)
<b>操作</b>	$Rz[31:0] \leftarrow \text{mem}(Rx)$ $Rz+1[31:0] \leftarrow \text{mem}(Rx+4)$ $Rx[31:0] \leftarrow Rx[31:0] + 8$ 并预取 $\text{mem}(Rx+8)$ , $\text{mem}(Rx+12)$ 上的数据存放于处理器的预取缓存中
<b>编译结果</b>	仅存在 32 位指令。 pldbi.d rz, (rx)

<b>说明:</b>	从 Rx 对应的地址加载一个字存入 Rz，从 Rx+1 对应的地址加载一个字存入 Rz+1，同时 Rx 加 8 后存入 Rx。该指令同时会预取 mem(Rx+8), mem(Rx+12) 上的数据存放于处理器的预取缓存中。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	未对齐访问异常、访问错误异常
<b>指令格式:</b>	

32 位指令	
<b>操作:</b>	Rz[31:0] <- mem(Rx) Rz+1[31:0] <- mem(Rx+4) Rx[31:0] <- Rx[31:0] + 8 并预取 mem(Rx+8), mem(Rx+12) 上的数据存放于处理器的预取缓存中
<b>语法:</b>	pldbi.d rz, (rx)
<b>说明:</b>	从 Rx 对应的地址加载一个字存入 Rz，从 Rx+1 对应的地址加载一个字存入 Rz+1，同时 Rx 加 8 后存入 Rx。该指令同时会预取 mem(Rx+8), mem(Rx+12) 上的数据存放于处理器的预取缓存中。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	未对齐访问异常、访问错误异常

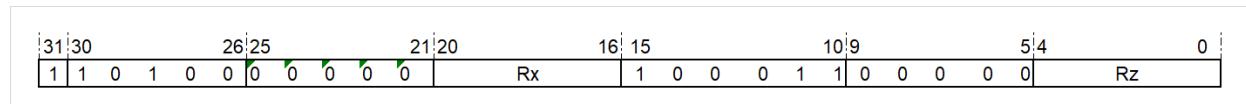
**指令格式:**

图 15.155: PLDBI.D

### 15.156 STBI.W——地址自增的字存储指令

<b>统一化指令</b>	
<b>语法</b>	stbi.w rz, (rx)
<b>操作</b>	mem(Rx) <- Rz[31:0] Rx[31:0] <- Rx[31:0] + 4
<b>编译结果</b>	仅存在 32 位指令。 stbi.w rz, (rx)

<b>说明:</b>	将 Rz 存入 Rx 对应的地址，同时 Rx 加 4 后存入 Rx。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	访问错误异常

<b>32 位指令</b>	
<b>操作:</b>	mem(Rx) <- Rz[31:0] Rx[31:0] <- Rx[31:0] + 4
<b>语法:</b>	stbi.w rz, (rx)
<b>说明:</b>	将 Rz 存入 Rx 对应的地址，同时 Rx 加 4 后存入 Rx。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	访问错误异常

指令格式：

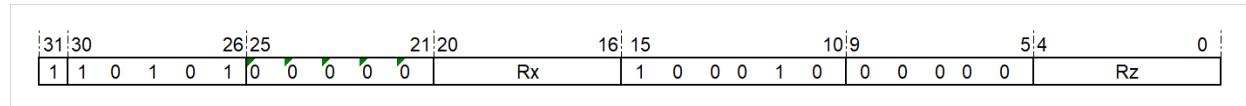


图 15.156: STBI.W

### 15.157 STBI.H——地址自增的半字存储指令

<b>统一化指令</b>	
<b>语法</b>	stbi.h rz, (rx)
<b>操作</b>	mem(Rx) <- Rx[15:0] Rx[31:0] <- Rx[31:0] + 2
<b>编译结果</b>	仅存在 32 位指令。 stbi.h rz, (rx)

<b>说明:</b>	将 Rz 的低半字存入 Rx 对应的地址，同时 Rx 加 2 后存入 Rx。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	访问错误异常

<b>32 位指令</b>	
<b>操作:</b>	mem(Rx) <- Rx[15:0] Rx[31:0] <- Rx[31:0] + 2
<b>语法:</b>	stbi.h rz, (rx)
<b>说明:</b>	将 Rz 的低半字存入 Rx 对应的地址，同时 Rx 加 2 后存入 Rx。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	访问错误异常

**指令格式:**

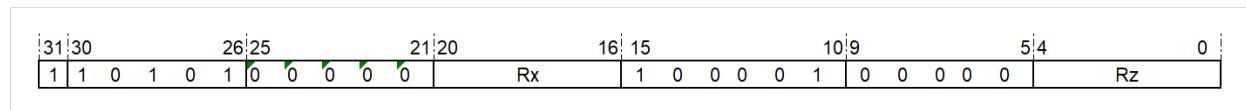


图 15.157: STBI.H

### 15.158 STBI.B——地址自增的字节存储指令

<b>统一化指令</b>	
<b>语法</b>	stbi.b rz, (rx)
<b>操作</b>	mem(Rx) <- Rx[7:0] Rx[31:0] <- Rx[31:0] + 1
<b>编译结果</b>	仅存在 32 位指令。 stbi.b rz, (rx)

<b>说明:</b>	将 Rz 的低 8 位存入 Rx 对应的地址，同时 Rx 加 1 后写入 Rx。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	访问错误异常

32 位指令	
<b>操作:</b>	mem(Rx) <- Rx[7:0] Rx[31:0] <- Rx[31:0] + 1
<b>语法:</b>	stbi.b rz, (rx)
<b>说明:</b>	将 Rz 的低 8 位存入 Rx 对应的地址，同时 Rx 加 1 后写入 Rx。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	访问错误异常

**指令格式:**

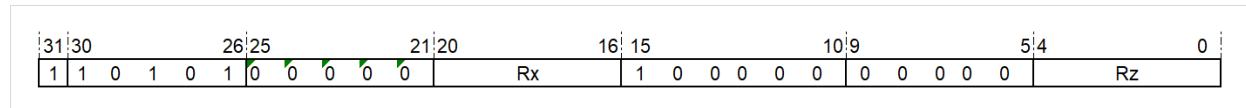


图 15.158: STBI.B

### 15.159 LDBIR.W——寄存器地址自增的字加载指令

统一化指令	
<b>语法</b>	ldbir.w rz, (rx), ry
<b>操作</b>	Rz[31:0] <- mem(Rx) Rx[31:0] <- Rx[31:0] + Ry[31:0]
<b>编译结果</b>	仅存在 32 位指令。 ldbir.w rz, (rx), ry

<b>说明:</b>	从 Rx 对应的地址加载一个字存入 Rz，同时 Rx 加 Ry 后存入 Rx。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	访问错误异常

32 位指令	
<b>操作:</b>	Rz[31:0] <- mem(Rx) Rx[31:0] <- Rx[31:0] + Ry[31:0]
<b>语法:</b>	ldbir.w rz, (rx), ry
<b>说明:</b>	从 Rx 对应的地址加载一个字存入 Rz，同时 Rx 加 Ry 后存入 Rx。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	访问错误异常

**指令格式：**

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 0 1 0 0	Ry	Rx	1 0 1 0 1 0	0 0 0 0 0 0	Rz	

图 15.159: LDBIR.W

### 15.160 LDBIR.H——寄存器地址自增的无符号半字加载指令

<b>统一化指令</b>	
<b>语法</b>	ldbir.h rz, (rx), ry
<b>操作</b>	$Rz[31:0] \leftarrow \text{zero\_extend}(\text{mem}(Rx))$ $Rx[31:0] \leftarrow Rx[31:0] + Ry[31:0]$
<b>编译结果</b>	仅存在 32 位指令。 ldbir.h rz, (rx), ry

<b>说明：</b>	从 Rx 对应的地址加载一个半字，高 16 位补 0 后存入 Rz，同时 Rx 加 Ry 后存入 Rx。
<b>影响标志位：</b>	无影响
<b>限制：</b>	无
<b>异常：</b>	访问错误异常

<b>32 位指令</b>	
<b>操作：</b>	$Rz[31:0] \leftarrow \text{zero\_extend}(\text{mem}(Rx))$ $Rx[31:0] \leftarrow Rx[31:0] + Ry[31:0]$
<b>语法：</b>	ldbir.h rz, (rx), ry
<b>说明：</b>	从 Rx 对应的地址加载一个半字，高 16 位补 0 后存入 Rz，同时 Rx 加 Ry 后存入 Rx。
<b>影响标志位：</b>	无影响
<b>限制：</b>	无
<b>异常：</b>	访问错误异常

**指令格式：**

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 0 1 0 0	Ry	Rx	1 0 1 S 0 1	0 0 0 0 0 0	Rz	

图 15.160: LDBIR.H

## 15.161 LDBIR.HS——寄存器地址自增的有符号半字加载指令

<b>统一化指令</b>	
<b>语法</b>	ldbir.hs rz, (rx), ry
<b>操作</b>	Rz[31:0] <- zero_extend(mem(Rx)) Rx[31:0] <- Rx[31:0] + Ry[31:0]
<b>编译结果</b>	仅存在 32 位指令。 ldbir.hs rz, (rx), ry

<b>说明:</b>	从 Rx 对应的地址加载一个半字，高 16 位进行符号位扩展，扩展后的结果存入 Rz，同时 Rx 加 Ry 后存入 Rx。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	访问错误异常

<b>32 位指令</b>	
<b>操作:</b>	Rz[31:0] <- zero_extend(mem(Rx)) Rx[31:0] <- Rx[31:0] + Ry[31:0]
<b>语法:</b>	ldbir.hs rz, (rx), ry
<b>说明:</b>	从 Rx 对应的地址加载一个半字，高 16 位进行符号位扩展，扩展后的结果存入 Rz，同时 Rx 加 Ry 后存入 Rx。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	访问错误异常

**指令格式:**

31:30	26:25	21:20	16:15	10:9	5:4	0
1 1 0 1 0 0	Ry	Rx	1 0 1 S 0 1	0 0 0 0 0	Rz	

图 15.161: LDBIR.HS

### 15.162 LDBIR.B——寄存器地址自增的无符号字节加载指令

<b>统一化指令</b>	
<b>语法</b>	ldbir.b rz, (rx), ry
<b>操作</b>	$Rz[31:0] \leftarrow \text{zero\_extend}(\text{mem}(Rx))$ $Rx[31:0] \leftarrow Rx[31:0] + Ry[31:0]$
<b>编译结果</b>	仅存在 32 位指令。 ldbir.b rz, (rx), ry

<b>说明:</b>	从 Rx 对应的地址加载一个字节，高 24 位补零后存入 Rz，同时 Rx 加 Ry 后存入 Rx。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	访问错误异常

<b>32 位指令</b>	
<b>操作:</b>	$Rz[31:0] \leftarrow \text{zero\_extend}(\text{mem}(Rx))$ $Rx[31:0] \leftarrow Rx[31:0] + Ry[31:0]$
<b>语法:</b>	ldbir.b rz, (rx), ry
<b>说明:</b>	从 Rx 对应的地址加载一个字节，高 24 位补零后存入 Rz，同时 Rx 加 Ry 后存入 Rx。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	访问错误异常

**指令格式:**

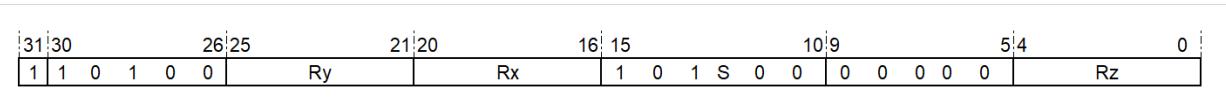


图 15.162: LDBIR.B

### 15.163 LDBIR.BS——寄存器地址自增的有符号字节加载指令

<b>统一化指令</b>	
<b>语法</b>	ldbir.bs rz, (rx), ry
<b>操作</b>	$Rz[31:0] \leftarrow \text{sign\_extend}(\text{mem}(Rx))$ $Rx[31:0] \leftarrow Rx[31:0] + Ry[31:0]$
<b>编译结果</b>	仅存在 32 位指令。 ldbir.bs rz, (rx), ry

<b>说明:</b>	从 Rx 对应的地址加载一个字节，高 24 位进行符号位扩展，扩展后的结果存入 Rz，同时 Rx 加 Ry 后存入 Rx。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	访问错误异常

<b>32 位指令</b>	
<b>操作:</b>	Rz[31:0] <- sign_extend(mem(Rx)) Rx[31:0] <- Rx[31:0] + Ry[31:0]
<b>语法:</b>	ldbir.bs rz, (rx), ry
<b>说明:</b>	从 Rx 对应的地址加载一个字节，高 24 位进行符号位扩展，扩展后的结果存入 Rz，同时 Rx 加 Ry 后存入 Rx。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	访问错误异常

**指令格式:**

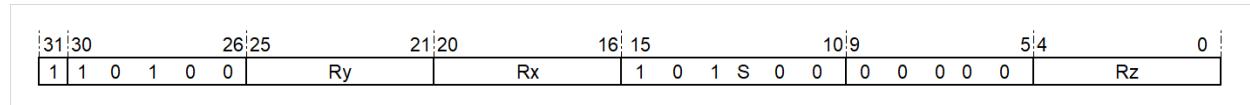


图 15.163: LDBIR.BS

### 15.164 PLDBIR.D——寄存器地址自增的双字加载指令

<b>统一化指令</b>	
<b>语法</b>	pldbir.d rz, (rx) , ry
<b>操作</b>	Rz[31:0] <- mem(Rx) Rz+1[31:0] <- mem(Rx+Ry) Rx[31:0] <- Rx[31:0] + 2XRy 并预取 mem(Rx+2XRy), mem(Rx+3XRy) 上的数据存放于处理器的预取缓存中
<b>编译结果</b>	仅存在 32 位指令。 pldbir.d rz, (rx) , ry

<b>说明:</b>	从 Rx 对应的地址加载一个字存入 Rz，从 Rx+Ry 对应的地址加载一个字存入 Rz+1，同时 Rx 加 2XRy 后存入 Rx。该指令同时会预取 mem(Rx+2XRy), mem(Rx+3XRy) 上的数据存放于处理器的预取缓存中。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	未对齐访问异常、访问错误异常

<b>32 位指令</b>	
<b>操作:</b>	Rz[31:0] <- mem(Rx) Rz+1[31:0] <- mem(Rx+Ry) Rx[31:0] <- Rx[31:0] + 2XRy 并预取 mem(Rx+2XRy), mem(Rx+3XRy) 上的数据存放于处理器的预取缓存中
<b>语法:</b>	pldbir.d rz, (rx), ry
<b>说明:</b>	从 Rx 对应的地址加载一个字存入 Rz，从 Rx+Ry 对应的地址加载一个字存入 Rz+1，同时 Rx 加 2XRy 后存入 Rx。该指令同时会预取 mem(Rx+2XRy), mem(Rx+3XRy) 上的数据存放于处理器的预取缓存中。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	未对齐访问异常、访问错误异常

### 指令格式:

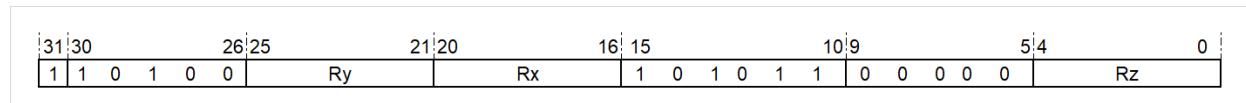


图 15.164: PLDBIR.D

## 15.165 STBIR.W——寄存器地址自增的字存储指令

<b>统一化指令</b>	
<b>语法</b>	stbir.w rz, (rx) , ry
<b>操作</b>	mem(Rx) <- Rx[31:0] Rx[31:0] <- Rx[31:0] + Ry[31:0]
<b>编译结果</b>	仅存在 32 位指令。 stbir.w rz, (rx) , ry

<b>说明:</b>	将 Rz 存入 Rx 对应的地址，同时 Rx 加 Ry 后存入 Rx。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	访问错误异常

<b>32 位指令</b>	
<b>操作:</b>	mem(Rx) <- Rx[31:0] Rx[31:0] <- Rx[31:0] + Ry[31:0]
<b>语法:</b>	stbir.w rz, (rx) , ry
<b>说明:</b>	将 Rz 存入 Rx 对应的地址，同时 Rx 加 Ry 后存入 Rx。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	访问错误异常

**指令格式:**

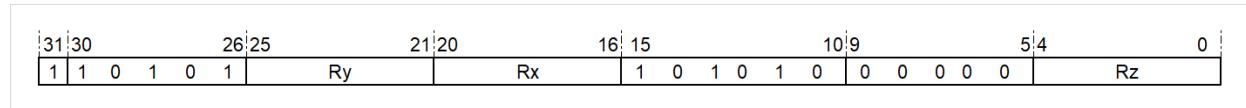


图 15.165: STBIR.W

## 15.166 STBIR.H——寄存器地址自增的半字存储指令

<b>统一化指令</b>	
<b>语法</b>	stbir.h rz, (rx) , ry
<b>操作</b>	mem(Rx) <- Rx[15:0] Rx[31:0] <- Rx[31:0] + Ry[31:0]
<b>编译结果</b>	仅存在 32 位指令。 stbir.h rz, (rx) , ry

<b>说明:</b>	将 Rz 的低半字存入 Rx 对应的地址，同时 Rx 加 Ry 后存入 Rx。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	访问错误异常

<b>32 位指令</b>	
<b>操作:</b>	mem(Rx) <- Rx[15:0] Rx[31:0] <- Rx[31:0] + Ry[31:0]
<b>语法:</b>	stbir.h rz, (rx) , ry
<b>说明:</b>	将 Rz 的低半字存入 Rx 对应的地址，同时 Rx 加 Ry 后存入 Rx。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	访问错误异常

**指令格式:**

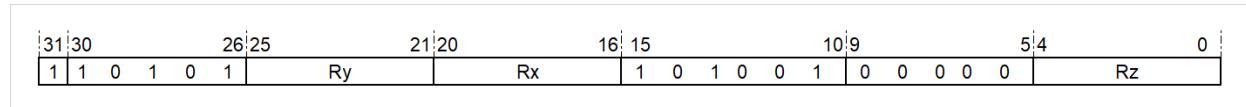


图 15.166: STBIR.H

### 15.167 STBIR.B——寄存器地址自增的字节存储指令

<b>统一化指令</b>	
<b>语法</b>	stbir.b rz, (rx) , ry
<b>操作</b>	mem(Rx) <- Rx[7:0] Rx[31:0] <- Rx[31:0] + Ry[31:0]
<b>编译结果</b>	仅存在 32 位指令。 stbir.b rz, (rx) , ry

<b>说明:</b>	将 Rz 的低 8 位存入 Rx 对应的地址，同时 Rx 加 Ry 后存入 Rx。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	访问错误异常

32 位指令	
<b>操作:</b>	mem(Rx) <- Rx[7:0] Rx[31:0] <- Rx[31:0] + Ry[31:0]
<b>语法:</b>	stbir.b rz, (rx) , ry
<b>说明:</b>	将 Rz 的低 8 位存入 Rx 对应的地址，同时 Rx 加 Ry 后存入 Rx。
<b>影响标志位:</b>	无影响
<b>限制:</b>	无
<b>异常:</b>	访问错误异常

**指令格式:**

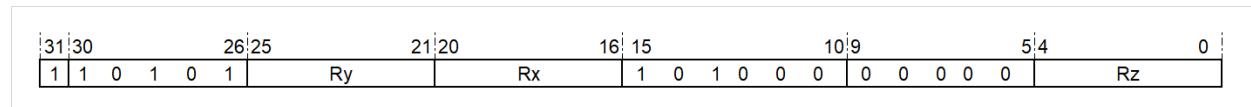


图 15.167: STBIR.B