XUANTIE

C908X

用户手册 (xrvm)

此手册支持的 CPU 版本为 R0S2

修订版 01

2025-11-14

杭州中天微系统有限公司 C-SKY Microsystems Co., Ltd.

Address: Room 201, 2/F, Building 5, No.699 Wangshang Road, Changhe Street, Binjiang District, Hangzhou, Zhejiang, China

Website: www.xrvm.cn

杭州中天微系统有限公司 C-SKY Microsystems Co., Ltd.

地址: 中国浙江省杭州市滨江区长河街道网商路 699 号 5 号楼 2 楼 201 室

网址:www.xrvm.cn

版本历史

版本	描述	日期
01	初版	2025-10-27

目录

图目录

表目录

1 概述

1.1 简介

C908X 是基于 RISC-V 指令架构的 64 位高性能多核心处理器，主要面向市场日益增强的图像、视觉处理领域，例如智慧视觉、车载视觉、行车记录仪、智能交互等。其他应用领域还包括扫地机器人、无人机、自动化驾驶、增强现实、医疗图像、机器人工业视觉、移动互联网等产品。

C908X 采用同构多核架构，支持多 cluster，每个 cluster 支持 1~4 个核心。每个 C908X 核心采用自主设计的微体系结构，并重点针对性能进行优化，采用按序双发射、多模式分支预测和多通道数据预取等高性能技术。此外，C908X 核心支持实时检测并关断内部空闲功能模块，降低处理器动态功耗。

1.2 特点

1.2.1 C908X 处理器体系结构的主要特点

- 同构多核架构，支持多 cluster，每个 cluster 支持 1~4 个 C908X 核心；
- 支持 1 个 AXI4.0 Master 接口，128/256 比特的总线宽度；
- 支持 1 个可配置的 AXI4.0 低延时外设 Master 接口（Low Latency Port, LLP），128 比特的总线宽度；
- 支持 1 个可配的 AXI4.0 设备一致性接口 (Device Coherence Port, DCP)，128 比特的总线宽度；
- 一级指令/数据缓存分别支持 16KB/32KB/64KB，缓存行 SIZE 为 64B；可配置 ECC/奇偶校验机制；
- 二级高缓 128KB/256KB/512KB/1MB/1.5MB/2MB/3MB/4MB，缓存行 SIZE 为 64B；可配置 ECC 校验机制；
- 一级缓存支持 MESI 一致性协议，二级缓存支持 MOESI 一致性协议；
- 支持私有中断控制器 CLINT 和公有中断控制器 PLIC；支持多 cluster 中断分发；
- 支持 RISC-V 性能计数器和计时器；
- 支持 Sv39 和 Sv48 内存管理，支持 SVNAPOT 和 SVPBMT 标准扩展；
- 支持 8/16/32/64 表项 PMP，支持 ePMP；
- 支持 XuanTie TEE 扩展；
- 支持各个核心独立下电以及 cluster 下电；
- 支持 RISC-V 调试框架和 RISC-V Trace，支持多核多 cluster 调试；

备注

C908X 对 RV64 的支持说明：

- 在机器模式（M Mode）和超级用户模式（S Mode）下，只支持 RV64
  - 在机器模式下只需考虑 RV64 的工作模式
- 在用户模式（U Mode）下支持 RV64

1.2.2 C908X 核心的主要特点

- RISC-V 64GCB[V] 指令架构；
- User Mode 支持 RV64 指令集；
- 支持小端模式；
- 9 级流水架构；
- 按序双发射，按序取指、发射、执行和退休；
- 两级 TLB 内存管理单元，实现虚实地址转换与内存管理；
- 指令高缓和数据高缓大小可配置，支持 16KB/32KB/64KB，缓存行为 64B；
- 指令高缓可配置奇偶校验，数据高缓可配置 ECC 或奇偶校验；
- 指令预取功能，硬件自动检测并动态启动；
- 指令高缓路预测的低功耗访问技术；
- 支持 2KB/4KB/8KB 的多算法分支预测器；
- 支持 256 表项的分支目标缓存器（BTB）；
- 支持 8 层的硬件返回地址堆栈；
- 支持 256 表项的间接跳转分支预测器；
- 支持循环终止预测；
- 支持指令融合技术；
- 双发射按序执行 Load、Store 指令；
- 读、写内存分别支持 8 路、12 路并发的总线访问；
- 支持写合并；
- 支持 8 通道数据预取，支持固定 stride 和规律性不定 stride 数据预取；

1.2.3 矢量计算单元的主要特点

- 遵循 RISC-V V 矢量扩展；
- 在 4 核、2GHz 最大配置下，算力可达 2048 Gops (@int8)/ 1024 GFlops (@FP16)；

- 矢量执行单元支持 FP16/BFP16/FP32 浮点和 INT8/INT16/INT32/INT64 整型的矢量运算；
- 支持 512/1024/4096 可配置的矢量寄存器位宽 VLEN；
- 支持 512/1024/4096 位矢量数据存储访问位宽；
- 支持 segment load、store 指令；
- 性能优化的非对齐内存访问；

1.3 可配置选项

C908X 的可配置选项如 表 1.1 所示。

表 1.1: C908X 可配置选项

功能	配置选项	详细描述
C908X Cluster		
核心数量	1/2/3/4	C908X 提供 1-4 个核心可配
VPU	VL512/VL1024/ VL4096	矢量计算单元支持 VLEN 可配
BHT	Pro	BHT 实现方式：Pro（tage）
MMU	SV39/SV48	支持 SV39 或 SV48 模式，配置为 SV48 可以同时支持 SV39
jTLB entry	512/1024	jTLB 表项数
Master 接口协议	AXI	Master 接口支持 AXI 协议
LLP	有/无	低延时外设端口可选
DCP	有/无	用于外设对片上高速缓存的访问，实现数据一致性，可用于连接 DMA
L1 I-Cache	16K/32K/64K	可以配置 16KB、32KB、64KB
L1 D-Cache	16K/32K/64K	可以配置 16KB、32KB、64KB
L1 ECC/Parity	有/无	L1 I-Cache 的 Parity 校验、L1 D-Cache 的 ECC 校验
L2 Cache	128K/ 256K/512K/1M/ 1.5M/2M/3M/4M	可以配置 128KB~4MB
L2 ECC	有/无	L2 Tag/Data RAM 的 ECC 校验
PMP 区域个数	8/16/32/64	PMP 区域个数可配
ePMP	有/无	Enhanced PMP 功能可配置
TEE	有/无	TEE 扩展可选

续下页

表 1.1 - 接上页

功能	配置选项	详细描述
L2 RAM 延时	支持配置	Tag RAM Setup: No flop/Flop Tag RAM Access: 1~5 cycles Data RAM Setup: No flop/Flop Data RAM Access: 1~8 cycles
调试资源级别	最小/典型/最大	推荐“典型”配置。详见表格下方注释。
Trace 功能	无/有	Trace 扩展功能可选
System Bus Access (SBA)	无/有	可以配置单独的 AXI 总线接口，用于调试器绕过 CPU 独立访问内存空间。
pic_top （外置的中断控制器）		
中断数量	32~1024，step 32	PLIC 支持的可接入中断数量，范围为 32-1024 ，且需设置为 32 的倍数
Cluster 数量	1~16	共享同一个 pic_top 的 Cluster 数量
Hart 数量	1~256	共享同一个 pic_top 的 Hart 数量，即核心的数量。注意：不需要说明 Hart 与 Cluster 的对应关系。
TEE	有/无	TEE 扩展可选
tdt_dmi_top （调试转换桥，JTAG 转 APB）		
APB 端口数量	1~32	同一个 tdt_dmi_top 可以调试多个 Cluster，一个 APB 端口对应一个 Cluster
Sys APB Access	有/无	Sys APB Access 允许 CPU 通过 Master Port 和系统总线访问调试寄存器
Trace 模块设置（tdt_trace_top）		
Master Number	1 ~ 30	用于配置 tdt_trace_top 模块 Master 的数量
PIB SINK	有/无	用于配置接收 Trace Message 的 Slave 种类
ATB_BRIDGE	有/无	
SBA_SINK	有/无	
SRAM_SINK	有/无	
Trace SRAM SINK Size	0K/2K/4K/8K/ 16K/32K/64K	用于配置 tdt_trace_top 内 SRAM SINK 的大小

注释：

- Cluster 和 pic_top 两处的 TEE 选项必须保持一致。
- 调试资源级别：

最小配置：1 个 trigger，可配置为指令地址类型，支持全等匹配、低位掩码匹配；

典型配置：3 个 trigger，每个 trigger 可配置为指令地址类型、访存地址类型，支持全等匹配、低位掩码匹配；

最大配置：8 个 trigger，每个 trigger 可配置为指令地址类型、指令数据类型、访存地址类型、访

存数据类型，支持全等匹配、低位掩码匹配、大于等于、小于、掩码匹配低位、掩码匹配高位。此配置下还支持设置两个 trigger 成组，当两个 trigger 的设置条件同时满足时 trigger 触发。2 个中断异常 trigger，每个 trigger 可配置为匹配中断或匹配异常两种模式。

具体内容请参照*调试*章节。

1.4 玄铁架构扩展技术

C908X 兼容于玄铁 C 系列 1.0 扩展架构，具体包括：

- 运算指令扩展：在整型、浮点、load/store 等方面提高运算能力，是 RISC-V 基础指令集的有力补充
- Cache 操作扩展：方便地进行 Cache 维护操作，提高 Cache 效率
- 内存模型扩展：高效管理地址属性，提高内存访问效率
- 控制寄存器扩展：在标准 RISC-V 的基础上提供更加丰富的功能
- 多核同步指令扩展：提高多核一致性维护的效率

1.5 版本说明

C908X 兼容 RISC-V 标准，具体信息如下 表 1.2 和 表 1.3 所示。

表 1.2: C908X 编程模型及对应版本

Specification	C908X
RISC-V Profile	RVA22, Version 0.8, October 25, 2022
RISC-V Instruction Set Maunal Volume I: User-Level ISA	User-Level ISA (20191213 Ratified)
RISC-V Instruction Set Manual Volume II: Privileged Architecture	Version 20211203
RISC-V “V” Vector Extension	Version 1.0
RISC-V Bit-Manipulation ISA-extensions	Version 1.0.0-38-g865e7a7, 2021-06-28: Release candidate
RISC-V Cryptography Extensions Volume I Scalar & Entropy Source Instructions	Version v1.0.0, 2’nd December, 2021: Ratified 少部分（Zbkc, Zkt）支持
RISC-V External Debug Support V0.13.2	Version 0.13.2
RISC-V Nexus Trace Specification	Version 1.0.0_rc9, Sep 11, 2023: Frozen state
RISC-V platform-level interrupt controller (PLIC)	Version 1.0
PMP Enhancements for memory access and execution prevention on Machine mode (Smepmp)	Version 1.0, 12/2021
玄铁扩展指令集	支持

表 1.3: C908X 指令集及对应版本

Specification	Modules	C908X
RISC-V Instruction Set Maunal Volume I: User-Level ISA	Width of an integer register in bits (XLEN)	RV64 + RV32 (User Mode)
	Control and Status Register (CSR) Instructions (Zicsr)	Version 2.0
	Instruction-Fetch Fence (Zifencei)	Version 2.0
	Standard Extensions for Half-Precision Floating-Point (Zfh, Zfhmin)	Version 1.0 (Zfh)
	Pause Hint (Zihintpause)	Version 2.0
	Standard Extension for Base Counters and Timers (Zicntr)	Version 2.0
	Standard Extension for Hardware Performance Counters (Zihpm)	支持
RISC-V Instruction Set Manual Volume II: Privileged Architecture	Virtual Memory System	SV39 + SV48
	NAPOT Translation Contiguity (Svnapot)	Version 1.0
	Page-Based Memory Types (Svpbmt)	Version 1.0
	Fine-Grained Address-Translation Cache Invalidation (Svinval)	Version 1.0
	“stimecmp / vstimecmp” Extension (Sstc)	Version 0.5.4-3f9ed34, 2021-10-13: frozen
	Base Cache Management Operation ISA Extensions (Zicbom, Zicboz, Zicbop)	Version 1.0.1-b34ea8a, 2022-05-13: Ratified
RISC-V “V” Vector Extension	Vector AMO Operations (Zvamo)	Version 0.10 (also 1.0-draft-20210129)
	Vector Extension for Half-Precision Floating-Point(Zvfh, Zfhmin)	Zvfh
RISC-V Bit-Manipulation ISA-extensions	Bit-manipulation (Zba, Zbb, Zbc, Zbs)	Zba, Zbb, Zbc, Zbs
RISC-V Cryptography Extensions Volume I Scalar & Entropy Source Instructions	RISC-V Cryptography Extensions Volume I: Scalar & Entropy Source Instructions	Zbkc, Zkt

1.6 命名规则

1.6.1 符号

本文档用到的标准符号和操作符如 图 1.1 所示。

符号	功能
+	加
-	减
*	乘
/	除
>	大于
<	小于
=	等于
≥	大于或等于
≤	小于或等于
≠	不等于
&	与
|	或
⊕	异或
NOT	取反
:	连接
⇒	传输
⇔	交换
±	误差
0b0011	二进制数
0x0F	十六进制数
rd	整型目的寄存器
rs1	整型源寄存器1
rs2	整型源寄存器2
rs3	整型源寄存器3
fd	浮点目的寄存器
fs1	浮点源寄存器1
fs2	浮点源寄存器2
fs3	浮点源寄存器3
vd	矢量目的寄存器
vs1	矢量源寄存器1
vs2	矢量源寄存器2
vs3	矢量源寄存器3

图 1.1: 符号列表

1.6.2 术语

- **逻辑 1** 是指对应于布尔逻辑真的电平值。
- **逻辑 0** 是指对应于布尔逻辑伪的电平值。
- **置位**是指使得某个或某几个位达到逻辑 1 对应的电平值。
- **清除**是指使得某个或某几个位达到逻辑 0 对应的电平值。
- **保留位**是为功能的扩展而预留的，没有特殊说明时其值为 0。
- **信号**是指通过它的状态或状态间的转换来传递信息的电气值。

- **引脚**是表示一种外部电气物理连接，同一个引脚可以连接多个信号。
- **使能**是指使某个离散信号处在有效的状态：
  - 低电平有效信号从高电平切换到低电平；
  - 高电平有效信号从低电平切换到高电平。
- **禁止**是指使某个处在使能状态的信号状态改变：
- 低电平有效信号从低电平切换到高电平；
- 高电平有效信号从高电平切换到低电平。
- **LSB** 代表最低有效位，**MSB** 代表最高有效位。
- **信号**、**位域**、**控制位**的表示都使用一种通用的规则。
- **标识符后来跟着表示范围的数字**，从高位到低位表示一组信号。

比如“addr[4:0]”就表示一组地址总线，最高位是 addr[4]，最低位是 addr[0]。

- **单个的标识符**就表示单个信号。

  例如“pad_cpu_rst_b”就表示单独的一个信号。

  有时候会在标识符后加上数字表示一定的意义，比如 addr15 就表示一组总线中的第 16 位。

2 处理器简介

2.1　结构框图

C908X 结构框图如 图 2.1 所示。

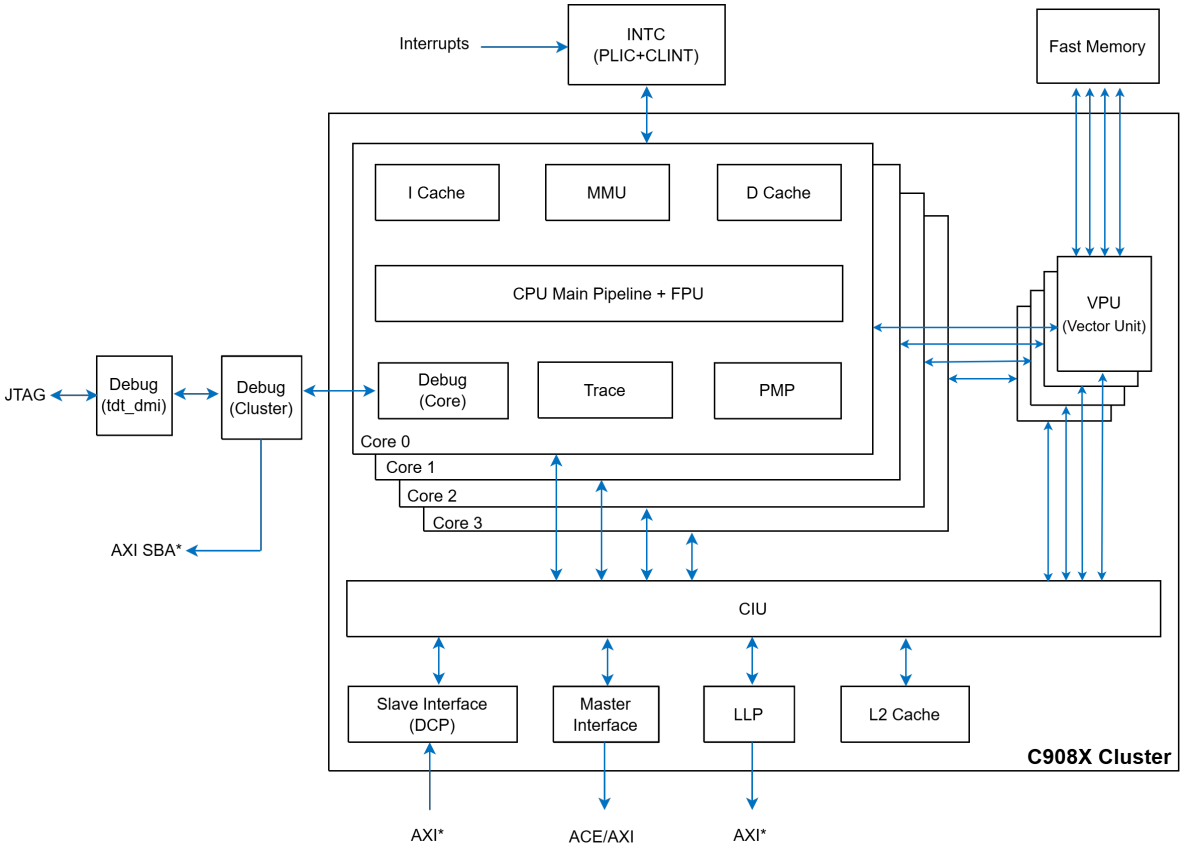


图 2.1: C908X 微体系结构图

备注

C908X 矢量执行单元（VPU）支持 RISC-V Vector 1.0 指令集和玄铁自定义矢量扩展指令集。

2.2　核内子系统

C908X 核内子系统主要包含：指令提取单元（IFU）、指令执行单元（IEU）、浮点执行单元（FPU）、存储载入单元（LSU）、虚拟内存管理单元（MMU）、物理内存保护单元（PMP）和矢量执行单元。

2.2.1　指令提取单元

指令提取单元（IFU），一次可最多提取 4 条指令并对其并行处理。实现了多项技术以提高访问效率，比如指令高缓路预测，指令暂存器，直接/间接分支预测等。整个指令提取单元具有低功耗，高分支预测准确率，高指令预取效率的特点。

2.2.2　指令执行单元

指令执行单元（IEU）可以同时译码和发射两条指令。其包含算术逻辑单元（ALU）、乘法单元（MULT）、除法单元（DIV）和跳转单元（BJU）。ALU 执行 32 或 64 位整型和位扩展操作。MULT 支持 16*16、32*32、64*64 整数乘法。除法器的设计采用了基 4 或基 16 的 SRT 算法，执行周期视操作数而变化。BJU 可以在单周期内完成分支预测错误处理。指令执行单元可以同时退休两条指令。

2.2.3　浮点执行单元

浮点单元包含浮点算术逻辑单元（FALU）、浮点融合乘累加单元（FMAU）和浮点除法开方单元（FDSU），支持半精度、单精度运算。浮点算术逻辑单元（FALU）负责加减、比较、转换、寄存器传输、符号注入、分类等操作。浮点融合乘累加单元 (FMAU) 负责普通乘法、融合乘累加等操作。浮点除法开方单元（FDSU）负责浮点除法、浮点开方等操作。

2.2.4　存储载入单元

存储载入单元（LSU）支持标量存储/加载指令的双发射、矢量存储/加载指令的单发射，支持高速缓存非阻塞访问。支持字节、半字、字、双字和四字的存储/载入指令，并支持字节和半字的载入指令的符号位和零扩展。存储/加载指令可以流水执行，使得数据吞吐量达到一个周期存取一个数据。支持 8 路数据流硬件预取技术，将数据提前放入 L1 数据高缓中。当数据高缓缺失后，支持总线的并行访问。

2.2.5　虚拟内存管理单元

虚拟内存管理单元（MMU）遵从 RISC-V SV39/SV48 标准，将 39/48 位虚拟地址转换为 40 位物理地址。C908X MMU 在 SV39/SV48 定义的硬件回填标准基础上，扩展了软件回填方式和地址属性。

具体信息参考*内存模型*。

2.2.6　物理内存保护单元

C908X 物理内存保护单元（PMP）遵从 RISC-V 标准，支持配置 8/16/32/64 个表项，最小粒度为 4KB，不支持 NA4 模式。在标准 PMP 的基础上，客户还可以选配 ePMP（Enhanced PMP）功能。

具体信息参考*内存模型*。

2.2.7　矢量执行单元（VPU）

矢量执行单元支持 RISC-V Vector 1.0 指令集和玄铁自定义矢量扩展指令集（EXP/TANH/SIGMOID/REC）。矢量执行单元包括矢量浮点执行单元和矢量整形执行单元。采用矢量和标量流水线解耦的设计架构，支持高性能、弹性的矢量执行单元设计，可支持 1024 bit 数据宽度。矢量执行单元采用多通道设计，每个通道包含独立的访存流水线和计算流水线。支持高性能矢量访存设计，可以容忍更长的访问延时。配置 512/1024/4096 bit AXI4.0 高带宽矢量专用访问总线，提供更大的访问带宽。

2.3　多核子系统

C908X 多核子系统包含：数据一致性接口单元（CIU，CPU Interface Unit）、二级高速缓存、主设备接口单元、可配置的 AXI4.0 设备一致性接口 (DCP, Device Coherence Port)、矢量专用总线接口、低延时外设接口 (LLP, Low Latency Port)。

2.3.1　数据一致性接口单元

数据一致性接口单元（CIU），采用 MESI 协议维护各个 L1 数据高缓的一致性。设置两路监听缓冲器，可并行处理多个监听请求，最大化利用监听带宽。另外，CIU 单元还支持 TLB 和 ICACHE 无效操作请求的广播，简化了 TLB/ICache 与 DCache 数据一致性的软件维护成本。

2.3.2　二级高速缓存

二级高速缓存，紧耦合于 CIU 单元，实现和 L1 数据高缓的同步访问；支持分块或不分块的流水线架构，单周期可并行处理两个访问请求，最大访问带宽可达到 1024 比特。二级高速缓存采用和 C908X 相同的工作频率，TAG RAM 和 DATA RAM 的访问延时可以由软件配置。二级高速缓存支持数据快速返回机制，当二级高速缓存命中时，可直接将数据通过旁路发给核心。

2.3.3　主设备接口

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

2.3.4　设备一致性接口

设备一致性接口（DCP）支持 AXI4.0 协议，可用于外设对片上高速数据缓存的访问，用硬件的方式实现 I/O 数据一致性，可以用来连接外部 DMA。

2.3.5　矢量专用总线接口

每个矢量执行单元配置一个矢量专用总线接口。当配置四个 C908X 核心搭载四个矢量执行单元时，通出四个矢量专用总线接口。每个矢量专用总线接口支持 AXI4.0，数据宽度和 VLEN 一致。当 VLEN 为 1024 比特时，数据位宽对应为 1024 比特。

2.3.6　低延时外设接口

低延时外设接口（LLP）支持 AXI4.0 协议，可以作为一个专用接口来访问系统外设。

2.4　多 cluster 子系统

C908X 多 cluster 子系统包括：中断控制器（PIC）、计时器和自定义多 cluster 多核单端口调试框架。

2.4.1　中断控制器

中断控制器（PIC）包括平台级中断控制器（PLIC）和处理器核局部中断控制器（CLINT）。平台级中断控制器（PLIC）支持最多 1023 个外部中断源采样和分发，支持电平和脉冲中断，可以设置 32 个级别的中断优先级。处理器核局部中断控制器（CLINT）用于处理软件中断和计时器中断。C908X 的中断控制器（PIC）采用外置式设计，可以支持在多个 cluster 之间处理外部中断和局部中断。

具体信息参考 *中断控制器*。

2.4.2　计时器

多 cluster 多核系统中共用一个 64 位系统计时器，各个核心拥有私有的计时器比较值寄存器，通过采集系统计时器的数值与软件设置的私有计时器比较值寄存器进行比较，产生计时器信号。

具体信息参考 *中断控制器*。

2.4.3　调试、追踪系统

C908X 支持和 RISC-V Trace 和 Debug，采用多 cluster 多核单端口调试框架，通过一个共享的 JTAG 接口访问各个 cluster 的调试单元（DM），控制核进出调试模式和访问处理器资源。JTAG 接口和调试单元（DM）支持 RISC-V debug V0.13.2 协议标准。

具体信息参考 *调试* 和*Trace* 。

2.5　接口概览

C908X 的接口按照功能主要分为：时钟复位、总线系统、中断系统、调试系统、低功耗系统、DFT 系统和 CPU 运行观测信号接口。

C908X 的主要接口如 图 2.2 所示。

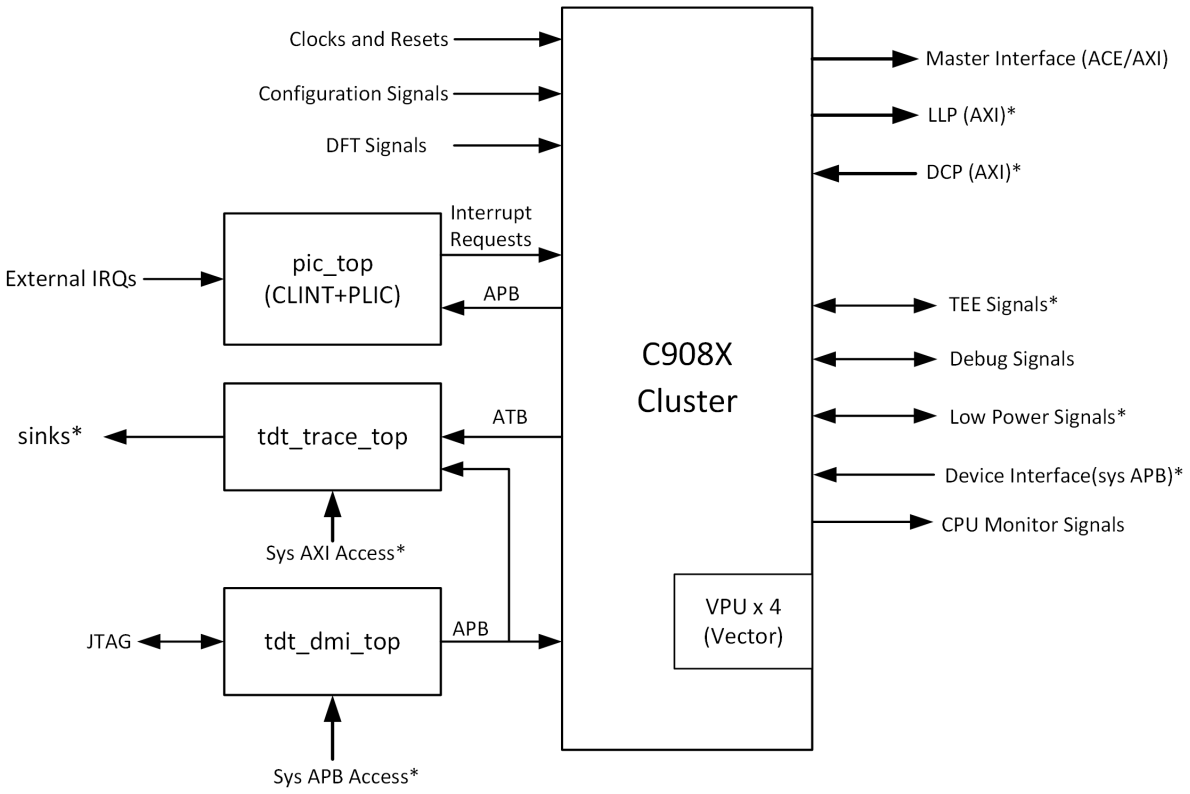


图 2.2: C908X 接口概览

3 指令集

本章主要介绍 C908X 中实现的指令集，分为两大部分：RV 基础指令集和玄铁扩展指令集。

3.1　RV 基础指令集

3.1.1　整型指令集（RV64I）

整型指令集按功能可以分为以下类型:

- 加减法指令
- 逻辑操作指令
- 移位指令
- 比较指令
- 数据传输指令
- 分支跳转指令
- 内存存取指令
- 控制寄存器操作指令:
- 低功耗指令
- 异常返回指令
- 特殊功能指令

表 3.1: 整型指令（RV64I）指令列表

指令名称	指令描述	执行延时
加减法指令		
ADD	有符号加法指令	1
ADDW	低 32 位有符号加法指令	1
ADDI	有符号立即数加法指令	1
ADDIW	低 32 位有符号立即数加法指令	1

续下页

表 3.1 - 接上页

指令名称	指令描述	执行延时
SUB	有符号减法指令	1
SUBW	低 32 位有符号减法指令	1
逻辑操作指令		
AND	按位与指令	1
ANDI	立即数按位与指令	1
OR	按位或指令	1
ORI	立即数按位或指令	1
XOR	按位异或指令	1
XORI	立即数按位异或指令	1
移位指令		
SLL	逻辑左移指令	1
SLLW	低 32 位逻辑字左移指令	1
SLLI	立即数逻辑左移指令	1
SLLIW	低 32 位立即数逻辑左移指令	1
SRL	逻辑右移指令	1
SRLW	低 32 位逻辑右移指令	1
SRLI	立即数逻辑右移指令	1
SRLIW	低 32 位立即数逻辑右移指令	1
SRA	算术右移指令	1
SRAW	低 32 位算数右移指令	1
SRAI	立即数算术右移指令	1
SRAIW	低 32 位立即数算数右移指令	1
比较指令		
SLT	有符号比较小于置位指令	1
SLTU	无符号比较小于置位指令	1
SLTI	有符号立即数比较小于置位指令	1
SLTIU	无符号立即数比较小于置位指令	1
数据传输指令		
LUI	高位立即数装载指令	1
AUIPC	PC 高位立即数加法指令	1
分支跳转指令		
BEQ	相等分支指令	1
BNE	不等分支指令	1

续下页

表 3.1 - 接上页

指令名称	指令描述	执行延时
BLT	有符号小于分支指令	1
BGE	有符号大于等于分支指令	1
BLTU	无符号小于分支指令	1
BGEU	无符号大于等于分支指令	1
JAL	直接跳转子程序指令	1
JALR	寄存器跳转子程序指令	1
内存存取指令		
LB	有符号扩展字节加载指令	WEAK ORDER : LOAD: >=3 STORE: 1 STRONG ORDER : 不定周期
LBU	无符号扩展字节加载指令	同上
LH	有符号扩展半字加载指令	同上
LHU	无符号扩展半字加载指令	同上
LW	有符号扩展字加载指令	同上
LWU	无符号扩展字加载指令	同上
LD	双字加载指令	同上
SB	字节存储指令	同上
SH	半字存储指令	同上
SW	字存储指令	同上
SD	双字存储指令	同上
控制与状态寄存器操作指令（Zicsr, Extension for Control and Status Register Instructions）		
CSRRW	控制寄存器读写传送指令	阻塞执行 不定周期
CSRRS	控制寄存器置位传送指令	同上
CSRRC	控制寄存器清零传送指令	同上
CSRRWI	控制寄存器立即数读写传送指令	同上
CSRRSI	控制寄存器立即数置位传送指令	同上
CSRRCI	控制寄存器立即数清零传送指令	同上
低功耗指令		
WFI	进入低功耗等待模式指令	不定周期
异常返回指令		

续下页

表 3.1 - 接上页

指令名称	指令描述	执行延时
MRET	机器模式异常返回指令	阻塞执行 不定周期
SRET	超级用户模式异常返回指令	同上
特殊功能指令		
FENCE	存储同步指令	不定周期
FENCE.I	指令流同步指令	阻塞执行 不定周期
SFENCE.VMA	虚拟内存同步指令	同上
ECALL	环境异常指令	1
EBREAK	断点指令	1

具体指令说明和定义，请参考*附录 A-1 I 指令术语*

3.1.2　乘除法指令集（RV64M）

表 3.2: 整型乘除法 (RV64M) 指令列表

指令名称	指令描述	执行延时
MUL	有符号乘法指令	4
MULW	低 32 位有符号乘法指令	4
MULH	有符号乘法取高位指令	4
MULHS	有符号无符号乘法取高位指令	4
MULHU	无符号乘法取高位指令	4
DIV	有符号除法指令	3-20
DIVW	低 32 位有符号除法指令	3-12
DIVU	无符号除法指令	3-20
DIVUW	低 32 位无符号除法指令	3-12
REM	有符号取余指令	3-20
REMW	低 32 位有符号取余指令	3-12
REMU	无符号取余指令	3-20
REMUW	低 32 位无符号取余指令	3-12

具体指令说明和定义，请参考*附录 A-2 M 指令术语*

3.1.3 原子指令集（RV64A）

表 3.3: 原子指令 (RV64A) 指令列表

指令名称	指令描述	执行延时
LR.W	字加载保留指令	拆分为多条原子指令执行。 可能拆分出阻塞执行，指令延时不可 预期。
LR.D	双字加载保留指令	
SC.W	字条件存储指令	
SC.D	双字条件存储指令	
AMOSWAP.W	低 32 位原子交换指令	
AMOSWAP.D	原子交换指令	
AMOADD.W	低 32 位原子加法指令	
AMOADD.D	原子加法指令	
AMOXOR.W	低 32 位原子按位异或指令	
AMOXOR.D	原子按位异或指令	
AMOAND.W	低 32 位原子按位与指令	
AMOAND.D	原子按位与指令	
AMOOR.W	低 32 位原子按位或指令	
AMOOR.D	原子按位或指令	
AMOMIN.W	低 32 位原子有符号取最小值指令	
AMOMIN.D	原子有符号取最小值指令	
AMOMAX.W	低 32 位原子有符号取最大值指令	
AMOMAX.D	原子有符号取最大值指令	
AMOMINU.W	低 32 位原子无符号取最小值指令	
AMOMINU.D	原子无符号取最小值指令	
AMOMAXU.W	低 32 位原子无符号取最大值指令	
AMOMAXU.D	原子无符号取最大值指令	

具体指令说明和定义，请参考*附录 A-3 A 指令术语*。

3.1.4 单精度浮点指令集（RV64F）

单精度浮点指令集按功能可以分为以下类型：

- 运算指令
- 符号注入指令
- 数据传输指令
- 比较指令

- 数据类型转换指令
- 内存存储指令
- 浮点数分类指令

表 3.4: 单精度浮点（RV64F）指令列表

指令名称	指令描述	执行延时
运算指令		
FADD.S	单精度浮点加法指令	3
FSUB.S	单精度浮点减法指令	3
FMUL.S	单精度浮点乘法指令	4
FMADD.S	单精度浮点乘累加指令	5
FMSUB.S	单精度浮点乘累减指令	5
FNMADD.S	单精度浮点乘累加取负指令	5
FNMSUB.S	单精度浮点乘累减取负指令	5
FDIV.S	单精度浮点除法指令	4-10
FSQRT.S	单精度浮点开方指令	4-10
符号注入指令		
FSGNJ.S	单精度浮点符号注入指令	3
FSGNJN.S	单精度浮点符号取反注入指令	3
FSGNJX.S	单精度浮点符号异或注入指令	3
数据传输指令		
FMV.X.W	单精度浮点读传送指令	拆分执行 1+1
FMV.W.X	单精度浮点写传送指令	拆分执行 1+1
比较指令		
FMIN.S	单精度浮点取最小值指令	3
FMAX.S	单精度浮点取最大值指令	3
FEQ.S	单精度浮点比较相等指令	拆分执行 3+1
FLT.S	单精度浮点比较小于指令	拆分执行 3+1
FLE.S	单精度浮点比较小于等于指令	拆分执行 3+1
数据类型转换指令		
FCVT.W.S	单精度浮点转换成有符号整型指令	拆分执行 3+1
FCVT.WU.S	单精度浮点转换成无符号整型指令	拆分执行 3+1
FCVT.S.W	有符号整型转换成单精度浮点指令	拆分执行 3+1
FCVT.S.WU	无符号整型转换成单精度浮点指令	拆分执行 3+1
FCVT.L.S	单精度浮点转换成有符号长整型指令	拆分执行 3+1

续下页

表 3.4 - 接上页

指令名称	指令描述	执行延时
FCVT.LU.S	单精度浮点转换成无符号长整型指令	拆分执行 3+1
FCVT.S.L	有符号长整型转换成单精度浮点指令	拆分执行 1+3
FCVT.S.LU	无符号长整型转换成单精度浮点指令	拆分执行 1+3
内存存储指令		
FLW	单精度浮点加载指令	WEAK ORDER : LOAD: >=3 STORE: 1 STRONG ORDER : 不定周期
FSW	单精度浮点存储指令	同上
浮点数分类指令		
FCLASS.S	单精度浮点分类指令	1+1

具体指令说明和定义，请参考*附录 A-4 F 指令术语*。

3.1.5 压缩指令集（RV64C）

压缩指令集按功能可以分为以下类型：

- 加减法指令
- 逻辑操作指令
- 移位指令
- 数据传输指令
- 分支跳转指令
- 立即数偏移存取指令

表 3.5: 压缩指令（RV64C）指令列表

指令名称	指令描述	执行延时
加减法指令		
C.ADD	有符号加法指令	1
C.ADDW	低 32 位有符号加法指令	1
C.ADDI	有符号立即数加法指令	1
C.ADDIW	低 32 位有符号立即数加法指令	1
C.SUB	有符号减法压缩指令	1
C.SUBW	低 32 位有符号减法指令	1

续下页

表 3.5 - 接上页

指令名称	指令描述	执行延时
C.ADDI16SP	加 16 倍立即数到堆栈指针	1
C.ADDI4SPN	4 倍立即数和堆栈指针相加	1
逻辑操作指令		
C.AND	按位与指令	1
C.ANDI	立即数按位与指令	1
C.OR	按位或指令	1
C.XOR	按位异或指令	1
移位指令		
C.SLLI	立即数逻辑左移指令	1
C.SRLI	立即数逻辑右移指令	1
C.SRAI	立即数算术右移指令	1
数据传输指令		
C.MV	数据传送指令	1
C.LI	低位立即数传送指令	1
C.LUI	高位立即数传送指令	1
分支跳转指令		
C.BEQZ	等于零分支指令	1
C.BNEZ	不等于零分支指令	1
C.J	无条件跳转指令	1
C.JR	寄存器跳转指令	1
C.JALR	寄存器跳转子程序指令	1
立即数偏移存取指令		
C.LW	字加载指令	WEAK ORDER : LOAD: >=3 STORE: 1 STRONG ORDER : 不定周期
C.SW	字存储指令	同上
C.LWSP	字堆栈加载指令	同上
C.SWSP	字堆栈存储指令	同上
C.LD	双字加载指令	同上
C.SD	双字存储指令	同上
C.LDSP	双字堆栈加载指令	同上

续下页

表 3.5 - 接上页

指令名称	指令描述	执行延时
C.SDSP	双字堆栈存储指令	同上
C.FLD	双精度加载指令	同上
C.FSD	双精度存储指令	同上
C.FLDSP	双精度堆栈存储指令	同上
C.FSDSP	双精度堆栈加载指令	同上
特殊指令		
C.NOP	空指令	1
C.EBREAK	断点指令	1

具体指令说明和定义，请参考*附录 A-6 C 指令术语*

3.1.6　矢量指令集（RV64V）

关于矢量指令集的具体细节，请参考 RISC-V "V" Vector Extension, Version 1.0-rc1-20210608 。

3.1.7　位操作指令集（RV64B）

关于位操作指令的具体细节，请参考 RISC-V Bit-Manipulation ISA-extensions, Version 1.0.0, 2021-06-12: public review 。

3.2　玄铁扩展指令集

C908X 除了支持 RV64GCB[V] 指令集之外，还在此基础上拓展了部分自定义指令，包括算术运算类指令、位操作类指令、内存访问类指令、SFU 扩展指令、Cache 指令子集、多核同步指令子集、半精度浮点指令集、AI 扩展指令、Vector 扩展指令、SCALAR 扩展指令和玄铁协处理器扩展指令。

C908X 的扩展指令集中，半精度浮点指令集可以直接使用，其它扩展指令使用前需要使能*机器模式扩展状态寄存器（MXSTATUS）*相应 bit，具体说明如下：

- 算术运算类指令、位操作类指令、内存访问类指令、SFU 扩展指令、AI 扩展指令、Vector 扩展指令、SCALAR 扩展指令在 mxstatus.XUANTIEISAEE == 1 时可以正常执行，否则将产生非法指令异常；
- 玄铁扩展 Cache 指令子集、多核同步指令子集在 mxstatus.XUANTIEISAEE == 1 或 mxstatus.COPINSTEE == 1 时可以正常执行；
- 玄铁协处理器扩展指令需要在 mxstatus.COPINSTEE == 1 时才能正常使用，否则将产生非法指令异常。

具体指令说明和定义，请参考*附录 B 玄铁扩展指令术语* 。

3.2.1　算术运算类指令

表 3.6: 算术运算指令集

指令名称	指令描述	执行延时
加减法指令		
ADDSL	寄存器移位相加指令	1
MULA	乘累加指令	非累加数相关性:4
MULS	乘累减指令	非累加数相关性:4
MULAW	低 32 位乘累加指令	累加数相关性:1
MULSW	低 32 位乘累减指令	累加数相关性:1
MULAH	低 16 位乘累加指令	累加数相关性:1
MULSH	低 16 位乘累减指令	累加数相关性:1
位移指令		
SRRI	循环右移指令	1
SRRIW	低 32 位循环右移指令	1
传送指令		
MVEQZ	寄存器为 0 传送指令	1
MVNEZ	寄存器非 0 传送指令	1

具体指令说明和定义，请参考*附录 B-3 算术运算指令术语*。

3.2.2 位操作类指令

表 3.7: 位操指令集

指令名称	指令描述	执行延时
位操作指令		
TST	比特为 0 测试指令	1
TSTNBZ	字节为 0 测试指令	1
REV	字节倒序指令	1
REVW	低 32 位字节倒序指令	1
FF0	快速找 0 指令	1
FF1	快速找 1 指令	1
EXT	寄存器连续位提取符号位扩展指令	1
EXTU	寄存器连续位提取零扩展指令	1

具体指令说明和定义，请参考*附录 B-4 位操作指令术语*。

3.2.3 内存访问类指令

表 3.8: 内存访问指令集

指令名称	指令描述	执行延时
FLRD	浮点寄存器移位双字加载指令	WEAK ORDER： >=3 STRONG ORDER： 不定周期
FLRW	浮点寄存器移位字加载指令	
FLURD	浮点寄存器低 32 位移位双字加载指令	
FLURW	浮点寄存器低 32 位移位字加载指令	
LRB	寄存器移位符号位扩展字节加载指令	
LRH	寄存器移位符号位扩展半字加载指令	
LRW	寄存器移位符号位扩展字加载指令	
LRD	寄存器移位双字加载指令	
LRBU	寄存器移位零扩展字节加载指令	
LRHU	寄存器移位零扩展半字加载指令	
LRWU	寄存器移位零扩展字加载指令	
LURB	寄存器低 32 位移位符号位扩展字节加载指令	
LURH	寄存器低 32 位移位符号位扩展半字加载指令	
LURW	寄存器低 32 位移位符号位扩展字加载指令	
LURD	寄存器低 32 位移位双字加载指令	
LURBU	寄存器低 32 位移位零扩展字节加载指令	
LURHU	寄存器低 32 位移位零扩展半字加载指令	
LURWU	寄存器低 32 位移位零扩展字加载指令	
LBIA	符号位扩展字节加载基地址自增指令	拆分为 load 指令和 alu 指令执行
LBIB	基地址自增符号位扩展字节加载指令	WEAK ORDER： >=3 STRONG ORDER： 不定周期
LHIA	符号位扩展半字加载基地址自增指令	
LHIB	基地址自增符号位扩展半字加载指令	
LWIA	符号位扩展字加载基地址自增指令	
LWIB	基地址自增符号位扩展字加载指令	
LDIA	符号位扩展双字加载基地址自增指令	
LDIB	基地址自增符号位扩展双字加载指令	
LBUIA	零扩展字节加载基地址自增指令	
LBUIB	基地址自增零扩展字节加载指令	
LHUIA	零扩展半字加载地址自增指令	
LHUIB	基地址自增零扩展半字加载指令	

续下页

表 3.8 - 接上页

指令名称	指令描述	执行延时
LWUIA	零拓展字加载地址自增指令	
LWUIB	基地址自增零扩展字加载指令	
LDD	双寄存器加载指令	拆分为两条 load 执行
LWD	符号位扩展双寄存器字加载指令	WEAK ORDER : >=3 STRONG ORDER : 不定周期
LWUD	零扩展双寄存器字加载指令	
FSRD	浮点寄存器移位双字存储指令	WEAK ORDER : 1 STRONG ORDER : 不定周期
FSRW	浮点寄存器移位字存储指令	
FSURD	浮点寄存器低 32 位移位双字存储指令	
FSURW	浮点寄存器低 32 位移位字存储指令	
SRB	寄存器移位字节存储指令	
SRW	寄存器移位字存储指令	
SRD	寄存器移位双字存储指令	
SURB	寄存器低 32 位移位字节存储指令	
SURH	寄存器低 32 位移位半字存储指令	
SURW	寄存器低 32 位移位字存储指令	
SURD	寄存器低 32 位移位双字存储指令	
SBIA	字节存储基地址自增指令	拆分为 store 指令和 alu 指令执行
SBIB	基地址自增字节存储指令	WEAK ORDER : 1 STRONG ORDER : 不定周期
SHIA	半字存储基地址自增指令	
SHIB	基地址自增半字存储指令	
SWIA	字存储基地址自增指令	
SWIB	基地址自增字存储指令	
SDIA	双字存储基地址自增指令	
SDIB	基地址自增双字存储指令	
SDD	双寄存器加存储指令	拆分为两条 store 执行
SWD	双寄存器低 32 位存储指令	WEAK ORDER : 1 STRONG ORDER : 不定周期

具体指令说明和定义，请参考*附录 B-5 存储指令术语*。

3.2.4　Cache 指令

表 3.9: Cache 指令列表

指令名称	指令描述	执行延时（LMUL=1）
DCACHE.CALL	DCACHE 清全部脏表项指令	阻塞执行 不定周期
DCACHE.CIALL	DCACHE 清全部脏表项后无效指令	
DCACHE.CIPA	DCACHE 按物理地址清脏表项并无效指令（作用域包含 L2CACHE）	
DCACHE.CISW	DCACHE 按 set/way 清脏表项并无效指令	
DCACHE.CIVA	DCACHE 按虚拟地址清脏表项并无效指令（作用域包含 L2CACHE）	
DCACHE.CPA	DCACHE 按物理地址清脏表项指令（作用域包含 L2CACHE）	
DCACHE.CPAL1	L1DCACHE 按物理地址清脏表项指令	
DCACHE.CSW	DCACHE 按 set/way 清脏表项指令	
DCACHE.CVA	DCACHE 按虚拟地址清脏表项指令（作用域包含 L2CACHE）	
DCACHE.CVAL1	L1DCACHE 按虚拟地址清脏表项指令	
DCACHE.IPA	DCACHE 按物理地址无效指令（作用域包含 L2CACHE）	
DCACHE.ISW	DCACHE 按 set/way 无效指令	
DCACHE.IVA	DCACHE 按虚拟地址无效指令（作用域包含 L2CACHE）	
DCACHE.IALL	DCACHE 无效所有表项指令	
ICACHE.IALL	ICACHE 无效所有表项指令	不定周期
ICACHE.IALLS	ICACHE 广播无效所有表项指令	
ICACHE.IPA	ICACHE 按物理地址无效表项指令	
ICACHE.IVA	ICACHE 按虚拟地址无效表项指令	

具体指令说明和定义，请参考*附录 B-1 Cache 指令术语*。

3.2.5　多核同步指令

表 3.10: 多核同步指令集

多核同步指令	描述
SYNC	同步指令

续下页

表 3.10 - 接上页

多核同步指令	描述
SYNC.S	同步广播指令
SYNC.I	同步清空指令
SYNC.IS	同步清空广播指令

具体指令说明和定义，请参考*附录 B-2 多核同步指令术语*。

3.2.6 半精度浮点类指令

表 3.11: 半精度浮点指令集

指令名称	指令描述	执行延时
运算指令		
FADD.H	半精度浮点加法指令	3
FSUB.H	半精度浮点减法指令	3
FMUL.H	半精度浮点乘法指令	3
FMADD.H	半精度浮点乘累加指令	4
FMSUB.H	半精度浮点乘累减指令	4
FNMADD.H	半精度浮点乘累加取负指令	4
FNMSUB.H	半精度浮点乘累减取负指令	4
FDIV.H	半精度浮点除法指令	4-7
FSQRT.H	半精度浮点开方指令	4-7
符号注入指令		
FSGNJ.H	半精度浮点符号注入指令	3
FSGNJN.H	半精度浮点符号取反注入指令	3
FSGNJX.H	半精度浮点符号异或注入指令	3
数据传输指令		
FMV.X.H	半精度浮点读传送指令	1+1
FMV.H.X	半精度浮点写传送指令	1+1
比较指令		
FMIN.H	半精度浮点取最小值指令	3
FMAX.H	半精度浮点取最大值指令	3
FEQ.H	半精度浮点比较相等指令	拆分执行 3+1
FLT.H	半精度浮点比较小于指令	拆分执行 3+1
FLE.H	半精度浮点比较小于等于指令	拆分执行 3+1

续下页

表 3.11 - 接上页

指令名称	指令描述	执行延时
数据类型转换指令		
FCVT.S.H	半精度浮点转换成单精度浮点指令	3
FCVT.H.S	单精度浮点转换成半精度浮点指令	3
FCVT.W.H	半精度浮点转换成有符号整型指令	拆分执行 3+1
FCVT.WU.H	半精度浮点转换成无符号整型指令	拆分执行 3+1
FCVT.H.W	有符号整型转换成半精度浮点指令	拆分执行 3+1
FCVT.H.WU	无符号整型转换成半精度浮点指令	拆分执行 3+1
FCVT.L.H	半精度浮点转换成有符号长整型指令	拆分执行 3+1
FCVT.LU.H	半精度浮点转换成无符号长整型指令	拆分执行 3+1
FCVT.H.L	有符号长整型转换成半精度浮点指令	拆分执行 3+1
FCVT.H.LU	无符号长整型转换成半精度浮点指令	拆分执行 3+1
内存存储指令		
FLH	半精度浮点加载指令	WEAK ORDER : LOAD: >=3 STORE: 1 STRONG ORDER : 不定周期
FSH	半精度浮点存储指令	同上
浮点数分类指令		
FCLASS.H	半精度浮点分类指令	1+1

具体指令说明和定义，请参考*附录 B-6 浮点半精度指令术语*。

3.2.7 AI 扩展指令

表 3.12: AI 扩展指令

指令名称	指令描述	执行延时
vmaqa.vv	8-bit 矢量有符号乘累加链加指令	3
vmaqau.vv	8-bit 矢量无符号乘累加链加指令	3
vmaqasu.vv	8-bit 矢量有符号无符号乘累加链加指令	3
vmaqa.vx	8-bit 矢量标量有符号乘累加链加指令	3
vmaqau.vx	8-bit 矢量标量无符号乘累加链加指令	3
vmaqasu.vx	8-bit 矢量标量有符号无符号乘累加链加指令	3
vmaqaus.vx	8-bit 矢量标量无符号有符号乘累加链加指令	3

续下页

表 3.12 - 接上页

指令名称	指令描述	执行延时
vpmaqa.vv	4-bit 矢量有符号乘累加链加指令	3
vpmaqau.vv	4-bit 矢量无符号乘累加链加指令	3
vpmaqasu.vv	4-bit 矢量有符号无符号乘累加链加指令	3
vpmaqa.vx	4-bit 矢量标量有符号乘累加链加指令	3
vpmaqau.vx	4-bit 矢量标量无符号乘累加链加指令	3
vpmaqasu.vx	4-bit 矢量标量有符号无符号乘累加链加指令	3
vpmaqaus.vx	4-bit 矢量标量无符号有符号乘累加链加指令	3
vpnclip.wv	8-bit 矢量有符号缩位算术右移指令	3
vpnclipu.wv	8-bit 矢量无符号缩位算术右移指令	3
vpnclip.wx	8-bit 矢量标量缩位算术右移指令	3
vpnclipu.wx	8-bit 矢量标量无符号缩位算数右移指令	3
vpwadd.vv	4-bit 扩位矢量整型有符号扩位加法指令	3
vpwaddu.vv	4-bit 扩位矢量整型无符号扩位加法指令	3
vpwadd.vx	4-bit 扩位矢量标量整型有符号扩位加法指令	3
vpwaddu.vx	4-bit 扩位矢量标量整型无符号扩位加法指令	3

具体指令说明和定义，请参考*附录 B-8 AI 扩展指令术语*。

3.2.8　Vector 扩展指令

表 3.13: 玄铁 Vector 扩展指令列表

类别	指令名称
SFU 计算 *	th.vfexp2
	th.vftanh
	th.vfsig
	th.vfrec
BF16 vs FP8 转换 *	th.vfncvt.e4.h
	th.vfncvt.e5.h
	th.vfncvt.e4.bf16
	th.vfncvt.e5.bf16
	th.vfncvt.rod.bf16.s
	th.vfwcvt.h.e4
	th.vfwcvt.h.e5

续下页

表 3.13 - 接上页

类别	指令名称
	th.vfwcvt.bf16.e4
	th.vfwcvt.bf16.e5
REDUCTION	th.vfredsum.dup.32
	th.vfredsum.dup.64
	th.vfredmax.dup.32
	th.vfredmax.dup.64
	th.vfredmin.dup.32
	th.vfredmin.dup.64
	th.vfredsum.c.32
	th.vfredsum.c.64
	th.vfredmax.c.32
	th.vfredmax.c.64
	th.vfredmin.c.32
	th.vfredmin.c.64
	th.vbfredsum.dup.32
	th.vbfredsum.dup.64
	th.vbfredmax.dup.32
	th.vbfredmax.dup.64
	th.vbfredmin.dup.32
	th.vbfredmin.dup.64
	th.vbfredsum.c.32
	th.vbfredsum.c.64
	th.vbfredmax.c.32
	th.vbfredmax.c.64
	th.vbfredmin.c.32
	th.vbfredmin.c.64
	th.vary.dup.32
	th.vary.dup.64

* : The destination vector register group for a masked vector instructioncannot overlap the source mask register (v0).

具体指令说明和定义，请参考*附录 B-9 Vector 扩展指令术语*和*附录 B-10 部分玄铁扩展 Vector 指令遵循的 RVV 标准规范说明*。

3.2.9　SCALAR 扩展指令

表 3.14: SCALAR 扩展指令列表

指令名称	指令描述
th.wfe	Enter a low power state

具体指令说明和定义，请参考*附录 B-11 SCALAR 扩展指令术语*。

3.2.10　玄铁协处理器扩展指令

为方便部分用户快速使用协处理器指令，玄铁 CPU 预设了部分用户可能常用的协处理器扩展指令，该部分指令已经在编译工具中进行了支持，指令编码使用 custom2 域。C908X 用户自扩展协处理器指令集需要在机器模式扩展状态寄存器（MXSTATUS）中打开用户自扩展协处理器指令集使能位（COPINSTEE）才能正常使用，否则将产生非法指令异常。

表 3.15: 玄铁协处理器扩展指令列表

分类	指令名称
整型指令	CPX0
	CPX1
	CPX2
	CPX3
	CPX4
	CPX5
	CPX6
	CPX7
	CPX8
	CPX9
	CPX10
浮点类型指令	FCPX0
	FCPX1
	FCPX2
	FCPX3
	FCPX4
	FCPX5
	FCPX6

具体指令说明和定义，请参考*附录 B-12 玄铁协处理器扩展指令术语*。

4 处理器模式与寄存器

4.1 处理器模式

C908X 支持 RISC-V 三种**特权模式**：机器模式、超级用户模式和用户模式。处理器复位后在机器模式下执行程序。三种运行模式对应不同的操作权限，区别主要体现在以下几个方面：

1. 对寄存器的访问；
2. 特权指令的使用；
3. 对内存空间的访问。

- **用户模式权限最低。**

  普通用户程序只允许访问指定给普通用户模式的寄存器。避免了普通用户程序接触特权信息，而操作系统通过协调普通用户程序的行为来为普通用户程序提供管理和服务。

- **超级用户模式权限比用户模式高，但比机器模式低。**

  超级用户模式下运行的程序不可以使用机器模式的控制寄存器，并且受到 PMP 的限制。使用基于页面的虚拟内存，这个功能构成了超级用户模式的核心。

- **机器模式拥有最高的权限。**

  在机器模式下运行的程序对内存、I/O 和一些对于启动和配置系统来说必要的底层功能有着完全的使用权。默认情况下（异常中断没有被降级处理），任何模式下发生的异常和中断都会切换到机器模式进行响应。

大多数指令在三种模式下都能执行，但是一些对系统产生重大影响的特权指令只能在超级用户模式或机器模式下执行，具体信息可参考*附录 A 标准指令术语*和*附录 B 玄铁扩展指令术语*，查看指令的执行权限。

处理器的工作模式在异常响应时发生变化（响应异常的特权模式不同于异常发生时所处的特权模式），进入更高的特权模式响应异常，异常响应完之后再回到低特权模式。

4.2 寄存器视图

C908X 的寄存器视图如 图 4.1 所示：

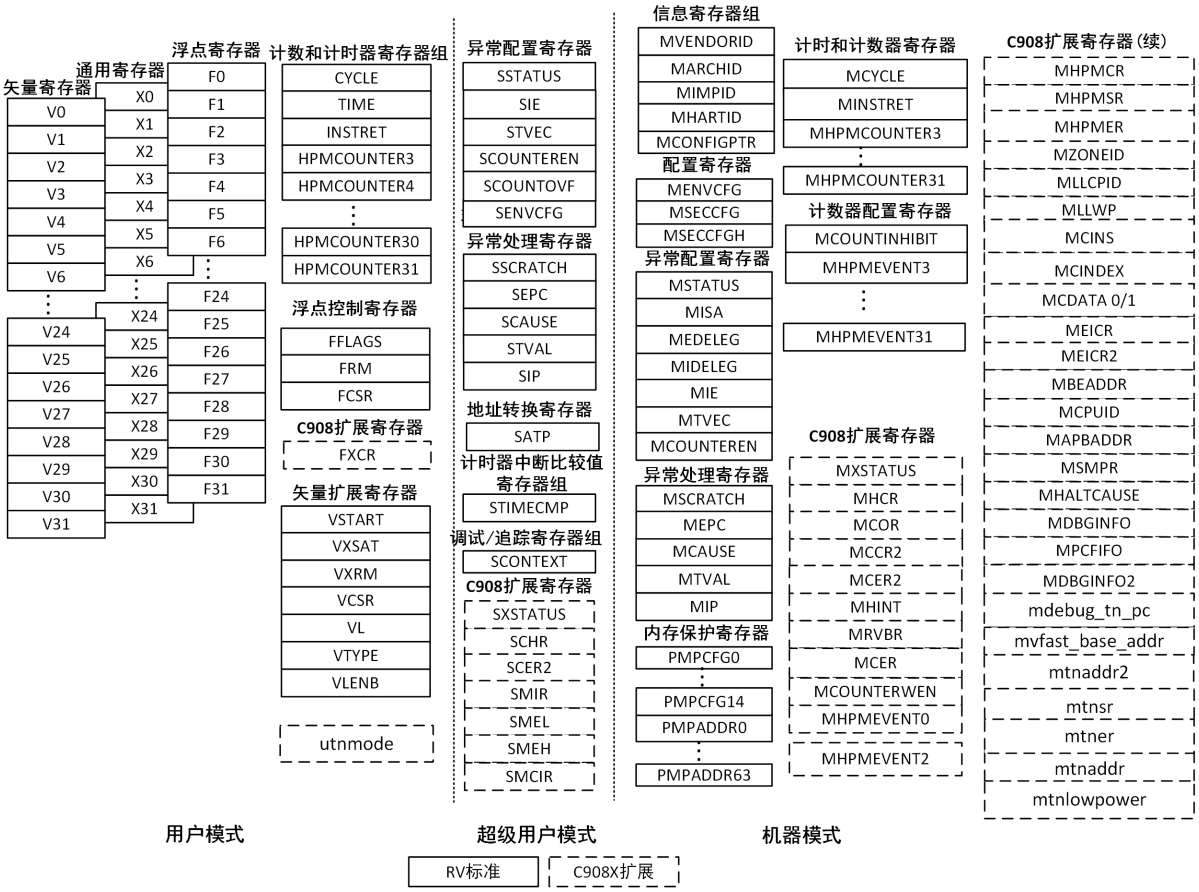


图 4.1: 寄存器视图

4.3　通用寄存器

C908X 拥有 32 个 64 位的通用寄存器，功能定义与 RISC-V 一致，如 表 4.1 所示。

表 4.1: 通用寄存器

寄存器	ABI 名称	描述
x0	zero	硬件绑 0
x1	ra	返回地址
x2	sp	堆栈指针
x3	gp	全局指针
x4	tp	线程指针
x5	t0	临时/备用链接寄存器
x6-7	t1-2	临时寄存器
x8	s0/fp	保留寄存器/帧指针
x9	s1	保留寄存器

续下页

表 4.1 - 接上页

寄存器	ABI 名称	描述
x10-11	a0-1	函数参数/返回值
x12-17	a2-7	函数参数
x18-27	s2-11	保留寄存器
x28-31	t3-6	临时寄存器

通用寄存器用于保存指令操作数、指令执行结果以及地址信息。

4.4　浮点寄存器

C908X 的浮点单元除了支持标准 RV64FD 指令集以外，还扩展支持了浮点半精度计算，拥有 32 个独立的 64 位浮点寄存器，可以在普通用户模式、超级用户模式和机器模式下被访问。

表 4.2: 浮点寄存器

寄存器	ABI 名称	描述
f0-7	ft0-7	浮点临时寄存器
f8-9	fs0-1	浮点保留寄存器
f10-11	fa0-1	浮点参数/返回值
f12-17	fa2-7	浮点参数
f18-27	fs2-11	浮点保留寄存器
f28-31	ft8-11	浮点临时寄存器

浮点寄存器 f0 和通用寄存器 x0 不同，并不是硬件绑死 0，而是和其他浮点寄存器一样，是可变的。单精度浮点数仅使用 64 位浮点寄存器的低 32 位，高 32 位必须全为 1，否则会被当做非数处理；半精度浮点数仅使用 64 位浮点寄存器的低 16 位，高 48 位必须全为 1，否则会被当作非数处理。

增加单独的浮点寄存器可以增大寄存器容量和带宽，进而提高处理器的性能。同时必须增加浮点加载和存储指令，还需要增加浮点和通用寄存器之间数据传递指令。

4.4.1　浮点寄存器与通用寄存器传输数据

通用寄存器与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：

- FMV.X.H/FMV.H.X 浮点寄存器半精度传送指令。
- FMV.X.W/FMV.W.X 浮点寄存器单精度传送指令。

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

具体指令说明和定义可以参考*附录 A-4 F 指令术语*

4.4.2　维护寄存器精度的一致

浮点寄存器可以存储半精度浮点数、单精度浮点数和整形数据。举例说明，在浮点寄存器 f1 中所存的数据类型，取决于上一次的写操作，可能是四种数据类型中的任何一种。

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

4.5　矢量寄存器

C908X 拥有 32 个独立的矢量寄存器，位宽为 512/1024/4096 位，可在普通用户模式、超级用户模式和机器模式下被访问。矢量寄存器通过矢量传送指令实现与通用寄存器/浮点寄存器的数据交换。

4.5.1　矢量寄存器与通用寄存器传输数据

矢量寄存器与通用寄存器之间的数据传输可以通过矢量整型寄存器传送指令实现。矢量整型寄存器传送指令包括：

- VMV.V.X 整型传送至矢量指令。
- VMV.S.X 整型传送至矢量首元素指令
- VEXT.X.V 矢量整型提取元素指令。

4.5.2　矢量寄存器与浮点寄存器传输数据

矢量寄存器与浮点寄存器之间的数据传输可以通过矢量浮点寄存器传送指令实现。矢量浮点寄存器传送指令包括：

- VFMV.V.F 浮点传送至矢量指令。
- VFMV.F.S 矢量首元素传送至浮点指令。
- VFMV.S.F 浮点传送至矢量首元素指令。

4.6　系统控制寄存器

4.6.1　标准控制寄存器

本章节描述 C908X 实现的 RISC-V 标准控制寄存器，按照机器模式、超级用户模式、用户模式分别描述。

C908X 中实现的 RISC-V 标准定义的机器模式控制寄存器如 表 4.3 所示。

表 4.3: RISC-V 标准机器模式控制寄存器

名称	读写权限	寄存器编号	描述
机器模式信息寄存器组			
mvendorid	机器模式只读	0xF11	供应商编号寄存器
marchid	机器模式只读	0xF12	架构编号寄存器
mimpid	机器模式只读	0xF13	机器模式硬件实现编号寄存器
mhartid	机器模式只读	0xF14	机器模式逻辑内核编号寄存器
mconfigptr	机器模式只读	0xF15	机器模式配置数据结构指针
机器模式异常配置寄存器组			
mstatus	机器模式读写	0x300	机器模式处理器状态寄存器
misa	机器模式读写	0x301	机器模式处理器指令集特性寄存器
medeleg	机器模式读写	0x302	机器模式异常降级控制寄存器
mideleg	机器模式读写	0x303	机器模式中断降级控制寄存器
mie	机器模式读写	0x304	机器模式中断使能控制寄存器
mtvec	机器模式读写	0x305	机器模式向量基址寄存器
mcounteren	机器模式读写	0x306	机器模式计数器授权控制寄存器
机器模式异常处理寄存器组			
mscratch	机器模式读写	0x340	机器模式异常临时数据备份寄存器
mepc	机器模式读写	0x341	机器模式异常保留程序计数器
mcause	机器模式读写	0x342	机器模式异常事件原因寄存器
mtval	机器模式读写	0x343	机器模式异常事件向量寄存器
mip	机器模式读写	0x344	机器模式中断等待状态寄存器
机器模式配置寄存器组			
menvcfg	机器模式读写	0x30A	机器模式环境配置寄存器
mseccfg	机器模式读写	0x747	机器模式安全配置寄存器
mseccfgh	机器模式读写	0x757	机器模式安全配置寄存器 (U32)
机器模式内存保护寄存器组			
pmpcfg0	机器模式读写	0x3A0	物理内存保护配置寄存器 0
pmpcfg2	机器模式读写	0x3A2	物理内存保护配置寄存器 2
……			
pmpcfg14	机器模式读写	0x3AE	物理内存保护配置寄存器 14
pmpaddr0	机器模式读写	0x3B0	物理内存保护基址寄存器 0
……			
pmpaddr63	机器模式读写	0x3EF	物理内存保护基址寄存器 63
机器模式计数器/计时器			

续下页

表 4.3 - 接上页

名称	读写权限	寄存器编号	描述
mcycle	机器模式读写	0xB00	机器模式周期计数器
minstret	机器模式读写	0xB02	机器模式退休指令计数器
mhpmcounter3	机器模式读写	0xB03	机器模式计数器 3
……			
mhpmcounter31	机器模式读写	0xB1F	机器模式计数器 31
机器模式计数器配置寄存器组			
mcountinhibit	机器模式读写	0x320	机器模式计数禁止寄存器
mhpmevent3	机器模式读写	0x323	机器模式事件选择寄存器 3
……			
mhpmevent31	机器模式读写	0x33F	机器模式事件选择寄存器 31
调试/追踪寄存器组（与调试模式共享）			
tselect	机器模式读写	0x7A0	调试/追踪触发器选择寄存器
tdata1	机器模式读写	0x7A1	调试/追踪触发器数据寄存器 1
tdata2	机器模式读写	0x7A2	调试/追踪触发器数据寄存器 2
tdata3	机器模式读写	0x7A3	调试/追踪触发器数据寄存器 3
tinfo	机器模式只读	0x7A4	调试/追踪触发器信息寄存器
tcontrol	机器模式读写	0x7A5	调试/追踪触发器控制寄存器
mcontext	机器模式读写	0x7A8	机器模式内容寄存器
调试模式寄存器组			
dcsr	调试模式读写	0x7B0	调试模式控制与状态寄存器
dpc	调试模式读写	0x7B1	调试模式程序计数器
dscratch0	调试模式读写	0x7B2	调试模式临时数据备份寄存器 0
dscratch1	调试模式读写	0x7B3	调试模式临时数据备份寄存器 1

C908X 中实现的 RISC-V 标准定义的超级用户模式控制寄存器如 表 4.4 所示。

表 4.4: RISC-V 标准超级用户模式控制寄存器

名称	读写权限	寄存器编号	描述
超级用户模式异常配置寄存器组			
sstatus	超级用户模式读写	0x100	超级用户模式处理器状态寄存器
sie	超级用户模式读写	0x104	超级用户模式中断使能控制寄存器
stvec	超级用户模式读写	0x105	超级用户模式向量基址寄存器
scounteren	超级用户模式读写	0x106	超级用户模式计数器使能控制寄存器

续下页

表 4.4 - 接上页

名称	读写权限	寄存器编号	描述
scountovf	超级用户模式只读	0xDA0	超级用户模式计数器溢出寄存器
超级用户模式配置寄存器组			
senvcfg	超级用户模式读写	0x10A	超级用户模式环境配置寄存器
超级用户模式异常处理寄存器组			
sscratch	超级用户模式读写	0x140	超级用户模式异常临时数据备份寄存器
sepc	超级用户模式读写	0x141	超级用户模式异常保留程序计数器
scause	超级用户模式读写	0x142	超级用户模式异常事件原因寄存器
stval	超级用户模式读写	0x143	超级用户模式异常事件向量寄存器
sip	超级用户模式读写	0x144	超级用户模式中断等待状态寄存器
超级用户模式计时器中断比较值寄存器组			
stimecmp	超级用户模式读写	0x14D	超级用户模式计时器中断比较值寄存器
超级用户模式地址转换寄存器组			
satp	超级用户模式读写	0x180	超级用户虚拟地址转换和保护寄存器
调试/追踪寄存器组			
scontext	超级用户模式读写	0x5A8	超级用户模式内容寄存器

C908X 中实现的 RISC-V 标准定义的用户模式控制寄存器如 表 4.5 所示。

表 4.5: RISC-V 标准用户模式控制寄存器

名称	读写权限	寄存器编号	描述
用户模式浮点控制寄存器组			
fflags	用户模式读写	0x001	浮点异常累积状态寄存器
frm	用户模式读写	0x002	浮点动态舍入模式控制寄存器
fcsr	用户模式读写	0x003	浮点控制状态寄存器
用户模式计数/计时器			
cycle	用户模式只读	0xC00	用户模式周期计数器
time	用户模式只读	0xC01	用户模式时间计数器
instret	用户模式只读	0xC02	用户模式退休指令计数器
hpmcounter3	用户模式只读	0xC03	用户模式计数器 3
……			
hpmcounter31	用户模式只读	0xC1F	用户模式计数器 31
矢量扩展寄存器组			
vstart	用户模式读写	0x008	矢量起始位置寄存器

续下页

表 4.5 - 接上页

名称	读写权限	寄存器编号	描述
vxsat	用户模式读写	0x009	定点溢出标志位寄存器
vxrm	用户模式读写	0x00A	定点舍入模式寄存器
vcsr	用户模式读写	0x00F	矢量控制和状态寄存器
vl	用户模式只读	0xC20	矢量长度寄存器
vtype	用户模式只读	0xC21	矢量数据类型寄存器
vlenb	用户模式只读	0xC22	矢量宽度，以字节为单位

4.6.2　扩展控制寄存器

本章节描述 C908X 实现的扩展控制寄存器，按照机器模式、超级用户模式、用户模式分别描述。

C908X 中扩展的机器模式控制寄存器如 表 4.6 所示。

表 4.6: C908X 扩展机器模式控制寄存器

名称	读写权限	寄存器编号	描述
机器模式处理器控制和状态扩展寄存器组			
mxstatus	机器模式读写	0x7C0	机器模式扩展状态寄存器
mhcr	机器模式读写	0x7C1	机器模式硬件配置寄存器
mcor	机器模式读写	0x7C2	机器模式硬件操作寄存器
mccr2	机器模式读写	0x7C3	机器模式 L2 Cache 控制寄存器
mcer2	机器模式读写	0x7C4	机器模式 L2 Cache 错误寄存器
mhint	机器模式读写	0x7C5	机器模式隐式操作寄存器
mrvbr	机器模式读写	0x7C7	机器模式复位向量基址寄存器
mcer	机器模式读写	0x7C8	机器模式 L1 Cache ECC 寄存器
mcounterwen	机器模式读写	0x7C9	机器模式计数器写使能寄存器
mhpmevent0	机器模式读写	0x7E0	机器模式 cycle 事件设置寄存器
mhpmevent2	机器模式读写	0x7E1	机器模式 inst retired 事件设置寄存器
mhpmcr	机器模式读写	0x7F0	性能监控控制寄存器
mhpmsr	机器模式读写	0x7F1	性能监控起始触发寄存器
mhpmer	机器模式读写	0x7F2	性能监控终止触发寄存器
mzoneid	机器模式读写	0x7F5	zoneid 寄存器
mllcpid	机器模式读写	0x7F6	细粒度回填 ID 寄存器
mllwp	机器模式读写	0x7F7	L2 细粒度配置寄存器
机器模式 Cache 访问扩展寄存器组			

续下页

表 4.6 - 接上页

名称	读写权限	寄存器编号	描述
mcins	机器模式读写	0x7D2	机器模式 Cache 指令寄存器
mcindex	机器模式读写	0x7D3	机器模式 Cache 访问索引寄存器
mcdata0	机器模式只读	0x7D4	机器模式 Cache 数据寄存器 0
mcdata1	机器模式只读	0x7D5	机器模式 Cache 数据寄存器 1
meicr	机器模式读写	0x7D6	L1 Cache 硬件错误注入寄存器
meicr2	机器模式读写	0x7D7	L2 Cache 硬件错误注入寄存器
mbeaddr	机器模式只读	0x7D8	L1 LD BUS ERR 地址寄存器
机器模式处理器型号扩展寄存器组			
mcpuid	机器模式只读	0xFC0	机器模式处理器型号寄存器
mapbaddr	机器模式只读	0xFC1	片上总线基地址
多核扩展寄存器组			
msmpr	机器模式读写	0x7F3	Snoop 监听使能寄存器
机器模式调试扩展控制寄存器			
mhaltcause	机器模式只读	0xFE0	HALT 原因寄存器
mdbginfo	机器模式只读	0xFE1	调试信息寄存器
mpcfifo	机器模式只读	0xFE2	pcfifo 寄存器
mdbginfo2	机器模式只读	0xFE3	调试信息寄存器 2

具体寄存器的定义和功能，请参考*附录 C-1 机器模式控制寄存器* 。

C908X 中扩展的超级用户模式控制寄存器如 表 4.7 所示。

表 4.7: C908X 扩展超级用户模式控制寄存器

名称	读写权限	寄存器编号	描述
超级用户模式处理器控制和状态扩展寄存器组			
sxstatus	超级用户模式读写	0x5C0	超级用户模式扩展状态寄存器
shcr	超级用户模式读写	0x5C1	超级用户模式硬件控制寄存器
scer2	超级用户模式只读	0x5C2	超级用户模式 L2Cache ECC 寄存器
scer	超级用户模式只读	0x5C3	超级用户模式 L1Cache ECC 寄存器
scountinhibit	超级用户模式读写	0x5C8	超级用户模式禁止计数寄存器
shpmcr	超级用户模式读写	0x5C9	超级用户模式性能监测控制寄存器
shpmsr	超级用户模式读写	0x5CA	超级用户模式监测起始触发寄存器
shpmer	超级用户模式读写	0x5CB	超级用户模式监测终止触发寄存器
sllcpid	超级用户模式读写	0x5CC	超级用户模式末级 cache 细粒度 ID 寄存器

续下页

表 4.7 - 接上页

名称	读写权限	寄存器编号	描述
sl2wp	超级用户模式读写	0x5CD	超级用户模式 L2 cache 细粒度回填配置寄存器
sbeaddr	超级用户模式读写	0x5CE	超级用户模式 L1 LD BUS ERR 地址寄存器
scycle	超级用户模式读写	0x5E0	超级用户模式周期计数器
……			
shpm-counter31	超级用户模式读写	0x5FF	超级用户模式计数器 31
超级用户模式 MMU 扩展寄存器组			
smir	超级用户模式读写	0x9C0	超级用户模式 MMU Index 寄存器
smel	超级用户模式读写	0x9C1	超级用户模式 MMU EntryLo 寄存器
smeh	超级用户模式读写	0x9C2	超级用户模式 MMU EntryHi 寄存器
smcir	超级用户模式读写	0x9C3	超级用户模式 MMU 控制寄存器

具体寄存器的定义和功能，请参考*附录 C-2 超级用户模式控制寄存器*

C908X 中扩展的用户模式控制寄存器如 表 4.8 所示。

表 4.8: C908X 扩展用户模式控制寄存器

名称	读写权限	寄存器编号	描述
用户模式扩展浮点控制寄存器组			
fxcr	用户模式读写	0x800	用户模式扩展浮点控制寄存器

具体寄存器的定义和功能，请参考*附录 C-3 用户模式控制寄存器* 。

4.7　VPU 相关的 CSR 寄存器

如 表 4.9 所示，本章节描述 C908X VPU 相关的 CSR 寄存器。

表 4.9: VPU 相关的 CSR 寄存器列表

名称	寄存器编号	读写权限	描述
vstart	0x008	用户模式读写	矢量起始位置寄存器
vxsat	0x009	用户模式读写	定点溢出标志位寄存器
vxrm	0x00A	用户模式读写	定点舍入模式寄存器
vcsr	0x00F	用户模式读写	矢量控制和状态寄存器
vl	0xC20	用户模式只读	矢量长度寄存器
vtype	0xC21	用户模式只读	矢量数据类型寄存器

续下页

表 4.9 - 接上页

名称	寄存器编号	读写权限	描述
vlenb	0xC22	用户模式只读	矢量位宽寄存器，以字节为单位
fflags	0x001	用户模式读写	浮点异常累积状态寄存器
frm	0x002	用户模式读写	浮点动态舍入模式控制寄存器
fcsr	0x003	用户模式读写	浮点控制状态寄存器
fxcr	0x800	用户模式读写	用户模式扩展浮点控制寄存器
utnmode	0x8DA	用户模式读写	2D saturation 寄存器
mtndebugpc	0x7CF	调试模式只读	Debug pc 扩展寄存器
mtnfastmba	0x7EB	机器模式读写	fast memory 地址范围配置寄存器
mtnsr	0x7E7	机器模式只读	状态地址寄存器
mtnlowpower	0x7EA	机器模式读写	low_power 寄存器

注意

1. misa.v = 0 时，vsetvl 指令需要上报非法指令异常（tval = opcode）。
2. 对于 fcsr、fflags、fxcr、mtnfastmba 寄存器：
   a. 下电时，需要通过先读后写寄存器 fcsr/fflags/fxcr/mtnfastmba 的方式同步数据到 Core 侧。
   b. 上电时，需要通过先读后写寄存器 fcsr/fflags/fxcr/mtnfastmba 的方式同步数据导 C908X VPU 侧。
3. 对于在 C908X VPU 侧管理的 CSR 寄存器，在 misa.v = 0 时，不会直接往 VPU 发相关 UOP。
   a. 对于 FCSR、FFLAGS、FXCR、MTNFASTMBA 寄存器，当 misa.v = 0 时，读写 csr 是不报异常的，支持读写，返回由 core 侧确定。
   b. 对于 VXSAT、VXRM、VCSR、VSTART、VL、VTYPE、VLENB 寄存器，当 misa.v = 0 或 vs = 0 时，读写 csr 上报非法指令异常（tval = opcode）。
   c. 对于 UTNMODE、MTNLOWPOWER、MTNDEBUGPC 自定义寄存器，当 misa.v = 0 时，读写 csr 是不报异常的，写 csr 无作用，读 csr 返回 0。

具体寄存器的定义和功能，请参考*附录 C-3 用户模式控制寄存器*和*附录 C-4 C908X VPU 访存类异常寄存器*。

4.8　多核与 L2 控制寄存器

4.8.1　多核与 L2 控制寄存器访问

C908X 多核与 L2 控制寄存器支持两种访问模式：

- 核心或一致性外设通过地址读写方式访问

- APB 调试接口访问

当通过地址读写方式配置寄存器时，寄存器的地址空间为 pad_cpu_apb_base[PA-1：27] 所配置地址的最后一个 4KB 页面，起始地址与结尾地址如下所示：

reg_addr_begin[PA-1:0] = {pad_cpu_apb_base[PA-1:27], 15'h7fff, 12'h0}

reg_addr_end[PA-1:0] = {pad_cpu_apb_base[PA-1:27], 15'h7fff, 12'hfff}

例如 pad_cpu_apb_base[39:0] 信号为 0x0008000000，则寄存器配置地址为 0x000ffff000 至 0x000fffffff。

当通过 APB 接口访问时，将根据地址偏移直接选中对应的配置寄存器，通过 APB 接口访问 L2CR 的方法如下所示：

1. Debug 接口读取公共寄存器 L2CR 全部内容（L2CR offset = 0x000）
   (1) 首先读取 L2CR 寄存器高 32 bit，通过 SoC 按照 APB 协议置位相关信号，其中地址线 pad_devif_paddr[14:0] 置位为 15'h004（L2CR 寄存器高半部分 offset 值）
   (2) 完成传输后采样 APB 接口数据信号 devif_pad_prdata[31:0] 即可得到 L2CR 寄存器高半部分的数据
   (3) 继续读取 L2CR 寄存器低 32 bit 数据，方式与读取高半部分一致，其中地址线 pad_devif_paddr[14:0] 置位为 15'h000（L2CR 寄存器低半部分 offset 值）
   (4) 完成传输后采样 APB 接口数据信号 devif_pad_prdata[31:0] 即可得到 L2CR 寄存器低半部分的数据
2. Debug 接口写公共寄存器 L2CR 全部内容
   (1) 首先写 L2CR 寄存器高 32 bit，通过 SoC 按照 APB 协议置位相关信号，其中地址线 pad_devif_paddr[14:0] 置位为 15'h004（L2CR 寄存器高半部分 offset 值）
   (2) 继续写 L2CR 寄存器低 32 bit，通过 SoC 按照 APB 协议置位相关信号，其中地址线 pad_devif_paddr[14:0] 置位为 15'h000（L2CR 寄存器低半部分 offset 值）

C908X 多核与 L2 控制寄存器仅支持以下几类访问属性：

RW：可读可写

RO：可读不可写

WO：可写不可读

RAZ：读为 0

WI：写无效

对于 RO 类型的寄存器，若发起写访问则 WI。

对于 WO 类型的寄存器，若发起读访问则 RAZ。

4.8.2　C908X 多核与 L2 控制寄存器访问权限

C908X 在进行寄存器访问时会受到访问权限的限制：

1. 对于核心发起的访问，仅支持机器模式下的访问，超级用户模式及用户模式下访问任何寄存器均读为 0 写无效。
2. 对于一致性外设，所有访问寄存器的请求均读为 0 写无效。

3. 调试接口无需鉴权，允许访问所有寄存器，若调试接口访问私有寄存器，默认访问核 0 的私有寄存器。

4.8.3　C908X 多核与 L2 控制寄存器汇总表

表 4.10: C908X 多核与 L2 控制寄存器

寄存器类别	寄存器名	寄存器功能描述	Offset
基础功能寄存器	L2CR	L2 Cache 设置信息寄存器	0x000
	L2SR	L2 Cache 状态信息寄存器	0x008
	L2MCR	L2 Cache 模式设定信息寄存器	0x010
	L2OPCR	L2 Cache 操作寄存器	0x018
	L2ER	L2 Cache 错误保留信息寄存器	0x020
	L2WP	L2 细粒度回填控制寄存器	0x070
	L2EIR	L2 ECC 注入寄存器	0x180
	L2DECR	L2 Cache 动态使能控制寄存器	0x300
	L2DESR	L2 Cache 动态使能状态寄存器	0x308
私有功能寄存器	SMPR	核心 Snoop 使能寄存器	0x810
	HPCPL2DA	L2 数据访问计数	0x900
	HPCPL2DM	L2 数据访问 MISS 计数	0x908
	HPCPL2IA	L2 指令访问计数	0x910
	HPCPL2IM	L2 指令访问 MISS 计数	0x918
	HPCPL2RVLD	L2 所有读类型请求数量计数	0x920
	HPCPL2RSTALL	L2 所有读类型请求阻塞周期计数	0x928

具体寄存器的定义和功能，请参考*附录 C-7 多核与 L2 控制寄存器*。

4.9　数据格式

4.9.1　整型数据格式

寄存器内部的数值并没有大小端之分，只有有符号和无符号的区别。其格式均为从右至左表示逻辑低位到高位的排布，如 图 4.2 所示。

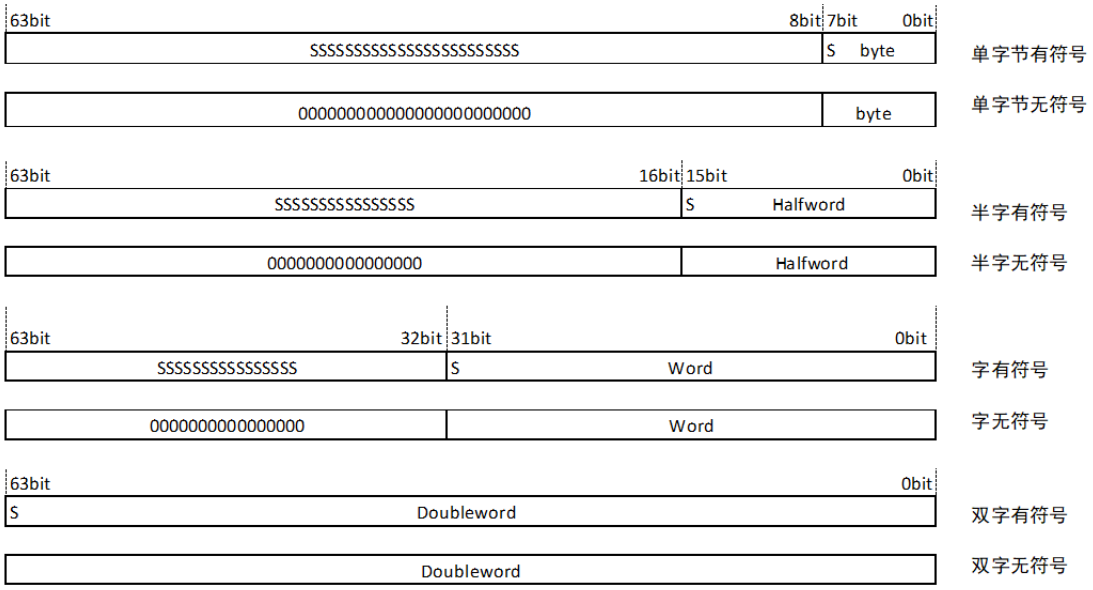


图 4.2: 寄存器中的整型数据组织结构

4.9.2　浮点数据格式

C908X 浮点单元遵从 RISC-V 标准，兼容 ANSI/IEEE 754-2008 浮点标准，支持半精度、单精度和 BF16 浮点运算，数据格式如 图 4.3 所示。其中，单精度数据仅使用 64 位浮点寄存器的低 32 位，高 32 位需要全为 1，否则会被当被非数处理；半精度和 BF16 数据仅使用 64 位浮点寄存器的低 16 位，高 48 位需要全为 1，否则会被当作非数处理。

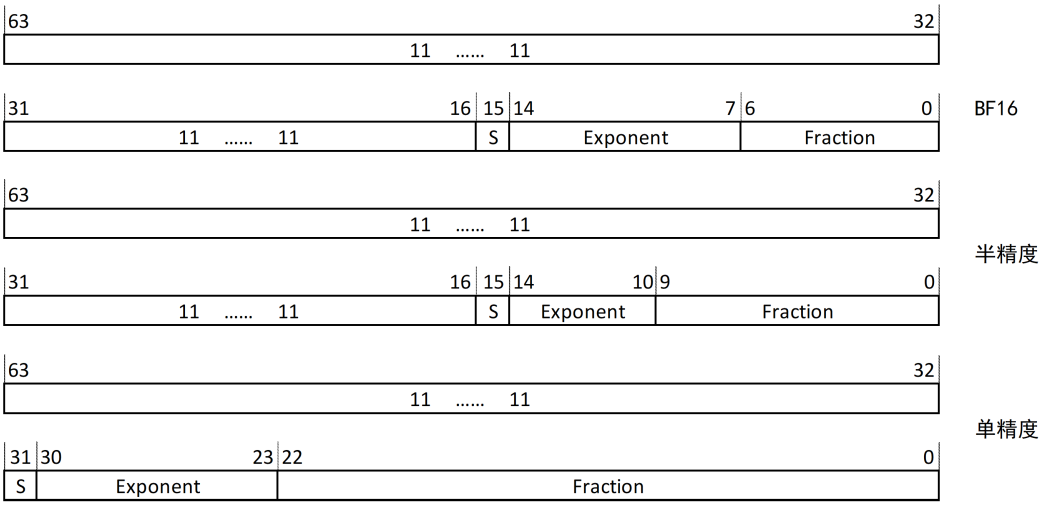


图 4.3: 寄存器中的浮点数据组织结构

4.9.3　矢量数据格式

矢量寄存器的宽度为 128 位或 256 位可配，矢量寄存器中包含的元素个数由当前矢量元素位宽决定。C908X 支持的矢量元素位宽为 8 位、16 位、32 位和 64 位。不同矢量元素位宽时矢量寄存器的数据排布如 图 4.4 所示。

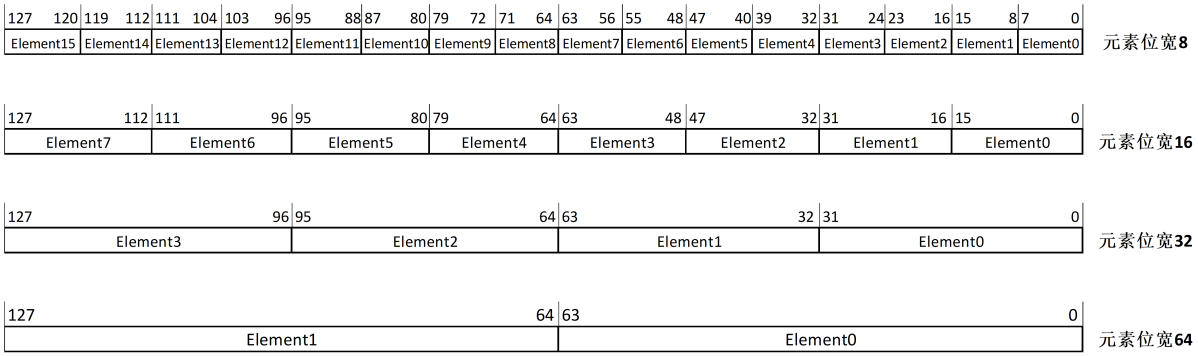


图 4.4: 寄存器中的矢量数据组织结构

4.10　大小端

大小端的概念是相对于存储器数据存储的格式而提出的。高地址字节存放至物理内存的低位被定义为大端；高地址字节存放至物理内存的高位被定义为小端，如 图 4.5 所示。

A+7	A+6	A+5	A+4	A+3	A+2	A+1	A	
Byte7	Byte6	Byte5	Byte4	Byte3	Byte2	Byte1	Byte0	Double word at A
Byte7	Byte6	Byte5	Byte4	Byte3	Byte2	Byte1	Byte0	Word at A
Byte7	Byte6	Byte5	Byte4	Byte3	Byte2	Byte1	Byte0	Half word at A
Byte7	Byte6	Byte5	Byte4	Byte3	Byte2	Byte1	Byte0	Byte at A

小端模式

A+4	A+5	A+6	A+7	A	A+1	A+2	A+3	
Byte3	Byte2	Byte1	Byte0	Byte7	Byte6	Byte5	Byte4	Double word at A
Byte4	Byte5	Byte6	Byte7	Byte3	Byte2	Byte1	Byte0	Word at A
Byte6	Byte7	Byte4	Byte5	Byte1	Byte0	Byte3	Byte2	Half word at A
Byte7	Byte6	Byte5	Byte4	Byte0	Byte1	Byte2	Byte3	Byte at A

大端V1模式

A+7	A+6	A+5	A+4	A+3	A+2	A+1	A	
Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7	Double word at A
Byte4	Byte5	Byte6	Byte7	Byte0	Byte1	Byte2	Byte3	Word at A
Byte6	Byte7	Byte4	Byte5	Byte2	Byte3	Byte0	Byte1	Half word at A
Byte7	Byte6	Byte5	Byte4	Byte3	Byte2	Byte1	Byte0	Byte at A

大端V2模式

图 4.5: 内存中的数据组织形式

C908X 仅支持小端模式，支持标准补码的二进制整数。每个指令操作数的长度既可以显式编码在程序中（load/store 指令），也可以隐含在指令操作中（index operation, byte extraction）。通常，指令接收 64 位操作数，产生 64 位结果。

5 异常与中断

5.1　概述

异常处理（包括指令异常和外部中断）是处理器的一项重要功能，在某些异常事件产生时，用来使处理器转入对这些事件的处理。这些事件包括硬件错误、指令执行错误、用户程序请求服务等。

异常处理的关键是在异常发生时，保存 CPU 当前运行的状态，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，CPU 硬件会保证后续指令不会改变 CPU 的状态。异常在指令的边界上被处理，即 CPU 在指令退休时响应异常，并保存退出异常处理时将被执行指令的地址。即使异常指令退休前被识别，异常也要在相应的指令退休时才会被处理。为了程序功能的正确性，CPU 在异常处理结束后要避免重复执行已执行完成的指令。

以在机器模式响应异常为例，具体步骤为：（这里的“异常”泛指指令异常和外部中断）

第一步：处理器保存 PC 到 mepc 中。

第二步：根据发生的异常类型更新 mcause 和 mtval。

第三步：将 mstatus 的中断使能位 MIE 保存到 MPIE 中，将 MIE 清零，禁止响应中断。

第四步：将发生异常之前的权限模式保存到 mstatus 的 MPP 中，切换到机器模式。

第五步：根据 mtvec 中的基址和模式，得到异常服务程序入口地址。处理器从异常服务程序的第一条指令处开始执行。

C908X 遵从 RISC-V 标准的异常向量表，如 表 5.1 所示。

表 5.1: 异常和中断向量分配

中断标记	异常向量号	描述
1	0	未实现
1	1	超级用户模式软件中断
1	2	保留
1	3	机器模式软件中断
1	4	未实现
1	5	超级用户模式计时器中断
1	6	保留
1	7	机器模式计时器中断

续下页

表 5.1 - 接上页

中断标记	异常向量号	描述
1	8	未实现
1	9	超级用户模式外部中断
1	10	保留
1	11	机器模式外部中断
1	13	性能检测溢出中断（如配置性能检测单元）
1	16	L1 数据缓存 ECC 中断（如配置 ECC） LSU 读总线 bus error 中断
1	其他	保留
0	0	未实现
0	1	取指令访问错误异常
0	2	非法指令异常
0	3	调试断点异常
0	4	加载指令非对齐访问异常
0	5	加载指令访问错误异常
0	6	存储/原子指令非对齐访问异常
0	7	存储/原子指令访问错误异常
0	8	用户模式环境调用异常
0	9	超级用户模式环境调用异常
0	10	保留
0	11	机器模式环境调用异常
0	12	取指页面错误异常
0	13	加载指令页面错误异常
0	14	保留
0	15	存储/原子指令页面错误异常
0	>= 16	保留

C908X 支持异常和中断的降级响应 (delegation)。在超级用户模式发生异常或者中断时，处理器需要切换到机器模式响应，模式切换会造成处理器性能损失。Delegation 机制支持配置中断和异常在超级用户模式响应。其中，机器模式下发生的异常不受 delegation 控制，只在机器模式响应。机器模式外部中断、机器模式软件中断、机器模式计时器中断不支持降级到超级用户模式响应，其他中断均可以被降级到超级用户模式态。在机器模式下不响应被降级的中断。

在超级用户模式和用户模式下均可响应所有符合条件的中断和异常。对于未被降级的中断和异常，进入机器模式进行处理，更新机器模式异常处理寄存器。对于被降级的中断和异常均在超级用户模式响应，更新超级用户模式异常处理寄存器。

5.2　异常

5.2.1　异常响应

以在机器模式响应异常为例，具体步骤为：（这里的“异常”特指非法指令，访问错误等事件）

第一步：处理器保存发生异常的 PC 到 mepc 中。

第二步：设置 mcause 的中断标记为 0，将异常编号写入 mcause，并按照 表 5.2 的规则更新 mtval。

第三步：将 mstatus 的中断使能位 MIE 保存到 MPIE 中，将 MIE 清零，禁止响应中断。

第四步：将发生异常之前的权限模式保存到 mstatus 的 MPP 中，切换到机器模式。

第五步：PC 从 mtvec.Base 处取指令并执行。通常，取回的指令是一条跳转指令，跳转至顶层处理函数。该函数通过分析 mcause 获取异常编号，并调用该编号对应的处理函数。

表 5.2: 异常发生时 mtval 的更新

异常向量号	异常类型	mtval 更新值
1	取指令访问错误异常	取指访问的虚拟地址
2	非法指令异常	指令码
3	调试断点异常	0
4	加载指令非对齐访问异常	加载访问的虚拟地址
5	加载指令访问错误异常	加载访问的虚拟地址
6	存储/原子指令非对齐访问异常	存储/原子访问的虚拟地址
7	存储/原子指令访问错误异常	存储/原子访问的虚拟地址
8	用户模式环境调用异常	0
9	超级用户模式环境调用异常	0
11	机器模式环境调用异常	0
12	取指页面错误异常	取指访问的虚拟地址
13	加载指令页面错误异常	加载访问的虚拟地址
15	存储/原子指令页面错误异常	存储/原子访问的虚拟地址

5.2.2　异常返回

执行 mret 指令可以实现异常返回。此时，处理器执行下列操作：

- 将 mepc 恢复到 PC。（mepc 保存的是发生异常的 PC。通过调整 mepc，可以跳过发生异常的指令；如果不调整，则重新执行发生异常的指令）
- 将 mstatus.MPIE 恢复到 mstatus.MIE。
- 从 mstatus.MPP 恢复发生异常之前的权限模式。

5.3　中断

5.3.1　中断优先级

当同时发生多个中断请求时，优先级按照下列顺序决定（从高到低）：

- L1 ECC 中断
- M 态外部中断
- M 态软件中断
- M 态计时器中断
- S 态外部中断
- S 态软件中断
- S 态计时器中断
- PMU 溢出中断
- L1 ECC 中断（被降级）
- S 态外部中断（被降级）
- S 态软件中断（被降级）
- S 态计时器中断（被降级）
- PMU 溢出中断（被降级）

5.3.2　中断响应

以在机器模式响应中断为例，具体步骤为：

第一步：处理器执行完当前指令，保存下一条指令的 PC 到 mepc 中。

第二步：设置 mcause 的中断标记为 1，将向量号写入 mcause，并更新 mtval 为 0。

第三步：将 mstatus 的中断使能位 MIE 保存到 MPIE 中，将 MIE 清零，禁止响应中断。

第四步：将发生中断之前的权限模式保存到 mstatus 的 MPP 中，切换到机器模式。

第五步（mtvec.Mode=0，直通中断）：PC 从 mtvec.Base 处取指令并执行。通常，取回的指令是一条跳转指令，跳转至顶层处理函数。该函数通过分析 mcause 获取向量号，并调用该编号对应的处理函数。

第五步（mtvec.Mode=1，矢量中断）：PC 从 mtvec.Base + 4 * 向量号处取指令并执行。通常，取回的指令是一条跳转指令，跳转至相应中断的处理函数。

5.3.3　中断返回

执行 mret 指令可以实现中断返回。此时，处理器执行下列操作：

- 将 mepc 恢复到 PC。（mepc 保存的是下一条指令的 PC，所以无需调整）
- 将 mstatus.MPIE 恢复到 mstatus.MIE。

- 从 mstatus.MPP 恢复发生中断之前的权限模式。

5.3.4　异步错误

在极少数情况下，处理器可能表现出“异步错误”的行为。“异步错误”指某条指令导致的错误没有在该指令退休时显现出来。例如，CPU 执行了一条 load 指令，总线返回 error。由于流水线具有指令快速退休的特性，当总线产生 error 时，load 指令已经退休。

值得注意的是，“异步错误”在实际系统中发生的概率极低，一旦发生，则意味着系统出现了 fatal 错误。C908X 中，ecc 校验错误或总线错误属于“异步错误”，产生后通过中断的形式上报这种“异步错误”。

6 内存模型

6.1　内存模型概述

6.1.1　内存属性

玄铁 CPU 支持两种内存类型，分别是内存（Memory）和外设（Device），通过 Strong Ordered(SO) 位进行地址属性区分。一般将外设地址空间配置为 so 的地址属性，常用的外设空间如 APB 空间。

注意

SO 区域不能存放指令和页表。

Memory 类型的内存支持投机执行和乱序执行，需要配置成 WeakOrder 属性（SO 比特位设置为 0）。根据是否可高缓（Cacheable, C）可进一步分为可高缓内存（Cacheable memory）和不可高缓内存（Non-cacheable memory）。Device 类型的内存特点是不可投机执行且必须按序执行，因此 Device 一定是带有 Non-cacheable 地址属性。

Bufferable（B）属性表示写访问允许在某个中间节点快速返回写响应；反之，Non-bufferable 表示写访问只有在最终设备真正写完成后才返回写响应。因此，Memory 地址属性均支持 Bufferable，对于 Device 可以根据是否可缓分为可缓存外设（Bufferable device）和不可缓存外设（Non-bufferable device）。

为了支持多核之间数据共享，C908X 增加了可共享的页面属性（Shareable，SH）。对于可共享的页面，表示该页面在多核间共享，由硬件维护数据的一致性；对于不可共享的页面，表示被某个单核独占，不要求硬件维护数据的一致性。不可共享页面的多核数据一致性需要软件来维护。可高缓内存可以配置 SH 的属性，而不可高缓内存类型和外设类型的 SH 属性不可配置，固定为可共享。

另外，C908X 还支持配置安全的页面属性（Security，SEC）。如果对于页面安全属性没有特殊要求，默认配置为 0。

表 6.1 给出了各个内存类型对应的页面属性。

表 6.1: 内存类型分类

内存类型	SO	C	B	SH	SEC
可高缓内存	0	1	可配	可配	Reserved
不可高缓内存	0	0	可配	1	Reserved

续下页

表 6.1 - 接上页

内存类型	SO	C	B	SH	SEC
可缓存外设	1	0	1	1	Reserved
不可缓存外设	1	0	0	1	Reserved

6.1.1.1　SYSMAP 配置说明

客户可以根据自身需要，通过对 sysmap.h 扩展配置文件的修改，实现对不同地址段的页面属性的定义。

C908X 同时支持利用端口对地址属性进行配置，具体可见*PMA 地址范围配置* 小节。

sysmap.h 支持对 8 个地址空间的属性设定。第 i(i=0~7) 个地址空间地址上限（不包含）由宏 $SYSMAP\_BASE\_ADDR_i$ 定义，地址下限（包含）由 $SYSMAP\_BASE\_ADDR_{i-1}$ 定义，即：

$SYSMAP\_BASE\_ADDR_{i-1}$ <= 第 i 个地址空间地址 < $SYSMAP\_BASE\_ADDR_i$

其中，第 0 个地址空间的下限是默认为 0x0；内存地址不在 sysmap.h 文件设定的 8 个地址区间的地址属性默认为 strong order + non-bufferable + shareable +security。

每个地址空间的上下边界都是 4KB 对齐，因此，宏 $SYSMAP\_BASE\_ADDR_i$ 定义的是地址的高 28 位。落在第 i(i=0~7) 个地址空间内的地址的属性由宏 $SYSMAP\_FLAG_i$ (i=0~7) 定义，属性的 bit 分布如 图 6.1 所示：

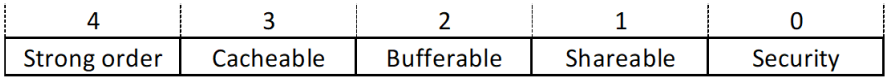


图 6.1: sysmap.h 地址属性格式

6.1.1.2　SYSMAP 配值参考

- 第 0 段地址空间属性定义：40'h0 <= addr0[39:0] < 40'h01000 000, flg0 = 5'b01111。该段地址空间包括 INST_RAM 和 DATA_RAM 的一部分，配置成 cacheable 属性。属性定义如下：

```
`define SYSMAP_BASE_ADDR0    28'h01000

`define SYSMAP_FLG0          5'b01111
```

- 第 1 段地址空间属性定义：40'h01000 000 <= addr1[39:0] < 40'h02000 000, flg1 = 5'b10010。该段地址空间主要对应 DATA_RAM 的另一部分和特殊函数写地址（如 print 函数），配置成 so 属性，保证操作一定可以从 core 发出到总线上。属性定义如下：

```
`define SYSMAP_BASE_ADDR1    28'h02000

`define SYSMAP_FLG1          5'b10010
```

- 第 2 段地址空间属性定义：40'h02000 000 <= addr2[39:0] < 40'hd0000 000, flg2=5'b10010。该段地址空间主要分配为 APB 地址空间，保证操作严格按序执行。属性定义如下：

```
`define SYSMAP_BASE_ADDR2    28'hd0000

`define SYSMAP_FLG2          5'b10010
```

- 第 3 段地址空间属性定义：40'hd0000 000 <= addr3[39:0] < 40'heffff 000, flg3=5'b01101。该地址段配置为 non-shareable 的缓存空间，仅供 SMART SOC 内部使用。属性定义如下:

```
`define SYSMAP_BASE_ADDR3    28'heffff

`define SYSMAP_FLG3          5'b01101
```

- 第 4 段地址空间属性定义：40'heffff 000 <= addr4[39:0] < 40'hffffff 000, flg4=5'b01111。该地址段在 SMART 上对应一段无效的 RAM，没有实际意义。属性定义如下:

```
`define SYSMAP_BASE_ADDR4    28'hffffff

`define SYSMAP_FLG4          5'b01111
```

- 第 5 段地址空间属性定义：40'h02000 000 <= addr5[39:0] < 40'h4000000 000, flg5=5'b01111。该地址段在 SMART 上对应一段无效的 RAM，没有实际意义。属性定义如下:

```
`define SYSMAP_BASE_ADDR5    28'h4000000

`define SYSMAP_FLG5          5'b01111
```

- 第 6 段地址空间属性定义：40'h4000000 000 <= addr6[39:0] <40'h5000000 000, flg6=5'b10010。该地址段在 SMART 上对应一段无效的 RAM，没有实际意义。属性定义如下:

```
`define SYSMAP_BASE_ADDR6    28'h5000000

`define SYSMAP_FLG6          5'b10010
```

- 第 7 段地址空间属性定义：40'h5000000 000 <= addr7[39:0] < 40'hffffffff 000, flg7=5'b01111。该地址段在 SMART 上对应一段无效的 RAM，没有实际意义。属性定义如下:

```
`define SYSMAP_BASE_ADDR7    28'hffffffff

`define SYSMAP_FLG7          5'b01111
```

6.1.2　内存一致性模型

C908X 采用宽松的内存一致性模型（Weak Memory Ordering），该模型具体定义如下：

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

由于 Weak Memory Ordering 的模型，导致多核之间内存实际的读写顺序和程序给定的访问顺序会不一致。因此，C908X 扩展了 SYNC 指令供软件强制规定内存访问的顺序性。

SYNC 指令限定了所有指令的执行顺序，保证了 SYNC 指令之前的所有指令一定在 SYNC 指令执行之前完成。另外，SYNC 指令还可以额外同步指令内存，即在 SYNC 指令前序指令完成时清空流水线，重新取指。具体指令如 表 6.2 所示。

表 6.2: SYNC 指令描述

助记符	指令描述	作用域
SYNC.IS	Synchronize data and instruction memory	Shareable
SYNC.I	Synchronize data and instruction memory	Non-shareable
SYNC.S	Synchronize data memory	Shareable
SYNC	Synchronize data memory	Non-shareable

6.2　虚拟内存管理

6.2.1　MMU 概述

C908X MMU（Memory Management Unit）兼容 RISC-V SV39/SV48 标准。其作用主要有:

- **地址转换**：将虚拟地址（39 位/48 位）转换成物理地址（40 位/48 位）。
- **页面保护**：通过对页面的访问者进行读/写/执行权限检查。
- **页面属性管理**：扩展地址属性位，根据访问地址，获取页面对应属性，供系统进一步使用。

C908X 中 SXLEN 固定为 64 位，MMU 按照 64 位虚拟地址进行 SV39/SV48 的地址转换。

6.2.2　PMA 地址范围配置

C908X VPU PMA 地址范围可配，支持软件 16 个 PMA 地址范围可配和硬件默认：

- 硬件默认有 16 个区间
- 这 16 个区间也可以通过软件进行配置
- 地址区间按照 4KB 对齐
- 0 地址为内存地址空间的下限

- sysmap_base_addr$_{i-1}$ <= 第 i 个区间的地址 < sysmap_base_addr$_i$
- 内存地址不在 16 个地址区间的默认为：SO、non_cacheable、non_bufferable
- pma0 ~ 16_attri_addr 和 offset 按照 4KB 对齐
  - PA40：只有 [27:0] 有效
  - PA48：只有 [35:0] 有效
- [48：44]bit 为对应区间的属性信息（低 5bits）如下 表 6.3 所示：

表 6.3: PMA 地址 [48：44] 对应区间的属性信息

[48:44]	属性
bit48	strong order
bit47	cacheable
bit46	bufferable
bit45	shareable
bit44	security

C908X 在每个 cluster 有 16*49 bits 的信息作为顶层输入，广播到每个 core 的 MMU，其中地址是以 4 KB 对齐，即需设置为预期地址右移 12 bit 的值。具体如 表 6.4 所示。此外，该组信号需要在复位释放前配置完毕。

表 6.4: SYSMAP PMA 信号列表

顶层输入信号名称	描述
pad_cpu_sysmap0[48:0]	[43:0]：地址段 0 的地址上限（不包含） [48:44]：属性信息
pad_cpu_sysmap1[48:0]	[43:0]：地址段 1 的地址上限（不包含） [48:44]：属性信息
pad_cpu_sysmap2[48:0]	[43:0]：地址段 2 的地址上限（不包含） [48:44]：属性信息
pad_cpu_sysmap3[48:0]	[43:0]：地址段 3 的地址上限（不包含） [48:44]：属性信息
pad_cpu_sysmap4[48:0]	[43:0]：地址段 4 的地址上限（不包含） [48:44]：属性信息
pad_cpu_sysmap5[48:0]	[43:0]：地址段 5 的地址上限（不包含） [48:44]：属性信息
pad_cpu_sysmap6[48:0]	[43:0]：地址段 6 的地址上限（不包含） [48:44]：属性信息
pad_cpu_sysmap7[48:0]	[43:0]：地址段 7 的地址上限（不包含） [48:44]：属性信息

续下页

表 6.4 - 接上页

顶层输入信号名称	描述
pad_cpu_sysmap8[48:0]	[43:0]：地址段 8 的地址上限（不包含） [48:44]：属性信息
pad_cpu_sysmap9[48:0]	[43:0]：地址段 9 的地址上限（不包含） [48:44]：属性信息
pad_cpu_sysmap10[48:0]	[43:0]：地址段 10 的地址上限（不包含） [48:44]：属性信息
pad_cpu_sysmap11[48:0]	[43:0]：地址段 11 的地址上限（不包含） [48:44]：属性信息
pad_cpu_sysmap12[48:0]	[43:0]：地址段 12 的地址上限（不包含） [48:44]：属性信息
pad_cpu_sysmap13[48:0]	[43:0]：地址段 13 的地址上限（不包含） [48:44]：属性信息
pad_cpu_sysmap14[48:0]	[43:0]：地址段 14 的地址上限（不包含） [48:44]：属性信息
pad_cpu_sysmap15[48:0]	[43:0]：地址段 15 的地址上限（不包含） [48:44]：属性信息

6.2.3　TLB 组织形式

MMU 主要利用 TLB (Translation Look-aside Buffer) 来实现上述功能。TLB 将 CPU 访存所使用的虚拟地址作为输入，转换前检查 TLB 的页属性，再输出该虚拟地址所对应的物理地址。

C908X MMU 采用两级 TLB，第一级为 uTLB，分别为指令 I-uTLB 和数据 D-uTLB，第二级为 jTLB。在处理器复位后，硬件会将 uTLB 和 jTLB 的所有表项进行无效化操作，软件无需初始化操作。

I-uTLB 有 32 个全相联表项，可以混合存储 4K、2M、1G 和 512G 四种大小的页面，取指请求命中 I-uTLB 时，当前时钟周期可以得到物理地址和相应权限属性。

D-uTLB 有 17 个全相联表项，可以混合存储 4K、2M、1G 和 512G 四种大小的页面，加载和存储请求命中 D-uTLB 时，下一个时钟周期可以得到物理地址和相应权限属性。

jTLB 为指令和数据共用，4 路组相联结构，表项大小 512 和 1024 可配，可以混合存储 4K、2M、1G 和 512G 四种大小页面。uTLB 缺失，jTLB 命中时，最快 3 个 cycle 返回物理地址和相应权限属性。

6.2.4　页表格式

MMU 的主要功能是将虚拟地址转换为物理地址并进行相应的权限检查。具体的地址映射关系和相应权限由操作系统进行配置，存放于页表中。页表用于存储下级页表的入口地址或者最终页表的物理信息，每个表项即 PTE 的结构如下：

标准模式：

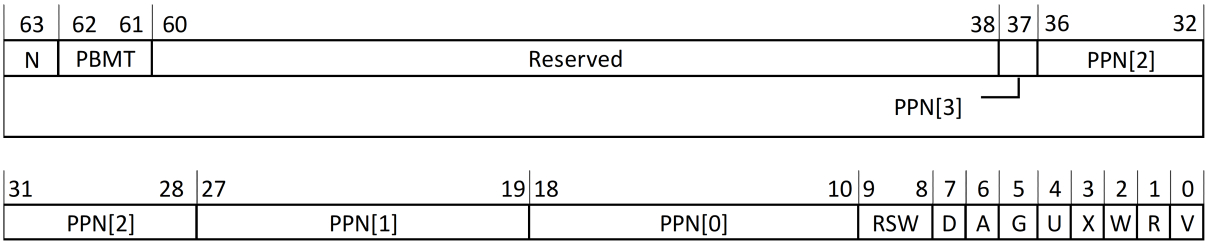


图 6.2: C908X 页表项（PA40 标准模式）

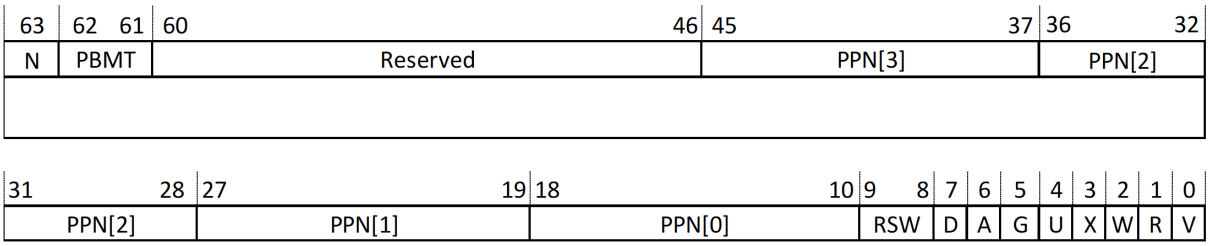


图 6.3: C908X 页表项（PA48 标准模式）

XuanTie 模式：

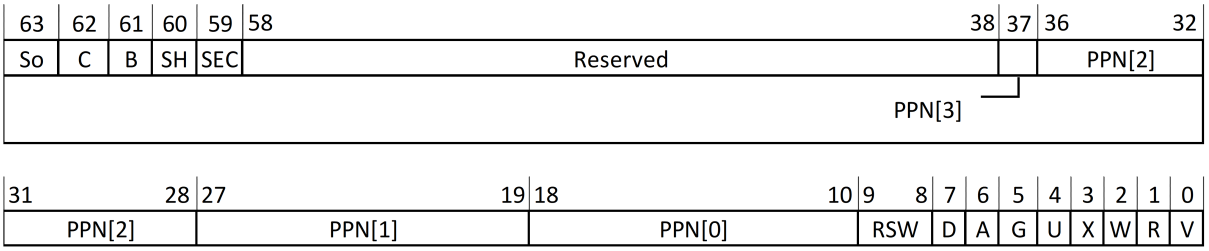


图 6.4: C908X 页表项（XuanTie 模式）

注意

fast memory 不支持存放页表

页表主要组成部分的含义为：

1. 页面基本属性

PTE[9:0] 为页表的基本属性位，各 bit 功能如下所述。

2. RSW –Reserved for Software

用于预留给软件做自定义页表功能的位。Default 为 2'b00。

3. D –Dirty

表示该页是否可写。

1'b0: 当前页不可写，写操作触发 Page Fault (对应访问类型) 异常。

1'b1: 当前页可写。

4. A –Accessed

表示该页是否可访问。

1'b0: 当前页不可访问，任何访问触发 Page Fault (对应访问类型) 异常。

1'b1: 当前页可访问。

5. G –Global

全局页面标识。

1'b0: 非共享页面，当前页面为对应 ASID 进程号私有。

1'b1: 共享页面，当前页可供多个进程共享。

6. U –User

是否为用户模式下可访问页表。

1'b0: 仅超级用户模式下可访问，当用户模式访问时，触发 Page Fault (对应访问类型) 异常。

1'b1: 仅用户模式下可访问，当超级用户模式访问时，触发 Page Fault (对应访问类型) 异常。

7. X W R –可执行，可写，可读

XWR bit 组合的含义规定为：

表 6.5: XWR bit 组合的含义

X	W	R	含义
0	0	0	当前页表不是叶子页表
0	0	1	可读页面
0	1	0	保留配置，page fault
0	1	1	可读、写页面
1	0	0	可取指页面
1	0	1	可读、取指页面
1	1	0	保留配置，page fault
1	1	1	可读、写、取指页面

8. 页面物理地址

PTE[37:10]/PTE[45:10] 表示页表的物理地址 PPN。C908X 的物理地址为 40-bit 时，PPN 占据 28-bit 空间；C908X 的物理地址为 48-bit 时，PPN 占据 36-bit 空间。每 9-bit 分为一段，PPN[i] 分别代表多级页表转换时所对应的 PPN 值。

9. 页面扩展属性

页表的扩展属性分为两种模式：标准扩展模式和 XuanTie 扩展模式。

当 mxstatus.maee 为 0 时，页表组织遵循标准扩展模式。标准扩展模式兼容 SVNAPOT 和 SVPBMT 两个标准扩展特性，分别用 PTE[63] 的 N 位和 PTE[62:61] 的 PBMT 位来实现。

N 位表示当前表项为一个 NAPOT 尺寸扩展表项，表项所覆盖的尺寸为一个 2 的幂指数所表示的连续区域，具体大小由下表定义。

表 6.6: NAPOT 扩展页面尺寸

i	pte.ppn[i]	Description	pte.napot bits
0	x xxxx xxx1	Reserved	-
0	x xxxx xx1x	Reserved	-
0	x xxxx x1xx	Reserved	-
0	x xxxx 1000	64 KiB contiguous region	4
0	x xxxx 0xxx	Reserved	-
>=1	x xxxx xxxx	Reserved	-

目前标准中只定义了 64KiB 的情况，N bit 有效时，若 PPN 的值使用其他情况，则被认为是一种 page fault，表项内容无效。

PBMT 位表示虚拟地址的 PMA 属性，具体定义按场景划分，如下 表 6.7 所示。

表 6.7: PBMT 的定义

Mode	Value	Requested Memory Attibutes
PMA	0	None
NC	1	Non-cacheable, idempotent, weakly-ordered (RVWMO or RVTSO), main memory
IO	2	Non-cacheable, non-idempotent, strongly-ordered (I/O ordering), I/O
-	3	Reserved for future standard use

- PMA 模式兼容原来未定义虚拟地址 PMA 时的系统属性设定，遵循原物理地址 PMA 属性，即 sysmap 属性；
- NC 模式下，强制覆盖不可缓存、幂等、弱排序属性，一般用于主存；
- IO 模式下，强制覆盖不可缓存、不幂等、强排序属性，一般用于外设；
- PBMT 位为 3 时的情况为保留模式，未来进行定义。
- SVPBMT 中未涉及到的 PMA 属性，以原来物理地址的 PMA 为准，强制覆盖的属性，以 PBMT 为准。

当 mxstatus.maee 为 1 时，页表组织遵循 XuanTie 扩展模式。PTE[63:59] 存储页面的内存属性，各 bit 定义见 表 6.1。

6.2.5　地址转换流程

SV39/48 标准规定的页表表项获取步骤为：

发生缺失页面的虚拟页面号 VPN，按照 9-bit 划分为 3 部分 (SV39) 或者 4 部分 (SV48)，最高部分为 VPN[max]，最低部分位 VPN[min]。

1. 根据 SATP.PPN 和 VPN[max] 得到一级页表访存地址 {satp.ppn, vpn[max], 3'b0}，检查地址是否具有 PMP 读权限，若不可读则产生相应 access fault 异常，若无异常则使用该地址访问 LSU，得到 64-bit 一级 PTE;
2. 当 PTE V 位有效时，根据上表所示的规则判断 X/W/R-bit 是否符合叶子页表条件，若符合叶子页表条件则说明已经找到最终物理地址，到第 3 步；若不符合则回到第 1 步，使用 pte.ppn 代替 satp.ppn，vpn 换为下一级 vpn，再拼接 3'b0 继续第一步流程；
3. 得到叶子页表，则检查页表是否有效：

- 访问类型违反 A/D/X/W/R/U-bit 的设置，产生对应的 page fault 异常；
- 直到使用 VPN[min] 访问结束仍未得到叶子页表，则产生对应的 page fault 异常；
- 访问 LSU 过程中得到总线 error 响应，产生 access fault 异常；
- 访问次数少于 3 次，则说明得到大页表，检查大页表的 PPN 是否按照页表尺寸对齐，若未对齐，则产生 page fault 异常。

若以上检查均已通过，则得到有效的页表内容。

叶子页表的内容（即由虚拟地址转换得到的物理地址和相应的权限属性）被缓存于 TLB 内以加速地址转换。

6.2.6　系统控制寄存器

C908X MMU 除了支持标准的 SATP 寄存器以外，还自定义扩展了 SMIR、SMCIR、SMEL 和 SMEH 控制寄存器，用户可以通过这几个扩展寄存器，直接对 TLB 进行读写、查询和无效操作。

6.2.6.1　MMU 地址转换寄存器 (SATP)

SV39/48 标准定义了 SATP CSR 作为 MMU 的配置寄存器，寄存器宽度为 64-bit，M/S 态可读写，U 态不可读写。

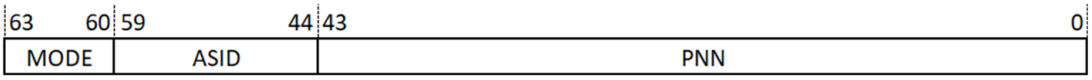


图 6.5: SATP 寄存器说明

MODE - 虚拟地址转换模式

MODE 域表示虚拟地址转换模式，标准编码规定如下表。C908X 实现 SATP MODE 域的 Bare(4'h0)、SV39(4'h8)、SV48(4'h9) 三种模式。

Linux 操作系统在每次系统 boot 时，会确定当前系统使用 SV39 还是 SV48 模式。一旦确定，就不会再有 39 和 48 之间的切换，只会在 Bare 和 SV39/48 之间切换。

ASID –当前进程号

ASID 域用于标识操作系统当前的进程 ID 号，当地址翻译请求产生 jTLB miss 后进行硬件回填时，jTLB 使用产生 miss 请求时的 ASID 值，拼接页表内容，回填至表项内。

PPN –硬件回填根 PPN

PPN 域定义硬件回填的根 PPN，用于构成硬件回填发起第一级页表的高位地址，具体硬件回填定义流程见*地址转换流程*。

表 6.8: MMU 地址翻译模式

Value	Name	Description
0	Bare	No translation or protection
1-7	-	Reserved
8	SV39	Page-based 39-bit virtual addressing
9	SV48	Page-based 48-bit virtual addressing
10	SV57	Reserved for page-based 57-bit virtual addressing
11	SV64	Reserved for page-based 64-bit virtual addressing
12-13	-	Reserved for standard use
14-15	-	Designated for custom use

6.2.6.2　MMU 控制寄存器 (SMCIR)

SMCIR 寄存器实现对 MMU 进行多种操作，包括 TLB 查找、TLB 读写、TLB 无效等操作。

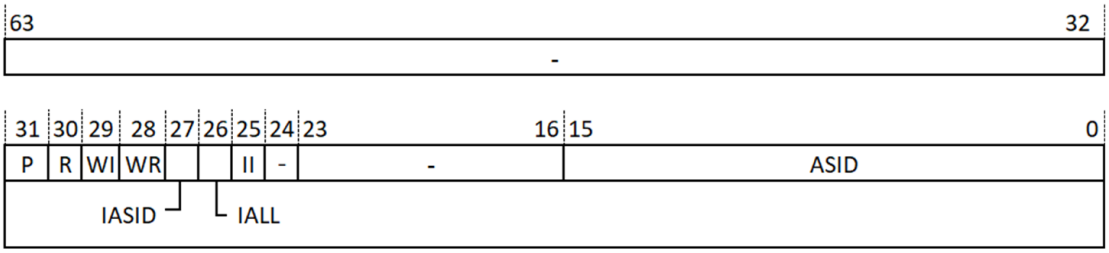


图 6.6: SMCIR 寄存器说明

TLBP：TLB 查询

根据 EntryHi 寄存器去查询 TLB。

当查询命中时，用 TLB 的序号去更新 Index 寄存器。

TLBR：TLB 读

根据 Index 寄存器索引，读出对应的 TLB 表项的值，并用这些值来更新 SMEH 和 SMEL 寄存器。

TLBWI：TLB 索引写

根据 Index 寄存器索引值，将寄存器 SMEH 和 SMEL 写入 TLB 对应表项。

TLBWR：TLB 随机写

根据 Random 寄存器索引值，将寄存器 SMEH，SMEL 写入 TLB 对应表项。

TLBIASID：TLB 根据 ASID 无效

所有匹配 ASID 的 TLB 表项全部无效。

TLBIALL：TLB 初始化

将所有 TLB 表项无效，初始化。

TLBII：TLB 根据索引无效

根据 Index 寄存器索引值，将对应的 TLB 表项无效。

ASID：ASID 号

TLBIASID 操作使用该 ASID 做匹配。

6.2.6.3　MMU Index 寄存器 (SMIR)

MMU 索引寄存器，用于索引 TLB。在进行 TLB 查询时，会更新命中表项的 index。在 TLB 写索引时，通过写 SMIR 的 index 域，可以将映射关系写入 jTLB 中对应 index 处。

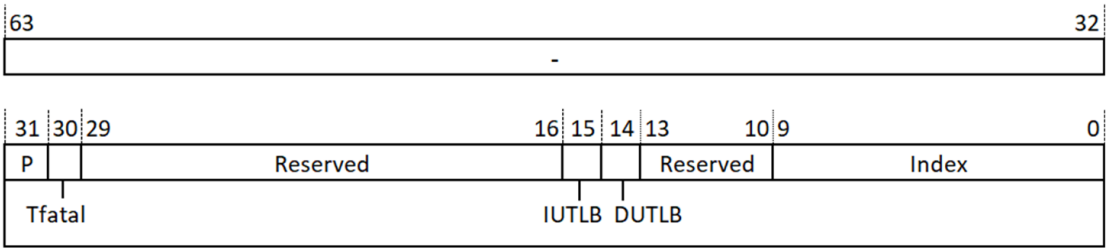


图 6.7: SMIR 寄存器说明

P –Probe Failure

0: TLBP 查询匹配命中。

1: TLBP 查询没命中。

Tfatal –Probe multiple

执行 TLBP 指令时，是否发生多重匹配。

0: 没有多重匹配。

1: 发生多重匹配。

Iutlb - Instruction Micro TLB

表示查询指令 uTLB。

Dutlb - Data Micro TLB

表示查询数据 uTLB。

Index –TLB Index

512-entry 配置：Index[8:7] 为 way 索引，Index[6:0] 为 set/entry 索引；（4 way，128 entries）

1024-entry 配置：Index[9:8] 为 way 索引，Index[7:0] 为 set/entry 索引；（4 way，256 entries）

6.2.6.4　MMU EntryHi 寄存器 (SMEH)

SMEH 寄存器用来保存 TLB 访问的虚拟地址信息，在 TLB 读时硬件更新，在软件写 TLB 表项之前由软件预先写入。

- TLB 查询时提供待查询页面的信息；
- TLB 读取时提供读取到页面的信息；
- TLB 写入时提供待写入页面的信息。

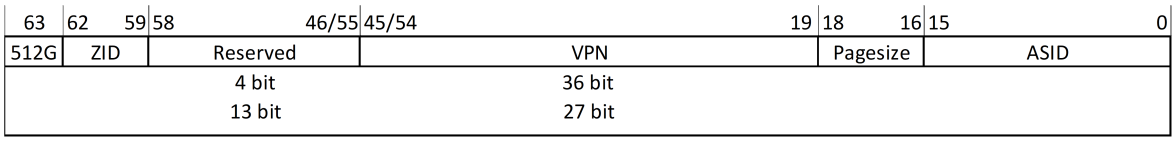


图 6.8: SMEH 寄存器说明

512G :

在 SV48 下，表示页面大小为 512G。

ZID :

在 TEE 配置下，表示页面对应的 ZoneID，该域在 TLB 读时硬件更新，在软件写 TLB 表项之前由软件预先写入。

VPN :

虚拟页帧号，该域在 TLB 读时硬件更新，在软件写 TLB 表项之前由软件预先写入。

Pagesize :

页面尺寸，Onehot 从低到高表示 4K，2M，1G 页面大小。

该域在 TLB 读时硬件更新，在软件写 TLB 表项之前由软件预先写入。

ASID :

地址空间标识。

6.2.6.5　MMU EntryLo 寄存器 (SMEL)

SMEL 包含 TLB 访问的物理地址和页面属性信息。

SMEL 的主要内容和页表相同，相关 bit 定义见*页表格式* 。SMEL[57:55] 从低到高用以存储页面的 PMP R/W/X 信息。

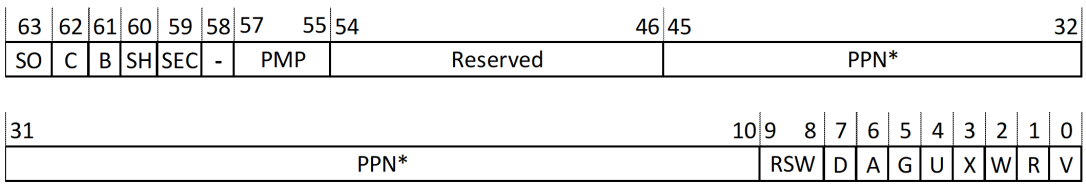


图 6.9: SMEL 寄存器说明

备注

在 PA48 时，PPN 整个域有效。在 PA40 时，bit[45:38] 恒为 0, 读为 0, 写 TLB 时忽略。

6.3　MMU 奇偶校验

MMU 支持可配置的奇偶校验，可以对 jTLB 的 TAG 和 DATA 进行校验。开启校验机制后，jTLB 在写入时对数据进行奇偶编码，在读取时进行校验。当检测到 1 bit 错误时，能够上报校验错误信息，并无效掉 jTLB 中出错的缓存行，同时此次请求当作 jTLB Miss 处理，发起硬件 Page Table Walk 并进行回填。软件可以查询 MCER/SCER 寄存器获取错误的相关信息，例如是否产生 jTLB 校验错误以及错误的位置信息。具体控制寄存器说明可以参考有关 MCER/SCER 的描述。1 bit 以上的错误无法检测或纠正。C908X MMU 支持软件注入错误功能，具体控制寄存器说明可以参考有关 MEICR 的描述。

6.4　物理内存保护

6.4.1　PMP 概述

C908X PMP（Physical Memory Protection）遵从 RISC-V 标准。PMP 单元负责对物理地址的访问权限进行检查，判定当前工作模式下 CPU 是否具备对该地址的读/写/执行权限。

C908X PMP 单元的主要特征有：

- 可配置 8/16/32/64 个 PMP 表项，每个表项通过 0-63 的号码来标识和索引
- 地址划分最小粒度为 4KB
- 支持 OFF、TOR、NAPOT 三种地址匹配模式，不支持 NA4 匹配模式
- 支持可读、可写、可执行三种权限的配置
- PMP 表项支持软件 Lock
- 可额外配置 EPMP 功能

6.4.2　PMP 控制寄存器

PMP 表项主要由一个 8 比特的设置寄存器和一个 64 比特的地址寄存器构成，所有 PMP 控制寄存器都只能在机器模式下访问，其他模式访问将产生非法指令异常。

6.4.2.1　物理内存保护设置寄存器（PMPCFG）

如 图 6.10 和 图 6.11 ，物理内存保护设置寄存器提供 PMP 表项的权限设置。一个 PMPCFG 寄存器可以覆盖 8 个表项的权限设置。

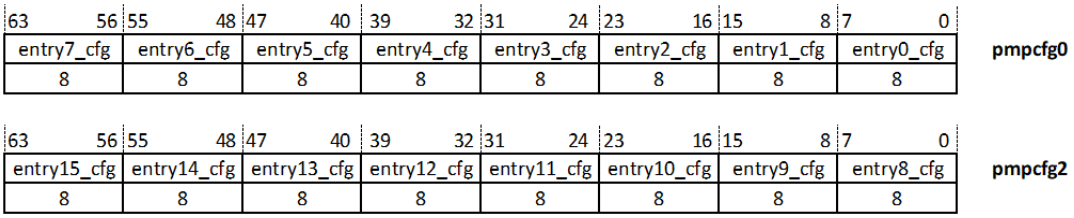


图 6.10: 物理内存保护设置寄存器整体分布

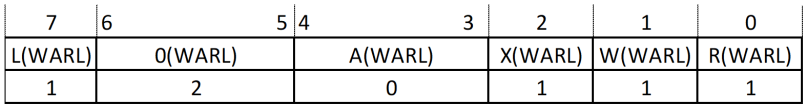


图 6.11: 物理内存保护设置寄存器

PMP 控制寄存器的具体描述如 表 6.9 所示。

表 6.9: PMP 控制寄存器描述

位	名称	描述
0	R	表项的可读属性： **0**：表项匹配地址不可读 **1**：表项匹配地址可读
1	W	表项的可写属性： **0**：表项匹配地址不可写 **1**：表项匹配地址可写
2	X	表项的可执行属性： **0**：表项匹配地址不可执行 **1**：表项匹配地址可执行
4:3	A	表项的地址匹配模式： **00**：OFF，无效表项 **01**：TOR（Top of range），使用相邻表项的地址作为匹配区间的模式 **10**：NA4（Naturally aligned four-byte region），区间大小为 4 字节的匹配模式，该模式不支持 **11**：NAPOT（Naturally aligned power-of-2 regions），区间大小为 2 的幂次方的匹配模式，至少为 4KB
7	L	表项的 Lock 使能位： **0**：机器模式的访问都将成功 系统模式/用户模式的访问根据 R/W/X 判定是否成功 **1**：表项被锁住，无法对相关表项进行修改 当配置 TOR 模式，其前一个表项的地址寄存器也无法被修改 所有模式都需要根据 R/W/X 判定是否访问成功

对于 TOR 模式，假设访问地址为 A，则其命中表项 i 的条件为：pmpaddr(i-1) <= A < pmpaddr(i)。对于 0 号表项，其使用 0 作为下边界。

对于 NAPOT 模式，其地址与区间大小的关系如 表 6.10 所示。

表 6.10: 保护区间编码

pmpaddr[45:9]	pmpcfg.A	保护区大小	备注
a_aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_aaa0	NAPOT	4KB	支持
a_aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_aa01	NAPOT	8KB	支持
a_aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_a011	NAPOT	16KB	支持
a_aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_0111	NAPOT	32KB	支持
a_aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_aaa0_1111	NAPOT	64KB	支持
a_aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_aa01_1111	NAPOT	128KB	支持
a_aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_a011_1111	NAPOT	256KB	支持
a_aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_0111_1111	NAPOT	512KB	支持
a_aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_aaa0_1111_1111	NAPOT	1M	支持
a_aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_aa01_1111_1111	NAPOT	2M	支持
a_aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_a011_1111_1111	NAPOT	4M	支持
a_aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_0111_1111_1111	NAPOT	8M	支持
a_aaaa_aaaa_aaaa_aaaa_aaaa_aaa0_1111_1111_1111	NAPOT	16M	支持
a_aaaa_aaaa_aaaa_aaaa_aaaa_aa01_1111_1111_1111	NAPOT	32M	支持
a_aaaa_aaaa_aaaa_aaaa_aaaa_a011_1111_1111_1111	NAPOT	64M	支持
a_aaaa_aaaa_aaaa_aaaa_aaaa_0111_1111_1111_1111	NAPOT	128M	支持
a_aaaa_aaaa_aaaa_aaaa_aaa0_1111_1111_1111_1111	NAPOT	256M	支持
a_aaaa_aaaa_aaaa_aaaa_aa01_1111_1111_1111_1111	NAPOT	512M	支持
a_aaaa_aaaa_aaaa_aaaa_a011_1111_1111_1111_1111	NAPOT	1G	支持
a_aaaa_aaaa_aaaa_aaaa_0111_1111_1111_1111_1111	NAPOT	2G	支持
a_aaaa_aaaa_aaaa_aaa0_1111_1111_1111_1111_1111	NAPOT	4G	支持
a_aaaa_aaaa_aaaa_aa01_1111_1111_1111_1111_1111	NAPOT	8G	支持
a_aaaa_aaaa_aaaa_a011_1111_1111_1111_1111_1111	NAPOT	16G	支持
a_aaaa_aaaa_aaaa_0111_1111_1111_1111_1111_1111	NAPOT	32G	支持
a_aaaa_aaaa_aaa0_1111_1111_1111_1111_1111_1111	NAPOT	64G	支持
a_aaaa_aaaa_aa01_1111_1111_1111_1111_1111_1111	NAPOT	128G	支持
a_aaaa_aaaa_a011_1111_1111_1111_1111_1111_1111	NAPOT	256G	支持
a_aaaa_aaaa_0111_1111_1111_1111_1111_1111_1111	NAPOT	512G	支持
a_aaaa_aaa0_1111_1111_1111_1111_1111_1111_1111	NAPOT	1T	支持
a_aaaa_aa01_1111_1111_1111_1111_1111_1111_1111	NAPOT	2T	支持
a_aaaa_a011_1111_1111_1111_1111_1111_1111_1111	NAPOT	4T	支持

续下页

表 6.10 - 接上页

pmpaddr[45:9]	pmpcfg.A	保护区大小	备注
a_aaaa_0111_1111_1111_1111_1111_1111_1111_1111	NAPOT	8T	支持
a_aaa0_1111_1111_1111_1111_1111_1111_1111_1111	NAPOT	16T	支持
a_aa01_1111_1111_1111_1111_1111_1111_1111_1111	NAPOT	32T	支持
a_a011_1111_1111_1111_1111_1111_1111_1111_1111	NAPOT	64T	支持
a_0111_1111_1111_1111_1111_1111_1111_1111_1111	NAPOT	128T	支持
0_1111_1111_1111_1111_1111_1111_1111_1111_1111	NAPOT	256T	支持
1_1111_1111_1111_1111_1111_1111_1111_1111_1111	Reserved	-	-

需要说明的是，C908X PMP NAPOT 模式支持的最小粒度为 4KB。不支持 NA4 模式。

6.4.2.2 物理内存保护地址寄存器（PMPADDR）

PMP 共实现了 8/16/32/64 个地址寄存器 pmpaddr0-pmpaddr7/15/31/63，存放表项的物理地址。

RISC-V 规定 PMP 地址寄存器存放的是物理地址的 [PA-1 ： 2] 比特，因为 C908X PMP 表项粒度最低支持 4KB，因此 bit[9:0] 不会用于地址鉴权逻辑。

	63 PA-2	PA-3 10	9 0
	0	address[PA-1 : 12] (WARL)	0 (WARL)
Reset	0	0	0

图 6.12: PMP 地址寄存器

6.5 内存访问顺序

在不同的场景下，C908X 对地址空间的访问过程简要归纳如下：

场景 1：不进行 VA-PA 转换

- CPU 要访问 PA；
- 通过 sysmap.h 得到该地址的属性；
- PMP 检查，确认读/写/执行权限符合 PMP 的设定；
- 执行对该地址的访问。

场景 2：进行 VA-PA 转换

- CPU 要访问 VA；
- 通过 MMU 进行地址翻译，得到页表项（pte）；
- 从 pte 可以得到以下信息：PA、地址属性（注 1）和读/写/执行权限；
- PMP 检查，确认读/写/执行权限符合 PMP 的设定；（最终的读/写/执行权限取 PMP 与 pte 的“最小值”）
- 执行对该地址的访问。

注：

当 mxstatus.maee=1 时，地址属性来自 pte；

当 mxstatus.maee=0 时，地址属性来自 pte.pbmt 和 sysmap.h。

7 内存子系统

7.1　内存子系统概述

C908X 每个核心有单独的指令 Cache 和数据 Cache，4 个核心共享一个 L2 Cache。多个核心之间的数据一致性由硬件维护。

7.2　L1 指令 Cache

7.2.1　概述

C908X L1 指令高速缓存的主要特征如下：

- 指令高速缓存大小硬件可配置，支持 16KB/32KB/64KB。
- 4 路组相联，缓存行大小为 64B。
- 虚拟地址索引，物理地址标记（VIPT）。
- 读访问数据位宽为 64 比特，写访问数据位宽为 128 比特。
- 采用伪随机替换策略。
- 支持对整个指令高速缓存的无效操作，支持对单条缓存行的无效操作。
- 支持指令预取功能。
- 支持路预测。
- 支持奇偶校验机制。
- 指令高速缓存缺失的请求会 snoop 数据高速缓存（存在开关位控制）。

7.2.2　指令预取

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

指令预取以 4KB 页表为边界，对于预取时发生跨 4KB 边界地址会主动停止预取，从而保证取指地址的安全。此外，读敏感的外设地址空间也禁止被分配到指令区空间。

7.2.3　路预测

C908X 指令高速缓存采用四路组相联结构，为了减少并行访问四路缓存的功耗，C908X 实现了指令高速缓存的路预测功能。在路预测信息有效时，关闭无效数据路的访问，仅访问预测路的数据。用户可以通过配置隐式操作寄存器 MHINT.IWPE 使能指令高速缓存的路预测功能。

根据取指行为的不同，路预测可分为以下两类：

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

7.2.4　分支历史表

C908X 采用分支历史表对条件分支的跳转方向进行预测。分支历史表容量为 64Kb，使用 TAGE Gshare 预测器作为预测机制，每周期支持一条分支结果预测。分支历史表由预测器和选择器两部分组成。其中预测器又分为跳转预测器和非跳转预测器，并根据分支历史信息对各预测器进行实时维护。分支历史表通过分支历史信息以及当前分支指令地址对各路进行索引，获得分支指令跳转方向的预测结果。

分支历史表进行预测的条件分支指令包括：

BEQ、BNE、BLT、BLTU、BGE、BGEU、C.BEQZ、C.BNEZ

7.2.5　分支跳转目标预测器

C908X 使用分支跳转目标预测器对分支指令的跳转目标地址进行预测。分支跳转目标预测器对分支指令历史目标地址进行记录。如果当前分支指令命中分支跳转目标预测器，则将记录目标地址作为当前分支指令预测目标地址。

分支跳转目标预测器主要特征包括：

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

分支跳转目标预测器进行预测的分支指令包括：

- BEQ、BNE、BLT、BLTU、BGE、BGEU、C.BEQZ、C.BNEZ
- JAL、C.J

7.2.6　间接分支预测器

C908X 使用间接分支预测器负责对间接分支的目标地址进行预测。间接分支指令通过寄存器获取目标地址，一条间接分支指令可包含多个分支目标地址，无法通过传统分支跳转目标预测器进行预测。因此，C908X 采用基于分支历史的间接分支预测机制，将间接分支指令的历史目标地址与该分支之前的分支历史

信息进行关联，用不同的分支历史信息将同一条间接分支的不同目标地址进行离散，从而实现多个不同目标地址的预测。

间接分支指令包括：

- JALR：源寄存器为 X1、X5 除外
- C.JALR：源寄存器为 X5 除外
- C.JR：源寄存器为 X1、X5 除外

7.2.7　返回地址预测器

返回地址预测器用于函数调用结束时，返回地址的快速准确预测。当取指单元译码得到有效的函数调用指令时，将函数返回地址压栈存入返回地址预测器；当取指单元译码得到有效的函数返回指令时，则从返回地址预测器弹栈，获取函数返回目标地址。返回地址预测器最多支持 12 层函数调用嵌套，超出嵌套次数会导致目标地址预测错误。

- 函数调用指令包括：JAL、JALR、C.JALR
- 函数返回指令包括：JALR、C.JR、C.JALR

  指令功能的具体划分可以如 表 7.1 。

表 7.1: 指令功能具体划分

rd	rs1	rs1=rd	RAS action
!link	!link	-	none
!link	link	-	pop
link	!link	-	push
link	link	0	push and pop
link	link	1	push

7.2.8　快速跳转目标预测器

为了加快连续跳转时取指单元的取指效率，C908X 在取指单元的第一级增加了快速跳转目标预测器。当取指单元发生连续跳转时，快速跳转目标预测器将会记录连续跳转的第二条跳转指令的地址和跳转的目标地址。若取指时命中了快速跳转目标预测器，则在第一级发起跳转，减少至少一个周期的性能损失。

快速跳转目标预测器进行预测的分支指令包括：

- BEQ、BNE、BLT、BLTU、BGE、BGEU、C.BEQZ、C.BNEZ
- JAL、C.J
- 函数返回指令

7.2.9　奇偶校验功能

指令缓存支持可配置的奇偶校验机制。该校验机制检查指令缓存的 tag array，校验粒度为 29bit；检查 data array，校验粒度为 34bit。开启校验机制后，指令缓存在写入时对数据进行奇偶编码，在数据读取时进行检查。当发生 1bit 数据错误时，能检测到错误，无效当前错误数据，重新向总线发起取指请求，回填缓存。同时上报错误信息，相关信息包括路信息，索引信息等，可在 MCER/SCER 寄存器上查询。具体控制寄存器说明可以参考附录 C-1 机器模式处理器控制和状态扩展寄存器中有关 MCER/SCER 的描述。

1bit 以上错误无法检测或纠正。

C908X 指令高速缓存支持软件注入错误功能，具体控制寄存器说明可以参考附录 C-1 机器模式处理器控制和状态扩展寄存器中有关 MEICR 的描述。

7.3　L1 数据 Cache

7.3.1　概述

L1 数据高速缓存的主要特征如下：

- 数据高速缓存大小硬件可配置，支持 16KB/32KB/64KB；
- 4 路组相联，缓存行大小为 64B；
- 虚拟地址索引，物理地址标记（VIPT）；
- 硬件自动消除别名（alias）的问题；
- 每次读访问的最大宽度为 128 比特，支持字节/半字/字/双字/四字访问；
- 每次写访问的最大宽度为 128 比特，支持任意字节组合的访问；
- 写策略支持写回-写分配模式和写回-写不分配模式；
- 采用先进先出的替换策略；
- 支持对整个数据高速缓存的无效和清除操作，支持对单条缓存行的无效和清除操作；
- 指令多通道的数据预取功能。
- 支持 ECC 和奇偶校验机制；

7.3.2　Cache 一致性

对于页面属性配置为 shareable 且 cacheable 的请求，硬件维护数据在不同核心的 L1 数据高速缓存的一致性。

对于页面属性配置为 non-shareable 且 cacheable 的请求，处理器不维护数据在多个 L1 数据高速缓存上的一致性。如果需要该属性的页面在多个核心上共享，则需要软件维护数据一致性。

C908X 一级高速缓存采用 MESI 协议维护多个处理器核心数据高速缓存的一致性。MESI 代表了每个缓存行在数据高速缓存上的 4 个状态，分别是：

- M：表示缓存行仅位于此数据高速缓存中，且被写脏；（UniqueDirty）

- E：表示缓存行仅位于此数据高速缓存中，且是干净的；（UniqueClean）
- S：表示缓存行可能位于多个数据高速缓存中，且是干净的；（ShareClean）
- I：表示缓存行不在该数据高速缓存中。（Invalid）

7.3.3　独占式访问

C908X 支持独占式的内存访问指令 LR 和 SC。用户可以使用这两条指令构成原子锁等同步原语实现同一个核不同进程之间或者不同核之间的同步。通过 LR 指令标记需要独占访问的地址，SC 指令判断被标记的地址是否被其他进程抢占。C908X 为每个核心上设置了一个位于 L1 数据高缓的局部监测器，监测器由一个状态机和一个地址缓存器组成。其中，状态机包含两个状态：IDLE 和 EXCLUSIVE 。

对于属性设置为可高缓的页面，通过局部监测器就能够实现独占式访问。LR 指令在执行过程中设置局部监测器的状态机为 EXCLUSIVE 态并将访问的地址和 Size 保存到缓存器中；SC 指令在执行过程中读取局部监测器的状态、地址和 Size，如果状态为 EXCLUSIVE 并且 SC 地址和 Size 属于局部监测器记录的地址范围子集，那么执行该写操作，返回写成功，并清除状态机回到 IDLE 态；否则如果状态或者地址/Size 有一项不满足条件，不执行该写操作，返回写失败，并清除状态机回到 IDLE 态。其他核的写操作在相同 cacheline 地址匹配局部监测器时，也会将状态机清回到 IDLE 态；本核的写操作不影响局部监测器。此外，在进程切换时需要清除局部监测器。

在基于 C908X 的系统中，推荐使用 LR 和 SC 指令实现原子锁操作。如果原子锁的地址属性是 cacheable（含 shared 和 non-shared），则不需要 SoC 系统做特别的设计。这是典型情况。如果原子锁的地址属性是 Non-cacheable/Device/Strongly Ordered，则需要用户在系统里（e.g. Slave 端）集成 exclusive monitor 功能。使用其他方式，操作结果为 UNPREDICTABLE。

7.4　L2 Cache

7.4.1　L2 Cache 概要

L2 高速缓存的主要特征如下：

- 高速缓存大小硬件可配置，支持 No/64KB/128KB/256KB/512KB/1MB/1.5MB/2MB/3MB/4MB；
- 16 路组相联，缓存行大小为 64B；在 1.5M，3M 配置下采用 12 路组相联；
- L2 Cache 与 L1 Cache 之间采用动态包含性策略；
- 物理地址索引，物理地址标记（PIPT）；
- 每次访问的最大宽度为 64B；
- 默认支持 Write Back 策略；
- 采用 RRIP（Re-Reference Interval Prediction）的替换策略；
- 可编程的 RAM 访问延时；
- 可选的 ECC 校验机制；
- 支持指令预取和 TLB 预取机制；
- 支持分块或不分块的流水线技术；

7.4.2　Cache 一致性

C908X 二级高速缓存采用 MESI 协议来维护多核之间的数据一致性，MESI 分别代表 Cacheline 的四种状态，分别为：

- M（Unique Dirty）：表示该缓存行仅存在于本 L2 Cache 系统中，并且是脏的
- E（Unique Clean）：表示该缓存行进存在于本 L2 Cache 系统中，并且是干净的
- S（Shared Clean）：表示该缓存中可能存在于多个 L2 Cache 系统中，并且是干净的
- I（Invalid）：表示该缓存行不存在于本 L2 Cache 系统中

备注

由于 C908X 采用动态包含性策略，上述内容中“本 L2 Cache 系统”包含了 L2 Cache 与系统中所有核内部 L1 Cache。

7.4.3　动态包含性策略

L2 Cache 采用动态包含性策略，其基本原则为：

- 若某 Cacheline 为单核独占，则采用伪 Exclusive 策略，L2 不保留备份
- 若某 Cacheline 为多核共享，则采用伪 Inclusive 策略，L2 保留备份

以下场景为动态包含性策略的示例：

- 核 0 请求回填某 Cacheline，该 Cacheline 不在系统中存在备份或仅在 L2 中存在备份，则数据回填核 0 后，L2 中不会存在该 Cacheline，此时为伪 Exclusive
- 核 0 请求踢出某 Cacheline，该 Cacheline 不存在于系统中，则会 Allocate 到 L2，此时为伪 Exclusive
- 核 0 请求回填某 Cacheline，该 Cacheline 存在于其他核中，且不会被本次请求 Invalid，则 L2 会保存该 Cacheline 备份，此时为伪 Inclusive

7.4.4　组织形式

C908X L2 高速缓存在组织形式上可支持分块的流水线架构，将访问地址离散在两个不同的块中，允许多个访问的并行处理，从而提高访问效率。

分块机制如 图 7.1 所示。

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

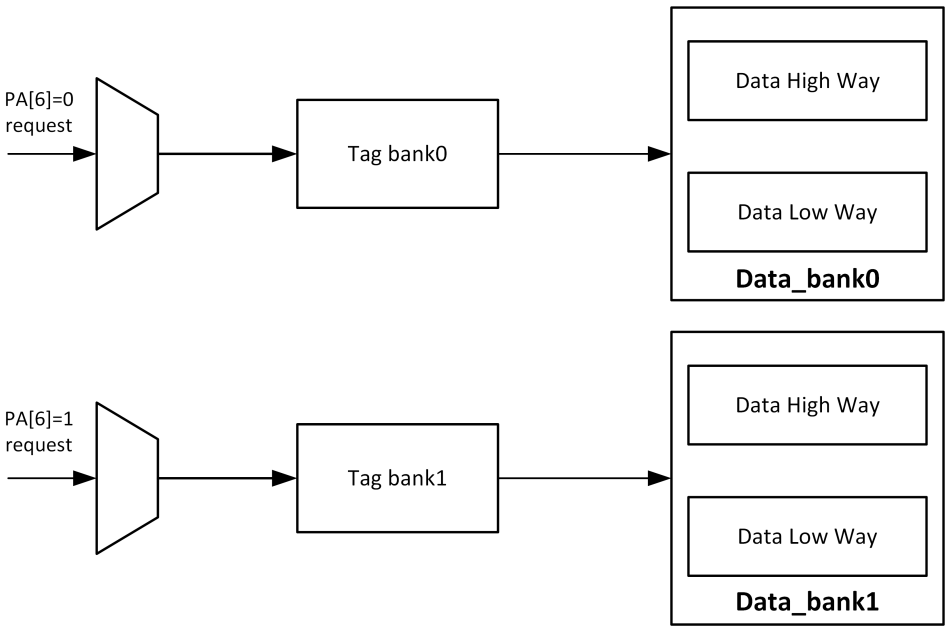


图 7.1: L2 Cache 组织形式

7.4.5　RAM 延时

由于 L2 Cache 较大，所以访问延迟较长，通常需要多个时钟周期才能完成访问。C908X 提供了可配的访问延迟，在不同工艺下能够根据所用 RAM 的 setup time 和 latency 进行手动设置。配置详情如 表 7.2 、表 7.3 所示。

TAG RAM 详细的配置选项与访问延时的关系如 表 7.2 所示。

表 7.2: TAG RAM 详细的配置选项与访问延时的关系

配置延时选项	是否带有 SETUP	延时（CYCLE）
1 cycle	×	1
2 cycle	√	3
3 cycle	√	4
4 cycle	√	5
5 cycle	√	6

DATA RAM 详细的配置选项与访问延时的关系如 表 7.3 所示：

- L2 TAG RAM 最大有效延时周期为 6 cycle
- L2 DATA RAM 最大有效延时周期为 9 cycle
- 带有 SETUP 时增加 1 周期的访问延时

表 7.3: DATA RAM 延迟

配置延时选项	是否带有 SETUP	配置 SETUP 延时（CYCLE）
1 cycle	×	-
2 cycle	√	3
3 cycle	√	4
4 cycle	√	5
5 cycle	√	6
6 cycle	√	7
7 cycle	√	8
8 cycle	√	9

7.5　内存加速访问

本小节集中描述 C908X L1/L2 cache 在内存加速访问方面的特性。

7.5.1　L1 I-Cache 指令预取

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

指令预取要求预取的缓存行与当前访问的缓存行位于同一个页面，从而保证取指地址的安全。此外，读敏感的外设地址空间也禁止被分配到指令区空间。

7.5.2　L1 D-Cache 多通道数据预取

为了减少 DDR 等大内存的存储访问延时，C908X 支持数据预取功能。通过检测数据高速缓存的缺失，匹配出固定的访问模式，然后硬件自动预取缓存行并回填 L1 数据高速缓存。

C908X 最多支持 8 个通道的数据预取，并能够支持规律可变间隔的预取（stride<=64 个缓存行）。

此外，还实现了正向预取和反向预取（即 stride 为负数），从而支持各种可能的访问模式。

在处理器执行数据高速缓存无效和清除操作时，停止数据预取功能。

用户可通过设置隐式寄存器 MHINT.DPLD，使能数据预取功能；并通过设置 MHINT.DPLD_DIS，决定一次预取的缓存行数量。

支持数据预取的指令如下：

- LB、LBU、LH、LHU、LW、LWU、LD
- FLW、FLD
- LRB、LRH、LRW、LRD、LRBU、LRHU、LRWU、LURB、LURH、LURW、LURD、LURBU、LURHU、LURWU、LBI、LHI、LWI、LDI、LBUI、LHUI、LWUI、LDD、LWD、LWUD

7.5.3　L1 自适应的写分配机制

C908X 的 L1 实现了自适应的写分配机制。当处理器检测到连续的内存写入操作时，页面的写分配属性会自动关闭。

用户可通过设置隐式寄存器 MHINT.AMR 来开启 L1 自适应写分配。

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

支持自适应写分配的指令如下：

- SB、SH、SW、SD
- FSW、FSD
- SRB、SRH、SRW、SRD、SURB、SURH、SURW、SURD、SBI、SHI、SWI、SDI、SDD、SWD

7.5.4　L2 预取机制

L2 高速缓存具有预取功能，支持取指令与 TLB 访问的预取工作。支持的特性如下：

- 软件可配的指令预取数量（0，2，4，8，16，32），所有预取都会回填 L2 高速缓存；
- TLB 预取数固定为 1；
- 预取机制以 4KB 页表为边界，对于预取时发生跨 4KB 边界地址会主动停止预取；
- 可以通过 MCCR2 对预取机制进行配置。

7.6　L1/L2 Cache 操作相关的指令和寄存器

在处理器复位后，指令和数据高速缓存会自动进行无效化操作，且默认关闭指令和数据高速缓存。

类似地，在处理器复位后，L2 高速缓存会自动进行无效化操作，完成操作后 L2 将自动开启，且不可关闭。值得注意的是，当 L1 高速缓存关闭时，L2 在缺失时不会发起回填操作。

7.6.1　L1 高速缓存扩展寄存器

C908X L1 高速缓存相关扩展寄存器，按功能主要分为：

- 高速缓存使能和模式配置：机器模式硬件配置寄存器 (mhcr) 可以实现对指令和数据高速缓存的开关以及写分配和写回模式的配置。超级用户模式硬件配置寄存器（shcr）是 mhcr 的映射，为只读寄存器。
- 脏表项清除和无效化操作：机器模式高速缓存操作寄存器（mcor）可以对指令和数据高速缓存进行脏表项和无效化操作。
- 高速缓存读操作：机器模式高速缓存访问指令寄存器（mcins）、高速缓存访问索引寄存器（mcindex）和高速缓存访问数据寄存器 0/1（mcdata0/1），通过这三个寄存器可以实现对指令和数据高速缓存的读操作。

具体控制寄存器说明可以参考*机器模式处理器控制和状态扩展寄存器组*和*机器模式 Cache 访问扩展寄存器组*。

7.6.2　L2 高速缓存扩展寄存器

C908X L2 高速缓存相关扩展寄存器，按功能主要分为：

- L2 高速缓存使能和延时配置：机器模式 L2 高速缓存使能寄存器 (mccr2) 可以实现对 L2 高速缓存访问延时设置。
- L2 高速缓存读操作：机器模式高速缓存访问指令寄存器（mcins）、高速缓存访问索引寄存器（mcindex）和高速缓存访问数据寄存器 0/1（mcdata0/1），通过这三个寄存器可以实现对 L2 高速缓存的读操作。

具体相关控制寄存器的定义和说明可以参考*机器模式处理器控制和状态扩展寄存器组*和*机器模式 Cache 访问扩展寄存器组*。

7.6.3　L1 Cache 操作指令

C908X 扩展了 L1 高速缓存操作指令，包括按地址进行无效化、无效化全部、按地址清脏表项、清全部脏表项、按地址清脏表项并无效化和清全部脏表项并无效化，具体如 表 7.4 所示。

表 7.4: L1 Cache 操作指令

指令名称	指令描述
ICACHE.IALL	ICACHE 无效所有表项
ICACHE.IALLS	ICACHE 广播无效所有表项
ICACHE.IPA	ICACHE 按物理地址无效表项
ICACHE.IVA	ICACHE 按虚拟地址无效表项
DCACHE.CALL	DCACHE 清全部脏表项
DCACHE.CIALL	DCACHE 清全部脏表项并无效
DCACHE.CIPA	DCACHE 按物理地址清脏表项并无效
DCACHE.CISW	DCACHE 按 set/way 清脏表项并无效
DCACHE.CIVA	DCACHE 按虚拟地址清脏表项并无效
DCACHE.CPA	DCACHE 按物理地址清脏表项
DCACHE.CPAL1	L1 DCACHE 按物理地址清脏表项
DCACHE.CVA	DCACHE 按虚拟地址清脏表项
DCACHE.CSW	DCACHE 按 set/way 清脏表项
DCACHE.CVAL1	L1 DCACHE 按虚拟地址清脏表项
DCACHE.IPA	DCACHE 按物理地址无效
DCACHE.ISW	DCACHE 按 set/way 无效

续下页

表 7.4 - 接上页

指令名称	指令描述
DCACHE.IVA	DCACHE 按虚拟地址无效
DCACHE.IALL	DCACHE 无效所有表项

指令的具体说明请参考*附录 B-1 Cache 指令术语*

7.7　L1/L2 Cache 的保护机制

C908X 实现的 Cache 保护机制包括：L1 I-Cache ECC 校验、jTLB 奇偶校验、L1 D-Cache ECC 校验和 L2 Cache ECC 校验。各种机制的检测/纠错能力及中断上报，如 表 7.5 所示。

表 7.5: ECC/奇偶校验检测纠错能力以及中断上报

缓存类型	1bit 错误	2bit 错误	2bit 以上错误
L1 高速指令缓存	可检测 不发出异常或中断	可检测 不发出异常或中断	无法检测 不发出异常或中断
L1 高速数据缓存	可检测并纠正 不发出异常或中断	可检测 发出中断请求	无法检测 不发出异常或中断
jTLB	可检测 不发出异常或中断	无法检测 不发出异常或中断	无法检测 不发出异常或中断
L2 缓存	可检测并纠正 不发出异常或中断	可检测 发出中断请求	无法检测 不发出异常或中断

7.7.1　L1 I-Cache 奇偶校验

L1 指令缓存支持可配置的奇偶校验机制。该校验机制检查指令缓存的 tag array，校验粒度为 28bit；检查 data array，校验粒度为 32bit。

L1 CACHE 奇偶校验/ECC 功能由机器模式隐式操作寄存器（MHINT）中的 bit 19 使能。开启后，指令缓存在写入时对数据进行奇偶编码，在数据读取时进行检查。当发生 1bit 数据错误时，能检测到错误，无效当前错误数据，重新向总线发起取指请求，回填缓存。同时记录错误信息，包括路信息，索引信息等，可在 MCER/SCER 寄存器中查询。仅可在机器模式通过写机器模式 L1 Cache ECC 寄存器（MCER）的方式清除。具体控制寄存器说明可以参考*机器模式处理器控制和状态扩展寄存器组*中有关 MCER/SCER 的描述。1bit 以上错误无法检测或纠正。

C908X L1 指令高速缓存支持软件注入错误功能，具体控制寄存器说明可以参考*机器模式处理器控制和状态扩展寄存器组*中有关 MEICR 的描述。

此外，还实现了对 jTLB 的奇偶校验，能够检测 1bit 错误。

7.7.2　L1 D-Cache ECC 校验

L1 数据高速缓存支持可配置的 ECC 校验机制，具体校验信息如 表 7.6 所示。

表 7.6: L1 Data Cache 校验

RAM	校验粒度	校验比特	校验方式
TAG	29	7	ECC
DATA	32	7	ECC

L1 CACHE 奇偶校验/ECC 功能由机器模式隐式操作寄存器（MHINT）中的 bit 19 使能。开启后，L1 数据高速缓存在写入时进行 ECC 编码，在读取时进行校验。当检测到 1bit ECC 错误时，能够自动纠正错误，返回正确数据；当发生 2bit 错误时，能够检测出错误，发起校验错误中断，并无效掉 L1 D-cache 中出错的缓存行。2bit 以上的错误不能准确检测或纠正。

软件可以查询 MCER/SCER 寄存器获取错误的相关信息，例如是否产生 2bit 错误以及错误在数据高速缓存发生的位置信息。仅可在机器模式通过写机器模式 L1 Cache ECC 寄存器（MCER）的方式清除。具体控制寄存器说明可以参考*机器模式处理器控制和状态扩展寄存器组* 中有关 MCER/SCER 的描述。

L1 数据缓存发生 2bit 错误引发的中断通过直接使能 MCIP 位触发，核内中断向量号为 16，具体控制寄存器描述可以参考*机器模式处理器控制和状态扩展寄存器组* 中有关 MIP 的描述。

C908X L1 数据缓存支持软件注入错误功能，具体控制寄存器说明可以参考*机器模式处理器控制和状态扩展寄存器组* 中有关 MEICR 的描述。

7.7.3　L2 ECC 校验

L2 Cache 支持可配置的 ECC 校验功能，可以对 TAG RAM、DATA RAM、SnoopFilter RAM 进行校验。当开启 ECC 校验时，会在向 SRAM 写入数据时对数据进行 ECC 编码，在读取 SRAM 数据时进行 ECC 校验。

当检测到 1 bit ECC 错误时，能够硬件纠正错误并返回正确的数据。

当发生 2 bit 错误时，能够检测出数据错误但无法硬件纠错，会上报 ECC 校验中断，返回错误的数据并无效掉出错的缓存行。

各 RAM 的校验粒度与校验比特位宽如 表 7.7 所示。

表 7.7: L2 ECC 校验粒度

RAM 类型	校验粒度 (bit)	校验比特 (bit)
TAG	23 ~ 35	7
DIRTY	4	4
DATA	64	8
SF TAG	23 ~ 35	7
SF INFO	9	5

8 矢量计算

8.1　版本支持

C908X 兼容 RISC-V “V” Vector Extension, Version 1.0-rc1-20210608 。

8.2　矢量编程模型

C908X 的矢量扩展支持以下特性：

- 32 个独立的矢量寄存器 v0-v31。矢量寄存器的位宽为 512/1024/4096 位，取决于矢量算力选项。
- 对于矢量浮点指令，支持的元素类型为 FP16、FP32 （即 SEW=16/32 ）。
- 对于矢量整型指令，支持的元素类型为 INT8、INT16、INT32 和 INT64（即 SEW=8/16/32/64）。
- 支持矢量寄存器的分组，以提高矢量运算的效率。支持 4 种分组方式：每组包含 1/2/4/8 个矢量寄存器，分成 32/16/8/4 个组。

需要注意的是，矢量访存指令目的地址的 SO 属性不能为 1。

8.3　矢量控制寄存器

8.3.1　RISC-V 矢量控制寄存器

RISC-V V 扩展增加了 7 个非特权 CSR，如下：

- vstart

  矢量起始位置寄存器指定了执行矢量指令时起始元素位置。每条矢量指令执行后 vstart 会被清零。在大多数情况下，软件不需要改动 vstart。在 C908X 中，只有矢量存储指令支持非 0 的 vstart；所有的运算类矢量指令都要求 vstart=0，否则会产生非法指令异常。

- vxsat

  定点溢出标志位寄存器，只有 bit 0 有效，表示是否有定点指令产生溢出结果。

- vxrm

  定点舍入模式寄存器，支持 4 种舍入模式：向大数舍入、向偶数舍入、向零舍入和向奇数舍入。

- vcsr

  矢量控制和状态寄存器。

- vl

  矢量长度寄存器指定了矢量指令更新目的寄存器的元素范围。矢量指令更新目的寄存器中元素序号小于 vl 的元素，清零目的寄存器中元素序号大于等于 vl 的元素。特别的，当 vstart>=vl 或 vl=0 时，目的寄存器的所有元素不被更新。

- vtype

  矢量数据类型寄存器，设定矢量计算的基本数据属性，包括：非法标记位、元素位宽设定、矢量寄存器分组设定。vtype 也包含 EDIV 使能位，C908X 不支持 EDIV，所以 EDIV 位为 0。

- vlenb

  矢量位宽寄存器, 以字节为单位表示 C908X 的矢量位宽。

C908X 支持矢量状态维护功能，在 mstatus[10:9] 处定义了 VS 位，可以用于判断上下文切换的时候，是否需要保存矢量相关的寄存器。

8.3.2　C908X VPU 矢量控制寄存器 (mtnfastmba)

为支持 fast memory 访问，C908X VPU 额外增加寄存器 MTNFASTMBA 用于标记 FAST MEMORY 的地址范围，寄存器在配置宏 *XX_FAST_MEM* 时有效。

表 8.1: fast memory 地址范围扩展寄存器 (mtnfastmba) 描述

bit	名称	读写权限	描述
63	mtnfasten	机器模式读写	fast memory 使能位，为 1 时表示有 fast memory
62	mpage2men	机器模式读写	2M 及以上页表优化使能位，为 1 时表示优化开启
61:PAB*	Reserved	机器模式只读	保留位
PAB-1:0	mtnfastmba	机器模式读写	fast memory 空间 base 地址（PA）

*：PA40 时，PAB = 28；PA48 时，PAB = 37。

注意

fast memory 不支持存放页表

矢量、标量访存指令，若基地址位于配置的范围之间，则通过 fast memory 访问外部 SRAM，具体说明如下：

- 配置 mtnfastmba，根据 PA40/PA48 对高位 mask 为 0
- 配置的数据中，低位有 N 个连续的 1，则表示 2^N 个 4 KB 的 size

如下 表 8.2 所示，以 RV64 系统下，PA48 mtnfastmba 地址范围为例：

表 8.2: RV64 系统下 PA48 mtnfastmba 地址范围

Size	mtnfastmba X 表示十六进制数，x 表示二进制数	Range
4KB	64'b1xxxXXX_XXXX_XXXX _xxxx_xxxx_xxxx_xxx0	{mtnfastmba[36:1]，12{1'b0}} ~ {mtnfastmba[36:1]，12{1'b1}}
8KB	64'b1xxxXXX_XXXX_XXXX _xxxx_xxxx_xxxx_xx01	{mtnfastmba[36:2]，13{1'b0}} ~ {mtnfastmba[36:2]，13{1'b1}}
16KB	64'b1xxxXXX_XXXX_XXXX _xxxx_xxxx_xxxx_x011	{mtnfastmba[36:3]，14{1'b0}} ~ {mtnfastmba[36:3]，14{1'b1}}
32KB	64'b1xxxXXX_XXXX_XXXX _xxxx_xxxx_xxxx_0111	{mtnfastmba[36:4]，15{1'b0}} ~ {mtnfastmba[36:4]，15{1'b1}}
64KB	64'b1xxxXXX_XXXX_XXXX _xxxx_xxxx_xxx0_1111	{mtnfastmba[36:5]，16{1'b0}} ~ {mtnfastmba[36:5]，16{1'b1}}
128KB	64'b1xxxXXX_XXXX_XXXX _xxxx_xxxx_xx01_1111	{mtnfastmba[36:6]，17{1'b0}} ~ {mtnfastmba[36:6]，17{1'b1}}
256KB	64'b1xxxXXX_XXXX_XXXX _xxxx_xxxx_x011_1111	{mtnfastmba[36:7]，18{1'b0}} ~ {mtnfastmba[36:7]，18{1'b1}}
512KB	64'b1xxxXXX_XXXX_XXXX _xxxx_xxxx_0111_1111	{mtnfastmba[36:8]，19{1'b0}} ~ {mtnfastmba[36:8]，19{1'b1}}
1MB	64'b1xxxXXX_XXXX_XXXX _xxxx_xxx0_1111_1111	{mtnfastmba[36:9]，20{1'b0}} ~ {mtnfastmba[36:9]，20{1'b1}}
2MB	64'b1xxxXXX_XXXX_XXXX _xxxx_xx01_1111_1111	{mtnfastmba[36:10]，21{1'b0}} ~ {mtnfastmba[36:10]，21{1'b1}}
4MB	64'b1xxxXXX_XXXX_XXXX _xxxx_x011_1111_1111	{mtnfastmba[36:11]，22{1'b0}} ~ {mtnfastmba[36:11]，22{1'b1}}
8MB	64'b1xxxXXX_XXXX_XXXX _xxxx_0111_1111_1111	{mtnfastmba[36:12]，23{1'b0}} ~ {mtnfastmba[36:12]，23{1'b1}}
16MB	64'b1xxxXXX_XXXX_XXXxxxx _xxx0_1111_1111_1111	{mtnfastmba[36:13]，24{1'b0}} ~ {mtnfastmba[36:13]，24{1'b1}}
32MB	64'b1xxxXXX_XXXX_XXXxxxx _xx01_1111_1111_1111	{mtnfastmba[36:14]，25{1'b0}} ~ {mtnfastmba[36:14]，25{1'b1}}
64MB	64'b1xxxXXX_XXXX_XXXxxxx _x011_1111_1111_1111	{mtnfastmba[36:15]，26{1'b0}} ~ {mtnfastmba[36:15]，26{1'b1}}
128MB	64'b1xxxXXX_XXXX_XXXxxxx _0111_1111_1111_1111	{mtnfastmba[36:16]，27{1'b0}} ~ {mtnfastmba[36:16]，27{1'b1}}
256MB	64'b1xxxXXX_XXXX_XXxxxx _xxx01111_1111_1111_1111	{mtnfastmba[36:17]，28{1'b0}} ~ {mtnfastmba[36:17]，28{1'b1}}

续下页

表 8.2 - 接上页

Size	mtnfastmba X 表示十六进制数，x 表示二进制数	Range
512MB	64'b1xxxXXX_XXXX_XXxxxx _xx011111_1111_1111_1111	{mtnfastmba[36:18]，29{1'b0}} ~ {mtnfastmba[36:18]，29{1'b1}}
1GB	64'b1xxxXXX_XXXX_XXxxxx _x0111111_1111_1111_1111	{mtnfastmba[36:19]，30{1'b0}} ~ {mtnfastmba[36:19]，30{1'b1}}
2GB	64'b1xxxXXX_XXXX_XXxxxx _01111111_1111_1111_1111	{mtnfastmba[36:20]，31{1'b0}} ~ {mtnfastmba[36:20]，31{1'b1}}
4GB	64'b1xxxXXX_XXXX_XXxxx0 _11111111_1111_1111_1111	{mtnfastmba[36:21]，32{1'b0}} ~ {mtnfastmba[36:21]，32{1'b1}}
8GB	64'b1xxxXXX_XXXX_XXxx01 _11111111_1111_1111_1111	{mtnfastmba[36:22]，33{1'b0}} ~ {mtnfastmba[36:22]，33{1'b1}}
16GB	64'b1xxxXXX_XXXX_XXx011 _11111111_1111_1111_1111	{mtnfastmba[36:23]，34{1'b0} ~ {mtnfastmba[36:23]，34{1'b1}}
32GB	64'b1xxxXXX_XXXX_XX0111 _11111111_1111_1111_1111	{mtnfastmba[36:24]，35{1'b0} ~ {mtnfastmba[36:24]，35{1'b1}}
64GB	64'b1xxxXXX_XXXX_Xxxx0 _1111_11111111_1111_1111_1111	{mtnfastmba[36:25]，36{1'b0} ~ {mtnfastmba[36:25]，36{1'b1}}
128GB	64'b1xxxXXX_XXXX_Xxx01 _1111_11111111_1111_1111_1111	{mtnfastmba[36:26]，37{1'b0} ~ {mtnfastmba[36:26]，37{1'b1}}
256GB	64'b1xxxXXX_XXXX_Xx011 _1111_11111111_1111_1111_1111	{mtnfastmba[36:27]，38{1'b0} ~ {mtnfastmba[36:27]，38{1'b1}}
512GB	64'b1xxxXXX_XXXX_X0111 _1111_11111111_1111_1111_1111	{mtnfastmba[36:28]，39{1'b0} ~ {mtnfastmba[36:28]，39{1'b1}}
1TB	64'b1xxxXXX_XXXX_xxx0 _1111_1111_11111111_1111_1111_1111	{mtnfastmba[36:29]，40{1'b0} ~ {mtnfastmba[36:29]，40{1'b1}}
2TB	64'b1xxxXXX_XXXX_xx01 _1111_1111_11111111_1111_1111_1111	{mtnfastmba[36:30]，41{1'b0} ~ {mtnfastmba[36:30]，41{1'b1}}
4TB	64'b1xxxXXX_XXXX_x011 _1111_1111_11111111_1111_1111_1111	{mtnfastmba[36:31]，42{1'b0} ~ {mtnfastmba[36:31]，42{1'b1}}
8TB	64'b1xxxXXX_XXXX_0111 _1111_1111_11111111_1111_1111_1111	{mtnfastmba[36:32]，43{1'b0} ~ {mtnfastmba[36:32]，43{1'b1}}
16TB	64'b1xxxXXX_XXXxxx0_1111 _1111_1111_11111111_1111_1111_1111	{mtnfastmba[36:33]，44{1'b0} ~ {mtnfastmba[36:33]，44{1'b1}}
32TB	64'b1xxxXXX_XXXxx01_1111 _1111_1111_11111111_1111_1111_1111	{mtnfastmba[36:34]，45{1'b0} ~ {mtnfastmba[36:34]，45{1'b1}}

续下页

表 8.2 - 接上页

Size	mtnfastmba X 表示十六进制数，x 表示二进制数	Range
64TB	64'b1xxxXXX_XXXx011_1111 _1111_1111_11111111_1111_1111_1111	{mtnfastmba[36:35]，46{1'b0} ~ {mtnfastmba[36:35]，46{1'b1}}
128TB	64'b1xxxXXX_XXX0111_1111 _1111_1111_11111111_1111_1111_1111	{mtnfastmba[36]，47{1'b0} ~ {mtnfastmba[36]，47{1'b1}}
256TB	64'b1xxxXXX_XXX1111_1111 _1111_1111_11111111_1111_1111_1111	{48'b0} ~ {48{1'b1}}

同时需要满足以下要求：

1. 要求外部 SRAM 地址空间为 Non-Cacheable 属性且不存在单条指令既访问 SRAM 也访问非 SRAM 空间的情况。

   1）如果出现单条指令即访问了 fast memory 空间又访问了非 fast memory 空间，报同步异常

   2）只要地址区间符合要求，对地址属性信息将强制为 Weak-Order、Non-Cacheable 属性

2. 如果之前这段区间未配置为 fast_memory 区间，软件配置 fast_memory 区间后，之前对该区域 store 的数据将不能被系统可见，直到该区域空间又配置为非 fast_memory 地址空间。

3. 对于 Cacheable 请求，如果矢量访存指令不是通过 fast memory 接口访问外部 SRAM，而是通过普通接口去访问 L2 Cache 或外部 SRAM，则需要使用 sync.s 指令来保证矢量 load/store 操作的执行顺序。

8.4　矢量相关异常

矢量指令可以分为三大类：

- 矢量 load；
- 矢量运算；
- 矢量 store。

矢量运算不会触发异常。矢量 store 由于总线忽略 BRESP 的出错信息，也不会触发异常。所以，只有矢量 load 可能触发异常。对于矢量 load 触发的异常，CPU 丢弃已经读取的数据，重置 vstart 为 0，并且 mepc 指向这条指令（例外：非精确异常时，mepc 可能指向后续指令）。

矢量指令执行期间发生中断，行为与一般指令相同，即：CPU 完成当前指令，mepc 指向下一条指令。其余的步骤符合中断处理的一般流程。

9 安全设计

9.1 安全需求

本章的主要目标是提供软硬件的安全概要设计以满足 TEE(Trusted Execution Environment) 的系统安全需求。系统安全需求主要包含如下几个方面：

- 支持相互独立的可执行域 (Zone)
- 支持 Zone 间相互隔离，隔离维度包括代码执行、内存访问、外设、I/O 资源等
- 支持每个 Zone 内的应用程序和应用程序之间、应用程序和内核之间的相互隔离
- 支持多核 SMP 架构
- 支持 Zone 之间共享内存访问
- 支持 RISC-V 32-bit 和 64-bit 架构
- 支持 Zone 之间的可信通信
- 支持提供兼容 GP 规范的 TEE 执行环境

9.2 处理器安全模型

如 图 9.1 ，RISC-V ISA 架构支持 M-mode、S-mode 和 U-mode 三种特权模式，这三种模式在执行和访问权限上都进行了分离：

- 普通用户模式 (U-mode)，只能执行非特权指令，通常在该特权模式上运行用户应用程序。
- 超级用户模式 (S-mode)，能额外执行超级用户特权的指令，拥有 MMU 管理权限，通常在该特权模式上运行复杂的操作系统，如 Linux。
- 机器模式 (M-mode)，拥有最高的执行和访问权限，具有包括中断/异常响应和管理、物理内存保护以及特权访问控制等管理权限。

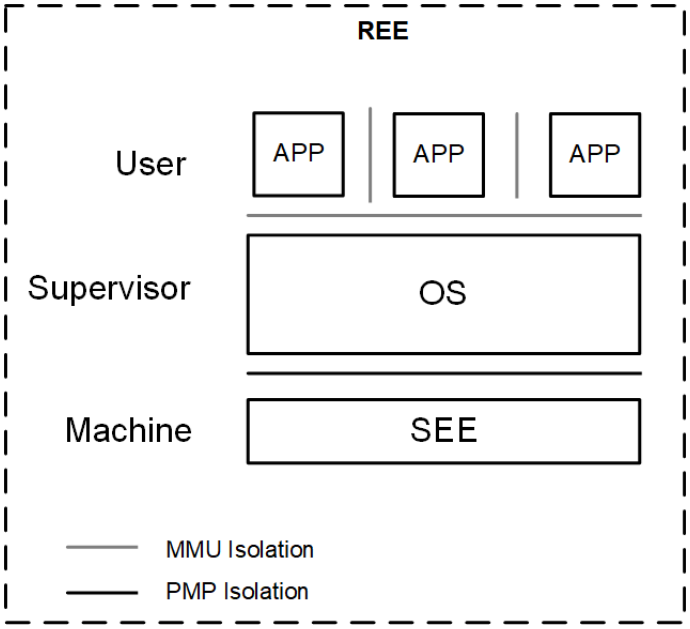


图 9.1: RISC-V 特权模型

RISC-V 的 S-mode 和 U-mode 和其他主流处理器架构并无太大差异，如 ARM 上的 Supervisor 模式和 User 模式。U-mode 只能运行非特权指令，运行在 U-mode 的应用程序只能通过系统调用自陷到超级用户模式后，在操作系统的管理下去访问系统资源。S-mode 除了能运行所有非特权指令外，还能运行 S-mode 下的特权指令和访问 S-mode 下的 CSR 系统寄存器。另外，超级用户模式具有访问内存管理单元 (MMU) 的权限，用户态和内核态的内存保护和隔离主要通过虚拟内存管理来实现的。M-mode 特权模式拥有最高的执行和访问权限。RISC-V 架构为 M 模式新增了只有该特权模式才能执行的特权指令和该模式下才能访问的系统寄存器，如物理内存保护 (PMP) 等。另外，机器模式最重要的特性是拦截和处理异常的能力。异常产生时，处理器默认将所有异常自陷到 M-mode，M-mode 异常处理程序再将中断“转发”到 S-mode。M-mode 通常运行可信固件 (Trusted Firmware), 用于对软硬件资源进行协调、分配和管理。

为了满足 TEE(Trusted Execution Environment) 的隔离要求，玄铁 C 系列处理器在 RISC-V 架构基础上进行了安全扩展。该系列处理器在软件的协调下可以虚拟出多个执行域 (Zone)，整体架构如 图 9.2 所示。每个 Zone 可以独立地运行各自的操作系统以及基于该操作系统的应用程序，操作系统运行在超级用户特权模式，应用程序运行在普通用户特权模式。处理器根据需要可以在不同的 Zone 里切换运行。当处理器切换到某一 Zone 运行时，他将实时占用整个物理核，并且处理器的域标识也将同时被更新成相应执行域的标识。Zone 的切换由运行在最高模式 (机器模式) 下的可信固件 (TF) 来完成。

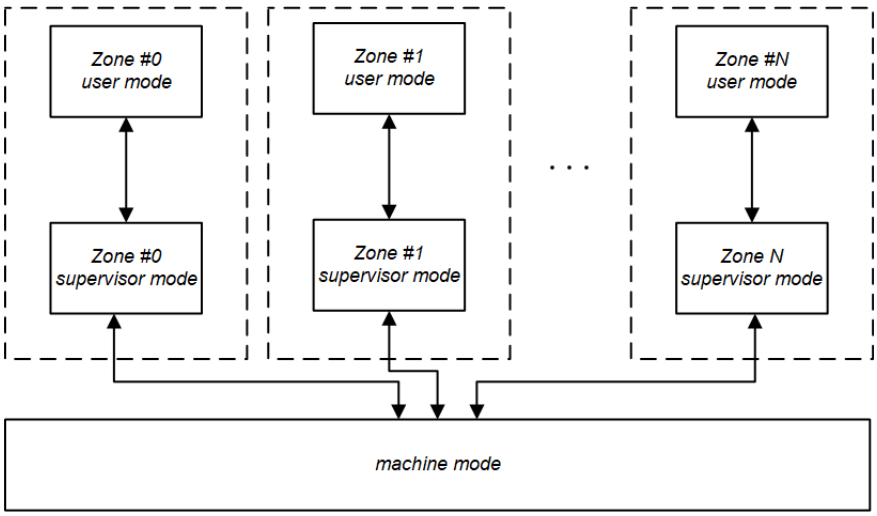


图 9.2: 玄铁 RISC-V 处理器的 Zones 和特权模式

9.3　系统安全架构

9.3.1　安全内存管理

每个硬件线程可以分时地运行在不同 Zone 里，当硬件线程在某个 Zone 里运行时，内存访问需要被隔离到相应的 Zone 内，其他 Zone 未经授权不允许访问该 Zone 的内存资源；同时该 Zone 未经授权也不允许访问属于其他 Zone 的内存资源。Zone 之间可以通过共享内存来传递数据。

Physical Memory Protection (PMP)

RISC-V 架构提供了一种 PMP 物理内存保护机制，用于隔离 M 模式与 S/U 模式下的内存访问。只有 M 模式才有权限配置 PMP。PMP 包含几组 (通常是 8 到 16 个) 地址寄存器以及相应的配置寄存器，这些配置寄存器可以授予或拒绝 S/U 模式的读、写和执行权限。PMP 同时也能保护内存映射 I/O (MMIO)，M 模式可信固件可以通过配置 PMP 来约束处理器对外设 I/O 的访问。

当硬件线程从一个 Zone 切换到另一个 Zone 时，PMP 配置同时也需要切换。M 模式可信固件需要先保存当前 Zone 的 PMP 配置，然后载入下一个即将切换的 Zone 的 PMP 配置，完成对内存和内存映射 I/O (MMIO) 访问权限的切换。

当多个 Zone 需要共享内存时，可以将需要共同访问的内存区域的访问权限同时授予给多个 Zone，也就是将该块内存的允许访问权限写到每个 Zone 的 PMP 配置表里，PMP 表会在 Zone 切换时由可信固件进行更新。图 9.3 是一个典型的多个 Zone 的 PMP 配置图，SHM 区域是 Zone 间允许共同访问的共享内存区域。

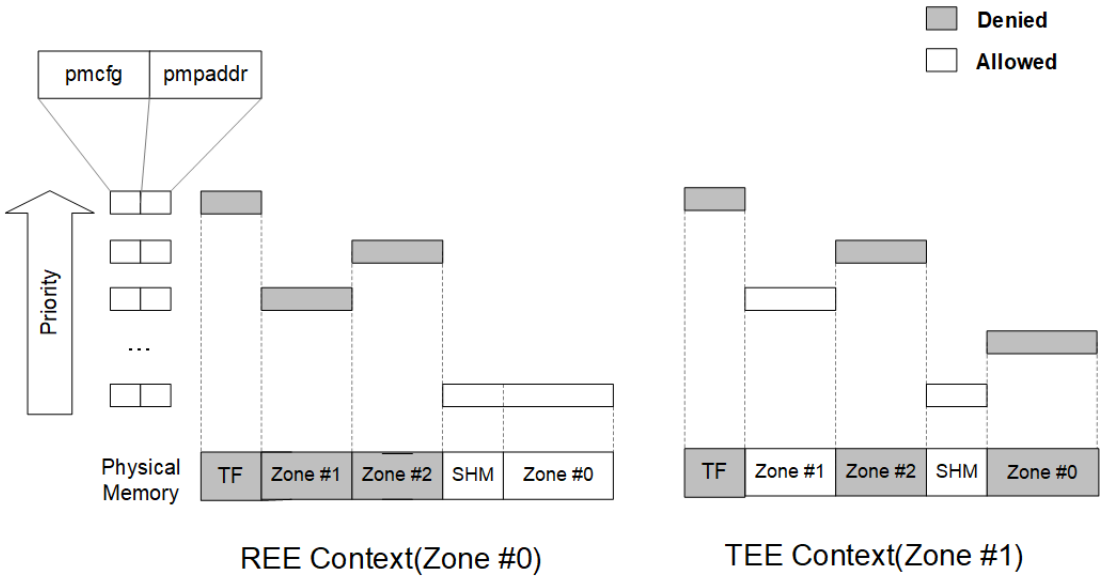


图 9.3: 不同 Zone 的 PMP 配置

I/O 物理内存保护 (IOPMP)

RISC-V 架构提供了一种 PMP 物理内存保护机制，用于保护 RISC-V 处理器在不同特权模式下的对内存和 MMIO 的访问。

总线上的其他主设备同样需要对内存的访问进行保护，也就是外设需要增加 IOPMP。IOPMP 可以像 PMP 一样定义访问权限，他会检查从总线过来的读、写传输是否符合权限访问规则，只有合法的读、写请求才能进一步传输到目标设备上。通常有两种方法来连接 IOPMP：

1. 请求端连接 IOPMP

在每个主设备和总线之间增加一个 IOPMP，类似 RISC-V 的 PMP。不同的主设备需要分别新增一个 IOPMP，IOPMP 之间互相独立。这种设计较为简单，也提供了灵活性，但 IOPMP 不能在主设备之间共享。如 图 9.4 所示:

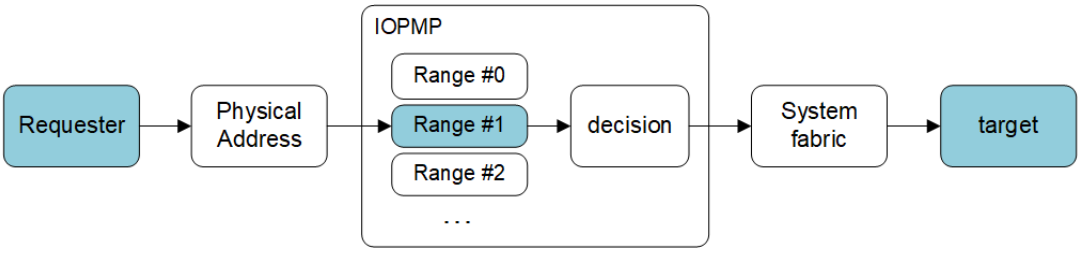


图 9.4: 请求端连接 IOPMP

2. 目标端连接 IOPMP

目标端的 IOPMP 需要对不同主设备发来的传输进行区分，这就需要每个主设备的访问请求都要附带额外的 Master ID。如 图 9.5 所示:

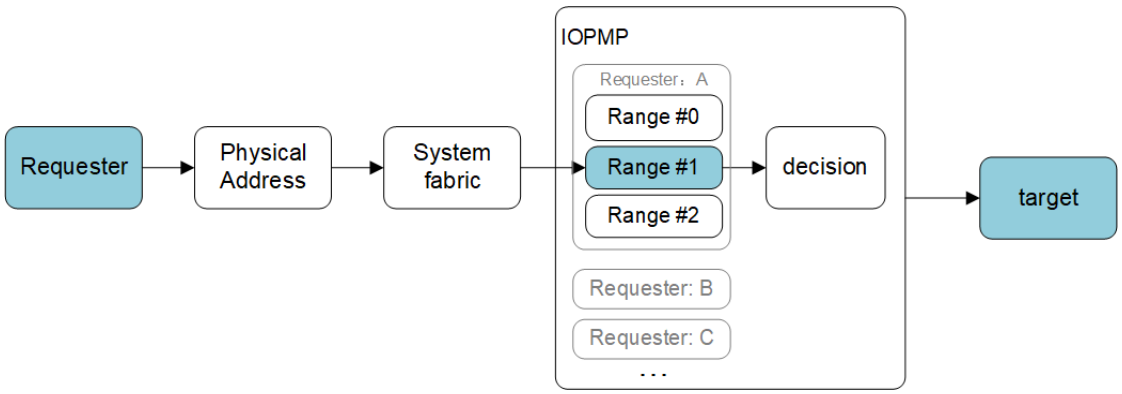


图 9.5: 目标端连接 IOPMP

图 9.6 是玄铁处理器采用请求端挂载 IOPMP 来构建安全 SoC 的系统框图:

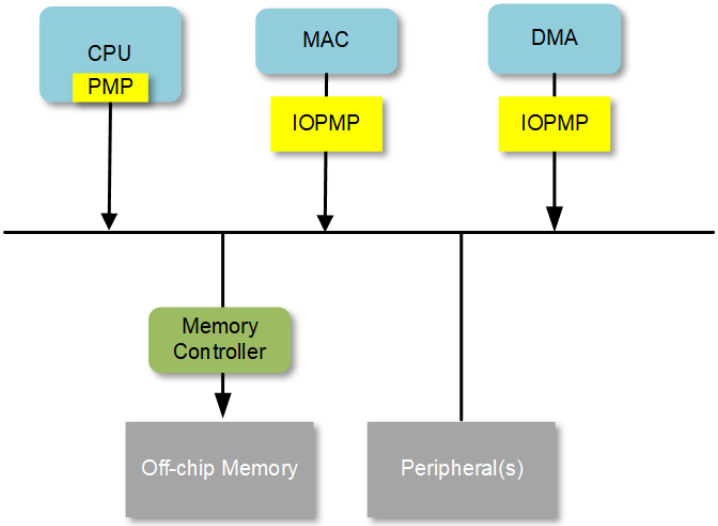


图 9.6: 基于 PMP 和 IOPMP 隔离的 SoC 架构

内存管理单元 (MMU)

内存管理单元 (MMU) 用于传统操作系统对虚拟内存的管理。MMU 实现了用户空间和内核空间的隔离。玄铁处理器的内存管理单元集成了可配置数量的 TLB 缓存, 每个 TLB 包含了从虚拟地址到物理地址的转换关系以及其访问权限。

Cache

当处理器在不同 Zone 运行时，由于每个 Zone 都有各自独立的 PMP 配置，PMP 限定了各自 Zone 对物理内存和内存映射 I/O 的访问权限和范围，PMP 确保内存、I/O 访问在 Zone 之间互不影响、干扰。

在玄铁 C 系列 RISC-V 架构处理器上，对内存的访问在 cache 命中后同样会受到物理内存保护 (PMP) 机制的保护，也就是任何对 cache 的访问都会先经过 PMP 的检查，只有 PMP 检查通过之后，才允许进一步访问 cache。多核的 Cache 一致性同样受到 PMP 保护。

Device Coherence Port (DCP)

玄铁 C908X 提供了 DCP 端口。如 图 9.7 , DCP 是处理器上的一个 AXI slave 接口，外部 master 设备可以通过 DCP 端口访问处理器内部具有 Cache 一致性的数据，来提高处理器和外部 master 交互数据的效率。玄铁

C908X 并没有在 DCP 端口上增加外部 master 访问的保护，这需要连接到 DCP 端口上的 master 外接 IOPMP，通过 IOPMP 对访问进行保护。

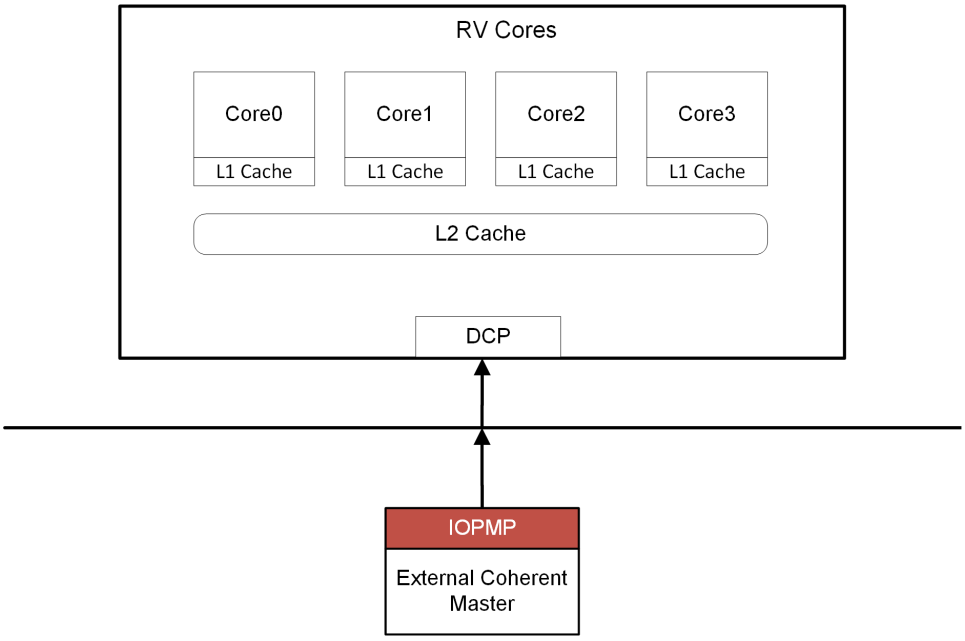


图 9.7: DCP 端口保护

9.3.2　安全中断

在 RISC-V 的 PLIC 规范中，存在 2 种中断模式的中断源：机器模式中断源、超级用户模式中断源。机器模式中断源只允许被机器模式响应；超级用户模式中断源允许被机器模式或超级用户模式响应，机器模式有权管控是否将该中断下发到超级用户模式去响应。RISC-V 架构的机器模式对中断的拦截特性为不同 Zone 的中断隔离提供了基础。不同模式的中断的响应情况如 表 9.1 所示：

表 9.1: RISC-V 中断响应模型

中断源目标模式	处理器当前模式	Delegation	是否响应中断	哪种模式响应中断
M-mode	M-mode	无效	响应	M-mode
	S-mode	无效	响应	M-mode
	U-mode	无效	响应	M-mode
S-mode	M-mode	0	响应	M-mode
		1	不响应	
	S-mode	0	响应	M-mode
		1	响应	S-mode
	U-mode	0	响应	M-mode
		1	响应	S-mode

利用 RISC-V 机器模式的对中断拦截的特性，常用的中断处理方式有以下两种：

1. 机器模式中断分发的方式

2. 中断分组的方式

机器模式中断分发的方式

机器模式具有对外部中断拦截的能力。所有的外部中断会先自陷到机器模式当中，运行在机器模式的可信固件 (TF) 会统一管控所有外部中断，并在识别中断源之后，再将中断“转发”到相应的 Zone 内完成响应。这种中断处理方法可以满足不同 Zone 间的中断隔离要求，但由于中断需要 TF 来转发，TF 在转发时需要对 Zone 上下文进行切换，这给中断响应带来了一定的中断延迟。

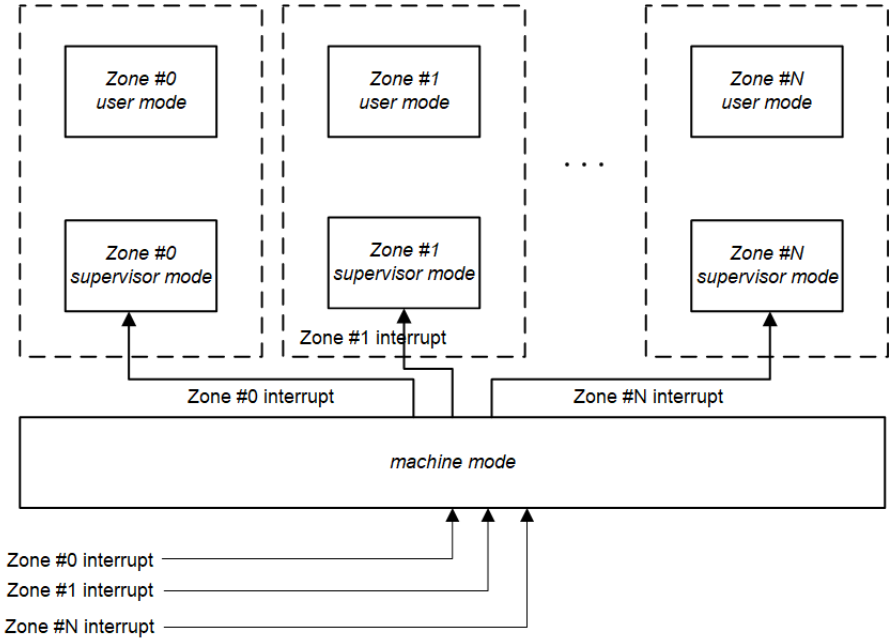


图 9.8: 玄铁 RISC-V 处理器 M 模式中断分发

该模式下，所有的外部中断都被配置成发往 M 模式的可信固件。可信固件先保存当前 Zone 的所有现场，读取外部中断号，再根据事先保存的 Zone 中断分配表，选择需要发往的 Zone 并获取该 Zone 的中断处理函数入口。可信固件可以通过读 stvec 寄存器获取该 Zone 的中断入口地址，在跳转到中断入口之前，需要将 PMP 配置切换到即将执行的中断处理函数所在的 Zone，并检查中断处理函数地址的合法性，最终通过 mret 指令跳到下一个 Zone 执行。中断处理完成之后，中断处理函数需要通过 ecall 回到 M 模式，M 模式可信固件将会恢复原先被打断的 Zone 的现场，回到该 Zone 继续运行。

中断分组模式

中断处理统一通过 M 模式转发会带来较大的中断延迟。并且中断处理程序在执行完成之后还需要通过 ecall 回到 M 模式，这会给已有的中断处理程序 (特别是 Linux) 造成不兼容性。

PLIC(Platform Level Interrupt Controller) 平台中断控制器提供了对每个中断源以及其中断目标的单独控制，也就是每个中断源的中断发往哪一个硬件线程以及该硬件线程上的哪种模式可以被单独配置。目前的应用场景通常会把处理器的执行环境分为 REE(Rich Execution Environment) 和 TEE(Trusted Execution Environment)，普通中断在 REE 里响应，安全中断会在 TEE 里响应，而且大部分的硬件中断都是普通中断，只有极少数的硬件中断 (如安全定时器) 才是安全中断。为了减少机器模式统一处理中断带来的中断延迟，本方案采用将中断分组的方式，将属于当前 Zone 的中断源的中断配置成在 Zone 内响应，不属于当前 Zone 的中断源的中断配置成在机器模式响应，以此来减少中断延迟。中断上下文有如下几种场景：

- REE 产生普通中断

- REE 产生安全中断
- TEE 产生普通中断
- TEE 产生安全中断

REE 产生普通中断、REE 产生安全中断

如 图 9.9 ，当处理器工作在 REE（Zone #0）时，TF 需要：

1. 将普通中断的中断源配置成超级用户模式使能
2. 将安全中断的中断源配置成机器模式使能
3. 将 mideleg 寄存器的第 1 位 (SSIE_DELEG)、第 5 位 (STIE_DELEG) 和第 9 位 (SEIE_DELEG) 置位（假设软件中断和时钟中断都被配置成普通中断）
4. 使能 mstatus.MIE 和 mstatus.SIE, 使能 mie.MEIE, mie.MSIE, mie.MTIE, mie.SEIE, mie.SSIE, mie.STIE

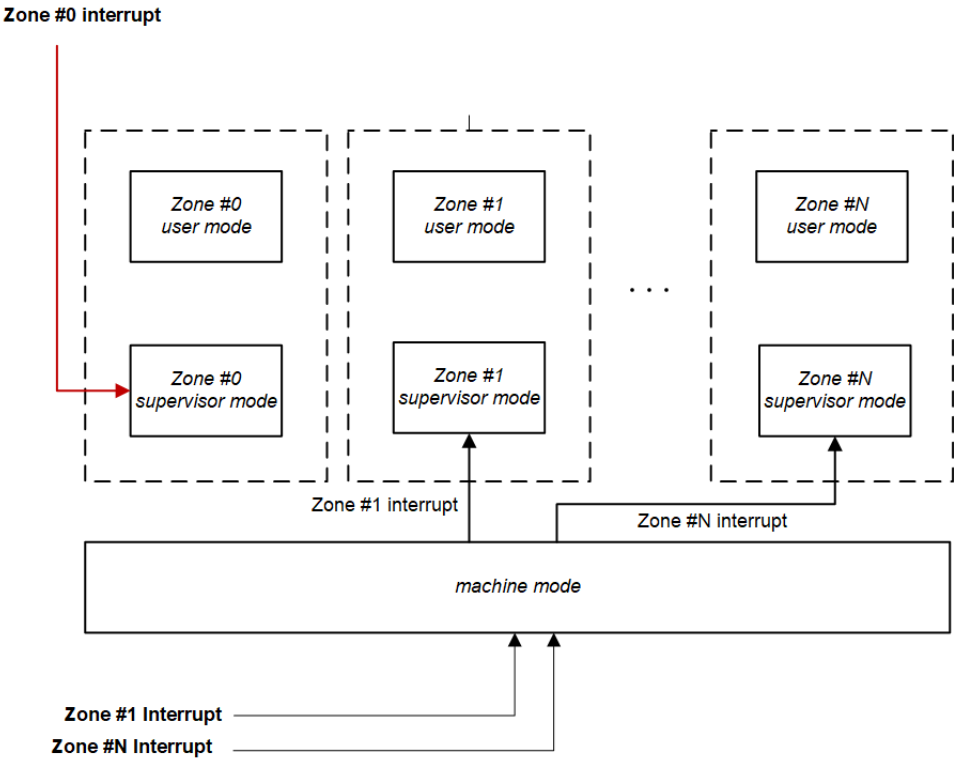


图 9.9: 处理器运行在 Zone #0 时的中断处理规则

TEE 产生普通中断、TEE 产生安全中断

如 图 9.10 ，当处理器工作在 TEE（Zone #1）时，TF 需要：

1. 将普通中断的中断源配置成超级用户模式使能
2. 将安全中断的中断源配置成机器模式使能
3. 将 mideleg 寄存器的第 1 位 (SSIE_DELEG)、第 5 位 (STIE_DELEG) 和第 9 位 (SEIE_DELEG) 置位（假设软件中断和时钟中断都被配置成普通中断）
4. 使能 mstatus.MIE 和 mstatus.SIE, 使能 mie.MEIE, mie.MSIE, mie.MTIE, mie.SEIE, mie.SSIE, mie.STIE

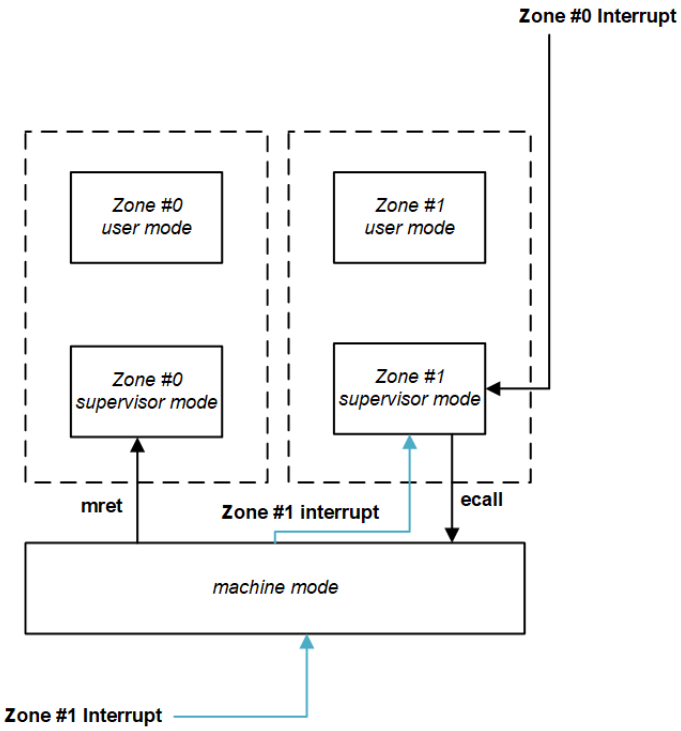


图 9.10: 处理器运行在 Zone #1 时的中断处理规则

9.3.3　安全访问控制

机器模式是 RISC-V 中 hart（hardware thread，硬件线程）可以执行的最高权限模式。在 M 模式下运行的 hart 对内存，I/O 和一些对于启动和配置系统来说必要的底层功能有着完全的使用权。因此它是唯一所有标准 RISC-V 处理器都必须实现的权限模式。实际上简单的 RISC-V 微控制器仅支持 M 模式。

机器模式最重要的特性是拦截和处理异常的能力。默认情况下，发生异常（不论在什么权限模式下）的时候，控制权都会被移交到 M 模式的异常处理程序。但是 Unix 系统中的大多数异常都应该在 S 模式下进行。M 模式的异常处理程序可以将异常重新导向 S 模式，但这些额外的操作会减慢大多数异常的处理速度。因此，RISC-V 提供了一种异常委托机制。通过该机制可以选择性地将中断和同步异常交给 S 模式处理，而完全绕过 M 模式。mideleg（Machine Interrupt Delegation，机器中断委托）CSR 控制将哪些中断或者异常委托给 S 模式。

请注意，无论委派设置是怎样的，发生中断/异常时控制权都不会移交给权限更低的模式。在 M 模式的中断/异常总是在 M 模式下处理。S 模式下的中断/异常，根据具体的委派设置，可能由 M 模式或 S 模式处理，但永远不会由 U 模式处理。

虽然机器模式对于简单的嵌入式系统已经足够，但它仅适用于那些整个代码库都可信的情况，因为 M 模式可以自由地访问硬件平台。更常见的情况是，不能信任所有的应用程序代码，因为不能事先得知这一点，或者它太大，难以证明正确性。因此，RISC-V 提供了保护系统免受不可信的代码危害的机制，并且为不受信任的进程提供隔离保护。这些不可信的代码还必须被限制只能访问自己那部分内存。实现了 M 和 S/U 模式的处理器具有一个叫做物理内存保护（PMP，Physical Memory Protection）的功能，允许 M 模式指定 S/U 模式可以访问的内存地址。除了内存，PMP 也可以用来约束对 Memory-Mapped I/O(MMIO) 的访问。有了 PMP，机器模式便可以对不受信任的 S/U 对内存和外设的访问进行管制。

9.3.4　安全调试

当 CPU 的安全调试使能关闭，CPU 将会忽略外部发起的同步调试请求、异步调试请求和内部的硬件断点所产生的调试请求。此时，EBREAK 指令不产生效果，会被当成 nop 指令处理。

10 中断控制器

10.1　CLINT 中断控制器

C908X 实现了处理器核局部中断控制器（以下简称 CLINT），是一个内存地址映射的模块，用于处理软件中断和计时器中断。

10.1.1　CLINT 寄存器地址映射

CLINT 中断控制器占据 64KB 内存空间。其高 13 位地址由 SoC 硬件集成决定，低 27 位地址映射如 表 10.1 所示。所有寄存器仅支持字对齐的访问。CLINT 采用连续编址的方式，对于多 cluster 多核，CLINT 不关心 cluster 的数量，仅关心核的数量，各个核对应的地址空间是连续的。比如，有两个 cluster，cluster0 有 2 个核，cluster1 有 4 个核，cluster0 的 2 个核各寄存器地址在下表中核 0、1 给出，cluster1 的 4 个核各寄存器地址在下表中核 2、3、4、5 给出。CLINT 最多支持 256 个核。

表 10.1: CLINT 寄存器存储器映射地址

寄存器	地址	名称	类型	初始值	描述
MSIP	0x4000000	MSIP0	读/写	0x00000000	核 0 机器模式 软件中断配置寄存器 高位绑 0 ， bit[0] 有效
	0x4000004	MSIP1	读/写	0x00000000	核 1 机器模式 软件中断配置寄存器 高位绑 0 ， bit[0] 有效
	…	…	…	…	…
	0x400003c	MSIP15	读/写	0x00000000	核 15 机器模式 软件中断配置寄存器 高位绑 0 ， bit[0] 有效
	0x4000040	MSIP16	读/写	0x00000000	核 16 机器模式 软件中断配置寄存器 高位绑 0 ， bit[0] 有效
	0x4000044	MSIP17	读/写	0x00000000	核 17 机器模式 软件中断配置寄存器 高位绑 0 ， bit[0] 有效

续下页

表 10.1 - 接上页

寄存器	地址	名称	类型	初始值	描述
	…	…	…	…	…
	0x4000000 +4*n	MSIPn	读/写	0x00000000	n=hart_id ， n<256
MTIMECMP	0x4004000	MTIMECMPL0	读/写	0xFFFFFFFF	核 0 机器模式 时钟计时器比较值寄存器 (低 32 位)
	0x4004004	MTIMECMPH0	读/写	0xFFFFFFFF	核 0 机器模式 时钟计时器比较值寄存器 (高 32 位)
	0x4004008	MTIMECMPL1	读/写	0xFFFFFFFF	核 1 机器模式 时钟计时器比较值寄存器 (低 32 位)
	0x400400c	MTIMECMPH1	读/写	0xFFFFFFFF	核 1 机器模式 时钟计时器比较值寄存器 (高 32 位)
	…	…	…	…	…
	0x4004078	MTIMECMPL15	读/写	0xFFFFFFFF	核 15 机器模式 时钟计时器比较值寄存器 (低 32 位)
	0x400407c	MTIMECMPH15	读/写	0xFFFFFFFF	核 15 机器模式 时钟计时器比较值寄存器 (高 32 位)
	0x4004080	MTIMECMPL16	读/写	0xFFFFFFFF	核 16 机器模式 时钟计时器比较值寄存器 (低 32 位)
	0x4004084	MTIMECMPH16	读/写	0xFFFFFFFF	核 16 机器模式 时钟计时器比较值寄存器 (高 32 位)
	0x4004088	MTIMECMPL17	读/写	0xFFFFFFFF	核 17 机器模式 时钟计时器比较值寄存器 (低 32 位)
	0x400408c	MTIMECMPH17	读/写	0xFFFFFFFF	核 17 机器模式 时钟计时器比较值寄存器 (高 32 位)
	…	…	…	…	…
	0x4004000 +8*n	MTIMECMPLn	读/写	0xFFFFFFFF	n =hart_id ， n<256

续下页

表 10.1 - 接上页

寄存器	地址	名称	类型	初始值	描述
	0x4004000 +8*n+4	MTIMECMPHn	读/写	0xFFFFFFFF	n =hart_id ， n<256
CLINT_ MTIME	0x400bff8	CLINT_MTIMEL	只读	0x00000000	机器模式时钟计时器（玄铁自扩展的寄存器）
	0x400bffc	CLINT_MTIMEH	只读	0x00000000	机器模式时钟计时器（玄铁自扩展的寄存器）
SSIP	0x400c000	SSIP0	读/写	0x00000000	核 0 超级用户模式软件中断配置寄存器高位绑 0 ， bit[0] 有效
	0x400c004	SSIP1	读/写	0x00000000	核 1 超级用户模式软件中断配置寄存器高位绑 0 ， bit[0] 有效
	…	…	…	…	…
	0x400c03c	SSIP15	读/写	0x00000000	核 15 超级用户模式软件中断配置寄存器高位绑 0 ， bit[0] 有效
	0x400c040	SSIP16	读/写	0x00000000	核 16 超级用户模式软件中断配置寄存器高位绑 0 ， bit[0] 有效
	0x400c044	SSIP17	读/写	0x00000000	核 17 超级用户模式软件中断配置寄存器高位绑 0 ， bit[0] 有效
	…	…	…	…	…
	0x400c000 +4*n	SSIPn	读/写	0x00000000	n =hart_id ， n<256
STIMECMP	0x400d000	STIMECMPL0	读/写	0xFFFFFFFF	核 0 超级用户模式时钟计时器比较值寄存器 (低 32 位)
	0x400d004	STIMECMPH0	读/写	0xFFFFFFFF	核 0 超级用户模式时钟计时器比较值寄存器 (高 32 位)
	0x400d008	STIMECMPL1	读/写	0xFFFFFFFF	核 1 超级用户模式时钟计时器比较值寄存器 (低 32 位)
	0x400d00c	STIMECMPH1	读/写	0xFFFFFFFF	核 1 超级用户模式时钟计时器比较值寄存器 (高 32 位)

续下页

表 10.1 - 接上页

寄存器	地址	名称	类型	初始值	描述
	…	…	…	…	…
	0x400d078	STIMECMPL15	读/写	0xFFFFFFFF	核 15 超级用户模式 时钟计时器比较值寄存器 (低 32 位)
	0x400d07c	STIMECMPH15	读/写	0xFFFFFFFF	核 15 超级用户模式 时钟计时器比较值寄存器 (高 32 位)
	0x400d080	STIMECMPL16	读/写	0xFFFFFFFF	核 16 超级用户模式 时钟计时器比较值寄存器 (低 32 位)
	0x400d084	STIMECMPH16	读/写	0xFFFFFFFF	核 16 超级用户模式 时钟计时器比较值寄存器 (高 32 位)
	0x400d088	STIMECMPL17	读/写	0xFFFFFFFF	核 17 超级用户模式 时钟计时器比较值寄存器 (低 32 位)
	0x400d08c	STIMECMPH17	读/写	0xFFFFFFFF	核 17 超级用户模式 时钟计时器比较值寄存器 (高 32 位)
	…	…	…	…	…
	0x400d000 +8*n	STIMECMPLn	读/写	0xFFFFFFFF	n = hart_id ， n<256
	0x400d000 +8*n+4	STIMECMPHn	读/写	0xFFFFFFFF	n = hart_id ， n<256
CLINT_ STIME	0x400fff8	CLINT_STIMEL	只读	0x00000000	超级用户模式时钟计时器 （玄铁自扩展的寄存器）
	0x400fffc	CLINT_STIMEH	只读	0x00000000	超级用户模式时钟计时器 （玄铁自扩展的寄存器）

10.1.2　软件中断

CLINT 可用于生成软件中断。

软件中断通过配置地址映射的软件中断配置寄存器进行控制。其中机器模式软件中断由机器模式软件中断配置寄存器（MSIP）控制，超级用户模式软件中断由超级用户模式软件中断配置寄存器（SSIP）控制。

用户可通过将 xSIP 位置 1 的方式，产生软件中断；可通过将 xSIP 位清 0 的方式，清除软件中断。其中 CLINT 超级用户模式软件中断请求，仅在对应核使能 CLINTEE 位时有效。

机器模式下拥有修改访问所有软件中断相关寄存器的权限；超级用户模式下仅具有访问修改超级用户

模式软件中断配置寄存器（SSIP）的权限；普通用户模式没有权限。

两组寄存器的结构相同，其寄存器位分布和位定义如 图 10.1 所示。

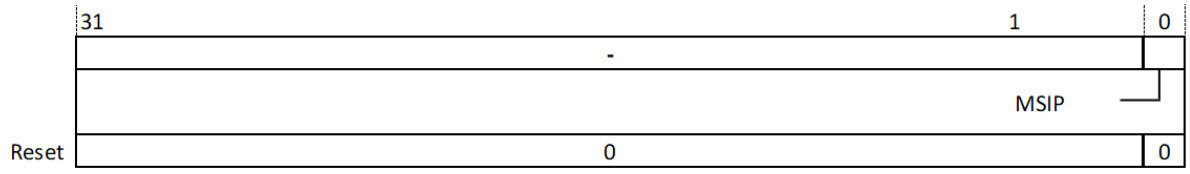


图 10.1: 机器模式软件中断配置寄存器（MSIP）

MSIP：机器模式软件中断等待位

该位表示机器模式软件中断的中断状态。

- 当 MSIP 位置 1，当前有有效的机器模式软件中断请求。
- 当 MSIP 位置 0，当前没有有效的机器模式软件中断请求。

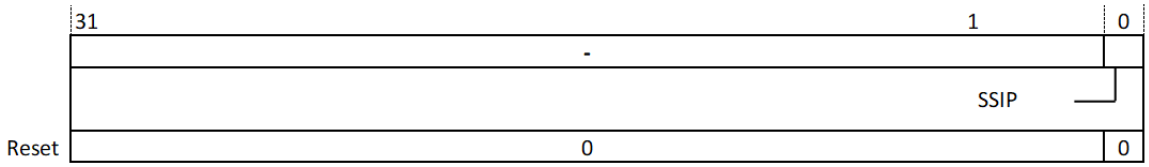


图 10.2: 超级用户模式软件中断配置寄存器（SSIP）

SSIP：超级用户模式软件中断等待位

该位表示超级用户模式软件中断的中断状态。

- 当 SSIP 位置 1，当前有有效的超级用户软件中断请求。
- 当 SSIP 位置 0，当前没有有效的超级用户软件中断请求。

10.1.3　计时器

在多核多 cluster 系统中仅存在一个 64 位系统计时器，要求在 always-on 电压域进行工作。系统计时器不可写，仅能通过 reset 清 0。系统计时器的当前值可以通过读取机器模式时钟计时器寄存器（CLINT_MTIME）和超级用户模式计时器寄存器（CLINT_STIME）获取，也可以通过读取 PMU 的 TIME 寄存器获取。系统计时器的主要作用是为多个核心提供统一的事件基准。

多核多 cluster 系统中仅有一组 64 位的机器模式时钟计时器寄存器（CLINT_MTIMEL，CLINT_MTIMEH）和一组 64 位的超级用户模式时钟计时器寄存器（CLINT_STIMEL，CLINT_STIMEH）。这些寄存器均可通过地址字对齐读取的方式，分别读取其高 32 位或低 32 位。

注解：CLINT_MTIME 和 CLINT_STIME 为玄铁自扩展的寄存器。

CLINT_MTIMEH/CLINT_MTIMEL：

机器模式时钟计时器寄存器高位/低位，该寄存器存储了时钟计时器值。

- CLINT_MTIMEH：时钟计时器高 32 位；
- CLINT_MTIMEL：时钟计时器低 32 位；

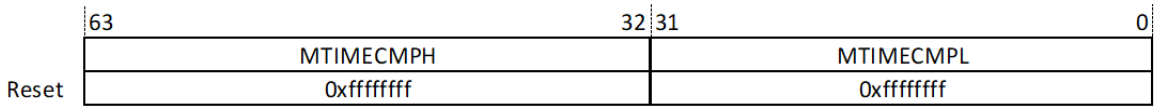


图 10.3: CLINT_MTIME 寄存器

CLINT_STIMEH/CLINT_STIMEL：

超级用户模式时钟计时器寄存器高位/低位，该寄存器存储了时钟计时器值。

- CLINT_STIMEH：时钟计时器高 32 位；
- CLINT_STIMEL：时钟计时器低 32 位；

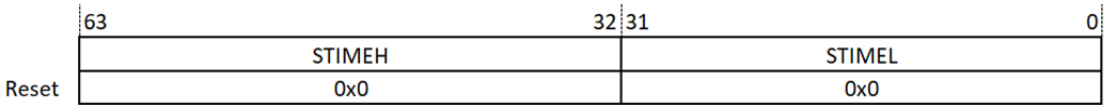


图 10.4: CLINT_STIME 寄存器

10.1.4　计时器中断

CLINT 可用于生成计时器中断。

每一个核均有一组 64 位的*机器模式时钟计时器比较值寄存器（MTIMECMPL, MTIMECMPH）*和一组 64 位的超级用户模式时钟计时器比较值寄存器*（STIMECMPL, STIMECMPH）*。这些寄存器均可以通过地址字对齐访问的方式，分别修改其高 32 位或低 32 位。

CLINT 通过比较 {CMPH[31:0],CMPL[31:0]} 的值与系统计时器的当前值，确定是否产生计时器中断。

当 {CMPH[31:0],CMPL[31:0]} 大于系统计时器的值时，不产生中断；

当 {CMPH[31:0],CMPL[31:0]} 小于或等于系统计时器的值时，CLINT 产生对应的计时器中断。

软件可通过改写 MTIMECMP/STIMECMP 的值来清除对应的计时器中断。其中, 超级用户模式计时器中断请求，仅在对应核使能 CLINTEE 位，且对应核的机器模式环境配置寄存器（MENVCFG）的 STCE 字段为 0 时有效。

机器模式下拥有修改访问所有计时器中断相关寄存器的权限；超级用户模式下仅具有访问修改超级用户模式时钟计时器比较值寄存器（STIMECMPL, STIMECMPH）的权限；普通用户模式没有权限。

每组寄存器结构相同，其寄存器位分布和位定义如 图 10.5 所示。

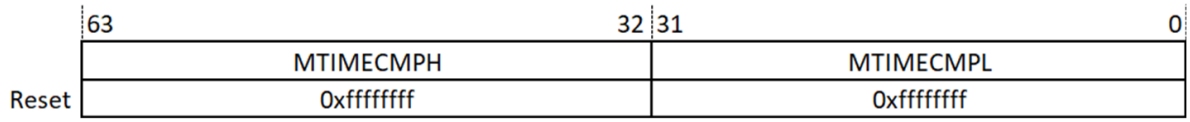


图 10.5: 机器模式计时器中断比较值寄存器（高/低）

MTIMECMPH/MTIMECMPL：

机器模式计时器中断比较值寄存器高位/低位。

该寄存器存储了计时器比较值

- MTIMECMPH：计时器比较值高 32 位；
- MTIMECMPL：计时器比较值低 32 位；

	63 — 32	31 — 0
	STIMECMPH	STIMECMPL
Reset	0xffffffff	0xffffffff

图 10.6: 超级用户模式计时器中断比较值寄存器（高/低）

STIMECMPH/STIMECMPL：

超级用户模式计时器中断比较值寄存器高位/低位。

该寄存器存储了计时器比较值

- STIMECMPH：计时器比较值高 32 位；
- STIMECMPL：计时器比较值低 32 位；

10.2　PLIC 中断控制器

平台级别中断控制器（以下简称 PLIC），用于对外部中断源进行采样，优先级仲裁和分发。

在 PLIC 模型中每个核的机器模式和超级用户模式均可作为有效中断目标。

C908X 实现的 PLIC 单元基本功能如下：

- PLIC 最多支持 256 个核，每个核有 2 个目标，分别是机器模式和超级用户模式。
- 最多 1023 个中断源采样，支持电平中断，脉冲中断；
- 32 个级别的中断优先级；
- 每个中断目标的中断使能独立维护；
- 每个中断目标的中断阈值独立维护；
- PLIC 寄存器访问权限可配置。

10.2.1　中断的仲裁

在 PLIC 中，只有符合条件的中断源才会参与对某个中断目标的仲裁。满足的条件如下：

- 中断源处于等待状态（IP = 1）
- 中断优先级大于 0。
- 对于该中断目标的使能位打开。

当 PLIC 中对某个中断目标有多个中断处于 Pending 状态时，PLIC 仲裁出优先级最高的中断。C908X 的 PLIC 实现中，机器模式中断优先级始终高于超级用户模式中断。当模式相同的情况下，优先级配置寄存器的值越大，优先级越高，优先级为 0 的中断无效；如多个中断拥有相同的优先级，则 ID 较小的优先处理。

PLIC 会将仲裁结果以中断 ID 的形式更新入对应中断目标的中断响应/完成寄存器。

10.2.2　中断的请求与响应

当 PLIC 对特定中断目标存在有效中断请求，且优先级大于该中断目标的中断阈值时，会向该中断目标发起中断请求。当该中断目标收到中断请求，且可响应该中断请求时，需要向 PLIC 发送中断响应消息。

中断响应机制如下：

- 中断目标向其对应的中断响应/完成寄存器发起一个读操作。该读操作将返回一个 ID，表示当前 PLIC 仲裁出的中断 ID。中断目标根据所获得的 ID 进行下一步处理。如果获得的中断 ID 为 0，表示没有有效中断请求，中断目标结束中断处理。
- 当 PLIC 收到中断目标发起的读操作，且返回相应 ID 后，会将该 ID 对应的中断源 IP 位清 0，且在中断完成之前屏蔽该中断源的后续采样。

当配置 L2 ECC 功能时，L2 ECC FATAL 中断号根据客户对中断控制器的集成决定。

10.2.3　中断的完成

当中断目标完成中断处理后，需要向 PLIC 发送中断完成消息。中断完成机制如下：

- 中断目标向中断响应/完成寄存器发起写操作，写操作的值为本次完成的中断 ID。如果中断类型为电平中断，则发起上述写操作之前还需清除外部中断源。
- PLIC 收到该中断完成请求后，不更新中断响应/完成寄存器，解除 ID 对应的中断源采样屏蔽，结束整个中断处理过程。

10.2.4　PLIC 寄存器地址映射

PLIC 中断控制器占据 64MB 内存空间。其高 13 位地址由 SoC 硬件集成决定，低 27 位地址映射如 表 10.2 所示。所有寄存器仅支持地址字对齐的访问，即 PLIC 的寄存器要通过 load word 指令进行访问，访问结果放在 64 位 GPR 的低 32 位。PLIC 采用连续编址的方式，对于多 cluster 多核，PLIC 不关心 cluster 的数量，仅关心核的数量，各个核对应的地址空间是连续的。比如有两个 cluster，cluster0 有 2 个核，cluster1 有 4 个核，cluster0 的 2 个核各寄存器地址在下表中核 0、1 给出，cluster1 的 4 个核各寄存器地址在下表中核 2、3、4、5 给出。

表 10.2: PLIC 地址映射

寄存器类别	地址	名称	类型	初始值	描述
PLIC_PRIO	0x0000000	-	-	-	-
	0x0000004	PLIC_PRIO1	R/W	0x0	中断源 1~1023 优先级配置寄存器
	0x0000008	PLIC_PRIO2	R/W	0x0	
	0x000000C	PLIC_PRIO3	R/W	0x0	
	…	…	…	…	
	0x0000FFC	PLIC_PRIO1023	R/W	0x0	
PLIC_IP	0x0001000	PLIC_IP0	R/W	0x0	1~31 号中断 中断等待寄存器

续下页

表 10.2 - 接上页

寄存器类别	地址	名称	类型	初始值	描述
	0x0001004	PLIC_IP1	R/W	0x0	32~63 号中断 中断等待寄存器
	…	…	…	…	…
	0x000107C	PLIC_IP31	R/W	0x0	992~1023 号中断 中断等待寄存器
-	Reserved	-	-	-	-
PLIC_MIE PLIC_SIE	0x0002000	PLIC_H0_MIE0	R/W	0x0	核 0 1~31 号 机器模式中断使能寄存器
	0x0002004	PLIC_H0_MIE1	R/W	0x0	核 0 32~63 机器模式中断使能寄存器
	…	…	…	…	…
	0x000207C	PLIC_H0_MIE31	R/W	0x0	核 0 992~1023 号 机器模式中断使能寄存器
	0x0002080	PLIC_H0_SIE0	R/W	0x0	核 0 1~31 号 超级用户模式中断使能寄存器
	0x0002084	PLIC_H0_SIE1	R/W	0x0	核 0 32~63 号 超级用户模式中断使能寄存器
	…	…	…	…	…
	0x00020FC	PLIC_H0_SIE31	R/W	0x0	核 0 992~1023 号 超级用户模式中断使能寄存器
	0x0002100	PLIC_H1_MIE0	R/W	0x0	核 1 1~31 号 机器模式中断使能寄存器
	0x0002104	PLIC_H1_MIE1	R/W	0x0	核 1 32~63 号 机器模式中断使能寄存器
	…	…	…	…	…
	0x000217C	PLIC_H1_MIE31	R/W	0x0	核 1 992~1023 号 机器模式中断使能寄存器
	0x0002180	PLIC_H1_SIE0	R/W	0x0	核 1 1~31 号 超级用户模式中断使能寄存器
	0x0002184	PLIC_H1_SIE1	R/W	0x0	核 1 32~63 号 超级用户模式中断使能寄存器
	…	…	…	…	…
	0x00021FC	PLIC_H1_SIE31	R/W	0x0	核 1 992~1023 号 超级用户模式中断使能寄存器
	…	…	…	…	…

续下页

表 10.2 - 接上页

寄存器类别	地址	名称	类型	初始值	描述
	0x0002000 +0x100*n	PLIC_Hn_MIE0	R/W	0x0	核 hart_id 1~31 号 机器模式中断使能寄存器 n=hart_id ， n<256
	0x0002004 +0x100*n	PLIC_Hn_MIE1	R/W	0x0	核 hart_id 32~63 号 机器模式中断使能寄存器 n=hart_id ， n<256
	…	…	…	…	…
	0x000207C +0x100*n	PLIC_Hn_MIE31	R/W	0x0	核 hart_id 992~1023 号 机器模式中断使能寄存器 n=hart_id ， n<256
	0x0002080 +0x100*n	PLIC_Hn_SIE0	R/W	0x0	核 hart_id 1~31 号 超级用户模式中断使能寄存器 n=hart_id ， n<256
	0x0002084 +0x100*n	PLIC_Hn_SIE1	R/W	0x0	核 hart_id 32~63 号 超级用户模式中断使能寄存器 n=hart_id ， n<256
	…	…	…	…	…
	0x00020FC +0x100*n	PLIC_Hn_SIE31	R/W	0x0	核 hart_id 992~1023 号 超级用户模式中断使能寄存器 n=hart_id ， n<256
PLIC_CTRL	0x01FFFFC	PLIC_CTRL	R/W	0x0	PLIC 权限控制寄存器
PLIC_MTH PLIC_MCLAIM PLIC_STH PLIC_SCLAIM	0x0200000	PLIC_H0_MTH	R/W	0x0	核 0 机器模式 中断阈值寄存器
	0x0200004	PLIC_H0_MCLAIM	R/W	0x0	核 0 机器模式 中断响应/完成寄存器
	Reserved	-	-	-	-
	0x0201000	PLIC_H0_STH	R/W	0x0	核 0 超级用户模式 中断阈值寄存器
	0x0201004	PLIC_H0_SCLAIM	R/W	0x0	核 0 超级用户模式 中断响应/完成寄存器
	Reserved	-	-	-	-
	0x0202000	PLIC_H1_MTH	R/W	0x0	核 1 机器模式 中断阈值寄存器
	0x0202004	PLIC_H1_MCLAIM	R/W	0x0	核 1 机器模式 中断响应/完成寄存器
	Reserved	-	-	-	-

续下页

表 10.2 - 接上页

寄存器类别	地址	名称	类型	初始值	描述
	0x0203000	PLIC_H1_STH	R/W	0x0	核 1 超级用户模式 中断阈值寄存器
	0x0203004	PLIC_H1_SCLAIM	R/W	0x0	核 1 超级用户模式 中断响应/完成寄存器
	Reserved	-	-	-	-
	0x0200000 +0x2000*n	PLIC_Hn_MTH	R/W	0x0	核 hart_id 机器模式 中断阈值寄存器 n=hart_id ， n<256
	0x0200004 +0x2000*n	PLIC_Hn_MCLAIM	R/W	0x0	核 hart_id 机器模式 中断响应/完成寄存器 n=hart_id ， n<256
	Reserved	-	-	-	-
	0x0201000 +0x2000*n	PLIC_Hn_STH	R/W	0x0	核 hart_id 超级用户模式 中断阈值寄存器 n=hart_id ， n<256
	0x0201004 +0x2000*n	PLIC_Hn_SCLAIM	R/W	0x0	核 hart_id 超级用户模式 中断响应/完成寄存器 n=hart_id ， n<256

如 图 10.7 所示, PLIC 和 CLINT 占据的总地址空间大小为 128 MB, 这段空间的基地址由 pad_cpu_apb_base（输入端口，请参照 C908X 集成手册）决定。需要注意的是，这段空间的属性需要设置为 Strong Ordered。

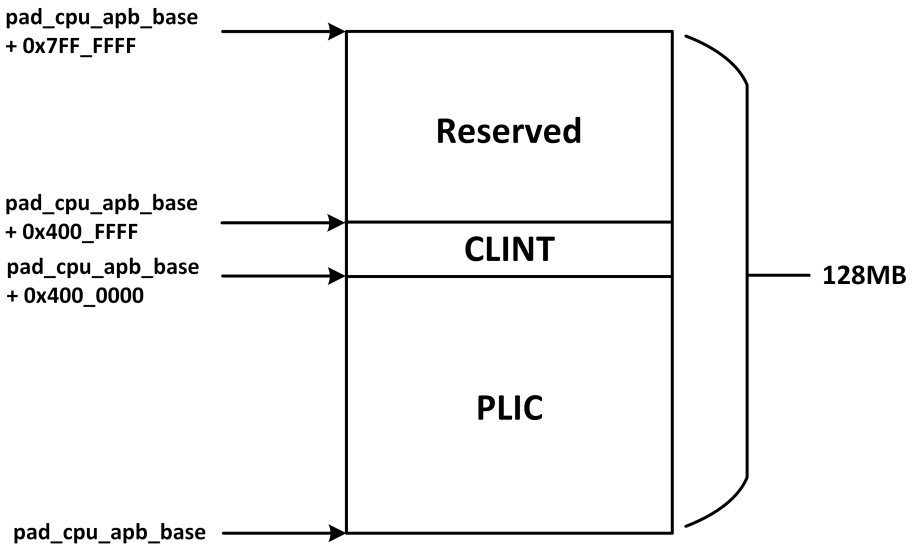


图 10.7: PLIC&CLINT 地址空间

10.2.5　中断优先级配置寄存器（PLIC_PRIO）

该寄存器设定中断源的优先级。寄存器读写权限参考权限控制寄存器（PLIC_CTRL）描述。寄存器位分布和位定义如 图 10.8 所示。

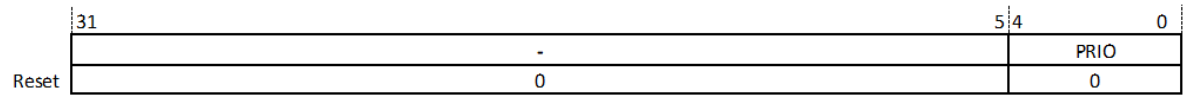


图 10.8: 中断优先级配置寄存器 (PLIC_PRIO)

- **PRIO：中断优先级**

  优先级配置寄存器低 5 位可写，支持 32 个不同级别的优先级。其中优先级设置为 0 表示该中断无效。

  机器模式中断优先级无条件高于超级用户模式中断。当模式相同时，优先级 1 为最低优先级，优先级 31 为最高。当优先级相同时，进一步比较中断源 ID，ID 较小的有高优先级。

10.2.6　中断等待寄存器（PLIC_IP）

每一个中断源的等待状态都可以通过读取中断等待寄存器中的信息获取。对于中断 ID 为 N 的中断，其中断信息存储于 PLIC_IP x（x=N/32）寄存器中的 IP y 上（y = N mod 32）。其中 PLIC_IP0 寄存器的第 0 位固定绑 0。寄存器读写权限参考权限控制寄存器（PLIC_CTRL）描述。寄存器位分布和位定义如 图 10.9 所示。

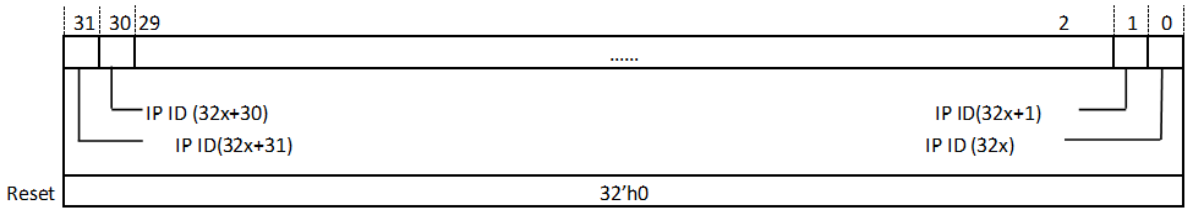


图 10.9: PLIC_IP x 中断等待寄存器 (PLIC_IP)

IP：中断等待状态

该位表示对应中断源的中断等待状态。

- 当 IP 位为 1 时，表示当前该外部中断源存在等待响应的中断。该位可通过内存存储指令置 1。在对应中断源采样逻辑采样到有效电平或脉冲中断后也会将该位置 1。
- 当 IP 位为 0 时，表示当前该外部中断源没有等待响应的中断。该位可通过内存存储指令清 0。当中断被响应后，PLIC 会将对应 IP 位清除。

10.2.7　中断使能寄存器（PLIC_IE）

每个中断目标对每个中断源均有一个中断使能位，可用于使能对应的中断。其中机器模式中断使能寄存器用于使能机器模式外部中断，超级用户模式中断使能寄存器用于使能超级用户模式外部中断。

对于中断 ID 为 N 的中断，其中断使能信息存储于 PLIC_IE x（x=N/32）寄存器中的 IE y 上（y = N mod 32）。其中 ID0 对应的 IE 位固定绑 0。寄存器读写权限参考权限控制寄存器（PLIC_CTRL）描述。

寄存器位分布和位定义如 图 10.10 所示。

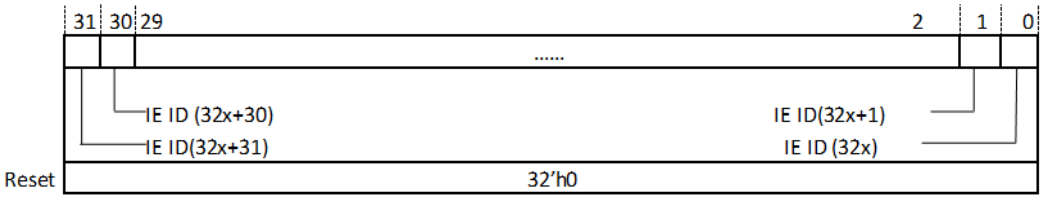


图 10.10: PLIC_IE x 中断使能寄存器 (PLIC_IE)

IE 中断使能：

该位表示对应中断源的中断使能状态。

- 当 IE 位为 1 时，表示中断对该目标使能。
- 当 IE 位为 0 时，表示中断对该目标屏蔽。

10.2.8　PLIC 权限控制寄存器（PLIC_CTRL）

PLIC 权限控制寄存器用于控制超级用户模式对 PLIC 部分寄存器的访问权限。

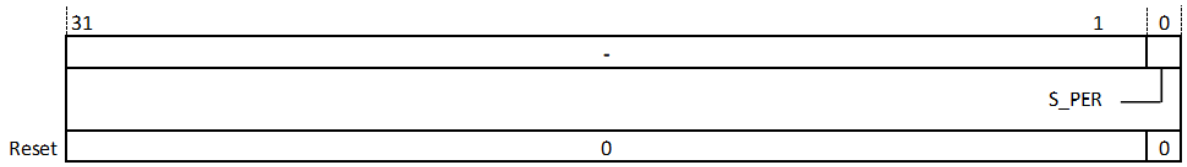


图 10.11: PLIC 权限控制寄存器 (PLIC_CTRL)

S_PER 访问权限控制位：

- 在 S_PER 为 0 时，仅机器模式拥有访问 PLIC 所有寄存器的权限。超级用户模式没有 PLIC 权限控制寄存器，中断优先级配置寄存器，中断等待寄存器和中断使能寄存器的访问权限，仅能访问超级用户模式中断阈值寄存器和超级用户模式中断响应/完成寄存器。普通用户模式没有任何 PLIC 寄存器的访问权限。
- 在 S_PER 为 1 时，机器模式拥有所有权限。超级用户模式拥有除 PLIC 权限控制寄存器以外的所有 PLIC 寄存器权限。普通用户模式没有任何 PLIC 寄存器的访问权限。

10.2.9　中断阈值寄存器（PLIC_TH）

每一个中断目标均有一个对应的中断阈值寄存器。仅有优先级大于中断阈值的有效中断才会向中断目标发起中断请求。寄存器读写权限参考权限控制寄存器（PLIC_CTRL）描述。

寄存器位分布和位定义如 图 10.12 所示。

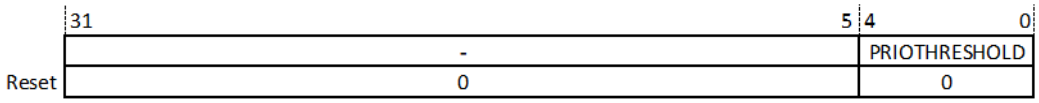


图 10.12: 中断阈值寄存器 (PLIC_TH)

PRIOTHRESHOLD 优先级阈值：

指示当前中断目标的中断阈值。阈值配置为 0，表示允许所有中断。

10.2.10　中断响应/完成寄存器（PLIC_CLAIM）

每一个中断目标均有一个对应的中断响应/完成寄存器。该寄存器在 PLIC 完成仲裁时更新，更新值为 PLIC 本次仲裁结果的中断 ID。寄存器读写权限参考权限控制寄存器（PLIC_CTRL）描述。

寄存器位分布和位定义如 图 10.13 所示。

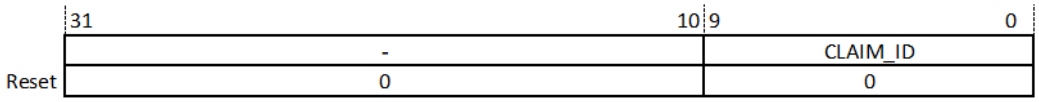


图 10.13: 中断响应/完成寄存器 (PLIC_CLAIM)

CLAIM_ID 中断请求 ID：

对该寄存器的读操作：返回寄存器当前存储的 ID 值。该读操作表示对应 ID 的中断已开始处理。PLIC 开始中断响应处理。

对该寄存器的写操作：表示写入值对应 ID 的中断已完成处理，该写操作不会更新中断响应/完成寄存器。PLIC 开始中断完成处理。

10.3　多核中断

下面简要说明两个常见的多核中断应用场景。

10.3.1　多个核心同时处理外部中断

在 PLIC 的模型下，允许把一个中断源同时映射到多个核心。当该中断源产生中断请求时，它相对于多个核心同时处于 Pending 状态。由于核心运行状态的不同，各个核心会先后响应这个中断并读取 CLAIM 寄存器以获得中断 ID。PLIC 的设计能够保证：只有第一个读取 CLAIM 寄存器的核心能够获得真正的 ID，而其他核心得到的是无效的 ID（即 ID=0），从而不作处理。因此，这个中断只会被处理一次。

将一个中断同时映射到多个核心，可以缩短总体中断响应时间（多个核心中的任何一个都可能处理这个中断），同时多占用一部分处理器资源（得到无效 ID 的核心白白消耗了带宽）。

举例：

假设有两个外部中断源：Source 1 和 Source 2。Source 1 同时映射到 Core 0，Core 1 和 Core 2；Source 2 同时映射到 Core 1，Core 2 和 Core 3。又假设 Source 2 的优先级更高。

- 当只有 Source 1 发生时，它可能被 Core 0，Core 1 和 Core 2 中的任何一个核心处理。
- 当只有 Source 2 发生时，它可能被 Core 1，Core 2 和 Core 3 中的任何一个核心处理。
- 当两个中断同时发生时，Core 1 和 Core 2 处会有优先级仲裁，结果是 Source 2 胜出。因此，Source 2 可能被 Core 1，Core 2 和 Core 3 中的任何一个核心处理。而 Source 1 可能被 Core 0 处理。

10.3.2　核间发送软件中断

在 CLINT 的编程模型中，软件中断有专门的寄存器，分别是：

- M 态软件中断：MSIP0 MSIP1 MSIP2 MSIP3
- S 态软件中断：SSIP0 SSIP1 SSIP2 SSIP3

上述 8 个寄存器的地址对于所有的核心都是统一而且可见的，因此每个核心只要针对上述 8 个寄存器执行写操作，即可实现向任意核心（包括自己）发送软件中断的功能。

11 总线接口

11.1 主设备接口

C908X 的主设备接口支持 AMBA4.0 AXI 协议。请参考 *AMBA 规格说明–AMBA ® AXI™ and ACE™ Protocol Specification*。

11.1.1 主设备接口的特点

AXI 主设备接口负责 C908X 和系统总线之间的地址控制和数据传输，其基本特点包括：

- 支持 AMBA4.0 AXI 总线协议；
- 支持 128/256 位总线宽度；
- 支持系统时钟与 CPU 主时钟的不同频率比；
- 所有的输出信号 flopped out、输入信号 flopped 以获得较好的时序。

11.1.2 主设备接口的 Outstanding 能力

本小节列出了 C908X AXI 主设备接口的 Outstanding 能力，具体如下：

表 11.1: AXI 主设备接口 Outstanding 能力

配置/outstanding 能力	Read Issuing Capability	Write Issuing Capability
多核	每个核心标量最多发出 8 个 Non-cacheable 读请求。	每个核心标量最多发出 12 个写请求。
	CIU 最多可配置对每个矢量核心 (VPU) 承接 8 或者 16 个读请求，不区分 Non-cacheable 和 Cacheable 属性。VPU 同时最多可向 CIU 发送 8 个 cacheable 读请求，根据 CIU 配置，最多可以发送 8 或者 16 个 Non-cacheable 读请求。	CIU 最多可配置对每个矢量核心 (VPU) 承接 8 或者 16 个写请求，不区分 Non-cacheable 和 Cacheable 属性。VPU 同时最多可向 CIU 发送 8 个 cacheable 写请求，根据 CIU 配置，最多可以发送 8 或者 16 个 Non-cacheable 写请求。

续下页

表 11.1 - 接上页

配置/outstanding 能力	Read Issuing Capability	Write Issuing Capability
	全局最多发出 min(8n, 24) 个 Non-cacheable 读请求。全局最多发出 17*m 个 Cacheable 读请求。	全局最多发出 24 个写请求。
单核	全局标量最多发出 8 个 Non-cacheable 读请求。	全局标量最多发出 12 个写请求。
	CIU 最多可配置对矢量核心 (VPU) 承接 8 或者 16 个读请求，不区分 Non-cacheable 和 Cacheable 属性。 VPU 同时最多可向 CIU 发送 8 个 cacheable 读请求，根据 CIU 配置，最多可以发送 8 或者 16 个 Non-cacheable 读请求。	CIU 最多可配置对矢量核心 (VPU) 承接 8 或者 16 个写请求，不区分 Non-cacheable 和 Cacheable 属性。 VPU 同时最多可向 CIU 发送 8 个 cacheable 写请求，根据 CIU 配置，最多可以发送 8 或者 16 个 Non-cacheable 写请求。
	全局最多发出 17*m 个 Cacheable 读请求。	全局最多发出 16 个写请求。

备注

表 11.1 中的“m”指 L2 Cache slice 的数量。当核心数量大于 2 或 L2 Cache size 大于 1M 时，L2 Cache slice 的数量为 2，否则 L2 Cache slice 的数量为 1。“n”指 cluster 中的核心数量。

表 11.2: AXI 主设备接口 ARID 编码

配置/ 读请求类型及 ID		多核	单核
Device Non-Cacheable (不包含 Exclusive Device Non-Cacheable 请求)	{1'b0, 4'b (coreid), 3'b000}	每个 ID 的 Outstanding 为 8，所有该类请求 Outstanding 最大为 min(8n, 24)	每个 ID 的 Outstanding 为 8，所有该类请求 Outstanding 最大为 8
Exclusive Non-Cacheable	{1'b0, 4'b (coreid), 3'b??1}	每个 ID 无 Outstanding，所有该类请求 Outstanding 最大为 n	每个 ID 无 Outstanding，所有该类请求 Outstanding 最大为 1
Normal Non-Cacheable 请求（不包含 Exclusive 请求）	{3'b100, 5'b?????}	每个 ID 无 Outstanding，所有该类请求 Outstanding 最大为 min(8n, 24)	每个 ID 无 Outstanding，所有该类请求 Outstanding 最大为 8
Cacheable 请求	{2'b11, 6'b??????}	每个 ID 无 Outstanding，所有该类请求 Outstanding 最大为 17*m	每个 ID 无 Outstanding，所有该类请求 Outstanding 最大为 17*m

备注

表 11.2 中 “?” 表示该位可为 1 或 0 。“m” 指 L2 Cache slice 的数量，当核心数量大于 2 或 L2 Cache size 大于 1M 时，L2 Cache slice 的数量为 2，否则 L2 Cache slice 的数量为 1。“n” 指 cluster 中的核心数量。

表 11.3: AXI 主设备接口 AWID 编码

配置/写请求类型及 ID		多核	单核
Device Non-Cacheable (不包含 Exclusive Device Non-Cacheable 请求)	{1’b0, 4’b (coreid), 3’b000}	每个 ID 的 Outstanding 为 12, 所有该类请求 Outstanding 最大为 24	每个 ID 的 Outstanding 为 12, 所有该类请求 Outstanding 最大为 12
Exclusive Non-Cacheable	{1’b0, 4’b (coreid), 3’b??1}	每个 ID 无 Outstanding 所有该类请求 Outstanding 最大为 n	每个 ID 无 Outstanding ，所有该类请求 Outstanding 最大为 1
Normal Non-Cacheable 请求 （不包含 Exclusive 请求）	{3’b100, 5’b?????}	每个 ID 无 Outstanding ，所有该类请求 Outstanding 最大为 24	每个 ID 无 Outstanding ，所有该类请求 Outstanding 最大为 12
Cacheable 请求	{3’b111, 5’b?????}	每个 ID 无 Outstanding ，所有该类请求 Outstanding 最大为 24	每个 ID 无 Outstanding ，所有该类请求 Outstanding 最大为 12

备注

- 表 11.3 中 “?” 表示该位可为 1 或 0 。“m” 指 L2 Cache slice 的数量，当核心数量大于 2 或 L2 Cache size 大于 1M 时，L2 Cache slice 的数量为 2，否则 L2 Cache slice 的数量为 1。“n” 指 cluster 中的核心数量。
- 上述 ARID/AWID 的编码可能随着处理器版本的演进而发生变化，因此，SoC 的集成不应该依赖于特定的 ID 值，而应该遵从 AXI 协议的通用规则。

11.1.3 支持的传输类型

主设备接口支持的传输特性如下：

- BURST 支持 INCR 传输，其它突发类型均不支持；
- 在数据宽度为 128 bits 配置下，LEN 支持传输长度为 1、4，其他传输长度均不支持；在数据宽度为 256 bits 配置下，LEN 支持传输长度为 1、2，其他传输长度均不支持；

- 支持独占式访问；
- 在数据宽度为 128 bits 配置下，SIZE 支持四字、双字、字、半字和字节传输，其它传输大小不支持；在数据宽度为 256 bits 配置下，SIZE 支持八字、四字、双字、字、半字和字节传输，其它传输大小不支持；
- 支持读和写操作。

注意：C908X 的主设备接口只实现了全部 AXI 传输的一个子集。但是，SoC 的集成不应该依赖于特定的传输类型，而应该遵从 AXI 协议的通用规则。

11.1.4　支持的响应类型

主设备接口接收从设备的响应类型为：

- OKAY
- EXOKAY
- SLVERR
- DECERR

11.1.5　不同总线响应下的行为

总线上出现不同总线响应时 CPU 的行为如 表 11.4 。

表 11.4: 总线异常处理

RRESP/BRESP	结果
OKAY	普通传输访问成功，或 exclusive 传输访问失败；读传输 exclusive 访问失败代表总线不支持 exclusive 传输，产生访问错误异常，写传输 exclusive 访问失败仅代表抢锁失败，不会返回异常
EXOKAY	exclusive 访问成功；
SLVERR/DECERR	访问出错，读传输产生访问错误异常，写传输访问出现错误时置起中断。

C908X 支持的 DVM message type 包括：

- TLB Invalidate
- Physical Instruction Cache Invalidate
- Synchronization

对应 ARADDR 如下 表 11.5 所示：

表 11.5: TLB Invalidate First Part

ARADDR bits	Name	Function
[43][1]	Reserved	1'b0
[42:40][1]	Virtual Address	VA[47:45]
[39:32]	ASID upper byte	ASID[15:8]
[31:24]	ZoneID	ZoneID[7:0]
[23:16]	ASID lower byte	ASID[7:0]
[15]	Completion	1'b0: DVM Completion is not required
[14:12]	Message type	3'b0: TLB Invalidate
[11:10]	Exception Level	2'b10: Applies to Guest OS
[9:8]	Security	2'b11: Applies to Non-secure only
[7]	SBZ or Range	1'b0: Message does not include address range information
[6： 5]	VA or ASID valid	2'b00: Invalidate all TLB entries， ADDR[0] must be 0. 2'b01: Reserved 2'b10: Invalidate TLB entries by VA， ADDR[0] must be 1. 2'b11: - ADDR[0]=1'b0, Invalidate TLB entries by ASID - ADDR[0]=1'b1, Invalidate TLB entries by ASID and VA.
[4]	Leaf	1'b0: Invalidate all associated translations.
[3:2]	Stage	2'b0: Stage 1 and Stage 2 invalidation
[1]	Reserved	1'b0
[0]	Addr	1'b0: The message does not require an address and has one part 1'b1: The message includes an address and has two parts

1: Present when Sv48 selected.

表 11.6: TLB Invalidate Second Part

ARADDR bits	Description
[43:40][1]	VA[44:41]
[39:4]	VA[39:4]
[3]	Sv48: VA[40] Sv39: 1'b0
[2:0]	3'b0

1: Present when Sv48 selected.

表 11.7: Physical Instruction Cache Invalidate First Part

ARADDR bits	Name	Function
[43:40][1]	Reserved	4'b0
[39:16]	Reserved	24'b0
[15]	Completion	1'b0: DVM Completion is not required
[14:12]	Message type	3'b010: Physical Instruction Cache Invalidate
[11:10]	Exception Level	2'b00: Applies to Hypervisor and all Guest OS
[9:8]	Security	2'b11: Applies to Non-secure only
[7]	SBZ or Range	1'b0: Message does not include address range information
[6:5]	VA or ASID valid	2'b00: Invalidate all cache lines/ Invalidate cache line by PA.
[4:1]	Reserved	4'b0
[0]	Addr	1'b0: The message does not require an address and has one part 1'b1: The message includes an address and has two parts

1:Present when Sv48 selected.

表 11.8: Physical Instruction Cache Invalidate Second Part

ARADDR bits	Description
[43:40][1]	4'b0
[39:4]	PA[39:4]
[3:0]	4'b0

1:Present when Sv48 selected.

表 11.9: Synchronization

ARADDR bits	Name	Function
[43:40][1]	Reserved	4'b0
[39:16]	Reserved	24'b0
[15]	Completion	1'b1: Completion required
[14:12]	Message type	3'b100: Synchronization
[11:10]	Exception Level	2'b00: Applies to Hypervisor and all Guest OS
[9:8]	Security	2'b00: Applies to Secure and Non-secure
[7]	SBZ or Range	1'b0: Message does not include address range information
[6:5]	VA or ASID valid	2'b00
[4:1]	Reserved	4'b0
[0]	Addr	1'b0: The message does not require an address and has one part

1: Present when Sv48 selected.

11.2　设备一致性接口

C908X 的设备一致性接口（DCP，Device Coherence Port）是一个用户可配置的接口，用来完成外设对 L2 Cache 和 L1 D-Cache 的访问，维护外设和处理器片上数据的一致性。设备一致性接口支持 AMBA AXI4 协议，具体请参考 *AMBA 规格说明–AMBA® AXI™ and ACE™ Protocol Specification*。

11.2.1　设备一致性接口的特点

设备一致性接口的基本特点包括：

- 支持 AMBA4.0 AXI 总线协议；
- 支持 128 位总线宽度；
- 支持系统时钟与 CPU 主时钟的不同频率比；
- 所有的输出信号 flopped out、输入信号 flopped in 以获得较好的时序；
- 读写分别最多支持 8 个并发的传输。

注意：多个并发传输之间要求 AxID 不相同。

11.2.2　支持的传输类型

设备一致性接口支持的传输特性如下：

11.2.2.1　读请求

对于 Non-Cacheable 的访问

- 访问 Device 时 ARSIZE 支持 3'b000 至 3'b011
- 访问 Normal 时 ARSIZE 仅支持 3'b100
- 访问 Device 时 ARLEN 支持 0
- 访问 Normal 时 ARLEN 支持 0/1/3
- 访问 Device 时 ARCACHE 支持 4'b0000/4'b0001
- 访问 Normal 时 ARBURST 支持 INCR、WRAP
- 访问 Device 时 ARBURST 仅支持 INCR
- ARADDR 应与 ARSIZE 对齐

对于 Cacheable 的访问

- ARSIZE 仅支持 3'b100
- ARLEN 支持 0/1/3

- ARCACHE 支持 4'b1011/4'b1111/4'b0111
- ARBURST 支持 INCR/WRAP
- ARADDR 应与 ARLEN、ARBURST 对齐

11.2.2.2 写请求

对 Non-Cacheable 的访问

- 访问 Device 时 AWSIZE 支持 3'b000 至 3'b011
- 访问 Normal 时 AWSIZE 仅支持 3'b100
- 访问 Device 时 AWLEN 支持 0
- 访问 Normal 时 AWLEN 支持 0/1/3
- 访问 Device 时 AWLEN 仅支持 0
- 访问 Device 时 AWCACHE 支持 4'b0000/4'b0001
- 访问 Normal 时 AWBURST 支持 INCR、WRAP
- 访问 Device 时 AWBURST 仅支持 INCR
- AWADDR 应与 AWSIZE 对齐

对 Cacheable 的访问

- AWSIZE 仅支持 3'b100
- AWLEN 支持 0/1/3
- AWCACHE 支持 4'b1011/4'b1111/4'b0111
- AWBURST 支持 INCR/WRAP
- AWADDR 应与 AWLEN、AWBURST 对齐

备注

- 读写传输 AxID 均为 16 bits
- 写传输均支持 Wstrb 任意字节有效
- 所有相同 ID 的请求不支持 OutStanding
- 不同 ID 的读写请求 OutStanding 均为 8
- 不支持独占式传输
- 不支持 CMO 请求
- 不支持原子操作
- 不支持 Barrier 请求

11.2.3 不同传输下的 L2 分配行为

对于读请求，若 ARCACHE[3:0]=4'b1111 或 4'b0111，当在读请求触发时该数据所在的 cache line 既不在 L1 DATA CACHE 中，也不在 L2 CACHE 中，则在读请求完成后该 cache line 会分配到 L2 中。该分配行为可能会触发 L2 替换；若 ARCACHE[3:0]=4'b1011，则不会对 L1 和 L2 产生任何影响。

对于写请求，若 AWCACHE[3:0]=4'b1111 或 4'b1011，则在写请求触发时无论该数据所在的 cache line 是否在 L2 CACHE 中，则在写请求完成后，更新后的 cache line 数据会分配到 L2 中。该分配行为可能会触发 L2 替换；若 AWCACHE[3:0]=4'b0111，当在写请求触发时该数据所在的 cache line 在 L2 CACHE 中，则写请求会更新该 cache line 的数据并写回到 L2，否则会直接通过外部总线写出。

11.2.4 支持的响应类型

设备一致性接口发出的响应类型为：

- OKAY；
- SLVERR。

11.2.5 不同行为发出的响应

从设备接口返回的响应类型如 表 11.10 所示。

表 11.10: 从设备响应类型

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

11.3 矢量专用总线接口

C908X 矢量专用总线接口是一个用户可配置 AXI4.0 的 master 接口，支持 AMBA® AXI™ Protocol Specification G 版本协议，用户可以外接 SRAM。

11.3.1 矢量专用总线接口的特点

总线接口的基本特点包括：

- 支持 AXI4.0 协议；
- 数据总线位宽 512/1024/4096 bits，地址总线位宽 40/48 bits；
- 最大支持 32 个读 outstanding 和 16 个写 outstanding；支持总线接口整数倍倍频 1:N(N <= 8)
- 支持所有总线响应
- 支持非对齐访问

11.3.2　矢量专用总线接口的 outstanding 能力

本小节列出了 C908X 矢量专用总线接口的 outstanding 能力，具体如下：

表 11.11: 矢量专用总线接口的 outstanding 能力

参数	数值	说明
Read Issuing Capability	32	最多 32 个读请求
Write Issuing Capability	32	最多 32 个写请求

表 11.12: 矢量专用总线接口 AXI ARID 编码

ARID[7:0]	适用场景	每个 ID 的 Outstanding
8'h80	适用于所有场景	最多 32 个读请求

表 11.13: 矢量专用总线接口 AXI AWID 编码

AWID[7:0]	适用场景	每个 ID 的 Outstanding
{4'b1000, 4'b0}	适用于所有场景	最多 32 个写请求

备注

注意：上述 ARID/AWID 的编码可能随着处理器版本的演进而发生变化，因此，SoC 的集成不应该依赖于特定的 ID 值，而应该遵从 AXI 协议的通用规则。

11.3.3　支持的传输类型

矢量专用总线接口支持的传输特性如下：

- AR request

（1）outstanding 能力为 32：最多有 32 个未回 R 的 AR 请求在外飞行，和 lane 的数据没有任何联系

（2）burst_type 为 incr length 可以为 0 ~ 8

（3）一笔请求不会跨 4KB

（4）addr 的低 [6:0] bits 不一定是全 0，可以为非对齐地址

（5）支持 INCR 传输

（6）仅支持 Weak-Order、Non-Cacheable 属性访问

（7）ARSize 支持为 3'b111（vlen = 1024）

- AW request

（1）outstanding 能力为 32，最多有 32 个未回 Bresp 的 AW W 请求在外飞行，和 lane 的数据没有任何联系

（2）写请求会有 wstrb 不是全 1

（3）写请求的 addr 的低 [6:0] bits 不一定是全 0，可以为非对齐地址

（4）burst_type 为 single

（5）支持单笔传输

（6）仅支持 Weak-Order、Non-Cacheable 属性访问

（7）AWSize 支持为 3'b111（vlen = 1024）

备注

- 对下游 slave 希望 AR 和 AW 通道是双通道独立。
- 假如 AR 和 AW 共用一个 outstanding buf 的话，AW 通道保持至少一个 outstanding 能力。
- 总线接口只实现了全部 AXI 传输的一个子集。但是，SoC 的集成不应该依赖于特定的传输类型，而应该遵从 AXI 协议的通用规则。

11.3.4　支持的响应类型

矢量专用总线接口支持的响应类型为：

- OKAY
- EXOKAY
- SLVERR
- DECERR

11.4　低延时外设接口

C908X 的低延时外设接口（LLP，Low Latency Port）是一个用户可配置的 master 接口，可以用来访问系统外设。低延时外设接口支持 AMBA AXI4 协议（具体请参考 *AMBA 规格说明–AMBA® AXI™ and ACE™ Protocol Specification*）。

11.4.1　低延时外设接口的特点

低延时外设接口的基本特点包括：

- 支持 AXI4.0 协议；
- 数据总线位宽 128 bits，地址总线位宽 40 bits；
- 支持每个核心 8 个读 outstanding 和 12 个写 outstanding；全局最多发出 min(8n, 24) 个读 outstanding 和最多 min(12n, 24) 个写 outstanding；

- 支持 CPU 和 LLP 的整数倍倍频 1:N（N <= 8）；
- 支持所有总线响应；
- 支持非对齐访问。

11.4.2　低延时外设接口的 Outstanding 能力

本小节列出了 C908X 低延时外设接口的 Outstanding 能力，具体如下：

表 11.14: 低延时外设接口 Outstanding 能力

参数	数值	说明
Read Issuing Capability	8n，n= 核心数量	每个核心最多发出 8 个，全局最多发出 min(8n, 24) 个读请求。
Write Issuing Capability	12n，n= 核心数量	每个核心最多发出 12 个，全局最多发出 min(12n, 24) 个写请求。

表 11.15: AXI LLP ARID 编码

ARID	请求类型	每个 ID 的 Outstanding
{1'b0, 4'b(coreid), 3'b000}	Device Non-Cacheable （不包含 DCP 请求和 Exclusive Device Non-Cacheable 请求）	每个 ID 的 Outstanding 为 8 所有该类请求 Outstanding 最大为 min(8n, 24)
{1'b0, 4'b(coreid), 3'b??1}	Exclusive Non-Cacheable	每个 ID 无 Outstanding 所有该类请求 Outstanding 最大为 min(n, 4)
{4'b100?, 4'b????}	Normal Non-Cacheable 请求 （包含所有 DCP Non-Cacheable 请求）	每个 ID 无 Outstanding 所有该类请求 Outstanding 最大为 min(8n, 24)

备注

- 表 11.15 中“n”指 cluster 中的核心数量。
- AR 通道同一个 ID 不会同时在主设备接口与 LLP 出现
- 上述 ARID/AWID 的编码可能随着处理器版本的演进而发生变化，因此，SoC 的集成不应该依赖于特定的 ID 值，而应该遵从 AXI 协议的通用规则。

表 11.16: AXI LLP AWID 编码

AWID	请求类型	每个 ID 的 Outstanding
{1'b0, 4'b(coreid), 3'b000}	Device Non-Cacheable（不包含 DCP 请求和 Exclusive Device Non-Cacheable 请求）	每个 ID 的 Outstanding 为 11 所有该类请求 Outstanding 最大为 min(11n, 24)
{1'b0, 4'b(coreid), 3'b??1}	Exclusive Non-Cacheable 请求	每个 ID 无 Outstanding 所有该类请求 Outstanding 最大为 min(n, 4)
{3'b100, 5'b?????}	Normal Non-Cacheable 请求（不包含 Exclusive 请求，包含 DCP Device Non-Cacheable 请求）	每个 ID 无 Outstanding 所有该类请求 Outstanding 最大为 min(12n, 24)

备注

- 表 11.16 中 "n" 指 cluster 中的核心数量。
- AW 通道同一个 ID 不会同时在主设备接口与 LLP 出现
- 上述 ARID/AWID 的编码可能随着处理器版本的演进而发生变化，因此，SoC 的集成不应该依赖于特定的 ID 值，而应该遵从 AXI 协议的通用规则。

11.4.3 支持的传输类型

低延时外设接口支持的传输特性如下：

- 仅支持 INCR 传输，不支持 FIXED 和 WRAP；
- LEN 仅支持 8'b0；
- 支持独占式访问；
- 支持 noncacheable normal memory 和 device 的访问；
- 支持非对齐访问；
- Size 支持 3'b000~3'b100(1B~16B)；
- 对于 normal memory non-cacheable 的传输，支持写 merge，wstrb 可以发出任意值，Axsize 固定为 3'b100；
- 对于 device 的传输，Axsize 可能为 3'b000~3'b011；

注意：C908X 的低延时外设接口只实现了全部 AXI 传输的一个子集。但是，SoC 的集成不应该依赖于特定的传输类型，而应该遵从 AXI 协议的通用规则。

11.4.4 支持的响应类型

低延时外设接口支持的响应类型为：

- OKAY
- EXOKAY
- SLVERR
- DECERR

12 协处理器接口

12.1　概述

C908X 支持的协处理器接口，用于增强用户 DSA（Domain Specific Accelerator）需求，加速特定应用执行，并支持用户进行自定义指令扩展执行，用户可将自定义编码的指令转发至协处理器工作，来规避 CPU 部分低效场景，获得 SoC 的整体能效提升。C908X 支持和多个协处理器协同工作，以提升对多种操作类型的复杂场景的加速能力。通过自定义指令扩展，既保证了编程的易用性，又保证了对协处理器的高效控制。C908X 用户自扩展协处理器指令集需要在机器模式扩展状态寄存器（MXSTATUS）中打开用户自扩展协处理器指令集使能位（COPINSTEE）才能正常使用，否则将产生非法指令异常。

备注

C908X 只在 RV64 模式下支持协处理器指令。

C908X 协处理器结构图如下 图 12.1 所示：

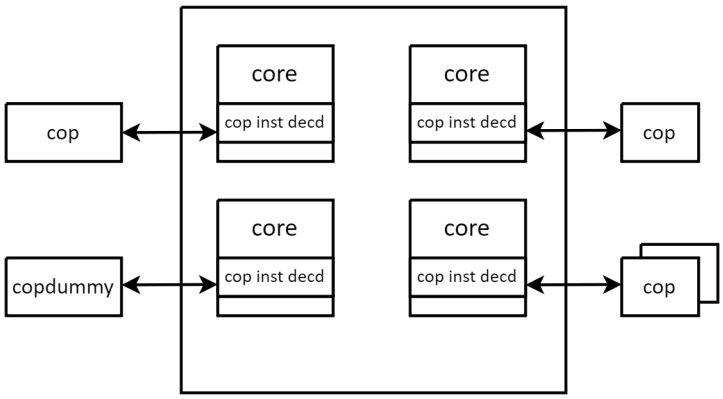


图 12.1: C908X 协处理器结构示意图

C908X 协处理器基本特性如下：

- 支持自定义拓展指令（custom0-custom3），将特殊场景的指令转发至协处理器工作；
- 开放自定义译码接口；
- 开放 GPR、FPR 信息，可用于自定义指令的数据源，并支持自定义指令的回写（也可不回写，取决于功能定义）；

- 支持快速按序通过接口发送自定义指令信息至协处理器；
- 支持协处理器快速进行数据回传并对目的寄存器进行回写；
- 支持配置最多 32 个协处理器扩展。

12.2　C908X 协处理器指令扩展

12.2.1　协处理器指令接口特性

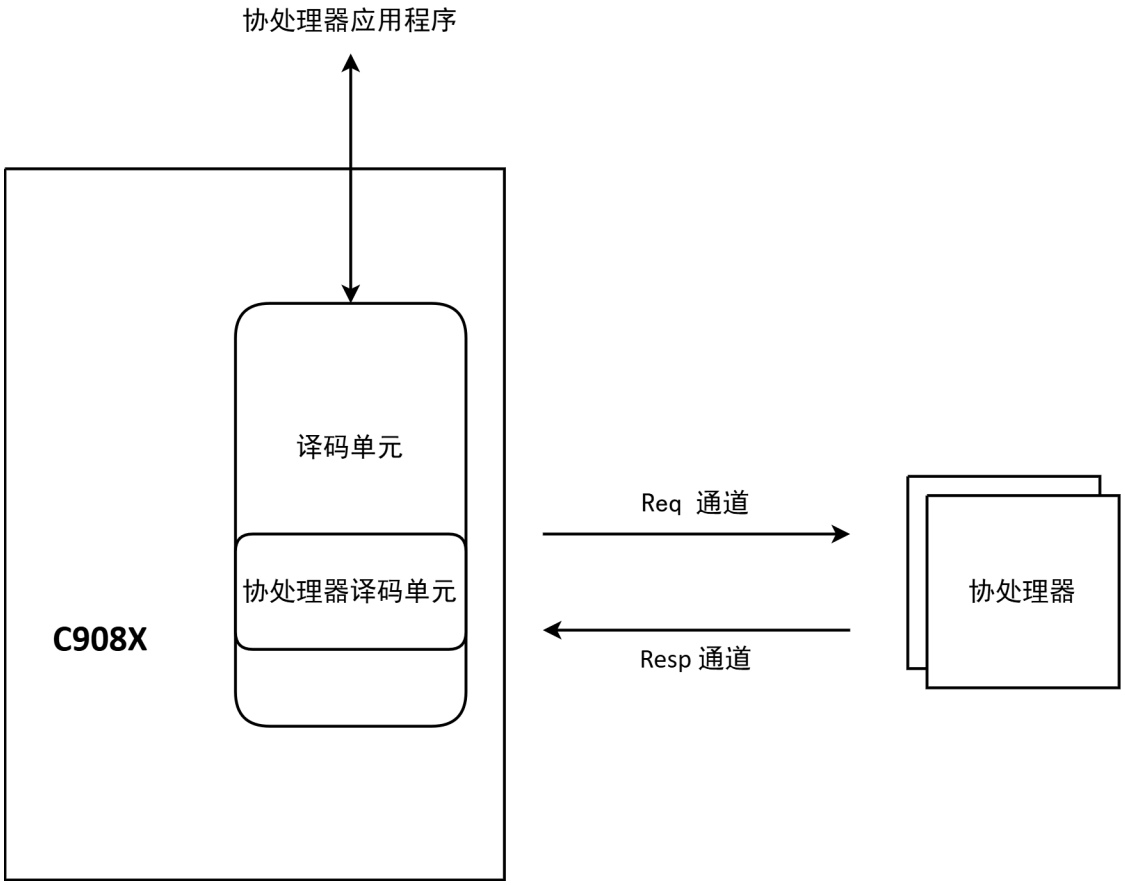


图 12.2: C908X 协处理器指令接口示意图

玄铁协处理器扩展指令支持使用 custom 域进行指令扩展，支持 scalar 指令功能扩展。具体功能由用户自定义，硬件上会将 opcode 通过译码接口导出，用户按照一定格式将指令进行译码，将需要的操作数和控制信息回传至处理器，处理器在解决数据和控制相关性之后将所需信息通过 Req 通道发送给指定协处理器，在得到运算结果后通过 resp 通道返回给处理器。

C908X 协处理器指令在确认不在错误路径之后才会按序发送给协处理器，协处理器收到的均为必须执行的指令。使用 valid/ready 握手方式通信，req 需要等待接收方返回 ready 才可以确认发送成功。

协处理器支持工具链快速适配，用户按照特定格式描述指令功能，编译工具会根据描述文件自动生成适配当前自定义指令功能的工具链，方便用户快速实现自己的协处理器功能软件开发。

12.2.2　协处理器扩展指令支持

用户可以通过使用 custom0 ~ custom3 域自定义扩展指令，可定义的指令类型包括 scalar 类型（其中 scalar 类型包括 scalar 整型和 scalar 浮点类型指令）。支持将 gpr、fpr gpr 作为指令操作数，并通过协处理器接口发送至协处理器。支持自定义协处理器个数，用户可以在编码空间划定几位用来表示该指令应该发给哪一个协处理器。协处理器 scalar 整型和浮点扩展指令最多支持 3 个寄存器作为 source 数据发送给协处理器。

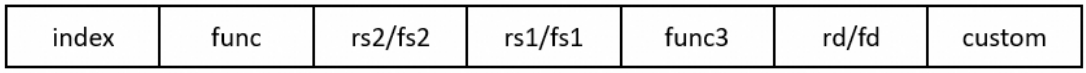


图 12.3: C908X 协处理器指令编码示例

图 12.3 为一个编码示例，其中 index 代表发向哪一个协处理器，func 用于识别不同指令，rs1、rs2、rd 可以根据不同指令功能当做 gpr、fpr 使用，部分编码空间也可以作为立即数使用。用户可以自由使用编码空间，只需要将所编码里的寄存器信息给出来即可，index 约定最大为 5 bit，最多可以控制 32 个协处理器。

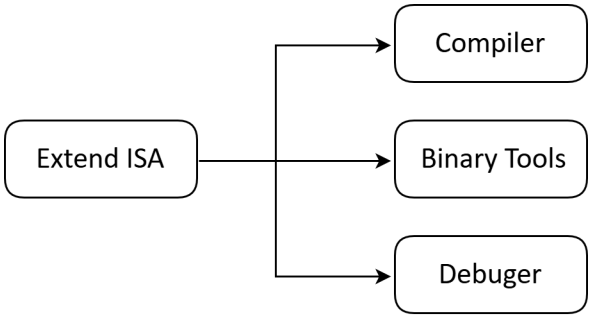


图 12.4: C908X 协处理器工具支持

用户可以通过快速指令功能描述接口对自定义指令功能进行定义，编译工具会自动化识别指令功能描述，生成指令 intrinsic、汇编器、反汇编器、调试器，轻松获得专属工具链支持。

12.2.3　协处理器指令译码

C908X 支持对用户协处理器支持的自定义指令进行译码，用户可在 pc_idu_id_cop_decd.v 文件内，对扩展的自定义指令进行自定义译码。

用户使用自扩展指令时，需要注意下述事项：

1. 满足 RISC-V 对 custom 指令编码的要求，在 custom 域内进行编码，并不能和玄铁扩展指令重合。其中，custom 域的编码要求详见 RISC-V Instruction Set Manual Volume I: Unprivileged Architecture --- *RV32/64G Instruction Set Listings* ，玄铁扩展指令编码详见*附录 B-12 玄铁协处理器扩展指令术语* 。
2. 若用户自扩展指令存在第三个源操作数，需要将第三个源操作数写入 rd 域中。

协处理器译码模块（pc_idu_id_cop_decd）在接口上需要暴露下列信息：

- 源操作数和目的操作数索引

- 指令非法信息标记
- 访问协处理器的 index 编号
- 浮点/向量类型指令标记

当 C908X 确定某条协处理器指令为非投机指令后，才会将指令发送给协处理器，并支持快速退休，不会等待协处理器对指令的接受响应。

同时，协处理器在设计时，需要保证其在处理需要回写的指令时，会对需要回写的寄存器进行回写，否则 C908X 会因为将所有处理器寄存器资源分配给协处理器导致寄存器资源耗尽，从而导致 C908X 卡死。

12.3　协处理器接口所适用的互联结构场景

该接口主要用于处理多核系统与多协处理器间的互联问题，可能的场景包括单核与单协处理器的互联，单核与多协处理器互联，多核共享单个协处理器，多核共享多个协处理器等等。主要的使用场景如下 图 12.5 。

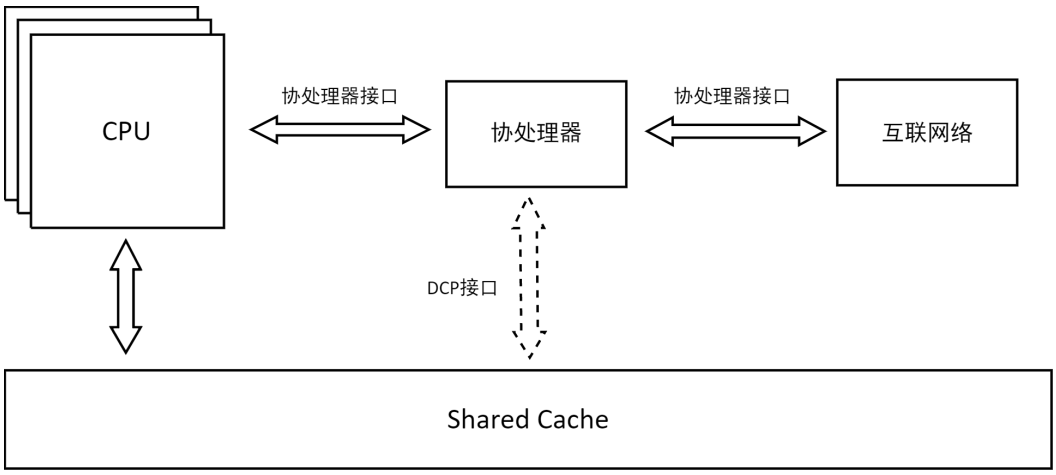


图 12.5: C908X 协处理器接口互联场景

用户可以通过 CPU 直接和协处理器互联，也可以通过互联网络将命令发送至协处理器，协处理器和 CPU 可以连接到同一个 shared cache，进行数据同步，加速数据交互场景。

对于通过互联网络连接协处理器，由于 CPU 做的乱序能力不是特别强，当发送指令均需要回写结果时，可能会拖累 CPU 的工作速度，因为当后续指令和前序存在相关性时会让后续指令持续等待。当发送指令不需要回写时，可以持续发送控制指令。

13 调试

13.1 Debug 单元的功能

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。C908X 兼容 RISC-V Debug V0.13.2 协议标准。对外调试接口支持两种模式：2 线 cJTAG 和 5 线 JTAG（标准 JTAG5）。

调试接口的主要特性如下：

- 支持 2 线 cJTAG 和 5 线 JTAG 两种模式；
- 支持多 cluster 调试；
- 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；
- 支持通过 abstract command 检查和设置 CPU 寄存器（仅支持控制状态寄存器 CSR 和通用寄存器 GPR）的值
- 不支持通过 abstract command 检查和改变内存值
- 支持通过 program buffer/system bus 检查和改变内存值
- 可进行指令单步执行或多步执行；
- 快速下载程序；
- 可在 CPU 复位之后进入调试模式。

C908X 的调试工作是由调试软件、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 13.1 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。

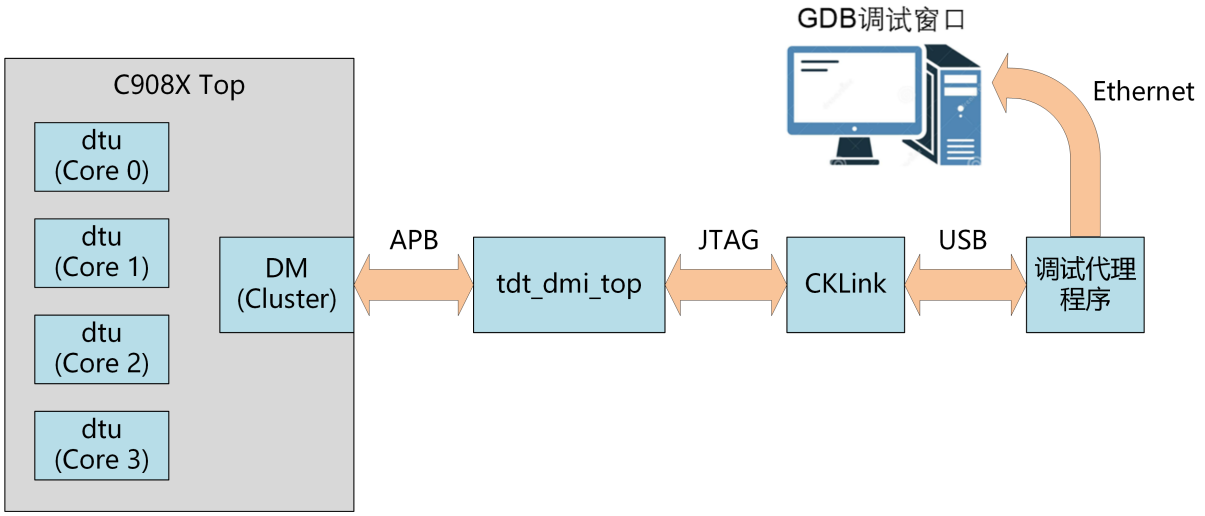


图 13.1: 调试接口在整个 CPU 调试环境中的位置

若客户选择配置了 SBA(System Bus Access) 功能, 那么 cluster 侧会多出一组 AXI 总线，用于调试工具绕过 CPU 直接进行内存访问。该总线特性如下：

- 支持 narrow 传输，最小 size 为 32 bit；
- burst 类型仅 increment 类型，且 length 固定为 0；
- resp 只支持 OKAY 和 SLVERR；
- 读写 outstanding 能力是 1；
- 不支持乱序；
- 不支持 interleaving；
- 不支持非对齐访问。

13.2　调试资源的配置

为了方便用户选择，C908X 提供了三种调试资源配置组合供用户选择：

- 最小配置：1 个硬断点；
- 典型配置：3 个硬断点；
- 最大配置：8 个硬断点且可以组成触发链。

RISC-V Debug 协议中定义了多功能的 trigger 用于实现指令断点（breakpoint）和数据断点（watchpoint），trigger 可以配置为 4 种类型：

- 指令地址类型：匹配指令地址，即 PC。此类似与传统的 breakpoint 功能相同。
- 指令数据类型：匹配指令码
- 访存地址类型：匹配访存指令的访存地址。此类型与传统的 watchpoint 功能相同
- 访存数据类型：匹配访存指令的访存数据。

RISC-V Debug 协议还定义 6 种匹配模式，匹配模式与上述类型是正交的：

1. 全等匹配：当 cpu 实际值与 trigger 设置值相等时，trigger 触发
2. 低位掩码匹配：可以设置不比较低位，当 cpu 实际值与 trigger 设置值相等时，trigger 触发。
3. 大于等于：当 cpu 实际值大于等于 trigger 设置值时，trigger 触发
4. 小于: 当 cpu 实际值小于 trigger 设置值时，trigger 触发。
5. 掩码匹配低位：把 trigger 设置值分成高一半 [63:32] 和低一半 [31:0]，用 trigger 设置值的高一半做掩码，trigger 设置值的低一半做模板，当满足 trigger 设置值 [31:0] = cpu 实际值 bit[31:0] & trigger 设置值 [63:32] 时，trigger 触发。
6. 掩码匹配高位：把 trigger 设置值分成高一半 [63:32] 和低一半 [31:0]，用 trigger 设置值的高一半做掩码，trigger 设置值的低一半做模板，当满足 trigger 设置值 [31:0] = cpu 实际值 bit[63:32] & trigger 设置值 [63:32] 时，trigger 触发。

详细描述可参考 RISC-V External Debug Support V0.13.2 当中 5.2.9 match control 小节。

除上述描述外，每一种配置组合都支持软断点，抽象命令寄存器，异步进调试，复位后进调试，指令单步等调试资源和方法。

13.3 C908X VPU 访存类指令 Debug 功能说明

13.3.1 Trigger 功能支持与限制

1. **Slow 模式限制**

   当触发条件（Trigger）启用时，C908X VPU 将进入 Slow 模式，此时仅允许单条访存指令处于执行流水线中。

2. **Data Trigger 支持**

   (1) 数据匹配格式：仅支持对 NaN（Not a Number）类数据的触发。

   (2) 配置要求

      i. 需将 tdata2 设置为 NaN 的最小值：0x7_FF00_0000_0000_0001。

      ii. 使用等于（Equal）匹配模式（mcontrol.match = 0）。

   (3) 异常优先级

      若 Store 指令的某个 segment 同时触发异常和 NaN Trigger，cpu 将优先处理异常事件，NaN Trigger 在此场景下不会触发。

3. **Address Trigger 支持**

   (1) 匹配模式：仅支持模糊地址匹配（Any Size）。

   (2) 配置要求：

      i. mcontrol.sizelo 与 mcontrol.sizehi 必须配置为 0，否则无法命中 Trigger。

      ii. 若需地址匹配生效，需确保 tdata1 中 size 字段配置为 0（即 Any Size 模式）。

(3) 匹配规则：

当设置的 tdata2 的值位于 C908X VPU 访存指令地址的 [va, va + segment inst size*) 内且指令类型与预设类型一致时，则触发命中。

备注

* segment inst size： 即 segment 的访存范围。

4. **Timing 模式限制**

仅支持 Timing1 模式。

注意

若用户错误配置为 Timing0 模式，且某条 C908X VPU 向量访存指令命中 Trigger，其行为将按 Timing1 模式执行，导致 CPU 状态不可预期。

5. **其他限制**

(1) 匹配模式：仅支持 match=0（Equal 模式）。

(2) Chain 功能：不支持 Trigger 链式触发（Chain）。

(3) mtval 更新：在 mcontrol 触发的断点异常中，mtval 寄存器将被强制清零。

备注

- 寄存器 mcontrol 详细描述可参考 RISC-V External Debug Support V0.13.2 。
- NaN 格式的判定参考以下三个文件：
  - 《RISC-V BF16 Extensions》，Version 1.0, 05 July 2024: Ratified
  - 《OCP 8-bit Floating Point Specification (OFP8)》，Revision 1.0
  - 《IEEE Standard for Floating-Point Arithmetic》，IEEE Std 754-2019

13.4 同步调试与异步调试

13.4.1 同步调试

调试器向 dmcontrol 寄存器中的 haltreq 域写 1 进入调试模式。

13.4.2 异步调试

调试器向 DM 模块中的 custom command 寄存器（属于玄铁调试系统自定义寄存器）的 type 域写 0 时，表示 debugger 向所选的核发出异步调试请求，则 DM 会向核内发出拉高的异步调试请求信号。与同步调试不同，当核内 RTU 模块接收到异步调试请求后，会立即进入调试模式。该功能为玄铁处理器调试系统扩展

功能。调试器发起同步调试请求后，会等当前指令退休后再进入调试模式。当 CPU 卡死（当前指令一直无法退休时），就无法响应同步调试请求，调试器等待几秒发现同步调试请求无法被响应时，就会自动发起异步调试请求，尝试在 CPU 卡死时也能让 CPU 进入调试模式。关于上述调试相关寄存器，请参考 RISC-V 官方文档 RISC-V External Debug Support V0.13.2 。

14 Trace

14.1 简介

Trace 系统用于在 CPU 正常运行时记录程序执行路径和数据读写信息，压缩成特定格式数据流，通过专用的数据通道和输出端口传输至调试主机，以供调试者了解 CPU 及程序的运行情况。与 debug 调试不同的是，trace 调试属于非侵入式调试，无需进入调试模式，可以在 CPU 正常运行时获取信息，在调试时间敏感的问题时有较大优势。C908X Trace 系统采用 RISC-V Nexus trace 方案，Trace message 采用《The Nexus 5001 Forum™ Standard for a Global Embedded Processor Debug Interface》中定义的 message 格式。

14.1.1 trace 功能点

trace 功能旨在通过输出的 trace 数据包解析 cpu 执行的指令流与执行状态信息，trace 包中输出的信息包括：

- srcid：表示 msg 产生的来源，可理解为 core id
- timestamp：32bit 时间戳，用来指示 msg 产生的时间
- u-addr/f-addr：pc 全地址或 pc 地址的异或值，可以解析出该条 msg 产生时最新一条退休或待退休指令的 pc
- icnt：instruction counter，以半字为单位对指令计数，可配合 addr 字段使用，统计出当前执行到的 pc
- b-type：指令流发生间接跳转的类型（expt/int/间接跳转指令）
- hist：条件跳转指令的 taken/non-taken 的状态信息，当配置为压缩模式时，对于条件跳转指令，不输出 direct branch msg，跳转信息在 hist 字段中记录。

14.1.2 trace 的开启与关闭

- 支持通过配置 trace 寄存器开启与关闭 trace，与 pc 不强绑定。
- 支持通过 trigger 开启与关闭 trace，例如可以设置当执行到某个 pc 值时开启或关闭 trace 流；支持通过 trigger 触发 trace 的 watchpoint 点，例如可以设置当执行到某个 pc 值时输出一条 sync msg
- 支持 trace filter 功能：可过滤 pc 段，privilege mode，scontext；即输出指定 pc 段的 trace 流或者在指定特权态/指定 scontext 配置下输出 trace msg

14.2 trace 支持的数据包类型

trace 支持的数据包类型如下：

1. ownership msg

   包含当前 cpu 的 priv，v，scontext 信息；当 cpu 的 privilege mode 或 scontext csr 变化后输出。

2. direct branch msg（instmode = 3 时独有）

   包含 icnt 信息；当 taken 的条件跳转指令退休时输出。

3. indirect branch msg（instmode=3 时独有）

   包含跳转指令的类型（expt/int/间接跳转指令），icnt 信息，跳转地址 pc 信息；当间接跳转产生时输出

4. indirect branch with hist msg（instmode=6 时独有）

   包含跳转指令的类型（expt/int/间接跳转指令），icnt 信息，跳转地址 pc 信息，hist 值；当间接跳转产生时输出

5. error msg

   包含 error 类型（FIFO overflow），当处理器已无资源处理当前的退休信息时输出。

5. sync msg

   包含 sync 数据包产生的原因，icnt 信息，当前 pc 信息。sync msg 产生的条件包括：

   a. trace 开启时

   b. 命中 watchpoint

   c. 从调试模式退出后

   d. 周期性 sync，可以计数 cycle/inst counter/msg number 并周期性输出 sync msg，通过配置 trace csr 实现

   e. error msg 后输出 sync，用于输出 error 后 restart 的指令信息

6. resource full msg

   包含 resource full msg 产生的原因和当前 icnt 或 hist 的值，当内部 icnt 或者 hist 数满时输出。

7. correlation msg

   包含 correlation msg 产生的原因和当前 icnt 或 hist 的值。correlation msg 产生的条件包括：

   a. 进入调试模式

   b. trace disable

14.3 Trace 寄存器编程模型

Trace 组件共有 5 种，encoder、funnel、SRAM sink、system bus sink、PIB sink、ATB bridge。每个组件有独立的控制寄存器，占据 4K 空间，所有寄存器都是 32 bit。通常 Trace 组件寄存器是通过 jtag 接口访问的，也

可以类似外设的形式通过 dmi 的 sys apb 接口连接在系统总线上，通过 debug module 上的 system bus access 接口访问 Trace 控制寄存器。

trace 组件寄存器列表具体如下各表所示，具体寄存器的定义和功能，请参考*附录 C-6 Trace 寄存器组*。

14.3.1 encoder 寄存器

表 14.1: encoder 寄存器列表

地址偏移	寄存器名称	描述
inst trace control		
0x000	trtecontrol	Trace Encoder/Funnel control register
0x004	trteImpl	Trace Encoder/Funnel implementation information
0x008	trteInstFeatures	Extra instruction trace encoder features
data trace control		
0x010	trteDataControl	Data trace control and features
timestamp control		
0x040	trtsControl	Timestamp control register
0x048	trtscounter	Lower 32 bits of timestamp counter
0x04c	trtscounterhigh	Upper bits of timestamp counter
trigger control		
0x050	trTeTrigExtInControl	External Trigger Input control register
0x054	trTeTrigExtOutControl	External Trigger Output control register

具体寄存器的定义和功能，请参考*encoder 控制寄存器组*。

14.3.2 funnel 控制寄存器

表 14.2: funnel 控制寄存器

地址偏移	寄存器名称	描述
0x000	trFunnelControl	Trace Funnel control register
0x004	trFunnelImpl	Trace Funnel Implementation information

具体寄存器的定义和功能，请参考*funnel 控制寄存器*。

14.3.3 SRAM sink/system bus sink 控制寄存器

两种 RAM sink 的控制寄存器描述相同，但其中某些寄存器在两种 sink 种的实现方式不同，具体差异见寄存器描述

表 14.3: SRAM sink/system bus sink 控制寄存器

地址偏移	寄存器名称	描述
0x000	trRamControl	Trace Funnel control register
0x004	trRamImpl	Trace Funnel Implementation information
0x010	trRamStartLow	Lower 32 bits of start address of circular trace buffer
0x014	trRamStartHigh	Upper bits of start address of circular trace buffer
0x018	trRamLimitLow	Lower 32 bits of end address of circular trace buffer
0x01C	trRamLimitHigh	Upper bits of end address of circular trace buffer
0x020	trRamWPLow	Lower 32 bits of current write location for trace data in circular buffer
0x024	trRamWPHigh	Upper bits of current write location for trace data in circular buffer
0x028	trRamRPLow	Lower 32 bits of access pointer for trace readback
0x02C	trRamRPHigh	Upper bits of access pointer for trace readback
0x040	trRamData	Read/write access to SRAM trace memory (32-bit data)

具体寄存器的定义和功能，请参考*sram sink 控制寄存器* 和*system memory sink 控制寄存器* 。

14.3.4 pib sink 寄存器

表 14.4: pib sink 寄存器列表

地址偏移	寄存器名称	描述
0x000	trPibControl	Trace PIB Sink control register
0x004	trPibImpl	Trace PIB Sink Implementation information

具体寄存器的定义和功能，请参考*PIB sink 控制寄存器* 。

14.3.5 ATB bridge 寄存器

表 14.5: ATB bridge 寄存器列表

地址偏移	寄存器名称	描述
0x000	trAtbbridgeControl	Trace ATB Sink control register
0x004	trAtbbridgeImpl	Trace ATB Sink Implementation information

具体寄存器的定义和功能，请参考*ATB bridge 控制寄存器* 。

15 功耗管理

C908X 实现了灵活的功耗管理功能，包括支持多个 power domain，支持单个 core 下电，支持 VPU 独立上下电，支持 cluster 下电，支持 L2 SRAM 处于 retention 模式（L2 使用带 retention 功能的 SRAM 时可以支持），支持通过外部硬件接口清空 L2 cache 等。本章将详细介绍 C908X 的功耗管理特性。

15.1 Power Domain

C908X 最多可以划分为 10 个 Power Domain，分别是：

- PDC0~3: 每个核心是一个 Power Domain，包括核心的计算单元、控制逻辑和 Cache RAM；
- 每个核心带对应的 VPU 可独立上下电，VPN 上电，对应的核心一定需要上电；
- PDL2RAM: L2 Cache 包含的所有 RAM 处于一个独立的 power domain（L2 使用带 retention 功能的 SRAM 时具有此 power domain）
- PDL2SYS: 覆盖 cluster 内部除了以上电源域之外的部分，包括 CIU、L2C CTRL、Debug 和 SYSIO 等子模块。

15.2 低功耗模式概要

C908X 支持下列几种低功耗模式：

- 正常模式：各个核心及对应的 VPU 与 L2 都处于正常运行的状态。
- 核心 WFI 模式：个别核心及对应的 VPU 处于 WFI 模式。
- 单个核心下电：个别核心及对应的 VPU 处于下电状态。
- L2 RAM retention 模式：Cluster 除了 L2 RAM 以外的部分处于下电状态，包括 4 个核心和 L2 控制逻辑等。
- Cluster 全部下电：Cluster 内部所有模块均处于下电状态

15.3 核心 WFI 流程

核心执行 WFI 低功耗指令，即可进入 WFI 模式，同时输出信号 core(x)_pad_lpmd_b[1:0]=2'b00，表示该核心已经进入 WFI 模式。此时，L2 子系统将在 cluster 内部关闭该核心的全局 ICG。

当发生下列事件时，核心会被唤醒并退出 WFI 模式：

- 复位；
- 中断请求：PLIC 或者 CLINT 发送的外部中断、软件中断或者 timer 中断请求。
- 调试请求。

当发生下列事件时，核心会被临时唤醒以处理该事件，处理完毕重新进入低功耗模式，但整个过程不会退出 WFI 模式：

- Snoop 请求：其他核心发送过来的 Snoop 请求。

15.4　典型低功耗切换流程

15.4.1　单核下电流程

系统可通过关断核心电源完全关断核心的静态功耗。核心电源关断流程如下：

1. C908X 被关断核心执行的操作：
   a. 屏蔽核心的所有中断请求，包括外部中断、软中断和 timer 中断，关闭 MSTATUS/SSTATUS 寄存器中断使能位（mie、sie）和 MIE/SIE 寄存器中中断使能位。

   > **备注**
   > - 下电流程执行在 M 态，关闭 MSTATUS 和 MIE 的中断使能；下电流程执行在 S 态，关闭 SSTATUS 和 SIE 的中断使能；
   > - 在支持安全和非安全异构操作系统中，核心上电复位工作在安全态，核心从安全态切换到非安全态完成初始化操作配置 PLIC 之前保持核心中断屏蔽。

   b. 关闭数据预取
   c. 等待 VPU 处于 idle 状态
   d. 核心执行 INV&CLR D-Cache ALL，将 dirty line 写回 L2C
   e. 核心关闭 D-Cache（在清 cache 和关 cache 之间不能有 store 指令）
   f. 关闭核心 snoop enable 寄存器，屏蔽对该核心的 snoop
   g. 查询 mxstatus.XUANTIEISAEE，如果开启玄铁拓展指令集，则执行 sync.is 指令；否则执行 fence.i 指令
   h. 设置系统中为核心提供的深睡眠模式寄存器（位于 cluster 外部）
   i. 执行低功耗指令 WFI，核心进入低功耗模式
2. VPU 独立下电流程
   a. 对同时在 C908X 和 VPU 中维护的 CSR 寄存器进行先读后写的方式同步数据到 Core 侧
   b. 对只在 VPU 中维护的 CSR 寄存器在进行读取并压栈
   c. 保证待下电的 VPU 处于 IDLE 状态

d. 将 misa.v 设置为 0，确保 VPU 处于 rst 时，core 不会向 VPU 发出请求。(由于设置 misa.v 为 0 时，一定没有需要 VPU 执行的在途指令，core 内待发射的向量指令队列一定会空的)

e. 激活 VPU 输出信号 isolator 的 iso_en

f. 确保 isolator 工作后，VPU 可以进入低功耗模式

3. 系统执行的操作（SOC PMC）

a. 系统检测到核心低功耗输出信号（corex_pad_lpmd_b）有效，且核心的深睡眠模式寄存器有效，触发 PMC 进入 C908X 核心下电流程

b. 系统将 pad_core(x)_dbg_mask 置高，屏蔽对该核心的调试请求

c. 系统关闭该核心的 trace 功能

d. 拉低 C908X 待下电核心的复位信号 pad_corex_rst_b，复位待下电的 CPU 核心

e. 激活核心输出信号 isolator 的 iso_en（该信号为 soc 通给 cpu，cpu 在 upf 里定义的 iso_en）

f. 在确保 isolator 工作后，外部 PMC 关闭核心电源

15.4.2 单核上电流程

核心在断电下只有通过复位才能重新启动核心。核心重新上电流程如下：

1. C908X 工作核心检测到系统负载大，需要唤醒 CPU 核心时，执行以下操作：

   通知 PMC 需要唤醒的核心

2. Power Management Controller(PMC)：

a. 设置唤醒核心对应复位地址 rvba

b. 拉低核心的复位信号

c. 打开电源，保持复位信号不释放，pll 稳定

d. 释放核心输出信号 isolator 的 iso_en

e. 释放核心复位信号

3. 被唤醒核心：

   核心执行初始化程序，开启 SMPEN 位，执行 MMU、DCACHE 使能等初始化操作。

4. VPU 上电流程

   VPU 在断电下只有通过复位才能重新启动核心。核心重新上电流程如下：

a. C908X 工作核心检测到需要唤醒 VPU 时，执行以下操作：

   通知 PMU controller 需要唤醒的 VPU

b. PMU controller：

i. 拉低 VPU 的复位信号

ii. 打开电源，保持复位信号不释放，pll 稳定

iii. 释放 VPU 输出信号 isolator 的 iso_en

iv. 释放 VPU 复位信号

c. VPU 被唤醒：

i. 核心执行初始化程序，开启 SMPEN 位，执行 MMU、DCACHE 使能等初始化操作。

ii. 将 misa.v 设置为 1

iii. 对同时在 C908X 和 VPU 中维护的 csr 寄存器进行先读后写的方式同步数据到 VPU 侧

iv. 对只在 VPU 中维护的 csr 寄存器，进行弹栈操作，并通过 csr 写操作写入 VPU 中

15.4.3　Cluster 下电流程

当 cluster 各个核心电源关断后，顶层电源可选择关断，从而关闭整个 cluster 的静态功耗。顶层电源关断流程如下：

1. Cluster 内除主核外其余三个核心电源均已关闭
2. 主核执行操作：（这里以主核为例方便说明，最后一个下电的核心可以是任意核）
   a. 屏蔽核心的所有中断请求，包括外部中断、软中断和 timer 中断，关闭 MSTATUS/SSTATUS 寄存器中断使能位（mie、sie）和 MIE/SIE 寄存器中中断使能位。
   b. 关闭数据预取
   c. 执行 D-Cache inv&clr all 操作，将 dirty line 写回 L2C
   d. 关闭 D-Cache
   e. 关闭核心的 SMPEN
   f. 查询 mxstatus.XUANTIEISAEE，如果开启玄铁拓展指令集，则执行 sync.is 指令；否则执行 fence.i 指令
   g. 设置系统中为主核提供的深睡眠模式寄存器（位于 cluster 外部）
   h. 执行低功耗指令 WFI，进入低功耗模式
3. 系统执行操作
   a. PMC 设置主核进入电源关断模式，激活主核输出信号 isolator 的 iso_en
   b. 外部 PMC 关闭主核电源
   c. SOC 通过 CIU 寄存器 L2OPCR 向 L2 发起 flush
   d. 等待 C908X 返回 cpu_pad_no_op，激活顶层输出信号 isolator 的 iso_en
   e. 如果 tdt_top 独立电源域且 tdt_top 不掉电，则将 tdt_top 内部输入信号 isolator 的 iso_en 拉高
   f. 关闭顶层（L2SYS 和 L2RAM）电源
   g. 如果 tdt_top 独立电源域且需要将 tdt_top 下电，则关闭 TDT_TOP 电源

15.4.4　Cluster 上电流程

顶层通过复位重新上电，上电流程如下：

1. 拉低 cluster 内所有核心和顶层的复位信号
2. 打开电源，保持复位信号不释放，pll 稳定
3. 如果 tdt_top 独立电源域，释放 tdt_top 输入信号钳位
4. 释放各核心和顶层的输出信号钳位
5. 保持 pad_cpu_l2cache_init_disable 为低，使能 L2C 上电初始化
6. 释放各核心和顶层的复位信号
7. 执行复位异常服务程序，恢复 CPU 状态。

15.4.5 TDT 单独上下电 (核心和顶层正常工作，SoC 控制)

当 cluster 各个核心在正常工作（非调试模式）时，SoC 可以将 PD_DBG 下电来节省功耗:

1. 确保没有任何核心处于调试模式并且 tdt_mp_top 处于 idle 状态
2. 停掉 trace 相关时钟，pad_tdt_trace_atb_clk 和 sink 的时钟
3. 停掉 dm 的时钟，sys_apb_clk, pad_tdt_dtm_tclk
4. 拉低 trace 复位信号，先拉低 pad_tdt_trace_atb_rst_b 再拉低 sink 相关复位信号
5. 拉低 dm 的复位信号，sys_apb_rst_b, pad_tdt_dtm_trst_b
6. 拉高 tdt_mp_top 输出信号 isolator 的 iso_en（该信号为 soc 通给 cpu，cpu 在 upf 里定义的 iso_en）
7. 关闭 PD_DBG 电源

tdt_mp_top 重新上电（调试模式下）:

1. 打开 PD_DBG 电源
2. 拉低 tdt_mp_top 输出信号 isolator 的 iso_en
3. 开启 dm 时钟，sys_apb_clk, pad_tdt_dtm_tclk
4. 开启 trace 相关时钟，pad_tdt_trace_atb_clk 和 sink 的时钟
5. 释放 dm 的复位信号，sys_apb_rst_b, pad_tdt_dtm_trst_b
6. 释放 trace 的相关复位信号，先释放 sink 的复位信号，再释放 pad_tdt_trace_atb_rst_b

15.5 简化场景：Cluster 整体下电流程（硬件清空 L2）

在某些系统中，SoC 设计人员可能采取比较简单的方式来划分 power domain，即：把 C908X Cluster（4 核心 +L2）整体作为一个 power domain，整体下电，不区分各个单独的核心。在这样的场景下，Cluster 下电流程（硬件清空 L2）可以采取如下的步骤:

系统执行的操作:

1. 通知 SoC 即将进入 Cluster 整体下电流程，具体方式取决于 SoC 的设计。
2. 确保 DCP（如果有）上现有的传输全部完成，并且不再向 DCP 发送新的读写请求。

核心执行的操作：（此时，不必区分“主核”与“副核”，他们的流程相同。）

1. 屏蔽核心的所有中断请求，包括外部中断、软中断和 timer 中断，关闭 MSTATUS/SSTATUS 寄存器中断使能位（mie、sie）和 MIE/SIE 寄存器中断使能位。
2. 关闭数据预取。
3. 核心执行 INV&CLR D-Cache ALL，将 dirty line 写回 L2 Cache。
4. 核心关闭 D-Cache（在清 cache 和关 cache 之间不能有 store 指令）。
5. 核心关闭 SMPEN 位，屏蔽对该核心的 snoop 请求。
6. 核心执行 fence iorw, iorw 指令。
7. 核心执行 WFI。

系统执行的操作：

1. 等待全部 core(x)_pad_lpmd_b[1:0]==2'b00，即：所有 CPU 都已进入低功耗状态。
2. 将全部 pad_tdt_dm_core_unavail[x] 置高，屏蔽调试请求。
3. 完成上述操作后，开始清空 L2Cache，首先 SoC 通过调试接口配置寄存器关闭 L2 RAM，向 L2DECR.EXP_L2EN 位写 00 关闭 L2 Cache
4. 写寄存器完成后 SoC 通过调试接口轮询 L2DECR 寄存器，等到 CUR_L2EN 为 00 后表示 L2Cache 进入空闲状态。（此时所有的 CPU 也仍然处于低功耗状态）
5. 激活 cluster 输出信号的 clamp。
6. Assert 全部复位信号。
7. 将整个 cluster 掉电。

15.6 低功耗相关的编程模型和接口信号

15.6.1 编程模型的变化

机器模式复位寄存器（MRMR）

该寄存器已经被删除。如果继续访问该寄存器，则读为零，写无效，不会上报异常。这一变化产生的影响是：各个核心的复位信号不再受到 MRMR 的控制，SoC 可以通过 pad_core(x)_rst_b 独立控制各个核心的复位与解复位。

机器模式复位向量基址寄存器（MRVBR）

该寄存器的编程模型有修改。实现方式由“4 核共享”改为“各核私有”。访问权限由“MRW”改为“MRO”。各个核心的 MRVBR 初始值各自独立，由硬件信号 pad_core(x)_rvba[39:1] 决定。

15.6.2 接口信号

C908X 与 SoC 功耗管理单元的沟通主要通过以下信号实现：

- core(x)_pad_lpmd_b：

  可以用来判断一个核心是否处于 WFI 模式。2'b11 代表正常模式；2'b00 代表 WFI 模式。

- cpu_pad_no_op：

  L2 Cache 空闲指示信号。当所有核心都进入低功耗模式，且 L2 Cache 完成了全部传输时，该信号有效（高电平有效）。

- tn(x)_pad_vector_idle:

  可以用来判断 VPU 处于 idle 状态，只有 VPU 处于 idle 状态时才能下电。

16 性能监测单元

16.1 PMU 简介

C908X 性能监测单元（PMU）遵从 RISC-V 标准，用于统计程序运行中的软件信息和部分硬件信息，供软件开发人员进行程序优化。

性能监测单元统计的软硬件信息分为以下几类：

- 运行时钟数和时间（cycle, time）
- 指令信息统计（instret）
- 处理器关键部件信息统计（hpmcounter3 ~ hpmcounter31。其中 hpmcounter19 ~ hpmcounter31 并不存在，且其对应的控制寄存器或控制位没有实现）

16.2 PMU 的编程模型

16.2.1 PMU 的基本用法

PMU 的基本用法如下：

- 通过 mcountinhibit 寄存器禁止所有事件的计数。
- 把各个 PMU counter 的当前值清零，包括：mcycle，minstret，mhpmcounter3 ~ mhpmcounter31。
- 为各个 PMU counter 配置对应的事件。C908X 中每个计数器可以配置任意一个事件。将事件索引值写入性能监测事件选择寄存器，该计数器即可对配置的事件正常计数。例如，向 mhpmevent5 写入 0x1，表示 mhpmcounter5 针对 0x1 号事件（L1 ICache 访问次数）计数；向 mhpmevent5 写入 0x2，表示 mhpmcounter5 针对 0x2 号事件（L1 ICache Miss 次数）计数。
- 访问权限授权：通过 mcounteren 决定 S 态是否可以访问 PMU counter；通过 scounteren 决定 U 态是否可以访问 PMU counter。
- 通过 mcountinhibit 寄存器解除禁止，开始计数。

具体例子可参考*PMU 设置示例*

16.2.2 PMU 事件溢出中断

由性能检测单元发起的溢出中断统一中断向量号为 13。中断的使能及处理过程同普通私有中断，详见*异常与中断*。

16.3 PMU 相关的控制寄存器

16.3.1 机器模式计数器访问授权寄存器（mcounteren）

机器模式计数器访问授权寄存器（mcounteren），用于授权超级用户模式是否可以访问用户模式计数器。

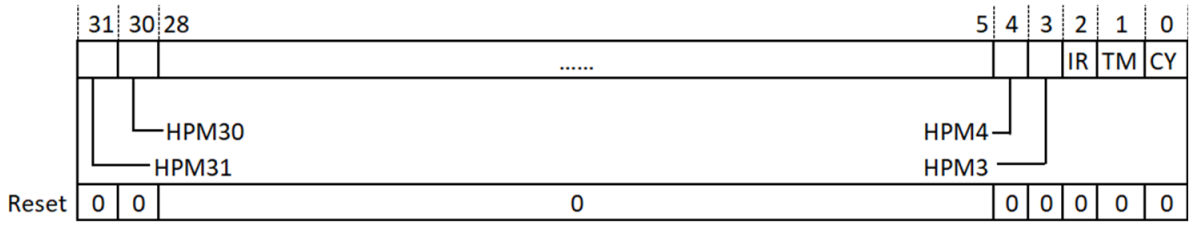


图 16.1: 机器模式计数器访问授权寄存器（MCOUNTEREN）

表 16.1: 机器模式计数器访问授权寄存器说明

位	读写	名称	介绍
31:3	读写	HPMn	shpmcountern/hpmcountern 寄存器 S-mode 访问位： 0：S-mode 访问 shpmcountern/ hpmcountern 将发生非法指令异常 1：S-mode 能正常访问 shpmcountern/ hpmcountern
2	读写	IR	sinstret/instret 寄存器 S-mode 访问位： 0：S-mode 访问 sinstret/instret 寄存器将发生非法指令异常 1：S-mode 能正常访问 sinstret/ instret 寄存器
1	读写	TM	time/stimecmp 寄存器 S-mode 访问位： 0：S-mode 访问 time/stimecmp 寄存器将发生非法指令异常 1：S-mode 能正常访问 time/stimecmp 寄存器
0	读写	CY	scycle/cycle 寄存器 S-mode 访问位： 0：S-mode 访问 scycle/cycle 寄存器将发生非法指令异常 1：S-mode 能正常访问 scycle/cycle 寄存器

16.3.2 超级用户模式计数器访问授权寄存器（scounteren）

超级用户模式计数器访问授权寄存器（scounteren），用于授权用户模式是否可以访问用户模式计数器。

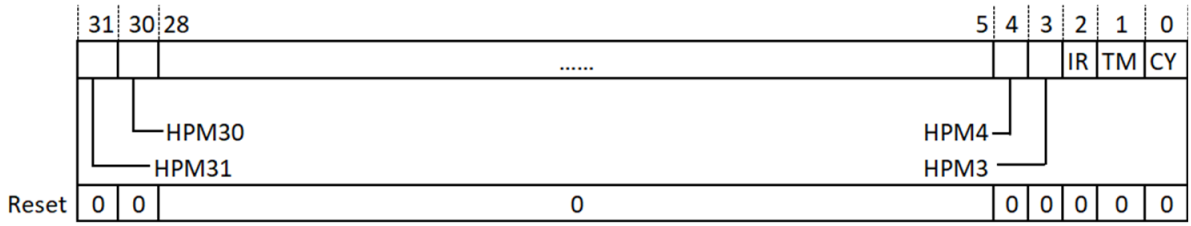


图 16.2: 超级用户模式计数器访问授权寄存器（SCOUNTEREN）

表 16.2: 超级用户模式计数器访问授权寄存器说明

位	读写	名称	介绍
31：3	读写	HPM*n*	hpmcounter*n* 寄存器 U-mode 访问位： 0：U-mode 访问 h pmcounter*n* 将发生非法指令异常 1：当 mcounteren 对应位为 1，则 U-mode 能正常访问 hpmcounter*n*，否则将发生非法指令异常
2	读写	IR	instret 寄存器 U-mode 访问位： 0：U-mode 访问 instret 寄存器将发生非法指令异常 1：当 mcounteren 对应位为 1，U-mode 能正常访问 instret 寄存器，否则将发生非法指令异常
1	读写	TM	time 寄存器 U-mode 访问位： 0：U-mode 访问 time 寄存器将发生非法指令异常 1：当 mcounteren 对应位为 1，U-mode 能正常访问 time 寄存器，否则将发生非法指令异常
0	读写	CY	cycle 寄存器 U-mode 访问位： 0：U-mode 访问 cycle 寄存器将发生非法指令异常 1：当 mcounteren 对应位为 1，U-mode 能正常访问 cycle 寄存器，否则将发生非法指令异常

16.3.3 机器模式计数禁止寄存器（mcountinhibit）

机器模式禁止计数寄存器（mcountinhibit），可以禁止机器模式计数器计数。在不需要性能分析的场景下，关闭计数器，可以降低处理器的功耗。

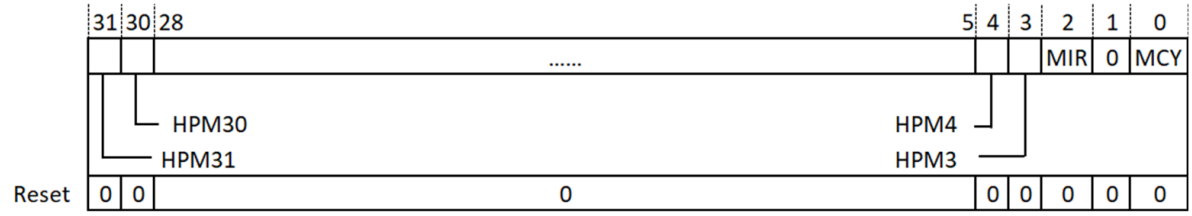


图 16.3: 机器模式计数禁止授权寄存器（MCOUNTINHIBIT）

表 16.3: 机器模式计数禁止寄存器说明

位	读写	名称	介绍
31：3	读写	MHPM*n*	mhpmcounter*n* 寄存器禁止计数位： 0：正常计数 1：禁止计数
2	读写	MIR	minstret 寄存器禁止计数位： 0：正常计数 1：禁止计数

续下页

表 16.3 - 接上页

位	读写	名称	介绍
1	-	-	-
0	读写	MCY	mcycle 寄存器禁止计数位： 0：正常计数 1：禁止计数

16.3.4 超级用户模式禁止计数寄存器（scountinhibit）

超级用户模式禁止计数寄存器（scountinhibit），可以禁止超级用户模式计数器计数。该寄存器可以在不需要性能分析的场景下，关闭计数器，从而降低处理器功耗。当 mcounterwen.bit[n] 为 1 时，S 态可以读写 scountinhibit [n]。

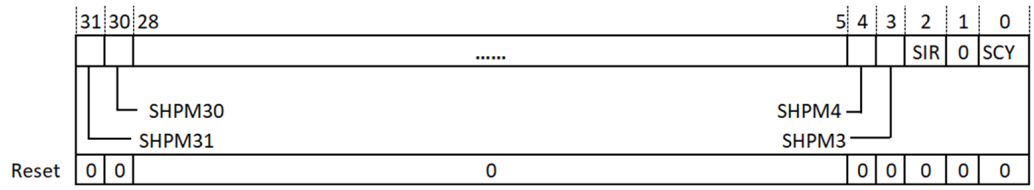


图 16.4: 超级用户模式禁止计数寄存器（scountinhibit）

表 16.4: 超级用户模式计数禁止寄存器说明

位	读写	名称	介绍
31:3	读写	SHPMn	shpmcountern 寄存器禁止计数位： 0：正常计数 1：禁止计数
2	读写	SIR	sinstret 寄存器禁止计数位： 0：正常计数 1：禁止计数
1	-	-	-
0	读写	SCY	scycle 寄存器禁止计数位： 0：正常计数 1：禁止计数

16.3.5 机器模式计数器写使能寄存器（mcounterwen）

机器模式计数器写使能寄存器（MCOUNTERWEN），用于授权超级用户模式是否可以写超级用户模式事件计数器。该寄存器为机器模式拓展寄存器，寄存器具体描述参见*附录 C-1 机器模式控制寄存器*。

16.3.6　超级用户模式计数器溢出寄存器（SCOUNTOVF）

超级用户模式计数器溢出寄存器，来自于 sscofpmf 扩展。

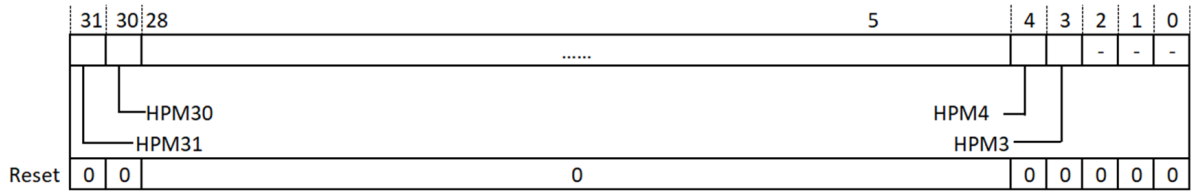


图 16.5: 超级用户模式计数器溢出寄存器（SCOUNTOVF）

表 16.5: 超级用户模式计数器溢出寄存器说明

位	读写	名称	介绍
31：3	只读	HPM*n*	shpmcounter*n* 寄存器上溢标志位： 1'b0：shpmcounter*n* 没有发生上溢 1'b1：shpmcounter*n* 发生上溢
2	-	-	-
1	-	-	-
0	-	-	-

- scountovf 寄存器各比特位为各事件选择器 [63] OF bit 的只读映射；
- 当 mcounteren[i] =1 时，scountovf 在 M 和 S 态可正常读，否则读值为 0。

16.3.7　性能监测事件选择寄存器

机器模式性能监测事件选择器（mhpmevent3~31），用于选择每个计数器对应的计数事件。C908X 中，每个计数器可以配置任意一个事件。将事件索引值写入性能监测事件选择寄存器，该计数器即可对配置的事件正常计数。

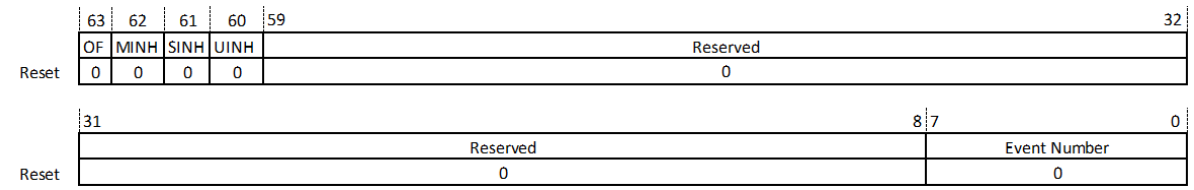


图 16.6: 机器模式性能监测事件选择寄存器（MHPMEVENT3~31）

表 16.6 为机器模式性能监测事件选择寄存器说明。

表 16.6: 机器模式性能监测事件选择寄存器说明

位	读写	名称	介绍
63	MRW	OF	sscofpmf 扩展新增性能计数上溢标志位: CP0 写初值 0, 对应性能事件计数器溢出时置 1, 产生溢出中断; 期间继续 wrap 计数, 之后不再产生新的溢出中断, 直至软件重写 CP0 写初值 1, 对应性能事件计数器溢出时值不变, 不产生溢出中断; 期间继续 wrap 计数, 直至软件重写
62	MRW	MINH	sscofpmf 扩展新增 M 态性能计数器禁止计数位
61	MRW	SINH	sscofpmf 扩展新增 S 态性能计数器禁止计数位
60	MRW	UINH	sscofpmf 扩展新增 U 态性能计数器禁止计数位
59~8	MRW	Reserved	sscofpmf 扩展保留
7~0	MRW	Event Number	事件索引号: 当 EVENT_NUMBER 为 0 时, 事件计数器 n 不使能, 不计数; 当 EVENT_NUMBER 为非 0 时, 事件计数器 n 使能, 并且正常对事件号的事件计数。

表 16.7 为计数器事件编号与计数器事件名称对照表。

表 16.7: 计数器事件对应列表

索引	事件
0x1	L1-icache Access
0x2	L1-icache Miss
0x3	iTLB Miss
0x4	未实现定义
0x5	jTLB Miss
0x6	Condition Branch Mispred
0x7	Condition Branch
0x8	Indirect Branch Miss
0x9	Indirect Branch
0xA	未实现定义
0xB	Store Instruction
0xC	L1-dcache load access
0xD	L1-dcache load miss
0xE	L1-dcache store access
0xF	L1-dcache store miss
0x10	未实现定义
0x11	未实现定义

续下页

表 16.7 - 接上页

索引	事件
0x12	未实现定义
0x13	未实现定义
0x14	未实现定义
0x15	未实现定义
0x16	Issue Instruction
0x17	未实现定义
0x18	未实现定义
0x19	未实现定义
0x1A	未实现定义
0x1B	IFU Branch Target Mispred
0x1C	IFU Branch Target Instruction
0x1D	ALU Instruction
0x1E	未实现定义
0x1F	Vector SIMD Instruction
0x20	CSR Instruction
0x21	ATOMIC Instruction
0x22	未实现定义
0x23	Interupt Number
0x24	未实现定义
0x25	Environment Call
0x26	Long Jump
0x27	Stalled Cycles Frontend
0x28	Stalled Cycles Backend
0x29	SYNC Stall
0x2A	Float Point Instruction
0x2B	M Mode Cycles
0x2C	S Mode Cycles
0x2D	U Mode Cycles
0x2E	Exception Number
0x2F	Flush Number
0x30	LOAD Instruction
0x31	Fused Instruction
0x32	MULT Instruction

续下页

表 16.7 - 接上页

索引	事件
0x33	DIV Instruction
0x34	Mult Inner Forward
0x35	DIV Buffer Hit
0x36	Branch Instruction
0x37	Uncondition Branch
0x38	Branch Mispred
0x39	Uncondition Branch Mispred
0x3A	Taken Branch Mispred
0x3B	Taken Condition Branch
0x3C	Taken Condition Branch Mispred
0x3D	Unalign LOAD Instruction
0x3E	Unalign STORE Instruction
0x3F	LR Instruction
0x40	SC Instruction
0x41	AMO Instruction
0x42	Barrier Instruction
0x43	Failed SC Instruction
0x44	Bus Barrier
0x45	FP DIV Instruction
0x46	FP LOAD Instruction
0x47	FP STORE Instruction
0x48	Vector DIV Instruction
0x49	未实现定义
0x4A	未实现定义
0x4B	Vector Micro Op
0x4C	ECC Interrupt
0x4D	Async Abort Interrupt
0x4E	IF Stall
0x4F	IP Stall
0x50	IB Stall
0x51	IF Refill Stall
0x52	IF Mmu Stall
0x53	IB Mispred Stall

续下页

表 16.7 - 接上页

索引	事件
0x54	IB Fifo Stall
0x55	IB Ind Btb Rd Stall
0x56	IB Vsetvl Stall
0x57	ID Stall
0x58	RF Stall
0x59	EU Stall
0x5A	ID Inst Pipedown
0x5B	RF Inst Pipedown
0x5C	ID One Inst Pipedown
0x5D	ID CSR Before Fence Stall
0x5E	ID VSETVL Fof Stall
0x5F	ID Flush Stall
0x60	ID Misprediction Stall
0x61	ID IID Not Vld Stall
0x62	RF One Inst Pipedown
0x63	RF RAW Stall
0x64	RF WAW Stall
0x65	RF Structure Stall
0x66	RF CSR After Fence Stall
0x67	EU IU Full
0x68	EU IU Control Full
0x69	EU CP0 Full
0x6A	EU LSU LOAD Full
0x6B	EU LSU STORE Full
0x6C	EU VFPU Full
0x6D	EU BJU Full
0x6E	IU Dp Stall Pipe0
0x6F	IU MULT Stall Pipe0
0x70	IU DIV EX1 Stall Pipe0
0x71	IU Dp Stall Pipe1
0x72	IU MULT Stall Pipe1
0x73	IU DIV EX1 Stall Pipe1
0x74	IU DP Wb Conflict Pipe0

续下页

表 16.7 - 接上页

索引	事件
0x75	IU DP WAW Stall Pipe0
0x76	IU DP Uncommit Pipe0
0x77	IU DP Wb Conflict Pipe1
0x78	IU DP WAW Stall Pipe1
0x79	IU DP Uncommit Pipe1
0x7A	IU MULT Uncommit
0x7B	IU MULT Wb Stall
0x7C	IU DIV Uncommit
0x7D	IU DIV Wb Stall
0x7E	LSU LOAD WAW Stall
0x7F	LSU LOAD Commit Stall
0x80	LSU LOAD RAW Stall
0x81	LSU STORE Commit Stall
0x82	Vidu Rf No Pipedown
0x83	VPU Stall Pipe0
0x84	VPU Stall Pipe1
0x85	VPU Struct Hazard Stall Pipe0
0x86	VPU Uncommit Stall Pipe0
0x87	VPU VLSU Stall Pipe0
0x88	VPU Struct Hazard Stall Pipe1
0x89	VPU Uncommit Stall Pipe1
0x8A	VPU VLSU Stall Pipe1
0x8B	VFPU FDIV/VDIV Busy
0x8C	BJU CP0 Stall
0x8D	BJU IBUF Stall
0x8E	BJU Wb Stall
0x8F	BJU Pipedown Stall
0x90	RTU Flush
0x91	RTU IU Not No OP
0x92	RTU BJU Not No OP
0x93	RTU LSU Not No OP
0x94	RTU CP0 Not No OP
0x95	RTU VFPU Not No OP

续下页

表 16.7 - 接上页

索引	事件
0x96	RTU Only IU Not No OP
0x97	RTU Only BJU Not No OP
0x98	RTU Only LSU Not No OP
0x99	RTU Only CP0 Not No OP
0x9A	RTU Only VFPU Not No OP
0x9B	L1 Dcache Access
0x9C	L1 Dcache Miss
0x9D	L1 Dcache Exclusive Eviction
0x9E	Icache Prefetch
0x9F	Dcache Amr Active
0xA0	Icache Prefetch Miss
0xA1	Dcache Refill Casued by Prefetch
0xA2	Dcache Hit Caused by Prefetch
0xA3	Store Dtlb Miss
0xA4	Load Dtlb Miss
0xA5	L2 DCache Access
0xA6	L2 DCache Miss
0xA7	L2 ICache Access
0xA8	L2 ICache Miss
0xA9	Snb Read Create Vld
0xAA	Snb Read Create Stall
0xAB	Iq Full
0xAC	Vidu Vec0 Stall
0xAD	Vidu Vec1 Stall
0xAE	Vidu Vec0 Depend Stall
0xAF	Vidu Vec0 Struct Hazard Stall
0xB0	Vidu Vec1 Depend Stall
0xB1	Vidu Vec1 Struct Hazard Stall
0xB2	Vidu Total Cycle
0xB3	Vidu Vec0 Cycle
0xB4	Vidu Vec1 Cycle
0xB5	Vector 2D reduce instruction
0xB6	Vector hbf16 to fp8 instruction

续下页

表 16.7 - 接上页

索引	事件
0xB7	VPU Micro TLB Miss
0xB8	LSOT Full
0xB9	VPU GIU Full
0xBA	VPU LSOT shade Stall
0xBB	VPU LIQ Full
0xBC	VPU SIQ Full
0xBD	VPU VLACQ Full
0xBE	VPU VSACQ Full
0xBF	VPU VLDCQ Full
0xC0	VPU VSDCQ Full
0xC1	VPU VLACQ addr depend
0xC2	VPU fast_memory access
0xC3	VPU L2 memory access
0xC4	Fast_complete
>0xC4	暂未实现定义

16.3.8 机器模式 cycle 事件设置寄存器、机器模式 inst retired 事件设置寄存器

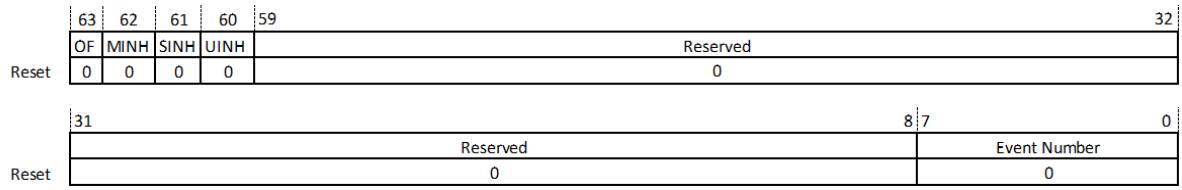


图 16.7: MHPMEVENT0/MHPMEVENT2

C908X 扩展实现 mhpmevent0、mhpmevent2 寄存器，分别用于对 cycle 计数和 inst-retired 事件计数的权限配置。

当 mxstatus.ofint=0 时，对 mhpmevent0、mhpmevent2 读为 0 写无效。

当 mxstatus.ofint=1 时，mhpmevent0、2 可读写。

- 当 mxstatus.ofint=1 ，mhpmevent0、2 寄存器的 minh、sinh、uinh 位为 1 时，表示禁止 m/s/u 态对应 counter 的计数。
- 当 mxstatus.ofint=1 ，mhpmevent0、2 寄存器的 minh、sinh、uinh 位为 0 时，表示允许 m/s/u 态对应 counter 的计数。

16.3.9 事件计数器

事件计数器有三组，分别为：机器模式事件计数器、用户模式事件计数器和 C908X 扩展的超级用户模式事件计数器。具体如 表 16.8 所示。

表 16.8: 机器模式事件计数器列表

名称	索引	读写	初始值	介绍
MCYCLE	0xB00	MRW	0x0	cycle counter
MINSTRET	0xB02	MRW	0x0	instructions-retired counter
MHPMCOUNTER3	0xB03	MRW	0x0	performance-monitoring counter
MHPMCOUNTER4	0xB04	MRW	0x0	performance-monitoring counter
...	...	...	...	...
MHPMCOUNTER31	0xB1F	MRW	0x0	performance-monitoring counter

用户模式事件计数器列表如 表 16.9 所示。

表 16.9: 用户模式事件计数器列表

名称	索引	读写	初始值	介绍
CYCLE	0xC00	URO	0x0	cycle counter
TIME	0xC01	URO	0x0	timer
INSTRET	0xC02	URO	0x0	instructions-retired counter
HPMCOUNTER3	0xC03	URO	0x0	performance-monitoring counter
HPMCOUNTER4	0xC04	URO	0x0	performance-monitoring counter
...	...	...	...	...
HPMCOUNTER31	0xC1F	URO	0x0	performance-monitoring counter

表 16.10: 超级用户模式事件计数器列表

名称	索引	读写	初始值	介绍
SCYCLE	0x5E0	SRO	0x0	cycle counter
SINSTRET	0x5E2	SRO	0x0	instructions-retired counter
SHPMCOUNTER3	0x5E3	SRO	0x0	performance-monitoring counter
SHPMCOUNTER4	0x5E4	SRO	0x0	performance-monitoring counter
...	...	...	...	...
SHPMCOUNTER31	0x5FF	SRO	0x0	performance-monitoring counter

其中，用户模式的 CYCLE、INSTRET 和 HPMCOUNTERn 是对应机器模式事件计数器的只读映射，TIME 计数器是 MTIME 的只读映射；超级用户模式的 SCYCLE、SINSTRET 和 SHPMCOUNTERn 是对应机器模式事件计数

器的映射。

16.4 触发寄存器

C908X 触发寄存器如下表所示：

表 16.11: 触发寄存器列表

名称	索引	读写	初始值	介绍
MHPMSR	0x7f1	MRW	0x0	M-mode 起始触发寄存器
MHPMER	0x7f2	MRW	0x0	M-mode 终止触发寄存器
SHPMSR	0x5ca	SRW	0x0	S-mode 起始触发寄存器
SHPMER	0x5cb	SRW	0x0	S-mode 终止触发寄存器

触发寄存器组织形式如下图所示：

16.4.1 起始触发寄存器

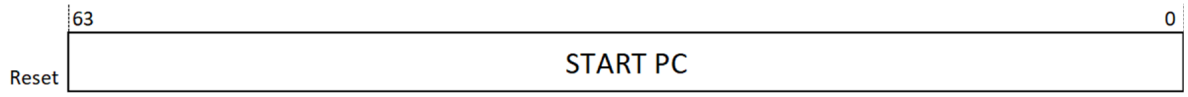


图 16.8: 起始触发寄存器

16.4.2 终止触发寄存器

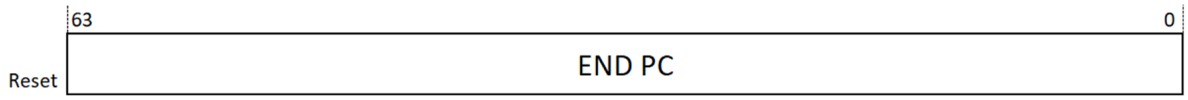


图 16.9: 终止触发寄存器

基于不同的触发模式，起始触发寄存器与终止触发寄存器联合提供用户指定的触发功能：

TRIGGER/STOP 模式:

用户通过起始触发寄存器指定计数器开始统计的起点，通过终止触发寄存器指定计数器结束统计的终点。当 MHPMCR.TME 使能时，一旦程序运行至起始触发寄存器指定的 PC，计数器开始计数，直到程序运行至终止触发寄存器方才结束统计。用户可通过触发功能实现对特定线程及特定函数等的统计。

START/END 模式:

当程序运行的 PC 在起始触发寄存器和终止触发寄存器的范围内时，计数器正常计数，一旦指令退休地址不在该范围内，所有计数器均停止计数。

相比于 TRIGGER/STOP 模式来说，START/END 模式对计数的条件更加严格，即当程序段中发生了子程序调用等导致指令运行不在 START 和 END 之间，都会引起计数器的停止。

另外，对于 S 态的触发寄存器，其访问受 mhpmcr.SCE 位控制，当 SCE 位为 1 时，触发寄存器可以正常读写，否则将触发非法指令。

17 程序示例

本章主要介绍多种程序示例，包含：MMU 设置示例、PMP 设置示例、高速缓存设置示例、同步原语示例、PLIC 设置示例和 PMU 设置示例。

17.1 处理器最优性能配置

采用下列配置，可以发挥 C908X 的最优性能：

- MHCR = 0x10011FF
- MHINT = 0x21AA10C
- MCCR2 = 0xA042000A （注：mccr2 包含 RAM 延时的设定，客户需要根据实际情况设定合适的 RAM 延时）
- MXSTATUS = 0x638000
- MSMPR = 0x1

```
# mhcr
  li x3, 0x10011ff
  csrs mhcr,x3

# mhint
  li x3, 0x21aa10c
  csrs mhint,x3

# mxstatus
  li x3, 0x638000
  csrs mxstatus,x3

# msmpr
  csrsi msmpr,0x1

# mccr2
  li x3, 0xa042000a
  csrs mccr2,x3
```

17.2 MMU 设置示例

```
/*****************************************************************************

* Function: An example of setting C908X MMU.
* Memory space: Virtual address <-> physical address.
*
* Pagesize 4K: vpn: {vpn2,vpn1,vpn0} <-> ppn: {ppn2,ppn1,ppn0}
* Pagesize 2M: vpn: {vpn2,vpn1} <-> ppn:{ppn2,ppn1}
* Pagesize 1G: vpn: {vnp2} <-> ppn: {ppn2}
*
*****************************************************************************/

  /*C908X will invalidate all MMU TLB entries automatically when reset*/
  /*You can use sfence.vma to invalid all MMU TLB entries if necessary*/
  sfence.vma x0, x0

  /* Pagesize 4K: vpn: {vpn2, vpn1, vpn0} <-> ppn: {ppn2, ppn1, ppn0}*/
  /* First-level page addr base: PPN (defined in satp)*/
  /* Second-level page addr base: BASE2 (self define)*/
  /* Third-level page addr base: BASE3 (self define)*/
  /* 1. Get first-level page addr base: PPN and vpn*/
  /* Get PPN*/
  csrr x3, satp
  li x4, 0xfffffffffff
  and x3, x3, x4

  /*2. Config first-level page*/
  /*First-level page addr: {PPN, vpn2, 3'b0}, first-level page pte:{ 44'b BASE2, 10
↪'b1} */
  /*Get first-level page addr*/
  slli x3, x3, 12
  /*Get vpn2*/
  li x4, VPN
  li x5, 0x7fc0000
  and x4, x4, x5
  srli x4, x4, 15
  and x5, x3, x4
  /*Store pte at first-level page addr*/
  li x6, {44'b BASE2, 10'b1}
  sd x6, 0(x5)

  /*3. Config second-level page*/
```

(续下页)

(接上页)

```
  /*Second-level page addr: {BASE2, vpn1, 3'b0}, second-level page pte:{ 44'b␣
↪BASE3, 10'b1} */
  /*Get second-level page addr*/
  /* VPN1*/
  li x4, VPN
  li x5, 0x3fe00
  and x4, x4, x5
  srli x4, x4, 9
  /*BASE2*/
  li x5, BASE2
  srli x5, x5, 12
  and x5, x5, x4
  /*Store pte at second-level page addr*
  li x6, {44'b BASE3, 10'b1}
  sd x6, 0(x5)
  /*4. Config third-level page*/
  /*Third-level page addr: {BASE3, vpn0, 3'b0}, third-level page pte:{
  theadflag, ppn2, ppn1, ppn0, 9'b flags,1'b1} */
  /*Get second-level page addr*/
  /* VPN0*/
  li x4, VPN
  li x5, 0x1ff
  and x4, x4, x5
  srli x4, x4, 3
  /*BASE3*/
  li x5, BASE3
  srli x5, x5, 12
  and x5, x5, x4
  /*Store pte at second-level page addr*/
  li x6, { theadflag, ppn2, ppn1, ppn0, 9'b flags, 1'b1}
  sd x6, 0(x5)


  /* Pagesize 2M: vpn: {vpn2, vpn1} <-> ppn: {ppn2, ppn1}*/
  /*First-level page addr base: PPN (defined in satp)*/
  /*Second-level page addr base: BASE2 (self define)*/

  /*1. Get first-level page addr base: PPN and vpn*/
  /* Get PPN*/
  csrr x3, satp
  li x4, 0xfffffffffff
  and x3, x3, x4
```

(续下页)

(接上页)

```
/*2. Config first-level page*/
/*First-level page addr: {PPN, vpn2, 3'b0}, first-level page pte:{ 44'b
BASE2, 10'b1}*/
/*Get first-level page addr*/
slli x3, x3, 12
/*Get vpn2*/
li x4, VPN
li x5, 0x7fc0000
and x4, x4, x5
srli x4, x4, 15
and x5, x3, x4
/*Store pte at first-level page addr*/
li x6, {44'b BASE2, 10'b1}
sd x6, 0(x5)

/*3. Config second-level page*/
/*Second-level page addr: {BASE2, vpn1, 3'b0}, second-level page pte:{
theadflag, ppn2, ppn1, 9'b0, 9'b flags,1'b1} */
/*Get second-level page addr*/
/*VPN1*/
li x4, VPN
li x5, 0x3fe00
and x4, x4, x5
srli x4, x4, 9
/*BASE2*/
li x5, BASE2
srli x5, x5, 12
and x5, x5, x4
/*Store pte at second-level page addr*/
li x6, { theadflag, ppn2, ppn1, 9'b0, 9'b flags,1'b1}
sd x6, 0(x5)

/* Pagesize 1G: vpn: {vpn2} <-> ppn: {ppn2}*/
/*First-level page addr base: PPN (defined in satp)*/
/*1. Get first-level page addr base: PPN and vpn*/
/* Get PPN*/
csrr x3, satp
li x4, 0xfffffffffff
and x3, x3, x4

/*2. Config first-level page*/
```

(续下页)

(接上页)

```
  /*First-level page addr: {PPN, vpn2, 3'b0}, first-level page pte:{
  theadflag, ppn2, 9'b0, 9'b0, 9'b flags,1'b1}*/
  /*Get first-level page addr*/
  slli x3, x3, 12
  /*Get vpn2*/
  li x4, VPN
  li x5, 0x7fc0000
  and x4, x4, x5
  srli x4, x4, 15
  and x5, x3, x4
  /*Store pte at first-level page addr*/
  li x6, { theadflag, ppn2, 9'b0, 9'b0, 9'b flags,1'b1}
  sd x6, 0(x5)
```

17.3 PMP 设置示例

```
/*****************************************************************************
* Function: An example of setting C908X PMP.
* 0x0 ~ 0xf0000000, TOR 模式, RWX
* 0xf0000000 ~ 0xf8000000, NAPOT 模式, RW
*0xfff73000 ~ 0xfff74000, NAPOT 模式, RW
*0xfffc0000 ~ 0xfffc2000, NAPOT 模式, RW
* 以上 4 片区域配置了不同的执行权限。另外，为了防止 CPU 在不同模式下投机执行到不支持的地址
区域，尤其是在默认拥有全部执行权限的机器模式下，需要对 PMP 进行相关配置。具体来说，就
是配置完需要执行权限的地址区域后，将剩余地址区域配置成无任何权限，如下例所示。
*****************************************************************************/

  # pmpaddr0,0x0 ~ 0xf0000000, TOR 模式，读写可执行权限
  li x3, (0xf0000000  >>  2)
  csrw pmpaddr0, x3
  # pmpaddr1,0xf0000000 ~ 0xf8000000, NAPOT 模式，读写权限
  li x3, ( 0xf0000000  >>  2 | (0x8000000-1)  >>  3))
  csrw pmpaddr1, x3
  # pmpaddr2,0xfff73000 ~ 0xfff74000, NAPOT 模式，读写权限
  li x3, ( 0xfff73000  >>  2 | (0x1000-1)  >>  3))
  csrw pmpaddr2, x3
  # pmpaddr3,0xfffc0000 ~ 0xfffc2000, NAPOT 模式，读写权限
  li x3, ( 0xfffc0000  >>  2 | (0x2000-1)  >>  3))
  csrw pmpaddr3, x3
  # pmpaddr4,0xf0000000 ~ 0x100000000, NAPOT 模式，无任何权限
```

(续下页)

(接上页)

```
  li x3, ( 0xf0000000  >>  2 | (0x10000000-1)  >>  3))
  csrw pmpaddr4, x3
  # pmpaddr5,0x100000000 ~ 0xfffffffff, TOR 模式，无任何权限
  li x3, (0xfffffffff  >>  2)
  csrw pmpaddr5, x3
  # PMPCFG0，配置各表项执行权限/模式/lock 位，
  lock 为 1 时，该表项在机器模式下才有效
  li x3,0x88989b9b9b8f
  csrw pmpcfg0, x3
  # pmpaddr5,0x100000000 ~ 0xfffffffff, TOR 模式，0x100000000 <= addr <␣
↪0xfffffffff 时都会命中 pmpaddr5，但是 0xfffffff000 ~ 0xfffffffff 地址区间无法命中
pmpaddr5（C908X 中 PMP 的最小粒度为 4K），如果需要屏蔽 1T 空间的最后一个 4K 空间，需要再配
置一个 NAPOT 模式的表项。
```

17.4 高速缓存示例

17.4.1 高速缓存的开启示例

```
/*C908X will invalidate all I-cache automatically when reset*/
/*You can invalidate I-cache by yourself if necessary*/
/*Invalidate I-cache*/
li x3, 0x33
csrc mcor, x3
li x3, 0x11
csrs mcor, x3
// You can also use icache instructions to replace the invalidate sequence
// if xuantieisaee is enabled.
//icache.iall
//sync.is

/*Enable I-cache*/
li x3, 0x1
csrs mhcr, x3

/*C908X will invalidate all D-cache automatically when reset*/
/*You can invalidate D-cache by yourself if necessary*/
/*Invalidate D-cache*/
li x3, 0x33
csrc mcor, x3
li x3, 0x12
```

(续下页)

(接上页)

```
csrs mcor, x3

// You can also use dcache instructions to replace the invalidate sequence
// if xuantieisaee is enabled.
// dcache.iall
// sync.is

/*Enable D-cache*/
li x3, 0x2
csrs mhcr, x3
```

17.4.2 指令高速缓存与数据高速缓存的同步示例

CPU0

```
sd x3,0(x4) // a new instruction defined in x3
            // is stored to program memory address defined in x4.
dcache.cval1 r0 // clean the new instrcution to the shared L2 cache.
sync.s          // ensure completion of clean operation.
               // the dcache clean is not necessarily if INSDE is not enabled.
icache.iva r0  // invalid icache according to shareable configuration.
sync.s/fence.i  // ensure completion in all CPUs.
sd x5,0(x6)    // set flag to signal operation completion.
sync.is
jr x4 // jmp to new code
```

CPU1~CPU3

```
WAIT_FINISH:
  ld x7,0(x6)
  bne x7,x5, WAIT_FINISH // wait CPU0 modification finish.
  sync.is
  jr x4                 // jmp to new code
```

17.4.3 TLB 与数据高速缓存的同步示例

CPU0

```
sd x4,0(x3) // update a new translation table entry
sync.is/fence.i // ensure completion of update operation.
sfence.vma x5,x0 // invalid the TLB by va
sync.is/fence.i // ensure completion of TLB invalidation and
                // synchronises context
```

17.4.4 L2 cache partition 功能设定

第一步，设置 MCCR2.PAE=1，以打开 partition 功能。

第二步，配置 MLLWP 寄存器，置位每个 id 允许放到哪个 group 情况。该寄存器所有核共用 1 个。

例如，pid0 仅允许放到 group0，pid1 仅允许放到 group1，这样就是:

MLLWP[63:56]=8'h80

MLLWP[55:48]=8'h40

...

MLLWP[7:0]=8'h01

需要注意，每个 id 至少需要使能至少 1 个 group，否则赋值后寄存器会显示这个 id 会对每个 group 都使能。

第三步，配置 MLLCPID，每个核 1 个，表示当前核的 PID。一种简单的配置方法是:

核 0 的 MLLCPID[2:0]=0

核 1 的 MLLCPID[2:0]=1

以此类推。

最后，可以通过配置 MXSTATUS[9]（即 SPCE 位）决定是否开放 SL2WP 和 SL2PID。

17.5 同步原语示例

CPU0

```
li x1, 0x1
li x6, 0x0

ACQUIRE_LOCK:               // (x3) is the lock address. 0: Free; 1: Busy.
lr x4, 0(x3)                // Read lock
bnez x4, ACQUIRE_LOCK       // Try again if the lock is in use
sc x5, x1, 0(x3)            // Attempt to store new value
bne x6, x5, ACQUIRE_LOCK    // Try again if fail
sync.s
```

(续下页)

(接上页)

```
...                         // Critical section code
```

CPU1

```
sync.s/fence.i      // Ensure all operations are observed before clearing the lock.
sd x0, 0(x3)        // Clear the lock.
```

17.6 PLIC 设置示例

```
//Init id 1 machine mode int for hart 0
/*1.set hart threshold if needed*/
li x3, (plic_base_addr + 0x200000) // h0 mthreshold addr
li x4, 0xa //threshold value
sw x4,0x0(x3) // set hart0 threshold as 0xa

/*2.set priority for int id 1*/
li x3, (plic_base_addr + 0x0) // int id 1 prio addr
li x4, 0x1f // prio value
sw x4,0x4(x3) // init id1 priority as 0x1f

/*3.enable m-mode int id1 to hart*/
li x3, (plic_base_addr + 0x2000) // h0 mie0 addr
li x4, 0x2
sw x4,0x0(x3) // enable int id1 to hart0

/*4.set ip or wait external int*/
/*following code set ip*/
li x3, (plic_base_addr + 0x1000) // h0 mthreshold addr
li x4, 0x2 // id 1 pending
sw x4, 0x0(x3) // set int id1 pending

/*5.core enters interrupt handler, read PLIC_CLAIM and get ID*/

/*6.core takes interrupt*/

/*7.core needs to clear external interrupt source if LEVEL(not PULSE)
configured, then core writes ID to PLIC_CLAIM and exits interrupt*/
```

17.7 PMU 设置示例

```
/*1.inhibit counters counting*/
li x3, 0xffffffff
csrw mcountinhibit, x3

/*2.C908X will initial all pmu counters when reset*/
/*you can initial pmu counters manually if necessarily*/
csrw mcycle, x0
csrw minstret, x0
csrw mhpmcounter3, x0
......
csrw mhpmcounter31, x0

/*3.configure mhpmevent*/
li x3, 0x1
csrw mhpmevent3, x3 // mhpmcounter3 count event: L1 ICache Access Counter
li x3, 0x2
csrw mhpmevent4, x3 // mhpmcounter4 count event: L1 ICache Miss Counter
......
li x3, 0xf
csrw mhpmevent18, x3 // mhpmcounter18 count event: L1 DCache store miss Counter

/*4. configure mcounteren and scounteren*/
li x3, 0xffffffff
csrw mcounteren, x3 // enable super mode to read hpmcounter
li x3, 0xffffffff
csrw scounteren, x3 // enable user mode to read hpmcounter

/*5. enable counters to count when you want*/
csrw mcountinhibit, x0
```

18 附录 A 标准指令术语

C908X 实现了 RV64IMAFDCB[V] 指令集包，以下各章节按照不同指令集对每条指令做具体描述。

注：B 扩展、V 扩展和 CMO 拓展请参考对应的 RISC-V Spec。

18.1　附录 A-1 I 指令术语

以下是对 C908X 实现的 RISC-V I 指令集的具体描述，指令按英文字母顺序排列。

本节指令位宽默认为 32 位，但是系统在特定情况下会将某些指令汇编成 16 位的压缩指令，关于压缩指令具体描述请见*附录 A-6 C 指令术语*。

18.1.1　ADD—有符号加法指令

语法：

add rd, rs1, rs2

操作：

rd ← rs1 + rs2

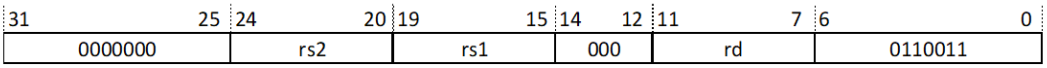
执行权限：

M mode/S mode/U mode

异常：

无

指令格式：



18.1.2　ADDI—有符号立即数加法指令

语法：

addi rd, rs1, imm12

操作：

rd ← rs1 + sign_extend(imm12)

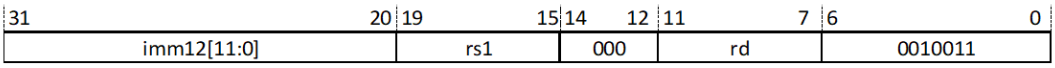
执行权限：

M mode/S mode/U mode

异常：

无

指令格式：



18.1.3　ADDIW—低 32 位有符号立即数加法指令

语法：

addiw rd, rs1, imm12

操作：

tmp[31:0] ← rs1[31:0] + sign_extend(imm12)[31:0]

rd ← sign_extend(tmp[31:0])

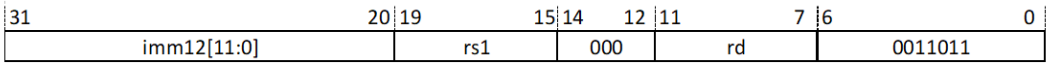
执行权限：

M mode/S mode/U mode

异常：

无

指令格式：



18.1.4　ADDW—低 32 位有符号加法指令

语法：

addw rd, rs1, rs2

操作：

tmp[31:0] ← rs1[31:0] + rs2[31:0]

rd ← sign_extend(tmp[31:0])

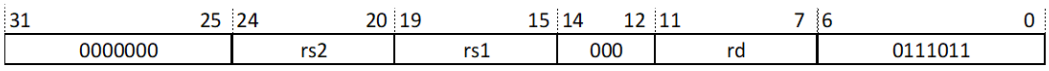
执行权限：

M mode/S mode/U mode

异常：

无

指令格式：



31 25	24 20	19 15	14 12	11 7	6 0
0000000	rs2	rs1	000	rd	0111011

18.1.5　AND——按位与指令

语法：

and rd, rs1, rs2

操作：

rd ← rs1 & rs2

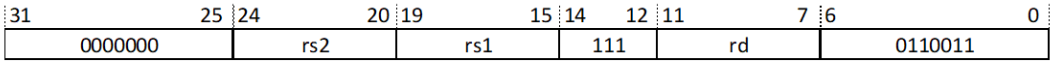
执行权限：

M mode/S mode/U mode

异常：

无

指令格式：



31 25	24 20	19 15	14 12	11 7	6 0
0000000	rs2	rs1	111	rd	0110011

18.1.6　ANDI——立即数按位与指令

语法：

andi rd, rs1, imm12

操作：

rd ← rs1 & sign_extend(imm12)

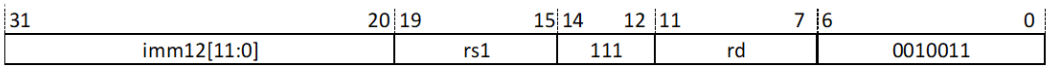
执行权限：

M mode/S mode/U mode

异常：

无

指令格式：



31 20	19 15	14 12	11 7	6 0
imm12[11:0]	rs1	111	rd	0010011

18.1.7　AUIPC——PC 高位立即数加法指令

语法：

auipc rd, imm20

操作：

rd ← current pc + sign_extend(imm20 << 12)

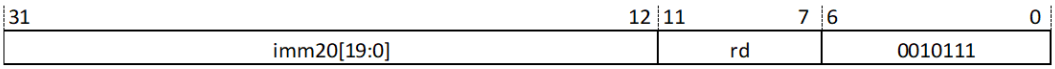
执行权限：

M mode/S mode/U mode

异常：

无

指令格式：



18.1.8　BEQ—相等分支指令

语法：

beq rs1, rs2, label

操作：

if (rs1 == rs2)

next pc = current pc +sign_extend(imm12 << 1)

else

next pc = current pc + 4

执行权限：

M mode/S mode/U mode

异常：

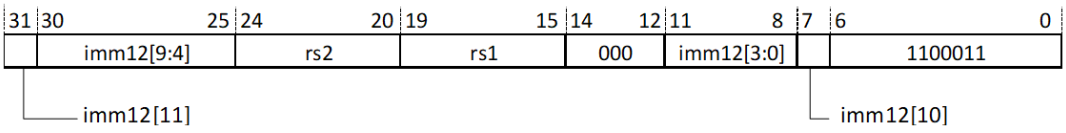
无

说明：

汇编器根据 label 算出 imm12

指令跳转范围为 ±4KB 地址空间

指令格式：



18.1.9　BGE—有符号大于等于分支指令

语法：

bge rs1, rs2, label

操作：

if (rs1 >= rs2)

next pc = current pc + sign_extend(imm12 << 1)

else

next pc = current pc + 4

执行权限：

M mode/S mode/U mode

异常：

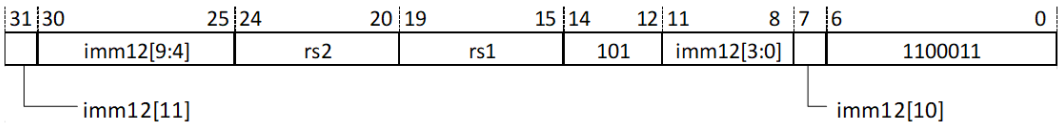
无

说明：

汇编器根据 label 算出 imm12

指令跳转范围为 ±4KB 地址空间

指令格式：



18.1.10　BGEU—无符号大于等于分支指令

语法：

bgeu rs1, rs2, label

操作：

if (rs1 >= rs2)

next pc = current pc + sign_extend(imm12 << 1)

else

next pc = current pc + 4

执行权限：

M mode/S mode/U mode

异常：

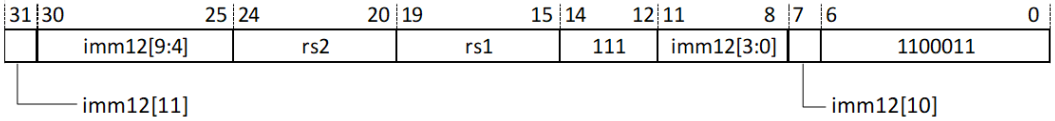
无

说明：

汇编器根据 label 算出 imm12

指令跳转范围为 ±4KB 地址空间

指令格式：



18.1.11　BLT—有符号小于分支指令

语法：

blt rs1, rs2, label

操作：

if (rs1 < rs2)

next pc = current pc + sign_extend(imm12 << 1)

else

next pc = current pc + 4

执行权限：

M mode/S mode/U mode

异常：

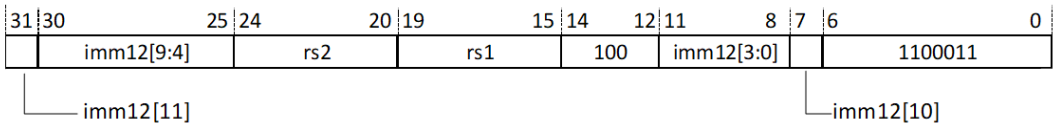
无

说明：

汇编器根据 label 算出 imm12

指令跳转范围为 ±4KB 地址空间

指令格式：



18.1.12　BLTU—无符号小于分支指令

语法：

bltu rs1, rs2, label

操作：

if (rs1 < rs2)

next pc = current pc + sign_extend(imm12 << 1)

else

next pc = current pc + 4

执行权限：

M mode/S mode/U mode

异常：

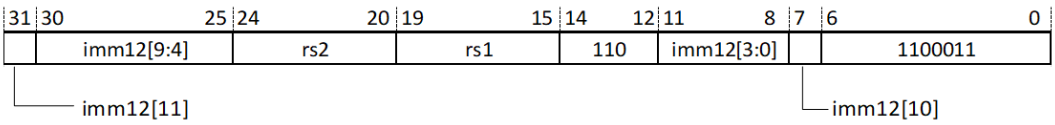
无

说明：

汇编器根据 label 算出 imm12

指令跳转范围为 ±4KB 地址空间

指令格式：



18.1.13　BNE——不等分支指令

语法：

bne rs1, rs2, label

操作：

if (rs1 != rs2)

next pc = current pc + sign_extend(imm12 << 1)

else

next pc = current pc + 4

执行权限：

M mode/S mode/U mode

异常：

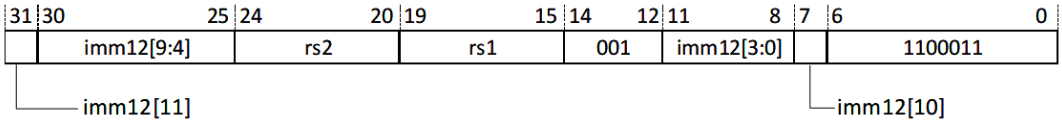
无

说明：

汇编器根据 label 算出 imm12

指令跳转范围为 ±4KB 地址空间

指令格式：



18.1.14　CSRRC—控制寄存器清零传送指令

语法：

csrrc rd, csr, rs1

操作：

rd ← csr

csr ← csr & (~rs1)

执行权限：

M mode/S mode/U mode

异常：

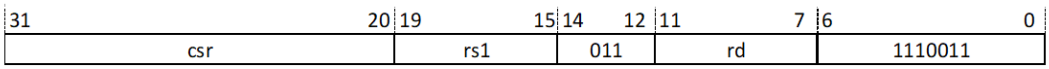
非法指令异常

说明：

各权限下允许访问的控制寄存器不同，具体请参考控制寄存器章节。

当 rs1=x0 时，该指令不产生写操作，不会产生写行为引发的异常。

指令格式：



18.1.15　CSRRCI—控制寄存器立即数清零传送指令

语法：

csrrci rd, csr, imm5

操作：

rd ← csr

csr ← csr & ~zero_extend(imm5)

执行权限：

M mode/S mode/U mode

异常：

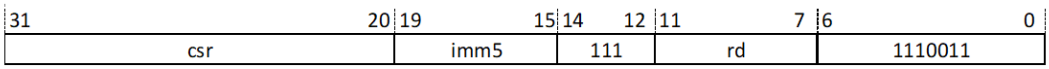
非法指令异常

说明：

各权限下允许访问的控制寄存器不同，具体请参考控制寄存器章节。

当 rs1=x0 时，该指令不产生写操作，不会产生写行为引发的异常。

指令格式：



18.1.16　CSRRS——控制寄存器置位传送指令

语法：

csrrs rd, csr, rs1

操作：

rd ← csr

csr ← csr | rs1

执行权限：

M mode/S mode/U mode

异常：

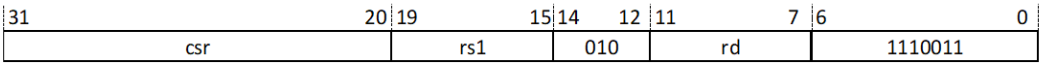
非法指令异常

说明：

各权限下允许访问的控制寄存器不同，具体请参考控制寄存器章节。

当 rs1=x0 时，该指令不产生写操作，不会产生写行为引发的异常。

指令格式：



18.1.17　CSRRSI——控制寄存器立即数置位传送指令

语法：

csrrsi rd, csr, imm5

操作：

rd ← csr

csr ← csr | zero_extend(imm5)

执行权限：

M mode/S mode/U mode

异常：

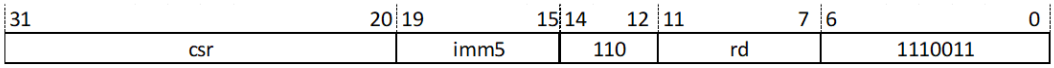
非法指令异常

说明：

各权限下允许访问的控制寄存器不同，具体请参考控制寄存器章节。

当 rs1=x0 时，该指令不产生写操作，不会产生写行为引发的异常。

指令格式：



31 20	19 15	14 12	11 7	6 0
csr	imm5	110	rd	1110011

18.1.18　CSRRW—控制寄存器读写传送指令

语法：

csrrw rd, csr, rs1

操作：

rd ← csr

csr ← rs1

执行权限：

M mode/S mode/U mode

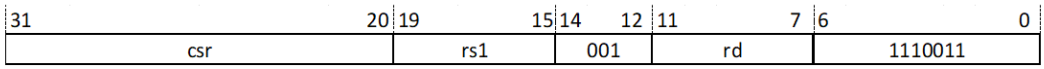
异常：

非法指令异常

说明：

各权限下允许访问的控制寄存器不同，具体请参考控制寄存器章节。

指令格式：



31 20	19 15	14 12	11 7	6 0
csr	rs1	001	rd	1110011

18.1.19　CSRRWI—控制寄存器立即数读写传送指令

语法：

csrrwi rd, csr, imm5

操作：

rd ← csr

csr ← zero_extend(imm5)

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

说明：

各权限下允许访问的控制寄存器不同，具体请参考控制寄存器章节。

指令格式：

31 20	19 15	14 12	11 7	6 0
csr	imm5	101	rd	1110011

18.1.20　EBREAK—断点指令

语法：

ebreak

操作：

产生断点异常或者进入调试模式

执行权限：

M mode/S mode/U mode

异常：

断点异常

指令格式：

31 20	19 15	14 12	11 7	6 0
000000000001	00000	000	00000	1110011

18.1.21　ECALL—环境异常指令

语法：

ecall

操作：

产生环境异常

执行权限：

M mode/S mode/U mode

异常：

用户模式环境调用异常、超级用户模式环境调用异常、机器模式环境调用异常

指令格式：

31 20	19 15	14 12	11 7	6 0
000000000000	00000	000	00000	1110011

18.1.22　FENCE—存储同步指令

语法：

fence iorw, iorw

操作：

保证该指令前序所有读写存储器或外设指令比该指令后序所有读写存储器或外设指令更早被观察到。

执行权限：

M mode/S mode/U mode

异常：

无

说明：

pi=1, so=1，指令语法为 fence i,o，以此类推

指令格式：

31 28	27	26	25	24	23	22	21	20	19 15	14 12	11 7	6 0
0000	pi	po	pr	pw	si	so	sr	sw	00000	000	00000	0001111

18.1.23　FENCE.I—指令流同步指令

语法：

fence.i

操作：

清空 icache, 保证该指令前序所有数据访存结果能够被指令后的取指操作访问到。

执行权限：

M mode/S mode/U mode

异常：

无

指令格式：

31 28	27 24	23 20	19 15	14 12	11 7	6 0
0000	0000	0000	00000	001	00000	0001111

18.1.24　JAL—直接跳转子程序指令

语法：

jal rd, label

操作：

next pc ← current pc + sign_extend(imm20 << 1)

rd ← currect pc + 4

执行权限：

M mode/S mode/U mode

异常：

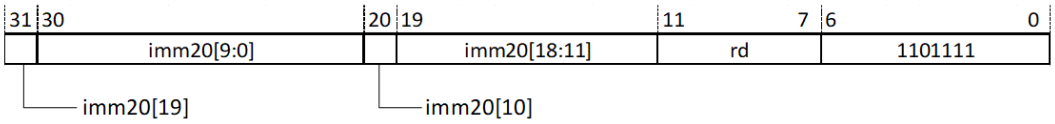
无

说明：

汇编器根据 label 算出 imm20

指令跳转范围为 ±1MB 地址空间

指令格式：



18.1.25　JALR——寄存器跳转子程序指令

语法：

jalr rd, rs1, imm12

操作：

next pc ← (rs1 + sign_extend(imm12)) & 64'hfffffffffffffffe

rd ← currect pc + 4

执行权限：

M mode/S mode/U mode

异常：

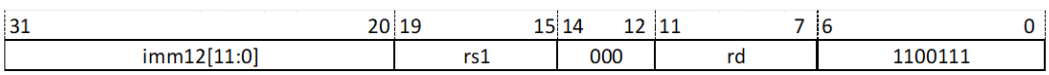
无

说明：

机器模式或者 MMU 关闭时，指令跳转范围为全部 1TB 地址空间

非机器模式且 MMU 打开时，指令跳转范围为全部 512GB 地址空间

指令格式：



18.1.26　LB——有符号扩展字节加载指令

语法：

lb rd, imm12(rs1)

操作：

address←rs1+sign_extend(imm12)

rd ← sign_extend(mem[address])

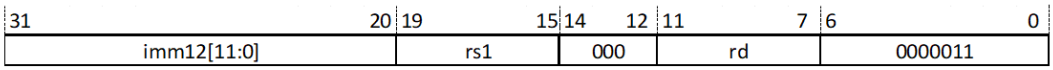
执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常

指令格式：



31 20	19 15	14 12	11 7	6 0
imm12[11:0]	rs1	000	rd	0000011

18.1.27　LBU——无符号扩展字节加载指令

语法：

lbu rd, imm12(rs1)

操作：

address←rs1+sign_extend(imm12)

rd ← zero_extend(mem[address])

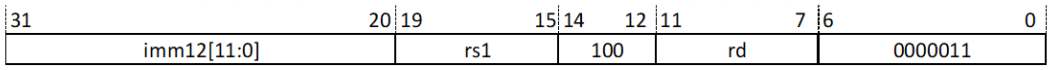
执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常

指令格式：



31 20	19 15	14 12	11 7	6 0
imm12[11:0]	rs1	100	rd	0000011

18.1.28　LD——双字加载指令

语法：

ld rd, imm12(rs1)

操作：

address←rs1+sign_extend(imm12)

rd ← mem[(address+7):address]

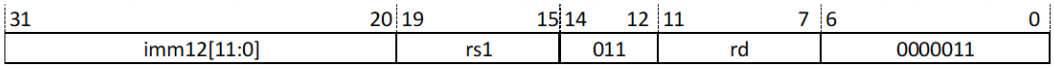
执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常

指令格式：



18.1.29　LH—有符号扩展半字加载指令

语法：

lh rd, imm12(rs1)

操作：

address←rs1+sign_extend(imm12)

rd ← sign_extend(mem[(address+1):address])

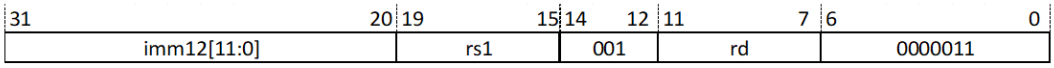
执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常

指令格式：



18.1.30　LHU—无符号扩展半字加载指令

语法：

lhu rd, imm12(rs1)

操作：

address←rs1+sign_extend(imm12)

rd ← zero_extend(mem[(address+1):address])

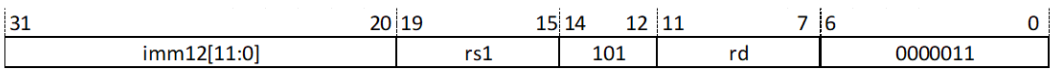
执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常

指令格式：



18.1.31　LUI—高位立即数装载指令

语法：

lui rd, imm20

操作：

rd←sign_extend(imm20 << 12)

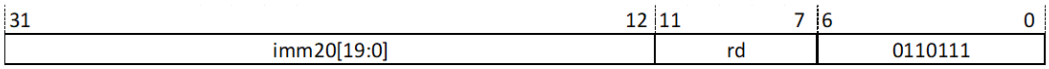
执行权限：

M mode/S mode/U mode

异常：

无

指令格式：



18.1.32　LW—有符号扩展字加载指令

语法：

lw rd, imm12(rs1)

操作：

address←rs1+sign_extend(imm12)

rd ← sign_extend(mem[(address+3):address])

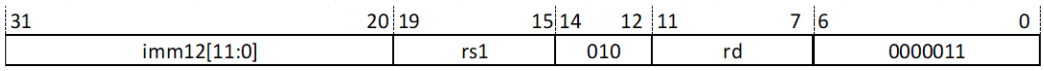
执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常

指令格式：



18.1.33　LWU—无符号扩展字加载指令

语法：

lwu rd, imm12(rs1)

操作：

address←rs1+sign_extend(imm12)

rd ← zero_extend(mem[(address+3):address])

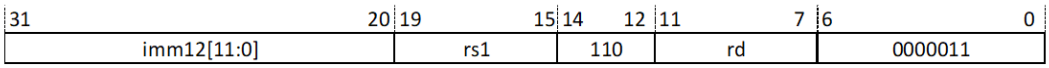
执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常

指令格式：



18.1.34　MRET——机器模式异常返回指令

语法：

mret

操作：

next pc← mepc

mstatus.mie ←mstatus.mpie

mstatus.mpie ←1

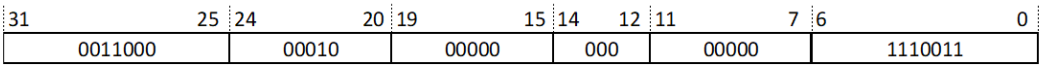
执行权限：

M mode

异常：

非法指令异常

指令格式：



18.1.35　OR——按位或指令

语法：

or rd, rs1, rs2

操作：

rd ← rs1 | rs2

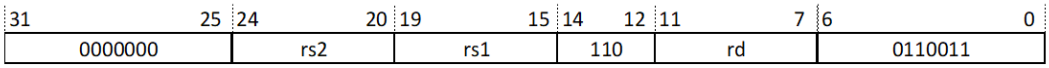
执行权限：

M mode/S mode/U mode

异常：

无

指令格式：



18.1.36　ORI—立即数按位或指令

语法：

ori rd, rs1, imm12

操作：

rd ← rs1 | sign_extend(imm12)

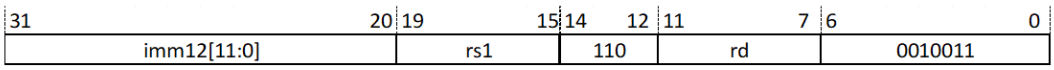
执行权限：

M mode/S mode/U mode

异常：

无

指令格式：



18.1.37　SB—字节存储指令

语法：

sb rs2, imm12(rs1)

操作：

address←rs1+sign_extend(imm12)

mem[:address] ← rs2[7:0]

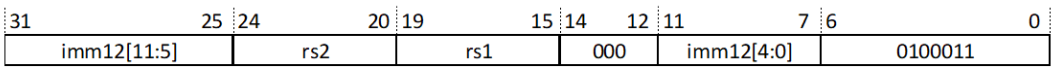
执行权限：

M mode/S mode/U mode

异常：

存储指令非对齐访问异常、存储指令访问错误异常、存储指令页面错误异常

指令格式：



18.1.38　SD—双字存储指令

语法：

sd rs2, imm12(rs1)

操作：

address←rs1+sign_extend(imm12)

mem[(address+7):address] ← rs2

执行权限：

M mode/S mode/U mode

异常：

存储指令非对齐访问异常、存储指令访问错误异常、存储指令页面错误异常

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
imm12[11:5]	rs2	rs1	011	imm12[4:0]	0100011

18.1.39　SFENCE.VMA——虚拟内存同步指令

语法：

sfence.vma rs1,rs2

操作：

用于虚拟内存的无效和同步操作

执行权限：

M mode/S mode

异常：

非法指令异常

说明：

mstatus.tvm=1，在超级用户模式下执行该指令引起非法指令异常。

rs1：虚拟地址，rs2：asid

- rs1=x0，rs2=x0 时，无效 TLB 中所有的表项。
- rs1!=x0，rs2=x0 时，无效 TLB 中所有命中 rs1 虚拟地址的表项。
- rs1=x0，rs2!=x0 时，无效 TB 中所有命中 rs2 进程号的表项。
- rs1!=x0，rs2!=x0 时，无效 TLB 中所有命中 rs1 虚拟地址和 rs2 进程号的表项。

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
0001001	rs2	rs1	000	00000	1110011

18.1.40　SH——半字存储指令

语法：

sh rs2, imm12(rs1)

操作：

address←rs1+sign_extend(imm12)

mem[(address+1):address] ← rs2[15:0]

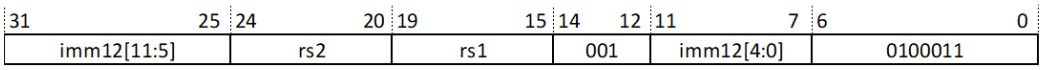
执行权限：

M mode/S mode/U mode

异常：

存储指令非对齐访问异常、存储指令访问错误异常、存储指令页面错误异常

指令格式：



18.1.41　SLL——逻辑左移指令

语法：

sll rd, rs1, rs2

操作：

rd← rs1 << rs2[5:0]

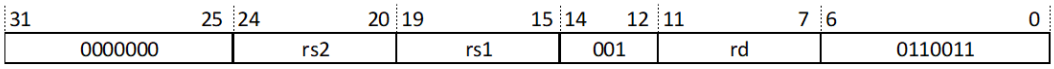
执行权限：

M mode/S mode/U mode

异常：

无

指令格式：



18.1.42　SLLI——立即数逻辑左移指令

语法：

slli rd, rs1, shamt6

操作：

rd← rs1 << shamt6

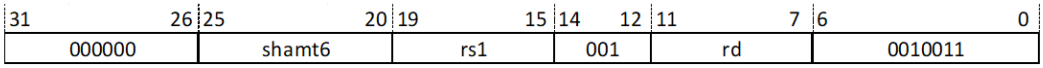
执行权限：

M mode/S mode/U mode

异常：

无

指令格式：



31 26	25 20	19 15	14 12	11 7	6 0
000000	shamt6	rs1	001	rd	0010011

18.1.43　SLLIW—低 32 位立即数逻辑左移指令

语法：

slliw rd, rs1, shamt5

操作：

tmp[31:0]←(rs1[31:0] << shamt5)[31:0]

rd← sign_extend(tmp[31:0])

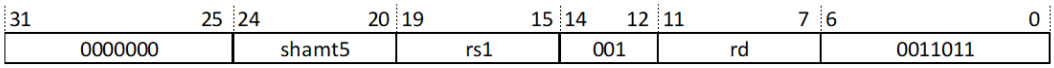
执行权限：

M mode/S mode/U mode

异常：

无

指令格式：



31 25	24 20	19 15	14 12	11 7	6 0
0000000	shamt5	rs1	001	rd	0011011

18.1.44　SLLW—低 32 位逻辑左移指令

语法：

sllw rd, rs1, rs2

操作：

tmp[31:0]← (rs1[31:0] << rs2[4:0])[31:0]

rd←sign_extend(tmp[31:0])

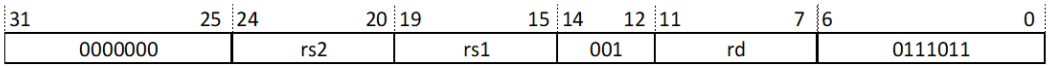
执行权限：

M mode/S mode/U mode

异常：

无

指令格式：



31 25	24 20	19 15	14 12	11 7	6 0
0000000	rs2	rs1	001	rd	0111011

18.1.45　SLT—有符号比较小于置位指令

语法：

slt rd, rs1, rs2

操作：

if (rs1 < rs2)

　　rd←1

else

　　rd←0

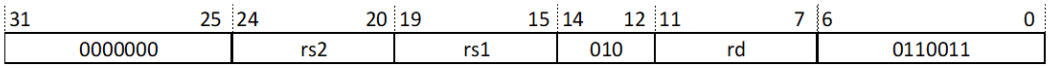
执行权限：

M mode/S mode/U mode

异常：

无

指令格式：



31 25	24 20	19 15	14 12	11 7	6 0
0000000	rs2	rs1	010	rd	0110011

18.1.46　SLTI—有符号立即数比较小于置位指令

语法：

slti rd, rs1, imm12

操作：

if (rs1 <sign_extend(imm12))

　　rd←1

else

　　rd←0

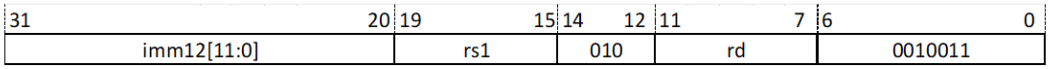
执行权限：

M mode/S mode/U mode

异常：

无

指令格式：



31 20	19 15	14 12	11 7	6 0
imm12[11:0]	rs1	010	rd	0010011

18.1.47　SLTIU—无符号立即数比较小于置位指令

语法：

sltiu rd, rs1, imm12

操作：

if (rs1 < sign_extend(imm12))

　　rd←1

else

　　rd←0

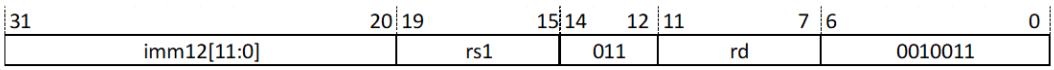
执行权限：

M mode/S mode/U mode

异常：

无

指令格式：



18.1.48　SLTU—无符号比较小于置位指令

语法：

sltu rd, rs1, rs2

操作：

if (rs1 < rs2)

　　rd←1

else

　　rd←0

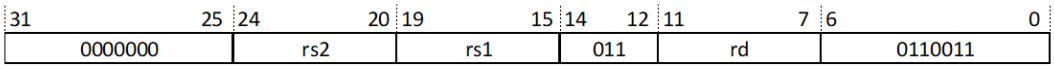
执行权限：

M mode/S mode/U mode

异常：

无

指令格式：



18.1.49　SRA——算数右移指令

语法：

sra rd, rs1, rs2

操作：

rd←rs1 >> rs2[5:0]

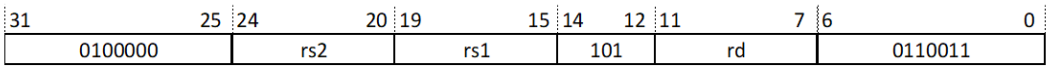
执行权限：

M mode/S mode/U mode

异常：

无

指令格式：



18.1.50　SRAI——立即数算数右移指令

语法：

srai rd, rs1, shamt6

操作：

rd← rs1 >> shamt6

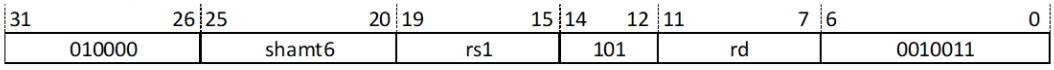
执行权限：

M mode/S mode/U mode

异常：

无

指令格式：



18.1.51　SRAIW——低 32 位立即数算数右移指令

语法：

sraiw rd, rs1, shamt5

操作：

tmp[31:0]←(rs1[31:0] >> shamt5)[31:0]

rd← sign_extend(tmp[31:0])

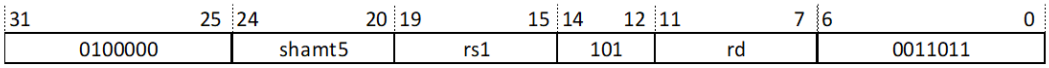
执行权限：

M mode/S mode/U mode

异常：

无

指令格式：



18.1.52　SRAW——低 32 位算数右移指令

语法：

sraw rd, rs1, rs2

操作：

tmp←(rs1[31:0] >> rs2[4:0])[31:0]

rd←sign_extend(tmp)

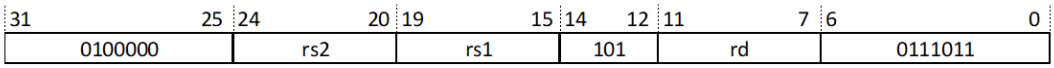
执行权限：

M mode/S mode/U mode

异常：

无

指令格式：



18.1.53　SRET——超级用户模式异常返回指令

语法：

sret

操作：

next pc← sepc

sstatus.sie ←sstatus.spie

sstatus.spie ←1

执行权限：

S mode

异常：

非法指令异常

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
0001000	00010	00000	000	00000	1110011

18.1.54　SRL——逻辑右移指令

语法：

srl rd, rs1, rs2

操作：

rd←rs1 >> rs2[5:0]

执行权限：

M mode/S mode/U mode

异常：

无

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
0000000	rs2	rs1	101	rd	0110011

18.1.55　SRLI——立即数逻辑右移指令

语法：

srli rd, rs1, shamt6

操作：

rd← rs1 >> shamt6

执行权限：

M mode/S mode/U mode

异常：

无

指令格式：

31 26	25 20	19 15	14 12	11 7	6 0
000000	shamt6	rs1	101	rd	0010011

18.1.56　SRLIW——低 32 位立即数逻辑右移指令

语法：

srliw rd, rs1, shamt5

操作：

tmp[31:0]←(rs1[31:0] >> shamt5)[31:0]

rd← sign_extend(tmp[31:0])

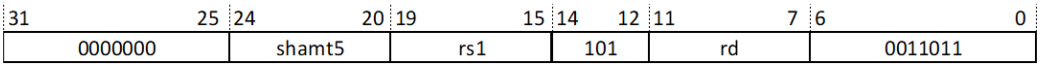
执行权限：

M mode/S mode/U mode

异常：

无

指令格式：



18.1.57　SRLW——低 32 位逻辑右移指令

语法：

srlw rd, rs1, rs2

操作：

tmp←(rs1[31:0] >> rs2[4:0])[31:0]

rd←sign_extend(tmp)

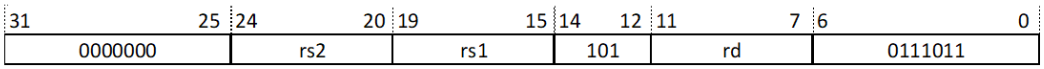
执行权限：

M mode/S mode/U mode

异常：

无

指令格式：



18.1.58　SUB——有符号减法指令

语法：

sub rd, rs1, rs2

操作：

rd ← rs1 - rs2

执行权限：

M mode/S mode/U mode

异常：

无

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
0100000	rs2	rs1	000	rd	0110011

18.1.59　SUBW——低 32 位有符号减法指令

语法：

subw rd, rs1, rs2

操作：

tmp[31:0] ← rs1[31:0] - rs2[31:0]

rd ← sign_extend(tmp[31:0])

执行权限：

M mode/S mode/U mode

异常：

无

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
0100000	rs2	rs1	000	rd	0111011

18.1.60　SW——字存储指令

语法：

sw rs2, imm12(rs1)

操作：

address←rs1+sign_extend(imm12)

mem[(address+3):address] ← rs2[31:0]

执行权限：

M mode/S mode/U mode

异常：

存储指令非对齐访问异常、存储指令访问错误异常、存储指令页面错误异常

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
imm12[11:5]	rs2	rs1	010	imm12[4:0]	0100011

18.1.61　WFI——进入低功耗模式指令

语法：

wfi

操作：

处理器进入低功耗模式，此时 CPU 时钟关闭，大部分外设时钟也关闭

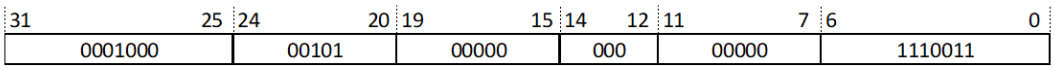
执行权限：

M mode/S mode

异常：

无

指令格式：



18.1.62　XOR——按位异或指令

语法：

xor rd, rs1, rs2

操作：

rd ← rs1 ^ rs2

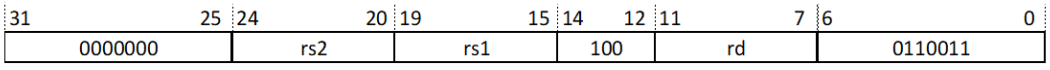
执行权限：

M mode/S mode/U mode

异常：

无

指令格式：



18.1.63　XORI——立即数按位异或指令

语法：

xori rd, rs1, imm12

操作：

rd ← rs1 & sign_extend(imm12)

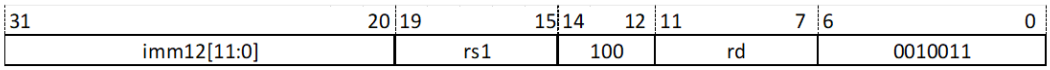
执行权限：

M mode/S mode/U mode

异常：

无

指令格式：



31 20	19 15	14 12	11 7	6 0
imm12[11:0]	rs1	100	rd	0010011

18.2　附录 A-2 M 指令术语

以下是对 C908X 实现的 RISC-V M 指令集的具体描述，本节指令位宽为 32 位，指令按英文字母顺序排列，

18.2.1　DIV—有符号除法指令

语法：

div rd, rs1, rs2

操作：

rd ← rs1 / rs2

执行权限：

M mode/S mode/U mode

异常：

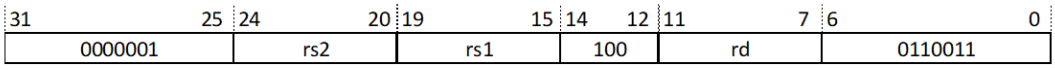
无

说明：

除数是 0 时，除法结果为 0xffffffffffffffff

产生 overflow 时，除法结果为 0x8000000000000000

指令格式：



31 25	24 20	19 15	14 12	11 7	6 0
0000001	rs2	rs1	100	rd	0110011

18.2.2　DIVU—无符号除法指令

语法：

divu rd, rs1, rs2

操作：

rd ← rs1 / rs2

执行权限：

M mode/S mode/U mode

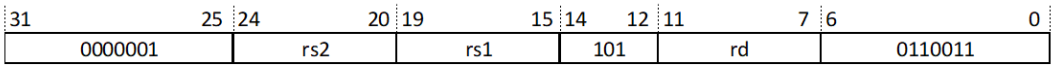
异常：

无

说明：

除数是 0 时，除法结果为 0xffffffffffffffff

指令格式：



18.2.3　DIVUW—低 32 位无符号除法指令

语法：

divuw rd, rs1, rs2

操作：

tmp[31:0] ← (rs1[31:0] / rs2[31:0])[31:0]

rd←sign_extend(tmp[31:0])

执行权限：

M mode/S mode/U mode

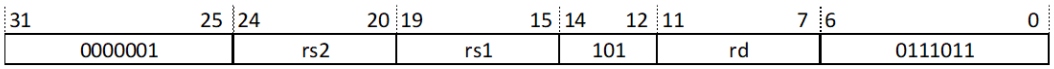
异常：

无

说明：

除数是 0 时，除法结果为 0xffffffffffffffff

指令格式：



18.2.4　DIVW—低 32 位有符号除法指令

语法：

divw rd, rs1, rs2

操作：

tmp[31:0] ← (rs1[31:0] / rs2[31:0])[31:0]

rd←sign_extend(tmp[31:0])

执行权限：

M mode/S mode/U mode

异常：

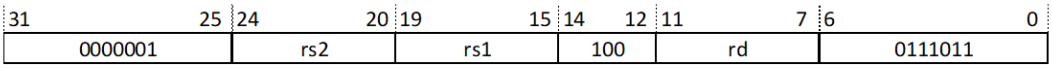
无

说明：

除数是 0 时，除法结果为 0xffffffffffffffff

产生 overflow 时，除法结果为 0xffffffff80000000

指令格式：



18.2.5　MUL——有符号乘法指令

语法：

mul rd, rs1, rs2

操作：

rd ← (rs1 * rs2)[63:0]

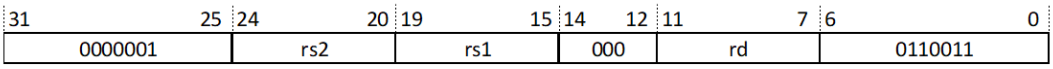
执行权限：

M mode/S mode/U mode

异常：

无

指令格式：



18.2.6　MULH——有符号乘法取高位指令

语法：

mulh rd, rs1, rs2

操作：

rd ← (rs1 * rs2)[127:64]

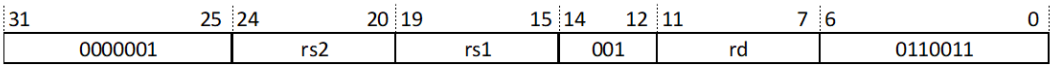
执行权限：

M mode/S mode/U mode

异常：

无

指令格式：



18.2.7　MULHSU——有符号无符号乘法取高位指令

语法：

mulusu rd, rs1, rs2

操作：

rd ← (rs1 * rs2)[127:64]

执行权限：

M mode/S mode/U mode

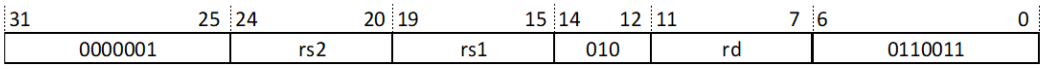
异常：

无

说明：

rs1 有符号数，rs2 无符号数

指令格式：



18.2.8　MULHU——无符号乘法取高位指令

语法：

mulhu rd, rs1, rs2

操作：

rd ← (rs1 * rs2)[127:64]

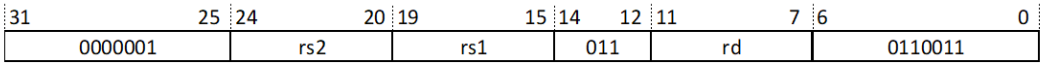
执行权限：

M mode/S mode/U mode

异常：

无

指令格式：



18.2.9　MULW——低 32 位有符号乘法指令

语法：

mulw rd, rs1, rs2

操作：

tmp ← (rs1[31:0] * rs2[31:0])[31:0]

rd ← sign_extend(tmp[31:0])

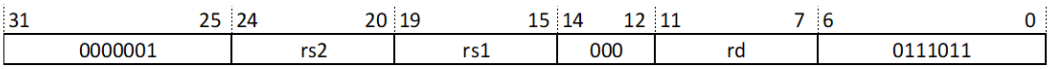
执行权限：

M mode/S mode/U mode

异常：

无

指令格式：



18.2.10　REM——有符号取余指令

语法：

rem rd, rs1, rs2

操作：

rd ← rs1 % rs2

执行权限：

M mode/S mode/U mode

异常：

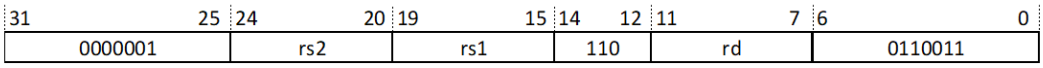
无

说明：

除数是 0 时，求余结果为被除数

产生 overflow 时，余数结果为 0x0

指令格式：



18.2.11　REMU——无符号取余指令

语法：

remu rd, rs1, rs2

操作：

rd ← rs1 % rs2

执行权限：

M mode/S mode/U mode

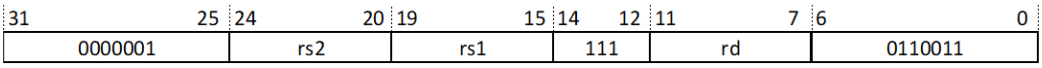
异常：

无

说明：

除数是 0 时，求余结果为被除数

指令格式：



18.2.12　REMUW——低 32 位无符号取余指令

语法：

remw rd, rs1, rs2

操作：

tmp ← (rs1[31:0] % rs2[31:0])[31:0]

rd ← sign_extend(tmp)

执行权限：

M mode/S mode/U mode

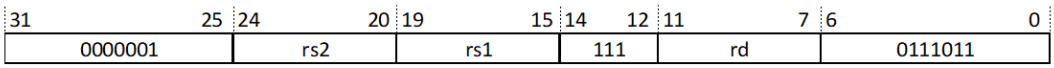
异常：

无

说明：

除数是 0 时，求余结果是对被除数 [31] 位符号位扩展后的结果

指令格式：



18.2.13　REMW——低 32 位有符号取余指令

语法：

remw rd, rs1, rs2

操作：

tmp[31:0] ← (rs1[31:0] % rs2[31:0])[31:0]

rd ← sign_extend(tmp[31:0])

执行权限：

M mode/S mode/U mode

异常：

无

说明：

除数是 0 时，求余结果是对被除数 [31] 位符号位扩展后的结果

产生 overflow 时，余数结果为 0x0

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
0000001	rs2	rs1	110	rd	0111011

18.3　附录 A-3 A 指令术语

以下是对 C908X 实现的 RISC-V A 指令的具体描述，本节指令位宽为 32 位，指令按英文字母顺序排列。

18.3.1　AMOADD.D——原子加法指令

语法：

amoadd.d.aqrl rd, rs2, (rs1)

操作：

rd ← mem[rs1+7: rs1]

mem[rs1+7:rs1] ← mem[rs1+7:rs1] + rs2

执行权限：

M mode/S mode/U mode

异常：

原子指令非对齐访问异常、原子指令访问错误异常、原子指令页面错误异常

影响标志位：

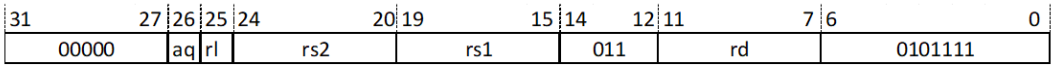
无

说明：

aq 位和 rl 位决定该指令前序和后序访问存储器指令的执行顺序：

- aq=0,rl=0: 对应的汇编指令 amoadd.d rd, rs2, (rs1)。
- aq=0,rl=1: 对应的汇编指令 amoadd.d.rl rd, rs2, (rs1)，该指令前序所有访问存储的指令的结果必须在该指令执行之前被观察到。
- aq=1,rl=0: 对应的汇编指令 amoadd.d.aq rd, rs2, (rs1)，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。
- aq=1,rl=1: 对应的汇编指令 amoadd.d.aqrl rd, rs2, (rs1)，该指令前序所有访问存储的指令的结果必须在该指令之前被观察到，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。

指令格式：



18.3.2　AMOADD.W——低 32 位原子加法指令

语法：

amoadd.w.aqrl rd, rs2, (rs1)

操作：

rd ←sign_extend(mem[rs1+3: rs1])

mem[rs1+3:rs1]← mem[rs1+3:rs1] + rs2[31:0]

执行权限：

M mode/S mode/U mode

异常：

原子指令非对齐访问异常、原子指令访问错误异常、原子指令页面错误异常

影响标志位：

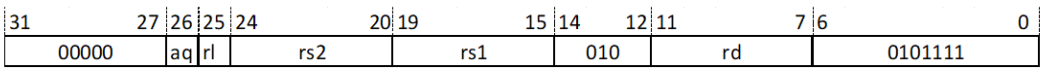
无

说明：

aq 位和 rl 位决定该指令前序和后序访问存储器指令的执行顺序：

- aq=0,rl=0: 对应的汇编指令 amoadd.w rd, rs2, (rs1)。
- aq=0,rl=1: 对应的汇编指令 amoadd.w.rl rd, rs2, (rs1)，该指令前序所有访问存储的指令的结果必须在该指令执行之前被观察到。
- aq=1,rl=0: 对应的汇编指令 amoadd.w.aq rd, rs2, (rs1)，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。
- aq=1,rl=1: 对应的汇编指令 amoadd.w.aqrl rd, rs2, (rs1)，该指令前序所有访问存储的指令的结果必须在该指令之前被观察到，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。

指令格式：



18.3.3　AMOAND.D——原子按位与指令

语法：

amoand.d.aqrl rd, rs2, (rs1)

操作：

rd ← mem[rs1+7: rs1]

mem[rs1+7:rs1] ← mem[rs1+7:rs1] & rs2

执行权限：

M mode/S mode/U mode

异常：

原子指令非对齐访问异常、原子指令访问错误异常、原子指令页面错误异常

影响标志位：

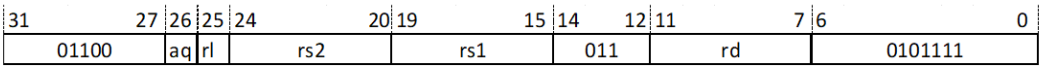
无

说明：

aq 位和 rl 位决定该指令前序和后序访问存储器指令的执行顺序：

- aq=0,rl=0: 对应的汇编指令 amoand.d rd, rs2, (rs1)。
- aq=0,rl=1: 对应的汇编指令 amoand.d.rl rd, rs2, (rs1)，该指令前序所有访问存储的指令的结果必须在该指令执行之前被观察到。
- aq=1,rl=0: 对应的汇编指令 amoand.d.aq rd, rs2, (rs1)，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。
- aq=1,rl=1: 对应的汇编指令 amoand.d.aqrl rd, rs2, (rs1)，该指令前序所有访问存储的指令的结果必须在该指令之前被观察到，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。

指令格式：



18.3.4　AMOAND.W—低 32 位原子按位与指令

语法：

amoand.w.aqrl rd, rs2, (rs1)

操作：

rd ← sign_extend(mem[rs1+3: rs1])

mem[rs1+3:rs1] ← mem[rs1+3:rs1] & rs2[31:0]

执行权限：

M mode/S mode/U mode

异常：

原子指令非对齐访问异常、原子指令访问错误异常、原子指令页面错误异常

影响标志位：

无

说明：

aq 位和 rl 位决定该指令前序和后序访问存储器指令的执行顺序：

- aq=0,rl=0: 对应的汇编指令 amoand.w rd, rs2, (rs1)。

- aq=0,rl=1: 对应的汇编指令 amoand.w.rl rd, rs2, (rs1)，该指令前序所有访问存储的指令的结果必须在该指令执行之前被观察到。
- aq=1,rl=0: 对应的汇编指令 amoand.w.aq rd, rs2, (rs1)，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。
- aq=1,rl=1: 对应的汇编指令 amoand.w.aqrl rd, rs2, (rs1)，该指令前序所有访问存储的指令的结果必须在该指令之前被观察到，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。

指令格式：

31 27	26	25	24 20	19 15	14 12	11 7	6 0
01100	aq	rl	rs2	rs1	010	rd	0101111

18.3.5　AMOMAX.D——原子有符号取最大值指令

语法：

amomax.d.aqrl rd, rs2, (rs1)

操作：

rd ← mem[rs1+7: rs1]

mem[rs1+7:rs1] ← max(mem[rs1+7:rs1], rs2)

执行权限：

M mode/S mode/U mode

异常：

原子指令非对齐访问异常、原子指令访问错误异常、原子指令页面错误异常

影响标志位：

无

说明：

aq 位和 rl 位决定该指令前序和后序访问存储器指令的执行顺序：

- aq=0,rl=0: 对应的汇编指令 amomax.d rd, rs2, (rs1)。
- aq=0,rl=1: 对应的汇编指令 amomax.d.rl rd, rs2, (rs1)，该指令前序所有访问存储的指令的结果必须在该指令执行之前被观察到。
- aq=1,rl=0: 对应的汇编指令 amomax.d.aq rd, rs2, (rs1)，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。
- aq=1,rl=1: 对应的汇编指令 amomax.d.aqrl rd, rs2, (rs1)，该指令前序所有访问存储的指令的结果必须在该指令之前被观察到，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。

指令格式：

31 27	26	25	24 20	19 15	14 12	11 7	6 0
10100	aq	rl	rs2	rs1	011	rd	0101111

18.3.6　AMOMAX.W——低 32 位原子有符号取最大值指令

语法：

amomax.w.aqrl rd, rs2, (rs1)

操作：

rd ← sign_extend(mem[rs1+3: rs1])

mem[rs1+3:rs1]← max(mem[rs1+3:rs1], rs2[31:0])

执行权限：

M mode/S mode/U mode

异常：

原子指令非对齐访问异常、原子指令访问错误异常、原子指令页面错误异常

影响标志位：

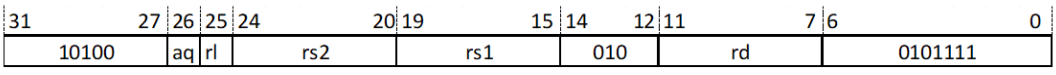
无

说明：

aq 位和 rl 位决定该指令前序和后序访问存储器指令的执行顺序：

- aq=0,rl=0: 对应的汇编指令 amomax.w rd, rs2, (rs1)。
- aq=0,rl=1: 对应的汇编指令 amomax.w.rl rd, rs2, (rs1)，该指令前序所有访问存储的指令的结果必须在该指令执行之前被观察到。
- aq=1,rl=0: 对应的汇编指令 amomax.w.aq rd, rs2, (rs1)，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。
- aq=1,rl=1: 对应的汇编指令 amomax.w.aqrl rd, rs2, (rs1)，该指令前序所有访问存储的指令的结果必须在该指令之前被观察到，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。

指令格式：



18.3.7　AMOMAXU.D——原子无符号取最大值指令

语法：

amomaxu.d.aqrl rd, rs2, (rs1)

操作：

rd ← mem[rs1+7: rs1]

mem[rs1+7:rs1] ← max(mem[rs1+7:rs1], rs2)

执行权限：

M mode/S mode/U mode

异常：

原子指令非对齐访问异常、原子指令访问错误异常、原子指令页面错误异常

影响标志位：

无

说明：

aq 位和 rl 位决定该指令前序和后序访问存储器指令的执行顺序：

- aq=0,rl=0: 对应的汇编指令 amomaxu.d rd, rs2, (rs1)。
- aq=0,rl=1：对应的汇编指令 amomaxu.d.rl rd, rs2, (rs1)，该指令前序所有访问存储的指令的结果必须在该指令执行之前被观察到。
- aq=1,rl=0：对应的汇编指令 amomaxu.d.aq rd, rs2, (rs1)，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。
- aq=1,rl=1: 对应的汇编指令 amomaxu.d.aqrl rd, rs2, (rs1)，该指令前序所有访问存储的指令的结果必须在该指令之前被观察到，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。

指令格式：

31 27	26	25	24 20	19 15	14 12	11 7	6 0
11100	aq	rl	rs2	rs1	011	rd	0101111

18.3.8　AMOMAXU.W——低 32 位原子无符号取最大值指令

语法：

amomaxu.w.aqrl rd, rs2, (rs1)

操作：

rd ← zero_extend(mem[rs1+3: rs1])

mem[rs1+3:rs1] ← max(mem[rs1+3:rs1], rs2[31:0])

执行权限：

M mode/S mode/U mode

异常：

原子指令非对齐访问异常、原子指令访问错误异常、原子指令页面错误异常

影响标志位：

无

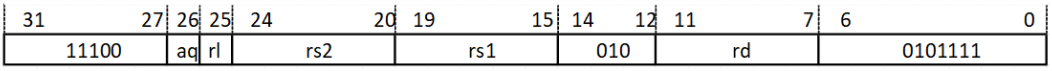
说明：

aq 位和 rl 位决定该指令前序和后序访问存储器指令的执行顺序：

- aq=0,rl=0: 对应的汇编指令 amomaxu.w rd, rs2, (rs1)。
- aq=0,rl=1：对应的汇编指令 amomaxu.w.rl rd, rs2, (rs1)，该指令前序所有访问存储的指令的结果必须在该指令执行之前被观察到。

- aq=1,rl=0：对应的汇编指令 amomaxu.w.aq rd, rs2, (rs1)，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。
- aq=1,rl=1：对应的汇编指令 amomaxu.w.aqrl rd, rs2, (rs1)，指令前序所有访问存储的指令的结果必须在该指令执行之前被观察到，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。

指令格式：



18.3.9　AMOMIN.D——原子有符号取最小值指令

语法：

amomin.d.aqrl rd, rs2, (rs1)

操作：

rd ← mem[rs1+7: rs1]

mem[rs1+7:rs1] ← min(mem[rs1+7:rs1],rs2)

执行权限：

M mode/S mode/U mode

异常：

原子指令非对齐访问异常、原子指令访问错误异常、原子指令页面错误异常

影响标志位：

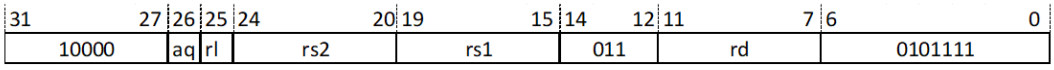
无

说明：

aq 位和 rl 位决定该指令前序和后序访问存储器指令的执行顺序：

- aq=0,rl=0: 对应的汇编指令 amomin.d rd, rs2, (rs1)。
- aq=0,rl=1：对应的汇编指令 amomin.d.rl rd, rs2, (rs1)，该指令前序所有访问存储的指令的结果必须在该指令执行之前被观察到。
- aq=1,rl=0：对应的汇编指令 amomin.d.aq rd, rs2, (rs1)，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。
- aq=1,rl=1：对应的汇编指令 amomin.d.aqrl rd, rs2, (rs1)，指令前序所有访问存储的指令的结果必须在该指令执行之前被观察到，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。

指令格式：



18.3.10　AMOMIN.W——低 32 位原子有符号取最小值指令

语法：

amomin.w.aqrl rd, rs2, (rs1)

操作：

rd ← sign_extend(mem[rs1+3: rs1])

mem[rs1+3:rs1] ← min(mem[rs1+3:rs1], rs2[31:0])

执行权限：

M mode/S mode/U mode

异常：

原子指令非对齐访问异常、原子指令访问错误异常、原子指令页面错误异常

影响标志位：

无

说明：

aq 位和 rl 位决定该指令前序和后序访问存储器指令的执行顺序：

- aq=0,rl=0: 对应的汇编指令 amomin.w rd, rs2, (rs1)。
- aq=0,rl=1：对应的汇编指令 amomin.w.rl rd, rs2, (rs1)，该指令前序所有访问存储的指令的结果必须在该指令执行之前被观察到。
- aq=1,rl=0：对应的汇编指令 amomin.w.aq rd, rs2, (rs1)，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。
- aq=1,rl=1：对应的汇编指令 amomin.w.aqrl rd, rs2, (rs1)，指令前序所有访问存储的指令的结果必须在该指令执行之前被观察到，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。

指令格式：

31 27	26	25	24 20	19 15	14 12	11 7	6 0
10000	aq	rl	rs2	rs1	010	rd	0101111

18.3.11　AMOMINU.D——原子无符号取最小值指令

语法：

amominu.d.aqrl rd, rs2, (rs1)

操作：

rd ← mem[rs1+7: rs1]

mem[rs1+7:rs1] ← min(mem[rs1+7:rs1], rs2)

执行权限：

M mode/S mode/U mode

异常：

原子指令非对齐访问异常、原子指令访问错误异常、原子指令页面错误异常

影响标志位：

无

说明：

aq 位和 rl 位决定该指令前序和后序访问存储器指令的执行顺序：

- aq=0,rl=0: 对应的汇编指令 amominu.d rd, rs2, (rs1)。
- aq=0,rl=1：对应的汇编指令 amominu.d.rl rd, rs2, (rs1)，该指令前序所有访问存储的指令的结果必须在该指令执行之前被观察到。
- aq=1,rl=0：对应的汇编指令 amominu.d.aq rd, rs2, (rs1)，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。
- aq=1,rl=1：对应的汇编指令 amominu.d.aqrl rd, rs2, (rs1)，指令前序所有访问存储的指令的结果必须在该指令执行之前被观察到，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。

指令格式：

31 27	26	25	24 20	19 15	14 12	11 7	6 0
11000	aq	rl	rs2	rs1	011	rd	0101111

18.3.12　AMOMINU.W—低 32 位原子无符号取最小值指令

语法：

amominu.w.aqrl rd, rs2, (rs1)

操作：

rd ← sign_extend(mem[rs1+3: rs1])

mem[rs1+3:rs1] ← min(mem[rs1+3:rs1], rs2[31:0])

执行权限：

M mode/S mode/U mode

异常：

原子指令非对齐访问异常、原子指令访问错误异常、原子指令页面错误异常

影响标志位：

无

说明：

aq 位和 rl 位决定该指令前序和后序访问存储器指令的执行顺序：

- aq=0,rl=0: 对应的汇编指令 amominu.w rd, rs2, (rs1)。
- aq=0,rl=1：对应的汇编指令 amominu.w.rl rd, rs2, (rs1)，该指令前序所有访问存储的指令的结果必须在该指令执行之前被观察到。

- aq=1,rl=0：对应的汇编指令 amominu.w.aq rd, rs2, (rs1)，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。
- aq=1,rl=1：对应的汇编指令 amominu.w.aqrl rd, rs2, (rs1)，指令前序所有访问存储的指令的结果必须在该指令执行之前被观察到，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。

指令格式：

31 27	26	25	24 20	19 15	14 12	11 7	6 0
11000	aq	rl	rs2	rs1	010	rd	0101111

18.3.13　AMOOR.D——原子按位或指令

语法：

amoor.d.aqrl rd, rs2, (rs1)

操作：

rd ← mem[rs1+7: rs1]

mem[rs1+7:rs1] ← mem[rs1+7:rs1] | rs2

执行权限：

M mode/S mode/U mode

异常：

原子指令非对齐访问异常、原子指令访问错误异常、原子指令页面错误异常

影响标志位：

无

说明：

aq 位和 rl 位决定该指令前序和后序访问存储器指令的执行顺序：

- aq=0,rl=0: 对应的汇编指令 amoor.d rd, rs2, (rs1)。
- aq=0,rl=1：对应的汇编指令 amoor.d.rl rd, rs2, (rs1)，该指令前序所有访问存储的指令的结果必须在该指令执行之前被观察到。
- aq=1,rl=0：对应的汇编指令 amoor.d.aq rd, rs2, (rs1)，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。
- aq=1,rl=1：对应的汇编指令 amoor.d.aqrl rd, rs2, (rs1)，指令前序所有访问存储的指令的结果必须在该指令执行之前被观察到，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。

指令格式：

31 27	26	25	24 20	19 15	14 12	11 7	6 0
01000	aq	rl	rs2	rs1	011	rd	0101111

18.3.14　AMOOR.W——低 32 位原子按位或指令

语法：

amoor.w.aqrl rd, rs2, (rs1)

操作：

rd ← sign_extend(mem[rs1+3: rs1])

mem[rs1+3:rs1] ← mem[rs1+3:rs1] | rs2[31:0]

执行权限：

M mode/S mode/U mode

异常：

原子指令非对齐访问异常、原子指令访问错误异常、原子指令页面错误异常

影响标志位：

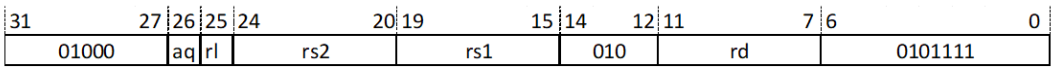
无

说明：

aq 位和 rl 位决定该指令前序和后序访问存储器指令的执行顺序：

- aq=0,rl=0: 对应的汇编指令 amoor.w rd, rs2, (rs1)。
- aq=0,rl=1：对应的汇编指令 amoor.w.rl rd, rs2, (rs1)，该指令前序所有访问存储的指令的结果必须在该指令执行之前被观察到。
- aq=1,rl=0：对应的汇编指令 amoor.w.aq rd, rs2, (rs1)，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。
- aq=1,rl=1：对应的汇编指令 amoor.w.aqrl rd, rs2, (rs1)，指令前序所有访问存储的指令的结果必须在该指令执行之前被观察到，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。

指令格式：



18.3.15　AMOSWAP.D——原子交换指令

语法：

amoswap.d.aqrl rd, rs2, (rs1)

操作：

rd ← mem[rs1+7: rs1]

mem[rs1+7:rs1] ←rs2

执行权限：

M mode/S mode/U mode

异常：

原子指令非对齐访问异常、原子指令访问错误异常、原子指令页面错误异常

影响标志位：

无

说明：

aq 位和 rl 位决定该指令前序和后序访问存储器指令的执行顺序：

- aq=0,rl=0: 对应的汇编指令 amoswap.d rd, rs2, (rs1)。
- aq=0,rl=1：对应的汇编指令 amoswap.d.rl rd, rs2, (rs1)，该指令前序所有访问存储的指令的结果必须在该指令执行之前被观察到。
- aq=1,rl=0：对应的汇编指令 amoswap.d.aq rd, rs2, (rs1)，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。
- aq=1,rl=1：对应的汇编指令 amoswap.d.aqrl rd, rs2, (rs1)，指令前序所有访问存储的指令的结果必须在该指令执行之前被观察到，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。

指令格式：

31 27	26	25	24 20	19 15	14 12	11 7	6 0
00001	aq	rl	rs2	rs1	011	rd	0101111

18.3.16　AMOSWAP.W——低 32 位原子交换指令

语法：

amoswap.w.aqrl rd, rs2, (rs1)

操作：

rd ← sign_extend(mem[rs1+3: rs1])

mem[rs1+3:rs1] ←rs2[31:0]

执行权限：

M mode/S mode/U mode

异常：

原子指令非对齐访问异常、原子指令访问错误异常、原子指令页面错误异常

影响标志位：

无

说明：

aq 位和 rl 位决定该指令前序和后序访问存储器指令的执行顺序：

- aq=0,rl=0: 对应的汇编指令 amoswap.w rd, rs2, (rs1)。
- aq=0,rl=1：对应的汇编指令 amoswap.w.rl rd, rs2, (rs1)，该指令前序所有访问存储的指令的结果必须在该指令执行之前被观察到。

- aq=1,rl=0：对应的汇编指令 amoswap.w.aq rd, rs2, (rs1)，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。
- aq=1,rl=1：对应的汇编指令 amoswap.w.aqrl rd, rs2, (rs1)，指令前序所有访问存储的指令的结果必须在该指令执行之前被观察到，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。

指令格式：

31 27	26	25	24 20	19 15	14 12	11 7	6 0
00001	aq	rl	rs2	rs1	010	rd	0101111

18.3.17　AMOXOR.D——原子按位异或指令

语法：

amoxor.d.aqrl rd, rs2, (rs1)

操作：

rd ← mem[rs1+7: rs1]

mem[rs1+7:rs1] ← mem[rs1+7:rs1] ^ rs2

执行权限：

M mode/S mode/U mode

异常：

原子指令非对齐访问异常、原子指令访问错误异常、原子指令页面错误异常

影响标志位：

无

说明：

aq 位和 rl 位决定该指令前序和后序访问存储器指令的执行顺序：

- aq=0,rl=0: 对应的汇编指令 amoxor.d rd, rs2, (rs1)。
- aq=0,rl=1：对应的汇编指令 amoxor.d.rl rd, rs2, (rs1)，该指令前序所有访问存储的指令的结果必须在该指令执行之前被观察到。
- aq=1,rl=0：对应的汇编指令 amoxor.d.aq rd, rs2, (rs1)，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。
- aq=1,rl=1：对应的汇编指令 amoxor.d.aqrl rd, rs2, (rs1)，指令前序所有访问存储的指令的结果必须在该指令执行之前被观察到，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。

指令格式：

31 27	26	25	24 20	19 15	14 12	11 7	6 0
00100	aq	rl	rs2	rs1	011	rd	0101111

18.3.18　AMOXOR.W——低 32 位原子按位异或指令

语法：

amoxor.w.aqrl rd, rs2, (rs1)

操作：

rd ← sign_extend(mem[rs1+3: rs1])

mem[rs1+3:rs1] ← mem[rs1+3:rs1] ^ rs2[31:0]

执行权限：

M mode/S mode/U mode

异常：

原子指令非对齐访问异常、原子指令访问错误异常、原子指令页面错误异常

影响标志位：

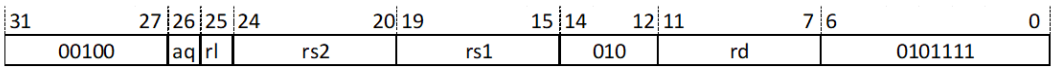
无

说明：

aq 位和 rl 位决定该指令前序和后序访问存储器指令的执行顺序：

- aq=0,rl=0: 对应的汇编指令 amoxor.w rd, rs2, (rs1)。
- aq=0,rl=1：对应的汇编指令 amoxor.w.rl rd, rs2, (rs1)，该指令前序所有访问存储的指令的结果必须在该指令执行之前被观察到。
- aq=1,rl=0：对应的汇编指令 amoxor.w.aq rd, rs2, (rs1)，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。
- aq=1,rl=1：对应的汇编指令 amoxor.w.aqrl rd, rs2, (rs1)，指令前序所有访问存储的指令的结果必须在该指令执行之前被观察到，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。

指令格式：



18.3.19　LR.D——双字加载保留指令

语法：

lr.d.aqrl rd, (rs1)

操作：

rd ← mem[rs1+7: rs1]

mem[rs1+7:rs1] is reserved

执行权限：

M mode/S mode/U mode

异常：

原子指令非对齐访问异常、原子指令访问错误异常、原子指令页面错误异常

影响标志位：

无

说明：

aq 位和 rl 位决定该指令前序和后序访问存储器指令的执行顺序：

- aq=0,rl=0: 对应的汇编指令 lr.d rd, (rs1)。
- aq=0,rl=1：对应的汇编指令 lr.d.rl rd, (rs1)，该指令前序所有访问存储的指令的结果必须在该指令执行之前被观察到。
- aq=1,rl=0：对应的汇编指令 lr.d.aq rd, (rs1)，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。
- aq=1,rl=1：对应的汇编指令 lr.d.aqrl rd, (rs1)，指令前序所有访问存储的指令的结果必须在该指令执行之前被观察到，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。

指令格式：

31 27	26	25	24 20	19 15	14 12	11 7	6 0
00010	aq	rl	00000	rs1	011	rd	0101111

18.3.20　LR.W——字加载保留指令

语法：

lr.w.aqrl rd, (rs1)

操作：

rd ←sign_extend(mem[rs1+3: rs1])

mem[rs1+3:rs1] is reserved

执行权限：

M mode/S mode/U mode

异常：

原子指令非对齐访问异常、原子指令访问错误异常、原子指令页面错误异常

影响标志位：

无

说明：

aq 位和 rl 位决定该指令前序和后序访问存储器指令的执行顺序：

- aq=0,rl=0: 对应的汇编指令 lr.w rd, (rs1)。
- aq=0,rl=1：对应的汇编指令 lr.w.rl rd, (rs1)，该指令前序所有访问存储的指令的结果必须在该指令执行之前被观察到。

- aq=1,rl=0：对应的汇编指令 lr.w.aq rd, (rs1)，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。
- aq=1,rl=1：对应的汇编指令 lr.w.aqrl rd, (rs1)，指令前序所有访问存储的指令的结果必须在该指令执行之前被观察到，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。

指令格式：

31 27	26	25	24 20	19 15	14 12	11 7	6 0
00010	aq	rl	00000	rs1	010	rd	0101111

18.3.21　SC.D—双字条件存储指令

语法：

sc.d.aqrl rd, rs2, (rs1)

操作：

If(mem[rs1+7:rs1] is reserved)

mem[rs1+7: rs1] ← rs2

rd ← 0

else

rd ← 1

执行权限：

M mode/S mode/U mode

异常：

原子指令非对齐访问异常、原子指令访问错误异常、原子指令页面错误异常

影响标志位：

无

说明：

aq 位和 rl 位决定该指令前序和后序访问存储器指令的执行顺序：

- aq=0,rl=0: 对应的汇编指令 sc.d rd, rs2, (rs1)。
- aq=0,rl=1：对应的汇编指令 sc.d.rl rd, rs2, (rs1)，该指令前序所有访问存储的指令的结果必须在该指令执行之前被观察到。
- aq=1,rl=0：对应的汇编指令 sc.d.aq rd, rs2, (rs1)，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。
- aq=1,rl=1：对应的汇编指令 sc.d.aqrl rd, rs2, (rs1)，指令前序所有访问存储的指令的结果必须在该指令执行之前被观察到，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。

指令格式：

31 27	26	25	24 20	19 15	14 12	11 7	6 0
00011	aq	rl	rs2	rs1	011	rd	0101111

18.3.22　SC.W—字条件存储指令

语法：

sc.w.aqrl rd, rs2, (rs1)

操作：

if(mem[rs1+3:rs1] is reserved)

mem[rs1+3:rs1] ← rs2[31:0]

rd ← 0

else

rd ← 1

执行权限：

M mode/S mode/U mode

异常：

原子指令非对齐访问异常、原子指令访问错误异常、原子指令页面错误异常

影响标志位：

无

说明：

aq 位和 rl 位决定该指令前序和后序访问存储器指令的执行顺序：

- aq=0,rl=0: 对应的汇编指令 sc.w rd, rs2, (rs1)。
- aq=0,rl=1：对应的汇编指令 sc.w.rl rd, rs2, (rs1)，该指令前序所有访问存储的指令的结果必须在该指令执行之前被观察到。
- aq=1,rl=0：对应的汇编指令 sc.w.aq rd, rs2, (rs1)，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。
- aq=1,rl=1：对应的汇编指令 sc.w.aqrl rd, rs2, (rs1)，指令前序所有访问存储的指令的结果必须在该指令执行之前被观察到，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行。

指令格式：

31 27	26	25	24 20	19 15	14 12	11 7	6 0
00011	aq	rl	rs2	rs1	010	rd	0101111

18.4　附录 A-4 F 指令术语

以下是对 C908X 实现的 RISC-V F 指令集的具体描述，本节指令位宽为 32 位，指令按英文字母顺序排列，

对于单精度浮点指令，如果源寄存器的高 32 位不全为 1，则该单精度数据当作 qNaN 处理。

当 mstatus.fs==2'b00 时，执行本节所有指令会产生非法指令异常，当 mstatus.fs!= 2'b00 时，执行本节任意指令后 mstatus.fs 置位为 2'b11。

18.4.1　FADD.S——单精度浮点加法指令

语法：

fadd.s fd, fs1, fs2, rm

操作：

frd ← fs1 + fs2

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

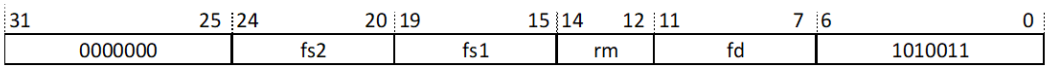
浮点状态位 NV/OF/NX

说明：

rm 决定舍入模式:

- 3'b000: 就近向偶数舍入，对应的汇编指令 fadd.s fd, fs1,fs2,rne。
- 3'b001: 向零舍入，对应的汇编指令 fadd.s fd, fs1,fs2,rtz。
- 3'b010: 向负无穷舍入，对应的汇编指令 fadd.s fd, fs1,fs2,rdn。
- 3'b011: 向正无穷舍入，对应的汇编指令 fadd.s fd, fs1,fs2,rup。
- 3'b100: 就近向大值舍入，对应的汇编指令 fadd.s fd, fs1,fs2,rmm。
- 3'b101: 暂未使用，不会出现该编码。
- 3'b110: 暂未使用，不会出现该编码。
- 3'b111: 动态舍入，根据浮点控制寄存器 fcsr 中的 rm 位来决定舍入模式，对应的汇编指令 fadd.s fd, fs1,fs2。

指令格式：



18.4.2　FCLASS.S——单精度浮点分类指令

语法：

fclass.s rd, fs1

操作：

if (fs1 = -inf)

　　rd ← 64'h1

if (fs1 = -norm)

rd ← 64'h2

if (fs1 = -subnorm)

rd ← 64'h4

if (fs1 = -zero)

rd ← 64'h8

if (fs1 = +zero)

rd ← 64'h10

if (fs1 = +subnorm)

rd ← 64'h20

if (fs1 = +norm)

rd ← 64'h40

if (fs1 = +Inf)

rd ← 64'h80

if (fs1 = sNaN)

rd ← 64'h100

if (fs1 = qNaN)

rd ← 64'h200

执行权限：

M mode/S mode/U mode

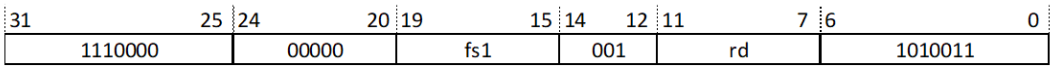
异常：

非法指令异常

影响标志位：

无

指令格式：



18.4.3　FCVT.L.S——单精度浮点转换成有符号长整型指令

语法：

fcvt.l.s rd, fs1, rm

操作：

rd ← single_convert_to_signed_long(fs1)

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

浮点状态位 NV/NX

说明：

rm 决定舍入模式:

- 3'b000: 就近向偶数舍入，对应的汇编指令 fcvt.l.s rd,fs1,rne。
- 3'b001: 向零舍入，对应的汇编指令 fcvt.l.s rd,fs1,rtz。
- 3'b010: 向负无穷舍入，对应的汇编指令 fcvt.l.s rd,fs1,rdn。
- 3'b011: 向正无穷舍入，对应的汇编指令 fcvt.l.s rd,fs1,rup。
- 3'b100: 就近向大值舍入，对应的汇编指令 fcvt.l.s rd,fs1,rmm。
- 3'b101: 暂未使用，不会出现该编码。
- 3'b110: 暂未使用，不会出现该编码。
- 3'b111: 动态舍入，根据浮点控制寄存器 fcsr 中的 rm 位来决定舍入模式，对应的汇编指令 fcvt.l.s rd, fs1。

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
1100000	00010	fs1	rm	rd	1010011

18.4.4　FCVT.LU.S—单精度浮点转换成无符号长整型指令

语法：

fcvt.lu.s rd, fs1, rm

操作：

rd ← single_convert_to_unsigned_long(fs1)

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

浮点状态位 NV/NX

说明：

rm 决定舍入模式:

- 3'b000: 就近向偶数舍入，对应的汇编指令 fcvt.lu.s rd,fs1,rne。
- 3'b001: 向零舍入，对应的汇编指令 fcvt.lu.s rd,fs1,rtz。
- 3'b010: 向负无穷舍入，对应的汇编指令 fcvt.lu.s rd,fs1,rdn。
- 3'b011: 向正无穷舍入，对应的汇编指令 fcvt.lu.s rd,fs1,rup。
- 3'b100: 就近向大值舍入，对应的汇编指令 fcvt.lu.s rd,fs1,rmm。
- 3'b101: 暂未使用，不会出现该编码。
- 3'b110: 暂未使用，不会出现该编码。
- 3'b111: 动态舍入，根据浮点控制寄存器 fcsr 中的 rm 位来决定舍入模式，对应的汇编指令 fcvt.lu.s rd, fs1。

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
1100000	00011	fs1	rm	rd	1010011

18.4.5　FCVT.S.L—有符号长整型转换成单精度浮点数指令

语法：

fcvt.s.l fd, rs1, rm

操作：

fd ← signed_long_convert_to_single(fs1)

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

浮点状态位 NX

说明：

rm 决定舍入模式:

- 3'b000: 就近向偶数舍入，对应的汇编指令 fcvt.s.l fd,rs1,rne。
- 3'b001: 向零舍入，对应的汇编指令 fcvt.s.l fd,rs1,rtz。
- 3'b010: 向负无穷舍入，对应的汇编指令 fcvt.s.l fd,fs1,rdn。
- 3'b011: 向正无穷舍入，对应的汇编指令 fcvt.s.l fd,fs1,rup。
- 3'b100: 就近向大值舍入，对应的汇编指令 fcvt.s.l fd,fs1,rmm。
- 3'b101: 暂未使用，不会出现该编码。
- 3'b110: 暂未使用，不会出现该编码。

- 3'b111: 动态舍入，根据浮点控制寄存器 fcsr 中的 rm 位来决定舍入模式，对应的汇编指令 fcvt.s.l fd, fs1。

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
1101000	00010	rs1	rm	fd	1010011

18.4.6　FCVT.S.LU—无符号长整型转换成单精度浮点数指令

语法：

fcvt.s.l fd, fs1, rm

操作：

fd ← unsigned_long_convert_to_single_fp(fs1)

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

浮点状态位 NX

说明：

rm 决定舍入模式:

- 3'b000: 就近向偶数舍入，对应的汇编指令 fcvt.s.lu fd,fs1,rne。
- 3'b001: 向零舍入，对应的汇编指令 fcvt.s.lu fd, fs1,rtz。
- 3'b010: 向负无穷舍入，对应的汇编指令 fcvt.s.lu fd, fs1,rdn。
- 3'b011: 向正无穷舍入，对应的汇编指令 fcvt.s.lu fd, fs1,rup。
- 3'b100: 就近向大值舍入，对应的汇编指令 fcvt.s.lu fd, fs1,rmm。
- 3'b101: 暂未使用，不会出现该编码。
- 3'b110: 暂未使用，不会出现该编码。
- 3'b111: 动态舍入，根据浮点控制寄存器 fcsr 中的 rm 位来决定舍入模式，对应的汇编指令 fcvt.s.lu fd, fs1。

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
1101000	00011	rs1	rm	fd	1010011

18.4.7　FCVT.S.W—有符号整型转换成单精度浮点数指令

语法：

fcvt.s.w fd, rs1, rm

操作：

fd ← signed_int_convert_to_single(fs1)

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

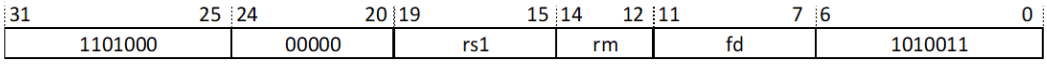
浮点状态位 NX

说明：

rm 决定舍入模式:

- 3'b000: 就近向偶数舍入，对应的汇编指令 fcvt.s.w fd,rs1,rne。
- 3'b001: 向零舍入，对应的汇编指令 fcvt.s.w fd,rs1,rtz。
- 3'b010: 向负无穷舍入，对应的汇编指令 fcvt.s.w fd,rs1,rdn。
- 3'b011: 向正无穷舍入，对应的汇编指令 fcvt.s.w fd,rs1,rup。
- 3'b100: 就近向大值舍入，对应的汇编指令 fcvt.s.w fd,rs1,rmm。
- 3'b101: 暂未使用，不会出现该编码。
- 3'b110: 暂未使用，不会出现该编码。
- 3'b111: 动态舍入，根据浮点控制寄存器 fcsr 中的 rm 位来决定舍入模式，对应的汇编指令 fcvt.s.w fd, rs1。

指令格式：



18.4.8　FCVT.S.WU—无符号整型转换成单精度浮点数指令

语法：

fcvt.s.wu fd, rs1, rm

操作：

fd ← unsigned_int_convert_to_single_fp(fs1)

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

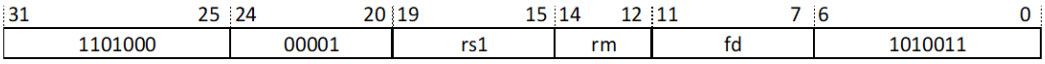
浮点状态位 NX

说明：

rm 决定舍入模式:

- 3'b000: 就近向偶数舍入，对应的汇编指令 fcvt.s.wu fd,rs1,rne。
- 3'b001: 向零舍入，对应的汇编指令 fcvt.s.wu fd,rs1,rtz。
- 3'b010: 向负无穷舍入，对应的汇编指令 fcvt.s.wu fd,rs1,rdn。
- 3'b011: 向正无穷舍入，对应的汇编指令 fcvt.s.wu fd,rs1,rup。
- 3'b100: 就近向大值舍入，对应的汇编指令 fcvt.s.wu fd,rs1,rmm。
- 3'b101: 暂未使用，不会出现该编码。
- 3'b110: 暂未使用，不会出现该编码。
- 3'b111: 动态舍入，根据浮点控制寄存器 fcsr 中的 rm 位来决定舍入模式，对应的汇编指令 fcvt.s.wu fd, rs1。

指令格式：



18.4.9　FCVT.W.S——单精度浮点转换成有符号整型指令

语法：

fcvt.w.s rd, fs1, rm

操作：

tmp ← single_convert_to_signed_int(fs1)

rd←sign_extend(tmp)

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

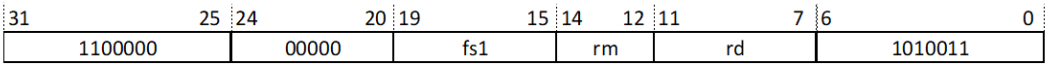
浮点状态位 NV/NX

说明：

rm 决定舍入模式:

- 3'b000: 就近向偶数舍入，对应的汇编指令 fcvt.w.s rd,fs1,rne。
- 3'b001: 向零舍入，对应的汇编指令 fcvt.w.s rd,fs1,rtz。
- 3'b010: 向负无穷舍入，对应的汇编指令 fcvt.w.s rd,fs1,rdn。
- 3'b011: 向正无穷舍入，对应的汇编指令 fcvt.w.s rd,fs1,rup。
- 3'b100: 就近向大值舍入，对应的汇编指令 fcvt.w.s rd,fs1,rmm。
- 3'b101: 暂未使用，不会出现该编码。
- 3'b110: 暂未使用，不会出现该编码。
- 3'b111: 动态舍入，根据浮点控制寄存器 fcsr 中的 rm 位来决定舍入模式，对应的汇编指令 fcvt.w.s rd, fs1。

指令格式：



18.4.10　FCVT.WU.S——单精度浮点转换成无符号整型指令

语法：

fcvt.wu.s rd, fs1, rm

操作：

tmp ← single_convert_to_unsigned_int(fs1)

rd←sign_extend(tmp)

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

浮点状态位 NV/NX

说明：

rm 决定舍入模式:

- 3'b000: 就近向偶数舍入，对应的汇编指令 fcvt.wu.s rd,fs1,rne。
- 3'b001: 向零舍入，对应的汇编指令 fcvt.wu.s rd,fs1,rtz。
- 3'b010: 向负无穷舍入，对应的汇编指令 fcvt.wu.s rd,fs1,rdn。
- 3'b011: 向正无穷舍入，对应的汇编指令 fcvt.wu.s rd,fs1,rup。
- 3'b100: 就近向大值舍入，对应的汇编指令 fcvt.wu.s rd,fs1,rmm。
- 3'b101: 暂未使用，不会出现该编码。

- 3'b110: 暂未使用，不会出现该编码。
- 3'b111: 动态舍入，根据浮点控制寄存器 fcsr 中的 rm 位来决定舍入模式，对应的汇编指令 fcvt.wu.s rd, fs1。

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
1100000	00001	fs1	rm	rd	1010011

18.4.11　FDIV.S——单精度浮点除法指令

语法：

fdiv.s fd, fs1, fs2, rm

操作：

fd ← fs1 / fs2

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

浮点状态位 NV/DZ/OF/UF/NX

说明：

rm 决定舍入模式:

- 3'b000: 就近向偶数舍入，对应的汇编指令 fdiv.s fs1,fs2,rne。
- 3'b001: 向零舍入，对应的汇编指令 fdiv.s fd fs1,fs2,rtz。
- 3'b010: 向负无穷舍入，对应的汇编指令 fdiv.s fd, fs1,fs2,rdn。
- 3'b011: 向正无穷舍入，对应的汇编指令 fdiv.s fd, fs1,fs2,rup。
- 3'b100: 就近向大值舍入，对应的汇编指令 fdiv.s fd, fs1,fs2,rmm。
- 3'b101: 暂未使用，不会出现该编码。
- 3'b110: 暂未使用，不会出现该编码。
- 3'b111: 动态舍入，根据浮点控制寄存器 fcsr 中的 rm 位来决定舍入模式，对应的汇编指令 fdiv.s fd, fs1,fs2。

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
0001100	fs1	fs2	rm	fd	1010011

18.4.12　FEQ.S——单精度浮点比较相等指令

语法：

feq.s rd, fs1, fs2

操作：

if(fs1 == fs2)

　　rd ← 1

else

　　rd ← 0

执行权限：

M mode/S mode/U mode

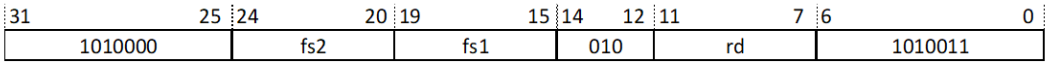
异常：

非法指令异常

影响标志位：

浮点状态位 NV

指令格式：



18.4.13　FLE.S——单精度浮点比较小于等于指令

语法：

fle.s rd, fs1, fs2

操作：

if(fs1 <= fs2)

　　rd ← 1

else

　　rd ← 0

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

浮点状态位 NV

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
1010000	fs2	fs1	000	rd	1010011

18.4.14　FLT.S——单精度浮点比较小于指令

语法：

flt.s rd, fs1, fs2

操作：

if(fs1 < fs2)

　　rd ← 1

else

　　rd ← 0

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

浮点状态位 NV

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
1010000	fs2	fs1	001	rd	1010011

18.4.15　FLW——单精度浮点加载指令

语法：

flw fd, imm12(rs1)

操作：

address←rs1+sign_extend(imm12)

fd[31:0] ← mem[(address+3):address]

fd[63:32] ← 32'hffffffff

执行权限：

M mode/S mode/U mode

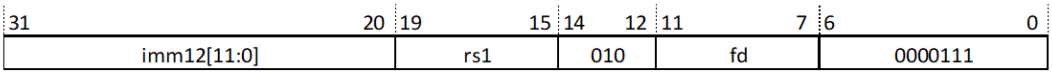
异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常、非法指令异常

影响标志位：

无

指令格式：



18.4.16　FMADD.S——单精度浮点乘累加指令

语法：

fmadd.s fd, fs1, fs2, fs3, rm

操作：

rd ← fs1*fs2 + fs3

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

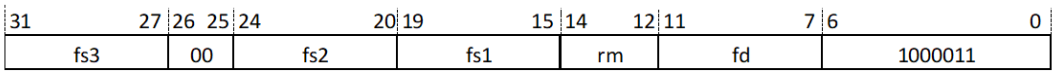
浮点状态位 NV/OF/UF/IX

说明：

rm 决定舍入模式:

- 3'b000: 就近向偶数舍入，对应的汇编指令 fmadd.s fd,fs1, fs2, fs3, rne。
- 3'b001: 向零舍入，对应的汇编指令 fmadd.s fd,fs1, fs2, fs3, rtz。
- 3'b010: 向负无穷舍入，对应的汇编指令 fmadd.s fd,fs1, fs2, fs3, rdn。
- 3'b011: 向正无穷舍入，对应的汇编指令 fmadd.s fd,fs1, fs2, fs3, rup。
- 3'b100: 就近向大值舍入，对应的汇编指令 fmadd.s fd,fs1, fs2, fs3, rmm。
- 3'b101: 暂未使用，不会出现该编码。
- 3'b110: 暂未使用，不会出现该编码。
- 3'b111: 动态舍入，根据浮点控制寄存器 fcsr 中的 rm 位来决定舍入模式，对应的汇编指令 fmadd.s fd,fs1, fs2, fs3。

指令格式：



18.4.17　FMAX.S——单精度浮点取最大值指令

语法：

fmax.s fd, fs1, fs2

操作：

if(fs1 >= fs2)

　　fd ← fs1

else

　　fd ← fs2

执行权限：

M mode/S mode/U mode

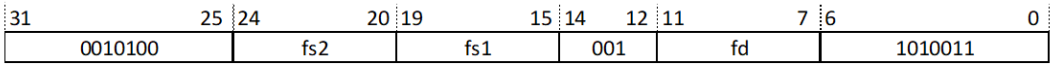
异常：

非法指令异常

影响标志位：

浮点状态位 NV

指令格式：



31 25	24 20	19 15	14 12	11 7	6 0
0010100	fs2	fs1	001	fd	1010011

18.4.18　FMIN.S——单精度浮点取最小值指令

语法：

fmin.s fd, fs1, fs2

操作：

if(fs1 >= fs2)

　　fd ← fs2

else

　　fd ← fs1

执行权限：

M mode/S mode/U mode

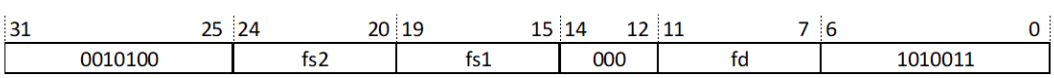
异常：

非法指令异常

影响标志位：

浮点状态位 NV

指令格式：



31 25	24 20	19 15	14 12	11 7	6 0
0010100	fs2	fs1	000	fd	1010011

18.4.19　FMSUB.S——单精度浮点乘累减指令

语法：

fmsub.s fd, fs1, fs2, fs3, rm

操作：

fd ← fs1*fs2 - fs3

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

浮点状态位 NV/OF/UF/IX

说明：

rm 决定舍入模式:

- 3'b000: 就近向偶数舍入，对应的汇编指令 fmsub.s fd,fs1, fs2, fs3, rne。
- 3'b001: 向零舍入，对应的汇编指令 fmsub.s fd,fs1, fs2, fs3, rtz。
- 3'b010: 向负无穷舍入，对应的汇编指令 fmsub.s fd,fs1, fs2, fs3, rdn。
- 3'b011: 向正无穷舍入，对应的汇编指令 fmsub.s fd,fs1, fs2, fs3, rup。
- 3'b100: 就近向大值舍入，对应的汇编指令 fmsub.s fd, fs1, fs2, fs3,rmm。
- 3'b101: 暂未使用，不会出现该编码。
- 3'b110: 暂未使用，不会出现该编码。
- 3'b111: 动态舍入，根据浮点控制寄存器 fcsr 中的 rm 位来决定舍入模式，对应的汇编指令 fmsub.s fd,fs1, fs2, fs3。

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
fs3	00	fs2	fs1	rm	fd	1000111

18.4.20　FMUL.S——单精度浮点乘法指令

语法：

fmul.s fd, fs1, fs2, rm

操作：

fd ← fs1 * fs2

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

浮点状态位 NV/OF/UF/NX

说明：

rm 决定舍入模式:

- 3'b000: 就近向偶数舍入，对应的汇编指令 fmul.s fd, fs1, fs2, rne。
- 3'b001: 向零舍入，对应的汇编指令 fmul.s fd, fs1, fs2, rtz。
- 3'b010: 向负无穷舍入，对应的汇编指令 fmul.s fd, fs1, fs2, rdn。
- 3'b011: 向正无穷舍入，对应的汇编指令 fmul.s fd, fs1, fs2, rup。
- 3'b100: 就近向大值舍入，对应的汇编指令 fmul.s fd, fs1,fs2, rmm。
- 3'b101: 暂未使用，不会出现该编码。
- 3'b110: 暂未使用，不会出现该编码。
- 3'b111: 动态舍入，根据浮点控制寄存器 fcsr 中的 rm 位来决定舍入模式，对应的汇编指令 fmul.s fs1,fs2。

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
0001000	fs2	fs1	rm	fd	1010011

18.4.21　FMV.W.X—单精度浮点写传送指令

语法：

fmv.w.x fd, rs1

操作：

fd[31:0] ← rs[31:0]

fd[63:32] ← 32'hffffffff

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

无

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
1111000	00000	rs1	000	fd	1010011

18.4.22　FMV.X.W——单精度浮点寄存器读传送指令

语法：

fmv.x.w rd, fs1

操作：

tmp[31:0] ← fs1[31:0]

rd ← sign_extend(tmp[31:0])

执行权限：

M mode/S mode/U mode

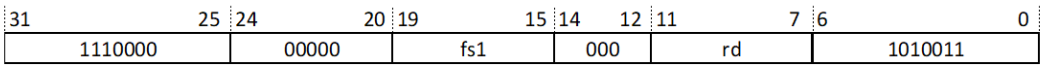
异常：

非法指令异常

影响标志位：

无

指令格式：



18.4.23　FNMADD.S——单精度浮点乘累加取负指令

语法：

fnmadd.s fd, fs1, fs2, fs3, rm

操作：

fd ←-(fs1*fs2 + fs3)

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

浮点状态位 NV/OF/UF/IX

说明：

rm 决定舍入模式:

- 3'b000: 就近向偶数舍入，对应的汇编指令 fnmadd.s fd,fs1, fs2, fs3, rne。
- 3'b001: 向零舍入，对应的汇编指令 fnmadd.s fd,fs1, fs2, fs3, rtz。
- 3'b010: 向负无穷舍入，对应的汇编指令 fnmadd.s fd,fs1, fs2, fs3, rdn。
- 3'b011: 向正无穷舍入，对应的汇编指令 fnmadd.s fd,fs1, fs2, fs3, rup。

- 3'b100: 就近向大值舍入，对应的汇编指令 fnmadd.s fd,fs1, fs2, fs3, rmm。
- 3'b101: 暂未使用，不会出现该编码。
- 3'b110: 暂未使用，不会出现该编码。
- 3'b111: 动态舍入，根据浮点控制寄存器 fcsr 中的 rm 位来决定舍入模式，对应的汇编指令 fnmadd.s fd,fs1, fs2, fs3。

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
fs3	00	fs2	fs1	rm	fd	1001111

18.4.24　FNMSUB.S——单精度浮点乘累减取负指令

语法：

fnmsub.s fd, fs1, fs2, fs3, rm

操作：

fd ← -(fs1*fs2 - fs3)

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

浮点状态位 NV/OF/UF/IX

说明：

rm 决定舍入模式:

- 3'b000: 就近向偶数舍入，对应的汇编指令 fnmsub.s fd,fs1, fs2, fs3, rne。
- 3'b001: 向零舍入，对应的汇编指令 fnmsub.s fd,fs1, fs2, fs3, rtz。
- 3'b010: 向负无穷舍入，对应的汇编指令 fnmsub.s fd,fs1, fs2, fs3, rdn。
- 3'b011: 向正无穷舍入，对应的汇编指令 fnmsub.s fd,fs1, fs2, fs3, rup。
- 3'b100: 就近向大值舍入，对应的汇编指令 fnmsub.s fd,fs1, fs2, fs3, rmm。
- 3'b101: 暂未使用，不会出现该编码。
- 3'b110: 暂未使用，不会出现该编码。
- 3'b111: 动态舍入，根据浮点控制寄存器 fcsr 中的 rm 位来决定舍入模式，对应的汇编指令 fnmsub.s fd,fs1, fs2, fs3。

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
fs3	00	fs2	fs1	rm	fd	1001011

18.4.25　FSGNJ.S——单精度浮点符号注入指令

语法：

fsgnj.s fd, fs1, fs2

操作：

fd[30:0] ← fs1[30:0]

fd[31] ← fs2[31]

fd[63:32] ← 32'hffffffff

执行权限：

M mode/S mode/U mode

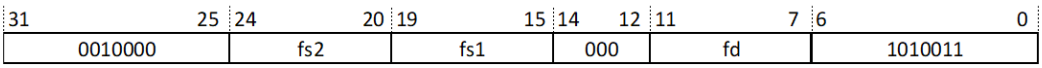
异常：

非法指令异常

影响标志位：

无

指令格式：



31 25	24 20	19 15	14 12	11 7	6 0
0010000	fs2	fs1	000	fd	1010011

18.4.26　FSGNJN.S——单精度浮点符号取反注入指令

语法：

fsgnjn.s fd, fs1, fs2

操作：

fd[30:0] ← fs1[30:0]

fd[31] ← ! fs2[31]

fd[63:32] ← 32'hffffffff

执行权限：

M mode/S mode/U mode

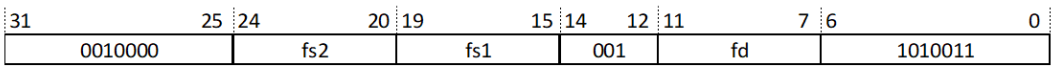
异常：

非法指令异常

影响标志位：

无

指令格式：



31 25	24 20	19 15	14 12	11 7	6 0
0010000	fs2	fs1	001	fd	1010011

18.4.27　FSGNJX.S——单精度浮点符号异或注入指令

语法：

fsgnjx.s fd, fs1, fs2

操作：

fd[30:0] ← fs1[30:0]

fd[31] ← fs1[31] ^ fs2[31]

fd[63:32] ← 32'hffffffff

执行权限：

M mode/S mode/U mode

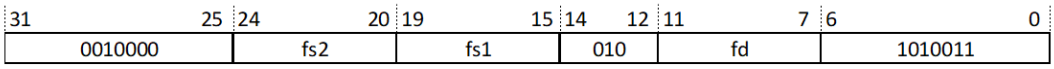
异常：

非法指令异常

影响标志位：

无

指令格式：



18.4.28　FSQRT.S——单精度浮点开方指令

语法：

fsqrt.s fd, fs1, rm

操作：

fd ← sqrt(fs1)

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

浮点状态位 NV/NX

说明：

rm 决定舍入模式:

- 3'b000: 就近向偶数舍入，对应的汇编指令 fsqrt.s fd, fs1,rne
- 3'b001: 向零舍入，对应的汇编指令 fsqrt.s fd, fs1,rtz

- 3'b010: 向负无穷舍入，对应的汇编指令 fsqrt.s fd, fs1,rdn
- 3'b011: 向正无穷舍入，对应的汇编指令 fsqrt.s fd, fs1,rup
- 3'b100: 就近向大值舍入，对应的汇编指令 fsqrt.s fd, fs1,rmm
- 3'b101: 暂未使用，不会出现该编码。
- 3'b110: 暂未使用，不会出现该编码。
- 3'b111：动态舍入，根据浮点控制寄存器 fcsr 中的 rm 位来决定舍入模式，对应的汇编指令 fsqrt.s fd, fs1。

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
0101100	00000	fs1	rm	fd	1010011

18.4.29　FSUB.S——单精度浮点减法指令

语法：

fsub.s fd, fs1, fs2, rm

操作：

fd ← fs1 - fs2

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

浮点状态位 NV/OF/NX

说明：

rm 决定舍入模式:

- 3'b000: 就近向偶数舍入，对应的汇编指令 fsub.fd, fs1,fs2,rne
- 3'b001: 向零舍入，对应的汇编指令 fsub.s fd, fs1,fs2,rtz
- 3'b010: 向负无穷舍入，对应的汇编指令 fsub.s fd, fs1,fs2,rdn
- 3'b011: 向正无穷舍入，对应的汇编指令 fsub.s fd, fs1,fs2,rup
- 3'b100: 就近向大值舍入，对应的汇编指令 fsub.s fd, fs1,fs2,rmm
- 3'b101: 暂未使用，不会出现该编码。
- 3'b110: 暂未使用，不会出现该编码。
- 3'b111：动态舍入，根据浮点控制寄存器 fcsr 中的 rm 位来决定舍入模式，对应的汇编指令 fsub.s fd, fs1,fs2。

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
0000100	fs2	fs1	rm	fd	1010011

18.4.30　FSW——单精度浮点存储指令

语法：

fsw fs2, imm12(rs1)

操作：

address←rs1+sign_extend(imm12)

mem[(address+31):address] ← fs2[31:0]

执行权限：

M mode/S mode/U mode

异常：

存储指令非对齐访问异常、存储指令访问错误异常、存储指令页面错误异常、非法指令异常

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
imm12[11:5]	fs2	rs1	010	imm12[4:0]	0100111

18.5　附录 A-6 C 指令术语

以下是对 C908X 实现的 RISC-V C 指令的具体描述，本节指令位宽为 16 位，指令按英文字母顺序排列。

18.5.1　C.ADD——有符号加法指令

语法：

c.add rd, rs2

操作：

rd ← rs1 + rs2

执行权限：

M mode/S mode/U mode

异常：

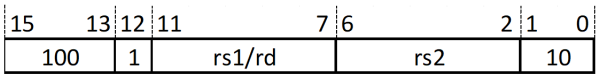
无

说明：

rs1 = rd != 0

rs2 ! = 0

指令格式：



18.5.2　C.ADDI—有符号立即数加法指令

语法：

c.addi rd, nzimm6

操作：

rd ← rs1 + sign_extend(nzimm6)

执行权限：

M mode/S mode/U mode

异常：

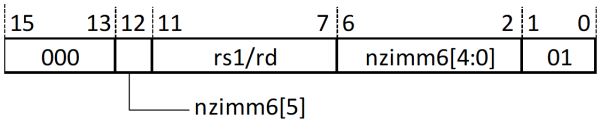
无

说明：

rs1 = rd != 0

nzimm6!=0

指令格式：



18.5.3　C.ADDIW—低 32 位有符号立即数加法指令

语法：

c.addiw rd, imm6

操作：

tmp[31:0] ← rs1[31:0] + sign_extend(imm6)

rd ←sign_extend(tmp[31:0])

执行权限：

M mode/S mode/U mode

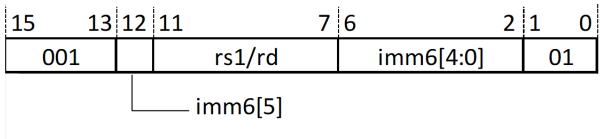
异常：

无

说明：

rs1 = rd != 0

指令格式：



18.5.4　C.ADDI4SPN——4 倍立即数和堆栈指针相加指令

语法：

c.addi4spn rd, sp, nzuimm8 << 2

操作：

rd ← sp + zero_extend(nzuimm8 << 2)

执行权限：

M mode/S mode/U mode

异常：

无

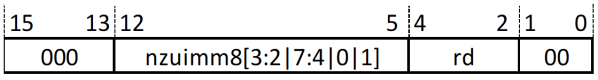
说明：

nzuimm8 != 0

rd 编码代表寄存器如下:

- 000 x8
- 001 x9
- 010 x10
- 011 x11
- 100 x12
- 101 x13
- 110 x14
- 111 x15

指令格式：



18.5.5　C.ADDI16SP——加 16 倍立即数到堆栈指针指令

语法：

c.addi16sp sp, nzuimm6 << 4

操作：

sp ← sp + sign_extend(nzuimm6 << 4)

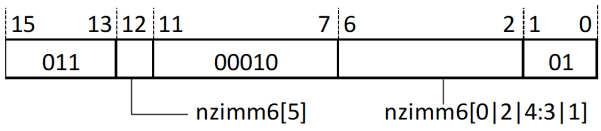
执行权限：

M mode/S mode/U mode

异常：

无

指令格式：



18.5.6　C.ADDW—低 32 位有符号加法指令

语法：

c.addw rd, rs2

操作：

tmp[31:0] ← rs1[31:0] + rs2[31:0]

rd ←sign_extend(tmp[31:0])

执行权限：

M mode/S mode/U mode

异常：

无

说明：

rs1 = rd

rd/rs1，rs2 编码代表寄存器如下：

- 000: x8
- 001: x9
- 010: x10
- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

指令格式：

15 13	12	11 10	9 7	6 5	4 2	1 0
100	1	11	rs1/rd	01	rs2	01

18.5.7　C.AND——按位与指令

语法：

c.and rd, rs2

操作：

rd ← rs1 & rs2

执行权限：

M mode/S mode/U mode

异常：

无

说明：

rs1 = rd

rd/rs1，rs2 编码代表寄存器如下：

- 000: x8
- 001: x9
- 010: x10
- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

指令格式：

15 13	12	11 10	9 7	6 5	4 2	1 0
100	0	11	rs1/rd	11	rs2	01

18.5.8　C.ANDI——立即数按位与指令

语法：

c.andi rd, imm6

操作：

rd ← rs1 & sign_extend(imm6)

执行权限：

M mode/S mode/U mode

异常：

无

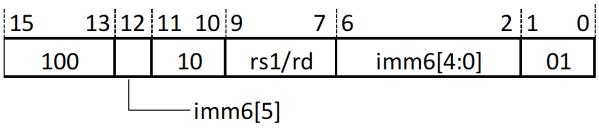
说明：

rs1 = rd

rd/rs1 编码代表寄存器如下：

- 000: x8
- 001: x9
- 010: x10
- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

指令格式：



18.5.9　C.BEQZ—等于零分支指令

语法：

c.beqz rs1, label

操作：

```
if (rs1 == 0)
        next pc = current pc + imm8 << 1;
else
        next pc = current pc + 2;
```

执行权限：

M mode/S mode/U mode

异常：

无

说明：

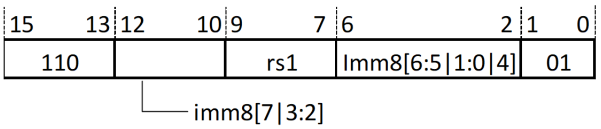
rs1 编码代表寄存器如下：

- 000: x8
- 001: x9
- 010: x10
- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

汇编器根据 label 算出 imm8

指令跳转范围为 ±256B 地址空间

指令格式：



18.5.10　C.BNEZ—不等于零分支指令

语法：

c.bnez rs1, label

操作：

```
if (rs1 != 0)
        next pc = current pc + imm8 << 1;
else
        next pc = current pc + 2;
```

执行权限：

M mode/S mode/U mode

异常：

无

说明：

rs1 编码代表寄存器如下：

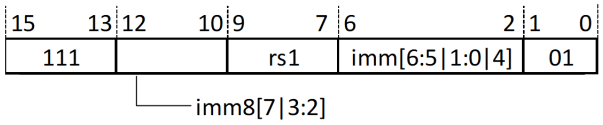
- 000: x8

- 001: x9
- 010: x10
- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

汇编器根据 label 算出 imm12

指令跳转范围为 ±256B 地址空间

指令格式：



18.5.11　C.EBREAK——断点指令

语法：

c.ebreak

操作：

产生断点异常或者进入调试模式

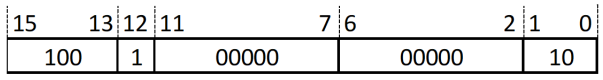
执行权限：

M mode/S mode/U mode

异常：

断点异常

指令格式：



18.5.12　C.FLD——浮点双字加载指令

语法：

c.fld fd, uimm5 << 3(rs1)

操作：

address ← rs1+ zero_extend(uimm5 << 3)

fd ←mem[address+7:address]

执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常

说明：

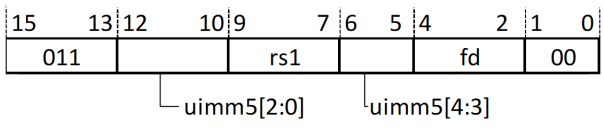
rs1 编码代表寄存器如下：

- 000: x8
- 001: x9
- 010: x10
- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

fd 编码代表寄存器如下：

- 000: f8
- 001: f9
- 010: f10
- 011: f11
- 100: f12
- 101: f13
- 110: f14
- 111: f15

指令格式：



18.5.13　C.FLDSP—浮点双字堆栈加载指令

语法：

c.fldsp fd, uimm6 << 3(sp)

操作：

address ← sp+ zero_extend(uimm6 << 3)

fd ←mem[address+7:address]

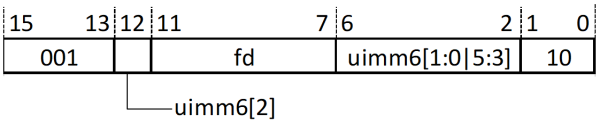
执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常

指令格式：



18.5.14　C.FSD——浮点双字存储指令

语法：

c.fsd fs2，uimm5 << 3(rs1)

操作：

address ← rs1+ zero_extend(uimm5 << 3)

mem[address+7:address] ←fs2

执行权限：

M mode/S mode/U mode

异常：

存储指令非对齐访问异常、存储指令访问错误异常、存储指令页面错误异常

说明：

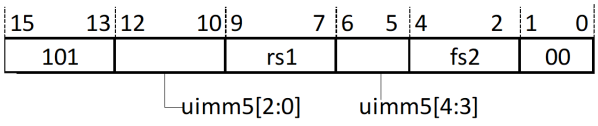
fs1 编码代表寄存器如下:

- 000: x8
- 001: x9
- 010: x10
- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

rs2 编码代表寄存器如下:

- 000: f8
- 001: f9
- 010: f10
- 011: f11
- 100: f12
- 101: f13
- 110: f14
- 111: f15

指令格式：



18.5.15　C.FSDSP—浮点双字堆栈存储指令

语法：

c.fsdsp fs2, uimm6 << 3(sp)

操作：

address ← sp+ zero_extend(uimm6 << 3)

mem[address+7:address] ←fs2

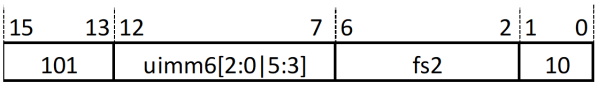
执行权限：

M mode/S mode/U mode

异常：

存储指令非对齐访问异常、存储指令访问错误异常、存储指令页面错误异常

指令格式：



18.5.16　C.J—无条件跳转指令

语法：

c.j label

操作：

next pc ← current pc + sign_extend(imm << 1);

执行权限：

M mode/S mode/U mode

异常：

无

说明：

汇编器根据 label 算出 imm11

指令跳转范围为 ±2KB 地址空间

指令格式：

15　13	12　2	1　0
101	imm11[10|3|8:7|9|5|6|2:0|4]	01

18.5.17　C.JALR——寄存器跳转子程序指令

语法：

c.jalr rs1

操作：

next pc ← rs1;

x1←current pc + 2;

执行权限：

M mode/S mode/U mode

异常：

无

说明：

rs1 != 0。

MMU 打开时，跳转范围是全部 512GB 地址空间。

MMU 关闭时，跳转范围是全部 1TB 地址空间。

指令格式：

15　13	12	11　7	6　2	1　0
100	1	rs1	00000	10

18.5.18　C.JR——寄存器跳转指令

语法：

c.jr rs1

操作：

next pc = rs1;

执行权限：

M mode/S mode/U mode

异常：

无

说明：

rs1 != 0。

MMU 打开时，跳转范围是全部 512GB 地址空间。

MMU 关闭时，跳转范围是全部 1TB 地址空间。

指令格式：

15　13	12	11　7	6　2	1　0
100	0	rs1	00000	10

18.5.19　C.LD—双字加载指令

语法：

c.ld rd, uimm5 << 3(rs1)

操作：

address ← rs1+ zero_extend(uimm5 << 3)

rd ←mem[address+7:address]

执行权限：

M mode/S mode/U mode

异常：

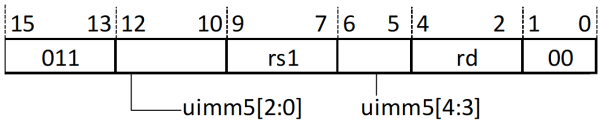
加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常

说明：

rs1/rd 编码代表寄存器如下:

- 000: x8
- 001: x9
- 010: x10
- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

指令格式：



18.5.20　C.LDSP—双字堆栈加载指令

语法：

c.ldsp rd, uimm6 << 3(sp)

操作：

address ← sp+ zero_extend(uimm6 << 3)

rd ←mem[address+7:address]

执行权限：

M mode/S mode/U mode

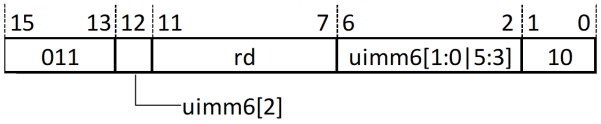
异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常

说明：

rd != 0

指令格式：



18.5.21　C.LI—立即数传送指令

语法：

c.li rd, imm6

操作：

rd ←sign_extend(imm6)

执行权限：

M mode/S mode/U mode

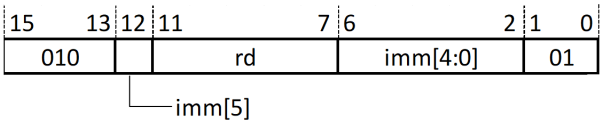
异常：

无

说明：

rd != 0。

指令格式：



18.5.22　C.LUI—高位立即数传送指令

语法：

c.lui rd, nzimm6

操作：

rd ←sign_extend(nzimm6 << 12)

执行权限：

M mode/S mode/U mode

异常：

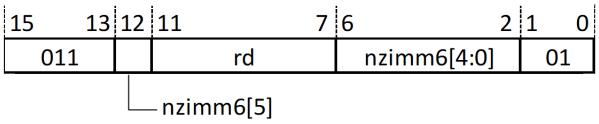
无

说明：

rd != 0。

Nzimm6 != 0。

指令格式：



18.5.23　C.LW—字加载指令

语法：

c.lw rd, uimm5 << 2(rs1)

操作：

address ← rs1+ zero_extend(uimm5 << 2)

tmp[31:0] ←mem[address+3:address]

rd ←sign_extend(tmp[31:0])

执行权限：

M mode/S mode/U mode

异常：

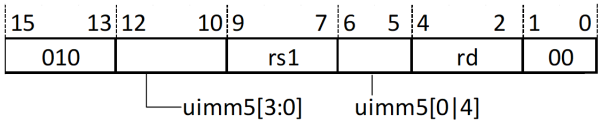
加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常

说明：

rs1/rd 编码代表寄存器如下:

- 000: x8
- 001: x9
- 010: x10
- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

指令格式：



18.5.24　C.LWSP—字堆栈加载指令

语法：

c.lwsp rd, uimm6 << 2(sp)

操作：

address ← sp+ zero_extend(uimm6 << 2)

tmp[31:0] ←mem[address+3:address]

rd ←sign_extend(tmp[31:0])

执行权限：

M mode/S mode/U mode

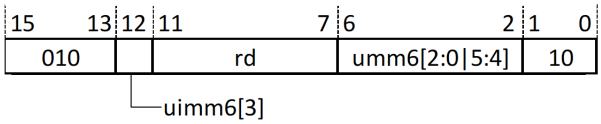
异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常

说明：

rd != 0

指令格式：



18.5.25　C.MV—数据传送指令

语法：

c.mv rd, rs2

操作：

rd ← rs2;

执行权限：

M mode/S mode/U mode

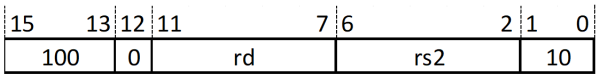
异常：

无

说明：

rs2 != 0, rd !=0。

指令格式：



18.5.26　C.NOP—空指令

语法：

c.nop

操作：

无操作

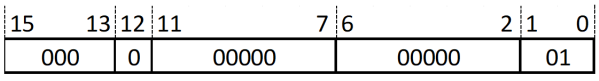
执行权限：

M mode/S mode/U mode

异常：

无

指令格式：



18.5.27　C.OR—按位或指令

语法：

c.or rd, rs2

操作：

rd ← rs1 | rs2

执行权限：

M mode/S mode/U mode

异常：

无

说明：

rs1 = rd

rd/rs1 编码代表寄存器如下：

- 000: x8
- 001: x9
- 010: x10
- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

指令格式：

15　13	12	11　10	9　7	6　5	4　2	1　0
100	0	11	rs1/rd	10	rs2	01

18.5.28　C.SD—双字存储指令

语法：

c.sd rs2, uimm5 << 3(rs1)

操作：

address ← rs1+ zero_extend(uimm5 << 3)

mem[address+7:address] ←rs2

执行权限：

M mode/S mode/U mode

异常：

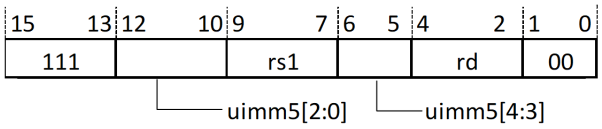
存储指令非对齐访问异常、存储指令访问错误异常、存储指令页面错误异常

说明：

rs1/rd 编码代表寄存器如下:

- 000: x8
- 001: x9
- 010: x10
- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

指令格式：



18.5.29　C.SDSP—双字堆栈存储指令

语法：

c.fsdsp rs2, uimm6 << 3(sp)

操作：

address ← sp+ zero_extend(uimm6 << 3)

mem[address+7:address] ←rs2

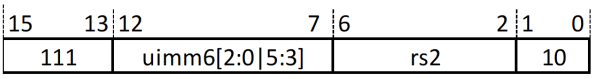
执行权限：

M mode/S mode/U mode

异常：

存储指令非对齐访问异常、存储指令访问错误异常、存储指令页面错误异常

指令格式：



18.5.30　C.SLLI—立即数逻辑左移指令

语法：

c.slli rd, nzuimm6

操作：

rd ←rs1 << nzuimm6

执行权限：

M mode/S mode/U mode

异常：

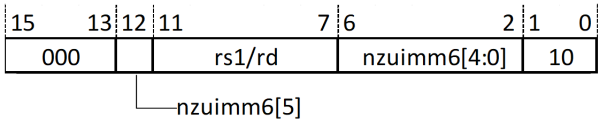
无

说明：

rs1==rd

rd/rs1 != 0，nzuimm6 != 0

指令格式：



18.5.31　C.SRAI—立即数算数右移指令

语法：

c.srli rd, nzuimm6

操作：

rd ←rs1 >> nzuimm6

执行权限：

M mode/S mode/U mode

异常：

无

说明：

nzuimm6 != 0

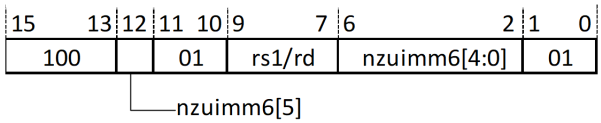
rs1 == rd

rs1/rd 编码代表寄存器如下:

- 000: x8
- 001: x9

- 010: x10
- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

指令格式：



18.5.32　C.SRLI—立即数逻辑右移指令

语法：

c.srli rd, nzuimm6

操作：

rd ←rs1 >> nzuimm6

执行权限：

M mode/S mode/U mode

异常：

无

说明：

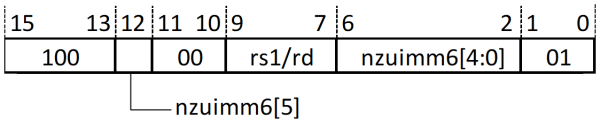
nzuimm6 != 0

rs1 == rd

rs1/rd 编码代表寄存器如下:

- 000: x8
- 001: x9
- 010: x10
- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

指令格式：



18.5.33　C.SW—字存储指令

语法：

c.sw rs2, uimm5 << 2(rs1)

操作：

address ← rs1+ zero_extend(uimm5 << 2)

mem[address+3:address] ←rs2

执行权限：

M mode/S mode/U mode

异常：

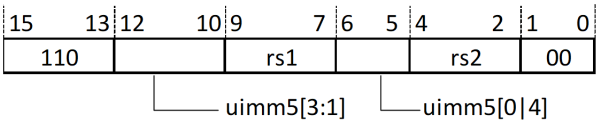
存储指令非对齐访问异常、存储指令访问错误异常、存储指令页面错误异常

说明：

rs1/rs2 编码代表寄存器如下:

- 000: x8
- 001: x9
- 010: x10
- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

指令格式：



18.5.34　C.SWSP—字堆栈存储指令

语法：

c.swsp rs2, uimm6 << 2(sp)

操作：

address ← sp+ zero_extend(uimm6 << 2)

mem[address+3:address] ←rs2

执行权限：

M mode/S mode/U mode

异常：

存储指令非对齐访问异常、存储指令访问错误异常、存储指令页面错误异常

指令格式：

15　13	12　7	6　2	1　0
110	uimm6[3:0|5:4]	rs2	10

18.5.35　C.SUB—有符号减法指令

语法：

c.sub rd, rs2

操作：

rd ← rs1 - rs2

执行权限：

M mode/S mode/U mode

异常：

无

说明：

rs1 == rd

rs1/rd 编码代表寄存器如下:

- 000: x8
- 001: x9
- 010: x10
- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

指令格式：

15　13	12	11　10	9　7	6　5	4　2	1　0
100	0	11	rs1/rd	00	rs2	01

18.5.36　C.SUBW——低 32 位有符号减法指令

语法：

c.subw rd, rs2

操作：

tmp[31:0] ← rs1[31:0] - rs2[31:0]

rd ←sign_extend(tmp)

执行权限：

M mode/S mode/U mode

异常：

无

说明：

rs1 == rd

rs1/rd 编码代表寄存器如下:

- 000: x8
- 001: x9
- 010: x10
- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

指令格式

15　13	12	11　10	9　7	6　5	4　2	1　0
100	1	11	rs1/rd	00	rs2	01

18.5.37　C.XOR——按位异或指令

语法：

c.xor rd, rs2

操作：

rd ← rs1 ^ rs2

执行权限：

M mode/S mode/U mode

异常：

无

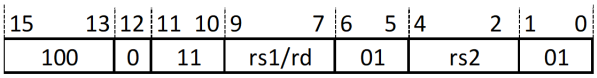
说明：

rs1 == rd

rs1/rd 编码代表寄存器如下：

- 000: x8
- 001: x9
- 010: x10
- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

指令格式：



15 13	12	11 10	9 7	6 5	4 2	1 0
100	0	11	rs1/rd	01	rs2	01

18.6　附录 A-8 伪指令列表

RISC-V 实现了一系列的伪指令，在此列出仅供参考，按英文字母顺序排列。

表 18.1: RISC-V 伪指令列表

伪指令	基础指令	含义
beqz rs, offset	beq rs, x0, offset	寄存器为零分支跳转
bnez rs, offset	bne rs, x0, offset	寄存器不为零分支跳转
blez rs, offset	bge x0,rs,offset	寄存器小于等于零跳转
bgez rs, offset	bge rs, x0, offset	寄存器大于等于零跳转
bltz rs, offset	blt rs, x0, offset	寄存器小于零跳转
bgtz rs, offset	blt x0, xs, offset	寄存器大于零跳转
bgt rs, rt, offset	blt rt, rs, offset	比较大于分支跳转
ble rs, rt, offset	bge rt, rs, offset	比较小于等于分支跳转
bgtu rs, rt, offset	bltu rt, rs, offset	无符号比较大于分支跳转

续下页

表 18.1 - 接上页

伪指令	基础指令	含义
bleu rs, rt, offset	bgeu rt, rs, offset	无符号比较小于等于分支跳转
call offset	auipc x6, offset[31:12] jalr x1, x6, offset[11:0]	跳转 4KB-4GB 空间的函数
csrc csr, rs	csrrc x0, csr, rs	清除控制寄存器中对应比特
csrci csr, imm	csrrci x0, csr, imm	清除控制寄存器低 5 位中对应比特
csrs csr, rs	csrrs x0, csr, rs	置位控制寄存器中对应比特
csrr, rd, csr	csrrs rd, csr, x0	读控制寄存器中对应比特
csrsi csr, imm	csrrsi x0, csr, imm	置位控制寄存器低 5 位中对应比特
csrw csr, rs	csrrw x0, csr, rs	写控制寄存器中对应比特
csrwi csr, imm	csrrwi x0, csr, imm	写控制寄存器低 5 位中对应比特
fabs.s rd , rs	fsgnjx.s rd, rs,rs	单精度浮点数取绝对值
fence	fence iorw, iorw	存储和外设同步指令
fl{w|d} rd, symbol, rt	auipc rt, symbol[31:12] fl{w|d} rd, symbol[11:0](rt)	4GB 地址空间浮点加载指令
fmv.s rd, rs	fsgnj.s rd, rs,rs	单精度浮点复制指令
fneg.s rd, rs	fsgnjn.s rd, rs,rs	单精度浮点取负指令
frcsr rd	csrrs x0, fcsr, x0	浮点控制寄存器读取指令
frflags rd	csrrs rd, fflags,x0	浮点异常位读取指令
frrm rd	csrrs rd, frm,x0	浮点舍入位读取指令
fscsr rs	csrrw x0, fcsr,rs	写浮点控制寄存器指令
fscsr rd, rs	csrrs rd, fcsr, rs	浮点控制寄存器读写指令
fsflags rs	csrrw x0, fcsr,rs	写浮点异常位指令
fsflags rd, rs	csrrs rd, fcsr, rs	浮点异常位读写指令
fsflagsi imm	csrrwi x0, fflags,imm	立即数写浮点异常位指令
fsflagsi rd, imm	csrrwi rd, fflags, imm	浮点异常位立即数读写指令
fsrm rs	csrrw x0, frm,rs	写浮点舍入位指令
fsrm rd, rs	csrrs rd, frm, rs	浮点舍入位读写指令
fsrmi imm	csrrwi x0, frm,imm	立即数写浮点舍入位指令
fsrmi rd, imm	csrrwi rd, frm, imm	浮点舍入位立即数读写指令
fs{w|d} rd, symbol, rt	auipc rt, symbol[31:12] fs{w|d} rd, symbol[11:0](rt)	4GB 地址空间浮点存储指令
j offset	jal x0, offset	直接跳转指令
jal offset	jal x1, offset	子程序跳转和链接指令

续下页

表 18.1 - 接上页

伪指令	基础指令	含义
jalr rs	jalr x1, rs, 0	子程序跳转寄存器和链接寄存器指令
jr rs	jalr x0, rs, 0	跳转寄存器指令
la rd, symbol	auipc rd, symbol[31:12] addi rd, rd, symbol[11:0]	指令地址加载指令
li rd, immediate	根据立即数大小拆分为多条指令	立即数加载指令
l{b|h|w|d} rd, symbol, rt	auipc rt, symbol[31:12] l{b|h|w|d} rd, symbol[11:0](rt)	4GB 地址空间加载指令
mv rd, rs	addi rd, rs, 0	数据传送指令
neg rd, rs	sub rd, x0, rs	寄存器取负指令
negw rd, rs	subw rd, x0, rs	寄存器低 32 位取负指令
nop	addi x0,x0,0	空指令
not rd, rs	xori rd, rs, -1	寄存器取反指令
rdcycle[h] rd	csrrs rd, cycle[h], x0	周期数读取指令
rdinstret[h] rd	csrrs rd, instret[h], x0	指令数读取指令
rdtime[h] rd	csrrs rd, time[h], x0	真实时钟读取指令
ret	jalr x0, x1,0	子程序返回指令
s{b|h|w|d} rd, symbol, rt	auipc rt, symbol[31:12] s{b|h|w|d} rd, symbol[11:0](rt)	4GB 地址空间存储指令
seqz rd, rs	sltiu rd, rs, 1	寄存器为 0 置 1 指令
sextw rd, rs	addiw rd, rs, 0	符号位扩展指令
sgtz rd, rs	slt rd, rs, x0, rs	寄存器大于 0 置 1 指令
sltz rd, rs	slt rd, rs, rs, x0	寄存器小于 0 置 1 指令
snez rd, rs	sltu rd, rs, x0, rs	寄存器不为 0 置 1 指令
tail offset	auipc x6, offset[31:12] jalr x0, x6, offset[11:0]	寄存器不链接跳转子程序指令

19 附录 B 玄铁扩展指令术语

C908X 除了支持 RV64GCB[V] 指令集之外，还在此基础上拓展了部分自定义指令，包括算术运算类指令、位操作类指令、内存访问类指令、SFU 扩展指令、Cache 指令子集、多核同步指令子集、半精度浮点指令集、AI 扩展指令、Vector 扩展指令、SCALAR 扩展指令和玄铁协处理器扩展指令。

C908X 的扩展指令集中，半精度浮点指令集可以直接使用，其它扩展指令使用前需要使能*机器模式扩展状态寄存器（MXSTATUS）*相应 bit，具体说明如下：

- 算术运算类指令、位操作类指令、内存访问类指令、SFU 扩展指令、AI 扩展指令、Vector 扩展指令、SCALAR 扩展指令在 mxstatus.XUANTIEISAEE == 1 时可以正常执行，否则将产生非法指令异常；
- 玄铁扩展 Cache 指令子集、多核同步指令子集在 mxstatus.XUANTIEISAEE == 1 或 mxstatus.COPINSTEE == 1 时可以正常执行；
- 玄铁协处理器扩展指令需要在 mxstatus.COPINSTEE == 1 时可以正常使用，否则将产生非法指令异常。

备注

- 在软件切换玄铁扩展和通用协处理器扩展指令过程中，需要先清原指令使能开关位，之后再打开新的指令开关位。同时设置玄铁扩展使能位和通用协处理器扩展指令使能位被认为无效操作，硬件保证不会有同时打开情况。

19.1 附录 B-1 Cache 指令术语

Cache 指令子集实现了对 cache 的操作，每条指令位宽为 32 位。

以下指令按英文字母顺序排列。

19.1.1 DCACHE.CALL——DCACHE 清全部脏表项指令

语法：

dcache.call

操作：

clear 所有 L1 dcache 表项，将所有 dirty 表项写回到下一级存储，仅操作当前核。

执行权限：

M mode/S mode

异常：

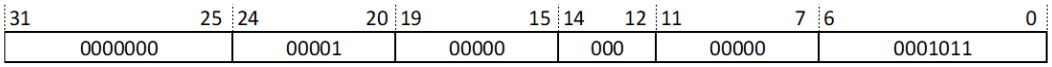
非法指令异常

说明：

mxstatus.xuantieisaee == 0 且 mxstatus.copinstee == 0 时，执行该指令产生非法指令异常。

U mode 下执行该指令产生非法指令异常。

指令格式：



31 25	24 20	19 15	14 12	11 7	6 0
0000000	00001	00000	000	00000	0001011

19.1.2　DCACHE.CIALL——DCACHE 清全部脏表项后无效指令

语法：

dcache.ciall

操作：

将所有 L1 dcache dirty 表项写回到下一级存储后，无效所有表项。

执行权限：

M mode/S mode

异常：

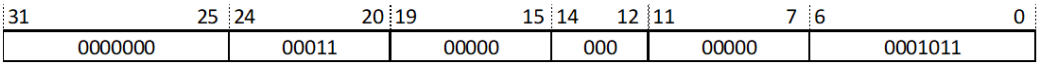
非法指令异常

说明：

mxstatus.xuantieisaee == 0 且 mxstatus.copinstee == 0 时，执行该指令产生非法指令异常。

U mode 下执行该指令产生非法指令异常。

指令格式：



31 25	24 20	19 15	14 12	11 7	6 0
0000000	00011	00000	000	00000	0001011

19.1.3　DCACHE.CIPA ——DCACHE 按物理地址清脏表项并无效

语法：

dcache.cipa rs1

操作：

将 rs1 中物理地址所属的 dcache/L2cache 表项写回下级存储并无效该表项，操作所有核和 L2CACHE。

执行权限：

M mode/S mode

异常：

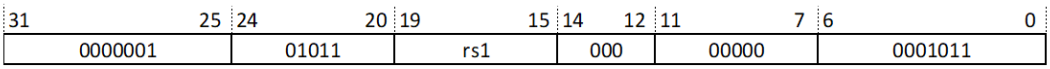
非法指令异常

说明：

mxstatus.xuantieisaee == 0 且 mxstatus.copinstee == 0 时，执行该指令产生非法指令异常。

U mode 下执行该指令产生非法指令异常。

指令格式：



31 25	24 20	19 15	14 12	11 7	6 0
0000001	01011	rs1	000	00000	0001011

19.1.4　DCACHE.CISW——DCACHE 按 way/set 清脏表项并无效指令

语法：

dcache.cisw rs1

操作：

按照 rs1 中指定的 way/set 将 L1 dache dirty 表项写回到下一级存储并无效该表项，仅操作当前核。

执行权限：

M mode/S mode

异常：

非法指令异常

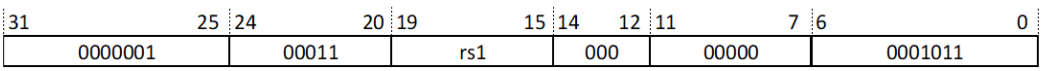
说明：

C908X dcache 为两路组相联，rs1[31] 为 way 编码，rs1[w:6] 为 set 编码。当 dcache 为 32K 时,w 为 13，dcache 为 64K 时，w 为 14。

mxstatus.xuantieisaee == 0 且 mxstatus.copinstee == 0 时，执行该指令产生非法指令异常。

U mode 下执行该指令产生非法指令异常。

指令格式：



31 25	24 20	19 15	14 12	11 7	6 0
0000001	00011	rs1	000	00000	0001011

19.1.5　DCACHE.CIVA——DCACHE 按虚拟地址清脏表项并无效

语法：

dcache.civa rs1

操作：

将 rs1 指定虚拟地址所属的 dcache/L2 cache 表项写回到下级存储，并无效该表项，操作当前核和 L2CACHE，并根据虚拟地址共享属性决定是否广播到其他核。

执行权限：

M mode/S mode/U mode

异常：

非法指令异常/加载指令页面错误异常

说明：

mxstatus.xuantieisaee == 0 且 mxstatus.copinstee == 0 时，执行该指令产生非法指令异常。

mxstatus.xuantieisaee=1，mxstatus.ucme =1，U mode 下可以执行该指令。

mxstatus.xuantieisaee=1，mxstatus.ucme =0，U mode 下执行该指令产生非法指令异常。

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
0000001	00111	rs1	000	00000	0001011

19.1.6　DCACHE.CPA——DCACHE 按物理地址清脏表项

语法：

dcache.cpa rs1

操作：

将 rs1 中物理地址所对应的 dcache/l2cache 表项写回到下一级存储，操作所有核和 L2CACHE。

执行权限：

M mode/S mode

异常：

非法指令异常

说明：

mxstatus.xuantieisaee == 0 且 mxstatus.copinstee == 0 时，执行该指令产生非法指令异常。

U mode 下执行该指令产生非法指令异常。

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
0000001	01001	rs1	000	00000	0001011

19.1.7　DCACHE.CPAL1 ——L1DCACHE 按物理地址清脏表项

语法：

dcache.cpal1 rs1

操作：

将 rs1 中物理地址所对应的 dcache 表项写回到下一级存储，操作所有核 L1CACHE。

执行权限：

M mode/S mode

异常：

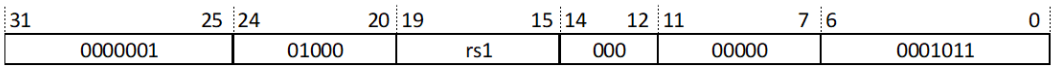
非法指令异常

说明：

mxstatus.xuantieisaee == 0 且 mxstatus.copinstee == 0 时，执行该指令产生非法指令异常。

U mode 下执行该指令产生非法指令异常。

指令格式：



31 25	24 20	19 15	14 12	11 7	6 0
0000001	01000	rs1	000	00000	0001011

19.1.8　DCACHE.CVA——DCACHE 按虚拟地址清脏表项

语法：

dcache.cva rs1

操作：

将 rs1 中虚拟地址所对应的 dcache/l2cache 表项写回到下一级存储，操作当前核和 L2CACHE，并根据虚拟地址共享属性决定是否广播到其他核

执行权限：

M mode/S mode

异常：

非法指令异常/加载指令页面错误异常

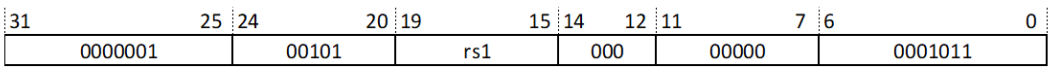
说明：

mxstatus.xuantieisaee == 0 且 mxstatus.copinstee == 0 时，执行该指令产生非法指令异常。

mxstatus.xuantieisaee=1，mxstatus.ucme=1，U mode 下可以执行该指令。

mxstatus.xuantieisaee=1，mxstatus.ucme=0，U mode 下执行该指令产生非法指令异常。

指令格式：



31 25	24 20	19 15	14 12	11 7	6 0
0000001	00101	rs1	000	00000	0001011

19.1.9　DCACHE.CVAL1——L1DCACHE 按虚拟地址清脏表项

语法：

dcache.cval1 rs1

操作：

将 rs1 中虚拟地址所对应的 dcache 表项写回到下一级存储，操作所有核 L1CACHE

执行权限：

M mode/S mode/U mode

异常：

非法指令异常/加载指令页面错误异常

说明：

mxstatus.xuantieisaee == 0 且 mxstatus.copinstee == 0 时，执行该指令产生非法指令异常。

mxstatus.ucme = 0，U mode 下执行该指令产生非法指令异常。

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
0000001	00100	rs1	000	00000	0001011

19.1.10　DCACHE.IALL——DCACHE 无效所有表项指令

语法：

dcache.iall

操作：

无效所有 L1 dcache 表项，仅操作当前核。

执行权限：

M mode/S mode

异常：

非法指令异常

说明：

- mxstatus.xuantieisaee == 0 且 mxstatus.copinstee == 0 时，执行该指令产生非法指令异常。
- U mode 下执行该指令产生非法指令异常。

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
0000000	00010	00000	000	00000	0001011

19.1.11　DCACHE.IPA ——DCACHE 按物理地址无效指令

语法：

dcache.ipa rs1

操作：

将 rs1 中物理地址所对应的 dcache/l2 cache 表项无效，操作所有核和 L2CACHE。

执行权限：

M mode/S mode

异常：

非法指令异常

说明：

mxstatus.xuantieisaee == 0 且 mxstatus.copinstee == 0 时，执行该指令产生非法指令异常。

U mode 下执行该指令产生非法指令异常。

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
0000001	01010	rs1	000	00000	0001011

19.1.12　DCACHE.CSW —DCACHE 按 set/way 清脏表项

语法:

dcache.csw rs1

操作:

按 SET 和 WAY 将 dcache 中的脏表项回写到下一级存储器

执行权限:

M mode/S mode

异常:

非法指令异常

说明：

C908X dcache 为两路组相联，rs1[31] 为 way 编码，rs1[w:6] 为 set 编码。当 dcache 为 32K 时,w 为 13，dcache 为 64K 时，w 为 14。

mxstatus.xuantieisaee == 0 且 mxstatus.copinstee == 0 时，执行该指令产生非法指令异常。

U mode 下执行该指令产生非法指令异常。

指令格式:

31 25	24 20	19 15	14 12	11 7	6 0
0000001	00001	rs1	000	00000	0001011

图 19.1: DCACHE.CSW

19.1.13　DCACHE.ISW —DCACHE 按 set/way 无效指令

语法：

dcache.isw rs1

操作：

无效指定 SET 和 WAY 的 dcache 表项，仅操作当前核。

执行权限：

M mode/S mode

异常：

非法指令异常

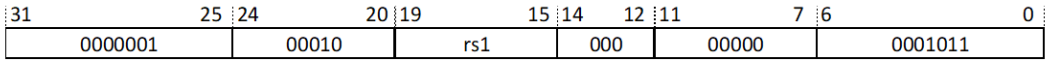
说明：

C908X dcache 为两路组相联，rs1[31] 为 way 编码，rs1[w:6] 为 set 编码。当 dcache 为 32K 时,w 为 13，dcache 为 64K 时，w 为 14。

mxstatus.xuantieisaee == 0 且 mxstatus.copinstee == 0 时，执行该指令产生非法指令异常。

U mode 下执行该指令产生非法指令异常。

指令格式：



31 25	24 20	19 15	14 12	11 7	6 0
0000001	00010	rs1	000	00000	0001011

19.1.14　DCACHE.IVA —DCACHE 按虚拟地址无效指令

语法：

dcache.iva rs1

操作：

将 rs1 中虚拟地址所对应的 dcache/l2 cache 表项无效，操作当前核和 L2CACHE，并根据虚拟地址共享属性决定是否广播到其他核。

执行权限：

M mode/S mode

异常：

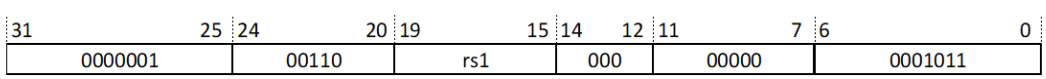
非法指令异常/加载指令页面错误异常

说明：

mxstatus.xuantieisaee == 0 且 mxstatus.copinstee == 0 时，执行该指令产生非法指令异常。

U mode 下执行该指令产生非法指令异常。

指令格式：



31 25	24 20	19 15	14 12	11 7	6 0
0000001	00110	rs1	000	00000	0001011

19.1.15　ICACHE.IPA——ICACHE 按物理地址无效表项指令

语法：

icache.ipa rs1

操作：

将 rs1 中物理地址所对应的 icache 表项无效，操作所有核。

执行权限：

M mode/S mode

异常：

非法指令异常

说明：

mxstatus.xuantieisaee == 0 且 mxstatus.copinstee == 0 时，执行该指令产生非法指令异常。

U mode 下执行该指令产生非法指令异常。

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
0000001	11000	rs1	000	00000	0001011

19.1.16　ICACHE.IVA——ICACHE 按虚拟地址无效表项指令

语法：

icache.iva rs1

操作：

将 rs1 中虚拟地址所对应的 icache 表项无效，操作当前核，并根据虚拟地址共享属性决定是否广播到其他核。

执行权限：

M mode/S mode/U mode

异常：

非法指令异常/加载指令页面错误异常

说明：

mxstatus.xuantieisaee == 0 且 mxstatus.copinstee == 0 时，执行该指令产生非法指令异常。

mxstatus.ucme=1，U mode 下可以执行该指令。

mxstatus.ucme=0，U mode 下执行该指令产生非法指令异常。

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
0000001	10000	rs1	000	00000	0001011

19.1.17　ICACHE.IALL——ICACHE 无效所有表项指令

语法：

icache.iall

操作：

无效所有 icache 表项，仅操作当前核。

执行权限：

M mode/S mode

异常：

非法指令异常

说明：

mxstatus.xuantieisaee == 0 且 mxstatus.copinstee == 0 时，执行该指令产生非法指令异常。

U mode 下执行该指令产生非法指令异常。

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
0000000	10000	00000	000	00000	0001011

19.1.18　ICACHE.IALLS——ICACHE 广播无效所有表项指令

语法：

icache.ialls

操作：

无效所有 icache 表项，并广播其他核去无效各自所有 icache 表项，操作所有核。

执行权限：

M mode/S mode

异常：

非法指令异常

说明：

mxstatus.xuantieisaee == 0 且 mxstatus.copinstee == 0 时，执行该指令产生非法指令异常。

U mode 下执行该指令产生非法指令异常。

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
0000000	10001	00000	000	00000	0001011

19.2　附录 B-2 多核同步指令术语

同步指令子集实现了多核同步指令的扩展，每条指令位宽为 32 位，以下指令按英文字母顺序排列。

19.2.1　SYNC—同步指令

语法：

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

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

说明：

mxstatus.xuantieisaee == 0 且 mxstatus.copinstee == 0 时，执行该指令产生非法指令异常。

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
0000000	11000	00000	000	00000	0001011

19.2.2　SYNC.I—同步清空指令

语法：

sync.i

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, 该指令退休时清空流水线

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

说明：

mxstatus.xuantieisaee == 0 且 mxstatus.copinstee == 0 时，执行该指令产生非法指令异常。

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
0000000	11010	00000	000	00000	0001011

19.2.3　SYNC.IS—同步清空广播指令

语法：

sync.is

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, 该指令退休时清空流水线，并将该请求广播给其他核

执行权限：

M mode/S mode/U mode

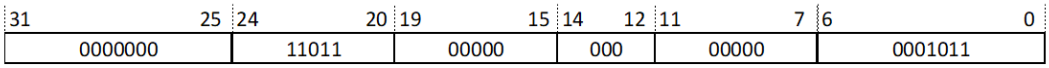
异常：

非法指令异常

说明：

mxstatus.xuantieisaee == 0 且 mxstatus.copinstee == 0 时，执行该指令产生非法指令异常。

指令格式：



31 25	24 20	19 15	14 12	11 7	6 0
0000000	11011	00000	000	00000	0001011

19.2.4　SYNC.S—同步广播指令

语法：

sync.s

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, 并将该请求广播给其他核

执行权限：

M mode/S mode/U mode

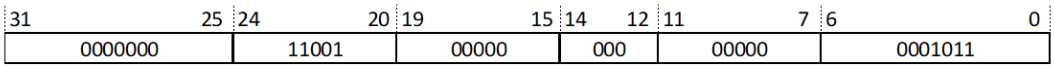
异常：

非法指令异常

说明：

mxstatus.xuantieisaee == 0 且 mxstatus.copinstee == 0 时，执行该指令产生非法指令异常。

指令格式：



31 25	24 20	19 15	14 12	11 7	6 0
0000000	11001	00000	000	00000	0001011

19.3　附录 B-3 算术运算指令术语

算术运算指令子集实现了对算术指令的扩展，每条指令位宽为 32 位。

以下指令按英文字母顺序排列。

19.3.1　ADDSL——寄存器移位相加指令

语法：

addsl rd rs1, rs2, imm2

操作：

rd ← rs1+ rs2 << imm2

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00000	imm2	rs2	rs1	001	rd	0001011

19.3.2　MULA——乘累加指令

语法：

mula rd, rs1, rs2

操作：

rd ← rd+ (rs1 * rs2)[63:0]

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00100	00	rs2	rs1	001	rd	0001011

19.3.3　MULAH——低 16 位乘累加指令

语法：

mulah rd, rs1, rs2

操作：

tmp[31:0] ← rd[31:0]+ (rs1[15:0] * rs[15:0])

rd ←sign_extend(tmp[31:0])

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00101	00	rs2	rs1	001	rd	0001011

19.3.4　MULAW——低 32 位乘累加指令

语法：

mulaw rd, rs1, rs2

操作：

tmp[31:0] ← rd[31:0]+ (rs1[31:0] * rs[31:0])[31:0]

rd ←sign_extend(tmp[31:0])

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00100	10	rs2	rs1	001	rd	0001011

19.3.5　MULS——乘累减指令

语法：

muls rd, rs1, rs2

操作：

rd ← rd- (rs1 * rs2)[63:0]

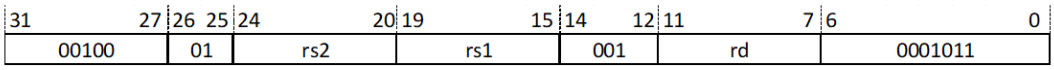
执行权限：

M mode/S mode/U mode

异常：

非法指令异常

指令格式：



19.3.6　MULSH——低 16 位乘累减指令

语法：

mulsh rd, rs1, rs2

操作：

tmp[31:0] ← rd[31:0]- (rs1[15:0] * rs[15:0])

rd ←sign_extend(tmp[31:0])

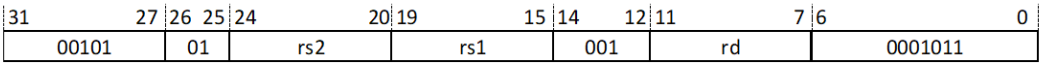
执行权限：

M mode/S mode/U mode

异常：

非法指令异常

指令格式：



19.3.7　MULSW——低 32 位乘累减指令

语法：

mulsw rd, rs1, rs2

操作：

tmp[31:0] ← rd[31:0]- (rs1[31:0] * rs[31:0])

rd ←sign_extend(tmp[31:0])

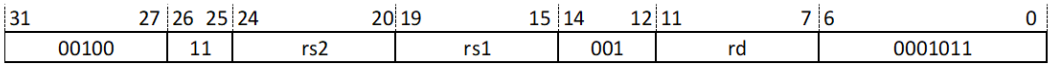
执行权限：

M mode/S mode/U mode

异常：

非法指令异常

指令格式：



19.3.8　MVEQZ——寄存器为 0 传送指令

语法：

mveqz rd, rs1, rs2

操作：

if (rs2 == 0)

rd ← rs1

else

rd ← rd

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
01000	00	rs2	rs1	001	rd	0001011

19.3.9　MVNEZ——寄存器非 0 传送指令

语法：

mvnez rd, rs1, rs2

操作：

if (rs2 != 0)

rd ← rs1

else

rd ← rd

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
01000	01	rs2	rs1	001	rd	0001011

19.3.10　SRRI—循环右移指令

语法：

srri rd, rs1, imm6

操作：

rd ← rs1 >> imm6

rs1 原值右移，左侧移入右侧移出位

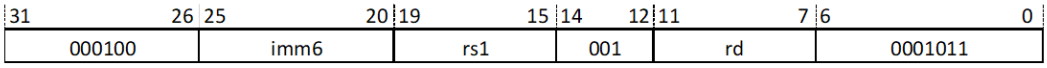
执行权限：

M mode/S mode/U mode

异常：

非法指令异常

指令格式：



19.3.11　SRRIW—低 32 位循环右移指令

语法：

srriw rd, rs1, imm5

操作：

rd ← sign_extend(rs1[31:0] >> imm5)

rs1[31:0] 原值右移，左侧移入右侧移出位

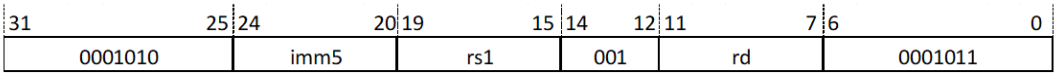
执行权限：

M mode/S mode/U mode

异常：

非法指令异常

指令格式：



19.4　附录 B-4 位操作指令术语

位操作指令子集实现了对位运算指令的扩展，每条指令位宽为 32 位。

以下指令按英文字母顺序排列。

19.4.1　EXT——寄存器连续位提取符号位扩展指令

语法：

ext rd, rs1, imm1,imm2

操作：

rd←sign_extend(rs1[imm1:imm2])

执行权限：

M mode/S mode/U mode

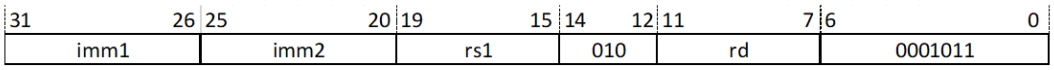
异常：

非法指令异常

说明：

若 imm1<imm2，该指令行为不可预测

指令格式：



19.4.2　EXTU——寄存器连续位提取零扩展指令

语法：

extu rd, rs1, imm1,imm2

操作：

rd←zero_extend(rs1[imm1:imm2])

执行权限：

M mode/S mode/U mode

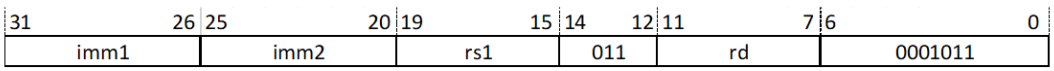
异常：

非法指令异常

说明：

若 imm1<imm2，该指令行为不可预测

指令格式：



19.4.3　FF0——快速找 0 指令

语法：

ff0 rd, rs1

操作：

从 rs1 最高位开始查找第一个为 0 的位，结果写回 rd。如果 rs1 的最高位为 0，则结果为 0，如果 rs1 中没有 0，结果为 64

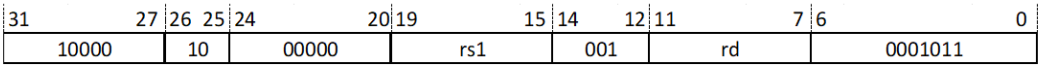
执行权限：

M mode/S mode/U mode

异常：

非法指令异常

指令格式：



31 27	26 25	24 20	19 15	14 12	11 7	6 0
10000	10	00000	rs1	001	rd	0001011

19.4.4　FF1——快速找 1 指令

语法：

ff1 rd, rs1

操作：

从 rs1 最高位开始查找第一个为 1 的位，将该位的索引写回 rd。如果 rs1 的最高位为 1，则结果为 0，如果 rs1 中没有 1，结果为 64

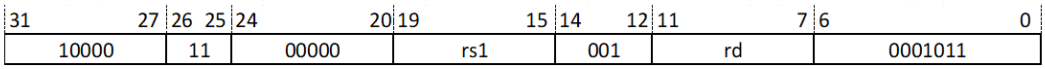
执行权限：

M mode/S mode/U mode

异常：

非法指令异常

指令格式：



31 27	26 25	24 20	19 15	14 12	11 7	6 0
10000	11	00000	rs1	001	rd	0001011

19.4.5　REV——字节倒序指令

语法：

rev rd, rs1

操作：

rd[63:56] ←rs1[7:0]

rd[55:48] ←rs1[15:8]

rd[47:40] ←rs1[23:16]

rd[39:32] ←rs1[31:24]

rd[31:24] ←rs1[39:32]

rd[23:16] ←rs1[47:40]

rd[15:8] ←rs1[55:48]

rd[7:0] ←rs1[63:56]

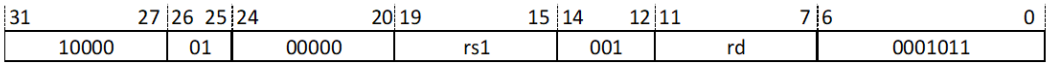
执行权限：

M mode/S mode/U mode

异常：

非法指令异常

指令格式：



31 27	26 25	24 20	19 15	14 12	11 7	6 0
10000	01	00000	rs1	001	rd	0001011

19.4.6　REVW——低 32 位字节倒序指令

语法：

revw rd, rs1

操作：

tmp[31:24] ←rs1[7:0]

tmp [23:16] ←rs1[15:8]

tmp [15:8] ←rs1[23:16]

tmp [7:0] ←rs1[31:24]

rd ←sign_extend(tmp[31:0])

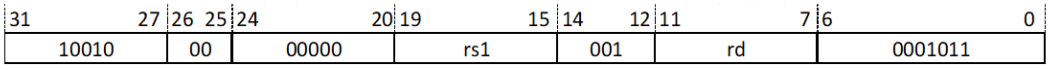
执行权限：

M mode/S mode/U mode

异常：

非法指令异常

指令格式：



31 27	26 25	24 20	19 15	14 12	11 7	6 0
10010	00	00000	rs1	001	rd	0001011

19.4.7　TST——比特为 0 测试指令

语法：

tst rd, rs1, imm6

操作：

if(rs1[imm6] == 1)

rd←1

else

rd←0

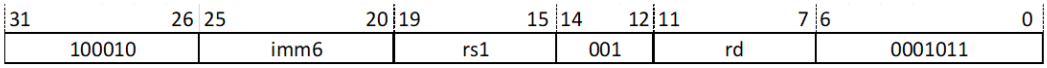
执行权限：

M mode/S mode/U mode

异常：

非法指令异常

指令格式：



31 26	25 20	19 15	14 12	11 7	6 0
100010	imm6	rs1	001	rd	0001011

19.4.8　TSTNBZ——字节为 0 测试指令

语法：

tstnbz rd, rs1

操作：

rd[63:56] ← (rs1[63:56] == 0) ? 8'hff : 8'h0

rd[55:48] ← (rs1[55:48] == 0) ? 8'hff : 8'h0

rd[47:40] ← (rs1[47:40] == 0) ? 8'hff : 8'h0

rd[39:32] ← (rs1[39:32] == 0) ? 8'hff : 8'h0

rd[31:24] ← (rs1[31:24] == 0) ? 8'hff : 8'h0

rd[23:16] ← (rs1[23:16] == 0) ? 8'hff : 8'h0

rd[15:8] ← (rs1[15:8] == 0) ? 8'hff : 8'h0

rd[7:0] ← (rs1[7:0] == 0) ? 8'hff : 8'h0

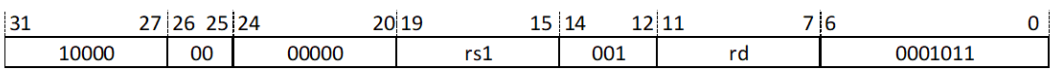
执行权限：

M mode/S mode/U mode

异常：

非法指令异常

指令格式：



31 27	26 25	24 20	19 15	14 12	11 7	6 0
10000	00	00000	rs1	001	rd	0001011

19.5　附录 B-5 存储指令术语

存储指令子集实现了对存储指令的扩展，每条指令位宽为 32 位。

以下指令按英文字母顺序排列。

19.5.1　FLRD——浮点寄存器移位双字加载指令

语法：

flrd rd, rs1, rs2, imm2

操作：

rd ←mem[(rs1+rs2 << imm2)+7: (rs1+rs2 << imm2)]

执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常, 非法指令异常

说明：

mxstatus.xuantieisaee=1’b0 或 mstatus.fs =2’b00 时，该指令产生非法指令异常

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
01100	imm2	rs2	rs1	110	rd	0001011

19.5.2　FLRW——浮点寄存器移位字加载指令

语法：

flrw rd, rs1, rs2, imm2

操作：

rd ←one_extend(mem[(rs1+rs2 << imm2)+3: (rs1+rs2 << imm2)])

执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常，非法指令异常

说明：

mxstatus.xuantieisaee=1’b0 或 mstatus.fs =2’b00 时，该指令产生非法指令异常

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
01000	imm2	rs2	rs1	110	rd	0001011

19.5.3　FLURD——浮点寄存器低 32 位移位双字加载指令

语法：

flurd rd, rs1, rs2, imm2

操作：

rd ←mem[(rs1+rs2[31:0] << imm2)+7: (rs1+rs2[31:0] << imm2)]

执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常、非法指令异常

说明：

rs2[31:0] 是无符号数，高位 [63:32] 补 0 进行地址运算

mxstatus.xuanTIEISAEE=1'b0 或 mstatus.fs = 2'b00 时，该指令产生非法指令异常

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
01110	imm2	rs2	rs1	110	rd	0001011

19.5.4　FLURW——浮点寄存器低 32 位移位字加载指令

语法：

flurw rd, rs1, rs2, imm2

操作：

rd ←one_extend(mem[(rs1+rs2[31:0] << imm2)+3: (rs1+rs2[31:0] << imm2)])

执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常、非法指令异常

说明：

rs2[31:0] 是无符号数，高位 [63:32] 补 0 进行地址运算

mxstatus.xuantieisaee=1'b0 或 mstatus.fs = 2'b00 时，该指令产生非法指令异常

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
01010	imm2	rs2	rs1	110	rd	0001011

19.5.5　FSRD——浮点寄存器移位双字存储指令

语法：

fsrd rd, rs1, rs2, imm2

操作：

mem[(rs1+rs2 << imm2)+7: (rs1+rs2 << imm2)] ←rd[63:0]

执行权限：

M mode/S mode/U mode

异常：

存储指令非对齐访问异常、存储指令访问错误异常、存储指令页面错误异常、非法指令异常

说明：

mxstatus.xuantieisaee=1'b0 或 mstatus.fs =2'b00 时，该指令产生非法指令异常

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
01100	imm2	rs2	rs1	111	rd	0001011

19.5.6　FSRW——浮点寄存器移位字存储指令

语法：

fsrw rd, rs1, rs2, imm2

操作：

mem[(rs1+rs2 << imm2)+3: (rs1+rs2 << imm2)] ←rd[31:0]

执行权限：

M mode/S mode/U mode

异常：

存储指令非对齐访问异常、存储指令访问错误异常、存储指令页面错误异常、非法指令异常

说明：

mxstatus.xuantieisaee=1'b0 或 mstatus.fs =2'b00 时，该指令产生非法指令异常

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
01000	imm2	rs2	rs1	111	rd	0001011

19.5.7　FSURD——浮点寄存器低 32 位移位双字存储指令

语法：

fsurd rd, rs1, rs2, imm2

操作：

mem[(rs1+rs2[31:0] << imm2)+7: (rs1+rs2[31:0] << imm2)] ←rd[63:0]

执行权限：

M mode/S mode/U mode

异常：

存储指令非对齐访问异常、存储指令访问错误异常、存储指令页面错误异常、非法指令异常

说明：

rs2[31:0] 是无符号数，高位 [63:32] 补 0 进行地址运算

mxstatus.xuantieisaee=1’b0 或 mstatus.fs = 2’b00 时，该指令产生非法指令异常

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
01110	imm2	rs2	rs1	111	rd	0001011

19.5.8　FSURW—浮点寄存器低 32 位移位字存储指令

语法：

fsurw rd, rs1, rs2, imm2

操作：

mem[(rs1+rs2[31:0] << imm2)+3: (rs1+rs2[31:0] << imm2)] ←rd[31:0]

执行权限：

M mode/S mode/U mode

异常：

存储指令非对齐访问异常、存储指令访问错误异常、存储指令页面错误异常、非法指令异常

说明：

rs2[31:0] 是无符号数，高位 [63:32] 补 0 进行地址运算

mxstatus.xuantieisaee=1’b0 或 mstatus.fs = 2’b00 时，该指令产生非法指令异常

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
01010	imm2	rs2	rs1	111	rd	0001011

19.5.9　LBIA—符号位扩展字节加载基地址自增指令

语法：

lbia rd, (rs1), imm5,imm2

操作：

rd ←sign_extend(mem[rs1])

rs1←rs1 + sign_extend(imm5 << imm2)

执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常、非法指令异常。

说明：

rd 和 rs1 不可相等

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00011	imm2	imm5	rs1	100	rd	0001011

19.5.10　LBIB—基地址自增符号位扩展字节加载指令

语法：

lbib rd, (rs1), imm5,imm2

操作：

rs1←rs1 + sign_extend(imm5 << imm2)

rd ←sign_extend(mem[rs1])

执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常、非法指令异常。

说明：

rd 和 rs1 不可相等

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00001	imm2	imm5	rs1	100	rd	0001011

19.5.11　LBUIA—零扩展字节加载基地址自增指令

语法：

lbuia rd, (rs1), imm5,imm2

操作：

rd ←zero_extend(mem[rs1])

rs1←rs1 + sign_extend(imm5 << imm2)

执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常、非法指令异常。

说明：

rd 和 rs1 不可相等

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
10011	imm2	imm5	rs1	100	rd	0001011

19.5.12　LBUIB——基地址自增零扩展字节加载指令

语法：

lbuib rd, (rs1), imm5,imm2

操作：

rs1←rs1 + sign_extend(imm5 << imm2)

rd ←zero_extend(mem[rs1])

执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常、非法指令异常。

说明：

rd 和 rs1 不可相等

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
10001	imm2	imm5	rs1	100	rd	0001011

19.5.13　LDD——双寄存器加载指令

语法：

ldd rd1,rd2, (rs1),imm2, 4

操作：

address←rs1 + zero_extend(imm2 << 4)

rd1←mem[address+7:address]

rd2←mem[address+15:address+8]

执行权限：

M mode/S mode/U mode

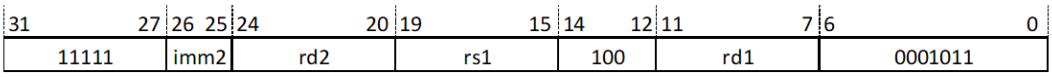
异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常、非法指令异常

说明：

rd1,rd2 ,rs1 互相不可相等

指令格式：



19.5.14　LDIA—符号位扩展双字加载基地址自增指令

语法：

ldia rd, (rs1), imm5,imm2

操作：

rd ←sign_extend(mem[rs1+7:rs1])

rs1←rs1 + sign_extend(imm5 << imm2)

执行权限：

M mode/S mode/U mode

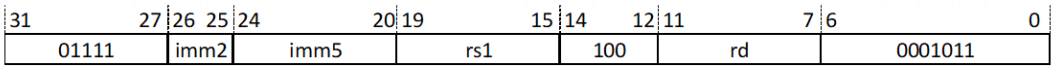
异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常、非法指令异常。

说明：

rd 和 rs1 不可相等

指令格式：



19.5.15　LDIB—基地址自增符号位扩展双字加载指令

语法：

ldib rd, (rs1), imm5,imm2

操作：

rs1←rs1 + sign_extend(imm5 << imm2)

rd ←sign_extend(mem[rs1+7:rs1])

执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常、非法指令异常。

说明：

rd 和 rs1 不可相等

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
01101	imm2	imm5	rs1	100	rd	0001011

19.5.16　LHIA—符号位扩展半字加载基地址自增指令

语法：

lhia rd, (rs1), imm5,imm2

操作：

rd ←sign_extend(mem[rs1+1:rs1])

rs1←rs1 + sign_extend(imm5 << imm2)

执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常、非法指令异常。

说明：

rd 和 rs1 不可相等

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00111	imm2	imm5	rs1	100	rd	0001011

19.5.17　LHIB—基地址自增符号位扩展半字加载指令

语法：

lhib rd, (rs1), imm5,imm2

操作：

rs1←rs1 + sign_extend(imm5 << imm2)

rd ←sign_extend(mem[rs1+1:rs1])

执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常、非法指令异常。

说明：

rd 和 rs1 不可相等

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00101	imm2	imm5	rs1	100	rd	0001011

19.5.18　LHUIA—零扩展半字加载基地址自增指令

语法：

lhuia rd, (rs1), imm5,imm2

操作：

rd ←zero_extend(mem[rs1+1:rs1])

rs1←rs1 + sign_extend(imm5 << imm2)

执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常、非法指令异常。

说明：

rd 和 rs1 不可相等

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
10111	imm2	imm5	rs1	100	rd	0001011

19.5.19　LHUIB—基地址自增零扩展半字加载指令

语法：

lhuib rd, (rs1), imm5,imm2

操作：

rs1←rs1 + sign_extend(imm5 << imm2)

rd ←zero_extend(mem[rs1+1:rs1])

执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常、非法指令异常。

说明：

rd 和 rs1 不可相等

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
10101	imm2	imm5	rs1	100	rd	0001011

19.5.20　LRB——寄存器移位符号位扩展字节加载指令

语法：

lrb rd, rs1, rs2, imm2

操作：

rd ←sign_extend(mem[(rs1+rs2 << imm2)])

执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常、非法指令异常

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00000	imm2	rs2	rs1	100	rd	0001011

19.5.21　LRBU——寄存器移位零扩展字节加载指令

语法：

lrbu rd, rs1, rs2, imm2

操作：

rd ←zero_extend(mem[(rs1+rs2 << imm2)])

执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常、非法指令异常

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
10000	imm2	rs2	rs1	100	rd	0001011

19.5.22　LRD——寄存器移位双字加载指令

语法：

lrd rd, rs1, rs2, imm2

操作：

rd ←mem[(rs1+rs2 << imm2)+7: (rs1+rs2 << imm2)]

执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常、非法指令异常

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
01100	imm2	rs2	rs1	100	rd	0001011

19.5.23　LRH——寄存器移位符号位扩展半字加载指令

语法：

lrh rd, rs1, rs2, imm2

操作：

rd ←sign_extend(mem[(rs1+rs2 << imm2)+1: (rs1+rs2 << imm2)])

执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常、非法指令异常

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00100	imm2	rs2	rs1	100	rd	0001011

19.5.24　LRHU——寄存器移位零扩展半字加载指令

语法：

lrhu rd, rs1, rs2, imm2

操作：

rd ←zero_extend(mem[(rs1+rs2 << imm2)+1: (rs1+rs2 << imm2)])

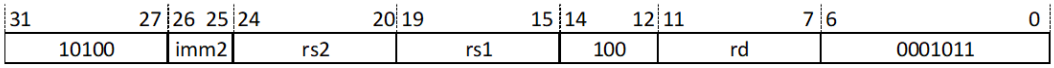
执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常、非法指令异常

指令格式：



31 27	26 25	24 20	19 15	14 12	11 7	6 0
10100	imm2	rs2	rs1	100	rd	0001011

19.5.25　LRW——寄存器移位符号位扩展字加载指令

语法：

lrw rd, rs1, rs2, imm2

操作：

rd ←sign_extend(mem[(rs1+rs2 << imm2)+3: (rs1+rs2 << imm2)])

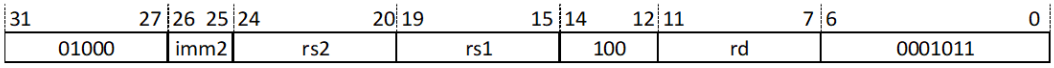
执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常、非法指令异常

指令格式：



31 27	26 25	24 20	19 15	14 12	11 7	6 0
01000	imm2	rs2	rs1	100	rd	0001011

19.5.26　LRWU——寄存器移位零扩展字加载指令

语法：

lrwu rd, rs1, rs2, imm2

操作：

rd ←zero_extend(mem[(rs1+rs2 << imm2)+3: (rs1+rs2 << imm2)])

执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常、非法指令异常

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
11000	imm2	rs2	rs1	100	rd	0001011

19.5.27　LURB——寄存器低 32 位移位符号位扩展字节加载指令

语法：

lurb rd, rs1, rs2, imm2

操作：

rd ←sign_extend(mem[(rs1+rs2[31:0] << imm2)])

执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常、非法指令异常

说明：

rs2[31:0] 是无符号数，高位 [63:32] 补 0 进行地址运算

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00010	imm2	rs2	rs1	100	rd	0001011

19.5.28　LURBU——寄存器低 32 位移位零扩展字节加载指令

语法：

lurbu rd, rs1, rs2, imm2

操作：

rd ←zero_extend(mem[(rs1+rs2[31:0] << imm2)])

执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常、非法指令异常

说明：

rs2[31:0] 是无符号数，高位 [63:32] 补 0 进行地址运算

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
10010	imm2	rs2	rs1	100	rd	0001011

19.5.29　LURD——寄存器低 32 位移位双字加载指令

语法：

lurd rd, rs1, rs2, imm2

操作：

rd ←mem[(rs1+rs2[31:0] << imm2)+7: (rs1+rs2[31:0] << imm2)]

执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常、非法指令异常

说明：

rs2[31:0] 是无符号数，高位 [63:32] 补 0 进行地址运算

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
01110	imm2	rs2	rs1	100	rd	0001011

19.5.30　LURH——寄存器低 32 位移位符号位扩展半字加载指令

语法：

lurh rd, rs1, rs2, imm2

操作：

rd ←sign_extend(mem[(rs1+rs2[31:0] << imm2)+1:

(rs1+rs2[31:0] << imm2)])

执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常、非法指令异常

说明：

rs2[31:0] 是无符号数，高位 [63:32] 补 0 进行地址运算

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00110	imm2	rs2	rs1	100	rd	0001011

19.5.31　LURHU——寄存器低 32 位移位零扩展半字加载指令

语法：

lurhu rd, rs1, rs2, imm2

操作：

rd ←zero_extend(mem[(rs1+rs2[31:0] << imm2)+1:

(rs1+rs2[31:0] << imm2)])

执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常、非法指令异常

说明：

rs2[31:0] 是无符号数，高位 [63:32] 补 0 进行地址运算

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
10110	imm2	rs2	rs1	100	rd	0001011

19.5.32　LURW——寄存器低 32 位移位符号位扩展字加载指令

语法：

lurw rd, rs1, rs2, imm2

操作：

rd ←sign_extend(mem[(rs1+rs2[31:0] << imm2)+3:

(rs1+rs2[31:0] << imm2)])

执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常、非法指令异常

说明：

rs2[31:0] 是无符号数，高位 [63:32] 补 0 进行地址运算

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
01010	imm2	rs2	rs1	100	rd	0001011

19.5.33　LURWU——寄存器低 32 位移位零扩展字加载指令

语法：

lurwu rd, rs1, rs2,imm2

操作：

rd ←zero_extend(mem[(rs1+rs2[31:0] << imm2)+3:(rs1+rs2[31:0] << imm2)])

执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常、非法指令异常

说明：

rs2[31:0] 是无符号数

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
11010	imm2	rs2	rs1	100	rd	0001011

19.5.34　LWD——符号位扩展双寄存器字加载指令

语法：

lwd rd1, rd2, (rs1), imm2

操作：

address←rs1+zero_extend(imm2 << 3)

rd1 ←sign_extend(mem[address+3: address])

rd2 ←sign_extend(mem[address+7: address+4])

执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常、非法指令异常

说明：

rd1, rd2, rs1 互相不可相等

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
11100	imm2	rd2	rs1	100	rd1	0001011

19.5.35　LWIA—符号位扩展字加载基地址自增指令

语法：

lwia rd, (rs1), imm5,imm2

操作：

rd ←sign_extend(mem[rs1+3:rs1])

rs1←rs1 + sign_extend(imm5 << imm2)

执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常、非法指令异常。

说明：

rd 和 rs1 不可相等

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
01011	imm2	imm5	rs1	100	rd	0001011

19.5.36　LWIB—基地址自增符号位扩展字加载指令

语法：

lwib rd, (rs1), imm5,imm2

操作：

rs1←rs1 + sign_extend(imm5 << imm2)

rd ←sign_extend(mem[rs1+3:rs1])

执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常、非法指令异常。

说明：

rd 和 rs1 不可相等

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
01001	imm2	imm5	rs1	100	rd	0001011

19.5.37　LWUD——零扩展双寄存器字加载指令

语法：

lwud rd1,rd2, (rs1),imm2

操作：

address←rs1+zero_extend(imm2 << 3)

rd1 ←zero_extend(mem[address+3: address])

rd2 ←zero_extend(mem[address+7: address+4])

执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常、非法指令异常

说明：

rd1,rd2 ,rs1 互相不可相等

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
11110	imm2	rd2	rs1	100	rd1	0001011

19.5.38　LWUIA——零扩展字加载基地址自增指令

语法：

lwuia rd, (rs1), imm5,imm2

操作：

rd ←zero_extend(mem[rs1+3:rs1])

rs1←rs1 + sign_extend(imm5 << imm2)

执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常、非法指令异常。

说明：

rd 和 rs1 不可相等

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
11011	imm2	imm5	rs1	100	rd	0001011

19.5.39　LWUIB—基地址自增零扩展字加载指令

语法：

lwuib rd, (rs1), imm5,imm2

操作：

rs1←rs1 + sign_extend(imm5 << imm2)

rd ←zero_extend(mem[rs1+3:rs1])

执行权限：

M mode/S mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常、非法指令异常。

说明：

rd 和 rs1 不可相等

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
11001	imm2	imm5	rs1	100	rd	0001011

19.5.40　SBIA—字节存储基地址自增指令

语法：

sbia rs2, (rs1), imm5,imm2

操作：

mem[rs1]←rs2[7:0]

rs1←rs1 + sign_extend(imm5 << imm2)

执行权限：

M mode/S mode/U mode

异常：

存储指令非对齐访问异常、存储指令访问错误异常、存储指令页面错误异常、非法指令异常

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00011	imm2	imm5	rs1	101	rs2	0001011

19.5.41　SBIB—基地址自增字节存储指令

语法：

sbib rs2, (rs1), imm5,imm2

操作：

rs1←rs1 + sign_extend(imm5 << imm2)

mem[rs1] ←rs2[7:0]

执行权限：

M mode/S mode/U mode

异常：

存储指令非对齐访问异常、存储指令访问错误异常、存储指令页面错误异常、非法指令异常

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00001	imm2	imm5	rs1	101	rs2	0001011

19.5.42　SDD—双寄存器存储指令

语法：

sdd rd1,rd2, (rs1),imm2, 4

操作：

address←rs1 + zero_extend(imm2 << 4)

mem[address+7:address] ←rd1

mem[address+15:address+8]←rd2

执行权限：

M mode/S mode/U mode

异常：

存储指令非对齐访问异常、存储指令访问错误异常、存储指令页面错误异常、非法指令异常

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
11111	imm2	rd2	rs1	101	rd1	0001011

19.5.43　SDIA—双字存储基地址自增指令

语法：

sdia rs2, (rs1), imm5,imm2

操作：

mem[rs1+7:rs1]←rs2[63:0]

rs1←rs1 + sign_extend(imm5 << imm2)

执行权限：

M mode/S mode/U mode

异常：

存储指令非对齐访问异常、存储指令访问错误异常、存储指令页面错误异常、非法指令异常

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
01111	imm2	imm5	rs1	101	rs2	0001011

19.5.44　SDIB——基地址自增双字存储指令

语法：

sdib rs2, (rs1), imm5,imm2

操作：

rs1←rs1 + sign_extend(imm5 << imm2)

mem[rs1+7:rs1] ←rs2[63:0]

执行权限：

M mode/S mode/U mode

异常：

存储指令非对齐访问异常、存储指令访问错误异常、存储指令页面错误异常、非法指令异常

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
01101	imm2	imm5	rs1	101	rs2	0001011

19.5.45　SHIA——半字存储基地址自增指令

语法：

shia rs2, (rs1), imm5,imm2

操作：

mem[rs1+1:rs1]←rs2[15:0]

rs1←rs1 + sign_extend(imm5 << imm2)

执行权限：

M mode/S mode/U mode

异常：

存储指令非对齐访问异常、存储指令访问错误异常、存储指令页面错误异常、非法指令异常

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00111	imm2	imm5	rs1	101	rs2	0001011

19.5.46　SHIB——基地址自增半字存储指令

语法：

shib rs2, (rs1), imm5,imm2

操作：

rs1←rs1 + sign_extend(imm5 << imm2)

mem[rs1+1:rs1] ←rs2[15:0]

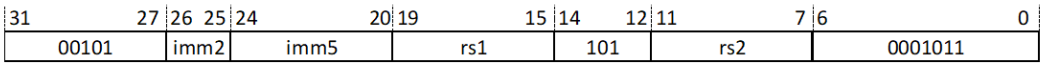
执行权限：

M mode/S mode/U mode

异常：

存储指令非对齐访问异常、存储指令访问错误异常、存储指令页面错误异常、非法指令异常

指令格式：



19.5.47　SRB——寄存器移位字节存储指令

语法：

srb rd, rs1, rs2, imm2

操作：

mem[(rs1+rs2 << imm2)] ←rd[7:0]

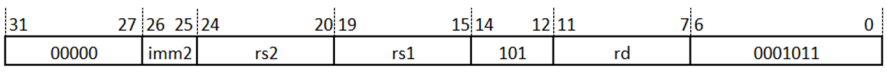
执行权限：

M mode/S mode/U mode

异常：

存储指令非对齐访问异常、存储指令访问错误异常、存储指令页面错误异常、非法指令异常

指令格式：



19.5.48　SRD——寄存器移位双字存储指令

语法：

srd rd, rs1, rs2, imm2

操作：

mem[(rs1+rs2 << imm2)+7: (rs1+rs2 << imm2)] ←rd[63:0]

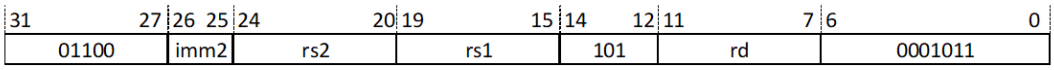
执行权限：

M mode/S mode/U mode

异常：

存储指令非对齐访问异常、存储指令访问错误异常、存储指令页面错误异常、非法指令异常

指令格式：



19.5.49　SRH——寄存器移位半字存储指令

语法：

srh rd, rs1, rs2, imm2

操作：

mem[(rs1+rs2 << imm2)+1: (rs1+rs2 << imm2)] ←rd[15:0]

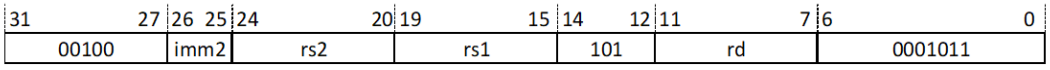
执行权限：

M mode/S mode/U mode

异常：

存储指令非对齐访问异常、存储指令访问错误异常、存储指令页面错误异常、非法指令异常

指令格式：



19.5.50　SRW——寄存器移位字存储指令

语法：

srw rd, rs1, rs2, imm2

操作：

mem[(rs1+rs2 << imm2)+3: (rs1+rs2 << imm2)] ←rd[31:0]

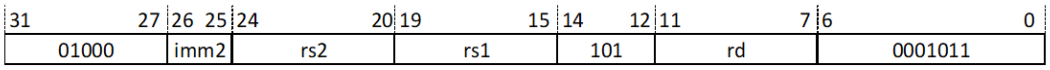
执行权限：

M mode/S mode/U mode

异常：

存储指令非对齐访问异常、存储指令访问错误异常、存储指令页面错误异常、非法指令异常

指令格式：



19.5.51　SURB——寄存器低 32 位移位字节存储指令

语法：

surb rd, rs1, rs2, imm2

操作：

mem[(rs1+rs2[31:0] << imm2)] ←rd[7:0]

执行权限：

M mode/S mode/U mode

异常：

存储指令非对齐访问异常、存储指令访问错误异常、存储指令页面错误异常、非法指令异常

说明：

rs2[31:0] 是无符号数, 高位 [63:32] 补 0 进行地址运算

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00010	imm2	rs2	rs1	101	rd	0001011

19.5.52　SURD——寄存器低 32 位移位双字存储指令

语法：

surd rd, rs1, rs2, imm2

操作：

mem[(rs1+rs2[31:0] << imm2)+7: (rs1+rs2[31:0] << imm2)] ←rd[63:0]

执行权限：

M mode/S mode/U mode

异常：

存储指令非对齐访问异常、存储指令访问错误异常、存储指令页面错误异常、非法指令异常

说明：

rs2[31:0] 是无符号数, 高位 [63:32] 补 0 进行地址运算

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
01110	imm2	rs2	rs1	101	rd	0001011

19.5.53　SURH——寄存器低 32 位移位半字存储指令

语法：

surh rd, rs1, rs2, imm2

操作：

mem[(rs1+rs2[31:0] << imm2)+1: (rs1+rs2[31:0] << imm2)] ←rd[15:0]

执行权限：

M mode/S mode/U mode

异常：

存储指令非对齐访问异常、存储指令访问错误异常、存储指令页面错误异常、非法指令异常

说明：

rs2[31:0] 是无符号数, 高位 [63:32] 补 0 进行地址运算

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00110	imm2	rs2	rs1	101	rd	0001011

19.5.54　SURW——寄存器低 32 位移位字存储指令

语法：

surw rd, rs1, rs2, imm2

操作：

mem[(rs1+rs2[31:0] << imm2)+3: (rs1+rs2[31:0] << imm2)] ←rd[31:0]

执行权限：

M mode/S mode/U mode

异常：

存储指令非对齐访问异常、存储指令访问错误异常、存储指令页面错误异常、非法指令异常

说明：

rs2[31:0] 是无符号数, 高位 [63:32] 补 0 进行地址运算

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
01010	imm2	rs2	rs1	101	rd	0001011

19.5.55　SWIA——字存储基地址自增指令

语法：

swia rs2, (rs1), imm5,imm2

操作：

mem[rs1+3:rs1]←rs2[31:0]

rs1←rs1 + sign_extend(imm5 << imm2)

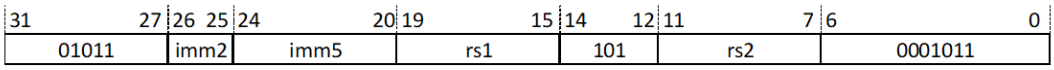
执行权限：

M mode/S mode/U mode

异常：

存储指令非对齐访问异常、存储指令访问错误异常、存储指令页面错误异常、非法指令异常

指令格式：



19.5.56　SWIB——基地址自增字存储指令

语法：

swib rs2, (rs1), imm5,imm2

操作：

rs1←rs1 + sign_extend(imm5 << imm2)

mem[rs1+3:rs1] ←rs2[31:0]

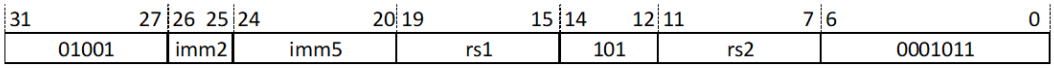
执行权限：

M mode/S mode/U mode

异常：

存储指令非对齐访问异常、存储指令访问错误异常、存储指令页面错误异常、非法指令异常

指令格式：



19.5.57　SWD——双寄存器低 32 位存储指令

语法：

swd rd1,rd2,(rs1),imm2

操作：

address←rs1+ zero_extend(imm2 << 3)

mem[address+3:address] ←rd1[31:0]

mem[address+7:address+4]←rd2[31:0]

执行权限：

M mode/S mode/U mode

异常：

存储指令非对齐访问异常、存储指令访问错误异常、存储指令页面错误异常、非法指令异常

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
11100	imm2	rd2	rs1	101	rd1	0001011

19.6　附录 B-6 浮点半精度指令术语

浮点半精度指令子集实现了对浮点半精度的支持，每条指令位宽为 32 位，以下指令按英文字母顺序排列。

19.6.1　FADD.H——半精度浮点加法指令

语法：

fadd.h fd, fs1, fs2, rm

操作：

fd ← fs1 + fs2

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

浮点状态位 NV/OF/NX

说明：

rm 决定舍入模式:

- 3'b000: 就近向偶数舍入，对应的汇编指令 fadd.h fd, fs1,fs2,rne。
- 3'b001: 向零舍入，对应的汇编指令 fadd.h fd, fs1,fs2,rtz。
- 3'b010: 向负无穷舍入，对应的汇编指令 fadd.h fd, fs1,fs2,rdn。
- 3'b011: 向正无穷舍入，对应的汇编指令 fadd.h fd, fs1,fs2,rup。
- 3'b100: 就近向大值舍入，对应的汇编指令 fadd.h fd, fs1,fs2,rmm。
- 3'b101: 暂未使用，不会出现该编码。
- 3'b110: 暂未使用，不会出现该编码。
- 3'b111: 动态舍入，根据浮点控制寄存器 fcsr 中的 rm 位来决定舍入模式，对应的汇编指令 fadd.h fd, fs1,fs2。

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
0000010	fs2	fs1	rm	fd	1010011

19.6.2　FCLASS.H——半精度浮点分类指令

语法：

fclass.h rd, fs1

操作：

```
if ( fs1 = -inf)
        rd ← 64'h1
if ( fs1 = -norm)
        rd ← 64'h2
if ( fs1 = -subnorm)
        rd ← 64'h4
if ( fs1 = -zero)
        rd ← 64'h8
if ( fs1 = +zero)
        rd ← 64'h10
if ( fs1 = +subnorm)
        rd ← 64'h20
if ( fs1 = +norm)
        rd ← 64'h40
if ( fs1 = +inf)
        rd ← 64'h80
if ( fs1 = sNaN)
        rd ← 64'h100
if ( fs1 = qNaN)
        rd ← 64'h200
```

执行权限：

M mode/S mode/U mode

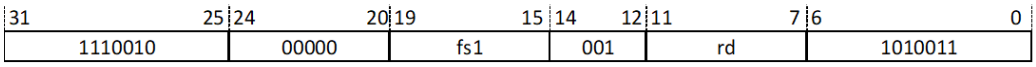
异常：

非法指令异常

影响标志位：

无

指令格式：



31 25	24 20	19 15	14 12	11 7	6 0
1110010	00000	fs1	001	rd	1010011

19.6.3　FCVT.H.L—有符号长整型转换成半精度浮点数指令

语法：

fcvt.h.l fd, rs1, rm

操作：

fd ← signed_long_convert_to_half(rs1)

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

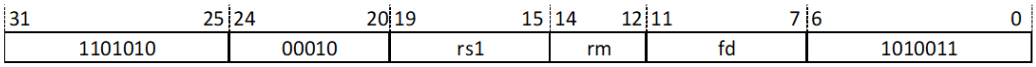
浮点状态位 NX/OF

说明：

rm 决定舍入模式:

- 3'b000: 就近向偶数舍入，对应的汇编指令 fcvt.h.l fd,rs1,rne。
- 3'b001: 向零舍入，对应的汇编指令 fcvt.h.l fd,rs1,rtz。
- 3'b010: 向负无穷舍入，对应的汇编指令 fcvt.h.l fd,rs1,fdn。
- 3'b011: 向正无穷舍入，对应的汇编指令 fcvt.h.l fd,rs1,rup。
- 3'b100: 就近向大值舍入，对应的汇编指令 fcvt.h.l fd,rs1,rmm。
- 3'b101: 暂未使用，不会出现该编码。
- 3'b110: 暂未使用，不会出现该编码。
- 3'b111: 动态舍入，根据浮点控制寄存器 fcsr 中的 rm 位来决定舍入模式，对应的汇编指令 fcvt.h.l fd, rs1。

指令格式：



31 25	24 20	19 15	14 12	11 7	6 0
1101010	00010	rs1	rm	fd	1010011

19.6.4　FCVT.H.LU—无符号长整型转换成半精度浮点数指令

语法：

fcvt.h.lu fd, rs1, rm

操作：

fd ← unsigned_long_convert_to_half_fp(rs1)

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

浮点状态位 NX/OF

说明：

rm 决定舍入模式:

- 3'b000: 就近向偶数舍入，对应的汇编指令 fcvt.h.lu fd,rs1,rne。
- 3'b001: 向零舍入，对应的汇编指令 fcvt.h.lu fd, rs1,rtz。
- 3'b010: 向负无穷舍入，对应的汇编指令 fcvt.h.lu fd, rs1,fdn。
- 3'b011: 向正无穷舍入，对应的汇编指令 fcvt.h.lu fd, rs1,rup。
- 3'b100: 就近向大值舍入，对应的汇编指令 fcvt.h.lu fd, rs1,rmm。
- 3'b101: 暂未使用，不会出现该编码。
- 3'b110: 暂未使用，不会出现该编码。
- 3'b111: 动态舍入，根据浮点控制寄存器 fcsr 中的 rm 位来决定舍入模式，对应的汇编指令 fcvt.h.lu fd, rs1。

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
1101010	00011	rs1	rm	fd	1010011

19.6.5　FCVT.H.S——单精度浮点转换成半精度浮点指令

语法：

fcvt.h.s fd, fs1, rm

操作：

fd ← single_convert_to_half(fs1)

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

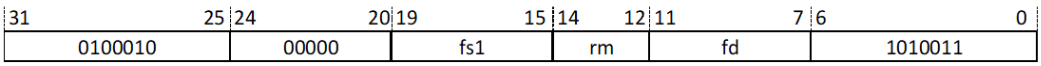
浮点状态位 NV/OF/UF/NX

说明：

rm 决定舍入模式:

- 3'b000: 就近向偶数舍入，对应的汇编指令 fcvt.h.s fd,fs1,rne。
- 3'b001: 向零舍入，对应的汇编指令 fcvt.h.s fd,fs1,rtz。
- 3'b010: 向负无穷舍入，对应的汇编指令 fcvt.h.s fd,fs1,fdn。
- 3'b011: 向正无穷舍入，对应的汇编指令 fcvt.h.s fd,fs1,rup。
- 3'b100: 就近向大值舍入，对应的汇编指令 fcvt.h.s fd,fs1,rmm。
- 3'b101: 暂未使用，不会出现该编码。
- 3'b110: 暂未使用，不会出现该编码。
- 3'b111: 动态舍入，根据浮点控制寄存器 fcsr 中的 rm 位来决定舍入模式，对应的汇编指令 fcvt.h.s fd, fs1。

指令格式：



19.6.6　FCVT.H.W—有符号整型转换成半精度浮点数指令

语法：

fcvt.h.w fd, rs1, rm

操作：

fd ← signed_int_convert_to_half(rs1)

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

浮点状态位 NX/OF

说明：

rm 决定舍入模式:

- 3'b000: 就近向偶数舍入，对应的汇编指令 fcvt.h.w fd,rs1,rne。
- 3'b001: 向零舍入，对应的汇编指令 fcvt.h.w fd,rs1,rtz。
- 3'b010: 向负无穷舍入，对应的汇编指令 fcvt.h.w fd,rs1,fdn。
- 3'b011: 向正无穷舍入，对应的汇编指令 fcvt.h.w fd,rs1,rup。
- 3'b100: 就近向大值舍入，对应的汇编指令 fcvt.h.w fd,rs1,rmm。

- 3'b101: 暂未使用，不会出现该编码。
- 3'b110: 暂未使用，不会出现该编码。
- 3'b111: 动态舍入，根据浮点控制寄存器 fcsr 中的 rm 位来决定舍入模式，对应的汇编指令 fcvt.h.w fd, rs1。

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
1101010	00000	rs1	rm	fd	1010011

19.6.7　FCVT.H.WU—无符号整型转换成半精度浮点数指令

语法：

fcvt.h.wu fd, rs1, rm

操作：

fd ← unsigned_int_convert_to_half_fp(rs1)

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

浮点状态位 NX/OF

说明：

rm 决定舍入模式:

- 3'b000: 就近向偶数舍入，对应的汇编指令 fcvt.h.wu fd,rs1,rne。
- 3'b001: 向零舍入，对应的汇编指令 fcvt.h.wu fd,rs1,rtz。
- 3'b010: 向负无穷舍入，对应的汇编指令 fcvt.h.wu fd,rs1,fdn。
- 3'b011: 向正无穷舍入，对应的汇编指令 fcvt.h.wu fd,rs1,rup。
- 3'b100: 就近向大值舍入，对应的汇编指令 fcvt.h.wu fd,rs1,rmm。
- 3'b101: 暂未使用，不会出现该编码。
- 3'b110: 暂未使用，不会出现该编码。
- 3'b111: 动态舍入，根据浮点控制寄存器 fcsr 中的 rm 位来决定舍入模式，对应的汇编指令 fcvt.h.wu fd, rs1。

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
1101010	00001	rs1	rm	fd	1010011

19.6.8　FCVT.L.H—半精度浮点转换成有符号长整型指令

语法：

fcvt.l.h rd, fs1, rm

操作：

rd ← half_convert_to_signed_long(fs1)

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

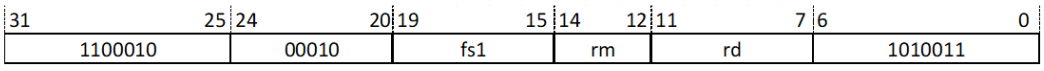
浮点状态位 NV/NX

说明：

rm 决定舍入模式:

- 3'b000: 就近向偶数舍入，对应的汇编指令 fcvt.l.h rd,fs1,rne。
- 3'b001: 向零舍入，对应的汇编指令 fcvt.l.h rd,fs1,rtz。
- 3'b010: 向负无穷舍入，对应的汇编指令 fcvt.l.h rd,fs1,rdn。
- 3'b011: 向正无穷舍入，对应的汇编指令 fcvt.l.h rd,fs1,rup。
- 3'b100: 就近向大值舍入，对应的汇编指令 fcvt.l.h rd,fs1,rmm。
- 3'b101: 暂未使用，不会出现该编码。
- 3'b110: 暂未使用，不会出现该编码。
- 3'b111: 动态舍入，根据浮点控制寄存器 fcsr 中的 rm 位来决定舍入模式，对应的汇编指令 fcvt.l.h rd, fs1。

指令格式：



19.6.9　FCVT.LU.H—半精度浮点转换成无符号长整型指令

语法：

fcvt.lu.h rd, fs1, rm

操作：

rd ← half_convert_to_unsigned_long(fs1)

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

浮点状态位 NV/NX

说明：

rm 决定舍入模式:

- 3'b000: 就近向偶数舍入，对应的汇编指令 fcvt.lu.h rd,fs1,rne。
- 3'b001: 向零舍入，对应的汇编指令 fcvt.lu.h rd,fs1,rtz。
- 3'b010: 向负无穷舍入，对应的汇编指令 fcvt.lu.h rd,fs1,rdn。
- 3'b011: 向正无穷舍入，对应的汇编指令 fcvt.lu.h rd,fs1,rup。
- 3'b100: 就近向大值舍入，对应的汇编指令 fcvt.lu.h rd,fs1,rmm。
- 3'b101: 暂未使用，不会出现该编码。
- 3'b110: 暂未使用，不会出现该编码。
- 3'b111: 动态舍入，根据浮点控制寄存器 fcsr 中的 rm 位来决定舍入模式，对应的汇编指令 fcvt.lu.h rd, fs1。

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
1100010	00011	fs1	rm	rd	1010011

19.6.10　FCVT.S.H—半精度浮点转换成单精度浮点指令

语法：

fcvt.s.h fd, fs1

操作：

fd ← half_convert_to_single(fs1)

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

无

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
0100000	00010	fs1	000	fd	1010011

19.6.11　FCVT.W.H—半精度浮点转换成有符号整型指令

语法：

fcvt.w.h rd, fs1, rm

操作：

tmp ← half_convert_to_signed_int(fs1)

rd←sign_extend(tmp)

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

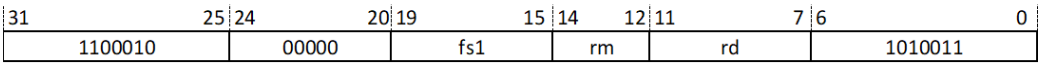
浮点状态位 NV/NX

说明：

rm 决定舍入模式:

- 3'b000: 就近向偶数舍入，对应的汇编指令 fcvt.w.h rd,fs1,rne。
- 3'b001: 向零舍入，对应的汇编指令 fcvt.w.h rd,fs1,rtz。
- 3'b010: 向负无穷舍入，对应的汇编指令 fcvt.w.h rd,fs1,rdn。
- 3'b011: 向正无穷舍入，对应的汇编指令 fcvt.w.h rd,fs1,rup。
- 3'b100: 就近向大值舍入，对应的汇编指令 fcvt.w.h rd,fs1,rmm。
- 3'b101: 暂未使用，不会出现该编码。
- 3'b110: 暂未使用，不会出现该编码。
- 3'b111: 动态舍入，根据浮点控制寄存器 fcsr 中的 rm 位来决定舍入模式，对应的汇编指令 fcvt.w.h rd, fs1。

指令格式：



31 25	24 20	19 15	14 12	11 7	6 0
1100010	00000	fs1	rm	rd	1010011

19.6.12　FCVT.WU.H—半精度浮点转换成无符号整型指令

语法：

fcvt.wu.h rd, fs1, rm

操作：

tmp ← half_convert_to_unsigned_int(fs1)

rd←sign_extend(tmp)

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

浮点状态位 NV/NX

说明：

rm 决定舍入模式:

- 3'b000: 就近向偶数舍入，对应的汇编指令 fcvt.wu.h rd,fs1,rne。
- 3'b001: 向零舍入，对应的汇编指令 fcvt.wu.h rd,fs1,rtz。
- 3'b010: 向负无穷舍入，对应的汇编指令 fcvt.wu.h rd,fs1,rdn。
- 3'b011: 向正无穷舍入，对应的汇编指令 fcvt.wu.h rd,fs1,rup。
- 3'b100: 就近向大值舍入，对应的汇编指令 fcvt.wu.h rd,fs1,rmm。
- 3'b101: 暂未使用，不会出现该编码。
- 3'b110: 暂未使用，不会出现该编码。
- 3'b111: 动态舍入，根据浮点控制寄存器 fcsr 中的 rm 位来决定舍入模式，对应的汇编指令 fcvt.wu.h rd, fs1。

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
1100010	00001	fs1	rm	rd	1010011

19.6.13　FDIV.H—半精度浮点除法指令

语法：

fdiv.h fd, fs1, fs2, rm

操作：

fd ← fs1 / fs2

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

浮点状态位 NV/DZ/OF/UF/NX

说明：

rm 决定舍入模式:

- 3'b000: 就近向偶数舍入，对应的汇编指令 fdiv.h fs1,fs2,rne。
- 3'b001: 向零舍入，对应的汇编指令 fdiv.h fd fs1,fs2,rtz。
- 3'b010: 向负无穷舍入，对应的汇编指令 fdiv.h fd, fs1,fs2,rdn。
- 3'b011: 向正无穷舍入，对应的汇编指令 fdiv.h fd, fs1,fs2,rup。
- 3'b100: 就近向大值舍入，对应的汇编指令 fdiv.h fd, fs1,fs2,rmm。
- 3'b101: 暂未使用，不会出现该编码。
- 3'b110: 暂未使用，不会出现该编码。
- 3'b111: 动态舍入，根据浮点控制寄存器 fcsr 中的 rm 位来决定舍入模式，对应的汇编指令 fdiv.h fd, fs1,fs2。

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
0001110	fs2	fs1	rm	fd	1010011

19.6.14　FEQ.H——半精度浮点比较相等指令

语法：

feq.h rd, fs1, fs2

操作：

```
if(fs1 == fs2)
        rd ← 1
else
        rd ← 0
```

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

浮点状态位 NV

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
1010010	fs2	fs1	010	rd	1010011

19.6.15　FLE.H——半精度浮点比较小于等于指令

语法：

fle.h rd, fs1, fs2

操作：

if(fs1 <= fs2)

rd ← 1

else

rd ← 0

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

浮点状态位 NV

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
1010010	fs2	fs1	000	rd	1010011

19.6.16　FLH——半精度浮点加载指令

语法：

flh fd, imm12(rs1)

操作：

address←rs1+sign_extend(imm12)

fd[15:0] ← mem[(address+1):address]

fd[63:16] ← 48'hffffffffffff

执行权限：

M mode/S mode/U mode

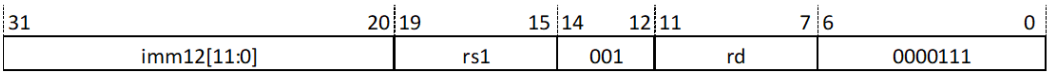
异常：

加载指令非对齐访问异常、加载指令访问错误异常、加载指令页面错误异常、非法指令异常

影响标志位：

无

指令格式：



19.6.17　FLT.H——半精度浮点比较小于指令

语法：

flt.h rd, fs1, fs2

操作：

if(fs1 < fs2)

　　rd ← 1

else

　　rd ← 0

执行权限：

M mode/S mode/U mode

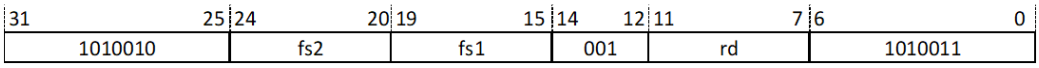
异常：

非法指令异常

影响标志位：

浮点状态位 NV

指令格式：



19.6.18　FMADD.H——半精度浮点乘累加指令

语法：

fmadd.h fd, fs1, fs2, fs3, rm

操作：

fd ← fs1*fs2 + fs3

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

浮点状态位 NV/OF/UF/IX

说明：

rm 决定舍入模式:

- 3'b000: 就近向偶数舍入，对应的汇编指令 fmadd.h fd,fs1, fs2, fs3, rne。
- 3'b001: 向零舍入，对应的汇编指令 fmadd.h fd,fs1, fs2, fs3, rtz。
- 3'b010: 向负无穷舍入，对应的汇编指令 fmadd.h fd,fs1, fs2, fs3, rdn。
- 3'b011: 向正无穷舍入，对应的汇编指令 fmadd.h fd,fs1, fs2, fs3, rup。
- 3'b100: 就近向大值舍入，对应的汇编指令 fmadd.h fd,fs1, fs2, fs3, rmm。
- 3'b101: 暂未使用，不会出现该编码。
- 3'b110: 暂未使用，不会出现该编码。
- 3'b111: 动态舍入，根据浮点控制寄存器 fcsr 中的 rm 位来决定舍入模式，对应的汇编指令 fmadd.h fd,fs1, fs2, fs3。

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
fs3	10	fs2	fs1	rm	fd	1000011

19.6.19　FMAX.H——半精度浮点取最大值指令

语法：

fmax.h fd, fs1, fs2

操作：

```
if(fs1 >= fs2)
        fd ← fs1
else
        fd ← fs2
```

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

浮点状态位 NV

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
0010110	fs2	fs1	001	fd	1010011

19.6.20　FMIN.H——半精度浮点取最小值指令

语法：

fmin.h fd, fs1, fs2

操作：

if(fs1 >= fs2)

　　fd ← fs2

else

　　fd ← fs1

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

浮点状态位 NV

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
0010110	fs2	fs1	000	fd	1010011

19.6.21　FMSUB.H——半精度浮点乘累减指令

语法：

fmsub.h fd, fs1, fs2, fs3, rm

操作：

fd ← fs1*fs2 - fs3

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

浮点状态位 NV/OF/UF/IX

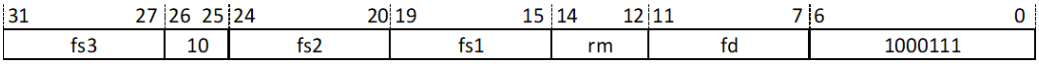
说明：

rm 决定舍入模式:

- 3'b000: 就近向偶数舍入，对应的汇编指令 fmsub.h fd,fs1, fs2, fs3, rne。

- 3'b001: 向零舍入，对应的汇编指令 fmsub.h fd,fs1, fs2, fs3, rtz。
- 3'b010: 向负无穷舍入，对应的汇编指令 fmsub.h fd,fs1, fs2, fs3, rdn。
- 3'b011: 向正无穷舍入，对应的汇编指令 fmsub.h fd,fs1, fs2, fs3, rup。
- 3'b100: 就近向大值舍入，对应的汇编指令 fmsub.h fd, fs1, fs2, fs3,rmm。
- 3'b101: 暂未使用，不会出现该编码。
- 3'b110: 暂未使用，不会出现该编码。
- 3'b111: 动态舍入，根据浮点控制寄存器 fcsr 中的 rm 位来决定舍入模式，对应的汇编指令 fmsub.h fd,fs1, fs2, fs3。

指令格式：



19.6.22　FMUL.H—半精度浮点乘法指令

语法：

fmul.h fd, fs1, fs2, rm

操作：

fd ← fs1 * fs2

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

浮点状态位 NV/OF/UF/NX

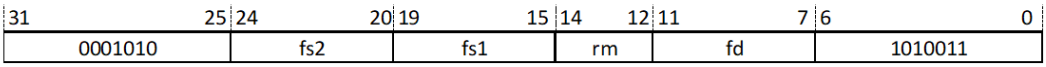
说明：

rm 决定舍入模式:

- 3'b000: 就近向偶数舍入，对应的汇编指令 fmul.h fd, fs1, fs2, rne。
- 3'b001: 向零舍入，对应的汇编指令 fmul.h fd, fs1, fs2, rtz。
- 3'b010: 向负无穷舍入，对应的汇编指令 fmul.h fd, fs1, fs2, rdn。
- 3'b011: 向正无穷舍入，对应的汇编指令 fmul.h fd, fs1, fs2, rup。
- 3'b100: 就近向大值舍入，对应的汇编指令 fmul.h fd, fs1,fs2 , rmm。
- 3'b101: 暂未使用，不会出现该编码。
- 3'b110: 暂未使用，不会出现该编码。

- 3'b111：动态舍入，根据浮点控制寄存器 fcsr 中的 rm 位来决定舍入模式，对应的汇编指令 fmul.h fs1,fs2。

指令格式：



19.6.23　FMV.H.X——半精度浮点写传送指令

语法：

fmv.h.x fd, rs1

操作：

fd[15:0] ← rs1[15:0]

fd[63:16] ← 48'hffffffffffff

执行权限：

M mode/S mode/U mode

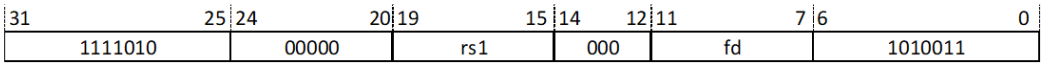
异常：

非法指令异常

影响标志位：

无

指令格式：



19.6.24　FMV.X.H——半精度浮点寄存器读传送指令

语法：

fmv.x.h rd, fs1

操作：

tmp[15:0] ← fs1[15:0]

rd ← sign_extend(tmp[15:0])

执行权限：

M mode/S mode/U mode

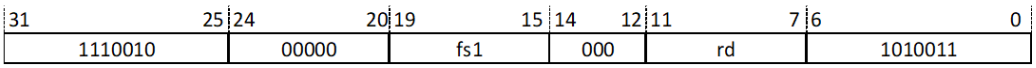
异常：

非法指令异常

影响标志位：

无

指令格式：



19.6.25　FNMADD.H——半精度浮点乘累加取负指令

语法：

fnmadd.h fd, fs1, fs2, fs3, rm

操作：

fd ←-(fs1*fs2 + fs3)

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

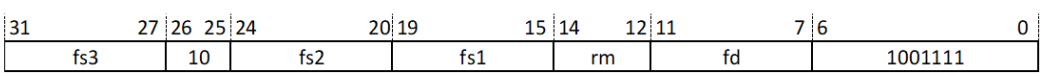
浮点状态位 NV/OF/UF/IX

说明：

rm 决定舍入模式:

- 3'b000: 就近向偶数舍入，对应的汇编指令 fnmadd.h fd,fs1, fs2, fs3, rne。
- 3'b001: 向零舍入，对应的汇编指令 fnmadd.h fd,fs1, fs2, fs3, rtz。
- 3'b010: 向负无穷舍入，对应的汇编指令 fnmadd.h fd,fs1, fs2, fs3, rdn。
- 3'b011: 向正无穷舍入，对应的汇编指令 fnmadd.h fd,fs1, fs2, fs3, rup。
- 3'b100: 就近向大值舍入，对应的汇编指令 fnmadd.h fd,fs1, fs2, fs3, rmm。
- 3'b101: 暂未使用，不会出现该编码。
- 3'b110: 暂未使用，不会出现该编码。
- 3'b111: 动态舍入，根据浮点控制寄存器 fcsr 中的 rm 位来决定舍入模式，对应的汇编指令 fnmadd.h fd,fs1, fs2, fs3。

指令格式：



19.6.26　FNMSUB.H——半精度浮点乘累减取负指令

语法：

fnmsub.h fd, fs1, fs2, fs3, rm

操作：

fd ← -(fs1*fs2 - fs3)

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

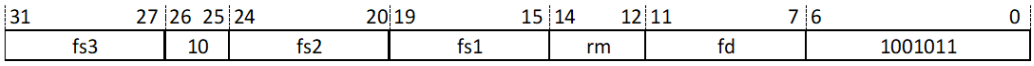
浮点状态位 NV/OF/UF/IX

说明：

rm 决定舍入模式:

- 3'b000: 就近向偶数舍入，对应的汇编指令 fnmsub.h fd,fs1, fs2, fs3, rne。
- 3'b001: 向零舍入，对应的汇编指令 fnmsub.h fd,fs1, fs2, fs3, rtz。
- 3'b010: 向负无穷舍入，对应的汇编指令 fnmsub.h fd,fs1, fs2, fs3, rdn。
- 3'b011: 向正无穷舍入，对应的汇编指令 fnmsub.h fd,fs1, fs2, fs3, rup。
- 3'b100: 就近向大值舍入，对应的汇编指令 fnmsub.h fd,fs1, fs2, fs3, rmm。
- 3'b101: 暂未使用，不会出现该编码。
- 3'b110: 暂未使用，不会出现该编码。
- 3'b111: 动态舍入，根据浮点控制寄存器 fcsr 中的 rm 位来决定舍入模式，对应的汇编指令 fnmsub.h fd,fs1, fs2, fs3。

指令格式：



19.6.27　FSGNJ.H—半精度浮点符号注入指令

语法：

fsgnj.h fd, fs1, fs2

操作：

fd[14:0] ← fs1[14:0]

fd[15] ← fs2[15]

fd[63:16] ← 48'hffffffffffff

执行权限：

M mode/S mode/U mode

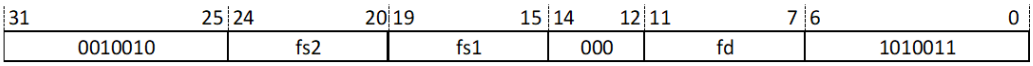
异常：

非法指令异常

影响标志位：

无

指令格式：



31 25	24 20	19 15	14 12	11 7	6 0
0010010	fs2	fs1	000	fd	1010011

19.6.28　FSGNJN.H——半精度浮点符号取反注入指令

语法：

fsgnjn.h fd, fs1, fs2

操作：

fd[14:0] ← fs1[14:0]

fd[15] ← ! fs2[15]

fd[63:16] ← 48'hffffffffffff

执行权限：

M mode/S mode/U mode

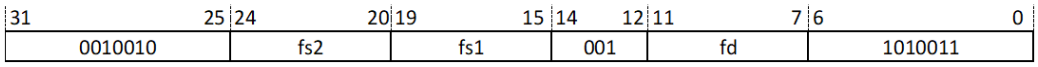
异常：

非法指令异常

影响标志位：

无

指令格式：



31 25	24 20	19 15	14 12	11 7	6 0
0010010	fs2	fs1	001	fd	1010011

19.6.29　FSGNJX.H——半精度浮点符号异或注入指令

语法：

fsgnjx.h fd, fs1, fs2

操作：

fd[14:0] ← fs1[14:0]

fd[15] ← fs1[15] ^ fs2[15]

fd[63:16] ← 48'hffffffffffff

执行权限：

M mode/S mode/U mode

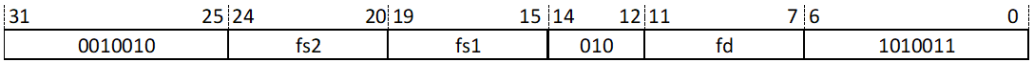
异常：

非法指令异常

影响标志位：

无

指令格式：



31 25	24 20	19 15	14 12	11 7	6 0
0010010	fs2	fs1	010	fd	1010011

19.6.30　FSH——半精度浮点存储指令

语法：

fsh fs2, imm12(fs1)

操作：

address←fs1+sign_extend(imm12)

mem[(address+1):address] ← fs2[15:0]

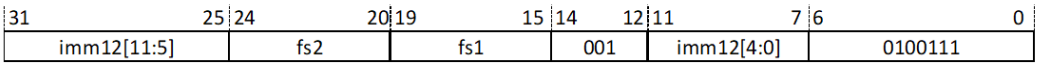
执行权限：

M mode/S mode/U mode

异常：

存储指令非对齐访问异常、存储指令访问错误异常、存储指令页面错误异常、非法指令异常

指令格式：



31 25	24 20	19 15	14 12	11 7	6 0
imm12[11:5]	fs2	fs1	001	imm12[4:0]	0100111

19.6.31　FSQRT.H——半精度浮点开方指令

语法：

fsqrt.h fd, fs1, rm

操作：

fd ← sqrt(fs1)

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

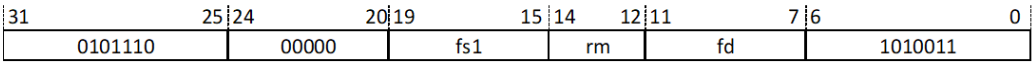
浮点状态位 NV/NX

说明：

rm 决定舍入模式:

- 3'b000: 就近向偶数舍入，对应的汇编指令 fsqrt.h fd, fs1,rne
- 3'b001: 向零舍入，对应的汇编指令 fsqrt.h fd, fs1,rtz
- 3'b010: 向负无穷舍入，对应的汇编指令 fsqrt.h fd, fs1,rdn
- 3'b011: 向正无穷舍入，对应的汇编指令 fsqrt.h fd, fs1,rup
- 3'b100: 就近向大值舍入，对应的汇编指令 fsqrt.h fd, fs1,rmm
- 3'b101: 暂未使用，不会出现该编码。
- 3'b110: 暂未使用，不会出现该编码。
- 3'b111: 动态舍入，根据浮点控制寄存器 fcsr 中的 rm 位来决定舍入模式，对应的汇编指令 fsqrt.h fd, fs1。

指令格式：



19.6.32　FSUB.H——半精度浮点减法指令

语法：

fsub.h fd, fs1, fs2, rm

操作：

fd ← fs1 - fs2

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

影响标志位：

浮点状态位 NV/OF/NX

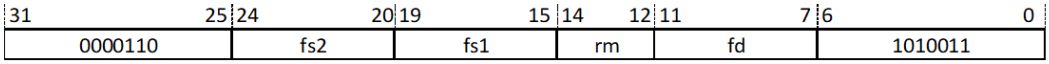
说明：

rm 决定舍入模式:

- 3'b000: 就近向偶数舍入，对应的汇编指令 fsub.h fd, fs1,fs2,rne
- 3'b001: 向零舍入，对应的汇编指令 fsub.h fd, fs1,fs2,rtz
- 3'b010: 向负无穷舍入，对应的汇编指令 fsub.h fd, fs1,fs2,rdn
- 3'b011: 向正无穷舍入，对应的汇编指令 fsub.h fd, fs1,fs2,rup

- 3'b100: 就近向大值舍入，对应的汇编指令 fsub.h fd, fs1,fs2,rmm
- 3'b101: 暂未使用，不会出现该编码。
- 3'b110: 暂未使用，不会出现该编码。
- 3'b111: 动态舍入，根据浮点控制寄存器 fcsr 中的 rm 位来决定舍入模式，对应的汇编指令 fsub.h fd, fs1,fs2。

指令格式：



19.7　附录 B-8 AI 扩展指令术语

AI 扩展指令子集实现了对 int8/int4 格式的数的运算操作，每条指令位宽为 32 位。以下指令按英文字母顺序排列

19.7.1　指令功能描述

19.7.1.1　VMAQA.VV——8-bit 矢量有符号乘累加链加指令

语法：

vmaqa.vv vd, vs1, vs2, vm #vector-vector

操作：

```
for(i=0; i<VLMAX; i++) {
  int32 temp = 0;
  for(j=0; j<4; j++){
        if unmasked or vm[i*4+j] == 1:
            temp += signed(vs2[i*4+j]) * signed(vs1[i*4+j]); #8-bit per element
  }
  if i<vl:
        vd[i] += temp
  else:
        vd[i] = vd[i]
}
```

执行权限：

M mode/ S mode/U mode

异常：

非法指令异常

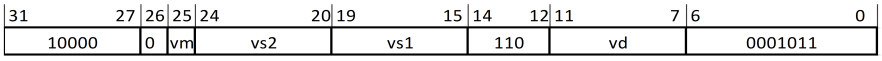
影响标志位：

无

说明：

- vd 数据宽度为 SEW=32-bit，vs2/vs1 数据宽度为 SEW/4=8-bit，其他 SEW 配置非法。
- LMUL 支持 1/2/4/8/0.5，其他配置非法。
- 当 vm=1 时，指令未被 mask，对应汇编指令 vmaqa.vv vd, vs1, vs2。
- 当 vm=0 时，指令被 mask，对应汇编指令 vmaqa.vv vd, vs1, vs2, v0.t。
- mask 对源操作数起作用，对应的操作数不参与乘法运算，一组 4 个源操作数均被 mask 时，累加操作数保持原值不变。

指令格式：



19.7.1.2　VMAQAU.VV—8-bit 矢量无符号乘累加链加指令

语法：

vmaqau.vv vd, vs1, vs2, vm #vector-vector

操作：

```
for(i=0; i<VLMAX; i++) {
  int32 temp = 0;
  for(j=0; j<4; j++){
        if unmasked or vm[i*4+j] == 1:
            temp += unsigned(vs2[i*4+j]) * unsigned(vs1[i*4+j]); #8-bit per element
  }
  if i<vl:
        vd[i] += temp
  else:
        vd[i] = vd[i]
}
```

执行权限：

M mode/ S mode/U mode

异常：

非法指令异常

影响标志位：

无

说明：

- vd 数据宽度为 SEW=32-bit，vs2/vs1 数据宽度为 SEW/4=8-bit，其他 SEW 配置非法。
- LMUL 支持 1/2/4/8/0.5，其他配置非法。
- 当 vm=1 时，指令未被 mask，对应汇编指令 vmaqau.vv vd, vs1, vs2。
- 当 vm=0 时，指令被 mask，对应汇编指令 vmaqau.vv vd, vs1, vs2, v0.t。
- mask 对源操作数起作用，对应的操作数不参与乘法运算，一组 4 个源操作数均被 mask 时，累加操作数保持原值不变。

指令格式：

31 27	26	25	24 20	19 15	14 12	11 7	6 0
10001	0	vm	vs2	vs1	110	vd	0001011

19.7.1.3　VMAQASU.VV——8-bit 矢量有符号无符号乘累加链加指令

语法：

vmaqasu.vv vd, vs1, vs2, vm #vector-vector

操作：

```
for(i=0; i<VLMAX; i++) {
  int32 temp = 0;
  for(j=0; j<4; j++){
        if unmasked or vm[i*4+j] == 1:
            temp += unsigned(vs2[i*4+j]) * signed(vs1[i*4+j]); #8-bit per element
  }
  if i<vl:
        vd[i] += temp
  else:
        vd[i] = vd[i]
}
```

执行权限：

M mode/ S mode/U mode

异常：

非法指令异常

影响标志位：

无

说明：

- vd 数据宽度为 SEW=32-bit，vs2/vs1 数据宽度为 SEW/4=8-bit，其他 SEW 配置非法。
- LMUL 支持 1/2/4/8/0.5，其他配置非法。
- 当 vm=1 时，指令未被 mask，对应汇编指令 vmaqasu.vv vd, vs1, vs2。
- 当 vm=0 时，指令被 mask，对应汇编指令 vmaqasu.vv vd, vs1, vs2, v0.t。
- mask 对源操作数起作用，对应的操作数不参与乘法运算，一组 4 个源操作数均被 mask 时，累加操作数保持原值不变。

指令格式：

31 27	26	25	24 20	19 15	14 12	11 7	6 0
10010	0	vm	vs2	vs1	110	vd	0001011

19.7.1.4　VMAQA.VX——8-bit 矢量标量有符号乘累加链加指令

语法：

vmaqa.vx vd, rs1, vs2, vm #vector-vector

操作：

```
for (i=0; i<4; i++) {
  operand1[i] = BITS(rs1, (i+1)*8-1, i*8);
}
for(i=0; i<VLMAX; i++) {
  int32 temp = 0;
  for(j=0; j<4; j++){
        if unmasked or vm[i*4+j] == 1:
            temp += signed(vs2[i*4+j]) * signed(operand1[j]); #8-bit per element
  }
  if i<vl:
        vd[i] += temp
  else:
        vd[i] = vd[i]
}
```

执行权限：

M mode/ S mode/U mode

异常：

非法指令异常

影响标志位：

无

说明：

- vd 数据宽度为 SEW=32-bit，vs2/vs1 数据宽度为 SEW/4=8-bit，其他 SEW 配置非法。
- LMUL 支持 1/2/4/8/0.5，其他配置非法。
- 当 vm=1 时，指令未被 mask，对应汇编指令 vmaqa.vx vd, rs1, vs2。
- 当 vm=0 时，指令被 mask，对应汇编指令 vmaqa.vx vd, rs1, vs2, v0.t。
- mask 对源操作数起作用，对应的操作数不参与乘法运算，一组 4 个源操作数均被 mask 时，累加操作数保持原值不变。

指令格式：

31 27	26	25	24 20	19 15	14 12	11 7	6 0
10000	1	vm	vs2	rs1	110	vd	0001011

19.7.1.5　VMAQAU.VX—8-bit 矢量标量无符号乘累加链加指令

语法：

vmaqau.vx vd, rs1, vs2, vm #vector-vector

操作：

```
for (i=0; i<4; i++) {
  operand1[i] = BITS(rs1, (i+1)*8-1, i*8);
}
for(i=0; i<VLMAX; i++) {
  int32 temp = 0;
  for(j=0; j<4; j++){
        if unmasked or vm[i*4+j] == 1:
            temp += unsigned(vs2[i*4+j]) * unsigned(operand1[j]); #8-bit per element
  }
  if i<vl:
        vd[i] += temp
  else:
        vd[i] = vd[i]
}
```

执行权限：

M mode/ S mode/U mode

异常：

非法指令异常

影响标志位：

无

说明：

- vd 数据宽度为 SEW=32-bit，vs2/vs1 数据宽度为 SEW/4=8-bit，其他 SEW 配置非法。
- LMUL 支持 1/2/4/8/0.5，其他配置非法。
- 当 vm=1 时，指令未被 mask，对应汇编指令 vmaqau.vx vd, rs1, vs2。
- 当 vm=0 时，指令被 mask，对应汇编指令 vmaqau.vx vd, vs1, rs2, v0.t。
- mask 对源操作数起作用，对应的操作数不参与乘法运算，一组 4 个源操作数均被 mask 时，累加操作数保持原值不变。

指令格式：

31 27	26	25	24 20	19 15	14 12	11 7	6 0
10001	1	vm	vs2	rs1	110	vd	0001011

19.7.1.6　VMAQASU.VX—8-bit 矢量标量有符号无符号乘累加链加指令

语法：

vmaqasu.vx vd, rs1, vs2, vm #vector-vector

操作：

```
for (i=0; i<4; i++) {
  operand1[i] = BITS(rs1, (i+1)*8-1, i*8);
}
for(i=0; i<VLMAX; i++) {
  int32 temp = 0;
  for(j=0; j<4; j++){
        if unmasked or vm[i*4+j] == 1:
            temp += unsigned(vs2[i*4+j]) * signed(operand1[j]); #8-bit per element
  }
  if i<vl:
        vd[i] += temp
  else:
        vd[i] = vd[i]
```

}

执行权限：

M mode/ S mode/U mode

异常：

非法指令异常

影响标志位：

无

说明：

- vd 数据宽度为 SEW=32-bit，vs2/vs1 数据宽度为 SEW/4=8-bit，其他 SEW 配置非法。
- LMUL 支持 1/2/4/8/0.5，其他配置非法。
- 当 vm=1 时，指令未被 mask，对应汇编指令 vmaqasu.vx vd, rs1, vs2。
- 当 vm=0 时，指令被 mask，对应汇编指令 vmaqasu.vx vd, rs1, vs2, v0.t。
- mask 对源操作数起作用，对应的操作数不参与乘法运算，一组 4 个源操作数均被 mask 时，累加操作数保持原值不变。

指令格式：

31 27	26	25	24 20	19 15	14 12	11 7	6 0
10010	1	vm	vs2	rs1	110	vd	0001011

19.7.1.7　VMAQAUS.VX—8-bit 矢量标量无符号有符号乘累加链加指令

语法：

vmaqaus.vv vd, rs1, vs2, vm #vector-vector

操作：

```
for (i=0; i<4; i++) {
  operand1[i] = BITS(rs1, (i+1)*8-1, i*8);
}
for(i=0; i<VLMAX; i++) {
  int32 temp = 0;
  for(j=0; j<4; j++){
        if unmasked or vm[i*4+j] == 1:
            temp += signed(vs2[i*4+j]) * unsigned(operand1[j]); #8-bit per element
  }
  if i<vl:
        vd[i] += temp
```

```
  else:
        vd[i] = vd[i]
}
```

执行权限：

M mode/ S mode/U mode

异常：

非法指令异常

影响标志位：

无

说明：

- vd 数据宽度为 SEW=32-bit，vs2/vs1 数据宽度为 SEW/4=8-bit，其他 SEW 配置非法。
- LMUL 支持 1/2/4/8/0.5，其他配置非法。
- 当 vm=1 时，指令未被 mask，对应汇编指令 vmaqaus.vx vd, rs1, vs2。
- 当 vm=0 时，指令被 mask，对应汇编指令 vmaqaus.vx vd, rs1, vs2, v0.t。
- mask 对源操作数起作用，对应的操作数不参与乘法运算，一组 4 个源操作数均被 mask 时，累加操作数保持原值不变。

指令格式：

31 27	26	25	24 20	19 15	14 12	11 7	6 0
10011	1	vm	vs2	rs1	110	vd	0001011

19.7.1.8　VPMAQA.VV—4-bit 矢量有符号乘累加链加指令

语法：

vpmaqa.vv vd, vs1, vs2, vm #vector-vector

操作：

```
for(i=0; i<VLMAX; i++) {
  int32 temp = 0;
  for(j=0; j<4; j++){
        if unmasked or vm[i*4+j] == 1:
            temp += signed(vs2[i*8+2*j]) * signed(vs1[i*8+2*j]); #4-bit per element
            temp += signed(vs2[i*8+2*j+1]) * signed(vs1[i*8+2*j+1]); #4-bit per element
  }
  if i<vl:
        vd[i] += temp
```

```
  else:
        vd[i] = vd[i]
}
```

执行权限：

M mode/ S mode/U mode

异常：

非法指令异常

影响标志位：

无

说明：

- vd 数据宽度为 SEW=32-bit，vs2/vs1 数据宽度为 EEW=SEW/8=4-bit，其他 SEW 配置非法。
- LMUL 支持 1/2/4/8/0.5，其他配置非法。
- 当 vm=1 时，指令未被 mask，对应汇编指令 vpmaqa.vv vd, vs1, vs2。
- 当 vm=0 时，指令被 mask，对应汇编指令 vpmaqa.vv vd, vs1, vs2, v0.t。
- mask 对源操作数起作用，对应的操作数不参与乘法运算。mask 寄存器每个 bit 作用于 1 对 4-bit 操作数，一组 8 个源操作数均被 mask 时，累加操作数保持原值不变。

指令格式：

31 27	26	25	24 20	19 15	14 12	11 7	6 0
10000	0	vm	vs2	vs1	111	vd	0001011

19.7.1.9　VPMAQAU.VV——4-bit 矢量无符号乘累加链加指令

语法：

vpmaqau.vv vd, vs1, vs2, vm #vector-vector

操作：

```
for(i=0; i<VLMAX; i++) {
  int32 temp = 0;
  for(j=0; j<4; j++){
        if unmasked or vm[i*4+j] == 1:
            temp += unsigned(vs2[i*8+2*j]) * unsigned(vs1[i*8+2*j]); #4-bit per element
            temp += unsigned(vs2[i*4+2*j+1]) * unsigned(vs1[i*8+j*2+1]); #4-bit per element
  }
  if i<vl:
        vd[i] += temp
```

```
  else:
        vd[i] = vd[i]
}
```

执行权限：

M mode/ S mode/U mode

异常：

非法指令异常

影响标志位：

无

说明：

- vd 数据宽度为 SEW=32-bit，vs2/vs1 数据宽度为 EEW=SEW/8=4-bit，其他 SEW 配置非法。
- LMUL 支持 1/2/4/8/0.5，其他配置非法。
- 当 vm=1 时，指令未被 mask，对应汇编指令 vpmaqau.vv vd, vs1, vs2。
- 当 vm=0 时，指令被 mask，对应汇编指令 vpmaqau.vv vd, vs1, vs2, v0.t。
- mask 对源操作数起作用，对应的操作数不参与乘法运算。mask 寄存器每个 bit 作用于 1 对 4-bit 操作数，一组 8 个源操作数均被 mask 时，累加操作数保持原值不变。

指令格式：

31 27	26	25	24 20	19 15	14 12	11 7	6 0
10001	0	vm	vs2	vs1	111	vd	0001011

19.7.1.10　VPMAQASU.VV—4-bit 矢量有符号无符号乘累加链加指令

语法：

vpmaqasu.vv vd, vs1, vs2, vm #vector-vector

操作：

```
for(i=0; i<VLMAX; i++) {
  int32 temp = 0;
  for(j=0; j<4; j++){
        if unmasked or vm[i*4+j] == 1:
            temp += usigned(vs2[i*8+j*2]) * signed(vs1[i*8+j*2]); #4-bit per element
            temp += usigned(vs2[i*8+j*2+1]) * signed(vs1[i*8+j*2+1]); #4-bit per element
  }
  if i<vl:
        vd[i] += temp
```

```
  else:
        vd[i] = vd[i]
}
```

执行权限：

M mode/ S mode/U mode

异常：

非法指令异常

影响标志位：

无

说明：

- vd 数据宽度为 SEW=32-bit，vs2/vs1 数据宽度为 EEW=SEW/8=4-bit，其他 SEW 配置非法。
- LMUL 支持 1/2/4/8/0.5，其他配置非法。
- 当 vm=1 时，指令未被 mask，对应汇编指令 vpmaqasu.vv vd, vs1, vs2。
- 当 vm=0 时，指令被 mask，对应汇编指令 vpmaqasu.vv vd, vs1, vs2, v0.t。
- mask 对源操作数起作用，对应的操作数不参与乘法运算。mask 寄存器每个 bit 作用于 1 对 4-bit 操作数，一组 8 个源操作数均被 mask 时，累加操作数保持原值不变。

指令格式：

31 27	26	25	24 20	19 15	14 12	11 7	6 0
10010	0	vm	vs2	vs1	111	vd	0001011

19.7.1.11　VPMAQA.VX—4-bit 矢量标量有符号乘累加链加指令

语法：

vpmaqa.vx vd, rs1, vs2, vm #vector-vector

操作：

```
for (i=0; i<8; i++) {
  operand1[i] = BITS(rs1, (i+1)*4-1, i*4);
}
  for(i=0; i<VLMAX; i++) {
        int32 temp = 0;
        for(j=0; j<4; j++){
            if unmasked or vm[i*4+j] == 1:
                temp += signed(vs2[i*8+2*j]) * signed(operand1[2*j]); #4-bit per element
                temp += signed(vs2[i*8+2*j+1]) * signed(operand1[2*j+1]); #4-bit per ele-
                ment
```

```
        }
        if i<vl:
            vd[i] += temp
        else:
            vd[i] = vd[i]
}
```

执行权限：

M mode/ S mode/U mode

异常：

非法指令异常

影响标志位：

无

说明：

- vd 数据宽度为 SEW=32-bit，vs2/vs1 数据宽度为 EEW=SEW/8=4-bit，其他 SEW 配置非法。
- LMUL 支持 1/2/4/8/0.5，其他配置非法。
- 当 vm=1 时，指令未被 mask，对应汇编指令 vpmaqa.vx vd, rs1, vs2。
- 当 vm=0 时，指令被 mask，对应汇编指令 vpmaqa.vx vd, rs1, vs2, v0.t。
- mask 对源操作数起作用，对应的操作数不参与乘法运算。mask 寄存器每个 bit 作用于 1 对 4-bit 操作数，一组 8 个源操作数均被 mask 时，累加操作数保持原值不变。

指令格式：

31 27	26	25	24 20	19 15	14 12	11 7	6 0
10000	1	vm	vs2	rs1	111	vd	0001011

19.7.1.12　VPMAQAU.VX——4-bit 矢量标量无符号乘累加链加指令

语法：

vpmaqau.vx vd, rs1, vs2, vm #vector-vector

操作：

```
for (i=0; i<8; i++) {
  operand1[i] = BITS(rs1, (i+1)*4-1, i*4);
}
for(i=0; i<VLMAX; i++) {
  int32 temp = 0;
  for(j=0; j<4; j++){
```

```
        if unmasked or vm[i*4+j] == 1:
            temp += unsigned(vs2[i*8+2*j]) * unsigned(operand1[2*j]); #4-bit per element
            temp += unsigned(vs2[i*8+2*j+1]) * unsigned(operands1[2*j+1]); #4-bit per element
    }
    if i<vl:
        vd[i] += temp
    else:
        vd[i] = vd[i]
}
```

执行权限：

M mode/ S mode/U mode

异常：

非法指令异常

影响标志位：

无

说明：

- vd 数据宽度为 SEW=32-bit，vs2/vs1 数据宽度为 EEW=SEW/8=4-bit，其他 SEW 配置非法。
- LMUL 支持 1/2/4/8/0.5，其他配置非法。
- 当 vm=1 时，指令未被 mask，对应汇编指令 vmaqau.vx vd, rs1, vs2。
- 当 vm=0 时，指令被 mask，对应汇编指令 vmaqau.vx vd, rs1, vs2, v0.t。
- mask 对源操作数起作用，对应的操作数不参与乘法运算。mask 寄存器每个 bit 作用于 1 对 4-bit 操作数，一组 8 个源操作数均被 mask 时，累加操作数保持原值不变。

指令格式：

31 27	26	25	24 20	19 15	14 12	11 7	6 0
10001	1	vm	vs2	rs1	111	vd	0001011

19.7.1.13　VPMAQASU.VX——4-bit 矢量标量有符号无符号乘累加链加指令

语法：

vpmaqasu.vx vd, rs1, vs2, vm #vector-vector

操作：

```
for (i=0; i<8; i++) {
  operand1[i] = BITS(rs1, (i+1)*4-1, i*4);
```

```
}
for(i=0; i<VLMAX; i++) {
  int32 temp = 0;
  for(j=0; j<4; j++){
        if unmasked or vm[i*4+j] == 1:
            temp += usigned(vs2[i*8+2*j]) * signed(operand1[2*j]); #4-bit per element
            temp += usigned(vs2[i*8+2*j+1]) * signed(operand1[2*j+1]); #4-bit per element
  }
  if i<vl:
        vd[i] += temp
  else:
        vd[i] = vd[i]
}
```

执行权限：

M mode/ S mode/U mode

异常：

非法指令异常

影响标志位：

无

说明：

- vd 数据宽度为 SEW=32-bit，vs2/vs1 数据宽度为 EEW=SEW/8=4-bit，其他 SEW 配置非法。
- LMUL 支持 1/2/4/8/0.5，其他配置非法。
- 当 vm=1 时，指令未被 mask，对应汇编指令 vmaqasu.vx vd, rs1, vs2。
- 当 vm=0 时，指令被 mask，对应汇编指令 vmaqasu.vx vd, rs1, vs2, v0.t。
- mask 对源操作数起作用，对应的操作数不参与乘法运算。mask 寄存器每个 bit 作用于 1 对 4-bit 操作数，一组 8 个源操作数均被 mask 时，累加操作数保持原值不变。

指令格式：

31 27	26	25	24 20	19 15	14 12	11 7	6 0
10010	1	vm	vs2	rs1	111	vd	0001011

19.7.1.14　VPMAQAUS.VX——4-bit 矢量标量无符号有符号乘累加链加指令

语法：

vpmaqaus.vv vd, rs1, vs2, vm #vector-vector

操作：

```
for (i=0; i<8; i++) {
  operand1[i] = BITS(rs1, (i+1)*4-1, i*4);
}
for(i=0; i<VLMAX; i++) {
  int32 temp = 0;
  for(j=0; j<4; j++){
        if unmasked or vm[i*4+j] == 1:
            temp += signed(vs2[i*8+2*j]) * unsigned(operand1[2*j]); #4-bit per element
            temp += signed(vs2[i*8+2*j+1]) * unsigned(operand1[2*j+1]); #4-bit per element
  }
  if i<vl:
        vd[i] += temp
  else:
        vd[i] = vd[i]
}
```

执行权限：

M mode/ S mode/U mode

异常：

非法指令异常

影响标志位：

无

说明：

- vd 数据宽度为 SEW=32-bit，vs2/vs1 数据宽度为 EEW=SEW/8=4-bit，其他 SEW 配置非法。
- LMUL 支持 1/2/4/8/0.5，其他配置非法。
- 当 vm=1 时，指令未被 mask，对应汇编指令 vpmaqaus.vx vd, rs1, vs2。
- 当 vm=0 时，指令被 mask，对应汇编指令 vpmaqaus.vx vd, rs1, vs2, v0.t。
- mask 对源操作数起作用，对应的操作数不参与乘法运算。mask 寄存器每个 bit 作用于 1 对 4-bit 操作数，一组 8 个源操作数均被 mask 时，累加操作数保持原值不变。

指令格式：

31 27	26	25	24 20	19 15	14 12	11 7	6 0
10011	1	vm	vs2	rs1	111	vd	0001011

19.7.1.15　VPNCLIP.WV——8-bit 矢量有符号缩位算术右移指令

语法：

vpnclip.wv vd, vs2, vs1, vm

操作：

```
for(i=0; i<VLMAX; i++) {
  char temp = 0;
  operand0[0] = vs2[i] & 0xff;
  operand0[1] = (vs2[i] & 0xff00) >> 8
  operand1[0] = vs1[i] & 0x7;
  operand1[1] = (vs1[i] & 0x70) >> 4
  temp = clip4(sign_extend(operand0[0]) >> operand1[0])&0xf;
  temp = temp | (clip4(sign_extend(operand0[1]) >> operand1[1]) << 4-bit);
  if vd[i] is active:
        vd[i] = temp
  else:
        vd[i] = vd[i]
}
```

执行权限：

M mode/ S mode/U mode

异常：

非法指令异常

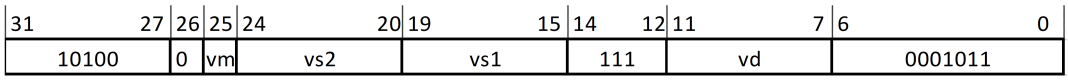
影响标志位：

VXSAT

说明：

- 支持 SEW=8-bit, LMUL 及寄存器遵从 narrow 指令定义。
- vd 数据位宽 SEW=8-bit，每个元素代表一对 4-bit 数据。
- vs2 数据位宽为 2*SEW=16-bit，每个元素代表一对 8-bit 数据。
- vs1 数据位宽为 SEW=8-bit，每个元素代表一对 4-bit 数据。
- 不受 vxrm 寄存器 rounding 模式影响，移出的数据直接舍弃。
- clip4 表示 saturation 到 4-bit 表示的有效数值，如果出现饱和，置 VXSAT 标识位。
- 当 vm=1 时，指令未被 mask，对应汇编指令 vpnclip.wv vd, vs2, vs1。
- 当 vm=0 时，指令被 mask，对应汇编指令 vpnclip.wv vd, vs2, vs1, v0.t。

指令格式：



31 27	26	25	24 20	19 15	14 12	11 7	6 0
10100	0	vm	vs2	vs1	111	vd	0001011

19.7.1.16　VPNCLIPU.WV—8-bit 矢量无符号缩位算术右移指令

语法：

vpnclipu.wv vd, vs2, vs1, vm

操作：

```
for(i=0; i<VLMAX; i++) {
  unsigned char temp = 0;
  unsigned char operand0[0] = vs2[i] & 0xff;
  unsigned char operand0[1] = (vs2[i] & 0xff00) >> 8
  unsigned char operand1[0] = vs1[i] & 0x7;
  unsigned char operand1[1] = (vs1[i] & 0x70) >> 4
  temp = clip4(zero_extend(operand0[0]) >> operand1[0]) & 0xf;
  temp = temp | (clip4(zero_extend(operand0[1]) >> operand1[1]) << 4-bit);
  if vd[i] is active:
        vd[i] = temp
  else:
        vd[i] = vd[i]
}
```

执行权限：

M mode/ S mode/U mode

异常：

非法指令异常

影响标志位：

VXSAT

说明：

- 支持 SEW=8-bit, LMUL 及寄存器遵从 narrow 指令定义。
- vd 数据位宽 SEW=8-bit，每个元素代表一对 4-bit 数据。
- vs2 数据位宽为 2*SEW=16-bit，每个元素代表一对 8-bit 数据。
- vs1 数据位宽为 SEW=8-bit，每个元素代表一对 4-bit 数据。
- 不受 vxrm 寄存器 rounding 模式影响，移出的数据直接舍弃。

- clip4 表示 saturation 到 4-bit 表示的有效数值，如果出现饱和，置 VXSAT 标识位。
- 当 vm=1 时，指令未被 mask，对应汇编指令 vpnclipu.wv vd, vs2, vs1。
- 当 vm=0 时，指令被 mask，对应汇编指令 vpnclipu.wv vd, vs2, vs1, v0.t。

指令格式：

31 27	26	25	24 20	19 15	14 12	11 7	6 0
10101	0	vm	vs2	vs1	111	vd	0001011

19.7.1.17　VPNCLIP.WX——8-bit 矢量标量缩位算术右移指令

语法：

vpnclip.wx vd, vs2, rs1, vm

操作：

operand1[0] = rs1 & 0x7;

operand1[1] = (rs1 & 0x70) >> 4

for(i=0; i<VLMAX; i++) {

  char temp = 0;

  operand0[0] = vs2[i] & 0xff;

  operand0[1] = (vs2[i] & 0xff00) >> 8

  temp = clip4(sign_extend(operand0[0]) >> operand1[0]) & 0xf; //4-bit element

  temp = temp | (clip4(sign_extend(operand0[1]) >> operand1[1]) << 4-bit);

  if vd[i] is active:

        vd[i] = temp

  else:

        vd[i] = vd[i]

}

执行权限：

M mode/ S mode/U mode

异常：

非法指令异常

影响标志位：

VXSAT

说明：

- 支持 SEW=8-bit, LMUL 及寄存器遵从 narrow 指令定义。
- vd 数据位宽 SEW=8-bit，每个元素代表一对 4-bit 数据。

- vs2 数据位宽为 2*SEW=16-bit，每个元素代表一对 8-bit 数据。
- vs1 数据位宽为 SEW=8-bit，每个元素代表一对 4-bit 数据。
- 不受 vxrm 寄存器 rounding 模式影响，移出的数据直接舍弃。
- clip4 表示 saturation 到 4-bit 表示的有效数值，如果出现饱和，置 VXSAT 标识位。
- 当 vm=1 时，指令未被 mask，对应汇编指令 vpnclip.wx vd, vs2, vs1。
- 当 vm=0 时，指令被 mask，对应汇编指令 vpnclip.wx vd, vs2, vs1, v0.t。

指令格式：

31 27	26	25	24 20	19 15	14 12	11 7	6 0
10100	1	vm	vs2	rs1	111	vd	0001011

19.7.1.18　VPNCLIPU.WX——8-bit 矢量标量无符号缩位算数右移指令

语法：

vpnclipu.wx vd, vs2, rs1, vm

操作：

```
unsigned char operand1[0] = rs1 & 0x7;
unsigned char operand1[1] = (rs1 & 0x70) >> 4
for(i=0; i<VLMAX; i++) {
  unsigned char temp = 0;
  unsigned char operand0[0] = vs2[i] & 0xff;
  unsigned char operand0[1] = (vs2[i] & 0xff00) >> 8
  temp = clip4(zero_extend(operand0[0]) >> operand1[0]) & 0xf;
  temp = temp | (clip4(zero_extend(operand0[1]) >> operand1[1]) << 4-bit);
  if vd[i] is active:
        vd[i] = temp
  else:
        vd[i] = vd[i]
}
```

执行权限：

M mode/ S mode/U mode

异常：

非法指令异常

影响标志位：

VXSAT

说明：

- 支持 SEW=8-bit, LMUL 及寄存器遵从 narrow 指令定义。
- vd 数据位宽 SEW=8-bit，每个元素代表一对 4-bit 数据。
- vs2 数据位宽为 2*SEW=16-bit，每个元素代表一对 8-bit 数据。
- vs1 数据位宽为 SEW=8-bit，每个元素代表一对 4-bit 数据。
- 不受 vxrm 寄存器 rounding 模式影响，移出的数据直接舍弃。
- clip4 表示 saturation 到 4-bit 表示的有效数值，如果出现饱和，置 VXSAT 标识位。
- 当 vm=1 时，指令未被 mask，对应汇编指令 vpnclipu.wx vd, vs2, rs1。
- 当 vm=0 时，指令被 mask，对应汇编指令 vpnclipu.wx vd, vs2, rs1, v0.t。

指令格式：

31 27	26	25	24 20	19 15	14 12	11 7	6 0
10101	1	vm	vs2	rs1	111	vd	0001011

19.7.1.19　VPWADD.VV——4-bit 扩位矢量整型有符号扩位加法指令

语法：

vpwadd.vv vd, vs2, vs1, vm

操作：

```
for(i=0; i<VLMAX; i++) {
  char res[0] = 0;
  char res[1] = 0;
  char operand0[0] = vs2[i] & 0xf;
  char operand0[1] = (vs2[i] & 0xf0) >> 4
  char operand1[0] = vs1[i] & 0xf;
  char operand1[1] = (vs1[i] & 0xf0) >> 4
  res[0] = signed_extend(operand0[0]) + signed_extend(operand1[0]);
  res[1] = signed_extend(operand0[1]) + signed_extend(operand1[1]);
  if vd[i] is active:
        vd[i] = {res[1], res[0]}
  else:
        vd[i] = vd[i]
}
```

执行权限：

M mode/ S mode/U mode

异常：

非法指令异常

影响标志位：

无

说明：

- 支持 SEW=8-bit, LMUL 及寄存器遵从 widen 指令定义。
- 不受 vxrm 寄存器 rounding 模式影响。
- vd 数据位宽为 2*SEW=16-bit，每个元素代表一对 8-bit 数据。
- vs1 数据位宽为 SEW=8-bit，每个元素代表一对 4-bit 数据。
- vs2 数据位宽 SEW=8-bit，每个元素代表一对 4-bit 数据。
- 当 vm=1 时，指令未被 mask，对应汇编指令 vpwadd.vv vd, vs2, rs1。
- 当 vm=0 时，指令被 mask，对应汇编指令 vpwadd.vv vd, vs2, rs1, v0.t。

指令格式：

31 27	26	25	24 20	19 15	14 12	11 7	6 0
10110	0	vm	vs2	vs1	111	vd	0001011

19.7.1.20　VPWADDU.VV—4-bit 扩位矢量整型无符号扩位加法指令

语法：

vpwaddu.vv vd, vs2, vs1, vm

操作：

```
for(i=0; i<VLMAX; i++) {
  char res[0] = 0;
  char res[1] = 0;
  char operand0[0] = vs2[i] & 0xf;
  char operand0[1] = (vs2[i] & 0xf0) >> 4
  char operand1[0] = vs1[i] & 0xf;
  char operand1[1] = (vs1[i] & 0xf0) >> 4
  res[0] = zero_extend(operand0[0]) + zero_extend(operand1[0]);
  res[1] = zero_extend(operand0[1]) + zero_extend(operand1[1]);
  if vd[i] is active:
        vd[i] = {res[1], res[0]}
  else:
        vd[i] = vd[i]
```

}

执行权限：

M mode/ S mode/U mode

异常：

非法指令异常

影响标志位：

无

说明：

- 支持 SEW=8-bit, LMUL 及寄存器遵从 widen 指令定义。
- 不受 vxrm 寄存器 rounding 模式影响。
- vd 数据位宽为 2*SEW=16-bit，每个元素代表一对 8-bit 数据。
- vs1 数据位宽为 SEW=8-bit，每个元素代表一对 4-bit 数据。
- vs2 数据位宽 SEW=8-bit，每个元素代表一对 4-bit 数据。
- 当 vm=1 时，指令未被 mask，对应汇编指令 vpwaddu.vv vd, vs2, rs1。
- 当 vm=0 时，指令被 mask，对应汇编指令 vpwaddu.vv vd, vs2, rs1, v0.t。

指令格式：

31 27	26	25	24 20	19 15	14 12	11 7	6 0
10111	0	vm	vs2	vs1	111	vd	0001011

19.7.1.21　VPWADD.VX——4-bit 扩位矢量标量整型有符号扩位加法指令

语法：

vpwadd.vx vd, vs2, rs1, vm

操作：

```
char operand1[0] = rs1 & 0xf;
char operand1[1] = (rs1 & 0xf0) >> 4
for(i=0; i<VLMAX; i++) {
  char res[0] = 0;
  char res[1] = 0;
  char operand0[0] = vs2[i] & 0xf;
  char operand0[1] = (vs2[i] & 0xf0) >> 4
  res[0] = signed_extend(operand0[0]) + signed_extend(operand1[0]);
  res[1] = signed_extend(operand0[1]) + signed_extend(operand1[1]);
  if vd[i] is active:
```

```
        vd[i] = {res[1], res[0]}
  else:
        vd[i] = vd[i]
}
```

执行权限：

M mode/ S mode/U mode

异常：

非法指令异常

影响标志位：

无

说明：

- 支持 SEW=8-bit, LMUL 及寄存器遵从 widen 指令定义。
- 不受 vxrm 寄存器 rounding 模式影响。
- vd 数据位宽为 2*SEW=16-bit，每个元素代表一对 8-bit 数据。
- vs1 数据位宽为 SEW=8-bit，每个元素代表一对 4-bit 数据。
- vs2 数据位宽 SEW=8-bit，每个元素代表一对 4-bit 数据。
- 当 vm=1 时，指令未被 mask，对应汇编指令 vpwadd.vx vd, vs2, rs1。
- 当 vm=0 时，指令被 mask，对应汇编指令 vpwadd.vx vd, vs2, rs1, v0.t。

指令格式：

31 27	26	25	24 20	19 15	14 12	11 7	6 0
10110	1	vm	vs2	rs1	111	vd	0001011

19.7.1.22　VPWADDU.VX—4-bit 扩位矢量标量整型无符号扩位加法指令

语法：

vpwaddu.vx vd, vs2, rs1, vm

操作：

```
char operand1[0] = rs1 & 0xf;
char operand1[1] = (rs1[i] & 0xf0) >> 4
for(i=0; i<VLMAX; i++) {
  char res[0] = 0;
  char res[1] = 0;
  char operand0[0] = vs2[i] & 0xf;
  char operand0[1] = (vs2[i] & 0xf0) >> 4
```

```
  res[0] = zero_extend(operand0[0]) + zero_extend(operand1[0]);
  res[1] = zero_extend(operand0[1]) + zero_extend(operand1[1]);
  if vd[i] is active:
        vd[i] = {res[1], res[0]}
  else:
        vd[i] = vd[i]
}
```

执行权限：

M mode/ S mode/U mode

异常：

非法指令异常

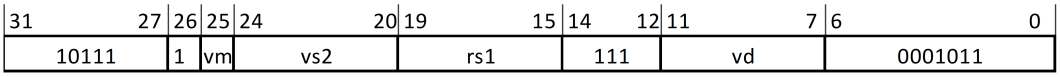
影响标志位：

无

说明：

- 支持 SEW=8-bit, LMUL 及寄存器遵从 widen 指令定义。
- 不受 vxrm 寄存器 rounding 模式影响。
- vd 数据位宽为 2*SEW=16-bit，每个元素代表一对 8-bit 数据。
- vs1 数据位宽为 SEW=8-bit，每个元素代表一对 4-bit 数据。
- vs2 数据位宽 SEW=8-bit，每个元素代表一对 4-bit 数据。
- 当 vm=1 时，指令未被 mask，对应汇编指令 vpwaddu.vx vd, vs2, rs1。
- 当 vm=0 时，指令被 mask，对应汇编指令 vpwaddu.vx vd, vs2, rs1, v0.t。

指令格式：



19.8　附录 B-9 Vector 扩展指令术语

19.8.1　Vector 扩展指令编码

1. Vector 扩展指令采用 custom-0 编码空间
2. 9 条 Vector 浮点转换扩展指令，26 条二维浮点 reduction 相关指令，复用浮点加载扩展指令和 AI 扩展指令的 func3 域（110）
3. 扩展指令的 func6 域编码接在 SFU 扩展指令的后面，使用 110010，采用 rs1 域对运算类型编码

表 19.1: 玄铁 Vector 扩展指令列表

类别	func6	vm	rs2	rs1	func3	rd	custom-0	指令名称
SFU 计算 *	110000	0/1	vs2	00000	110	vd	0001011	th.vfexp2
	110000	0/1	vs2	00001	110	vd	0001011	th.vftanh
	110000	0/1	vs2	00010	110	vd	0001011	th.vfsig
	110000	0/1	vs2	00011	110	vd	0001011	th.vfrec
BF16 vs FP8 转换 *	110010	0/1	vs2	10000	110	vd	0001011	th.vfncvt.e4.h
	110010	0/1	vs2	10001	110	vd	0001011	th.vfncvt.e5.h
	110010	0/1	vs2	10010	110	vd	0001011	th.vfncvt.e4.bf16
	110010	0/1	vs2	10011	110	vd	0001011	th.vfncvt.e5.bf16
	110010	0/1	vs2	10111	110	vd	0001011	th.vfncvt.rod.bf16.s
	110010	0/1	vs2	11000	110	vd	0001011	th.vfwcvt.h.e4
	110010	0/1	vs2	11001	110	vd	0001011	th.vfwcvt.h.e5
	110010	0/1	vs2	11010	110	vd	0001011	th.vfwcvt.bf16.e4
	110010	0/1	vs2	11011	110	vd	0001011	th.vfwcvt.bf16.e5
REDUCTION	110100	0	vs2	00000	110	vd	0001011	th.vfredsum.dup.32
	110100	0	vs2	00001	110	vd	0001011	th.vfredsum.dup.64
	110100	0	vs2	00010	110	vd	0001011	th.vfredmax.dup.32
	110100	0	vs2	00011	110	vd	0001011	th.vfredmax.dup.64
	110100	0	vs2	00100	110	vd	0001011	th.vfredmin.dup.32
	110100	0	vs2	00101	110	vd	0001011	th.vfredmin.dup.64
	110100	0	vs2	10000	110	vd	0001011	th.vfredsum.c.32
	110100	0	vs2	10001	110	vd	0001011	th.vfredsum.c.64
	110100	0	vs2	10010	110	vd	0001011	th.vfredmax.c.32
	110100	0	vs2	10011	110	vd	0001011	th.vfredmax.c.64
	110100	0	vs2	10100	110	vd	0001011	th.vfredmin.c.32
	110100	0	vs2	10101	110	vd	0001011	th.vfredmin.c.64
	110100	1	vs2	00000	110	vd	0001011	th.vbfredsum.dup.32
	110100	1	vs2	00001	110	vd	0001011	th.vbfredsum.dup.64
	110100	1	vs2	00010	110	vd	0001011	th.vbfredmax.dup.32
	110100	1	vs2	00011	110	vd	0001011	th.vbfredmax.dup.64
	110100	1	vs2	00100	110	vd	0001011	th.vbfredmin.dup.32
	110100	1	vs2	00101	110	vd	0001011	th.vbfredmin.dup.64
	110100	1	vs2	10000	110	vd	0001011	th.vbfredsum.c.32
	110100	1	vs2	10001	110	vd	0001011	th.vbfredsum.c.64

续下页

表 19.1 - 接上页

类别	func6	vm	rs2	rs1	func3	rd	custom-0	指令名称
	110100	1	vs2	10010	110	vd	0001011	th.vbfredmax.c.32
	110100	1	vs2	10011	110	vd	0001011	th.vbfredmax.c.64
	110100	1	vs2	10100	110	vd	0001011	th.vbfredmin.c.32
	110100	1	vs2	10101	110	vd	0001011	th.vbfredmin.c.64
	110100	0	vs2	01000	110	vd	0001011	th.vary.dup.32
	110100	0	vs2	01001	110	vd	0001011	th.vary.dup.64

* : The destination vector register group for a masked vector instructioncannot overlap the source mask register (v0).

19.8.2　VECTOR 扩展指令描述

19.8.2.1　th.vfexp2.v

Synopsis:

Output vector 2^x of the input vector x.

Mnemonic:

th.vfexp2.v vd, vs2, vm

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
110000	vm	vs2	00000	110	vd	0001011

Description:

This instruction returns 2^x estimate value of each elements of vs2 to vd. This instruction is based on RISC-V vector extension (RVV). The vl, vm, vta, vma is the same with RVV spec.

Operation:

vd[i] ← $2^{vs2[i]}$

Permission:

M mode/S mode/U mode

Exceptions:

if ((LMUL == 1/8)|(LMUL == 1/4)|(SEW == 8)|(SEW == 16)|(SEW == 64))

happens illegal

Supplement:

The subnormal input is supported. The following table describes the instruction's behavior for all classes of floating-point inputs:

表 19.2: the instruction's behavior for all classes of floating-point inputs

Input	Output	Exceptions raised
-∞	+0	None
-∞ < x < 0	estimate of 2^x	None
-0/+0	+1	None
0 < x < +∞	estimate of 2^x	None
+∞	+∞	None
QNAN	CNAN	None
SNAN	CNAN	NV

备注

- The rounding mode is specified by the hardware CSR register(frm), and all rounding mode is legal
- degree of accuracy is 2ulp
- The effected floating-point exception flags are NV/OF
- There is no NX, DZ and UF
- The instruction returns an estimate value of 2^x, and the accuracy of the results depends on the specific hardware implementation
- The instruction supports LMUL=1/2,1, 2, 4, 8 and SEW=32

19.8.2.2 th.vftanh.v

Synopsis:

Output vector tanh(x) of the input vector x

Mnemonic:

th.vftanh.v vd, vs2, vm

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
110000	vm	vs2	00001	110	vd	0001011

Description:

This instruction returns tanh(x) estimate value of each elements of vs2 to vd. This instruction is based on RISC-V vector extension (RVV). The vl, vm, vta, vma is the same with RVV spec

Operation:

1. tanh(x) ← $(e^{x} - e^{-x})/(e^{x} + e^{-x})$
2. vd[i] ← tanh(vs2[i])

Permission:

M mode/S mode/U mode

Exceptions:

if ((LMUL == 1/8)|(LMUL == 1/4)|(SEW == 8)|(SEW == 16)|(SEW == 64))

happens illegal

Supplement:

The subnormal input is supported. The following table describes the instruction's behavior for all classes of floating-point inputs:

表 19.3: the instruction's behavior for all classes of floating-point inputs

Input	Output	Exceptions raised
-∞	-1	None
-∞ < x < 0	estimate of tanh(x)	None
-0	-0	None
+0	+0	None
0 < x < +∞	estimate of tanh(x)	None
+∞	+1	None
QNAN	CNAN	None
SNAN	CNAN	NV

备注

- The rounding mode is specified by the hardwareCSR register(frm), and all rounding mode is legal
- degree of accuracy is 2ulp
- The effected floating-point exception flags are NV/OF
- There is no NX, DZ， OF and UF
- The instruction returns an estimate value of tanh(x), and the accuracy of the results depends on the specific hardware implementation
- The instruction supports LMUL=1/2,1, 2, 4, 8 and SEW=32.

19.8.2.3　th.vfsig.v

Synopsis:

Output vector sigmoid(x) of the input vector x

Mnemonic:

th.vfsig.v vd, vs2, vm

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
110000	vm	vs2	00010	110	vd	0001011

Description:

This instruction returns sigmoid(x) estimate value of each elements of vs2 to vd. This instruction is based on RISC-V vector extension (RVV). The vl, vm, vta, vma is the same with RVV spec

Operation:

1. sigmoid(x) ← e^x /(1+e^x)
2. vd[i] ← sigmoid(vs2[i])

Permission:

M mode/S mode/U mode

Exceptions:

if ((LMUL == 1/8)|(LMUL == 1/4)|(SEW == 8)|(SEW == 16)|(SEW == 64))

happens illegal

Supplement:

The subnormal input is supported. The following table describes the instruction's behavior for all classes of floating-point inputs:

表 19.4: the instruction's behavior for all classes of floating-point inputs

Input	Output	Exceptions raised
-∞	+0	None
-∞ < x < 0	estimate of sigmoid(x)	None
+0/-0	+0.5	None
0 < x < +∞	estimate of sigmoid(x)	None
+∞	+1	None
QNAN	CNAN	None
SNAN	CNAN	NV

备注

- The rounding mode is specified by the hardwareCSR register(frm), and all rounding mode is legal
- Degree of accuracy is 2ulp
- The effected floating-point exception flags are NV/OF
- There is no NX, DZ, OF and UF
- The instruction returns an estimate value of sigmoid(x), and the accuracy of the results depends on the specific hardware implementation
- The instruction supports LMUL=1/2,1, 2, 4, 8 and SEW=32

19.8.2.4　th.vfrec.v

Synopsis:

Output vector 1/x of the input vector x.

Mnemonic:

th.vfrec.v vd, vs2, vm

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
110000	vm	vs2	00000	110	vd	0001011

Description:

This instruction returns 1/x estimate value of each elements of vs2 to vd. This instruction is based on RISC-V vector extension (RVV). The vl, vm, vta, vma is the same with RVV spec

Operation:

vd[i] = 1/vs2[i]

Permission:

M mode/S mode/U mode

Exceptions:

if ((LMUL == 1/8)|(LMUL == 1/4)|(SEW == 8)|(SEW == 16)|(SEW == 64))

happens illegal

Supplement:

The subnormal input is supported. The following table describes the instruction's behavior for all classes of floating-point inputs:

表 19.5: the instruction's behavior for all classes of floating-point inputs

Input	Output	Exceptions raised
-∞	-0	None
-∞ < x < 0	estimate of 1/x	None
-0	-∞	DZ
+0	+∞	DZ
0 < x < +∞	estimate of 1/x	None
+∞	+0	None
QNAN	CNAN	None
SNAN	CNAN	NV

备注

- The rounding mode is specified by the hardwareCSR register(frm), and all rounding mode is legal
- Degree of accuracy is 2ulp
- The effected floating-point exception flags are NV/DZ/OF
- There is no NX and UF
- The instruction returns an estimate value of 1/x, and the accuracy of the results depends on the specific hardware implementation
- The instruction supports LMUL=1/2,1, 2, 4, 8 and SEW=32

19.8.2.5　th.vfncvt.e4.h

Synopsis:

Output the FP8.E4M3 format of elements This instruction is based on RISC-V vector extension (RVV). The vl, vm, vta, vma is the same with RVV spec

Mnemonic:

th.vfncvt.e4.h vd, vs2, vm

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
110010	0/1	vs2	10000	110	vd	0001011

Description:

Convert FP16 to FP8.E4M3 format

Operation:

FP8.E4M3 ← CONVERT(FP16)

Permission:

M mode/S mode/U mode

Exceptions:

if ((LMUL == 8)|(SEW == 16)|(SEW == 32)|(SEW == 64))

happens illegal

Supplement:

The subnormal input is supported. non-saturation mode or saturation mode is enabled by CSR. The following table describes the instruction's behavior for all classes of floating-point inputs:

- non-saturation mode

Input （e）	Output	Exceptions raised
SNAN/QNAN	NAN(0.1111.111)	SNAN?NV:None
+∞	NAN(0.1111.111)	NV
-∞	NAN(0.1111.111)	NV
Greater than max FP8 magnitude	NAN(0.1111.111)	NV
In FP8 range	CONVERT(FP16)	None
Small than FP8 subnormal magnitude	{+/-0}	None
{+/-0}	{+/-0}	None

- saturation mode

Input （e）	Output	Exceptions raised
SNAN/QNAN	NAN(0.1111.111)	SNAN?NV:None
+∞	+max_E4M3	None
-∞	-max_E4M3	None
Greater than max FP8 magnitude	+/- max_E4M3	None
In FP8 range	CONVERT(FP16)	None
Small than FP8 subnormal magnitude	{+/-0}	None
{+/-0}	{{+/-0}	None

备注

- The rounding mode is specified by the hardwareCSR register(frm), and all rounding mode is legal
- The instruction supports LMUL=1/8,1/4,1/2,1, 2, 4, vs2's element data width of 16 bits, vd's element data width of 8 bits, SEW=8.

- Normalization maybe generate OF,UF and NX exception

19.8.2.6　th.vfncvt.e5.h

Synopsis:

Output the FP8.E5M2 format of elements.This instruction is based on RISC-V vector extension (RVV). The vl, vm, vta, vma is the same with RVV spec

Mnemonic:

th.vfncvt.e5.h vd, vs2, vm

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
110010	0/1	vs2	10001	110	vd	0001011

Description:

Convert FP16 to FP8.E5M2 format

Operation:

FP8.E5M2 ← CONVERT(FP16)

Permission:

M mode/S mode/U mode

Exceptions:

if ((LMUL == 8)|(SEW == 16)|(SEW == 32)|(SEW == 64))

happens illegal

Supplement:

The subnormal input is supported. non-saturation mode or saturation mode is enabled by CSR. The following table describes the instruction’s behavior for all classes of floating-point inputs:

- non-saturation mode

Input （e）	Output	Exceptions raised
SNAN/QNAN	NAN(0.11111.11)	SNAN?NV:None
+∞	+∞	None
-∞	-∞	None
Greater than max FP8 magnitude	+/-∞	None
In FP8 range	CONVERT(FP16)	None
Small than FP8 subnormal magnitude	{+/-0}	None
{+/-0}	{+/-0}	None

- saturation mode

Input （e）	Output	Exceptions raised
SNAN/QNAN	NAN(0.11111.11)	SNAN?NV:None
+∞	+max_E5M2	None
-∞	-max_E5M2	None
Greater than max FP8 magnitude	+/- max_E5M2	None
In FP8 range	CONVERT(FP16)	None
Small than FP8 subnormal magnitude	{+/-0}	None
{+/-0}	{+/-0}	None

备注

- The rounding mode is specified by the hardwareCSR register(frm), and all rounding mode is legal
- The instruction supports LMUL=1/8,1/4,1/2,1, 2, 4, vs2 's element data width of 16 bits, vd's element data width of 8 bits, SEW=8
- Normalization maybe generate OF,UF and NX exception

19.8.2.7　th.vfncvt.e4.bf16

Synopsis:

Output the FP8.E4M3 format of elements.This instruction is based on RISC-V vector extension (RVV). The vl, vm, vta, vma is the same with RVV spec

Mnemonic:

th.vfncvt.e4.bf16 vd, vs2, vm

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
110010	0/1	vs2	10010	110	vd	0001011

Description:

Convert BF16 to FP8.E4M3 format

Operation:

FP8.E4M3 ← CONVERT(BF16)

Permission:

M mode/S mode/U mode

Exceptions:

if ((LMUL == 8)|(SEW == 16)|(SEW == 32)|(SEW == 64))

happens illegal

Supplement:

The subnormal input is supported. non-saturation mode or saturation mode is enabled by CSR. The following table describes the instruction's behavior for all classes of floating-point inputs:

- non-saturation

Input （e）	Output	Exceptions raised
SNAN/QNAN	NAN (E4M3 NAN=0.1111.111, 无 CNAN)	SNAN?NV:None
+∞	NAN(0.1111.111)	NV
-∞	NAN(0.1111.111)	NV
Greater than max FP8 magnitude	NAN(0.1111.111)	NV
In OFP8 range	CONVERT(BF16)	None
Small than FP8 subnormal magnitude	{+/-0}	None
{+/-0}	{+/-0}	None

- saturation

Input （e）	Output	Exceptions raised
SNAN/QNAN	NAN(0.1111.111)	SNAN?NV:None
+∞	+max_E4M3	None
-∞	-max_E4M3	None
Greater than max FP8 magnitude	+/- max_E4M3	None
In OFP8 range	CONVERT(BF16)	None
Small than FP8 subnormal magnitude	{+/-0}	None
{+/-0}	{{+/-0}	None

备注

- The rounding mode is specified by the hardwareCSR register(frm), and all rounding mode is legal
- The instruction supports LMUL=1/8,1/4,1/2,1, 2, 4, vs2 's element data width of 16 bits, vd's element data width of 8 bits, SEW=8
- Normalization maybe generate OF,UF and NX exception

19.8.2.8　th.vfncvt.e5.bf16

Synopsis:

Output the FP8.E5M2 format of elements.This instruction is based on RISC-V vector extension (RVV). The vl, vm, vta, vma is the same with RVV spec

Mnemonic:

th.vfncvt.e5.bf16 vd, vs2, vm

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
110010	0/1	vs2	10011	110	vd	0001011

Description:

Convert BF16 to FP8.E5M2 format

Operation:

FP8.E5M2 ← CONVERT(BF16)

Permission:

M mode/S mode/U mode

Exceptions:

if ((LMUL == 8)|(SEW == 16)|(SEW == 32)|(SEW == 64))

happens illegal

Supplement:

The subnormal input is supported. non-saturation mode or saturation mode is enabled by CSR. The following table describes the instruction's behavior for all classes of floating-point inputs:

- non-saturation mode

Input （e）	Output	Exceptions raised
SNAN/QNAN	NAN(0.11111.11)	SNAN?NV:None
+∞	+∞	None
-∞	-∞	None
Greater than max FP8 magnitude	+/-∞	
In FP8 range	CONVERT(BF16)	None
Small than FP8 subnormal magnitude	{+/-0}	None
{+/-0}	{+/-0}	None

- saturation mode

Input （e）	Output	Exceptions raised
SNAN/QNAN	NAN(0.11111.11)	SNAN?NV:None
+∞	+max_E5M2	None
-∞	-max_E5M2	None
Greater than max FP8 magnitude	+/- max_E5M2	
In FP8 range	CONVERT(BF16)	None
Small than FP8 subnormal magnitude	{+/-0}	None
{+/-0}	{+/-0}	None

备注

- The rounding mode is specified by the hardwareCSR register(frm), and all rounding mode is legal
- The instruction supports LMUL=1/8,1/4,1/2,1, 2, 4, vs2 's element data width of 16 bits, vd's element data width of 8 bits, SEW=8
- Normalization maybe generate OF,UF and NX exception

19.8.2.9　th.vfncvt.rod.bf16.s

Synopsis:

Output the BF16 format of elements.This instruction is based on RISC-V vector extension (RVV). The vl, vm, vta, vma is the same with RVV spec

Mnemonic:

th.vfncvt.rod.bf16.s vd, vs2, vm

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
110010	0/1	vs2	10111	110	vd	0001011

Description:

Convert FP32 to BF16 format. Rounding mode is ROD

Operation:

BF16 ← CONVERT(FP32)

Permission:

M mode/S mode/U mode

Exceptions:

if ((LMUL == 1/8)|(LMUL == 8)|(SEW == 8)|(SEW == 32)|(SEW == 64))

happens illegal

Supplement:

The subnormal input is supported. The following table describes the instruction's behavior for all classes of floating-point inputs:

Input （e）	Output	Exceptions raised
NAN	CNAN	SNAN?NV:None
+∞	+∞	None
-∞	-∞	None
{+/-0}	{+/-0}	None
others	CONVERT(FP32)	None

备注

- The rounding mode is ROD
- The instruction supports LMUL=1/4, 1/2,1, 2, 4, vs2 's element data width of 32 bits, vd's element data width of 16 bits, SEW=16
- Normalization maybe generate OF, UF and NX exception, none OF

19.8.2.10　th.vfwcvt.h.e4

Synopsis:

Output the FP16 format of elements.This instruction is based on RISC-V vector extension (RVV). The vl, vm, vta, vma is the same with RVV spec

Mnemonic:

th.vfwcvt.h.e4 vd, vs2, vm

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
110010	0/1	vs2	11000	110	vd	0001011

Description:

Convert FP8.E4M3 to FP16 format

Operation:

FP16 ← CONVERT(FP8.E4M3)

Permission:

M mode/S mode/U mode

Exceptions:

if ((LMUL == 8)|(SEW == 16)|(SEW == 32)|(SEW == 64))

happens illegal

Supplement:

The subnormal input is supported. The following table describes the instruction's behavior for all classes of floating-point inputs:

Input （e）	Output	Exceptions raised
NAN	CNAN	None
{+/-0}	{+/-0}	None
others	CONVERT(FP8.E4M3)	None

备注

- The rounding mode is specified by the hardwareCSR register(frm), and all rounding mode is legal
- The instruction supports LMUL=1/8,1/4,1/2,1, 2, 4, vd's element data width of 16 bits, vs2's element data width of 8 bits, SEW=8
- Normalization none OF,UF and NX exception

19.8.2.11　th.vfwcvt.h.e5

Synopsis:

Output the FP16 format of elements.This instruction is based on RISC-V vector extension (RVV). The vl, vm, vta, vma is the same with RVV spec

Mnemonic:

th.vfwcvt.h.e5 vd, vs2, vm

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
110010	0/1	vs2	11001	110	vd	0001011

Description:

Convert FP8.E5M2 to FP16 format

Operation:

FP16 ← CONVERT(FP8.E5M2)

Permission:

M mode/S mode/U mode

Exceptions:

if ((LMUL == 8)|(SEW == 16)|(SEW == 32)|(SEW == 64))

happens illegal

Supplement:

The subnormal input is supported. The following table describes the instruction's behavior for all classes of floating-point inputs:

Input （e）	Output	Exceptions raised
NAN	CNAN	SNAN?NV:None
+∞	+∞	None
-∞	-∞	None
{+/-0}	{+/-0}	None
others	CONVERT(FP8.E5M2)	None

备注

- The rounding mode is specified by the hardwareCSR register(frm), and all rounding mode is legal

- The instruction supports LMUL=1/8,1/4,1/2,1, 2, 4, vd's element data width of 16 bits, vs2's element data width of 8 bits, SEW=8
- Normalization none OF,UF and NX exception
- e5 的 snan 定义符合 RISC-V 规范，尾数为.01

19.8.2.12　th.vfwcvt.bf16.e4

Synopsis:

Output the BF16 format of elements.This instruction is based on RISC-V vector extension (RVV). The vl, vm, vta, vma is the same with RVV spec

Mnemonic:

th.vfwcvt.bf16.e4 vd, vs2, vm

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
110010	0/1	vs2	11010	110	vd	0001011

Description:

Convert FP8.E4M3 to BF16 format

Operation:

BF16 ← CONVERT(FP8.E4M3)

Permission:

M mode/S mode/U mode

Exceptions:

if ((LMUL == 8)|(SEW == 16)|(SEW == 32)|(SEW == 64))

happens illegal

Supplement:

The subnormal input is supported. The following table describes the instruction's behavior for all classes of floating-point inputs:

Input （e）	Output	Exceptions raised
NAN	CNAN	None
{+/-0}	{+/-0}	None
others	CONVERT(FP8.E4M3)	None

备注

- The rounding mode is specified by the hardwareCSR register(frm), and all rounding mode is legal
- The instruction supports LMUL=1/8,1/4,1/2,1, 2, 4, vd's element data width of 16 bits, vs2's element data width of 8 bits, SEW=8
- Normalization none OF,UF and NX exception

19.8.2.13　th.vfwcvt.bf16.e5

Synopsis:

Output the BF16 format of elements.This instruction is based on RISC-V vector extension (RVV). The vl, vm, vta, vma is the same with RVV spec

Mnemonic:

th.vfwcvt.bf16.e5 vd, vs2, vm

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
110010	0/1	vs2	11011	110	vd	0001011

Description:

Convert FP8.E5M2 to BF16 format

Operation:

BF16 ← CONVERT(FP8.E5M2)

Permission:

M mode/S mode/U mode

Exceptions:

if ((LMUL == 8)|(SEW == 16)|(SEW == 32)|(SEW == 64))

happens illegal

Supplement:

The subnormal input is supported. The following table describes the instruction's behavior for all classes of floating-point inputs:

Input （e）	Output	Exceptions raised
NAN	CNAN	SNAN?NV:None
+∞	+∞	None
-∞	-∞	None
{+/-0}	{+/-0}	None
others	CONVERT(FP8.E5M2)	None

备注

- The rounding mode is specified by the hardwareCSR register(frm), and all rounding mode is legal
- The instruction supports LMUL=1/8,1/4,1/2,1, 2, 4, vd's element data width of 16 bits, vs2's element data width of 8 bits, SEW=8
- Normalization none OF,UF and NX exception

19.8.2.14　th.vfredsum.dup.32

Synopsis:

Output the SUM of elements

Mnemonic:

th.vfredsum.dup.32 vd, vs2

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
110100	0	vs2	00000	110	vd	0001011

Description:

This instruction returns the SUM of elements.

Operation:

```
1. // vs2[i] = FP16/FP32
2. // default vlen = 1024b
3. // vlmax = LMUL*vlen/(16,32)//(16,32)=SEW16?16:32
4.
5. for(i=0; i<vlmax/32; i++)
6. temp = 0;
7. for(j=0; j<32; j++)
```

(续下页)

(接上页)

```
8. temp = temp + vs2[i*32+j];
9. for(k=0; k<32; k++)
10.  vd[i*32+k] = temp;
```

Permission:

M mode/S mode/U mode

Exceptions:

if ((LMUL == 1/8)|(LMUL == 1/4)|(LMUL == 1/2)|(SEW == 8)|(SEW == 64))

happens illegal

Supplement:

The subnormal input is supported. non-saturation mode or saturation mode is enabled by CSR. The following table describes the instruction's behavior for all classes of floating-point inputs:

- non-saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	CNAN	NV
{+∞, -∞}	CNAN	NV
{+∞, +∞}	+∞	None
{-∞, -∞}	-∞	None
{+∞, None SNAN/QNAN and none {+/-∞}}	+∞	None
{-∞, None SNAN/QNAN and none {+/-∞}}	-∞	None
{None SNAN/QNAN and none {+/-∞},+∞}	+∞	None
{None SNAN/QNAN and none {+/-∞},-∞}	-∞	None
others	SUM of elements	None

- saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	CNAN	NV
{+∞, -∞}	CNAN	NV
{+∞, +∞}	+max_FP16/FP32	None
{-∞, -∞}	-max_FP16/FP32	None
{+∞, None SNAN/QNAN and none {+/-∞}}	+max_FP16/FP32	None
{-∞, None SNAN/QNAN and none {+/-∞}}	-max_FP16/FP32	None
{None SNAN/QNAN and none {+/-∞},+∞}	+max_FP16/FP32	None
{None SNAN/QNAN and none {+/-∞},-∞}	-max_FP16/FP32	None
others	SUM of elements	None

备注

- The rounding mode is specified by the hardwareCSR register(frm), and all rounding mode is legal
- Instructions are not affected by VM and VL
- When SEW=16, The instruction supports LMUL=1, 2, 4, 8, vd/vs2's element data width of 16bits
- When SEW=32, The instruction supports LMUL=1, 2, 4, 8, vd/vs2's element data width of 32 bits
- When SEW=16, VLEN * LMUL is not an integer multiple of 512 bit(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction
- When SEW=32, VLEN * LMUL is not an integer multiple of 1024 bit(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction
- Elements are unorder
- Normalization maybe generate OF,UF and NX exception

19.8.2.15　th.vbfredsum.dup.32

Synopsis:

Output the SUM of elements

Mnemonic:

th.vbfredsum.dup.32 vd, vs2

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
110100	1	vs2	00000	110	vd	0001011

Description:

This instruction returns the SUM of elements.

Operation:

```
// vs2[i] = BF16
// default vlen = 1024b
// vlmax = LMUL*vlen/16

for(i=0; i<vlmax/32; i++)
temp = 0;
for(j=0; j<32; j++)
temp = temp + vs2[i*32+j];
for(k=0; k<32; k++)
vd[i*32+k] = temp;
```

Permission:

M mode/S mode/U mode

Exceptions:

if ((LMUL == 1/8)|(LMUL == 1/4)|(LMUL == 1/2)|(SEW == 8)|(SEW == 32)|(SEW == 64))

happens illegal

Supplement:

The subnormal input is supported. non-saturation mode or saturation mode is enabled by CSR. The following table describes the instruction's behavior for all classes of floating-point inputs:

- non-saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	CNAN	NV
{+∞, -∞}	CNAN	NV
{+∞, +∞}	+∞	None
{-∞, -∞}	-∞	None
{+∞, None SNAN/QNAN and none {+/-∞}}	+∞	None
{-∞, None SNAN/QNAN and none {+/-∞}}	-∞	None
{None SNAN/QNAN and none {+/-∞},+∞}	+∞	None
{None SNAN/QNAN and none {+/-∞},-∞}	-∞	None
others	SUM of elements	None

- non-saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	CNAN	NV
{+∞, -∞}	CNAN	NV
{+∞, +∞}	+max_BF16	None
{-∞, -∞}	-max_BF16	None
{+∞, None SNAN/QNAN and none {+/-∞}}	+max_BF16	None
{-∞, None SNAN/QNAN and none {+/-∞}}	-max_BF16	None
{None SNAN/QNAN and none {+/-∞},+∞}	+max_BF16	None
{None SNAN/QNAN and none {+/-∞},-∞}	-max_BF16	None
others	SUM of elements	None

备注

- The rounding mode is specified by the hardwareCSR register(frm), and all rounding mode is legal
- Instructions are not affected by VM and VL
- When SEW=16, The instruction supports LMUL=1, 2, 4, 8, vd/vs2's element data width of 16bits
- When SEW=16, VLEN * LMUL is not an integer multiple of 512 bit(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction
- elements are unorder

- Normalization maybe generate OF,UF and NX exception

19.8.2.16　th.vfredsum.dup.64

Synopsis:

Output the SUM of elements.

Mnemonic:

th.vfredsum.dup.64 vd, vs2

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
110100	0	vs2	00001	110	vd	0001011

Description:

This instruction returns the SUM of elements.

Operation:

```
1.  // vs2[i] = FP16/FP32
2.  // default vlen = 1024b
3.  // vlmax = LMUL*vlen/(16,32)//(16,32)=SEW16?16:32
4.
5. for(i=0; i<vlmax/64; i++)
6. temp = 0;
7. for(j=0; j<64; j++)
8. temp = temp + vs2[i*64+j];
9. for(k=0; k<64; k++)
10. vd[i*64+k] = temp;
```

Permission:

M mode/S mode/U mode

Exceptions:

if ((LMUL == 1/8)|(LMUL == 1/4)|(LMUL == 1/2)|((VLEN == 1024)&(LMUL == 1)&(SEW == 32))|(SEW == 8)|(SEW == 64))

happens illegal

Supplement:

The subnormal input is supported. non-saturation mode or saturation mode is enabled by CSR. The following table describes the instruction's behavior for all classes of floating-point inputs:

- non-saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	CNAN	NV
{+∞, -∞}	CNAN	NV
{+∞, +∞}	+∞	None
{-∞, -∞}	-∞	None
{+∞, None SNAN/QNAN and none {+/-∞}}	+∞	None
{-∞, None SNAN/QNAN and none {+/-∞}}	-∞	None
{None SNAN/QNAN and none {+/-∞}, +∞}	+∞	None
{None SNAN/QNAN and none {+/-∞}, -∞}	-∞	None
others	SUM of elements	None

- saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	CNAN	NV
{+∞, -∞}	CNAN	NV
{+∞, +∞}	+max_FP16/FP32	None
{-∞, -∞}	-max_FP16/FP32	None
{+∞, None SNAN/QNAN and none {+/-∞}}	+max_FP16/FP32	None
{-∞, None SNAN/QNAN and none {+/-∞}}	-max_FP16/FP32	None
{None SNAN/QNAN and none {+/-∞}, +∞}	+max_FP16/FP32	None
{None SNAN/QNAN and none {+/-∞}, -∞}	-max_FP16/FP32	None
others	SUM of elements	None

备注

- The rounding mode is specified by the hardwareCSR register(frm), and all rounding mode is legal
- Instructions are not affected by VM and VL
- When SEW=16, The instruction supports LMUL=1, 2, 4, 8, vd/vs2's element data width of 16bits
- When SEW=32, VLEN=1024, The instruction supports LMUL=2, 4, 8, vd/vs2's element data width of 32 bits

- When SEW=32, VLEN>1024, The instruction supports LMUL=1, 2, 4, 8, vd/vs2's element data width of 32 bits
- When SEW=16, VLEN * LMUL is not an integer multiple of 512 bit(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction
- When SEW=32, VLEN * LMUL is not an integer multiple of 1024 bit(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction
- Elements are unorder
- Normalization maybe generate OF,UF and NX exception

19.8.2.17　th.vbfredsum.dup.64

Synopsis:

Output the SUM of elements

Mnemonic:

th.vbfredsum.dup.64 vd, vs2

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
110100	1	vs2	00001	110	vd	0001011

Description:

This instruction returns the SUM of elements.

Operation:

```
1.  // vs2[i] = BF16
2.  // default vlen = 1024b
3.  // vlmax = LMUL*vlen/16
4.
5. for(i=0; i<vlmax/64; i++)
6. temp = 0;
7. for(j=0; j<64; j++)
8. temp = temp + vs2[i*64+j];
9. for(k=0; k<64; k++)
10. vd[i*64+k] = temp;
```

Permission:

M mode/S mode/U mode

Exceptions:

if ((LMUL == 1/8)|(LMUL == 1/4)|(LMUL == 1/2)|(SEW == 8)|(SEW == 32)|(SEW == 64))

happens illegal

Supplement:

The subnormal input is supported. non-saturation mode or saturation mode is enabled by CSR. The following table describes the instruction's behavior for all classes of floating-point inputs:

- non-saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	CNAN	NV
{+∞, -∞}	CNAN	NV
{+∞, +∞}	+∞	None
{-∞, -∞}	-∞	None
{+∞, None SNAN/QNAN and none {+/-∞}}	+∞	None
{-∞, None SNAN/QNAN and none {+/-∞}}	-∞	None
{None SNAN/QNAN and none {+/-∞}, +∞}	+∞	None
{None SNAN/QNAN and none {+/-∞}, -∞}	-∞	None
others	SUM of elements	None

- saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	CNAN	NV
{+∞, -∞}	CNAN	NV
{+∞, +∞}	+max_BF16	None
{-∞, -∞}	-max_BF16	None
{+∞, None SNAN/QNAN and none {+/-∞}}	+max_BF16	None
{-∞, None SNAN/QNAN and none {+/-∞}}	-max_BF16	None
{None SNAN/QNAN and none {+/-∞}, +∞}	+max_BF16	None
{None SNAN/QNAN and none {+/-∞}, -∞}	-max_BF16	None
others	SUM of elements	None

备注

- The rounding mode is specified by the hardwareCSR register(frm), and all rounding mode is legal
- Instructions are not affected by VM and VL
- When SEW=16, The instruction supports LMUL=1, 2, 4, 8, vd/vs2's element data width of 16bits
- When SEW=16, VLEN * LMUL is not an integer multiple of 512 bit(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction
- Elements are unorder
- Normalization maybe generate OF,UF and NX exception

19.8.2.18　th.vfredsum.c.32

Synopsis:

Output the SUM of elements

Mnemonic:

th.vfredsum.c.32 vd, vs2

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
110100	0	vs2	10000	110	vd	0001011

Description:

This instruction returns the SUM of elements.

Operation:

```
1. // vs2[i] = FP16/FP32
2. // default vlen = 1024b
3. // vlmax = LMUL*vlen/(16,32)//(16,32)=SEW16?16:32
4.
5. for(i=0; i<vlmax/32; i++)
6. temp = 0;
7. for(j=0; j<32; j++)
8. temp = temp + vs2[i*32+j];
9. vd[i] = temp;
```

Permission:

M mode/S mode/U mode

Exceptions:

if ((LMUL == 1/8)|(LMUL == 1/4)|(LMUL == 1/2)|(SEW == 8)|(SEW == 64))

happens illegal

Supplement:

The subnormal input is supported. non-saturation mode or saturation mode is enabled by CSR. The following table describes the instruction's behavior for all classes of floating-point inputs:

- non-saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	CNAN	NV
{+∞, -∞}	CNAN	NV
{+∞, +∞}	+∞	None
{-∞, -∞}	-∞	None
{+∞, None SNAN/QNAN and none {+/-∞}}	+∞	None
{-∞, None SNAN/QNAN and none {+/-∞}}	-∞	None
{None SNAN/QNAN and none {+/-∞}, +∞}	+∞	None
{None SNAN/QNAN and none {+/-∞}, -∞}	-∞	None
others	SUM of elements	None

- saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	CNAN	NV
{+∞, -∞}	CNAN	NV
{+∞, +∞}	+max_FP16/FP32	None
{-∞, -∞}	-max_FP16/FP32	None
{+∞, None SNAN/QNAN and none {+/-∞}}	+max_FP16/FP32	None
{-∞, None SNAN/QNAN and none {+/-∞}}	-max_FP16/FP32	None
{None SNAN/QNAN and none {+/-∞}, +∞}	+max_FP16/FP32	None
{None SNAN/QNAN and none {+/-∞}, -∞}	-max_FP16/FP32	None
others	SUM of elements	None

备注

- The rounding mode is specified by the hardwareCSR register(frm), and all rounding mode is legal
- Instructions are not affected by VM and VL
- When SEW=16,The instruction supports LMUL=1, 2, 4, 8, vd/vs2's element data width of 16bits
- When SEW=32,The instruction supports LMUL=1, 2, 4, 8, vd/vs2's element data width of 32 bits
- When SEW=16, VLEN * LMUL is not an integer multiple of 512 bit(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction
- When SEW=32, VLEN * LMUL is not an integer multiple of 1024 bit(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction
- Elements are unorder
- Normalization maybe generate OF,UF and NX exception

19.8.2.19　th.vbfredsum.c.32

Synopsis:

Output the SUM of elements

Mnemonic:

th.vbfredsum.c.32 vd, vs2

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
110100	1	vs2	10000	110	vd	0001011

Description:

This instruction returns the SUM of elements.

Operation:

```
// vs2[i] = BF16
// default vlen = 1024b
// vlmax = LMUL*vlen/16

for(i=0; i<vlmax/32; i++)
temp = 0;
for(j=0; j<32; j++)
temp = temp + vs2[i*32+j];
vd[i] = temp;
```

Permission:

M mode/S mode/U mode

Exceptions:

if ((LMUL == 1/8)|(LMUL == 1/4)|(LMUL == 1/2)|(SEW == 8)|(SEW == 32)|(SEW == 64))

happens illegal

Supplement:

The subnormal input is supported. non-saturation mode or saturation mode is enabled by CSR. The following table describes the instruction's behavior for all classes of floating-point inputs:

- non-saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	CNAN	NV
{+∞, -∞}	CNAN	NV
{+∞, +∞}	+∞	None
{-∞, -∞}	-∞	None
{+∞, None SNAN/QNAN and none {+/-∞}}	+∞	None
{-∞, None SNAN/QNAN and none {+/-∞}}	-∞	None
{None SNAN/QNAN and none {+/-∞}, +∞}	+∞	None
{None SNAN/QNAN and none {+/-∞}, -∞}	-∞	None
others	SUM of elements	None

- saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	CNAN	NV
{+∞, -∞}	CNAN	NV
{+∞, +∞}	+max_BF16	None
{-∞, -∞}	-max_BF16	None
{+∞, None SNAN/QNAN and none {+/-∞}}	+max_BF16	None
{-∞, None SNAN/QNAN and none {+/-∞}}	-max_BF16	None
{None SNAN/QNAN and none {+/-∞}, +∞}	+max_BF16	None
{None SNAN/QNAN and none {+/-∞}, -∞}	-max_BF16	None
others	SUM of elements	None

备注

- The rounding mode is specified by the hardwareCSR register(frm), and all rounding mode is legal
- Instructions are not affected by VM and VL
- When SEW=16, The instruction supports LMUL=1, 2, 4, 8, vd/vs2's element data width of 16 bits
- When SEW=16, VLEN * LMUL is not an integer multiple of 512 bit(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction

- Elements are unorder
- Normalization maybe generate OF,UF and NX exception

19.8.2.20　th.vfredsum.c.64

Synopsis:

Output the SUM of elements

Mnemonic:

th.vfredsum.c.64 vd, vs2

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
110100	0	vs2	10001	110	vd	0001011

Description:

This instruction returns the SUM of elements.

Operation:

```
// vs2[i] = FP16/FP32
// default vlen = 1024b
// vlmax = LMUL*vlen/(16,32)//(16,32)=SEW16?16:32

for(i=0; i<vlmax/64; i++)
temp = 0;
for(j=0; j<64; j++)
temp = temp + vs2[i*64+j];
vd[i] = temp;
```

Permission:

M mode/S mode/U mode

Exceptions:

if ((LMUL == 1/8)|(LMUL == 1/4)|(LMUL == 1/2)|((VLEN == 1024)&(LMUL == 1)&(SEW == 32))|(SEW == 8)|(SEW == 64))

happens illegal

Supplement:

The subnormal input is supported. non-saturation mode or saturation mode is enabled by CSR. The following table describes the instruction's behavior for all classes of floating-point inputs:

- non-saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	CNAN	NV
{+∞, -∞}	CNAN	NV
{+∞, +∞}	+∞	None
{-∞, -∞}	-∞	None
{+∞, None SNAN/QNAN and none {+/-∞}}	+∞	None
{-∞, None SNAN/QNAN and none {+/-∞}}	-∞	None
{None SNAN/QNAN and none {+/-∞}, +∞}	+∞	None
{None SNAN/QNAN and none {+/-∞}, -∞}	-∞	None
others	SUM of elements	None

- saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	CNAN	NV
{+∞, -∞}	CNAN	NV
{+∞, +∞}	+max_FP16/FP32	None
{-∞, -∞}	-max_FP16/FP32	None
{+∞, None SNAN/QNAN and none {+/-∞}}	+max_FP16/FP32	None
{-∞, None SNAN/QNAN and none {+/-∞}}	-max_FP16/FP32	None
{None SNAN/QNAN and none {+/-∞}, +∞}	+max_FP16/FP32	None
{None SNAN/QNAN and none {+/-∞}, -∞}	-max_FP16/FP32	None
others	SUM of elements	None

备注

- The rounding mode is specified by the hardwareCSR register(frm), and all rounding mode is legal
- Instructions are not affected by VM and VL
- When SEW=16, The instruction supports LMUL=1, 2, 4, 8, vd/vs2's element data width of 16bits
- When SEW=32, VLEN=1024, The instruction supports LMUL=2, 4, 8, vd/vs2's element data width of 32 bits
- When SEW=32, VLEN>1024, The instruction supports LMUL=1, 2, 4, 8, vd/vs2's element data width of 32 bits
- When SEW=16, VLEN * LMUL is not an integer multiple of 512 bit(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction
- When SEW=32, VLEN * LMUL is not an integer multiple of 1024 bit(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction
- Elements are unorder
- Normalization maybe generate OF,UF and NX exception

19.8.2.21　th.vbfredsum.c.64

Synopsis:

Output the SUM of elements.

Mnemonic:

th.vbfredsum.c.64 vd, vs2

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
110100	1	vs2	10001	110	vd	0001011

Description:

This instruction returns the SUM of elements.

Operation:

```
// vs2[i] = BF16
// default vlen = 1024b
// vlmax = LMUL*vlen/16
for(i=0; i<vlmax/64; i++)
temp = 0;
for(j=0; j<64; j++)
temp = temp + vs2[i*64+j];
vd[i] = temp;
```

Permission:

M mode/S mode/U mode

Exceptions:

if ((LMUL == 1/8)|(LMUL == 1/4)|(LMUL == 1/2)|(SEW == 8)|(SEW == 32)|(SEW == 64))

happens illegal

Supplement:

The subnormal input is supported. non-saturation mode or saturation mode is enabled by CSR. The following table describes the instruction's behavior for all classes of floating-point inputs:

- non-saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	CNAN	NV
{+∞, -∞}	CNAN	NV
{+∞, +∞}	+∞	None
{-∞, -∞}	-∞	None
{+∞, None SNAN/QNAN and none {+/-∞}}	+∞	None
{-∞, None SNAN/QNAN and none {+/-∞}}	-∞	None
{None SNAN/QNAN and none {+/-∞}, +∞}	+∞	None
{None SNAN/QNAN and none {+/-∞}, -∞}	-∞	None
others	SUM of elements	None

- saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	CNAN	NV
{+∞, -∞}	CNAN	NV
{+∞, +∞}	+max_BF16	None
{-∞, -∞}	-max_BF16	None
{+∞, None SNAN/QNAN and none {+/-∞}}	+max_BF16	None
{-∞, None SNAN/QNAN and none {+/-∞}}	-max_BF16	None
{None SNAN/QNAN and none {+/-∞}, +∞}	+max_BF16	None
{None SNAN/QNAN and none {+/-∞}, -∞}	-max_BF16	None
others	SUM of elements	None

备注

- The rounding mode is specified by the hardwareCSR register(frm), and all rounding mode is legal
- Instructions are not affected by VM and VL
- When SEW=16, The instruction supports LMUL=1, 2, 4, 8, vd/vs2's element data width of 16 bits
- When SEW=16, VLEN * LMUL is not an integer multiple of 512 bit(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction
- Elements are unorder

- Normalization maybe generate OF,UF and NX exception

19.8.2.22　th.vfredmax.dup.32

Synopsis:

Output the MAX of elements.

Mnemonic:

th.vfredmax.dup.32 vd, vs2

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
110100	0	vs2	10010	110	vd	0001011

Description:

This instruction returns the MAX of elements.

Operation:

```
1. // vs2[i] = FP16/FP32
2. // default vlen = 1024b
3. // vlmax = LMUL*vlen/(16,32)//(16,32)=SEW16?16:32
4.
5. for(i=0; i<vlmax/32; i++)
6. temp = vs2[i*32+0];
7. for(j=1; j<32; j++)
8. temp = temp > vs2[i*32+j] ? temp : vs2[i*32+j];
9. for(k=0; k<32; k++)
10. vd[i*32+k] = temp;
```

Permission:

M mode/S mode/U mode

Exceptions:

if ((LMUL == 1/8)|(LMUL == 1/4)|(LMUL == 1/2)|(SEW == 8)|(SEW == 64))

happens illegal

Supplement:

The subnormal input is supported. non-saturation mode or saturation mode is enabled by CSR. The following table describes the instruction's behavior for all classes of floating-point inputs:

- non-saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	+∞	None
{+∞, -∞}	+∞	None
{+∞, +∞}	+∞	None
{-∞, -∞}	-∞	None
{+∞, None SNAN/QNAN and none {+/-∞}}	+∞	None
{-∞, None SNAN/QNAN and none {+/-∞}}	e1	None
{None SNAN/QNAN and none {+/-∞}, +∞}	+∞	None
{None SNAN/QNAN and none {+/-∞}, -∞}	e0	None
others	MAX of elements	None

- saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	+max_FP16/FP32	None
{+∞, -∞}	+max_FP16/FP32	None
{+∞, +∞}	+max_FP16/FP32	None
{-∞, -∞}	-max_FP16/FP32	None
{+∞, None SNAN/QNAN and none {+/-∞}}	+max_FP16/FP32	None
{-∞, None SNAN/QNAN and none {+/-∞}}	e1	None
{None SNAN/QNAN and none {+/-∞}, +∞}	+max_FP16/FP32	None
{None SNAN/QNAN and none {+/-∞}, -∞}	e0	None
others	MAX of elements	None

备注

- The rounding mode is specified by the hardwareCSR register(frm), and all rounding mode is legal
- Instructions are not affected by VM and VL
- When SEW=16, The instruction supports LMUL=1, 2, 4, 8, vd/vs2's element data width of 16bits
- When SEW=32, The instruction supports LMUL=1, 2, 4, 8, vd/vs2's element data width of 32 bits
- When SEW=16, VLEN * LMUL is not an integer multiple of 512 bit(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction

- When SEW=32, VLEN * LMUL is not an integer multiple of 1024 bit(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction
- Elements are unorder

19.8.2.23　th.vbfredmax.dup.32

Synopsis:

Output the MAX of elements

Mnemonic:

th.vfredmax.dup.32 vd, vs2

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
110100	1	vs2	00010	110	vd	0001011

Description:

This instruction returns the MAX of elements.

Operation:

```
1. // vs2[i] = BF16
2. // default vlen = 1024b
3. // vlmax = LMUL*vlen/16
4.
5. for(i=0; i<vlmax/32; i++)
6. temp = vs2[i*32+0];
7. for(j=1; j<32; j++)
8. temp = temp > vs2[i*32+j] ? temp : vs2[i*32+j];
9. for(k=0; k<32; k++)
10. vd[i*32+k] = temp;
```

Permission:

M mode/S mode/U mode

Exceptions:

if ((LMUL == 1/8)|(LMUL == 1/4)|(LMUL == 1/2)|(SEW == 8)|(SEW == 32)|(SEW == 64))

happens illegal

Supplement:

The subnormal input is supported. non-saturation mode or saturation mode is enabled by CSR. The following table describes the instruction's behavior for all classes of floating-point inputs:

- non-saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	+∞	None
{+∞, -∞}	+∞	None
{+∞, +∞}	+∞	None
{-∞, -∞}	-∞	None
{+∞, None SNAN/QNAN and none {+/-∞}}	+∞	None
{-∞, None SNAN/QNAN and none {+/-∞}}	e1	None
{None SNAN/QNAN and none {+/-∞}, +∞}	+∞	None
{None SNAN/QNAN and none {+/-∞}, -∞}	e0	None
others	MAX of elements	None

- saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	+max_BF16	None
{+∞, -∞}	+max_BF16	None
{+∞, +∞}	+max_BF16	None
{-∞, -∞}	-max_BF16	None
{+∞, None SNAN/QNAN and none {+/-∞}}	+max_BF16	None
{-∞, None SNAN/QNAN and none {+/-∞}}	e1	None
{None SNAN/QNAN and none {+/-∞}, +∞}	+max_BF16	None
{None SNAN/QNAN and none {+/-∞}, -∞}	e0	None
others	MAX of elements	None

备注

- The rounding mode is specified by the hardwareCSR register(frm), and all rounding mode is legal
- Instructions are not affected by VM and VL
- When SEW=16, The instruction supports LMUL=1, 2, 4, 8, vd/vs2's element data width of 16bits

- When SEW=16, VLEN * LMUL is not an integer multiple of 512 bit(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction
- Elements are unorder

19.8.2.24　th.vfredmax.dup.64

Synopsis:

Output the MAX of elements

Mnemonic:

th.vfredmax.dup.64 vd, vs2

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
110100	0	vs2	00011	110	vd	0001011

Description:

This instruction returns the MAX of elements.

Operation:

```
1. // vs2[i] = FP16/FP32
2. // default vlen = 1024b
3. // vlmax = LMUL*vlen/(16,32)//(16,32)=SEW16?16:32
4.
5. for(i=0; i<vlmax/64; i++)
6. temp = vs2[i*64+0];
7. for(j=1; j<64; j++)
8. temp = temp > vs2[i*64+j] ? temp : vs2[i*64+j];
9. for(k=0; k<64; k++)
10. vd[i*64+k] = temp;
```

Permission:

M mode/S mode/U mode

Exceptions:

if ((LMUL == 1/8)|(LMUL == 1/4)|(LMUL == 1/2)|((VLEN == 1024)&(LMUL == 1)&(SEW == 32))|(SEW == 8)|(SEW == 64))

happens illegal

Supplement:

The subnormal input is supported. non-saturation mode or saturation mode is enabled by CSR. The following table describes the instruction's behavior for all classes of floating-point inputs:

- non-saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	+∞	None
{+∞, -∞}	+∞	None
{+∞, +∞}	+∞	None
{-∞, -∞}	-∞	None
{+∞, None SNAN/QNAN and none {+/-∞}}	+∞	None
{-∞, None SNAN/QNAN and none {+/-∞}}	e1	None
{None SNAN/QNAN and none {+/-∞}, +∞}	+∞	None
{None SNAN/QNAN and none {+/-∞}, -∞}	e0	None
others	MAX of elements	None

- saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	+max_FP16/FP32	None
{+∞, -∞}	+max_FP16/FP32	None
{+∞, +∞}	+max_FP16/FP32	None
{-∞, -∞}	-max_FP16/FP32	None
{+∞, None SNAN/QNAN and none {+/-∞}}	+max_FP16/FP32	None
{-∞, None SNAN/QNAN and none {+/-∞}}	e1	None
{None SNAN/QNAN and none {+/-∞}, +∞}	+max_FP16/FP32	None
{None SNAN/QNAN and none {+/-∞}, -∞}	e0	None
others	MAX of elements	None

备注

- The rounding mode is specified by the hardwareCSR register(frm), and all rounding mode is legal
- Instructions are not affected by VM and VL
- When SEW=16, The instruction supports LMUL=1, 2, 4, 8, vd/vs2's element data width of 16bits

- When SEW=32, VLEN=1024, The instruction supports LMUL=2, 4, 8, vd/vs2's element data width of 32 bits
- When SEW=32, VLEN>1024, The instruction supports LMUL=1, 2, 4, 8, vd/vs2's element data width of 32 bits
- When SEW=16, VLEN * LMUL is not an integer multiple of 512 bit(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction
- When SEW=32, VLEN * LMUL is not an integer multiple of 1024 bit(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction
- Elements are unorder

19.8.2.25　th.vbfredmax.dup.64

Synopsis:

Output the MAX of elements

Mnemonic:

th.vbfredmax.dup.64 vd, vs2

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
110100	1	vs2	00011	110	vd	0001011

Description:

This instruction returns the MAX of elements.

Operation:

```
1. // vs2[i] = BF16
2. // default vlen = 1024b
3. // vlmax = LMUL*vlen/16
4.
5. for(i=0; i<vlmax/64; i++)
6. temp = vs2[i*64+0];
7. for(j=1; j<64; j++)
8. temp = temp > vs2[i*64+j] ? temp : vs2[i*64+j];
9. for(k=0; k<64; k++)
10. vd[i*64+k] = temp;
```

Permission:

M mode/S mode/U mode

Exceptions:

if ((LMUL == 1/8)|(LMUL == 1/4)|(LMUL == 1/2)|(SEW == 8)|(SEW == 32)|(SEW == 64))

happens illegal

Supplement:

The subnormal input is supported. non-saturation mode or saturation mode is enabled by CSR. The following table describes the instruction's behavior for all classes of floating-point inputs:

- non-saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	+∞	None
{+∞, -∞}	+∞	None
{+∞, +∞}	+∞	None
{-∞, -∞}	-∞	None
{+∞, None SNAN/QNAN and none {+/-∞}}	+∞	None
{-∞, None SNAN/QNAN and none {+/-∞}}	e1	None
{None SNAN/QNAN and none {+/-∞}, +∞}	+∞	None
{None SNAN/QNAN and none {+/-∞}, -∞}	e0	None
others	MAX of elements	None

- saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	+max_BF16	None
{+∞, -∞}	+max_BF16	None
{+∞, +∞}	+max_BF16	None
{-∞, -∞}	-max_BF16	None
{+∞, None SNAN/QNAN and none {+/-∞}}	+max_BF16	None
{-∞, None SNAN/QNAN and none {+/-∞}}	e1	None
{None SNAN/QNAN and none {+/-∞}, +∞}	+max_BF16	None
{None SNAN/QNAN and none {+/-∞}, -∞}	e0	None
others	MAX of elements	None

备注

- The rounding mode is specified by the hardwareCSR register(frm), and all rounding mode is legal
- Instructions are not affected by VM and VL
- When SEW=16, The instruction supports LMUL=1, 2, 4, 8, vd/vs2's element data width of 16bits
- When SEW=16, VLEN * LMUL is not an integer multiple of 512 bit(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction
- Elements are unorder

19.8.2.26　th.vfredmax.c.32

Synopsis:

Output the MAX of elements

Mnemonic:

th.vfredmax.c.32 vd, vs2

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
110100	0	vs2	10010	110	vd	0001011

Description:

This instruction returns the MAX of elements.

Operation:

```
1. // vs2[i] = FP16/FP32
2. // default vlen = 1024b
3. // vlmax = LMUL*vlen/(16,32)//(16,32)=SEW16?16:32
4.
5. for(i=0; i<vlmax/32; i++)
6. temp = vs2[i*32+0];
7. for(j=1; j<32; j++)
8. temp = temp > vs2[i*32+j] ? temp : vs2[i*32+j];
9. vd[i] = temp;
```

Permission:

M mode/S mode/U mode

Exceptions:

if ((LMUL == 1/8)|(LMUL == 1/4)|(LMUL == 1/2)|(SEW == 8)|(SEW == 64))

happens illegal

Supplement:

The subnormal input is supported. non-saturation mode or saturation mode is enabled by CSR. The following table describes the instruction's behavior for all classes of floating-point inputs:

- non-saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{ayn value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	+∞	None
{+∞, -∞}	+∞	None
{+∞, +∞}	+∞	None
{-∞, -∞}	-∞	None
{+∞, None SNAN/QNAN and none {+/-∞}}	+∞	None
{-∞, None SNAN/QNAN and none {+/-∞}}	e1	None
{None SNAN/QNAN and none {+/-∞}, +∞}	+∞	None
{None SNAN/QNAN and none {+/-∞}, -∞}	e0	None
others	MAX of elements	None

- saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	+max_FP16/FP32	None
{+∞, -∞}	+max_FP16/FP32	None
{+∞, +∞}	+max_FP16/FP32	None
{-∞, -∞}	-max_FP16/FP32	None
{+∞, None SNAN/QNAN and none {+/-∞}}	+max_FP16/FP32	None
{-∞, None SNAN/QNAN and none {+/-∞}}	e1	None
{None SNAN/QNAN and none {+/-∞}, +∞}	+max_FP16/FP32	None
{None SNAN/QNAN and none {+/-∞}, -∞}	e0	None
others	MAX of elements	None

备注

- The rounding mode is specified by the hardwareCSR register(frm), and all rounding mode is legal
- Instructions are not affected by VM and VL
- When SEW=16, The instruction supports LMUL=1, 2, 4, 8, vd/vs2's element data width of 16bits
- When SEW=32, The instruction supports LMUL=1, 2, 4, 8, vd/vs2's element data width of 32 bits
- When SEW=16, VLEN * LMUL is not an integer multiple of 512 bit(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction
- When SEW=32, VLEN * LMUL is not an integer multiple of 1024 bit(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction
- Elements are unorder

19.8.2.27　th.vbfredmax.c.32

Synopsis:

Output the MAX of elements

Mnemonic:

th.vbfredmax.c.32 vd, vs2

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
110100	1	vs2	10010	110	vd	0001011

Description:

This instruction returns the MAX of elements.

Operation:

```
1. // vs2[i] = BF16
2. // default vlen = 1024b
3. // vlmax = LMUL*vlen/16
4.
5. for(i=0; i<vlmax; i++)
6. temp = vs2[i*32+0];
7. for(j=1; j<32; j++)
8. temp = temp > vs2[i*32+j] ? temp : vs2[i*32+j];
9. vd[i] = temp;
```

Permission:

M mode/S mode/U mode

Exceptions:

if ((LMUL == 1/8)|(LMUL == 1/4)|(LMUL == 1/2)|(SEW == 8)|(SEW == 32)|(SEW == 64))

happens illegal

Supplement:

The subnormal input is supported. non-saturation mode or saturation mode is enabled by CSR. The following table describes the instruction's behavior for all classes of floating-point inputs:

- non-saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	+∞	None
{+∞, -∞}	+∞	None
{+∞, +∞}	+∞	None
{-∞, -∞}	-∞	None
{+∞, None SNAN/QNAN and none {+/-∞}}	+∞	None
{-∞, None SNAN/QNAN and none {+/-∞}}	e1	None
{None SNAN/QNAN and none {+/-∞}, +∞}	+∞	None
{None SNAN/QNAN and none {+/-∞}, -∞}	e0	None
others	MAX of elements	None

- saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	+max_BF16	None
{+∞, -∞}	+max_BF16	None
{+∞, +∞}	+max_BF16	None
{-∞, -∞}	-max_BF16	None
{+∞, None SNAN/QNAN and none {+/-∞}}	+max_BF16	None
{-∞, None SNAN/QNAN and none {+/-∞}}	e1	None
{None SNAN/QNAN and none {+/-∞}, +∞}	+max_BF16	None
{None SNAN/QNAN and none {+/-∞}, -∞}	e0	None
others	MAX of elements	None

备注

- The rounding mode is specified by the hardwareCSR register(frm), and all rounding mode is legal
- Instructions are not affected by VM and VL
- When SEW=16, The instruction supports LMUL=1, 2, 4, 8, vd/vs2's element data width of 16 bits
- When SEW=16, VLEN * LMUL is not an integer multiple of 512 bit(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction
- Elements are unorder

19.8.2.28　th.vfredmax.c.64

Synopsis:

Output the MAX of elements

Mnemonic:

th.vfredmax.c.64 vd, vs2

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
110100	0	vs2	10011	110	vd	0001011

Description:

This instruction returns the MAX of elements.

Operation:

```
1. // vs2[i] = FP16/FP32
2. // default vlen = 1024b
3. // vlmax = LMUL*vlen/(16,32)//(16,32)=SEW16?16:32
4.
5. for(i=0; i<vlmax/64; i++)
6. temp = vs2[i*64+0];
7. for(j=1; j<64; j++)
8. temp = temp > vs2[i*64+j] ? temp : vs2[i*64+j];
9. vd[i] = temp;
```

Permission:

M mode/S mode/U mode

Exceptions:

if ((LMUL == 1/8)|(LMUL == 1/4)|(LMUL == 1/2)|((VLEN == 1024)&(LMUL == 1)&(SEW == 32))|(SEW == 8)|(SEW == 64))

happens illegal

Supplement:

The subnormal input is supported. non-saturation mode or saturation mode is enabled by CSR. The following table describes the instruction's behavior for all classes of floating-point inputs:

- non-saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	+∞	None
{+∞, -∞}	+∞	None
{+∞, +∞}	+∞	None
{-∞, -∞}	-∞	None
{+∞, None SNAN/QNAN and none {+/-∞}}	+∞	None
{-∞, None SNAN/QNAN and none {+/-∞}}	e1	None
{None SNAN/QNAN and none {+/-∞}, +∞}	+∞	None
{None SNAN/QNAN and none {+/-∞}, -∞}	e0	None
others	MAX of elements	None

- saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	+max_FP16/FP32	None
{+∞, -∞}	+max_FP16/FP32	None
{+∞, +∞}	+max_FP16/FP32	None
{-∞, -∞}	-max_FP16/FP32	None
{+∞, None SNAN/QNAN and none {+/-∞}}	+max_FP16/FP32	None
{-∞, None SNAN/QNAN and none {+/-∞}}	e1	None
{None SNAN/QNAN and none {+/-∞}, +∞}	+max_FP16/FP32	None
{None SNAN/QNAN and none {+/-∞}, -∞}	e0	None
others	MAX of elements	None

备注

- The rounding mode is specified by the hardwareCSR register(frm), and all rounding mode is legal
- Instructions are not affected by VM and VL
- When SEW=16, The instruction supports LMUL=1, 2, 4, 8, vd/vs2's element data width of 16bits
- WhenSEW=32, VLEN=1024, The instruction supports LMUL=2, 4, 8, vd/vs2's element data width of 32 bits
- WhenSEW=32, VLEN>1024, The instruction supports LMUL=1, 2, 4, 8, vd/vs2's element data width of 32 bits
- WhenSEW=16, VLEN * LMUL is not an integer multiple of 512 bit(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction
- WhenSEW=32, VLEN * LMUL is not an integer multiple of 1024 bit(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction
- Elements are unorder

19.8.2.29　th.vbfredmax.c.64

Synopsis:

Output the MAX of elements

Mnemonic:

th.vbfredmax.c.64 vd, vs2

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
110100	1	vs2	10011	110	vd	0001011

Description:

This instruction returns the MAX of elements.

Operation:

```
1. // vs2[i] = BF16
2. // default vlen = 1024b
3. // vlmax = LMUL*vlen/16
4.
5. for(i=0; i<vlmax/64; i++)
6. temp = vs2[i*64+0];
7. for(j=1; j<64; j++)
```

(续下页)

(接上页)

```
8. temp = temp > vs2[i*64+j] ? temp : vs2[i*64+j];
9. vd[i] = temp;
```

Permission:

M mode/S mode/U mode

Exceptions:

if ((LMUL == 1/8)|(LMUL == 1/4)|(LMUL == 1/2)|(SEW == 8)|(SEW == 32)|(SEW == 64))

happens illegal

Supplement:

The subnormal input is supported. non-saturation mode or saturation mode is enabled by CSR. The following table describes the instruction's behavior for all classes of floating-point inputs:

- non-saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	+∞	None
{+∞, -∞}	+∞	None
{+∞, +∞}	+∞	None
{-∞, -∞}	-∞	None
{+∞, None SNAN/QNAN and none {+/-∞}}	+∞	None
{-∞, None SNAN/QNAN and none {+/-∞}}	e1	None
{None SNAN/QNAN and none {+/-∞}, +∞}	+∞	None
{None SNAN/QNAN and none {+/-∞}, -∞}	e0	None
others	MAX of elements	None

- saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	+max_BF16	None
{+∞, -∞}	+max_BF16	None
{+∞, +∞}	+max_BF16	None
{-∞, -∞}	-max_BF16	None
{+∞, None SNAN/QNAN and none {+/-∞}}	+max_BF16	None
{-∞, None SNAN/QNAN and none {+/-∞}}	e1	None
{None SNAN/QNAN and none {+/-∞}, +∞}	+max_BF16	None
{None SNAN/QNAN and none {+/-∞}, -∞}	e0	None
others	MAX of elements	None

备注

- The rounding mode is specified by the hardwareCSR register(frm), and all rounding mode is legal
- Instructions are not affected by VM and VL
- WhenSEW=16, The instruction supports LMUL=1, 2, 4, 8, vd/vs2's element data width of 16 bits
- When SEW=16, VLEN * LMUL is not an integer multiple of 512 bit(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction
- Elements are unorder

19.8.2.30　th.vfredmin.dup.32

Synopsis:

Output the MIN of elements

Mnemonic:

th.vfredmin.dup.32 vd, vs2

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
110100	0	vs2	00100	110	vd	0001011

Description:

This instruction returns the MIN of elements.

Operation:

```
1.  // vs2[i] = FP16/FP32
2.  // default vlen = 1024b
3.  // vlmax = LMUL*vlen/(16,32)//(16,32)=SEW16?16:32
4.
5. for(i=0; i<vlmax/32; i++)
6. temp = vs2[i*32+0];
7. for(j=1; j<32; j++)
8. temp = temp < vs2[i*32+j] ? temp : vs2[i*32+j];
9. for(k=0; k<32; k++)
10. vd[i*32+k] = temp;
```

Permission:

M mode/S mode/U mode

Exceptions:

if ((LMUL == 1/8)|(LMUL == 1/4)|(LMUL == 1/2)|(SEW == 8)|(SEW == 64))

happens illegal

Supplement:

The subnormal input is supported. non-saturation mode or saturation mode is enabled by CSR. The following table describes the instruction's behavior for all classes of floating-point inputs:

- non-saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	-∞	None
{+∞, -∞}	-∞	None
{+∞, +∞}	+∞	None
{-∞, -∞}	-∞	None
{+∞, None SNAN/QNAN and none {+/-∞}}	e1	None
{-∞, None SNAN/QNAN and none {+/-∞}}	-∞	None
{None SNAN/QNAN and none {+/-∞}, +∞}	e0	None
{None SNAN/QNAN and none {+/-∞}, -∞}	-∞	None
others	MIN of elements	None

- saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	-max_FP16/FP32	None
{+∞, -∞}	-max_FP16/FP32	None
{+∞, +∞}	+max_FP16/FP32	None
{-∞, -∞}	-max_FP16/FP32	None
{+∞, None SNAN/QNAN and none {+/-∞}}	e1	None
{-∞, None SNAN/QNAN and none {+/-∞}}	-max_FP16/FP32	None
{None SNAN/QNAN and none {+/-∞}, +∞}	e0	None
{None SNAN/QNAN and none {+/-∞}, -∞}	-max_FP16/FP32	None
others	MIN of elements	None

备注

- The rounding mode is specified by the hardwareCSR register(frm), and all rounding mode is legal
- Instructions are not affected by VM and VL
- When SEW=16, The instruction supports LMUL=1, 2, 4, 8, vd/vs2's element data width of 16bits
- When SEW=32, The instruction supports LMUL=1, 2, 4, 8, vd/vs2's element data width of 32 bits
- When SEW=16, VLEN * LMUL is not an integer multiple of 512 bit(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction
- When SEW=32, VLEN * LMUL is not an integer multiple of 1024 bit(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction
- Elements are unorder

19.8.2.31　th.vbfredmin.dup.32

Synopsis:

Output the MIN of elements

Mnemonic:

th.vfredmin.dup.32 vd, vs2

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
110100	1	vs2	00100	110	vd	0001011

Description:

This instruction returns the MIN of elements.

Operation:

```
// vs2[i] = BF16
// default vlen = 1024b
// vlmax = LMUL*vlen/16

for(i=0; i<vlmax/32; i++)
temp = vs2[i*32+0];
for(j=1; j<32; j++)
temp = temp < vs2[i*32+j] ? temp : vs2[i*32+j];
for(k=0; k<32; k++)
vd[i*32+k] = temp;
```

Permission:

M mode/S mode/U mode

Exceptions:

if ((LMUL == 1/8)|(LMUL == 1/4)|(LMUL == 1/2)|(SEW == 8)|(SEW == 32)|(SEW == 64))

happens illegal

Supplement:

The subnormal input is supported. non-saturation mode or saturation mode is enabled by CSR. The following table describes the instruction's behavior for all classes of floating-point inputs:

- non-saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	-∞	None
{+∞, -∞}	-∞	None
{+∞, +∞}	+∞	None
{-∞, -∞}	-∞	None
{+∞, None SNAN/QNAN and none {+/-∞}}	e1	None
{-∞, None SNAN/QNAN and none {+/-∞}}	-∞	None
{None SNAN/QNAN and none {+/-∞}, +∞}	e0	None
{None SNAN/QNAN and none {+/-∞}, -∞}	-∞	None
others	MIN of elements	None

- saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	-max_BF16	None
{+∞, -∞}	-max_BF16	None
{+∞, +∞}	+max_BF16	None
{-∞, -∞}	-max_BF16	None
{+∞, None SNAN/QNAN and none {+/-∞}}	e1	None
{-∞, None SNAN/QNAN and none {+/-∞}}	-max_BF16	None
{None SNAN/QNAN and none {+/-∞}, +∞}	e0	None
{None SNAN/QNAN and none {+/-∞}, -∞}	-max_BF16	None
others	MIN of elements	None

备注

- The rounding mode is specified by the hardware CSR register(frm), and all rounding mode is legal
- Instructions are not affected by VM and VL
- When SEW=16, The instruction supports LMUL=1, 2, 4, 8, vd/vs2's element data width of 16bits
- When SEW=16, VLEN * LMUL is not an integer multiple of 512 bit(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction
- Elements are unorder

19.8.2.32　th.vfredmin.dup.64

Synopsis:

Output the MIN of elements

Mnemonic:

th.vfredmin.dup.64 vd, vs2

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
110100	0	vs2	00101	110	vd	0001011

Description:

This instruction returns the MIN of elements.

Operation:

```
// vs2[i] = FP16/FP32
// default vlen = 1024b
// vlmax = LMUL*vlen/(16,32)//(16,32)=SEW16?16:32

for(i=0; i<vlmax/64; i++)
temp = vs2[i*64+0];
for(j=1; j<64; j++)
temp = temp < vs2[i*64+j] ? temp : vs2[i*64+j];
for(k=0; k<64; k++)
vd[i*64+k] = temp;
```

Permission:

M mode/S mode/U mode

Exceptions:

if ((LMUL == 1/8)|(LMUL == 1/4)|(LMUL == 1/2)|((VLEN == 1024)&(LMUL == 1)&(SEW == 32))|(SEW == 8)|(SEW == 64))

happens illegal

Supplement:

The subnormal input is supported. non-saturation mode or saturation mode is enabled by CSR. The following table describes the instruction's behavior for all classes of floating-point inputs:

- non-saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	-∞	None
{+∞, -∞}	-∞	None
{+∞, +∞}	+∞	None
{-∞, -∞}	-∞	None
{+∞, None SNAN/QNAN and none {+/-∞}}	e1	None
{-∞, None SNAN/QNAN and none {+/-∞}}	-∞	None
{None SNAN/QNAN and none {+/-∞}, +∞}	e0	None
{None SNAN/QNAN and none {+/-∞}, -∞}	-∞	None
others	MIN of elements	None

- saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	-max_FP16/FP32	None
{+∞, -∞}	-max_FP16/FP32	None
{+∞, +∞}	+max_FP16/FP32	None
{-∞, -∞}	-max_FP16/FP32	None
{+∞, None SNAN/QNAN and none {+/-∞}}	e1	None
{-∞, None SNAN/QNAN and none {+/-∞}}	-max_FP16/FP32	None
{None SNAN/QNAN and none {+/-∞}, +∞}	e0	None
{None SNAN/QNAN and none {+/-∞}, -∞}	-max_FP16/FP32	None
others	MIN of elements	None

备注

- The rounding mode is specified by the hardware CSR register(frm), and all rounding mode is legal
- Instructions are not affected by VM and VL
- When SEW=16, The instruction supports LMUL=1, 2, 4, 8, vd/vs2's element data width of 16bits
- When SEW=32, VLEN=1024, The instruction supports LMUL=2, 4, 8, vd/vs2's element data width of 32 bits
- When SEW=32, VLEN>1024, The instruction supports LMUL=1, 2, 4, 8, vd/vs2's element data width of 32 bits
- When SEW=16, VLEN * LMUL is not an integer multiple of 512 bit(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction
- When SEW=32, VLEN * LMUL is not an integer multiple of 1024 bit(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction
- Elements are unorder

19.8.2.33　th.vbfredmin.dup.64

Synopsis:

Output the MIN of elements

Mnemonic:

th.vbfredmin.dup.64 vd, vs2

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
110100	1	vs2	00101	110	vd	0001011

Description:

This instruction returns the MIN of elements.

Operation:

```
1.  // vs2[i] = BF16
2.  // default vlen = 1024b
3.  // vlmax = LMUL*vlen/16
4.
5. for(i=0; i<vlmax/64; i++)
6. temp = vs2[i*64+0];
7. for(j=1; j<64; j++)
8. temp = temp < vs2[i*64+j] ? temp : vs2[i*64+j];
9. for(k=0; k<64; k++)
10.  vd[i*64+k] = temp;
```

Permission:

M mode/S mode/U mode

Exceptions:

if ((LMUL == 1/8)|(LMUL == 1/4)|(LMUL == 1/2)|(SEW == 8)|(SEW == 32)|(SEW == 64))

happens illegal

Supplement:

The subnormal input is supported. non-saturation mode or saturation mode is enabled by CSR. The following table describes the instruction’s behavior for all classes of floating-point inputs:

- non-saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	-∞	None
{+∞, -∞}	-∞	None
{+∞, +∞}	+∞	None
{-∞, -∞}	-∞	None
{+∞, None SNAN/QNAN and none {+/-∞}}	e1	None
{-∞, None SNAN/QNAN and none {+/-∞}}	-∞	None
{None SNAN/QNAN and none {+/-∞}, +∞}	e0	None
{None SNAN/QNAN and none {+/-∞}, -∞}	-∞	None
others	MIN of elements	None

- saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	-max_BF16	None
{+∞, -∞}	-max_BF16	None
{+∞, +∞}	+max_BF16	None
{-∞, -∞}	-max_BF16	None
{+∞, None SNAN/QNAN and none {+/-∞}}	e1	None
{-∞, None SNAN/QNAN and none {+/-∞}}	-max_BF16	None
{None SNAN/QNAN and none {+/-∞}, +∞}	e0	None
{None SNAN/QNAN and none {+/-∞}, -∞}	-max_BF16	None
others	MIN of elements	None

备注

- The rounding mode is specified by the hardwareCSR register(frm), and all rounding mode is legal
- Instructions are not affected by VM and VL
- When SEW=16, The instruction supports LMUL=1, 2, 4, 8, vd/vs2's element data width of 16bits
- When SEW=16, VLEN * LMUL is not an integer multiple of 512 bit(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction
- Elements are unorder

19.8.2.34　th.vfredmin.c.32

Synopsis:

Output the MIN of elements.

Mnemonic:

th.vfredmin.c.32 vd, vs2

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
110100	0	vs2	10100	110	vd	0001011

Description:

This instruction returns the MIN of elements.

Operation:

```
1.  // vs2[i] = FP16/FP32
2.  // default vlen = 1024b
3.  // vlmax = LMUL*vlen/(16,32)//(16,32)=SEW16?16:32
4.
5. for(i=0; i<vlmax/32; i++)
6. temp = vs2[i*32+0];
7. for(j=1; j<32; j++)
8. temp = temp < vs2[i*32+j] ? temp : vs2[i*32+j];
9. vd[i] = temp;
```

Permission:

M mode/S mode/U mode

Exceptions:

if ((LMUL == 1/8)|(LMUL == 1/4)|(LMUL == 1/2)|(SEW == 8)|(SEW == 64))

happens illegal

Supplement:

The subnormal input is supported. non-saturation mode or saturation mode is enabled by CSR. The following table describes the instruction's behavior for all classes of floating-point inputs:

- non-saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	-∞	None
{+∞, -∞}	-∞	None
{+∞, +∞}	+∞	None
{-∞, -∞}	-∞	None
{+∞, None SNAN/QNAN and none {+/-∞}}	e1	None
{-∞, None SNAN/QNAN and none {+/-∞}}	-∞	None
{None SNAN/QNAN and none {+/-∞}, +∞}	e0	None
{None SNAN/QNAN and none {+/-∞}, -∞}	-∞	None
others	MIN of elements	None

- saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	-max_FP16/FP32	None
{+∞, -∞}	-max_FP16/FP32	None
{+∞, +∞}	+max_FP16/FP32	None
{-∞, -∞}	-max_FP16/FP32	None
{+∞, None SNAN/QNAN and none {+/-∞}}	e1	None
{-∞, None SNAN/QNAN and none {+/-∞}}	-max_FP16/FP32	None
{None SNAN/QNAN and none {+/-∞}, +∞}	e0	None
{None SNAN/QNAN and none {+/-∞}, -∞}	-max_FP16/FP32	None
others	MIN of elements	None

备注

- The rounding mode is specified by the hardwareCSR register(frm), and all rounding mode is legal
- Instructions are not affected by VM and VL
- When SEW=16, The instruction supports LMUL=1, 2, 4, 8, vd/vs2's element data width of 16bits
- When SEW=32, The instruction supports LMUL=1， 2, 4, 8, vd/vs2's element data width of 32 bits
- When SEW=16, VLEN * LMUL is not an integer multiple of 512 bit(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction

- When SEW=32, VLEN * LMUL is not an integer multiple of 1024 bit(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction
- Elements are unorder

19.8.2.35　th.vbfredmin.c.32

Synopsis:

Output the MIN of elements.

Mnemonic:

th.vbfredmin.c.32 vd, vs2

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
110100	1	vs2	10100	110	vd	0001011

Description:

This instruction returns the MIN of elements.

Operation:

```
1.  // vs2[i] = BF16
2.  // default vlen = 1024b
3.  // vlmax = LMUL*vlen/16
4.
5. for(i=0; i<vlmax/32; i++)
6. temp = vs2[i*32+0];
7. for(j=1; j<32; j++)
8. temp = temp < vs2[i*32+j] ? temp : vs2[i*32+j];
9. vd[i] = temp;
```

Permission:

M mode/S mode/U mode

Exceptions:

if ((LMUL == 1/8)|(LMUL == 1/4)|(LMUL == 1/2)|(SEW == 8)|(SEW == 32)|(SEW == 64))

happens illegal

Supplement:

The subnormal input is supported. non-saturation mode or saturation mode is enabled by CSR. The following table describes the instruction's behavior for all classes of floating-point inputs:

- non-saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	-∞	None
{+∞, -∞}	-∞	None
{+∞, +∞}	+∞	None
{-∞, -∞}	-∞	None
{+∞, None SNAN/QNAN and none {+/-∞}}	e1	None
{-∞, None SNAN/QNAN and none {+/-∞}}	-∞	None
{None SNAN/QNAN and none {+/-∞}, +∞}	e0	None
{None SNAN/QNAN and none {+/-∞}, -∞}	-∞	None
others	MIN of elements	None

- saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	-max_BF16	None
{+∞, -∞}	-max_BF16	None
{+∞, +∞}	+max_BF16	None
{-∞, -∞}	-max_BF16	None
{+∞, None SNAN/QNAN and none {+/-∞}}	e1	None
{-∞, None SNAN/QNAN and none {+/-∞}}	-max_BF16	None
{None SNAN/QNAN and none {+/-∞}, +∞}	e0	None
{None SNAN/QNAN and none {+/-∞}, -∞}	-max_BF16	None
others	MIN of elements	None

备注

- The rounding mode is specified by the hardwareCSR register(frm), and all rounding mode is legal
- Instructions are not affected by VM and VL
- When SEW=16, The instruction supports LMUL=1, 2, 4, 8, vd/vs2's element data width of 16 bits
- When SEW=16, VLEN * LMUL is not an integer multiple of 512 bit(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction

- Elements are unorder

19.8.2.36　th.vfredmin.c.64

Synopsis:

Output the MIN of elements.

Mnemonic:

th.vfredmin.c.64 vd, vs2

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
110100	0	vs2	10101	110	vd	0001011

Description:

This instruction returns the MIN of elements.

Operation:

```
1.  // vs2[i] = FP16/FP32
2.  // default vlen = 1024b
3.  // vlmax = LMUL*vlen/(16,32)//(16,32)=SEW16?16:32
4.
5. for(i=0; i<vlmax/64; i++)
6. temp = vs2[i*64+0];
7. for(j=1; j<64; j++)
8. temp = temp < vs2[i*64+j] ? temp : vs2[i*64+j];
9. vd[i] = temp;
```

Permission:

M mode/S mode/U mode

Exceptions:

if ((LMUL == 1/8)|(LMUL == 1/4)|(LMUL == 1/2)|((VLEN == 1024)&(LMUL == 1)&(SEW == 32))|(SEW == 8)|(SEW == 64))

happens illegal

Supplement:

The subnormal input is supported. non-saturation mode or saturation mode is enabled by CSR. The following table describes the instruction's behavior for all classes of floating-point inputs:

- non-saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	-∞	None
{+∞, -∞}	-∞	None
{+∞, +∞}	+∞	None
{-∞, -∞}	-∞	None
{+∞, None SNAN/QNAN and none {+/-∞}}	e1	None
{-∞, None SNAN/QNAN and none {+/-∞}}	-∞	None
{None SNAN/QNAN and none {+/-∞}, +∞}	e0	None
{None SNAN/QNAN and none {+/-∞}, -∞}	-∞	None
others	MIN of elements	None

- saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	-max_FP16/FP32	None
{+∞, -∞}	-max_FP16/FP32	None
{+∞, +∞}	+max_FP16/FP32	None
{-∞, -∞}	-max_FP16/FP32	None
{+∞, None SNAN/QNAN and none {+/-∞}}	e1	None
{-∞, None SNAN/QNAN and none {+/-∞}}	-max_FP16/FP32	None
{None SNAN/QNAN and none {+/-∞}, +∞}	e0	None
{None SNAN/QNAN and none {+/-∞}, -∞}	-max_FP16/FP32	None
others	MIN of elements	None

备注

- The rounding mode is specified by the hardwareCSR register(frm), and all rounding mode is legal
- Instructions are not affected by VM and VL
- When SEW=16, The instruction supports LMUL=1, 2, 4, 8, vd/vs2's element data width of 16bits
- When SEW=32, VLEN=1024, The instruction supports LMUL=2, 4, 8, vd/vs2's element data width of 32 bits
- When SEW=32, VLEN>1024, The instruction supports LMUL=1, 2, 4, 8, vd/vs2's element data width of 32 bits

- When SEW=16, VLEN * LMUL is not an integer multiple of 512 bit(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction
- When SEW=32, VLEN * LMUL is not an integer multiple of 1024 bit(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction
- Elements are unorder

19.8.2.37　th.vbfredmin.c.64

Synopsis:

Output the MIN of elements.

Mnemonic:

th.vbfredmin.c.64 vd, vs2

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
110100	1	vs2	10101	110	vd	0001011

Description:

This instruction returns the MIN of elements.

Operation:

```
// vs2[i] = BF16
// default vlen = 1024b
// vlmax = LMUL*vlen/16

for(i=0; i<vlmax/64; i++)
temp = vs2[i*64+0];
for(j=1; j<64; j++)
temp = temp < vs2[i*64+j] ? temp : vs2[i*64+j];
vd[i] = temp;
```

Permission:

M mode/S mode/U mode

Exceptions:

if ((LMUL == 1/8)|(LMUL == 1/4)|(LMUL == 1/2)|(SEW == 8)|(SEW == 32)|(SEW == 64))

happens illegal

Supplement:

The subnormal input is supported. non-saturation mode or saturation mode is enabled by CSR. The following table describes the instruction's behavior for all classes of floating-point inputs:

- non-saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	-∞	None
{+∞, -∞}	-∞	None
{+∞, +∞}	+∞	None
{-∞, -∞}	-∞	None
{+∞, None SNAN/QNAN and none {+/-∞}}	e1	None
{-∞, None SNAN/QNAN and none {+/-∞}}	-∞	None
{None SNAN/QNAN and none {+/-∞}, +∞}	e0	None
{None SNAN/QNAN and none {+/-∞}, -∞}	-∞	None
others	MIN of elements	None

- saturation mode

Input （e0,e1）	Output	Exceptions raised
{SNAN/QNAN, any value except NAN}	CNAN	SNAN?NV:None
{any value except NAN, SNAN/QNAN}	CNAN	SNAN?NV:None
{-∞, +∞}	-max_BF16	None
{+∞, -∞}	-max_BF16	None
{+∞, +∞}	+max_BF16	None
{-∞, -∞}	-max_BF16	None
{+∞, None SNAN/QNAN and none {+/-∞}}	e1	None
{-∞, None SNAN/QNAN and none {+/-∞}}	-max_BF16	None
{None SNAN/QNAN and none {+/-∞}, +∞}	e0	None
{None SNAN/QNAN and none {+/-∞}, -∞}	-max_BF16	None
others	MIN of elements	None

备注

- The rounding mode is specified by the hardware CSR register(frm), and all rounding mode is legal
- Instructions are not affected by VM and VL
- When SEW=16, The instruction supports LMUL=1, 2, 4, 8, vd/vs2's element data width of 16 bits

- When SEW=16, VLEN * LMUL is not an integer multiple of 512 bit(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction
- Elements are unorder

19.8.2.38　th.vary.dup.32

Synopsis:

Output the new order of elements.

Mnemonic:

th.vary.dup.32 vd, vs2

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
110100	0	vs2	01000	110	vd	0001011

Description:

This instruction returns the new order of elements.

Operation:

```
// vs2[i] = FP16/FP32
// default vlen = 1024b
// vlmax = LMUL*vlen/32

for(i=0; i<vlmax/32; i++)
for(j=0; j<32; j++)
vd[i*32+j] = vs2[i];
```

Permission:

M mode/S mode/U mode

Exceptions:

if ((LMUL == 1/8)|(LMUL == 1/4)|(LMUL == 1/2)|(SEW == 8)|(SEW == 64))

happens illegal

Supplement:

- Instructions are not affected by VM and VL
- When SEW=16, The instruction supports LMUL=1, 2, 4, 8, vd/vs2's element data width of 16bits
- When SEW=32, The instruction supports LMUL=1， 2, 4, 8, vd/vs2's element data width of 32 bits

- When SEW=16, VLEN * LMUL is not an integer multiple of 512 bits(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction
- When SEW=32, VLEN * LMUL is not an integer multiple of 1024 bits(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction
- Elements are unorder

19.8.2.39　th.vary.dup.64

Synopsis:

Output the new order of elements.

Mnemonic:

th.vary.dup.64 vd, vs2

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
110100	0	vs2	01001	110	vd	0001011

Description:

This instruction returns the new order of elements.

Operation:

```
// vs2[i] = FP16/FP32
// default vlen = 1024b
// vlmax = LMUL*vlen/32

for(i=0; i<vlmax/64; i++)
for(j=0; j<64; j++)
vd[i*64+j] = vs2[i];
```

Permission:

M mode/S mode/U mode

Exceptions:

if ((LMUL == 1/8)|(LMUL == 1/4)|(LMUL == 1/2)|((VLEN == 1024)&(LMUL == 1)&(SEW == 32))|(SEW == 8)|(SEW == 64))

happens illegal

Supplement:

- Instructions are not affected by VM and VL

- When SEW=16, The instruction supports LMUL=1, 2, 4, 8, vd/vs2's element data width of 16 bits
- When SEW=32, VLEN=1024, The instruction supports LMUL=2, 4, 8, vd/vs2's element data width of 32 bits
- When SEW=32, VLEN>1024, The instruction supports LMUL=1, 2, 4, 8, vd/vs2's element data width of 32 bits
- When SEW=16, VLEN * LMUL is not an integer multiple of 512 bit(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction
- When SEW=32, VLEN * LMUL is not an integer multiple of 1024 bit(TITIAN configuration is for LANE_NUM=1), indicating an illegal instruction
- Elements are unorder

19.9　附录 B-10 部分玄铁扩展 Vector 指令遵循的 RVV 标准规范说明

SFU 类型、fp8 转换类型、th.vfncvt.rod.bf16.s 及 reduce 类型指令的特殊规定，在指令定义的地方已经给出，以下为这些指令遵循的 RISC-V 标准规范:

19.9.1　SFU 类型指令产生非法的情况

1. vstart 非 0。
2. enable vm 时，目的寄存器为 v0。
3. lmul=2 时，源和目的寄存器不是 2 的倍数。
4. lmul=4 时，源和目的寄存器不是 4 的倍数。
5. lmul=8 时，源和目的寄存器不是 8 的倍数。

19.9.2　FP8 和 th.vfncvt.rod.bf16.s narrow 类型指令产生非法的情况

1. vstart 非 0。
2. enable vm 的时候, 目的寄存器为 v0。
3. lmul=1 时，源寄存器不是 2 的倍数。
4. lmul=2 时，源寄存器不是 4 的倍数。
5. lmul=4 时，源寄存器不是 8 的倍数。
6. lmul=2 时，目的寄存器不是 2 的倍数。
7. lmul=4 时，目的寄存器不是 4 的倍数。
8. 目的寄存器不能和源寄存器高 group 重叠 (The destination EEW is smaller than the source EEW and the overlap is in the lowest-numbered part of the source register group.)

   e.g., when LMUL=1, vnsrl.wi v0, v0, 3 is legal, but a destination of v1 is not

19.9.3　FP8 wide 类型指令产生非法的情况

1. vstart 非 0。
2. enable vm 的时候, 目的寄存器为 v0。
3. lmul=1 时，目的寄存器不是 2 的倍数。
4. lmul=2 时，目的寄存器不是 4 的倍数。
5. lmul=4 时，目的寄存器不是 8 的倍数。
6. lmul=2 时，源寄存器不是 2 的倍数。
7. lmul=4 时，源的寄存器不是 4 的倍数。
8. lmul<1, 源和目的寄存器相同。
9. 源寄存器不能和目的寄存器低 group 重叠 (The destination EEW is greater than the source EEW, the source EMUL is at least 1, and the overlap is in the highest numbered part of the destination register group.)

   e.g., when LMUL=8, vzext.vf4 v0, v6 is legal, but a source of v0,v2, or v4 is not

19.9.4　Reduce 类型指令产生非法的情况

1. vstart 非 0。
2. lmul=2 时，源和目的寄存器不是 2 的倍数。
3. lmul=4 时，源和目的寄存器不是 4 的倍数。
4. lmul=8 时，源和目的寄存器不是 8 的倍数。

19.9.5　Reduce 类型指令特别补充说明

1. 目的寄存器不更新的元素，维持原值不变。
2. vary 类型指令，源和目的寄存器相同时，遵循指令执行顺序，目的寄存器更新为最新的数据。

19.10　附录 B-11 SCALAR 扩展指令术语

19.10.1　SCALAR 扩展指令编码

Wait for event for low power

31 26	25	24 20	19 15	14 12	11 7	6 0
000000	0	11100	00000	000	0	0001011

19.10.2　SCALAR 扩展指令描述

19.10.2.1　th.wfe

Synopsis:

Enter a low power state

Mnemonic:

th.wfe

Encoding:

31 26	25	24 20	19 15	14 12	11 7	6 0
000000	0	11100	00000	000	0	0001011

Description:

Enter a low power mode and wait for a wake-up event go out of low power mode.

Operation:

Execute wfe enters low power mode. go out of low power mode when happens one external level signal. in debug model, WFE is a NOP.

Permission:

M mode/S mode/U mode

Exceptions:

None

Supplement:

Scalar instruction

19.11　附录 B-12 玄铁协处理器扩展指令术语

为方便部分用户快速使用协处理器指令，玄铁 CPU 预设了部分用户可能常用的协处理器扩展指令，该部分指令已经在编译工具中进行了支持，指令编码使用 custom2 域。支持功能如下：

- 支持部分整型指令扩展
- 支持部分浮点类型指令扩展
- 支持多协处理器配置，最大支持 32 个协处理器，使用编码域中的 index 表示，index 信息使用协处理器接口中的 req_cop 信号域发送
- 指令中的立即数是使用协处理器接口中的 req_insn 信号域发送出去，用户需要从 req_insn 域自己进行获取

用户需要在玄铁协处理器接口上接收这些指令，并根据要求产生对应的结果，指令功能由用户自己定义。

备注

- 本章节协处理器指令仅作为示例，如果预定义指令不满足用户的需求，玄铁开放协处理器编码并提供工具，用户可以自定义协处理器指令编码使用和功能定义，定制符合自己需求的指令。
- 玄铁协处理器扩展指令编码低 7 bit 即 custom2 域，均为 [6:0]:1011011
- 玄铁协处理器扩展指令 reserved 域当作 0 处理；

详细指令描述如下各小节所述：

19.11.1　整型指令

19.11.1.1　cpx0

语法：

cpx0 index，rs1，uimm

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

说明：

source 来源 rs1、uimm，其中 uimm 数据共 10 bit，高 5 bit 是 uimm1，低 5 bit 是 uimm2，由两个编码拼接而成，该指令不产生回写

index 指示发往哪一个协处理器, 位宽为 2 bit，值可以为 0 ~ 3

指令格式：

31 30	29	28	27	26	25	24 20	19 15	14	13	12	11 7	6	5	4	3	2	1	0	
index	0	0	0	0	0	uimm1	rs1	0	0	0	uimm2	1	0	1	1	0	1	1	type1

19.11.1.2　cpx1

语法：

cpx1 index，rs1

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

说明：

source 来源 rs1，该指令不产生回写

index 指示发往哪一个协处理器, 位宽为 2 bit，值可以为 0 ~ 3

指令格式：

31 30	29	28	27	26	25	24 20	19 15	14	13	12	11 7	6	5	4	3	2	1	0	
index	0	0	0	0	1	reserved	rs1	0	0	0	reserved	1	0	1	1	0	1	1	type1

19.11.1.3　cpx2

语法：

cpx2 index，rd，rs1，uimm

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

说明：

source 来源 rs1，uimm，该指令需要回写 rd 寄存器

index 指示发往哪一个协处理器, 位宽为 2 bit，值可以为 0 ~ 3

指令格式：

31 30	29	28	27	26	25	24 20	19 15	14	13	12	11 7	6	5	4	3	2	1	0	
index	0	0	0	0	0	uimm	rs1	0	0	1	rd	1	0	1	1	0	1	1	type1

19.11.1.4　cpx3

语法：

cpx3 index，rd，rs1

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

说明：

source 来源 rs1，该指令需要回写 rd 寄存器

index 指示发往哪一个协处理器, 位宽为 2 bit，值可以为 0 ~ 3

指令格式：

31 30	29	28	27	26	25	24 20	19 15	14	13	12	11 7	6	5	4	3	2	1	0	
index	0	0	0	0	1	reserved	rs1	0	0	1	rd	1	0	1	1	0	1	1	type1

19.11.1.5　cpx4

语法：

cpx4 index，rs1，rs2，uimm

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

说明：

source 来源 rs1, rs2，uimm，该指令不产生回写

index 指示发往哪一个协处理器, 位宽为 2 bit，值可以为 0 ~ 3

指令格式：

31 30	29	28	27	26	25	24 20	19 15	14	13	12	11 7	6	5	4	3	2	1	0	
index	0	0	0	1	0	rs2	rs1	0	0	0	uimm	1	0	1	1	0	1	1	type1

19.11.1.6　cpx5

语法：

cpx5 index，rs1，rs2

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

说明：

source 来源 rs1, rs2，该指令不产生回写

index 指示发往哪一个协处理器, 位宽为 2 bit，值可以为 0 ~ 3

指令格式：

31 30	29	28	27	26	25	24 20	19 15	14	13	12	11 7	6	5	4	3	2	1	0	
index	0	0	0	1	1	rs2	rs1	0	0	0	reserved	1	0	1	1	0	1	1	type1

19.11.1.7　cpx6

语法：

cpx6 index，rd，rs1，rs2

执行权限：

M mode/S mode/U mode

异常：

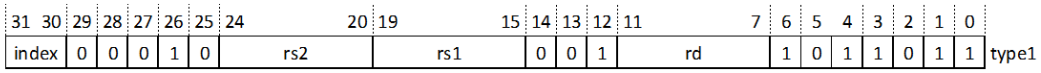
非法指令异常

说明：

source 来源 rs1, rs2，该指令需要回写 rd 寄存器

index 指示发往哪一个协处理器, 位宽为 2 bit，值可以为 0 ~ 3

指令格式：



19.11.1.8　cpx7

语法：

cpx7 index，rd，rs1，rs2

执行权限：

M mode/S mode/U mode

异常：

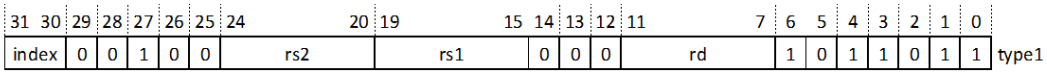
非法指令异常

说明：

source 来源 rs1, rs2, rd，该指令不产生回写

index 指示发往哪一个协处理器, 位宽为 2 bit，值可以为 0 ~ 3

指令格式：



19.11.1.9　cpx8

语法：

cpx8 index，rd，rs1，rs2

执行权限：

M mode/S mode/U mode

异常：

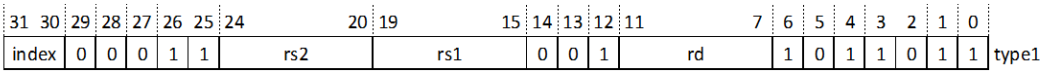
非法指令异常

说明：

source 来源 rs1，rs2，rd，该指令同时需要回写 rd 寄存器，rd 既是 source 又是目的寄存器

index 指示发往哪一个协处理器, 位宽为 2 bit，值可以为 0 ~ 3

指令格式：



19.11.1.10　cpx9

语法：

cpx9 index，rd，rs1，uimm

执行权限：

M mode/S mode/U mode

异常：

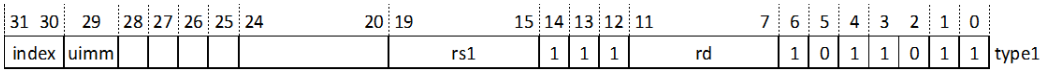
非法指令异常

说明：

source 来源 rs1，rd，uimm，该指令同时需要回写 rd 寄存器，rd 既是 source 又是目的寄存器

index 指示发往哪一个协处理器, 位宽为 2 bit，值可以为 0 ~ 3

指令格式：



19.11.1.11　cpx10

语法：

cpx10 index，rd，uimm

执行权限：

M mode/S mode/U mode

异常：

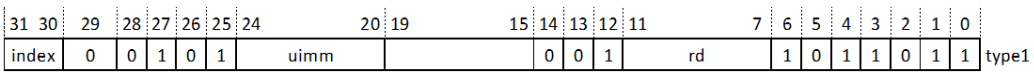
非法指令异常

说明：

source 来源 uimm，该指令同时需要回写 rd 寄存器

index 指示发往哪一个协处理器, 位宽为 2 bit，值可以为 0 ~ 3

指令格式：



19.11.2　浮点类型指令

浮点 rm 信息通过玄铁协处理器接口 req_hint 域伴随指令发出。

19.11.2.1　fcpx0

语法：

fcpx0 index，fs1

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

说明：

source 来源 fs1，该指令不需要回写

index 指示发往哪一个协处理器, 位宽为 2 bit，值可以为 0 ~ 3

该指令在 fs 打开情况下才可以执行，否则非法

指令格式：

31 30	29	28	27	26	25	24 20	19 15	14	13	12	11 7	6	5	4	3	2	1	0	
index	0	0	0	0	0	reserved	fs1	1	0	0	reserved	1	0	1	1	0	1	1	type5

19.11.2.2　fcpx1

语法：

fcpx1 index，fd，fs1

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

说明：

source 来源 fs1，该指令需要回写 fd 寄存器

index 指示发往哪一个协处理器, 位宽为 2 bit，值可以为 0 ~ 3

该指令在 fs 打开情况下才可以执行，否则非法

指令格式：

31 30	29	28	27	26	25	24 20	19 15	14	13	12	11 7	6	5	4	3	2	1	0	
index	0	0	0	0	0	reserved	fs1	1	0	1	fd	1	0	1	1	0	1	1	type5

19.11.2.3　fcpx2

语法：

fcpx2 index，fs1, fs2

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

说明：

source 来源 fs1，fs2，该指令不需要回写

index 指示发往哪一个协处理器, 位宽为 2 bit，值可以为 0- ~ 3

该指令在 fs 打开情况下才可以执行，否则非法

指令格式：

31 30	29	28	27	26	25	24 20	19 15	14	13	12	11 7	6	5	4	3	2	1	0	
index	0	0	0	0	1	fs2	fs1	1	0	0	reserved	1	0	1	1	0	1	1	type5

19.11.2.4　fcpx3

语法：

fcpx3 index，fd，fs1, fs2

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

说明：

source 来源 fs1, fs2，该指令需要回写 fd 寄存器

index 指示发往哪一个协处理器, 位宽为 2 bit，值可以为 0 ~ 3

该指令在 fs 打开情况下才可以执行，否则非法

指令格式：

31 30	29	28	27	26	25	24 20	19 15	14	13	12	11 7	6	5	4	3	2	1	0	
index	0	0	0	0	1	fs2	fs1	1	0	1	fd	1	0	1	1	0	1	1	type5

19.11.2.5　fcpx4

语法：

fcpx4 index，fd，fs1, fs2

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

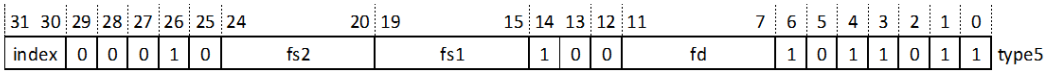
说明：

source 来源 fs1，fs2，fd，该指令不需要回写

index 指示发往哪一个协处理器, 位宽为 2 bit，值可以为 0 ~ 3

该指令在 fs 打开情况下才可以执行，否则非法

指令格式：



19.11.2.6　fcpx5

语法：

fcpx5 index，fd，fs1, fs2

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

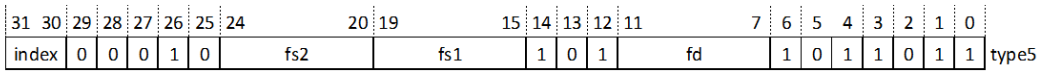
说明：

source 来源 fs1, fs2，fd，该指令需要回写 fd 寄存器

index 指示发往哪一个协处理器, 位宽为 2 bit，值可以为 0 ~ 3

该指令在 fs 打开情况下才可以执行，否则非法

指令格式：



19.11.2.7　fcpx6

语法：

fcpx6 index，fd，fs2, uimm

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

说明：

source 来源 fs2, uimm source，该指令需要回写 fd 寄存器

index 指示发往哪一个协处理器, 位宽为 2 bit，值可以为 0 ~ 3

该指令在 fs 打开情况下才可以执行，否则非法

指令格式：

31 30	29	28	27	26	25	24 20	19 15	14	13	12	11 7	6	5	4	3	2	1	0	
index	0	0	0	1	1	fs2	uimm	1	0	1	fd	1	0	1	1	0	1	1	type5

20 附录 C 控制寄存器

本附录对机器模式控制寄存器、超级用户模式控制寄存器和用户模式控制寄存器进行详细说明。

20.1　附录 C-1 机器模式控制寄存器

机器模式控制寄存器按照功能分为：机器模式信息寄存器组、机器模式异常配置寄存器组、机器模式异常处理寄存器组、机器模式内存保护寄存器组、机器模式计数器和机器模式计数器配置寄存器组。

20.1.1　机器模式信息寄存器组

20.1.1.1　机器模式供应商编号寄存器（MVENDORID）

机器模式供应商编号寄存器（MVENDORID）存储了玄铁 CPU 的厂商编号信息，目前该寄存器绑定为 0x5B7。

该寄存器的位长是 64 位，寄存器的读写权限是机器模式只读，即非机器模式访问和机器模式写操作访问都会导致非法指令异常。

20.1.1.2　机器模式架构编号寄存器（MARCHID）

机器模式架构编号寄存器（MARCHID）存储了处理器核的架构编号，用于存放玄铁 CPU 产品的内部编号，其复位值由产品本身决定。

该寄存器的位长是 64 位，寄存器的读写权限是机器模式只读，即非机器模式访问和机器模式写操作访问都会导致非法指令异常。

20.1.1.3　机器模式硬件实现编号寄存器（MIMPID）

机器模式硬件实现编号寄存器（MIMPID）存储了处理器核的硬件实现编号。C908X 目前未实现该寄存器，该寄存器存储了只读非零值。

该寄存器的位长是 64 位，寄存器的读写权限是机器模式只读，即非机器模式访问和机器模式写操作访问都会导致非法指令异常。

20.1.1.4　机器模式逻辑内核编号寄存器（MHARTID）

机器模式逻辑内核编号寄存器（MHARTID）存储了处理器核的硬件逻辑内核编号。

该寄存器的位长是 64 位，寄存器的读写权限是机器模式只读，即非机器模式访问和机器模式写操作访问都会导致非法指令异常。

20.1.1.5　机器模式配置数据结构指针（MCONFIGPTR）

机器模式配置数据结构指针（MCONFIGPTR）存储了配置数据结构对应的物理地址。当该寄存器值为 0 时，表示不存在这样的数据结构。

该寄存器的位长是 64 位，寄存器的读写权限是机器模式只读，即非机器模式访问和机器模式写操作访问都会导致非法指令异常。

20.1.2　机器模式异常配置寄存器组

20.1.2.1　机器模式处理器状态寄存器（MSTATUS）

机器模式处理器状态寄存器（MSTATUS）存储了处理器在机器模式下的状态和控制信息，包括全局中断有效位、异常保留中断有效位、异常保留特权模式位等。

该寄存器的位长是 64 位，寄存器的读写权限是机器模式可读写，即非机器模式访问都会导致非法指令异常。

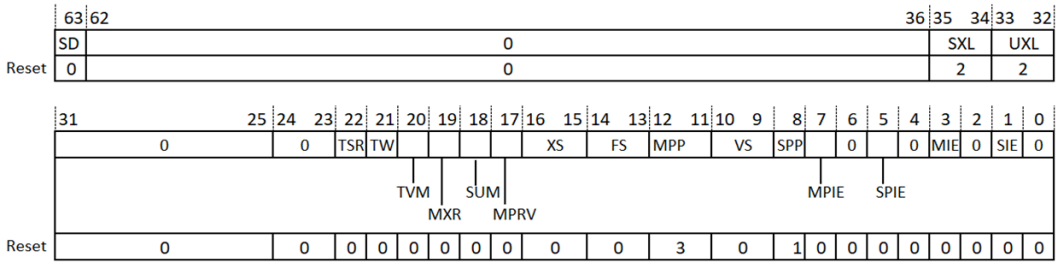


图 20.1: 机器模式处理器状态寄存器（MSTATUS）

SIE-超级用户模式中断使能位：

- 当 SIE 为 0 时，超级用户中断无效；
- 当 SIE 为 1 时，超级用户中断有效；

  该位会被 reset 清零，处理器被降级到超级用户模式响应中断时被清零；在处理器退出中断服务程序时被置为 SPIE 的值。

MIE-机器模式中断使能位：

- 当 MIE 为 0 时，机器模式中断无效；

- 当 MIE 为 1 时，机器模式中断有效；

  该位会被 reset 清零，处理器在机器模式响应中断时被清零；在处理器退出中断服务程序时被置为 MPIE 的值。

SPIE-超级用户模式保留中断使能位：

该位用于保存处理器在超级用户模式响应中断前 SIE 位的值。

该位会被 reset 清零，在处理器退出中断服务程序时被置 1。

MPIE-机器模式保留中断使能位：

该位用于保存处理器在机器模式响应中断前 MIE 位的值。

该位会被 reset 清零，在处理器退出中断异常服务程序时被置 1。

SPP-超级用户模式保留特权状态位：

该位用于保存处理器在超级用户模式进入异常服务程序前的特权状态。

- 当 SPP 为 2'b00 时，表示处理器进入异常服务程序前处于用户模式；
- 当 SPP 为 2'b01 时，表示处理器进入异常服务程序前处于超级用户模式；

  该位会被 reset 置 2'b01。

MPP-机器模式保留特权状态位：

该位用于保存处理器在机器模式进入异常服务程序前的特权状态。

- 当 MPP 为 2'b00 时，表示处理器进入异常服务程序前处于用户模式；
- 当 MPP 为 2'b01 时，表示处理器进入异常服务程序前处于超级用户模式；
- 当 MPP 为 2'b11 时，表示处理器进入异常服务程序前处于机器模式；

  该位会被 reset 置 2'b11。

FS-浮点单元状态位：

根据浮点状态位，可以判断上下文切换的时候，是否需要保存浮点相关寄存器。

- 当 FS 为 2'b00 时，浮点单元处于关闭状态，此时访问浮点相关寄存器会产生异常。
- 当 FS 为 2'b01 时，浮点单元处于初始化状态。
- 当 FS 为 2'b10 时，浮点单元处于 clean 态。
- 当 FS 为 2'b11 时，浮点单元处于 dirty 态，表明浮点寄存器和控制寄存器被修改过。

XS-扩展单元状态位：

C908X 没有扩展单元，固定为 0。

MPRV-修改特权模式：

- 当 MPRV=1 时，加载和存储请求执行时根据 MPP 中的特权态进行执行。
- 当 MPRV=0 时，加载和存储请求执行时根据当前处理器所处特权模式进行执行。

SUM-允许超级用户模式下访问 U 态虚拟内存空间

- 当 SUM=1 时，超级用户模式下，加载、存储请求可以访问标记为用户态的虚拟内存空间。

- 当 SUM=0 时，超级用户模式下，加载、存储和取指请求不可以访问标记为用户态的虚拟内存空间。

MXR-允许加载请求访问标记为可执行的内存空间

- 当 MXR=1 时，允许加载请求访问标记为可执行或者可读的虚拟内存空间。
- 当 MXR=0 时，允许加载请求只能访问标记为可读的虚拟内存空间。

TVM-陷阱虚拟内存

- 当 TVM=1 时，超级用户模式读写 satp 控制寄存器以及执行 sfence 指令，产生非法指令异常。
- 当 TVM=0 时，超级用户模式可以读写 satp 控制寄存器以及执行 sfence 指令。

TW-超时等待

- 当 TW=1 时，超级用户模式执行低功耗指令 wfi，产生非法指令异常。
- 当 TW=0 时，超级用户模式执行低功耗指令 wfi。

TSR-陷阱 sret

- 当 TSR=1 时，超级用户模式执行 sret 指令，产生非法指令异常。
- 当 TSR=0 时，允许超级用户模式执行 sret 指令。

VS-矢量单元状态位

根据矢量状态位，可以判断上下文切换的时候，是否需要保存矢量相关寄存器。

- 当 VS 为 2'b00 时，矢量单元处于关闭状态，此时访问矢量相关寄存器会产生异常。
- 当 VS 为 2'b01 时，矢量单元处于初始化状态。
- 当 VS 为 2'b10 时，矢量单元处于 clean 态。
- 当 VS 为 2'b11 时，矢量单元处于 dirty 态，表明矢量寄存器和矢量控制寄存器被修改过。

VS 位仅当配置矢量执行单元时有效，不配置时恒为 0。

UXL-寄存器位宽

只读，固定值是 2，表示在 U 态下，寄存器的位宽是 64bit。

SXL-寄存器位宽

只读，固定值是 2，表示在 S 态下，寄存器的位宽是 64bit。

SD-浮点、矢量和扩展单元 dirty 状态总和位

- 当 SD=1 时，表明浮点或矢量或扩展单元处在 dirty 状态。
- 当 SD=0 时，表明浮点和矢量和扩展单元处都不处在 dirty 状态。

20.1.2.2　机器模式处理器指令集特性寄存器（MISA）

机器模式处理器指令集特性寄存器（MISA）存储了处理器所支持的指令集架构特性。

该寄存器的位长是 64 位，寄存器的读写权限是机器模式可读写，即非机器模式访问都会导致非法指令异常。

C908X 支持的指令集架构为 RV64GC，对应的 MISA 寄存器复位值为 0x800000000094112d。具体的赋值规则请参考 RISCV 官方文档《riscv-privileged》。

C908X 不支持动态配置 MISA 寄存器，对该寄存器进行写操作不产生任何效果。

20.1.2.3　机器模式异常降级控制寄存器（MEDELEG）

机器模式异常降级控制寄存器（MEDELEG）可以将超级用户和用户模式发生的异常降级到超级用户模式响应。MEDELEG 低 16 比特和异常向量表一一对应，可以选择将哪些异常降级到超级用户模式响应。

该寄存器的位长是 64 位，寄存器的读写权限是机器模式可读写，即非机器模式访问都会导致非法指令异常。

20.1.2.4　机器模式中断降级控制寄存器（MIDELEG）

机器模式中断降级控制寄存器（MIDELEG）可以将超级用户模式中断降级到超级用户模式响应。

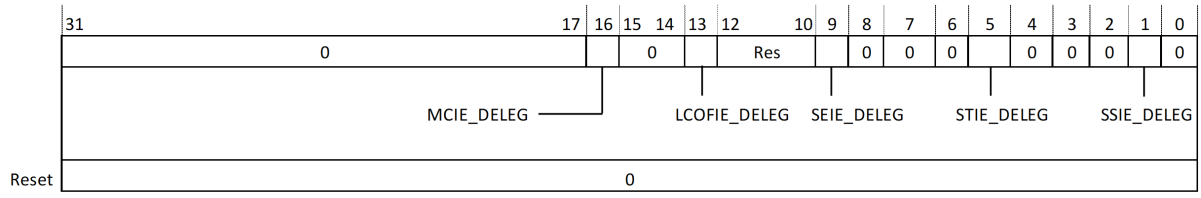


图 20.2: 机器模式中断降级控制寄存器（MIDELEG）

该寄存器的位长是 64 位，寄存器的读写权限是机器模式可读写，即非机器模式访问都会导致非法指令异常。

20.1.2.5　机器模式中断使能控制寄存器（MIE）

机器模式中断使能控制寄存器（MIE）用于控制不同中断类型的使能和屏蔽。该寄存器的位长是 64 位，寄存器的读写权限是机器模式可读写，即非机器模式访问都会导致非法指令异常。

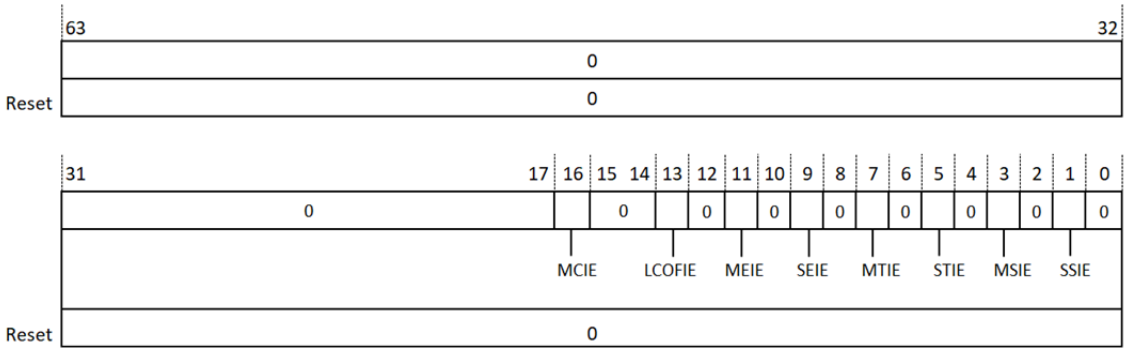


图 20.3: 机器模式中断使能控制寄存器（MIE）

SSIE-超级用户模式软件中断使能位：

- 当 SEIE 为 0 时，超级用户模式软件外部中断无效。
- 当 SEIE 为 1 时，超级用户模式软件外部中断有效。

MSIE-机器模式软件中断使能位：

- 当 MSIE 为 0 时，机器模式软件中断无效。
- 当 MSIE 为 1 时，机器模式软件中断有效。

STIE-超级用户模式计时器中断使能位：

- 当 STIE 为 0 时，超级用户模式计时器中断无效。
- 当 STIE 为 1 时，超级用户模式计时器外部中断有效。

MTIE-机器模式计时器中断使能位：

- 当 MTIE 为 0 时，机器模式计时器中断无效。
- 当 MTIE 为 1 时，机器模式计时器中断有效。

SEIE-超级用户模式外部中断使能位：

- 当 SEIE 为 0 时，超级用户模式外部中断无效。
- 当 SEIE 为 1 时，超级用户模式外部中断有效。

MEIE-机器模式外部中断使能位：

- 当 MEIE 为 0 时，机器模式外部中断无效。
- 当 MEIE 为 1 时，机器模式外部中断有效。

LCOFIE-机器模式事件计数器溢出中断使能位：

- 当 LCOFIE 为 0 时，机器模式计数器溢出中断无效。
- 当 LCOFIE 为 1 时，机器模式计数器溢出中断有效。

MCIE-机器模式 ECC 和 BUS ERROR 中断使能位

- 当 MCIE 为 0 时，机器模式 ECC 和 BUS ERROR 中断无效。
- 当 MCIE 为 1 时，机器模式 ECC 和 BUS ERROR 中断有效。

20.1.2.6　机器模式向量基址寄存器（MTVEC）

机器模式向量基址寄存器（MTVEC）用于配置异常服务程序的入口地址。

该寄存器的位长是 64 位，寄存器的读写权限是机器模式可读写，即非机器模式访问都会导致非法指令异常。

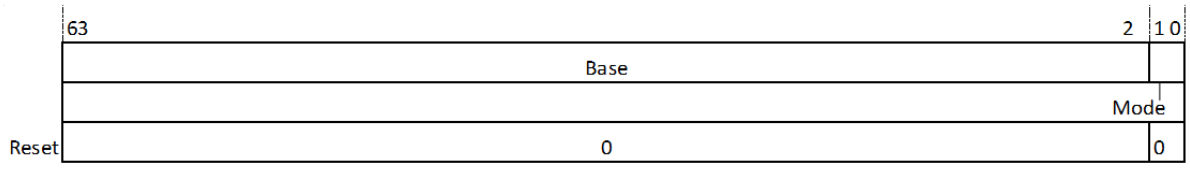


图 20.4: 机器模式向量基址寄存器（MTVEC）

BASE-向量基址位：

向量基址位指示了异常服务程序入口地址的高 62 位，将此基址拼接 2'b00 即可得到异常服务程序入口地址。

该位会被 reset 清零。

MODE-向量入口模式位：

- 当 MODE[1:0] 为 2'b00 时，异常和中断都统一使用 BASE 地址作为异常入口地址。
- 当 MODE[1:0] 为 2'b01 时，异常使用 BASE 地址作为入口地址，中断使用 BASE + 4*cause。

注意

- 在 SV39 的情况下，bit38 为符号位，mtvec 和 stvec 的高位需要按照 bit38 进行符号位拓展。
- 在 SV48 的情况下，bit47 为符号位，mtvec 和 stvec 的高位需要按照 bit47 进行符合位拓展。

20.1.2.7　机器模式计数器访问授权寄存器（MCOUNTEREN）

机器模式计数器访问授权寄存器（mcounteren），用于授权超级用户模式是否可以访问用户模式计数器。

具体请参考*机器模式计数器访问授权寄存器（mcounteren）*。

20.1.3　机器模式异常处理寄存器组

20.1.3.1　机器模式异常临时数据备份寄存器（MSCRATCH）

机器模式异常临时数据备份寄存器（MSCRATCH）用于处理器在异常服务程序中备份临时数据。一般用来存储机器模式本地上下文空间的入口指针值。

该寄存器的位长是 64 位，寄存器的读写权限是机器模式可读写，即非机器模式访问都会导致非法指令异常。

20.1.3.2　机器模式异常保留程序计数器寄存器（MEPC）

机器模式异常保留程序计数器（MEPC）用于存储程序从异常服务程序退出时的程序计数器值（即 PC 值）。C908X 支持 16 位宽指令，MEPC 的值以 16 位宽对齐，最低位为零。

该寄存器的位长是 64 位，寄存器的读写权限是机器模式可读写，即非机器模式访问都会导致非法指令异常。

20.1.3.3　机器模式异常事件向量寄存器（MCAUSE）

机器模式异常事件向量寄存器（MCAUSE）用于保存触发异常的异常事件向量号，用于在异常服务程序中处理对应事件。

该寄存器的位长是 64 位，寄存器的读写权限是机器模式可读写，即非机器模式访问都会导致非法指令异常。

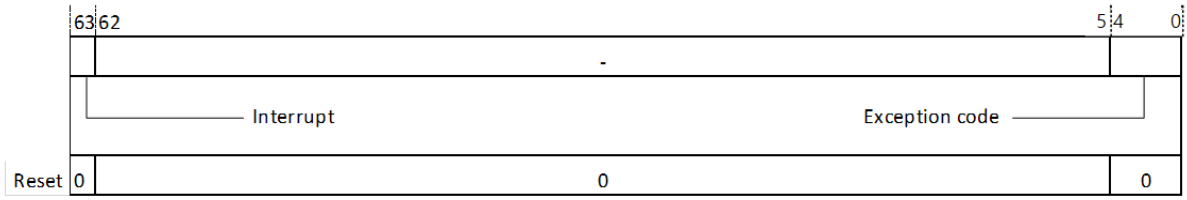


图 20.5: 机器模式异常事件向量寄存器（MCAUSE）

Interrupt-中断标记位：

- 当 Interrupt 位为 0 时，表示触发异常的来源不是中断，Exception Code 按照异常解析。
- 当 Interrupt 位为 1 时，表示触发异常的来源是中断，Exception Code 按照中断解析。

Exception Code-异常向量号位：

在处理器进入异常时，异常向量位会被更新为异常来源的对应值。

20.1.3.4　机器模式中断等待状态寄存器（MIP）

机器模式中断等待状态寄存器（MIP）用于保存处理器的中断等待状态。当处理器出现中断无法立即响应的情况时，MIP 寄存器中的对应位会被置位。

写 CLINT 中的 MSIP 和 SSIP 寄存器可以出发对应的中断，中断有效后可以通过 MIP 中对应的 bit 为 MSIP bit 以及 SSIP bit 进行查询。

该寄存器的位长是 64 位，寄存器的读写权限是机器模式可读写，即非机器模式访问都会导致非法指令异常。

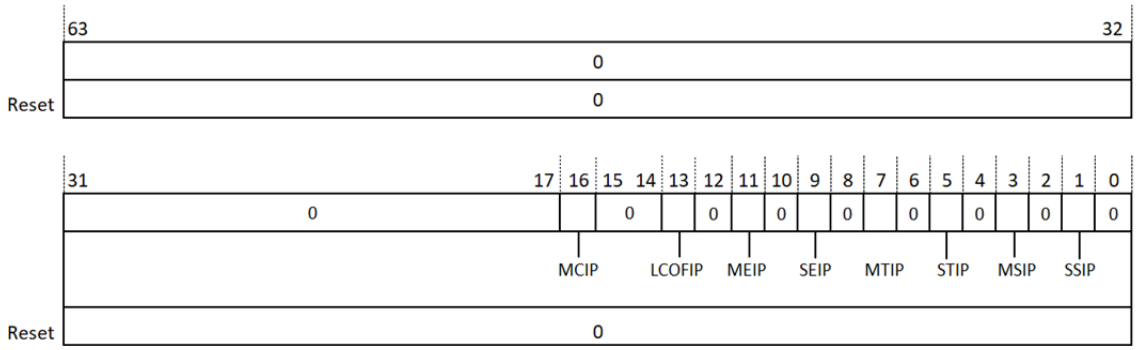


图 20.6: 机器模式中断等待状态寄存器（MIP）

SSIP-超级用户模式软件中断等待位：

- 当 SSIP 为 0 时，处理器当前没有处于等待状态的超级用户模式软件中断。
- 当 SSIP 为 1 时，处理器当前有处于等待状态的超级用户模式软件中断。

M 态可读写，SSIP delegate 到 S 态之后，S 态可读写，否则 S 态只读。

MSIP-机器模式软件中断等待位：

- 当 MSIP 为 0 时，处理器当前没有处于等待状态的机器模式软件中断。
- 当 MSIP 为 1 时，处理器当前有处于等待状态的机器模式软件中断。

此 bit 为只读。

STIP-超级用户模式计时器中断等待位：

- 当 STIP 为 0 时，处理器当前没有处于等待状态的超级用户模式计时器中断。
- 当 STIP 为 1 时，处理器当前有处于等待状态的超级用户模式计时器中断。

MTIP-机器模式计时器中断等待位：

- 当 MTIP 为 0 时，处理器当前没有处于等待状态的机器模式计时器中断。
- 当 MTIP 为 1 时，处理器当前有处于等待状态的机器模式计时器中断。

SEIP-超级用户模式外部中断等待位：

- 当 SEIP 为 0 时，处理器当前没有处于等待状态的超级用户模式外部中断。
- 当 SEIP 为 1 时，处理器当前有处于等待状态的超级用户模式外部中断。

MEIP-外部中断等待位：

- 当 MEIP 为 0 时，处理器当前没有处于等待状态的机器模式外部中断。
- 当 MEIP 为 1 时，处理器当前有处于等待状态的机器模式外部中断。

LCOFIP-机器模式事件计数器溢出中断等待位：

- 当 LCOFIP 为 0 时，处理器当前没有处于等待状态的机器模式计数器溢出中断。
- 当 LCOFIP 为 1 时，处理器当前有处于等待状态的机器模式计数器溢出中断。

MCIP-机器模式 ECC 和 BUS ERROR 中断等待位

- 当 MCIP 为 0 时，处理器当前没有处于等待状态的机器模式 ECC 和 BUS ERROR 中断。
- 当 MCIP 为 1 时，处理器当前有处于等待状态的机器模式 ECC 和 BUS ERROR 中断。

20.1.4　机器模式配置寄存器组

20.1.4.1　机器模式环境配置寄存器（MENVCFG）

机器模式环境配置寄存器（MENVCFG）用于控制低于机器模式时执行环境中的特性。

该寄存器的位长是 64 位，寄存器的读写权限是机器模式可读写，即非机器模式访问会导致非法指令异常。

	63	62	61 … 8	7	6	5 … 4	3 … 1	0
	STCE	PBMTE	0	CBZE	CBCFE	CBIE	0	FIOM
Reset	0	0	0	0	0	0	0	0

图 20.7: 机器模式环境配置寄存器（MENVCFG）

STCE - 超级用户模式计时器中断比较值寄存器（STIMECMP）功能使能位

- 当 STCE 为 0 时，CLINT 中断控制器中的超级用户模式计时器中断比较值寄存器高位/低位（STIMECMPH/STIMECMPL）小于或等于系统计时器时，产生超级用户模式计时器中断。STIMECMPH/STIMECMPL 是内存地址映射寄存器;
- 当 STCE 为 1 时，超级用户模式计时器中断比较值寄存器（STIMECMP）小于或等于系统计时器时，产生超级用户模式计时器中断。STIMECMP 是 CSR 寄存器。

PBMTE - Svpbmt 使能位

- 当 PBMTE=0 时，Svpbmt 功能关闭；
- 当 PBMTE=1 时，Svpbmt 功能开启。

CBZE - CBO.ZERO 指令使能位

- 当 CBZE=0 时，在更低特权模式下执行 CBO.ZERO 指令产生非法指令异常;
- 当 CBZE=1 时，在更低特权模式下 CBO.ZERO 指令正常执行。

CBCFE - CBO.CLEAN 和 CBO.FLUSH 指令使能位

- 当 CBCFE=0 时，在更低特权模式下执行 CBO.CLEAN 和 CBO.FLUSH 指令产生非法指令异常;
- 当 CBCFE=1 时，在更低特权模式下 CBO.CLEAN 和 CBO.FLUSH 指令正常执行。

CBIE - CBO.INVAL 指令使能位

- 当 CBIE=0 时，在更低特权模式下执行 CBO.INVAL 指令产生非法指令异常;
- 当 CBIE=1 时，在更低特权模式下 CBO.INVAL 指令按照 CBO.FLUSH 指令执行;
- 当 CBIE=2 时，保留值，不应当被配置;

- 当 CBIE=3 时，在更低特权模式下 CBO.INVAL 指令正常执行。

FIOM - IO fence 包含内存访问

对于 RV 标准的定义来说，menvcfg.FIOM = 1 时，低于机器模式的 IO 同步请求需要同时包括内存 RW 同步。

注：玄铁 C908X 中所有对 IO 的 fence 同步均会包括 RW 同步，所以不再受该位进行控制。无论该位为何值，对 IO 的 fence 同步均会包括 RW 同步。

20.1.4.2　机器模式安全配置寄存器（MSECCFG）

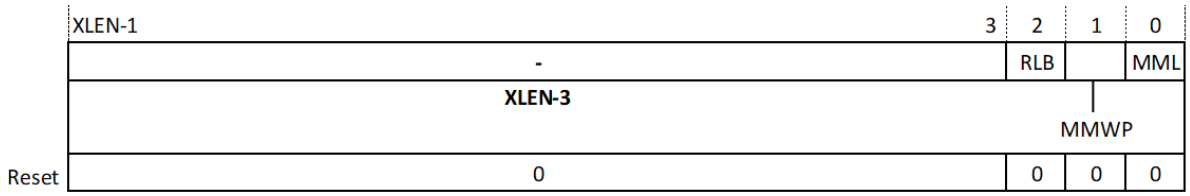


图 20.8: 机器模式安全配置寄存器（MSECCFG）

机器模式安全配置寄存器（MSECCFG）用于记录执行安全配置信息，扩展 PMP 的权限规则，实现机器模式和超级用户模式/普通用户模式之间的内存访问保护（MAP）和内存执行保护（MEP）。该寄存器只允许机器模式访问。

RLB-Rule Locking Bypass

- 当 RLB = 1 时，pmpcfg.L 位允许被编辑，也就是被锁定的 PMP 表项在 RLB = 1 时，仍然可以取消锁定，表项规则允许被重新修改。
- 当 RLB = 0 时，如果 PMP 的表项中存在任何被锁定的表项（pmpcfg.L = 1），则 mseccfg.RLB 位也会被锁定，该位将不可修改，只能通过硬件复位到初始值。

MMWP-机器模式白名单策略（Machine Mode Whitelist Policy）

MMWP 位一旦被置 1，该位将被锁定不可修改，只能通过硬件复位到初始值。

当 MMWP = 1 时，机器模式只允许按照 PMP 配置表里的规则进行访问，不在 PMP 配置表规则里的访问都将被拒绝。

MML - 机器模式锁定（Machine Mode Lockdown）

MML 位一旦被置 1，该位将被锁定不可修改，只能通过硬件复位到初始值。

MML = 1 时，PMP 表项的访问规则定义如下 表 20.1 所示:

表 20.1: mseccfg.MML=1 时 PMP 表项的权限规则

Bits on pmpcfg register				Result	
L	R	W	X	M Mode	S/U Mode
0	0	0	0	Inaccessible region (Access Exception)	
0	0	0	1	Access Exception	Execute-only region
0	0	1	0	Shared data region: Read/write on M mode, read-only on S/U mode	
0	0	1	1	Shared data region: Read/write for both M and S/U mode	
0	1	0	0	Access Exception	Read-only region
0	1	0	1	Access Exception	Read/Execute region
0	1	1	0	Access Exception	Read/Write region
0	1	1	1	Access Exception	Read/ Write/Execute region
1	0	0	0	Locked inaccessible region* (Access Exception)	
1	0	0	1	Locked Execute-only region*	Access Exception
1	0	1	0	Locked Shared code region: Execute only on both M and S/U mode.*	
1	0	1	1	Locked Shared code region: Execute only on S/U mode, read/execute on M mode.*	
1	1	0	0	Locked Read-only region*	Access Exception
1	1	0	1	Locked Read/Execute region*	Access Exception
1	1	1	0	Locked Read/Write region*	Access Exception
1	1	1	1	Locked Shared data region: Read only on both M and S/U mode.*	

*: Locked entries cannot be removed or modified until a hard reset, unless mseccfg.RLB is set.

具体信息请参考 RISC-V PMP Enhancements for memory access and execution prevention on Machine mode (Smepmp) 。

20.1.5　机器模式内存保护寄存器组

机器模式内存保护寄存器组是和内存保护单元设置相关的控制寄存器。

20.1.5.1　机器模式物理内存保护配置寄存器（PMPCFG）

机器模式物理内存保护配置寄存器（PMPCFG）用于配置物理内存的访问权限、地址匹配模式。

该寄存器的位长是 64 位，寄存器的读写权限是机器模式可读写，即非机器模式访问都会导致非法指令异常。

具体信息请参考*物理内存保护设置寄存器（PMPCFG）* 。

20.1.5.2　机器模式物理内存地址寄存器（PMPADDR）

机器模式物理内存地址寄存器（PMPADDR）用于配置物理内存的每个表项的地址区间。

该寄存器的位长是 64 位，寄存器的读写权限是机器模式可读写，即非机器模式访问都会导致非法指令异常。

具体信息请参考*物理内存保护地址寄存器（PMPADDR）*。

20.1.6　机器模式计数器寄存器组

机器模式计数器寄存器组属于性能监测单元，用于统计程序运行中的软件信息和部分硬件信息，供软件开发人员进行程序优化。

20.1.6.1　机器模式周期计数器（MCYCLE）

机器模式周期计数器（MCYCLE）用于存储处理器已经执行的周期数，当处理器处于执行状态（即非低功耗状态）下，MCYCLE 寄存器就会在每个执行周期自增计数。

周期计数器为 64 位，周期计数器会被 reset 清零。

具体信息请参考*事件计数器*。

20.1.6.2　机器模式退休指令计数器（MINSTRET）

机器模式退休指令计数器（MINSTRET）用于存储处理器已经退休的指令数，MINSTRET 寄存器会在每条指令退休时自增计数。

退休指令计数器为 64 位，退休计数器会被 reset 清零。

具体信息请参考*事件计数器*。

20.1.6.3　机器模式事件计数器（MHPMCOUNTERn）

机器模式事件计数器（MHPMCOUNTERn）用于对事件进行计数。

事件计数器为 64 位，事件计数器会被 reset 清零。

具体信息请参考*事件计数器*。

20.1.7　机器模式计数器配置寄存器组

机器模式计数器配置寄存器用于给机器模式事件计数器选择计数的事件。

20.1.7.1　机器模式事件选择器（MHPMEVENTn）

机器模式性能监测事件选择寄存器（mhpmevent3-31），用于选择每个计数器对应的计数事件。C908X 中每个计数器可以配置任意一个事件。将事件索引值写入性能监测事件选择寄存器，该计数器即可对配置的事件正常计数。

具体信息请参考*性能监测事件选择寄存器*。

20.1.8　机器模式处理器控制和状态扩展寄存器组

20.1.8.1　机器模式扩展状态寄存器（MXSTATUS）

机器模式扩展状态寄存器（MXSTATUS）存储了处理器当前所处特权模式和 C908X 扩展功能开关位。

该寄存器的位长是 64 位，寄存器的读写权限是机器模式可读写，即非机器模式访问都会导致非法指令异常。

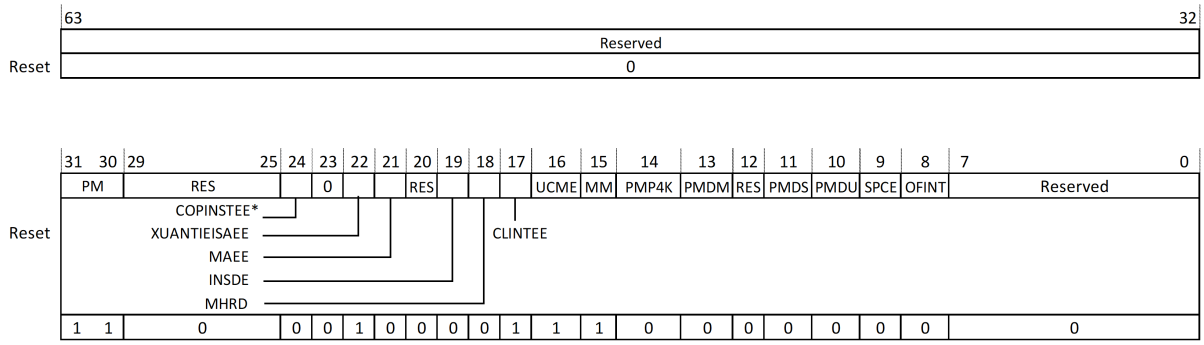


图 20.9: 机器模式扩展状态寄存器（MXSTATUS）

OFINT-HPCP cycle、instret 事件溢出中断使能:

当 OFINT=1'b0 时，表征 cycle、instret 事件不能产生溢出中断。

当 OFINT=1'b1 时，表征 cycle、instret 事件可以产生溢出中断。

SPCE-S 态 Cache Partition 控制使能：

当 SPCE=1'b0 时，表征 S 态不能配置 last level cache 的 partition。

当 SPCE=1'b1 时，表征 S 态可以配置 last level cache 的 partition。

PMDU-用户模式性能监测计数使能位：

当 PMDU 为 0 时，用户模式下允许性能计数器计数。

当 PMDU 为 1 时，用户模式下禁止性能计数器计数。

PMDS-超级用户模式性能监测计数使能位：

当 PMDS 为 0 时，超级用户模式下允许性能计数器计数。

当 PMDS 为 1 时，超级用户模式下禁止性能计数器计数。

PMDM-机器模式性能监测计数使能位：

当 PMDM 为 0 时，机器模式下允许性能计数器计数。

当 PMDM 为 1 时，机器模式下禁止性能计数器计数。

PMP4K-PMP 最小粒度控制位：

C908X 当前只支持 PMP 最小粒度为 4K，不受该位影响。

MM-非对齐访问使能位：

当 MM 为 0 时，不支持非对齐访问，非对齐访问将产生非对齐异常。

当 MM 为 1 时，且只有在 weak order（SO=0）区域支持非对齐访问，硬件处理非对齐访问。在 MM 设置为 1 时，如果对 strong order 区域发起非对齐访问，则会报访问错误异常，而不是报非对齐访问异常。（C908X 默认值为 1）

UCME-U 态执行扩展 cache 指令：

当 UCME 为 0 时，用户模式不能执行扩展的 cache 操作指令，产生非法指令异常。

当 UCME 为 1 时，用户模式可以执行扩展的 cache 操作指令。

CLINTEE-Clint 计时器/软件中断超级用户扩展使能位：

当 CLINTEE 为 0 时，CLINT 发起的超级用户软件中断和计时器中断不会被响应。

当 CLINTEE 为 1 时，CLINT 发起的超级用户软件中断和计时器中断可以被响应。

MHRD-关闭硬件回填：

当 MHRD 为 0 时，TLB 缺失后，硬件会进行硬件回填。

当 MHRD 为 1 时，TLB 缺失后，硬件不会进行硬件回填。

INSDE-关闭 Icache snoop Dcache 功能：

当 INSDE 为 0 时，Icache 缺失后，会通过 CIU 去 snoop Dcache。

当 INSDE 为 1 时，Icache 缺失后，不会通过 CIU 去 snoop Dcache。

MAEE-扩展 MMU 地址属性：

当 MAEE 为 0 时，不扩展 MMU 地址属性。

当 MAEE 为 1 时，MMU 的 pte 中扩展地址属性位，用户可以配置页面的地址属性。

XUANTIEISAEE-使能扩展指令集：

当 XUANTIEISAEE 为 0 时，使用 C908X 扩展指令集时（除 Cache 指令子集、多核同步指令子集之外）产生非法指令异常；

当 XUANTIEISAEE 为 1 时，可以使用 C908X 扩展指令集。

备注

玄铁扩展 Cache 指令子集、多核同步指令子集在 mxstatus.XUANTIEISAEE == 1、mxstatus.COPINSTEE == 0 或 mxstatus.XUANTIEISAEE == 0、mxstatus.COPINSTEE == 1 时可以正常执行。

COPINSTEE-使能用户自扩展协处理器指令集：

当 COPINSTEE 为 0 时，使用用户自扩展协处理器指令集产生非法指令异常。

当 COPINSTEE 为 1 时，可以使用用户自扩展协处理器指令集。

备注

- 在软件切换玄铁扩展和通用协处理器扩展指令过程中，需要先清原指令使能开关位，之后再打开新的指令开关位。同时设置玄铁扩展使能位和通用协处理器扩展指令使能位被认为无效操作，硬件保证不会有同时打开情况。

PM-处理器所处特权模式：

当 PM=2'b00 时，表征当前处理器运行在用户模式。

当 PM=2'b01 时，表征当前处理器运行在超级用户模式。

当 PM=2'b11 时，表征当前处理器运行在机器模式。（Reset 后为机器模式）

20.1.8.2　机器模式硬件配置寄存器（MHCR）

机器模式硬件配置寄存器（MHCR）用于对处理器进行配置，包括性能和功能。

该寄存器的位长是 64 位，寄存器的读写权限是机器模式可读写，即非机器模式访问都会导致非法指令异常。

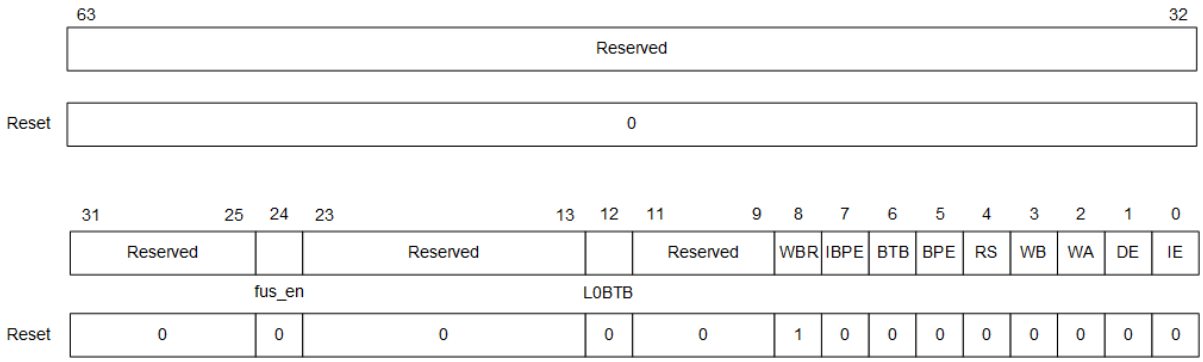


图 20.10: 机器模式硬件配置寄存器（MHCR）

IE-Icache 使能位：

当 IE=0 时，Icache 关闭。

当 IE=1 时，Icache 打开。

DE-Dcache 使能位：

当 DE=0 时，Dcache 关闭。

当 DE=1 时，Dcache 打开。

WA-高速缓存写分配设置位：

当 WA=0 时，数据高速缓存为 write non-allocate 模式。

当 WA=1 时，数据高速缓存为 write allocate 模式。

WB-高速缓存写回设置位：

当 WB=0 时，数据高速缓存为写直模式。

当 WB=1 时，数据高速缓存为写回模式。

C908X 只支持写回模式，WB 固定为 1。

RS-地址返回栈设置位：

当 RS=0 时，返回栈关闭。

当 RS=1 时，返回栈开启。

BPE-允许预测跳转设置位：

当 BPE=0 时，预测跳转关闭。

当 BPE=1 时，预测跳转开启。

BTB-分支目标预测使能位：

当 BTB=0 时，分支目标预测关闭。

当 BTB=1 时，分支目标预测开启。

IBPE-间接分支跳转预测跳转使能位：

当 IBPE=0 时，间接分支跳转预测关闭。

当 IBPE=1 时，间接分支跳转预测开启。

WBR-写突发传输使能位：

当 WBR=0 时，不支持写突发传输。

当 WBR=1 时，支持写突发传输。

C908X 默认为 1，不可设置。

L0BTB-第一级分支目标预测使能位：

当 L0BTB=0 时，第一级分支目标预测关闭。

当 L0BTB=1 时，第一级分支目标预测开启。

fus_en - 指令融合使能位：

当 fus_en=0 时，关闭指令融合。

当 fus_en=1 时，开启指令融合。

20.1.8.3　机器模式硬件操作寄存器（MCOR）

机器模式硬件操作寄存器（MCOR）用于对高速缓存和分支预测部件进行相关操作。

该寄存器的位长是 64 位，寄存器的读写权限是机器模式可读写，即非机器模式访问都会导致非法指令异常。

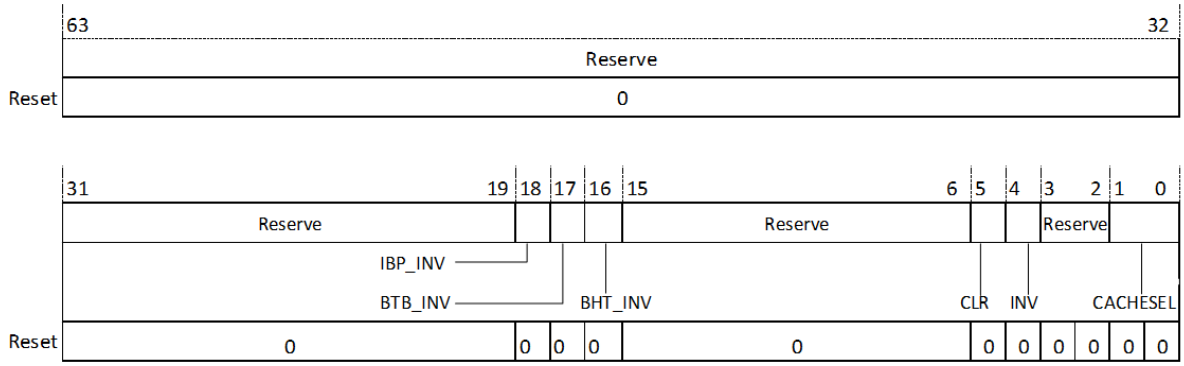


图 20.11: 机器模式硬件操作寄存器（MCOR）

CACHESEL-高速缓存选择位：

当 CACHE_SEL=2'b01 时，选中指令高速缓存。

当 CACHE_SEL=2'b10 时，选中数据高速缓存。

当 CACHE_SEL=2'b11 时，选中指令和数据高速缓存。

INV-高速缓存无效化设置位：

当 INV=0 时，高速缓存不进行无效化。

当 INV=1 时，高速缓存进行无效化。

CLR-高速缓存脏表现清除设置位：

当 CLR=0 时，高速缓存被标记为脏的表项不会被写到片外。

当 CLR=1 时，高速缓存被标记为脏的表项会被写到片外。

BHT_INV-BHT 无效设置位：

当 BHT_INV=0 时，分支历史表内的数据不进行无效化。

当 BHT_INV=1 时，分支历史表内的数据进行无效化。

BTB_INV-BTB 无效设置位：

当 BTB_INV=0 时，分支目标缓冲器内的数据不进行无效化。

当 BTB_INV=1 时，分支目标缓冲器内的数据进行无效化。

IBP_INV-IBP 无效设置位：

当 IBP_INV=0 时，间接跳转分支预测的数据不进行无效化。

当 IBP_INV=1 时，间接跳转分支预测的数据进行无效化。

以上所有无效化操作和清脏表现操作，在写的时候置高，在操作完成时，清 0。

20.1.8.4　机器模式 L2Cache 控制寄存器（MCCR2）

机器模式 L2Cache 控制寄存器（MCCR2）用来配置共享的二级高速缓存中各个存储器的访问延时，二级高速缓存有效／无效，指令预取能力和 TLB 预取使能，以及 ECC 校验使能。

该寄存器的位长是 64 位，寄存器的读写权限是机器模式可读写，即非机器模式访问都会导致非法指令异常。

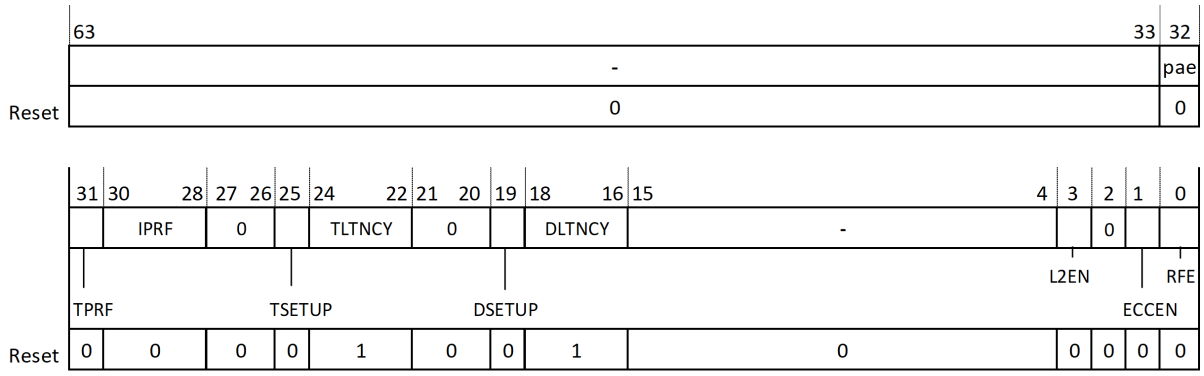


图 20.12: 机器模式 L2Cache 控制寄存器（MCCR2）

RFE-数据访问读分配使能位：

当 RFE=0 时，数据访问 L2 Cache 缺失时，不回填 L2 Cache，直接回填 D Cache，即 L1 DCache 与 L2Cache 之间为 exclusive 包含性关系。

当 RFE=1 时，数据访问 L2 Cache 缺失时，回填 L2 Cache，同时回填 D Cache，即 L1 DCache 与 L2Cache 之间为 inclusive 包含性关系（C908X 固定为 0）。

ECCEN-ECC 使能位：

当 ECCEN=0 时，L2Cache ECC 关闭。

当 ECCEN=1 时，L2Cache ECC 开启。

L2EN-L2Cache 使能位：

当 L2EN=0 时，L2Cache 关闭。

当 L2EN=1 时，L2Cache 开启。

DLTNCY-L2Cache DATA RAM 访问周期配置位：

当 DLTNCY=0 时，DATA RAM 访问周期为 1。

当 DLTNCY=1 时，DATA RAM 访问周期为 2。

当 DLTNCY 为 2 时，DATA RAM 访问周期为 3。

当 DLTNCY 为 3 时，DATA RAM 访问周期为 4。

当 DLTNCY 为 4 时，DATA RAM 访问周期为 5。

当 DLTNCY 为 5 时，DATA RAM 访问周期为 6。

当 DLTNCY 为 6 时，DATA RAM 访问周期为 7。

当 DLTNCY 为 7 时，DATA RAM 访问周期为 8。

DSETUP-L2Cache DATA RAM 的 setup 配置位

当 DSETUP 为 0 时，DATA RAM 不需要额外的 setup 周期；

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

TLTNCY-L2Cache TAG RAM 的访问周期配置位：

当 TLTNCY 为 0 时，TAG RAM 访问周期为 1；

当 TLTNCY 为 1 时，TAG RAM 访问周期为 2；

当 TLTNCY 为 2 时，TAG RAM 访问周期为 3；

当 TLTNCY 为 3 时，TAG RAM 访问周期为 4；

当 TLTNCY 为 4 时，TAG RAM 访问周期为 5。

TSETUP-L2 CACHE TAG RAM 的 setup 配置位：

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

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

IPRF-L2Cache 指令预取能力：

指示取指请求访问 L2Cache 缺失时预取的缓存行数量：

当 IPRF 为 0 时，L2Cache 指令预取功能关闭；

当 IPRF 为 1 时，预取 1 条缓存行；

当 IPRF 为 2 时，预取 2 条缓存行；

当 IPRF 为 3 时，预取 4 条缓存行；

当 IPRF 为 4 时，预取 8 条缓存行；

当 IPRF 为 5 时，预取 16 条缓存行；

当 IPRF 为 6 时，预取 32 条缓存行。

TPRF-L2Cache TLB 预取使能：

当 TPRF 为 0 时，L2Cache TLB 预取功能关闭；

当 TPRF 为 1 时，L2Cache TLB 预取功能开启。

pae-Partition 访问使能位：

当 pae=0 时，表示 L2cache 不支持 parttion 访问。

当 pae=1 时，表示 L2cache 支持 partition 访问。

20.1.8.5　机器模式 L2 Cache 错误控制寄存器（MCER2）

机器模式 L2 Cache 错误控制寄存器（MCER2）用于指示 CPU 写数据错误以及 L2 Cache ECC 错误信息。当 CPU 向总线发起写请求返回错误时，硬件自动更新 MCER2 寄存器的 ERR_VLD 比特、ST_ERR 比特以及错误的页面、核信息，供软件查询。二级高速缓存支持可配置的 ECC，实现 1 比特错误可纠正、2 比特错误可检测的功能。当出现 1 比特或 2 比特错误时，硬件自动置位 MCER2 寄存器的 ERR_VLD 比特、ECC_ERR 比特以及错误的位置信息，供软件查询。软件可通过写入 0 的方式清除 ERR_VLD 比特，但不能对其置 1。

该寄存器的位长是 64 位，寄存器的读写权限是机器模式可读写，即非机器模式访问都会导致非法指令异常。

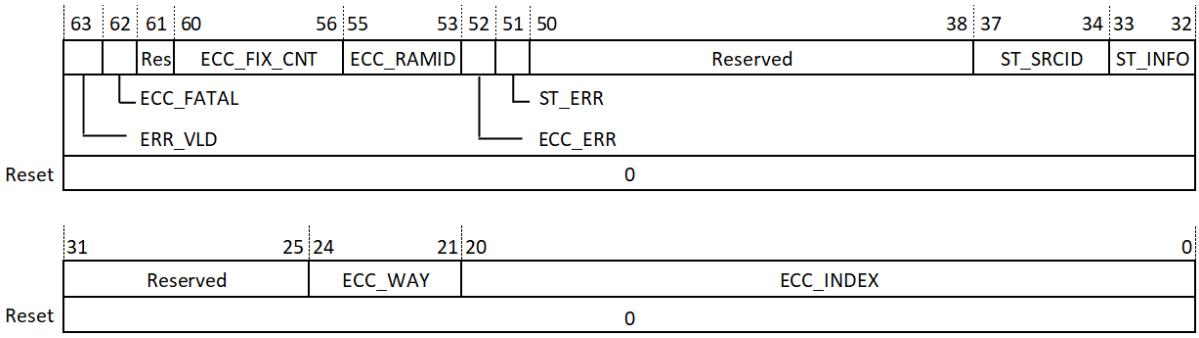


图 20.13: 机器模式 L2Cache ECC 控制寄存器（MCER2）

ERR_VLD-错误指示位：

当 ERR_VLD 为 0，无校验错误、写总线错误发生。

当 ERR_VLD 为 1，出现校验错误或写总线错误，可通过进一步查看 ECC_ERR、ST_ERR 确认错误源头。

软件可在异常服务程序中清除该错误位，但无法置高。

ECC_FATAL-L2 CACHE ECC Fatal 错误位：

当 ECC_FATAL 为 0，L2 CACHE 没有出现 2 比特 ECC 错误。

当 ECC_FATAL 为 1，L2 CACHE 发生 2 比特 ECC 错误。

该位机器模式只读。

ECC_FIX_CNT[4:0]-已修复的 ECC Error 数量：

记录已修复的 ECC Error 数量。

FIX_CNT 计满保持。

该位机器模式只读。

ECC_RAMID[2:0]-发生 ECC 错误的 SRAM ID 号：

记录发生 ECC 错误的 SRAM ID 号。

ID=0：L2 CACHE TAG RAM

ID=1：L2 CACHE DATA RAM

ID=2：L2 CACHE DIRTY RAM

ID=3：Snoop Filter Core 0 RAM

ID=4：Snoop Filter Core 1 RAM

ID=5：Snoop Filter Core 2 RAM

ID=6：Snoop Filter Core 3 RAM

该位机器模式只读。

ECC_ERR-L2 CACHE 校验错误位：

当 ECC_ERR 为 0，无校验错误发生。

当 ECC_ERR 为 1，出现校验错误。

该位机器模式只读。

ST_ERR-写总线错误位：

当 ST_ERR 为 0，无写总线发生。

当 ST_ERR 为 1，出现写总线错误。

该位机器模式只读。

ST_SRCID[3:0]-写总线错误源

记录触发写总线错误的源头。

ID=0：核 0

ID=1：核 1

ID=2：核 2

ID=3：核 3

ID=4：核 4

ID=8：DCP 接口

ID=15：L2 CACHE 替换

该位机器模式只读。

ST_INFO[1:0]-写总线错误信息

记录触发写总线错误的页面信息。

ST_INFO=0：AXI 主接口 cacheable 页面

ST_INFO=1：AXI 主接口/LLP 接口 non-cacheable weak-ordered 页面

ST_INFO=2：AXI 主接口/LLP 接口 non-cacheable strong-ordered 页面

ST_INFO=3：APB non-cacheable strong-ordered 页面

该位机器模式只读。

ECC_WAY-L2 CACHE 校验错误的路位置信息：

记录 L2 CACHE 第一次出现 2 比特校验错误的路位置。

该位机器模式只读。

ECC_INDEX-L2 CACHE 校验错误的索引信息：

记录 L2 CACHE 第一次出现 2 比特校验错误的索引位置。

该位机器模式只读。

20.1.8.6　机器模式隐式操作寄存器（MHINT）

机器模式隐式操作寄存器（MHINT）用于高速缓存多种功能开关控制。

该寄存器的位长是 64 位，寄存器的读写权限是机器模式可读写，即非机器模式访问都会导致非法指令异常。

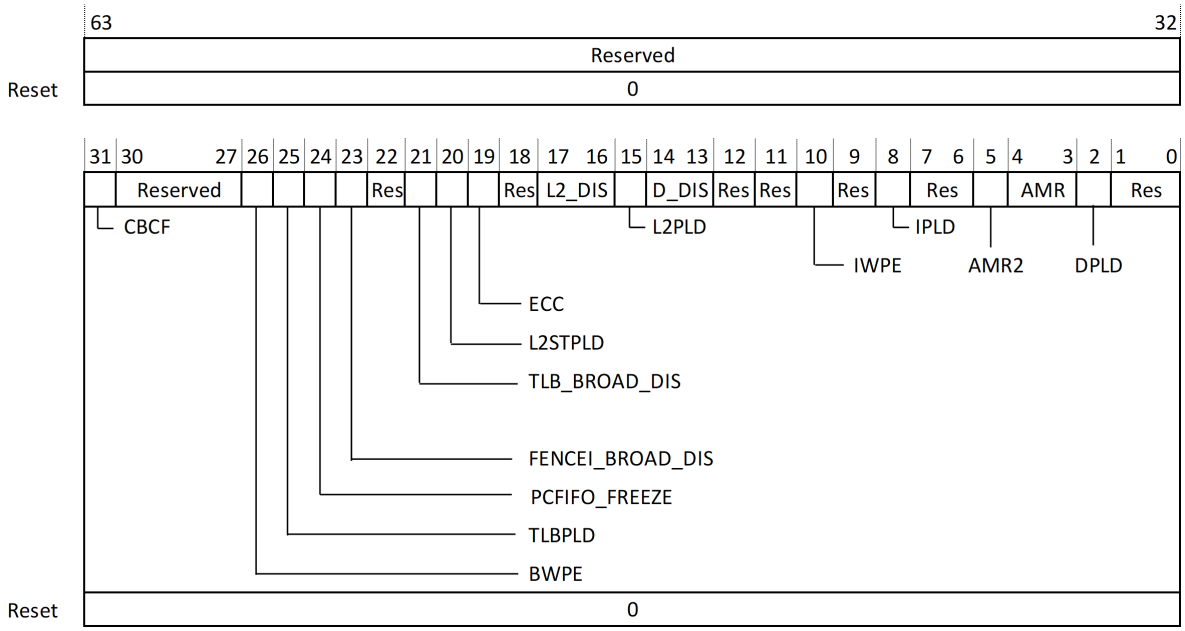


图 20.14: 机器模式隐式操作寄存器（MHINT）

DPLD-DCACHE 预取使能位：

当 DPLD 为 0 时，DCache 预取关闭；

当 DPLD 为 1 时，DCache 预取开启。

AMR-L1 Cache 写分配策略自动调整使能位：

当 AMR 为 0 时，写分配策略由访问地址的页面属性 WA 决定。

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

当 AMR 为 2 时，在出现连续 15 条缓存行存储操作时后续连续地址的存储操作不再写入 L1 Cache。

当 AMR 为 3 时，在出现连续 63 条缓存行存储操作时后续连续地址的存储操作不再写入 L1 Cache。

AMR2-L2 Cache 写分配策略自动调整使能位

当 AMR2 为 0 时，写分配策略由访问地址的页面属性 WA 决定。

当 AMR2 为 1 时，在出现连续 4 条缓存行的存储操作时后续连续地址的存储操作不再写入 L2 Cache。

IPLD-ICACHE 预取使能位：

当 IPLD 为 0 时，ICache 预取关闭。

当 IPLD 为 1 时，ICache 预取开启。

IWPE-ICache 路预测使能位：

当 IWPE 为 0 时，ICache 路预测关闭；（默认绑 0）

当 IWPE 为 1 时，ICache 路预测开启。

D_DIS-DCache 预取缓存行数量：

当 DPLD 为 0 时，预取 2 条缓存行。

当 DPLD 为 1 时，预取 4 条缓存行。

当 DPLD 为 2 时，预取 8 条缓存行。

当 DPLD 为 3 时，预取 16 条缓存行。

默认为 0。

L2PLD-L2 Cache 预取使能位：

当 L2PLD 为 0 时，L2 Cache 预取关闭；

当 L2PLD 为 1 时，L2 Cache 预取开启。

L2_DIS-L2 Cache 预取缓存行数量：

当 L2_DIS 为 0 时，预取 2 条缓存行；

当 L2_DIS 为 1 时，预取 4 条缓存行；

当 L2_DIS 为 2 时，预取 8 条缓存行；

当 L2_DIS 为 3 时，预取 16 条缓存行；

L2 Cache 的预取是在 L1 Cache 预取的基础上再次进行预取。

ECC-L1 CACHE ECC 校验使能位：

当 ECC 为 0 时，L1Cache 校验功能关闭；

当 ECC 为 1 时，L1Cache 校验功能开启。

L2STPLD-L2 Cache Store 预取使能位：

当 L2STPLD 为 0 时，L2 CACHE store 预取关闭；

当 L2STPLD 为 1 时，L2 CACHE store 预取开启。

TLB_BROAD_DIS-TLB fence 操作广播取消位：

当 TLB_BROAD_DIS 为 0 时，sfence.vma 指令操作广播到其他核；

当 TLB_BROAD_DIS 为 1 时，sfence.vma 指令操作不广播。

单核情况下，该位不存在。

FENCEI_BROAD_DIS-fence.i 操作广播取消位：

当 FENCEI_BROAD_DIS 为 0 时，fence.i 指令操作广播到其他核；

当 FENCEI_BROAD_DIS 为 1 时，fence.i 指令操作不广播。

单核情况下，该位不存在。

PCFIFO_FREEZE-Debug 内 PCFIFO 记录跳转目标 PC 使能位：

当 PCFIFO_FREEZE 为 0 时，PCFIFO 正常记录跳转目标 PC；

当 PCFIFO_FREEZE 为 1 时，PCFIFO 停止记录跳转目标 PC；

TLBPLD-TLB 预取使能位：

当 TLBPLD 为 0 时，TLB 预取关闭；

当 TLBPLD 为 1 时，TLB 预取打开；

BWPE-分支预测路预测使能位

当 BWPE 为 0 时，分支预测路预测关闭；

当 BWPE 为 1 时，分支预测路预测开启。

CBCF-Data Cache Clear 指令增加 Invalid 操作：

当 CBCF 为 0 时，Data Cache Clear 指令正常执行，仅包含 Clear 操作；

当 CBCF 为 1 时，Data Cache Clear 指令除 Clear 外，还包含 Invalid 操作（Flush）。

20.1.8.7　机器模式复位向量基址寄存器（MRVBR）

机器模式复位寄存器（MRVBR）用来保存复位异常向量的基址。每个 C908X 核心拥有独立的 MRVBR 寄存器。

该寄存器的位长是 64 位，寄存器的读写权限是机器模式只读，即非机器模式访问都会导致非法指令异常。

	63 … 10	9 … 0
	Reset vector base	Reserve
Reset	0	0

图 20.15: 机器模式复位向量基址寄存器（MRVBR）

Reset vector base-复位基址：

控制核心的复位基址。

20.1.8.8　机器模式 L1 Cache ECC 寄存器（MCER）

机器模式 L1 Cache ECC 控制寄存器（MCER）用于对 L1 Cache ECC 进行配置。一级高速缓存支持可配置的 ECC，实现 1 比特错误可纠正、2 比特错误可检测的功能。当发现是 2 比特以上错误时，硬件自动设置 MCER 寄存器的 ERR_FATAL 比特以及错误的位置信息，供软件查询，软件可通过写入 0 的方式清除 ERR_FATAL 比特，但不能对其置 1。

该寄存器的位长是 64 位，寄存器的读写权限是机器模式可读写，即非机器模式访问都会导致非法指令异常。

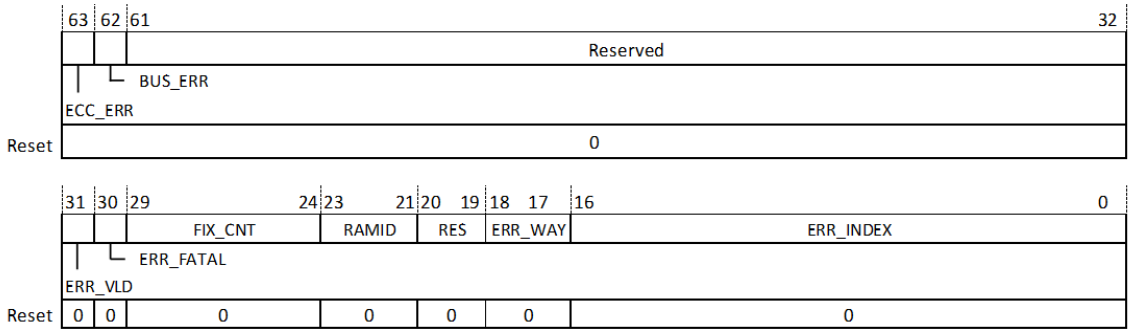


图 20.16: L1 Cache ECC 寄存器（MCER）

ECC_ERR - ECC 信息有效位：

- 当 ERR_VLD 为高时，当 ECC_ERR 为 0 时，L1 CACHE ECC 信息无效；
- 当 ERR_VLD 为高时，当 ECC_ERR 为 1 时，L1 CACHE ECC 信息有效。

BUS_ERR - 总线异常有效位：

- 当 ERR_VLD 为高时，当 BUS_ERR 为 0 时，表示没有发生 ld 总线异常；
- 当 ERR_VLD 为高时，当 BUS_ERR 为 1 时，表示发生 ld 总线异常。

ERR_VLD - 异步异常有效位：

- 当 ERR_VLD 为 0 时，表示没有产生总线异常或 ECC err；
- 当 ERR_VLD 为 1 时，表示产生总线异常或者发生 ECC err。

ERR_FATAL - L1 CACHE 校验错误位：

- 当 ERR_FATAL 为 0 时，表示硬件可修复 ERR；
- 当 ERR_FATAL 为 1 时，表示发生 2bit 以上 ECC 错误，该位只能由软件清除。

FIX_CNT - 已修复 ERROR 计数位：

- 记录已修复 ERR 的数量，当 ERR_VLD 被清除时该位被自动清零。

RAMID - 出现 ECC FATAL 错误的 RAM：

- 当 RAMID 为 0 时，L1 ICACHE TAG RAM 校验出错；
- 当 RAMID 为 1 时，L1 ICACHE DATA RAM 校验出错；
- 当 RAMID 为 2 时，L1 DCACHE TAG RAM 校验出错；
- 当 RAMID 为 3 时，L1 DCACHE DATA RAM 校验出错；
- 当 RAMID 为 4 时，JTLB TAG RAM 校验出错；
- 当 RAMID 为 5 时，JTLB DATA RAM 校验出错。

ERR_WAY - 第一次出现 ECC FATAL 错误时的路位置：

记录出错 RAM 第一次出现 ECC FATAL ERROR 时的路位置，在第一次错误没有被软件处理前再次发生的错误不会更新该信息。

ERR_INDEX - 第一次出现 ECC FATAL 错误时的索引位置：

记录出错 RAM 第一次出现 ECC FATAL ERROR 时的索引位置，在第一次错误没有被软件处理前再次发生的错误不会更新该信息。

20.1.8.9　性能监控控制寄存器（MHPMCR）

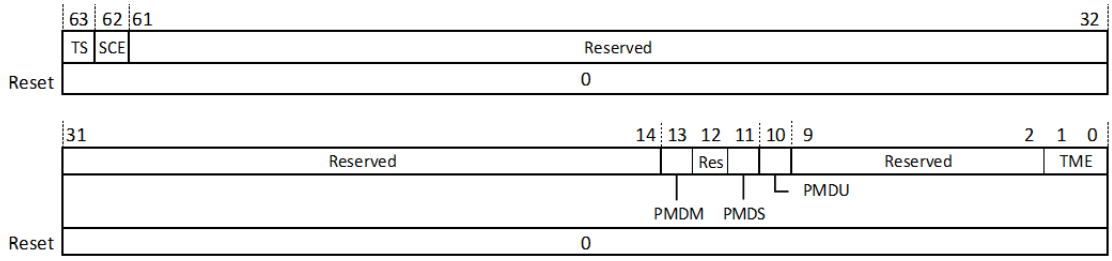


图 20.17: 性能监控控制寄存器（MHPMCR）

TS-触发状态位

指示性能监测是否处于触发态。

- 1'b0：未处于触发状态
- 1'b1：处于触发状态

SCE-超级用户模式控制使能位

控制超级用户模式是否有权限控制性能监控寄存器：

- 1'b0：S 态读写 shpmcr、触发寄存器将触发非法指令
- 1'b1：S 态读写使能，shpmcr、触发寄存器可以正常读写

PMDM-机器模式性能监测计数使能位，为 mxstatus.pmdm 位的映射

- 当 PMDM 为 0 时，机器模式下允许性能计数器计数。
- 当 PMDM 为 1 时，机器模式下禁止性能计数器计数。

PMDS-超级用户模式性能监测计数使能位，为 mxstatus.pmds 位的映射

- 当 PMDS 为 0 时，超级用户模式下允许性能计数器计数。
- 当 PMDS 为 1 时，超级用户模式下禁止性能计数器计数。

PMDU-用户模式性能监测计数使能位，为 mxstatus.pmdu 位的映射

- 当 PMDU 为 0 时，用户模式下允许性能计数器计数。
- 当 PMDU 为 1 时，用户模式下禁止性能计数器计数。

TME-性能监测计数触发模式位

- 当 TME 为 2'b00：触发模式未使能，使用原有模式进行计数；

- 当 TME 为 2'b01：Trigger/Stop 触发模式使能，当程序地址与起始触发寄存器匹配且触发功能使能时，触发功能开启，事件计数器开始计数。当程序地址与终止触发寄存器匹配，则触发功能关闭，结束计数
- 当 TME 为 2'b10：Start/End 触发功能使能。当程序地址落在起始触发寄存器和终止触发寄存器标志的范围内时，事件计数器正常计数，否则不进行计数
- 当 TME 为 2'b11：保留状态，无意义

【注】TS 字段的访问权限为 MRO，其余字段的访问权限为 MRW。

20.1.8.10　性能监控起始/终止触发寄存器（MHPMSR/MHPMER）

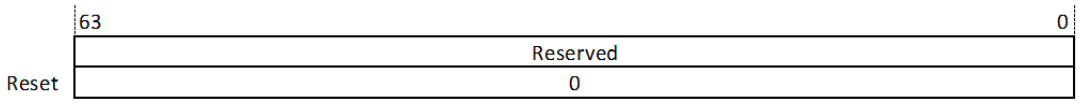


图 20.18: 性能监控起始/终止触发寄存器（MHPMSR/MHPMER）

MHPMSR 和 MHPMER 一起完成性能监控的触发监控任务。这两寄存器的宽度均为 64 bit，分别存放开始和终止计数的 PC 值。处理器将使用虚拟地址作为匹配对象，若 MMU 不使能或为 M mode，使用物理地址作为匹配对象。

- 当 MHPMCR.TME 为 2'b01 时，即为 TRIGGER/STOP 模式：

  程序运行至 MHPMSR 指定的 PC，计数器开始计数，直到程序运行至终止触发寄存器（MHPMER）结束统计。

- 当 MHPMCR.TME 为 2'b10 时，即为 START/END 模式：

  当程序运行 PC 在起始触发寄存器（MHPMSR）和终止触发寄存器（MHPMER）的范围内时，计数器触发并计数，一旦指令退休地址不在该范围内，所有计数器停止计数。

相比于 TRIGGER/STOP 模式来说，START/END 模式对计数的条件更加严格，即当程序段中发生了子程序调用等导致指令运行不在 START 和 END 之间，都会引起计数器的停止。

20.1.8.11　处理器 ZONE ID 寄存器（MZONEID）

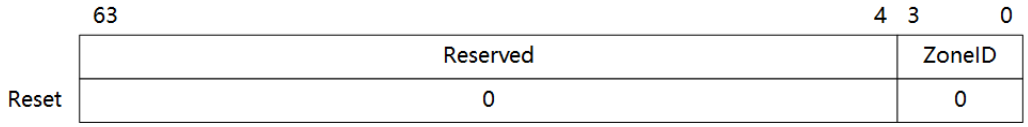


图 20.19: 处理器 ZONE ID 寄存器（MZONEID）

ZoneID

该字段访问权限为 MRW。处理器可以配置当前处于的 ZoneID 号。不同的 ZoneID，对应不同的权限。

20.1.8.12　细粒度回填 ID 寄存器（MLLCPID）

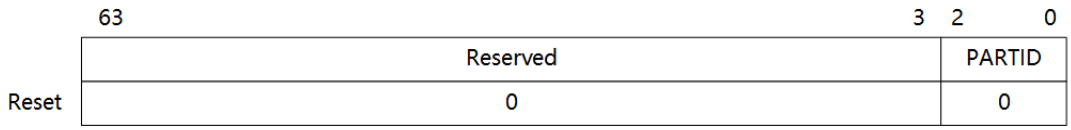


图 20.20: 细粒度回填 ID 寄存器（MLLCPID）

PARTID - 处理器末级 cache partition 访问 ID

该字段访问权限为 MRW。在末级 cache 中，通过匹配 PARTID 来决定，当前处理器可以使用的 cache way group。

20.1.8.13　L2 细粒度配置寄存器（MLLWP）

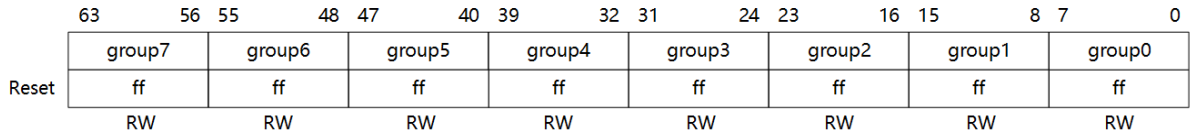


图 20.21: L2 细粒度配置寄存器（MLLWP）

group0[7:0] - group 0 可访问 ID

包含 8 个 id，当处理器 MLLCPID 号对应的 bit 为 1 时，则表示此处理器可以访问 group0 的 cache way。

group1[7:0] - group 1 可访问 ID

包含 8 个 id，当处理器 MLLCPID 号对应的 bit 为 1 时，则表示此处理器可以访问 group1 的 cache way。

group2[7:0] - group 2 可访问 ID

包含 8 个 id，当处理器 MLLCPID 号对应的 bit 为 1 时，则表示此处理器可以访问 group2 的 cache way。

group3[7:0] - group 3 可访问 ID

包含 8 个 id，当处理器 MLLCPID 号对应的 bit 为 1 时，则表示此处理器可以访问 group3 的 cache way。

group4[7:0] - group 4 可访问 ID

包含 8 个 id，当处理器 MLLCPID 号对应的 bit 为 1 时，则表示此处理器可以访问 group4 的 cache way。

group5[7:0] - group 5 可访问 ID

包含 8 个 id，当处理器 MLLCPID 号对应的 bit 为 1 时，则表示此处理器可以访问 group5 的 cache way。

group6[7:0] - group 6 可访问 ID

包含 8 个 id，当处理器 MLLCPID 号对应的 bit 为 1 时，则表示此处理器可以访问 group6 的 cache way。

group7[7:0] - group 7 可访问 ID

包含 8 个 id，当处理器 MLLCPID 号对应的 bit 为 1 时，则表示此处理器可以访问 group7 的 cache way。

20.1.8.14　机器模式计数器写使能寄存器（MCOUNTERWEN）

机器模式计数器写使能寄存器（MCOUNTERWEN），用于授权超级用户模式是否可以写超级用户模式事件计数器。

该寄存器的位长是 64 位，寄存器的读写权限是机器模式可读写，即非机器模式访问都会导致非法指令异常。

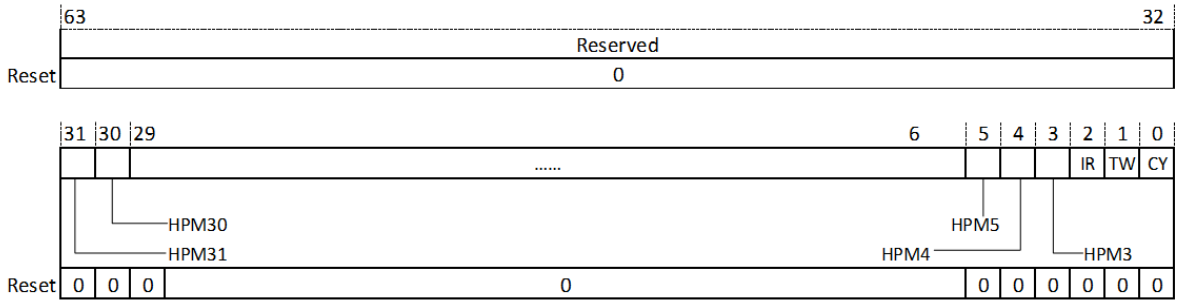


图 20.22: 机器模式计数器写使能寄存器（MCOUNTERWEN）

- 当 mcounterwen.bit[n] 为 1 时，允许超级用户模式下写对应 shpmcounter。
- 当 mcounterwen.bit[n] 为 0 时，不允许超级用户模式下写对应的 shpmcounter，否则产生非法指令异常。

20.1.9　机器模式 Cache 访问扩展寄存器组

机器模式 Cache 访问扩展寄存器用于直接读取 L1 和 L2 高速缓存中内容，便于对高速缓存进行调试。

20.1.9.1　机器模式 Cache 指令寄存器（MCINS）

机器模式 Cache 指令寄存器（MCINS）用于向 L1 或 L2 高速缓存发起读请求。

该寄存器的位长是 64 位，寄存器的读写权限是机器模式可读写，即非机器模式访问都会导致非法指令异常。

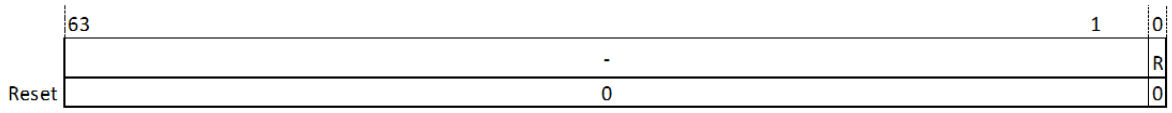


图 20.23: 机器模式 Cache 指令寄存器（MCINS）

R-Cache 读访问：

- 当 R 为 0 时，不发起 Cache 读请求。
- 当 R 为 1 时，发起 Cache 读请求。

20.1.9.2　机器模式 Cache 访问索引寄存器（MCINDEX）

机器模式 Cache 访问索引寄存器（MCINDEX）用于配置读请求访问的 Cache 位置信息。

该寄存器的位长是 64 位，寄存器的读写权限是机器模式可读写，即非机器模式访问都会导致非法指令异常。

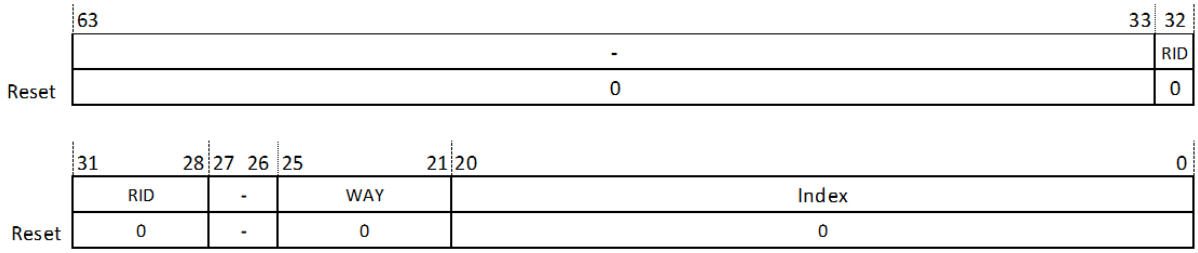


图 20.24: 机器模式 Cache 访问索引寄存器（MCINDEX）

RID-RAM 标志位：

指示访问的 RAM 信息。

- 当 RID 为 0 时，表示访问的是 ICACHE TAG RAM。
- 当 RID 为 1 时，表示访问的是 ICACHE DATA RAM。
- 当 RID 为 2 时，表示访问的是 DCACHE ST TAG RAM。
- 当 RID 为 3 时，表示访问的是 DCACHE DATA RAM。
- 当 RID 为 4 时，表示访问的是 L2CACHE TAG RAM。
- 当 RID 为 5 时，表示访问的是 L2CACHE DATA RAM。
- 当 RID 为 6 时，表示访问的是 ICACHE TAG ECC RAM。
- 当 RID 为 7 时，表示访问的是 ICACHE DATA ECC RAM。
- 当 RID 为 8 时，表示访问的是 DCACHE ST TAG ECC RAM。
- 当 RID 为 9 时，表示访问的是 DCACHE DATA ECC RAM。

- 当 RID 为 10 时，表示访问的是 L2CACHE TAG ECC RAM。
- 当 RID 为 11 时，表示访问的是 L2CACHE DATA ECC RAM。
- 当 RID 为 12 时，表示访问的是 DCACHE LD TAG RAM。
- 当 RID 为 13 时，表示访问的是 DCACHE LD TAG ECC RAM。
- 当 RID 为 19 时，表示访问的是 ICACHE PREDECODE RAM。
- 当 RID 为 20 时，表示访问的是 SNOOP FILTER RAM。
- 当 RID 为 21 时，表示访问的是 SNOOP FILTER ECC RAM。

WAY-Cache 路信息：

指示 RAM 访问的路位置信息。

INDEX-Cache 索引：

指示 RAM 访问的索引位置信息。该域按照地址的格式写入：

- 对于 data arrary，每次可以读取 128 bit，index 的使用会忽略低 4 位地址；
- 对于 tag array，针对不同的处理器的 cache line 大小，忽略的地址会有不同，对于 cache line 大小为 512 bit，index 会忽略低 6 位的地址，并以此类推。

  [20:19] == 2'b00 时，读 jTLB；

  [20:19] == 2'b10 时，读 dutlb；

  [20:19] == 2'b01 时，读 iutlb。

20.1.9.3　机器模式 Cache 数据寄存器（MCDATA0/1）

机器模式 Cache 数据寄存器（MCDATA0/1）用于记录读取 L1 或 L2 高速缓存的数据。

该寄存器的位长是 64 位，寄存器的读写权限是机器模式只读，即非机器模式访问都会导致非法指令异常。

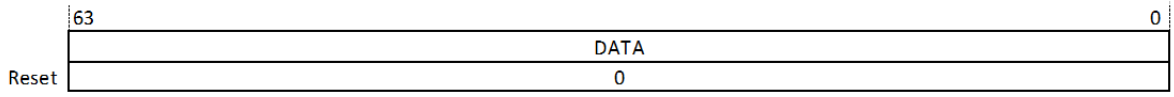


图 20.25: 机器模式 Cache 访问数据寄存器（MCDATA）

表 20.2: 机器模式 Cache 访问数据寄存器与 RAM 类型对应关系

RAM 类型	CDATA 内容
ICACHE TAG	CDATA0[39:12]: TAG CDATA0[0]: VALID
ICACHE DATA	CDATA0~CDATA1: 128 bit DATA

续下页

表 20.2 - 接上页

RAM 类型	CDATA 内容
DCACHE LD TAG	CDATA0[38:13]: 26 bit tag CDATA0[12:11]: cindex[13:12] CDATA0[0]: VALID
DCACHE DATA	CDATA0~CDATA1: 128 bit DATA
DCACHE TAG	CDATA0[39:12]: TAG CDATA0[2]: DIRTY CDATA0[1]: SHARED CDATA0[0]: VALID
L2CACHE TAG	CDATA0[39:12]: TAG+INDEX CDATA0[3:0]: {valid, share, dirty, pend}
L2CACHE DATA	CDATA0~CDATA1: 128 bit DATA
L2CACHE ECC	CDATA0[63:0]: ECC
SNOOP FILTER DATA	CDATA0[39:12]: tag+index of this way CDATA0[5:4]: rrpv of this way CDATA0[3:0]: valid
SNOOP FILTER ECC	CDATA0[11:5]: tag ecc of this way CDATA0[3:0]: info ecc of this way
ICACHE PRECODE	CDATA0[31:0]: Precode Data

20.1.9.4　机器模式 L1Cache 硬件错误注入寄存器（MEICR）

机器模式 L1Cache 硬件错误注入寄存器（MEICR）用于向 L1 Cache 注入 ECC error。

该寄存器的位长是 64 位，寄存器的读写权限是机器模式可读写，即非机器模式访问都会导致非法指令异常。

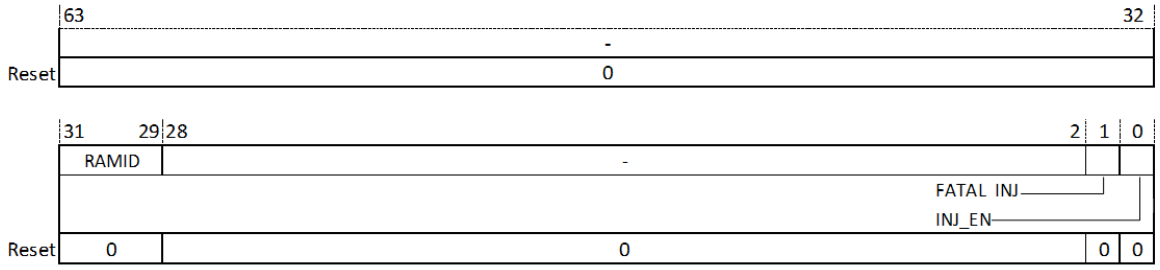


图 20.26: 机器模式 L1Cache 硬件错误注入寄存器（MEICR）

INJ_EN-ECC error 注入使能位：

当 INJ_EN 为 1 时，L1cache ECC error 注入开启；

当 INJ_EN 为 0 时，L1cache ECC error 注入关闭。

FATAL_INJ-ECC ERROR 注入选择位：

当 FATAL_INJ 为 1 时，注入 2bit error；

当 FATAL_INJ 为 0 时，注入 1bit error。

RAMID-ECC RAM 索引：

当 RAMID 为 0 时，注入 ICACHE TAG RAM；

当 RAMID 为 1 时，注入 ICACHE DATA RAM；

当 RAMID 为 2 时，随机注入 DCACHE TAG RAM 或者 DCACHE DIRTY RAM。当随机注入到 DCACHE DIRTY RAM 时，需要 store 操作才能触发错误。

当 RAMID 为 3 时，注入 DCACHE DATA RAM；

当 RAMID 为 4 时，注入 JTLB TAG RAM；

当 RAMID 为 5 时，注入 JTLB DATA RAM。

注意

软件使用 meicr 寄存器向 DCache 注入 1 bit error 并可以成功触发 ECC_ERR 的前提条件为：

在软件注入错误之后，需要保证发起的第一个访问 DCache 的请求必须命中，且发起第一个访问 DCache 请求的指令前面不能存在 fence 类指令或 sync 类指令。

20.1.9.5　机器模式 L2Cache 硬件错误注入寄存器（MEICR2）

机器模式 L2Cache 硬件错误注入寄存器（MEICR2）用于向 L2cache 注入 ECC error。

该寄存器的位长是 64 位，寄存器的读写权限是机器模式可读写，即非机器模式访问都会导致非法指令异常。

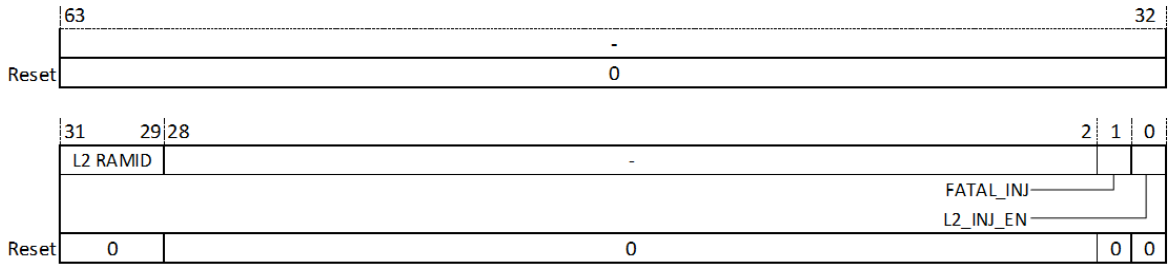


图 20.27: 机器模式 L2Cache 硬件错误注入寄存器（MEICR2）

L2_INJ_EN-L2 ECC ERROR 注入使能位：

当 L2_INJ_EN 为 1 时，L2Cache ECC error 注入开启；

当 L2_INJ_EN 为 0 时，L2Cache ECC error 注入关闭。

FATAL_INJ-ECC ERROR 注入选择位：

当 FATAL_INJ 为 1 时，注入 2bit error；

当 FATAL_INJ 为 0 时，注入 1bit error。

L2_RAMID-ECC RAM 索引：

当 RAMID 为 0 时，注入 L2 CACHE TAG RAM；

当 RAMID 为 1 时，注入 L2 CACHE DATA RAM；

当 RAMID 为 2 时，注入 L2 CACHE DIRTY RAM。

20.1.9.6 L1 LD BUS ERR 地址寄存器（MBEADDR）

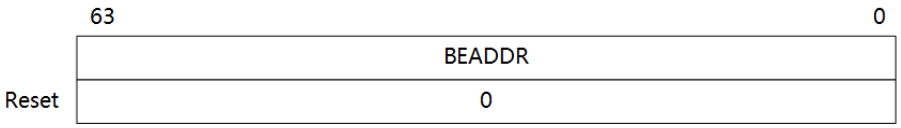


图 20.28: L1 LD BUS ERR 地址寄存器（MBEADDR）

BEADDR - 处理器总线异常地址寄存器

当处理器发生 ld 总线异常，该寄存器存储总线异常物理地址。

20.1.10 机器模式处理器型号寄存器组

20.1.10.1 机器模式处理器型号寄存器（MCPUID）

机器模式处理器型号寄存器（MCPUID）存储了处理器型号信息，其复位值由产品本身决定。

20.1.10.2 片上总线基地址寄存器（MAPBADDR）

该寄存器反映处理器片上寄存器（CLINT，PLIC）的基地址。该寄存器的值由硬件集成决定。

20.1.11 多核扩展寄存器组

20.1.11.1 Snoop 监听使能寄存器（MSMPR）

侦听使能寄存器，控制核心能否处理侦听请求。每个核心独立配置本核是否能够处理侦听请求，顶层一致性总线根据各个核的侦听状态控制侦听请求的发送。读写权限为机器模式可读写。

该寄存器的宽度为 64 位。

bit 0：SMPEN-核心侦听使能位

- 当 SMPEN 为 1'b0 时，核心不能够处理侦听请求，顶层屏蔽发送侦听请求给核心。（复位值）
- 当 SMPEN 为 1'b1 时，核心能够处理侦听请求，顶层发送侦听请求给核心。

处理器核心在下电前，必须设置核心对应的 SMPEN=0，以关闭核心的侦听功能。核心上电后，软件在打开 D-Cache 和 MMU 之前，必须设置核心的 SMPEN=1。核心在正常工作时（包括 WFI 模式），必须保持 SMPEN=1，否则结果不可预期。

20.2　附录 C-2 超级用户模式控制寄存器

超级用户模式控制寄存器按照功能分为：超级用户模式异常配置寄存器组、超级用户模式异常处理寄存器组、超级用户模式地址转换寄存器组。

20.2.1　超级用户模式异常配置寄存器组

当异常和中断被降级到超级用户模式响应时，跟机器模式一样，需要通过超级用户模式异常配置寄存器组进行异常的配置。

20.2.1.1　超级用户模式处理器状态寄存器（SSTATUS）

超级用户模式处理器状态寄存器（SSTATUS）存储了处理器在超级用户模式下的状态和控制信息，包括全局中断有效位、异常保留中断有效位、异常保留特权模式位等，是 MSTATUS 的部分映射。

该寄存器的位长是 64 位，寄存器的读写权限是超级用户模式可读写，用户模式访问都会导致非法指令异常。

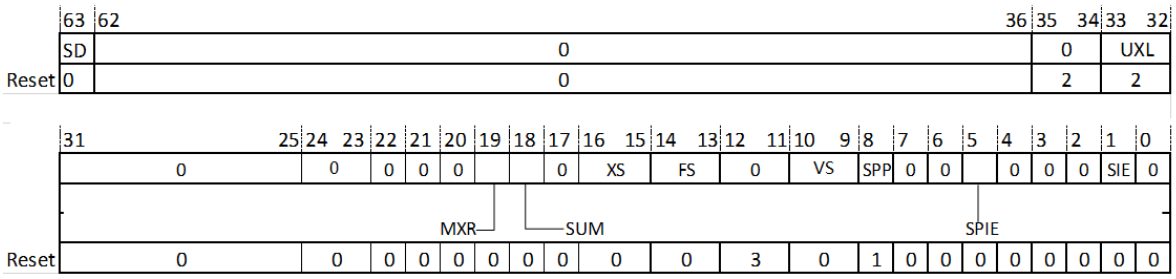


图 20.29: 超级用户模式处理器状态寄存器（SSTATUS）

具体信息请参考*机器模式处理器状态寄存器（MSTATUS）*。

20.2.1.2　超级用户模式中断使能控制寄存器（SIE）

超级用户模式中断使能控制寄存器（SIE）用于控制不同中断类型的使能和屏蔽，是 MIE 的部分映射。该寄存器的位长是 64 位，寄存器的读写权限是超级用户模式可读，超级用户模式下的写权限由对应位的 mideleg 决定，用户模式访问会导致非法指令异常。

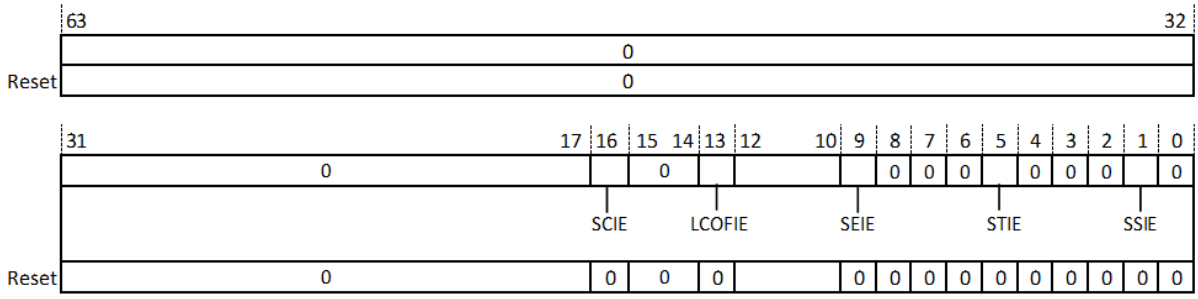


图 20.30: 超级用户模式中断使能控制寄存器（SIE）

具体信息请参考*机器模式中断使能控制寄存器（MIE）*。

20.2.1.3　超级用户模式向量基址寄存器（STVEC）

超级用户模式向量基址寄存器（STVEC）用于配置异常服务程序的入口地址。

该寄存器的位长是 64 位，寄存器的读写权限是超级用户模式可读写，用户模式访问都会导致非法指令异常。

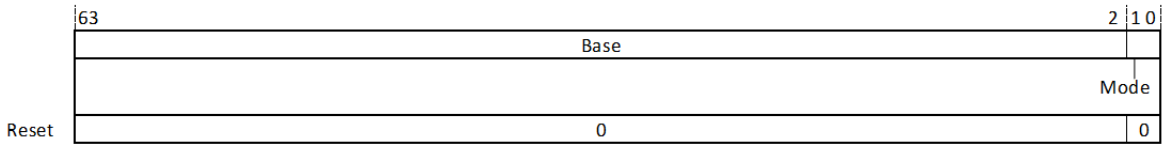


图 20.31: 超级用户模式向量基址寄存器（STVEC）

具体信息请参考*机器模式向量基址寄存器（MTVEC）*。

注意

- 在 SV39 的情况下，bit38 为符号位，mtvec 和 stvec 的高位需要按照 bit38 进行符号位拓展。
- 在 SV48 的情况下，bit47 为符号位，mtvec 和 stvec 的高位需要按照 bit47 进行符合位拓展。

20.2.1.4　超级用户模式计数器访问授权寄存器（SCOUNTEREN）

超级模式计数器访问授权寄存器（scounteren），用于授权用户模式是否可以访问用户模式计数器。

具体信息，请参考*超级用户模式计数器访问授权寄存器（scounteren）*。

20.2.1.5　超级用户模式计数器溢出寄存器（SCOUNTOVF）

S-mode 计数器溢出寄存器，来自于 sscofpmf 扩展。

具体信息请参考*超级用户模式计数器溢出寄存器（SCOUNTOVF）*。

20.2.2　超级用户模式异常处理寄存器组

20.2.2.1　超级用户模式异常临时数据备份寄存器（SSCRATCH）

超级用户模式异常临时数据备份寄存器（SSCRATCH）用于处理器在异常服务程序中备份临时数据。一般用来存储超级用户模式本地上下文空间的入口指针值。

该寄存器的位长是 64 位，寄存器的读写权限是超级用户模式可读写，用户模式访问会导致非法指令异常。

20.2.2.2　超级用户模式异常保留程序计数器寄存器（SEPC）

超级用户模式异常保留程序计数器（SEPC）用于存储程序从异常服务程序退出时的程序计数器值（即 PC 值）。C908X 支持 16 位宽指令，SEPC 的值以 16 位宽对齐，最低位为零。

该寄存器的位长是 64 位，寄存器的读写权限是超级用户模式可读写，用户模式访问会导致非法指令异常。

20.2.2.3　超级用户模式异常事件向量寄存器（SCAUSE）

超级用户模式异常事件向量寄存器（SCAUSE）用于保存触发异常的异常事件向量号，用于在异常服务程序中处理对应事件。

该寄存器的位长是 64 位，寄存器的读写权限是超级用户模式可读写，用户模式访问都会导致非法指令异常。

20.2.2.4　超级用户模式中断等待状态寄存器（SIP）

超级用户模式中断等待状态寄存器（SIP）用于保存处理器的中断等待状态。当处理器出现中断无法立即响应的情况时，SIP 寄存器中的对应位会被置位。

该寄存器的位长是 64 位，寄存器的读写权限是超级用户模式可读，写权限由对应位的 mideleg 决定，用户模式访问都会导致非法指令异常。

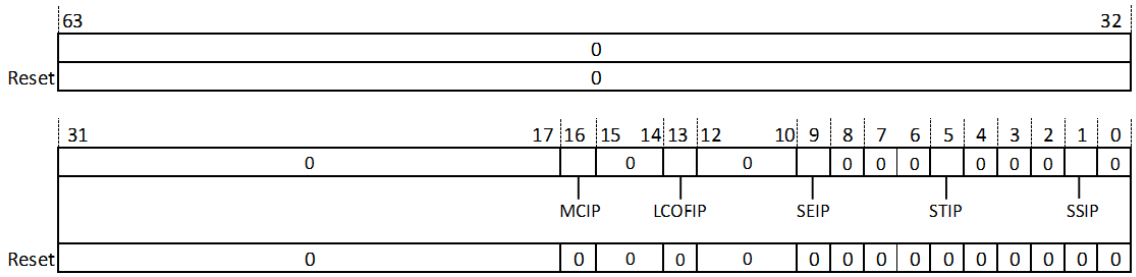


图 20.32: 超级用户模式中断等待状态寄存器（SIP）

具体信息请参考*机器模式中断等待状态寄存器（MIP）*。

20.2.3　超级用户模式计时器中断比较值寄存器组

20.2.3.1　超级用户模式计时器中断比较值寄存器（STIMECMP）

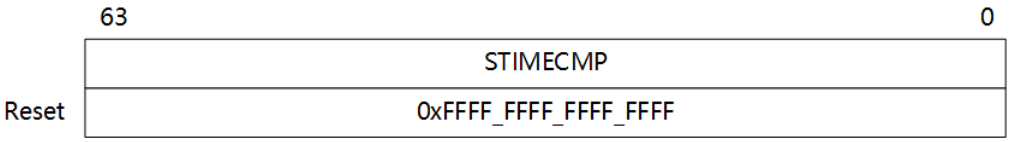


图 20.33: 超级用户模式计时器中断比较值寄存器（STIMECMP）

超级用户模式计时器中断比较值寄存器（STIMECMP）用于存储计时器比较值，STIMECMP 的值与系统计时器当前值作比较，确定是否产生超级用户模式计时器中断。当 STIMECMP 大于系统计时器的值时不产生中断；当 STIMECMP 小于或等于系统计时器的值，且机器模式环境配置寄存器（MENVCFG）的 STCE 字段为 1 时产生超级用户模式计时器中断。软件可通过改写 STIMECMP 的值来清除由 STIMECMP 造成的超级用户模式计时器中断。

该寄存器的位宽是 64 位。读写权限是在超级用户模式下，当机器模式环境配置寄存器（MENVCFG）的 STCE 字段为 1，且机器模式计数器访问授权寄存器（MCOUNTEREN）的 TM 字段为 1 时可读写，否则导致非法指令异常。用户模式访问会导致非法指令异常。

20.2.4　超级用户模式地址转换寄存器组

超级用户模式下，需要访问虚拟内存空间。超级用户模式地址转换寄存器（SATP）用于控制 MMU 单元的模式切换、硬件回填基地址和进程号。

20.2.4.1　超级用户模式地址转换寄存器（SATP）

超级用户模式地址转换寄存器（SATP）用于控制 MMU 单元的模式切换、硬件回填基地址和进程号。

该寄存器的位长是 64 位，寄存器的读写权限是超级用户模式可读写，用户模式访问会导致非法指令异常。

具体信息请参考*MMU 地址转换寄存器 (SATP)* 。

20.2.5　超级用户模式配置寄存器组

20.2.5.1　超级用户模式环境配置寄存器（SENVCFG）

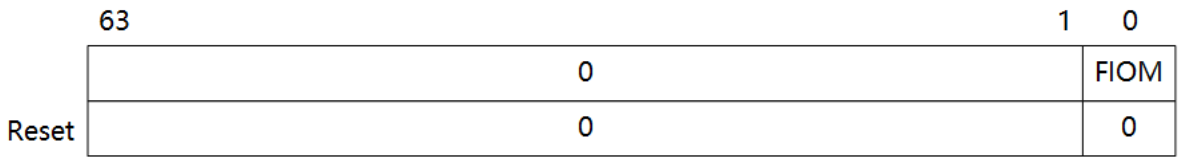


图 20.34: 超级用户模式环境配置寄存器（SENVCFG）

超级用户模式环境配置寄存器（SENVCFG）用于控制低于超级用户模式时执行环境中的特性。

该寄存器的位长是 64 位，寄存器的读写权限是超级用户模式可读写，用户模式访问都会导致非法指令异常。

FIOM - IO fence 包含内存访问

无论该位为何值，对 IO 的 fence 同步均会包括 RW 同步。

注：玄铁 C908X 中所有对 IO 的 fence 同步均会包括 RW 同步，所以不再受该位进行控制。对于 RV 标准的定义来说，senvcfg.FIOM = 1 时，低于超级用户模式的 IO 同步请求需要同时包括内存 RW 同步。

20.2.6 调试/追踪寄存器组

20.2.6.1 超级用户模式内容寄存器（SCONTEXT）

SCONTEXT[63:0] - 超级用户模式内容

超级用户模式软件可以写入特定的 context，结合调试/追踪触发数器据寄存器 3（TDATD3）中 SSELECT 与 SVALUE，可以控制触发仅在特定超级用户模式 context 下触发。

20.2.7 超级用户模式处理器控制和状态扩展寄存器组

20.2.7.1 超级用户模式扩展状态寄存器（SXSTATUS）

超级用户模式扩展状态寄存器（SXSTATUS）是机器模式扩展状态寄存器（MXSTATUS）的映射，具体信息请参考*机器模式扩展状态寄存器（MXSTATUS）*。

该寄存器的位长是 64 位，寄存器的读写权限是超级用户模式可读，只有 MM 位可写，用户模式访问会导致非法指令异常。

20.2.7.2 超级用户模式硬件控制寄存器（SHCR）

超级用户模式硬件控制寄存器（SHCR）是机器模式硬件控制寄存器（MHCR）的映射，具体信息请参考*机器模式硬件配置寄存器（MHCR）*。

该寄存器的位长是 64 位，寄存器的读写权限是超级用户模式可读，用户模式访问会导致非法指令异常。

20.2.7.3 超级用户模式 L2Cache ECC 寄存器（SCER2）

超级用户模式 L2Cache ECC 寄存器（SCER2）是机器模式 L2Cache ECC 寄存器（MCER2）的映射，具体信息请参考*机器模式 L2 Cache 错误控制寄存器（MCER2）*。

该寄存器的位长是 64 位，寄存器的读写权限是超级用户模式可读，用户模式访问会导致非法指令异常。

20.2.7.4　超级用户模式 L1Cache ECC 寄存器（SCER）

超级用户模式 L1Cache ECC 寄存器（SCER）是机器模式 L1 Cache ECC 寄存器（MCER）的映射，具体信息请参考*机器模式 L1 Cache ECC 寄存器（MCER）*。

该寄存器的位长是 64 位，寄存器的读写权限是超级用户模式可读，用户模式访问会导致非法指令异常。

20.2.7.5　超级用户模式禁止计数寄存器（SCOUNTINHIBIT）

SCOUNTINHIBIT 寄存器为 MCOUNTINHIBIT 寄存器的映射，当 mhpmcr.sce 为高时，超级用户模式可以通过控制 SCOUNTINHIBIT 位进行性能监控计数器控制。

20.2.7.6　超级用户性能监控控制寄存器（SHPMCR）

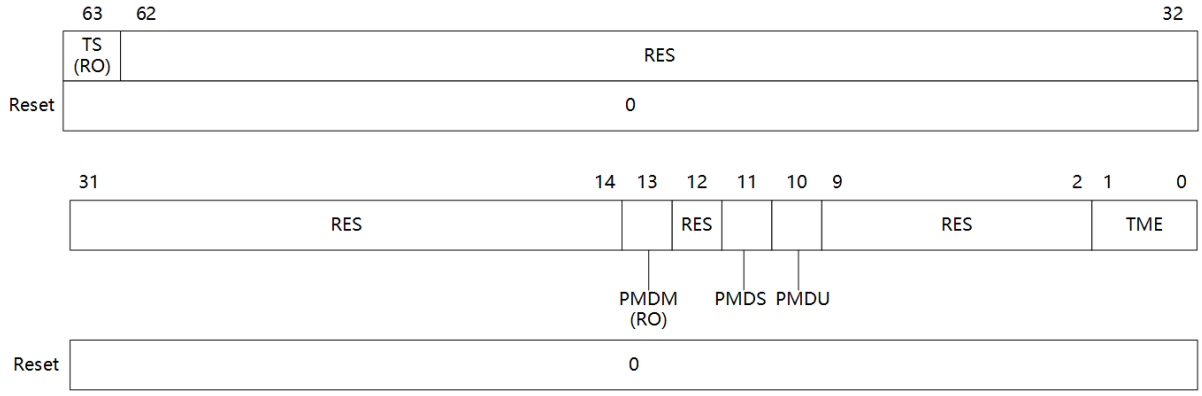


图 20.35: 超级用户性能监控控制寄存器（SHPMCR）

SHPMCR 为 MHPMCR 的映射，当 mhpmcr.sce 为高时，超级用户模式可以通过该寄存器对性能监控进行控制。

具体信息请参考*性能监控控制寄存器（MHPMCR）*。

20.2.7.7　超级用户性能监控起始/终止触发寄存器（SHPMSR/SHPMER）

SHPMSR/SHPMER 为 MHPMSR/MHPMER 的映射，当 MHPMCR.sce 为高时，超级用户模式可以通过该组寄存器控制触发的起止。处理器将使用虚拟地址作为匹配对象，若 MMU 不使能或为 M mode，使用物理地址作为匹配对象。

20.2.7.8　超级用户处理器末级 Cache partition ID 寄存器（SLLCPID）

当 mxstatus.bit[SPCE] 为 1，超级用户态可以通过配置 SLLCPID 来配置 L2 cache 的 patition 访问 id 号。

20.2.7.9　超级用户处理器 L2 Cache partition 访问配置寄存器（SL2WP）

当 mxstatus.bit[SPCE] 为 1，超级用户态可以通过配置 SL2WP 来配置 L2 cache 的 patition 访问。

20.2.7.10　S 态 L1 LD BUS ERR 地址寄存器（SBEADDR）

该寄存器的字段定义、功能与 MBEADDR 相同。

20.2.7.11　超级用户模式周期计数器（SCYCLE）

超级用户模式周期计数器（SCYCLE）用于存储处理器已经执行的周期数，当处理器处于执行状态（即非低功耗状态）下，SCYCLE 寄存器就会在每个执行周期自增计数。

周期计数器为 64 位，周期计数器会被 reset 清零。

具体信息请参考*事件计数器*。

20.2.7.12　超级用户模式退休指令计数器（SINSTRET）

超级用户模式退休指令计数器（SINSTRET）用于存储处理器已经退休的指令数，SINSTRET 寄存器会在每条指令退休时自增计数。

退休指令计数器为 64 位，退休指令计数器会被 reset 清零。

具体信息请参考*事件计数器*。

20.2.7.13　超级用户模式事件计数器（SHPMCOUNTERn）

超级用户模式事件计数器（SHPMCOUNTERn）是机器模式事件计数器（MHPMCOUNTERn）的映射。

具体信息请参考*事件计数器*。

20.2.8　超级用户模式 MMU 扩展寄存器

C908X 中 MMU 单元扩展了 MMU 相关寄存器，实现软件回填功能，软件可以直接对 TLB 进行读写等操作。

20.2.8.1　超级用户模式 MMU 控制寄存器（SMCIR）

该寄存器的位长是 64 位，寄存器的读写权限是超级用户模式可读，用户模式访问会导致非法指令异常。

具体信息请参考*MMU 控制寄存器 (SMCIR)*。

20.2.8.2　超级用户模式 MMU 控制寄存器（SMIR）

该寄存器的位长是 64 位，寄存器的读写权限是超级用户模式可读，用户模式访问会导致非法指令异常。

具体信息请参考*MMU Index 寄存器 (SMIR)* 。

20.2.8.3　超级用户模式 MMU 控制寄存器（SMEH）

该寄存器的位长是 64 位，寄存器的读写权限是超级用户模式可读，用户模式访问会导致非法指令异常。

具体信息请参考*MMU EntryHi 寄存器 (SMEH)* 。

20.2.8.4　超级用户模式 MMU 控制寄存器（SMEL）

该寄存器的位长是 64 位，寄存器的读写权限是超级用户模式可读，用户模式访问会导致非法指令异常。

具体信息请参考*MMU EntryLo 寄存器 (SMEL)* 。

20.3　附录 C-3 用户模式控制寄存器

用户模式控制寄存器按照功能主要分为浮点寄存器、计数器和矢量控制寄存器。

20.3.1　用户模式浮点控制寄存器组

20.3.1.1　浮点异常累积状态寄存器（FFLAGS）

浮点异常累积状态寄存器（FFLAGS）是浮点控制状态寄存器（FCSR）的异常累积域映射，具体信息请参考*浮点控制状态寄存器（FCSR）* 。

20.3.1.2　浮点动态舍入模式寄存器（FRM）

浮点动态舍入寄存器（FRM）是浮点控制状态寄存器（FCSR）的舍入模式域映射，具体信息请参考*浮点控制状态寄存器（FCSR）* 。

20.3.1.3　浮点控制状态寄存器（FCSR）

浮点控制状态寄存器（FCSR）用于记录浮点的异常累积和舍入模式控制。

该寄存器的位长是 64 位，该寄存器任何模式都可以读写。

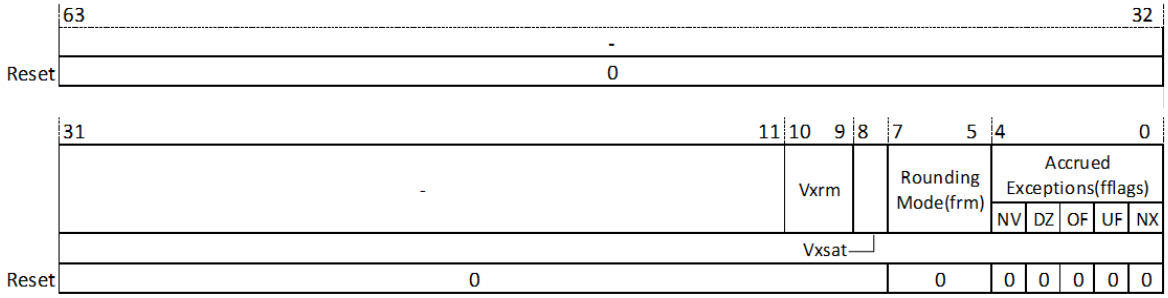


图 20.36: 浮点控制状态寄存器（FCSR）

NX-非精确异常：

- 当 NX=0 时，没有产生非精确异常。
- 当 NX=1 时，产生非精确异常。

UF-下溢异常：

- 当 UF=0 时，没有产生下溢异常。
- 当 UF=1 时，产生下溢异常。

OF-上溢异常：

- 当 OF=0 时，没有产生上溢异常。
- 当 OF=1 时，产生上溢异常。

DZ-除 0 异常：

- 当 DZ=0 时，没有产生除 0 异常。
- 当 DZ=1 时，产生除 0 异常。

NV-无效操作数异常：

- 当 NV=0 时，没有产生无效操作数异常。
- 当 NV=1 时，产生无效操作数异常。

RM-舍入模式：

- 当 RM=0 时，RNE 舍入模式，向最近偶数舍入。
- 当 RM=1 时，RTZ 舍入模式，向 0 舍入。
- 当 RM=2 时，RDN 舍入模式，向负无穷舍入。
- 当 RM=3 时，RUP 舍入模式，向正无穷舍入。
- 当 RM=4 时，RMM 舍入模式，向最近舍入。

VXSAT-矢量溢出标志位：

VXSAT 对应位的映射。

VXRM-矢量舍入模式位：

VXRM 对应位的映射。

20.3.2　用户模式计数/计时寄存器组

20.3.2.1　用户模式周期计数器（CYCLE）

用户模式周期计数器（CYCLE）用于存储处理器已经执行的周期数，当处理器处于执行状态（即非低功耗状态）下，CYCLE 寄存器就会在每个执行周期自增计数。

周期计数器为 64 位，周期计数器会被 reset 清零。

具体信息请参考*事件计数器*。

20.3.2.2　用户模式时间计数器（TIME）

用户模式时间计数器（TIME）是 MTIME 的只读映射。

具体信息请参考*事件计数器*。

20.3.2.3　用户模式退休指令计数器（INSTRET）

用户模式退休指令计数器（MINSTRET）用于存储处理器已经退休的指令数，INSTRET 寄存器会在每条指令退休时自增计数。

退休指令计数器为 64 位，退休指令计数器会被 reset 清零。

具体信息请参考*事件计数器*。

20.3.2.4　用户模式事件计数器（HPMCOUNTERn）

用户模式事件计数器（HPMCOUNTERn）是机器模式事件计数器（MHPMCOUNTERn）的映射。

具体信息请参考*事件计数器*。

20.3.2.5　用户模式周期计数器高位（CYCLEH）

用户模式周期计数器（CYCLE）用于存储处理器已经执行的周期数高 32 位，当处理器处于执行状态（即非低功耗状态）下，CYCLE 寄存器就会在每个执行周期自增计数。

具体信息请参考*事件计数器*。

20.3.2.6　用户模式时间计数器高位（TIMEH）

用户模式时间计数器（TIME）是 MTIME 的高 32 位只读映射。

具体信息请参考*事件计数器*。

20.3.2.7 用户模式退休指令计数器高位（INSTRETH）

用户模式退休指令计数器高位（INSTRETH）用于存储处理器已经退休的指令数高 32 位，INSTRET 寄存器会在每条指令退休时自增计数。

具体信息请参考*事件计数器*。

20.3.2.8 用户模式事件计数器高位（HPMCOUNTERnH）

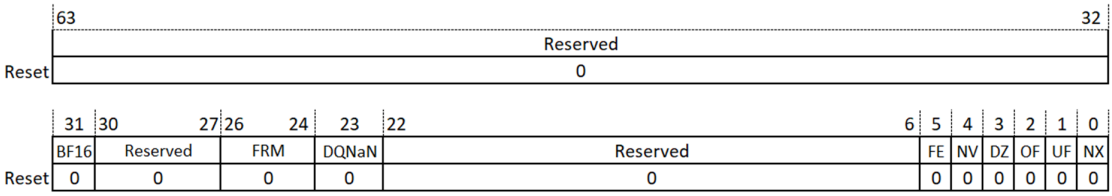
用户模式事件计数器高位（HPMCOUNTERnH）是机器模式事件计数器（MHPMCOUNTERn）的高 32 位映射。

具体信息请参考*事件计数器*。

20.3.3 用户模式扩展浮点控制寄存器组

20.3.3.1 用户模式浮点扩展控制寄存器（FXCR）

用户模式浮点扩展控制寄存器（FXCR）用于浮点扩展功能开关和浮点异常累积位。



	63 … 32
	Reserved
Reset	0

	31	30 … 27	26 … 24	23	22 … 6	5	4	3	2	1	0
	BF16	Reserved	FRM	DQNaN	Reserved	FE	NV	DZ	OF	UF	NX
Reset	0	0	0	0	0	0	0	0	0	0	0

图 20.37: 浮点扩展控制寄存器（FXCR）

NX-非精确异常：

FCSR 对应位的映射。

UF-下溢异常：

FCSR 对应位的映射。

OF-上溢异常：

FCSR 对应位的映射。

DZ-除 0 异常：

FCSR 对应位的映射。

NV-无效操作数异常：

FCSR 对应位的映射。

FE-浮点异常累积位：

当有任何一个浮点异常发生时，该位将被置为 1。

DQNaN-输出 QNaN 模式位：

当 DQNaN 为 0 时，计算输出的 QNaN 值为默认的固定值。（C908X DQNaN 位固定接 0）

当 DQNaN 为 1 时，计算输出的 QNaN 值根 IEEE754 标准一致。（C908X DQNaN 位不支持接 1）

FRM-舍入模式：

FCSR 对应位的映射。

BF16-半精度运算格式：

当 BF16 为 0 时，当前半精度运算处于 FP16 格式；

当 BF16 为 1 时, 当前半精度运算处于 BF16 格式。

20.3.4 矢量扩展寄存器组

20.3.4.1 矢量起始位置寄存器（VSTART）

矢量起始位置寄存器指定了执行矢量指令时起始元素位置，每条矢量指令执行后 VSTART 会被清零。

20.3.4.2 定点溢出标志位寄存器（VXSAT）

定点溢出标志位寄存器表示是否有定点指令产生溢出结果。

20.3.4.3 定点舍入模式寄存器（VXRM）

定点舍入模式寄存器指定了定点指令采用的舍入模式。

	63	2	1 0
	Base		RM
Reset	0		0

图 20.38: 定点舍入模式寄存器（VXRM）

RM-定点舍入模式：

- 当 RM=0 时，RNU 舍入模式，向大数舍入。
- 当 RM=1 时，RNE 舍入模式，向偶数舍入。
- 当 RM=2 时，RDN 舍入模式，向零舍入。
- 当 RM=3 时，ROD 舍入模式，向奇数舍入。

20.3.4.4 矢量长度寄存器（VL）

矢量长度寄存器指定了矢量指令更新目的寄存器的范围，矢量指令更新目的寄存器中元素序号小于 VL 的元素，清零目的寄存器中元素序号大于等于 VL 的元素。特别的，当 VSTART >= VL 或 VL 为 0 时，目的寄存器的所有元素不被更新。

该寄存器是任意模式下的只读寄存器，但是 vsetvli、vsetvl 以及 fault-only-first 指令能够更新该寄存器的值。

20.3.4.5　矢量数据类型寄存器（VTYPE）

VTYPE 寄存器指定了矢量寄存器组的数据类型以及矢量寄存器的元素组成。

	XLEN-1	XLEN-2 … 8	7	6	5 … 3	2 … 0
	VILL	Reserved	VMA	VTA	VSEW	VMUL
Reset	0	0	0	0	0	0

图 20.39: 定点类型寄存器（VTYPE）

该寄存器是任意模式下的只读寄存器，但是通过 vsetvli 和 vsetvl 指令能够更新该寄存器的值。

VILL-非法操作标志位：

当 vsetvli/vsetvl /vsetivli 指令以 C908X 不支持的值更新 VTYPE 寄存器时该位置起，否则为 0。当该位置起时，执行依赖 vtype 的矢量指令产生非法指令异常。

注意：不依赖 vtype 的矢量指令包括 vset{i}vl{i} 以及整体寄存器组的 load/store/move 相关指令。

VMA-mask element 配置位

VTA-tail element 配置位

VTA 和 VMA 在矢量指令执行期间修改目的尾部元素和目的被掩码的元素的行为。

VTA 修改目的尾部元素在矢量指令执行期间的行为。

- VTA=0 表示，目的尾部元素会被设置为 undisturbed 状态。
- VTA=1 表示，目的尾部元素会被设置为 agnostic 状态。

VMA 修改目的被掩码的元素在矢量指令执行期间的行为。

- VMA=0 表示，目的被掩码的元素会被设置为 undisturbed 状态。
- VMA=1 表示，目的被掩码的元素会被设置为 agnostic 状态。

VSEW-矢量元素位宽设置位：

VSEW 决定了矢量元素位宽（SEW），C908X 支持的矢量元素位宽如 表 20.3 所示。

表 20.3: 矢量元素位宽

VSEW[2:0]			元素位宽
0	0	0	8
0	0	1	16
0	1	0	32
0	1	1	64

当 VSEW 为其他值时，C908X 执行矢量指令产生非法指令异常。

VLMUL-矢量寄存器分组设置位：

多个矢量寄存器可以构成矢量寄存器组，矢量指令作用于寄存器组中的所有矢量寄存器。VLMUL 决定了矢量寄存器组中矢量寄存器的数量（LMUL），如 表 20.4 所示。

表 20.4: 矢量寄存器组中寄存器数量

VLMUL[2:0]			LMUL
1	0	1	1/8
1	1	0	1/4
1	1	1	1/2
0	0	0	1
0	0	1	2
0	1	0	4
0	1	1	8

20.3.4.6　矢量位宽（单位：字节）寄存器（VLENB）

VLENB 寄存器指示处理器的矢量位宽，以字节为单位。

当 C908X 的矢量位宽配为 128 位时（即 VLEN=128），VLENB = 128 / 8 = 16。

20.4　附录 C-4 C908X VPU 访存类异常寄存器

20.4.1　2D saturation 寄存器（UTNMODE）

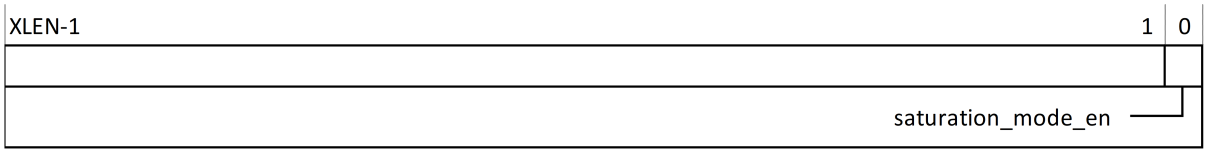


图 20.40: 2D saturation 寄存器（UTNMODE）

saturation_mode_en:

saturation_mode_en 为二维 reduce 指令和 fp8 的 cvt 指令的饱和模式使能开关，默认值为 1，即默认打开饱和模式。

20.4.2　状态地址寄存器（MTNSR）

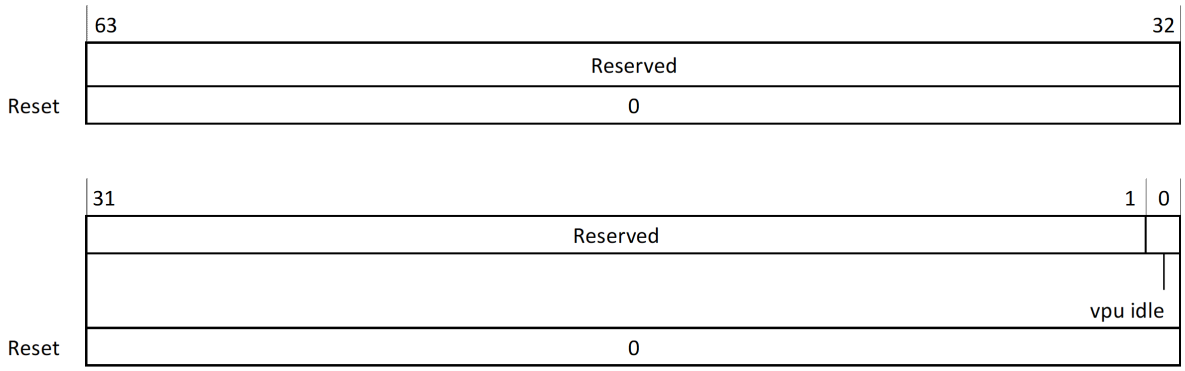


图 20.41: 状态地址寄存器（MTNSR）

vpu_idle:

1 表示 vpu idle

20.4.3　Debug trigger pc 扩展寄存器（mtndebugpc）

发生 Debug va 或 data nan trigger 时，需要将 pc 信息保存在 CSR 寄存器中，供软件进行读取。该寄存器调试模式只读。

	63 MAW*	MAW-1 0
	Reserved	mtndebugpc
Reset	0	0

图 20.42: Debug trigger pc 扩展寄存器（mtndebugpc）

> **备注**
>
> * MAW，虚拟地址位宽，在 SV39 时为 39；在 SV48 时为 48。

mtndebugpc:

发生 Debug va 或 data nan trigger 的 pc 信息。

20.4.4　fast memory 地址范围扩展寄存器（mtnfastmba）

为支持 fast memory 访问，玄铁 C908X VPU 额外增加寄存器 MTNFASTMBA 用于标记 FAST MEMORY 的地址范围，寄存器在配置宏 *XX_FAST_MEM* 时有效。

表 20.5: fast memory 地址范围扩展寄存器 (mtnfastmba) 描述

bit	名称	读写权限	描述
63	mtnfasten	机器模式读写	fast memory 使能位，为 1 时表示有 fast memory
62	mpage2men	机器模式读写	2m 即以上页优化使能位，为 1 时表示优化开启
61:PAB*	Reserved	机器模式只读	保留位
PAB-1:0	mtnfastmba	机器模式读写	fast memory 空间 base 地址（PA）

*：PA40 时，PAB = 28；PA48 时，PAB = 36。

矢量、标量访存指令，若基地址位于配置的范围之间，则通过 fast memory 访问外部 SRAM，具体说明如下：

- 配置 mtnfastmba，根据 PA40/PA48 对高位 mask 为 0
- 配置的数据中，低位有 N 个连续的 1，则表示 2^N 个 4 KB 的 size

如下 表 20.6 所示，以 RV64 系统下，PA48 mtnfastmba 地址范围为例：

表 20.6: RV64 系统下 PA48 mtnfastmba 地址范围

Size	mtnfastmba X 表示十六进制数，x 表示二进制数	Range
4KB	64'b1xxx_XXXX_XXXX_xxxx_xxxx_xxx0	{mtnfastmba[36:1]，12{1'b0}} ~ {mtnfastmba[36:1]，12{1'b1}}
8KB	64'b1xxx_XXXX_XXXX_xxxx_xxxx_xx01	{mtnfastmba[36:2]，13{1'b0}} ~ {mtnfastmba[36:2]，13{1'b1}}
16KB	64'b1xxx_XXXX_XXXX_xxxx_xxxx_x011	{mtnfastmba[36:3]，14{1'b0}} ~ {mtnfastmba[36:3]，14{1'b1}}
32KB	64'b1xxx_XXXX_XXXX_xxxx_xxxx_0111	{mtnfastmba[36:4]，15{1'b0}} ~ {mtnfastmba[36:4]，15{1'b1}}
64KB	64'b1xxx_XXXX_XXXX_xxxx_xxx0_1111	{mtnfastmba[36:5]，16{1'b0}} ~ {mtnfastmba[36:5]，16{1'b1}}
128KB	64'b1xxx_XXXX_XXXX_xxxx_xx01_1111	{mtnfastmba[36:6]，17{1'b0}} ~ {mtnfastmba[36:6]，17{1'b1}}
256KB	64'b1xxx_XXXX_XXXX_xxxx_x011_1111	{mtnfastmba[36:7]，18{1'b0}} ~ {mtnfastmba[36:7]，18{1'b1}}
512KB	64'b1xxx_XXXX_XXXX_xxxx_0111_1111	{mtnfastmba[36:8]，19{1'b0}} ~ {mtnfastmba[36:8]，19{1'b1}}
1MB	64'b1xxx_XXXX_XXXX_xxx0_1111_1111	{mtnfastmba[36:9]，20{1'b0}} ~ {mtnfastmba[36:9]，20{1'b1}}
2MB	64'b1xxx_XXXX_XXXX_xx01_1111_1111	{mtnfastmba[36:10]，21{1'b0}} ~ {mtnfastmba[36:10]，21{1'b1}}

续下页

表 20.6 - 接上页

Size	mtnfastmba X 表示十六进制数，x 表示二进制数	Range
4MB	64'b1xxx_XXXX_XXXX_x011_1111_1111	{mtnfastmba[36:11]，22{1'b0}} ~ {mtnfastmba[36:11]，22{1'b1}}
8MB	64'b1xxx_XXXX_XXXX_0111_1111_1111	{mtnfastmba[36:12]，23{1'b0}} ~ {mtnfastmba[36:12]，23{1'b1}}
16MB	64'b1xxx_XXXX_XXXxxx0_1111_1111_1111	{mtnfastmba[36:13]，24{1'b0}} ~ {mtnfastmba[36:13]，24{1'b1}}
32MB	64'b1xxx_XXXX_XXXxx01_1111_1111_1111	{mtnfastmba[36:14]，25{1'b0}} ~ {mtnfastmba[36:14]，25{1'b1}}
64MB	64'b1xxx_XXXX_XXXx011_1111_1111_1111	{mtnfastmba[36:15]，26{1'b0}} ~ {mtnfastmba[36:15]，26{1'b1}}
128MB	64'b1xxx_XXXX_XXX0111_1111_1111_1111	{mtnfastmba[36:16]，27{1'b0}} ~ {mtnfastmba[36:16]，27{1'b1}}
256MB	64'b1xxx_XXXX_XXxxx01111_1111_1111_1111	{mtnfastmba[36:17]，28{1'b0}} ~ {mtnfastmba[36:17]，28{1'b1}}
512MB	64'b1xxx_XXXX_XXxx011111_1111_1111_1111	{mtnfastmba[36:18]，29{1'b0}} ~ {mtnfastmba[36:18]，29{1'b1}}

同时需要满足以下要求：

同时需要满足以下要求：

1. 要求外部 SRAM 地址空间为 Non-Cacheable 属性且不存在单条指令既访问 SRAM 也访问非 SRAM 空间的情况。

   1）如果出现单条指令即访问了 fast memory 空间又访问了非 fast memory 空间，报同步异常

   2）只要地址区间符合要求，对地址属性信息将强制为 Weak-Order、Non-Cacheable 属性

2. 如果之前这段区间未配置为 fast_memory 区间，软件配置 fast_memory 区间后，之前对该区域 store 的数据将不能被系统可见，直到该区域空间又配置为非 fast_memory 地址空间。

3. 对于 Cacheable 请求，如果矢量访存指令不是通过 fast memory 接口访问外部 SRAM，而是通过普通接口去访问 L2 Cache 或外部 SRAM，则需要使用 sync.s 指令来保证矢量 load/store 操作的执行顺序。

20.4.5　Low_power 寄存器（mtnlowpower）

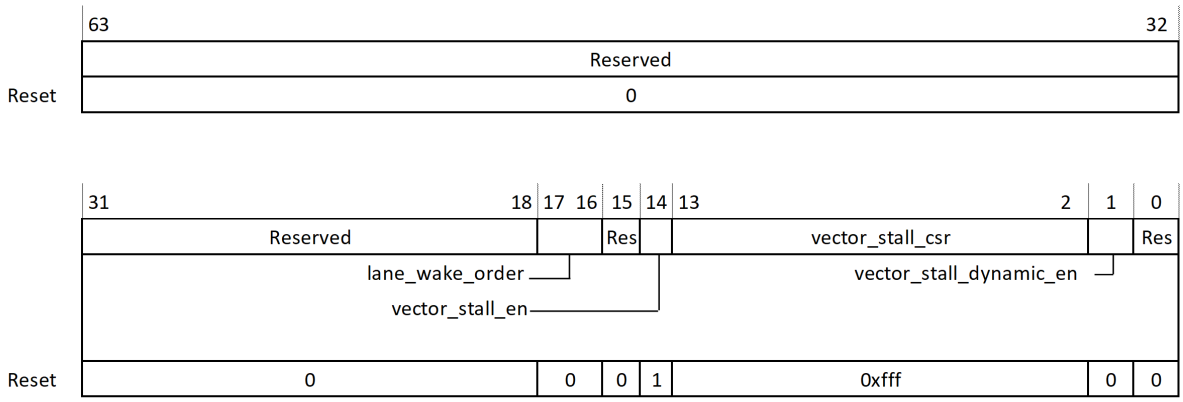


图 20.43: Low_power 寄存器（mtnlowpower）

lane_wake_order：

从 stall 恢复到 active 状态下，每个 cycle 最多恢复工作的 lane_number，00 ~ 11 依次表示最多恢复工作状态的 lane 数量为 1，2，4 ~ 8。

vector_stall_en：

VID 级反压前级流水线，vector stall 为高，一直挡住指令进入 C908X VPU 内部执行；vetor stall 为低，指令继续进入 C908X VPU VID 后级流水线执行，优先级比 vector_stall_dynamic_en 使能的模式高，为默认模式。

1: 打开。（C908X 默认打开）

0: 关闭

vector_stall_csr:

每隔多少个 cycle stall，允许 VID 译码一条指令进入后级流水线

默认值：0xfff

vector_stall_dynamic_en:

使能 VID 级动态调限制令译码带宽，降低功耗

1: 支持

0: 关闭。默认关闭

20.5　附录 C-5 调试寄存器组

20.5.1　调试模式寄存器组

20.5.1.1　调试模式控制与状态寄存器（DCSR）

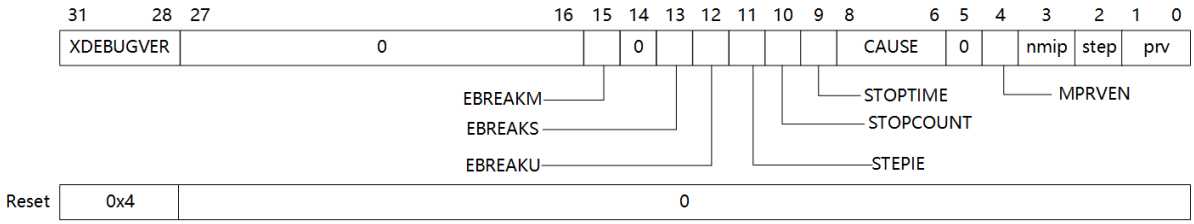


图 20.44: 调试模式控制与状态寄存器（DCSR）

XDEBUGVER：

- 0：没有调试系统
- 4：有调试系统，调试系统支持 riscv debug spec v0.13.2
- 15：有调试系统，调试系统不支持 riscv debug spec v0.13.2

EBREAKM：

- 0：机器模式下执行 ebreak 指令产生 breakpoint 异常
- 1：机器模式下执行 ebreak 指令进入调试模式

EBREAKS：

- 0：超级用户模式下执行 ebreak 指令产生 breakpoint 异常
- 1：超级用户模式模式下执行 ebreak 指令进入调试模式

EBREAKU：

- 0：用户模式下执行 ebreak 指令产生 breakpoint 异常
- 1：用户模式下执行 ebreak 指令进入调试模式

STEPIE：

- 0：在单步调试的过程中不响应中断
- 1：在单步调试的过程中响应中断

STOPCOUNT：

- 0：调试模式下性能监测计数器正常计数
- 1：调试模式下性能监测计数器不计数，包括 MCYCLE 和 MINSTRET

STOPTIME：

- 0：调试模式下处理器核私有的时钟计数器正常计数
- 1：调试模式下处理器核私有的时钟计数器不计数

CAUSE：

- 1: 表示进入调试模式的原因是 ebreak 指令的执行

- 2: 表示进入调试模式的原因是触发器的触发
- 3: 表示进入调试模式的原因是同步调试请求
- 4: 表示进入调试模式的原因是单步调试请求
- 5: 表示进入调试模式的原因是复位调试请求

MPRVEN：

- 0：MSTATUS 寄存器中 MPRV 字段在调试模式失效
- 1：MSTATUS 寄存器中 MPRV 字段在调试模式有效，处理器根据 MPRV 字段和 MPP 字段的设置处理访存指令的地址翻译与保护

NMIP：

- 0：表示处理器中不存在 NMI 中断
- 1：表示处理器中出现 NMI 中断

STEP：

- 0：无单步调试
- 1：发起单步调试模式

PRV：

进入调试模式时，将处理器的特权模式存入 PRV；退出调试模式后根据 PRV 字段设置处理器所处的特权模式

20.5.1.2　调试模式程序计数器（DPC）

DPC[63:0]：进入调试模式时，硬件将下一条指令的地址写入 DPC；退出调试模式时，处理器从 DPC 内保存的地址开始取指执行。

20.5.1.3　调试模式临时数据备份寄存器 0（DSCRATCH0）

DSCRATCH0[63:0]：硬件上用于调试系统与处理器核之间的数据交换。

20.5.1.4　调试模式临时数据备份寄存器 1（DSCRATCH1）

DSCRATCH1[63:0]：硬件上用于调试系统与处理器核之间的数据交换。

20.5.2　调试扩展寄存器组

20.5.2.1　玄铁调试原因寄存器（MHALTCAUSE）

	63 … 4	3 … 0
	0	MHALTCAUSE
Reset	0	0

图 20.45: 玄铁调试原因寄存器（MHALTCAUSE）

MHALTCAUSE：指示进入调试模式的原因

- 当 MHALTCAUSE 为 1 时：表示进入调试模式的原因是 ebreak 指令的执行
- 当 MHALTCAUSE 为 2 时：表示进入调试模式的原因是触发器的触发
- 当 MHALTCAUSE 为 3 时：表示进入调试模式的原因是同步调试请求
- 当 MHALTCAUSE 为 4 时：表示进入调试模式的原因是单步调试请求
- 当 MHALTCAUSE 为 5 时：表示进入调试模式的原因是复位调试请求
- 当 MHALTCAUSE 为 8 时：表示进入调试模式的原因是异步调试请求

20.5.2.2　玄铁调试信息寄存器（MDBGINFO）

MDBGINFO[63:0]：用于异步调试时记录处理器核的调试信息。

20.5.2.3　玄铁分支目标地址记录寄存器（MPCFIFO）

MPCFIFO[63:0]：记录分支/跳转指令的目标地址。

20.5.2.4　玄铁调试信息寄存器 2（MDBGINFO2）

MDBGINFO2[63:0]：用于异步调试时记录 CIU 和 L2-Cache 的调试信息。

20.5.3　调试/追踪寄存器组（与调试模式共享）

20.5.3.1　调试/追踪触发寄存器选择寄存器（TSELECT）

调试/追踪触发器选择寄存器（TSELECT）用于在多个触发器（trigger）之间选中一个，以进行下一步对该触发器的寄存器读写。

	63 … 4	3 … 0
	0	SELECT
Reset	0	0

图 20.46: 调试/追踪触发寄存器选择寄存器（TSELECT）

SELECT - 调试/追踪触发器选择

记录目前选中的调试/追踪触发器编号。例如，要配置 2 号触发器时，SELECT 写入 0x2。

20.5.3.2 调试/追踪触发数器据寄存器 1（TDATA1）

	63 60	59	58 0
	TYPE	DEMODE	DATA
Reset	0x2/0x5	0	0

图 20.47: 调试/追踪触发数器据寄存器 1（TDATA1）

TYPE - 调试/追踪触发器类型选择

- TYPE = 2，表示当前触发器类型为 mcontrol；
- TYPE = 3，表示当前触发器类型为 icount；
- TYPE = 4，表示当前触发器类型为 itrigger；
- TYPE = 5，表示当前触发器类型为 etrigger。

C908X 支持两种类型的触发器：

1. mcontrol 触发器，TYPE 字段被硬连线为 0x2；
2. itrigger/etrigger/icount 可配置触发器，TYPE 字段可配置为 0x3、0x4、0x5，复位值为 0x5；

DEMODE - 控制调试/追踪触发数器据寄存器 123（TDATA1/TDATA2/TDATA3）的写权限

- 当 DEMODE 为 0 时，调试模式和机器模式可以写入调试/追踪触发数器据寄存器；
- 当 DEMODE 为 1 时，仅调试模式可以写入调试/追踪触发数器据寄存器。

DATA - 调试/追踪触发数器据寄存器 1 控制

DATA 字段的具体含义由 TYPE 决定（具体描述建议参考 RISC-V debug spec v0.13.2 中 5.2 小节）。

20.5.3.3 调试/追踪触发数器据寄存器 2（TDATA2）

	63 0
	DATA
Reset	0

图 20.48: 调试/追踪触发数器据寄存器 2（TDATA2）

DATA - 调试/追踪触发数器据寄存器 2 数据

用于设置触发值，具体含义由 TDATA1 中 TYPE 字段决定。

20.5.3.4　调试/追踪触发数器据寄存器 3（TDATA3）

	63 51	50 49	36 35	2 1 0
	MVALUE	MSELECT	0	SVALUE	SSELECT
Reset	0	0	0	0	0

图 20.49: 调试/追踪触发数器据寄存器 3（TDATA3）

MVALUE - 触发器机器模式内容匹配数据

用于设置希望匹配的机器模式内容数值。

MSELECT - 触发器机器模式内容匹配控制

- 当 MSELECT 为 0 时：关闭触发器机器模式内容匹配
- 当 MSELECT 为 1 时：当机器模式内容寄存器（MCONTEXT）与触发器机器模式内容匹配数据（MVALUE）相等时，该触发器可以对处理器信息进行匹配。

SVALUE - 触发器超级用户模式内容匹配数据

用于设置希望匹配的超级用户模式内容数值。

SSELECT - 触发器超级用户模式内容匹配控制

- 当 SSELECT 为 0 时：关闭触发器超级用户模式内容匹配
- 当 SSELECT 为 1 时：当超级用户模式内容寄存器（SCONTEXT）与触发器超级用户模式内容匹配数据（SVALUE）相等时，该触发器可以对处理器信息进行匹配。
- 当 SSELECT 为 2 时：当 satp 寄存器中 ASID 字段的值与触发器超级用户模式内容匹配数据（SVALUE）相等时，该触发器可以对处理器信息进行匹配。

20.5.3.5　调试/追踪触发器信息寄存器（TINFO）

	63 16	15 0
	0	INFO
Reset	0	0x100 or 0x111000

图 20.50: 调试/追踪触发器信息寄存器（TINFO）

INFO - 表示该触发器支持的类型

bit[n] 为 1 代表该触发器 TDATA1 的 TYPE 字段可配置为 n。

C908X 支持两种类型的触发器：

1. mcontrol 触发器，INFO 被硬连线为 0x100，表示 TDATA1 的 TYPE 字段只能为 2；
2. itrigger/etrigger/icount 可配置触发器，INFO 被硬连线为 0x111000，表示 TYPE 字段可配置为 0x3、0x4、0x5。

20.5.3.6　调试/追踪触发器控制寄存器（TCONTROL）

	63 … 9	8 … 7	6 … 4	3	2 … 0
	0	MPTE	0	MTE	0
Reset	0	0	0	0	0

图 20.51: 调试/追踪触发器控制寄存器（TCONTROL）

MTE - 机器模式触发器使能控制

- 当 MTE 为 0 时：触发行为是产生 breakpoint 异常的触发器不能在机器模式触发；
- 当 MTE 为 1 时：触发器可以在机器模式触发。

进入机器模式异常/中断处理程序时，硬件将 MTE 置 0；从机器模式异常/中断处理程序返回时，硬件将 MTE 置为 MPTE 的值。

MPTE - 机器模式触发器使能备份

进入机器模式异常/中断处理程序时，硬件将 MTE 的值存进 MPTE 中。

20.5.3.7　机器模式内容寄存器（MCONTEXT）

	63 … 0
	MCONTEXT
Reset	0

图 20.52: 机器模式内容寄存器（MCONTEXT）

MCONTEXT - 机器模式内容

机器模式软件可以写入特定的 context，结合调试/追踪触发数器据寄存器 3（TDATA3）中 MSELECT 与 MVALUE，可以控制触发仅在特定机器模式 context 下触发。

20.6　附录 C-6 Trace 寄存器组

20.6.1　encoder 控制寄存器组

20.6.1.1　trtecontrol(0x000)

表 20.7: trtecontrol 寄存器描述

bit	Field	Description	RW	Reset
0	trteactive	TE 的主使能开关，置 0 可以复位 TE 内所有寄存器，用于 TE 的时钟门控。当 teActive 为 0 时，其他 TE 寄存器不可访问。	RW	0
1	trteenable	为 1 时表示 TE 使能。允许 trteinsttracing 开始 trace 或者关闭 trace。写 0 后停止接收 rtu 信息，待内部 trace 流传输完毕后，trteenable 读回值为 0	RW	0
2	trteinsttracing	为 1 时，trace 开始产生。通过软件写或者受 trigger 控制。当 trteenable=1 时，该字段才能成功写 1	RW	0
3	trteempty	当产生的 trace 都发送完毕时，读出值为 1	R	1
6:4	trteinstmode	指令 trace 产生的模式 0：指令 trace 关闭 1-2：reserve 3：每条 taken 的 conditional branch 均产生 msg 4-5：reserve 6：产生 branch history trace，即每条 branch 指令产生 1bit 跳转历史表 7：reserved	WARL	0
8:7	-	reserved	WARL	0
9	trtecontext	为 1 时，在 scontext、v 或者 prv 变化的时候发送 ownership message 传输 cpu 的 context 信息。并且在每条 syn message 后传输 ownership message。	WARL	0
10	-	reserved	WARL	0
11	trteinsttrigen	为 1 时，trigger 产生的 trace 行为将被响应	WARL	0
12	trteInstStallOrOverflow	在以下两种情况下硬件自动置 1： 1. 产生了 overflow message 2.TE 将 core stall 在 TE Reset 或者 teenable 被置 1 时清 0。写 1 清 0	RWC1	0
13	trteInstStallEn	0：当 TE 中有 message 溢出时，会丢掉部分信息，并在重新开始时发送 error msg 和 restart msg 1：当 TE 中即将溢出时，将 core stall 直到 TE 能够处理溢出	WARL	0
14	-	reserved	WARL	0

续下页

表 20.7 - 接上页

bit	Field	Description	RW	Reset
15	trteInhibitSrc	0：每一条 trace message 都要携带 source 信息（teinstfeatures.trtesrcbits） 1：trace message 中不携带 source 信息	WARL	0
17:16	trteSyncMode	选择周期性 sync 机制。以下机制中至少实现一个 0：off 1：对 trace message 做计数 2：对 clock cycle 做计数 3：对指令以半字为单位计数	WARL	0
19:18	Reserved	-	-	0
23:20	trteSyncMax	指定两个 sync message 之间的最大间隔。在 sync 计数器达到 $2^{teSyncMax+4}$ 时，产生 sync message。当由于其他原因发送了 sync message 后，sync 计数器要清 0. syncmode=2 时，trteSyncMax 最小配置为 4。	WARL	0
26:24	trteFormat	trace message 格式 0：E trace spec 中定义的 trace 格式 1：使用 MDO + 2 MSEO 的 nexus trace 格式 2-6：reserve 7：厂商自定义格式 tdt trace 系统此字段绑 1	WARL	1

20.6.1.2 trteimpl(0x004)

表 20.8: trteimpl 寄存器描述

bit	Field	Description	RW	Reset
3:0	trtevermajor	major version，用于软件区分不兼容的 feature，当做出了不兼容以前版本的修改，需修改 major version 0：表示兼容以前的 spec 版本 1：表示兼容当前 spec 版本	R	1
7:4	trteverminor	minor version，用于区分软件兼容的 feature，当做出了某些新扩展 feature，与以前版本兼容，需修改 minor version	R	0
11-8	trTeCompType	trace 组件类型，encoder 类型绑 0x1	R	0x1
12-15	-	reserved for standard	-	-

续下页

表 20.8 - 接上页

bit	Field	Description	RW	Reset
19-16	trTeProtocolMajor	protocol major version。 用于区分不兼容的协议修改	R	1
23-20	trTeProtocolMinor	protocol minor version。 用于区分兼容的协议修改	R	0
31:24	-	reserved for vendor	-	-

20.6.1.3 trteinstfeatures(0x008)

表 20.9: trteinstfeatures 寄存器描述

bit	Field	Description	RW	Reset
0	trteInstNoAddrDiff	为 1 时，发送完整地址，不发送地址差值，硬件绑 0.	WARL	0
1	trteInstNoTrapAddr	为 1 时，exception packet 中只发送异常 cause，不发送异常地址，硬件绑 0。	WARL	0
2	trteInstEnSequentialJump	为 1 时，连续可推测的 jump 不需要发送 message。 可推测的 jump 指令指前面一条指令是写立即数的 lui 或 auipc，后面的 indirect jump 的源寄存器使用前一条的目的寄存器。硬件绑 0.	WARL	0
3	trTeInstEnImplicitReturn	为 1 时。正常的 retu rn 指令不需要输出 tdt trace 系统不支持，硬件绑 0.	WARL	0
4	trteInstEnBranchPrediction	为 1 时，启动分支预测模式。预测正确，不输出 branch 信息；预测错误，输出 branch 信息，并且更新分支预测器 tdt trace 系统不支持，硬件绑 0.	WARL	0
5	trteInstEnJumpTargetCache	为 1 时，启动 jump 目标 cache 。用 jump 指令 pc 去索引全相联 cache，如果命中，不输出 jump 信息；不命中，输出 jump 信息，并且更新 cache tdt trace 系统不支持，硬件绑 0.	WARL	0
7:6	trTeInstImplicitReturnMode	如何处理 retur n 指令，trTeInstE nImplic-itReturn= 1 时有效，硬件绑 0	WARL	0
8	trTeInstEnRepeatedHistory	允许 repeated branch hist ory 探测，硬件绑 0	WARL	0

续下页

表 20.9 - 接上页

bit	Field	Description	RW	Reset
9	trTeInstEnAllJumps	对于所有的跳转指令（包括无条件的，可推测的）均输出 direct branch m sg 或记录在 histor y 表项中，硬件绑 0	WARL	0
10	trTeInstExtendAddrMSB	置 1 后，将虚拟地址最高位做符号位扩展优化，硬件绑 0。	WARL	0
15:11	-	reserved	-	-
27:16	trtesrcid	TE 的 source id。如果 trteInhibitSrc 为 0，每条 message 都要携带 id 信息。	WARL	0
31:28	trtesrcbits	trtesrcid 在 m essage 中占用的 bi t 数。硬件绑定为 6	WARL	6

20.6.1.4 trTeInstFilters(0x00C)

表 20.10: trTeInstFilters 寄存器描述

bit	Field	Description	RW	Reset
15:0	trTeInstFilters	bit n 为 1 表示 filter n 使能，作用于指令 trace	WARL	0

20.6.1.5 trtedatacontrol(0x010)

表 20.11: trtedatacontrol 寄存器描述

bit	Field	Description	RW	Reset
0	trteDataImplemented	如果实现了 data trace，绑 1	R	1
1	trteDataTracing	当 trTeEnable 为 1 时，该字段可以通过工具写 1 或者 tr igger 触发置 1，该字段为 1 时，data t race 正常产生。该字段为 1 时，data trace 要受到 filter 的过滤	WARL	0
2	trteDataTrigEnable	trigger 控制 data trace on/off 的使能开关	WARL	0
3	trTeDataStallOrOver-flow	当产生 overflow msg 或者由于 data tr ace 导致核 stall 时硬件置 1。写 1 清 0	WC1	0
4	trteDataStallEna	为 1 时，如果 data trace 无法产生，stall 住核	WARL	0
5	trteDataDrop	从上一次读这个寄存器开始，是否出现过丢弃的 data trace。写 1 清 0；当 encoder 被 Reset 或者 trtee nable 从 0 变 1 时清 0	WC1	0

续下页

表 20.11 - 接上页

bit	Field	Description	RW	Reset
6	trteDataDropEna	为 1 时，允许丢弃 data trace 以保证指令 trace 不溢出。但是不能保证不会发生指令 tr ace 溢出。硬件绑 0	WARL	0
15:7	-	reserve	-	-
16	trteDataNoValue	为 1 时，data trace pack et 中不传递 data 值	WARL	0
17	trteDataNoAddr	为 1 时，data trace packet 中不传递 address 值	WARL	0
19:18	trTeDataAddrCompress	00：data trace pack et 中传递完整地址 01：用 XO R 模式压缩 address 10/11：不支持	WARL	01

20.6.1.6 trTeDataFilters(0x01C)

表 20.12: trTeDataFilters 寄存器描述

bit	Field	Description	RW	Reset
15:0	trTeDataFilters	bit n 为 1 表示 filter n 使能，作用于数据 trace	WARL	0

20.6.1.6.1 filter 控制寄存器

20.6.1.7 trtefilter*i*control(0x400+0x20*i),i = fliter_id

表 20.13: trtefilter*i*control 寄存器描述

bit	Field	Description	RW	Reset
0	trTeFilterEnable	filter 使能寄存器	WARL	0
1	trTeFilterMatchPrivilege	该字段为 1 时，当特权态与 trTeFilter-MatchChoicePrivilege 字段配置一致时允许生成 trace	WARL	0
2	trTeFilterMatchEcause	该字段为 1 时，当产生与 trTeFilterMatch-ChoiceEcause 字段一致的异常时开始生成 trace，当从异常处理程序退出时停止 trace		
3	trTeFilterMatchInterrupt	该字段为 1 时，当产生与 trTeFilterMatch-ValueInterrupt 字段一致的中断等级时开始生成 trace，当从异常处理程序退出时停止 trace		

续下页

表 20.13 - 接上页

bit	Field	Description	RW	Reset
4	trTeFilterMatchComp1	该字段为 1 时，被 trTeFilterComp1 选中的比较器为真时允许 trace		
7:5	trTeFilterComp1	定义所用的 comparator 单元		0
8	trTeFilterMatchComp2	该字段为 1 时，被 trTeFilterComp2 选中的比较器为真时允许 trace		
11:9	trTeFilterComp2	定义所用的 comparator 单元		
12	trTeFilterMatchComp3	该字段为 1 时，被 trTeFilterComp3 选中的比较器为真时允许 trace		
15:13	trTeFilterComp3	定义所用的 comparator 单元		
16	trTeFilterMatchImpdef			
23:17	-	Reserved	-	-
24	trTeFilterMatchDtype	该字段为 1 时，被指定的 dtype 才会被 trace		
25	trTeFilterMatchDsize	该字段为 1 时，被指定的 dsize 才会被 trace		
31:26	-	Reserved	-	-

20.6.1.8 trTeFilter*i*MatchInst(0x404+0x20*i),i = fliter_id

表 20.14: trTeFilter*i*MatchInst 寄存器描述

bit	Field	Description	RW	Reset
7:0	trTeFilterMatchChoicePrivilege	bitN 置 1 时，当 priv=N 时成功 match；trTeFilt erMatchPrivilege = 1 时生效	WARL	0
8	trTeFilterMatchValueInterrupt	0：匹配异常 1：匹配中断 trTeFilte rMatchInterrupt= 1 时生效	WARL	0
31:9	-	Reserved		

20.6.1.9 trTeFilter*i*MatchEcause(0x408+0x20*i),i = fliter_id

表 20.15: trTeFilter*i*MatchEcause 寄存器描述

bit	Field	Description	RW	Reset
31:0	trTeFilterMatchChoiceEcause	bitN 置 1 时，当 exception cause=N 时成功 match；trTeFi lterMatchEcause= 1 时生效	WARL	0

20.6.1.10 trTeFilter*i*MatchValueImpdef(0x410+0x20*i),i = fliter_id

表 20.16: trTeFilter*i*MatchValueImpdef 寄存器描述

bit	Field	Description	RW	Reset
31:0	trTeFilterMatchValueImpdef	(impdef & trTeFilterMatchMaskImpdef) == (trTeFilterMatchValueImpdef & trTeFilterMatchMaskImpdef) 时成功 match；trTeFilterMatchimpdef = 1 时生效	WARL	0

20.6.1.11 trTeFilter*i*MatchMaskImpdef(0x414+0x20*i),i = fliter_id

表 20.17: trTeFilter*i*MatchMaskImpdef 寄存器描述

bit	Field	Description	RW	Reset
31:0	trTeFilterMatchMaskImpdef	配合 trTeFilterMatchValueImpdef 使用	WARL	0

20.6.1.12 trTeFilter*i*MatchData(0x418+0x20*i),i = fliter_id

表 20.18: trTeFilter*i*MatchData 寄存器描述

bit	Field	Description	RW	Reset
15:0	trTeFilterMatchChoiceDtype	bitN 置 1 时，当 dtype=N 时成功 match；trTeFilterMatchDtype = 1 时生效（仅支持 dtype =0/1，因此只允许写入 bit0 和 bit1）	WARL	0
23:16	trTeFilterMatchChoiceDsize	b itN 置 1 时，当 dsiz e=N 时成功 match；trTeFilterMatchDsize = 1 时生效	WARL	0
31:24	-	Reserved		

20.6.1.12.1 fliter comparator csr

20.6.1.13 trTeComp*j*Control(0x600 + 0x20*j),j = comparator_id

表 20.19: trTeComp*j*Control 寄存器描述

bit	Field	Description	RW	Reset
1:0	trTeCompPInput	表示第一个 comparator 的比较内容 0：iaddr 1：context 2：tval 3：daddr	WARL	0
3:2	trTeCompSInput	表示第二个 comparator 的比较内容 0：iaddr 1：context 2：tval 3：daddr	WARL	0
6:4	trTeCompPFunction	第一个 comparator match 条件： 0：等于 trTeCompPMatch 1：不等于 trTeCompPMatch 2：小于 trTeCompPMatch 3：小于等于 trTeCompPMatch 4：大于 trTeCompPMatch 5：大于等于 trTeCompPMatch 6：永远匹配失败 7：永远匹配成功	WARL	0
7	-	-	-	-
10:8	trTeCompSFunction	第二个 comparator match 条件： 0：等于 trTeCompPMatch 1：不等于 trTeCompPMatch 2：小于 trTeCompPMatch 3：小于等于 trTeCompPMatch 4：大于 trTeCompPMatch 5：大于等于 trTeCompPMatch 6：永远匹配失败 7：永远匹配成功	WARL	0
11	-	-	-	-

续下页

表 20.19 - 接上页

bit	Field	Description	RW	Reset
13:12	trTeCompMatchMode	表示该比较器最终 match 成功的条件 0：第一个比较器 match 1：两个比较器 match 2：任意一个比较器 match 3：从第一个比较器 match 成功到第二个比较器 match 成功的整个过程中	WARL	0
14	trTeCompPNotify	为 1 生成触发第一个比较器 match 的地址 trace 包，trTeCompPInput=0 时该字段生效	WARL	0
15	trTeCompSNotify	为 1 生成触发第二个比较器 match 的地址 trace 包，trTeCompSInput=0 时该字段生效	WARL	0
31:16	-	Reserved	-	-

20.6.1.14 trTeComp*j*PMatchLow(0x610+0x20*j),j = comparator_id

表 20.20: trTeComp*j*PMatchLow 寄存器描述

bit	Field	Description	RW	Reset
31:0	trTeCompPMatchLow	第一个 comparator 的 match value，低 32bit	WARL	0

20.6.1.15 trTeComp*j*PMatchHigh(0x614+0x20*j),j = comparator_id

表 20.21: trTeComp*j*PMatchHigh 寄存器描述

bit	Field	Description	RW	Reset
31:0	trTeCompPMatchHigh	第一个 comparator 的 match value，高 32bit	WARL	0

20.6.1.16 trTeComp*j*SMatchLow(0x618+0x20*j),j = comparator_id

表 20.22: trTeComp*j*SMatchLow 寄存器描述

bit	Field	Description	RW	Reset
31:0	trTeCompPMatchLow	第二个 comparator 的 match value，低 32bit	WARL	0

20.6.1.17 trTeComp*j*SMatchHigh(0x61C+0x20*j),j = comparator_id

表 20.23: trTeCompjSMatchHigh 寄存器描述

bit	Field	Description	RW	Reset
31:0	trTeCompPMatchHigh	第二个 comparator 的 match value，高 32bit	WARL	0

20.6.1.17.1 timerstamp 控制寄存器

20.6.1.18 trtscontrol(0x040)

表 20.24: trtscontrol 寄存器描述

bit	field	Description	RW	reset
0	trtsActive	时间戳单元的主 reset/enable 开关	RW	0
1	trtsCount	internal 类型专用。为 1 时 c ounter 开始计数，为 0 时 counter 停止 tdt trace 系统中不实现 internal 类型 timestamp，绑 0。	WARL	0
2	trtsReset	inter nal 类型专用。写 1 复位时间戳计数器 tdt trace 系统中不实现 internal 类型 timestamp，绑 0。	W1	0
3	trTsRunInDebug	internal 类型专用。为 1 时，在调试模式下时间戳仍然工作；为 0 时调试模式下不工作 tdt trace 系统中不实现 internal 类型 timestamp，绑 0。	WARL	0
6:4	trtsType	时间戳单元的类型，如果仅实现一种就绑定值 0= 没有时间戳 1=external 2=internal 3=reserved 4=slave 5-7= 厂商定制类型 tdt trace 系统中实现 external 类型 timestamp，绑 1。	WARL	1
9:8	trTsPrescale	intern al 类型专用。分频器控制位，分频 2^(2*TsPrescale)。可以控制时间戳在 1、4、16、64 个 c ycle 内增加一次。 tdt trace 系统中不实现 internal 类型 timestamp，绑 0。	WARL	0
15	trTsEnable	为 1 时，trace message 中携带 timestamp 字段。 为 0 时，trace message 中不携带 timestamp 字段。	WARL	0

续下页

表 20.24 - 接上页

bit	field	Description	RW	reset
23:16		系统决定的字段，用于控制哪些消息类型包含时间戳。	WARL	0
29:24	trtsWidth	时间戳以 bit 计数的位宽数	R	32

20.6.1.19 trTsCounterLow(0x048)

表 20.25: trTsCounterLow 寄存器描述

bit	field	Description	RW	reset
31:0	trTsCounterLow	时间戳计数器的低 32bit	R	0

20.6.1.20 trTsCounterHigh(0x04c)

表 20.26: trTsCounterHigh 寄存器描述

bit	field	Description	RW	reset
31:0	trTsCounterHigh	时间戳计数器的高 32bit	R	0

20.6.1.20.1 external trigger 控制寄存器

20.6.1.21 trTeTrigExtInControl(0x050)

表 20.27: 寄存器描述

bit	field	Description	RW	reset
3:0	trTeTrigExtInAction0	当 external trigger input0 触发的时候，根据本字段选择的行为做响应，如果 input0 不存在，action 绑 0 0：没有 action 1：reserved 2：开始 trace 3：停止 trace 4：输出指令 trace sync 信息 5-15：reserved	WARL	0
31:4	trTeTrigExtInActionn	配置 1-7 号 external trigger input 的 action。没有 trig gerN 就没有对应的 actionn。编码方式与 input0 相同。	WARL	0

20.6.1.22 trTeTrigExtOutControl(0x054)

表 20.28: trTeTrigExtOutControl 寄存器描述

bit	field	Description	RW	reset
[3:0]	trTeTrigExtOutEvent0	配置什么事件会造成 external trigger output0 触发。如果 external trigger output0 不存在，这个字段绑 0 0：开始 trace 时触发 external trigger output0 1：停止 trace 时触发 external trigger output0 2：厂商定义 3：厂商定义	WARL	0
31:4	trTeTrigExtOutEventn	配置触发 1-7 号 external trigger output 的 event	WARL	0

20.6.1.22.1 trace 系统版本寄存器

20.6.1.23 TDT_TRID（0xe00）（扩展寄存器）

trace 版本信息寄存器 TDT_TRID 是 encoder 内寄存器，通过 dmi 寄存器，编址在 encoder 空间 0xe00。该寄存器结构符合 productID(Ver6.0).

表 20.29: TDT_TRID 寄存器描述

bit	field	Description	RW	reset
3:0	Version	ID Revision	R	0001
11:4	Reserved	-	R	0
17:12	Patch	产品补丁号，表示代码 bug 修复 Patch 号。对于研发版本，Patch 域可用于表示 bug 修复，也可用于功能增加记录；对于发布版本，Patch 域只用来表示 bug 修复。	R	0
23:18	Sub Revision	产品子版本号，表示代码分支与研发阶段功能升级。根据代码分支和研发阶段功能改动升级。当存在并行开发时，拉出新起项目与版本，子版本号升级，加 1 ，子版本被发布时，Patch 域清零。	R	0
27:24	Revision	产品主版本号，产品升级、编程模型改动时，此版本号加 1。	R	1
31:28	index0	每个 ID 寄存器都有 index 段，该段表明当前 ID 寄存器的编号。软件用户从 0 号 ID 寄存器开始读取，直至再次读到 0 号 ID 寄存器完成所有 ID 寄存器的读取。	R	0000

20.6.2 funnel 控制寄存器

20.6.2.1 trFunnelControl(0x000)

表 20.30: trFunnelControl 寄存器描述

bit	Field	Description	RW	Reset
0	trFunnelActive	Trace funnel 的总 enable 开关，写 0 复位 TF，可以使 TF power down 或者关闭 clk	RW	0
1	trFunnelEnable	为 1 时，从每个 encoder 过来的信息可以被接收、合并、发送到目标 sink。写 0 后停止接收上级传输的信息，待内部 trace 流传输完毕后，trFunnelEnable 读回值为 0	RW	0
3	trFunnelEmpty	当产生的 trace 都被发出去的话，读出值为 1。	R	1

20.6.2.2 trFunnelImpl(0x004)

表 20.31: trFunnelImpl 寄存器描述

bit	Field	Description	RW	Reset
3:0	trfunnelvermajor	major version 用于软件区分不兼容的 feature，当做出了不兼容以前版本的修改，需修改 major version 0：表示兼容以前的 spec 版本 1：表示兼容当前 spec 版本	R	1
7:4	trfunnelverminor	minor version，用于区分软件兼容的 feature，当做出了某些新扩展 feature，与以前版本兼容，需修改 minor version	R	0
11-8	trFunnelCompType	trace 组件类型，funnel 类型绑 0x8	R	0x8
23-12	-	reserved for standard	R	-
31-24	-	reserved for vendor	R	-

20.6.3 sram sink 控制寄存器

20.6.3.1 trramcontrol(0x000)

表 20.32: trramcontrol 寄存器描述

bit	Field	Description	RW	Reset
0	trRamActive	sram sink 的总 enable 开关，写 0 复位，可以使 sram sink power down 或者关闭 clk	RW	0
1	trRamEnable	为 1 时 sram sink 工作。写 0 后停止接收上级传输的信息，待内部 trace 流完全存入 sram 中后，trRamEnable 读回值为 0	RW	0
3	trRamEmpty	当 sram 所有数据都被读出时，读为 1	R	1
4	trRamMode	0 表示 sramsink 工作在 sram mode 1 表示 ramsink 工作在 sba mode 玄铁 trace 系统中 sram sink 和 ram sink 分开实现，sram sink 中本字段硬件绑 0	WARL	0
8	trRamStopOnWrap	写 1 时，当 sram 第一次满后，拉低 trramenable，停止将 trace 流存储进 sram 中，后续传输进 sram sink 的 trace message 被丢弃。	WARL	0

20.6.3.2　trRamImpl(0x004)

表 20.33: trRamImpl 寄存器描述

bit	Field	Description	RW	Reset
3:0	trramvermajor	major version，用于软件区分不兼容的 feature，当做出了不兼容以前版本的修改，需修改 major version 0：表示兼容以前的 spec 版本 1：表示兼容当前 spec 版本	R	1
7:4	trramverminor	minor version，用于区分软件兼容的 feature，当做出了某些新扩展 feature，与以前版本兼容，需修改 minor version	R	0
11-8	trRamCompType	trace 组件类型，sram 类型绑 0x9	R	0x9
12	trRamHasSRAM	本 ram sink 支持 sram 模式，硬件绑 1	R	1
13	trRamHaSMEM	本 ram sink 支持 sba 模式，硬件绑 0	R	0
23-14	-	reserved for standard	R	-
31-24	-	reserved for vendor	R	-

20.6.3.3　trRamStartLow(0x010)

表 20.34: trRamStartLow 寄存器描述

bit	Field	Description	RW	Reset
1:0	-	ba se 地址 4-byte 对齐，低两 bit 绑定为 0	R	0
31:2	trRamStartLow	trace sink circular buffer 的基字节地址。地址是 4-bytes 对齐的。 SRAM sink 中基地址 trRamStartLow 绑 0	WARL	0

20.6.3.4 trRamStartHigh(0x014)

表 20.35: trRamStartHigh 寄存器描述

bit	Field	Description	RW	Reset
31:0	trRamStartHigh	trace sink circular buffer 的基地址的高位。 SRAM sink 中基地址 trRamStartHigh 绑 0	WARL	0

20.6.3.5 trRamLimitLow(0x018)

表 20.36: trRamLimitLow 寄存器描述

bit	Field	Description	RW	Reset
1:0	-	base 地址 4-byte 对齐，低两 bit 绑定为 0	R	0
31:2	trRamLimitLow	trace sink circular buffer 的最高地址，根据不同配置硬件绑定。在一个 trace word 被写进 trRam-Limit 地址时，trRamWP 寄存器会被 Reset 到 tr-RamStart。	WARL	undef

20.6.3.6 trRamLimitHigh(0x01C)

表 20.37: trRamLimitHigh 寄存器描述

bit	Field	Description	RW	Reset
31:0	trRamLimitHigh	trace sink circular buffer 最高地址高位。sram 最大配置为 64k，地址位宽不到 32bit，此寄存器硬件绑 0	WARL	0

20.6.3.7 trRamWPLow(0x020)

表 20.38: trRamWPLow 寄存器描述

bit	Field	Description	RW	Reset
0	trRamWrap	当 trramwp 发生溢出回滚时会把这一 bit 置 1，等到 trRamWPLow 被写入的时候被置 0	WARL	0
1	-	base 地址 4-byte 对齐，bit1 绑定为 0	R	0
31:2	trRamWPLow	下一条 trace message 要写入 trace sink 的地址。当 trr amwp=trramlimit 时，若写入一笔 trace 数据，则 trra mwp 会被置位到 trramstart	WARL	undef

20.6.3.8 trRamWPHigh(0x024)

表 20.39: trRamWPHigh 寄存器描述

bit	Field	Description	RW	Reset
31:0	trRamWPHigh	写指针地址高位。 sram 最大配置为 64k，地址位宽不到 32bit，此寄存器硬件绑 0	WARL	0

20.6.3.9 trRamRPLow(0x028)

表 20.40: trRamRPLow 寄存器描述

bit	Field	Description	RW	Reset
31:2	trRamRPLow	trRamdata 寄存器对应的 circular buffer 中的地址，即读指针。访问一次 trRamdata ，trRamrp 自动加一。当 trRamRP = trramlimit 时，若读出一笔 trace 数据，则 trramrp 会被置位到 trramstart。sram sink 必须有这个寄存器的，其他类型的 sink 是可选的。	WARL	0

20.6.3.10 trRamRPHigh(0x02c)

表 20.41: trRamRPHigh 寄存器描述

bit	Field	Description	RW	Reset
31:0	trRamRPHigh	读指针地址高位 sram 最大配置为 64k，地址位宽不到 32bit，此寄存器硬件绑 0	WARL	0

20.6.3.11 trRamdata(0x040)

表 20.42: trRamdata 寄存器描述

bit	Field	Description	RW	Reset
31:0	trRamdata	sram sink 当前 trRamrp 指向位置的读 value。对于 sram sink，该寄存器是必须的。	R	0

20.6.4 system memory sink 控制寄存器

20.6.4.1 trramcontrol(0x000)

表 20.43: trramcontrol 寄存器描述

bit	Field	Description	RW	Reset
0	trRamActive	system memory sink 的总 enable 开关，写 0 复位，可以使 system memory sink power down 或者关闭 clk	RW	0
1	trRamEnable	为 1 时 system memory sink 工作。写 0 后停止接收上级传输的信息，待内部 trace 流传输完毕后，trRamEnable 读回值为 0	RW	0
3	trRamEmpty	当 system memory sink 中为空时，读为 1	R	1
4	trRamMode	0 表示 sramsink 工作在 sram mode 1 表示 ramsink 工作在 sba mode 玄铁 trace 系统中 sram sink 和 ram sink 分开实现，sba sink 中本字段硬件绑 1	WARL	1
8	trRamStopOnWrap	写 1 时，当 sram 第一次满后，拉低 trramenable，停止将 trace 流存储进 sram 中，后续传输进 sram sink 的 trace message 被丢弃。	WARL	0

20.6.4.2 trRamImpl(0x004)

表 20.44: trRamImpl 寄存器描述

bit	Field	Description	RW	Reset
3:0	trramvermajor	major version，用于软件区分不兼容的 feature，当做出了不兼容以前版本的修改，需修改 major version 0：表示兼容以前的 spec 版本 1：表示兼容当前 spec 版本	R	1

续下页

表 20.44 - 接上页

bit	Field	Description	RW	Reset
7:4	trramverminor	minor version，用于区分软件兼容的 feature，当做出了某些新扩展 feature ，与以前版本兼容，需修改 minor version	R	0
11-8	trRamCompType	trace 组件类型，sram 类型绑 0x9	R	0x9
12	trRamHasSRAM	本 ram sink 支持 sram 模式，硬件绑 0	R	0
13	trRamHaSMEM	本 ram sink 支持 sba 模式，硬件绑 1	R	1
23-14	-	reserved for standard	R	-
31-24	-	reserved for vendor	R	-

20.6.4.3 trRamStartLow(0x010)

表 20.45: trRamStartLow 寄存器描述

bit	Field	Description	RW	Reset
1:0	-	base 地址 4-byte 对齐，低两 bit 绑定为 0	R	0
31:2	trRamStartLow	trace sink circular buffer 的基字节地址。地址是 4-bytes 对齐的。	WARL	0

20.6.4.4 trRamStartHigh(0x014)

表 20.46: trRamStartHigh 寄存器描述

bit	Field	Description	RW	Reset
31:0	trRamStartHigh	trace sink circular buffer 的高位地址。	WARL	0

20.6.4.5 trRamLimitLow(0x018)

表 20.47: trRamLimitLow 寄存器描述

bit	Field	Description	RW	Reset
1:0	-	base 地址 4-byte 对齐，低两 bit 绑定为 0	R	0
31:2	trRamLimit	trace sink circular buffer 的最高地址，根据不同配置硬件绑定。在一个 trace word 被写进 trRamLimit 地址时，trRamWP 寄存器会被 Reset 到 trRamStart。	WARL	undef

20.6.4.6 trRamLimitHigh(0x01C)

表 20.48: trRamLimitHigh 寄存器描述

bit	Field	Description	RW	Reset
31:0	trRamLimitHigh	trace sink circular buffer 最高地址高位。	WARL	0

20.6.4.7 trRamWPLow(0x020)

表 20.49: trRamWPLow 寄存器描述

bit	Field	Description	RW	Reset
0	trRamWrap	当 trramwp 发生溢出回滚时会把这一 bit 置 1，等到只有在 trramwp 被写入的时候被置 0	WARL	0
1	-	base 地址 4-byte 对齐，bit1 绑定为 0	R	0
31:2	trRamWPLow	下一条 trace message 要写入 trace sink 的地址。当写一个 trace word 时，如果 trramwp = trram-limit，那 trramwp 会被置位到 trrambase	WARL	undef

20.6.4.8 trRamWPHigh(0x024)

表 20.50: trRamWPHigh 寄存器描述

bit	Field	Description	RW	Reset
31:0	trRamWPHigh	写指针地址高位。	WARL	0

20.6.4.9 trRamRPLow(0x028)

表 20.51: trRamRPLow 寄存器描述

bit	Field	Description	RW	Reset
N:2	trRamRPLow	硬件绑 0	R	0

20.6.4.10 trRamRPHigh(0x02c)

表 20.52: trRamRPHigh 寄存器描述

bit	Field	Description	RW	Reset
31:0	trRamRPHigh	读指针地址高位 硬件绑 0	WARL	0

20.6.4.11　trramdata(0x040)

表 20.53: trramdata 寄存器描述

bit	Field	Description	RW	Reset
31:0	trRamdata	硬件绑 0	R	0

20.6.5　ATB bridge 控制寄存器

20.6.5.1　trAtbBridgeControl(0x000)

表 20.54: trAtbBridgeControl 寄存器描述

bit	Field	Description	RW	Reset
0	trAtbBridgeActive	atb sink 的使能位，写 0 将复位 atb sink	RW	0
1	trAtbBridgeEnable	使能 trace 信息从 encoder /funnel 中传到 SOC atb 总线上。写 0 后停止接收上级传输的信息，待内部 trace 流传输完毕后，tratbbridgeEnable 读回值为 0	RW	0
2	Reserved	-	-	0
3	trAtbBridgeEmpty	当 ATB 内部 buffer 为空的时候，读为 1	R	1
14:8	trAtbBridgeID	从 atb sink 输出的 atbid，00 与 70—7F 是 reserved 的。	RW	0x01

20.6.5.2　trAtbBridgeImpl(0x004)

表 20.55: trAtbBridgeImpl 寄存器描述

bit	Field	Description	RW	Reset
3:0	tratbvermajor	major version，用于软件区分不兼容的 feature，当做出了不兼容以前版本的修改，需修改 major version 0：表示兼容以前的 spec 版本 1：表示兼容当前 spec 版本	R	1
7:4	tratbverminor	minor version，用于区分软件兼容的 feature，当做出了某些新扩展 feature，与以前版本兼容，需修改 minor version	R	0
11-8	trAtbBridgeCompType	trace 组件类型，ATB Bridge 类型绑 0xE	R	0xE
23-12	-	reserved for standard	R	-
31-24	-	reserved for vendor	R	-

20.6.6 PIB sink 控制寄存器

20.6.6.1 trpibcontrol(0x000)

表 20.56: trpibcontrol 寄存器描述

bit	Field	Description	RW	Reset
0	trpibActive	PIB Sink 的主 enable/Reset 开关，写 0 将复位 pib sink	RW	0
1	trpibEnable	0：不接受输入，根据 pibMode 将输出置为 idle 1：使能 PIB 产生输出 写 0 后停止接收上级传输的信息，待内部 trace 流传输完毕后，trpibEnable 读回值为 0	RW	0
2	reserved	-	-	0
3	trpibempty	当 pib 内部没有 trace data 的时候，pibempty 读为 1	R	1
7:4	trpibMode	选择输出模式 0：off 4：SWT Manchester 5：UART 8：tref+1 data 9：tref+2 data 10：tref+4 data 11：tref+8 data 12：tref+16 data	WARL	0

续下页

表 20.56 - 接上页

bit	Field	Description	RW	Reset
8	trpibclkCenter	当 pibm ode 选择了 parallel 模式，该 bit 才可以置 1，否则清 0。 置 1 时，tref 上升沿出现在 tdata 的中间（见后续时序图）	WARL	SD 0
9	trpibCalibrate	设置这个 bit 为 1 时，PIB 产生循环的校准模板，用于使外部探针维持同步，校准模式在下面给出	WARL	0
31:16	trpibDivider	PIB module 的时钟选择。将 input clock 做 pibDivider +1 分频。根据 pibMode 的选择，PIB data 根据这个分频时钟或者这个分频时钟的二分之一输出。pibDivider 的位宽是实现定义的	WARL	SD 0

20.6.6.2 trpibImpl(0x004)

表 20.57: trpibImpl 寄存器描述

bit	Field	Description	RW	Reset
3:0	trpibvermajor	major v resion，用于软件区分不兼容的 fea ture，当做出了不兼容以前版本的修改，需修改 major version 0：表示兼容以前的 spec 版本 1：表示兼容当前 spec 版本	R	1
7:4	trpibverminor	minor version，用于区分软件兼容的 f eature，当做出了某些新扩展 featu re，与以前版本兼容，需修改 minor version	R	0
11-8	trpibCompType	trace 组件类型，PIB sink 类型绑 0xa	R	0xA
23-12	-	reserved for standard	R	-
31-24	-	reserved for vendor	R	-

20.7 附录 C-7 多核与 L2 控制寄存器

20.7.1 基础功能寄存器简介

基础功能寄存器为公有寄存器，允许每个核通过地址发起寄存器访问，若该寄存器存在寄存器编号，则允许通过 CSR 指令发起访问。

20.7.1.1 L2CR 寄存器

功能描述：

L2CR 寄存器用来控制 L2 Cache 特定功能的开关。

寄存器位宽：

64

寄存器编号：

0x7C3 (同 MCCR2)

访问类型：

视寄存器内容描述决定

寄存器内容描述：

表 20.58: L2CR 寄存器

位	名称	读写属性	功能描述	复位值
32	PARTITION_EN	RW	细粒度回填使能位 1：允许细粒度回填 0：禁止细粒度回填	0
31	TPRF_EN	RW	TLB 预取使能位 1：允许 TLB 预取 0：禁止 TLB 预取	0
30:28	IPRF_EN	RW	指令预取使能位 3'b000：禁止指令预取 3'b001：允许指令预取，预取距离为 1 3'b010：允许指令预取，预取距离为 2 3'b011：允许指令预取，预取距离为 4 3'b100：允许指令预取，预取距离为 8 3'b101：允许指令预取，预取距离为 16 3'b110：允许指令预取，预取距离为 32	0
25	TESETUP_EN	RO	是否开启 Tag Setup 流水级 1：开启 0：关闭	由宏配置决定
24:22	TLATAENCY	RW	Tag 访问延时周期数： 3'b001：Tag 访问延时周期为 1 3'b010：Tag 访问延时周期为 2 3'b011：Tag 访问延时周期为 3 3'b100：Tag 访问延时周期为 4 3'b101：Tag 访问延时周期为 5	由宏配置决定

续下页

表 20.58 - 接上页

位	名称	读写属性	功能描述	复位值
19	DESETUP_EN	RO	是否开启 Data Setup 流水级 1：开启 0：关闭	由宏配置决定
18:16	DLATENCY	RW	Data 访问延时周期数： 3'b001：Data 访问延时周期为 1 3'b010：Data 访问延时周期为 2 3'b011：Data 访问延时周期为 3 3'b100：Data 访问延时周期为 4 3'b101：Data 访问延时周期为 5 3'b110：Data 访问延时周期为 6 3'b111：Data 访问延时周期为 7	由宏配置决定
3	L2_EN	RW	是否开启 L2 Cache	0
1	ECC_EN	RW	是否开启 ECC 校验 1：开启 0：关闭	0

20.7.1.2 L2SR 寄存器

功能描述：

L2SR 寄存器用来指示 L2 Cache 当前是否处于 Busy（未执行完成）状态，修改 L2 Cache 电源状态会使 L2 Cache 处于 Busy 状态。

寄存器位宽：

64

寄存器编号：

无

访问类型：

视寄存器内容描述决定

寄存器内容描述：

表 20.59: L2SR 寄存器

位	名称	读写属性	功能描述	复位值
0	BUSY	RO	L2 Cache 是否处于 BUSY 状态 1：处于 BUSY 状态 0：未处于 BUSY 状态	0

20.7.1.3　L2MCR 寄存器

功能描述：

L2MCR 寄存器用来控制 L2 Cache 的部分模式设定，包括设置收到外部 Snoop 时是否需要返回 Clean 的数据、L2 Cache 进行替换时是否需要向外部进行广播、来自核心或 DCP 的 AWLEN 不为 3 的写请求是否需要进行数据 Merge 后写出。

寄存器位宽：

64

寄存器编号：

无

访问类型：

视寄存器内容描述决定

寄存器内容描述：

表 20.60: L2MCR 寄存器

位	名称	读写属性	功能描述	复位值
28:24	NCWN	RW	Non-Cacheable 写请求 Outstanding 数量控制位	0
20:16	NCRN	RW	Non-Cacheable 读请求 Outstanding 数量控制位	0
7:6	SDCC	RW	收到外部 Snoop 时处于 Clean 状态的 Cacheline 是否需要传输数据 00：所有 Clean 的数据都会传输 01：仅 Unique Clean 的数据会传输 10：保留 11 ： 所有 Clean 的数据都不会传输	01
5:4	RDCC	RW	进行 L2 Cache 替换时 Clean 态的被替换数据是否需要广播到外部 00：Clean 态的数据不广播 01：Clean 态的数据均通过 Evict 广播 10：Share 态的数据通过 Evict 广播，Unique 态的数据通过 WriteEvict 广播 11：保留	10
0	WPE	RW	来自核心或 DCP 的 AWLEN 不为 3 的写请求是否需要与总线数据进行 Merge 1：不需要进行 Merge，直接写到总线 0：该类写请求会 allocate 到 L2 Cache，不会直接写到总线	0

20.7.1.4　L2OPCR 寄存器

功能描述：

L2OPCR 寄存器用来对 L2 Cache 和 SnoopFilter 执行 Flush 类请求，可执行的 Flush 请求类型包括：

- BLOCK L2 CACHE：阻塞 L2 Cache 访问
- INVALID L2 CACHE：将 L2 Cache 内容全部无效
- CLEAN L2 CACHE：将 L2 CacheDirty 内容全部清到 Memory
- CLEAN&INVALID L2 CACHE：将 L2 CacheDirty 内容全部清到 Memory 并将所有 Cacheline 无效
- INVALID SNOOPFILTER：将 SnoopFilter 内容全部无效
- INVALID L2 CACHE&SNOOPFILTER：将 L2 Cache 和 SnoopFilter 内容全部无效

寄存器位宽：

64

访问类型：

视寄存器内容描述决定

寄存器内容描述：

表 20.61: L2OPCR 寄存器

位	名称	读写属性	功能描述	复位值
6:4	OP	RW	操作类型 000：仅 block 001：仅 invalid l2 010：仅 clear 011：clear&invalid 100：invalid snoopfilter 101：invalid l2&snoopfilter	0
0	OP_EN	RW	使能开关，写 1 开始执行 OP 对应操作 当操作开始执行后该位会自动置 0 1：执行操作	0

20.7.1.5　L2ER 寄存器

功能描述：

L2ER 寄存器用来指示系统是否出现 ECC ERROR 或总线 ERROR 并指示发生错误的组件与错误属性

寄存器位宽：

64

寄存器编号：

0x7C4

访问类型：

视寄存器内容描述决定

寄存器内容描述：

表 20.62: L2ER 寄存器

位	名称	读写属性	功能描述	复位值
63	ERR_VLD	R/W	指示是否出现 ECC 错误或总线错误	0
62	ECC_FATAL	RO	指示本次错误是否是 ECC Fatal 错误	0
60:56	FIX_ERR_CNT	RO	表示目前已经修正的 ECC 错误数量	0
55:53	RAMID	RO	指示出现 ECC 错误的组件 00：Tag 01：DATA 11：SnoopFilter	0
52	ECC_ERR	RO	指示出现 ECC 错误	0
51	STORE_ERR	RO	指示出现 Store 错误	0
37:34	STORE_ERR_CLSTID	RO	指示出现 Store 错误的核心 ID 0000：核心 0 0001：核心 1 0010：核心 2 0011：核心 3 1111：L2 Eviction	0
33:32	STORE_ERR_INFO	RO	指示出现 Store 错误的访问属性 00：Weak-Order Cacheable 01：Weak-Order Non-Cacheable 10：Strong-Order Non-Cacheable 11：APB Strong-Order Non-Cacheable	0
31:25	ECC_WAY	RO	指示出现 ECC 错误的 Way 信息	0
20:0	ECC_SET	RO	指示出现 ECC 错误的 Set 信息	0

20.7.1.6　L2WP 寄存器

功能描述：

L2WP 寄存器用来控制进程细粒度回填 L2 Cache，可以为不同进程分配不同的可回填 Cache Way，共 8 个 Group，每个 Group 表示 Cache 中 2 个 way，包含 8 bit 信息，每个 bit 分别表示进程 ID 0~8 是否允许回填该 Group 对应的 Cache way，为 1 表示允许回填，为 0 表示禁止回填。

寄存器位宽：

64

寄存器编号：

0x7F7

访问类型：

视寄存器内容描述决定

寄存器内容描述：

表 20.63: L2WP 寄存器

位	名称	读写属性	功能描述	复位值
63:56	GROUP7_PARTITION	RW	L2 Cache Way 14~15 是否允许对应 ID 的进程回填	FF
55:48	GROUP6_PARTITION	RW	L2 Cache Way 12~13 是否允许对应 ID 的进程回填	FF
47:40	GROUP5_PARTITION	RW	L2 Cache Way 10~11 是否允许对应 ID 的进程回填	FF
39:32	GROUP4_PARTITION	RW	L2 Cache Way 8~9 是否允许对应 ID 的进程回填	FF
31:24	GROUP3_PARTITION	RW	L2 Cache Way 6~7 是否允许对应 ID 的进程回填	FF
23:16	GROUP2_PARTITION	RW	L2 Cache Way 4~5 是否允许对应 ID 的进程回填	FF
15:8	GROUP1_PARTITION	RW	L2 Cache Way 2~3 是否允许对应 ID 的进程回填	FF
7:0	GROUP0_PARTITION	RW	L2 Cache Way 0~1 是否允许对应 ID 的进程回填	FF

20.7.1.7　L2EIR 寄存器

功能描述：

L2EIR 寄存器用于进行 ECC 注错，可通过 RAMID 来选择要进行 ECC 注错的 RAM ID，可以通过设置 CNTDN 来设置定时注错，会在倒计时结束后向 RAM 进行 ECC 注入。若需要重复注入，可通过开启 ECC_REPEAT 来进行，ERR_TYPE 控制注入 1 bit 或 2 bit 错误，进行注错时需要手动将 INJECT_EN 置 1，开启注错使能；若未设置倒计时，则会立即注错。

寄存器位宽：

64

寄存器编号：

0x7D7

访问类型：

视寄存器内容描述决定

寄存器内容描述：

表 20.64: L2EIR 寄存器

位	名称	读写属性	功能描述	复位值
31:29	RAMID	RW	指示注入 ECC 错误的 RAM 000:TAG 001:DATA 011:SnoopFilter	0
7:3	CNTDN	RW	指示下次注入时机为 2^{CNTDN} 周期后	0
2	ECC_REPEAT	RW	是否重复注入 1：重复注入 0：单次注入	0
1	ERR_TYPE	RW	错误信号 0：1 bit 错误 1：2 bit 错误	0
0	INJECT_EN	RW	注入使能信号，手动置 1 非重复注入时，会在注入成功后清 0； 重复注入一直为 1	0

20.7.1.8 L2DECR 寄存器

功能描述：

L2DECR 用于进行 L2 Cache 电源状态转换，通过向 EXP_L2EN 位写入电源状态译码来改变 L2 Cache 电源状态。

寄存器位宽：

64

寄存器编号：

无

访问类型：

视寄存器内容描述决定

寄存器内容描述：

表 20.65: L2DECR 寄存器

位	名称	读写属性	功能描述	复位值
1:0	EXP_L2EN	RW	期望的 L2 使能情况： 11：打开全部 10：非法值，会强制修改为 11 01：Reserved 00：全部关闭	2'b11

20.7.1.9　L2DESR 寄存器

功能描述：

L2DESR 用于指示当前 L2 Cache 电源状态。通过读取 CUR_L2EN 位来获取当前 L2 Cache 电源状态，通过读取 SF_OCCUPIED 获取当前 SnoopFilter 是否可以下电。

寄存器位宽：

64

寄存器编号：

无

访问类型：

视寄存器内容描述决定

寄存器内容描述：

表 20.66: L2DESR 寄存器

位	名称	读写属性	功能描述	复位值
1:0	CUR_L2EN	RO	当前 L2 Cache 使能情况： 11：打开全部 10：非法值，会强制修改为 11 01：Reserved 00：全部关闭	0
2	SF_OCCUPIED	RO	当前 SnoopFilter 是否可以下电: 0: 允许下电 1: 禁止下电	0

20.7.2　核心私有寄存器简介

每个核心均配置有一组私有寄存器，当核心发起访问时仅会读到属于本核的寄存器，无法读取或修改其他核心的私有寄存器。

20.7.2.1　SMPR 寄存器

功能描述：

SMPR 寄存器用来控制核心是否可以接受 Snoop 请求（包括 DVM 请求），该寄存器与 MSMPR 为同一物理寄存器。

寄存器位宽：

64

寄存器编号：

0x7F3

访问类型：

视寄存器内容描述决定

寄存器内容描述：

表 20.67: SMPR 寄存器

位	名称	读写属性	功能描述	复位值
0	SMPEN	RW	核心 Snoop 传输使能位： 0：禁止向核心发起 Snoop 1：允许向核心发起 Snoop	0

20.7.2.2　HPCPL2DA 寄存器

功能描述：

HPCPL2DA 寄存器用来记录本核 L2 数据访问次数

寄存器位宽：

64

寄存器编号：

无

访问类型：

视寄存器内容描述决定

寄存器内容描述：

表 20.68: HPCPL2DA 寄存器

位	名称	读写属性	功能描述	复位值
63:0	CNT	RW	L2 数据访问计数值	0

20.7.2.3　HPCPL2DM 寄存器

功能描述：

HPCPL2DM 寄存器用来记录本核 L2 数据访问 MISS 次数

寄存器位宽：

64

寄存器编号：

无

访问类型：

视寄存器内容描述决定

寄存器内容描述：

表 20.69: HPCPL2DM 寄存器

位	名称	读写属性	功能描述	复位值
63:0	CNT	RW	L2 数据访问 MISS 计数值	0

20.7.2.4　HPCPL2IA 寄存器

功能描述：

HPCPL2IA 寄存器用来记录本核 L2 指令访问次数

寄存器位宽：

64

寄存器编号：

无

访问类型：

视寄存器内容描述决定

寄存器内容描述：

表 20.70: HPCPL2IA 寄存器

位	名称	读写属性	功能描述	复位值
63:0	CNT	RW	L2 指令访问计数值	0

20.7.2.5　HPCPL2IM 寄存器

功能描述：

HPCPL2IM 寄存器用来记录本核 L2 指令访问 MISS 次数

寄存器位宽：

64

寄存器编号：

无

访问类型：

视寄存器内容描述决定

寄存器内容描述：

表 20.71: HPCPL2IM 寄存器

位	名称	读写属性	功能描述	复位值
63:0	CNT	RW	L2 指令访问 MISS 计数值	0

20.7.2.6　HPCPL2RVLD 寄存器

功能描述：

HPCPL2RVLD 寄存器用来记录本核 L2 所有类型读请求数量

寄存器位宽：

64

寄存器编号：

无

访问类型：

视寄存器内容描述决定

寄存器内容描述：

表 20.72: HPCPL2RVLD 寄存器

位	名称	读写属性	功能描述	复位值
63:0	CNT	RW	本核所有类型读请求计数值	0

20.7.2.7　HPCPL2RSTALL 寄存器

功能描述：

HPCPL2RSTALL 寄存器用来记录本核 L2 所有读请求阻塞周期数

寄存器位宽：

64

寄存器编号：

无

访问类型：

视寄存器内容描述决定

寄存器内容描述：

表 20.73: HPCPL2RSTAL 寄存器

位	名称	读写属性	功能描述	复位值
63:0	CNT	RW	本核所有读请求访问阻塞周期计数值	0

21 附录 D 玄铁 C900 多核同步相关指令和程序实现

21.1 概述

玄铁 C900 处理器的多核同步基于 RISC-V 架构，遵循 RISC-V 特权规范中关于指令区同步 (fence.i) 、TLB 维护 (sfence.vma) 和原子指令集扩展的定义。

同时，针对多核，非一致性总线情景下的维护效率，在规范之外，玄铁分别对指令区同步，TLB 维护，和 DMA 同步，进行了独有的增强，以满足不同的市场需求。

21.2 RISC-V 规范指令

21.2.1 fence 指令

基础的 RISC-V 指令集包含了 fence 指令，它用来显式地保证程序指令的顺序:

```
FENCE IORW, IORW
```

fence 指令区分 IO 地址空间和内存地址空间，所以 IO 就是 Input Output，RW 就是 Read Write。

FENCE RW 保证前序的读写指令，不能落后于本 FENCE 指令发生。FENCE ,RW 保证后续的读写指令，不能超前于本 FENCE 指令发生。

同理可以单独组合出：FENCE R, RW / FENCE R, R / FENCE R / FENCE RW / FENCE RW, W ... IO 就相当于 RW，所以可以组合出 FENCE I, IO / FENCE I, I / FENCE I / FENCE IO / FENCE IO, O ... 甚至混合使用，FENCE RI, IORW / FENCE IORW, IORW ...

总之，FENCE 通过定义前后序的 R, W, I, O 这 8 个 bit，让程序员能清晰明确 load/store 对内存或 IO 操作的顺序要求。

21.2.2 fence.i 指令

清空 I-Cache, 保证该指令前序所有数据访存结果能够被指令后的取指操作访问到。

21.2.3 sfence.vma 指令

sfence.vma rs1,rs2 用于虚拟内存的无效和同步操作，rs1：虚拟地址，rs2：asid

- rs1=x0，rs2=x0 时，无效 TLB 中所有的表项

- rs1!=x0，rs2=x0 时，无效 TLB 中所有命中 rs1 虚拟地址的表项
- rs1=x0，rs2!=x0 时，无效 TLB 中所有命中 rs2 进程号的表项
- rs1!=x0，rs2!=x0 时，无效 TLB 中所有命中 rs1 虚拟地址和 rs2 进程号的表项

21.2.4　AMO 指令

原子操作，简单讲就是多个线程对一个共享内存地址的独占“读 - 修改 - 写回”的连续操作。

在单核系统中，只要保证不被中断/异常打断，那么独占就成立。并且在单核系统中，内存模型相对简单，符合编程者的直觉，即：一个读操作返回值总是来自上一次同地址的写入操作。无论单核 CPU 的 LSU 如何设计，总能保证编程者对内存访问的预期。

在多核系统中，内存模型变得异常复杂，情况可能不再直观。“什么是最后一次写入值？这个读序列的结果是否保证顺序？这是下一个写操作？”这些在单核中不需要担心的场景，在多核环境下可能变得错综复杂。

目前，不同的硬件实现定义了多种内存一致性模型：

- Sequential consistency
- Processor consistency
- Weak consistency
- Release consistency （RISC-V）

这也导致了每种架构对原子操作的定义也有所不同。

在 RISC-V 架构中，AMO 指令涵盖了丰富的 ALU 操作（加法、与/或/异或、MIN/MAX 等），基本满足了 Linux 对原子操作原语的需求。但是也存在一个问题，就是支持的数据类型比较有限。RV64 支持 word/double word，缺乏对 half word 的支持。而恰恰 qspinlock 对 half word 的 xchg 操作有强需求，导致 RISC-V 目前无法有效支持 qspinlock。

21.2.5　Load-Reserved/Store-Conditional 指令

LR/SC 指令在 ARM 架构中被广泛应用，与之功能相当的是 x86 的 compare-and-swap (CAS) 指令。

RISC-V 的 LR/SC 定义如下：

LR 与 load 类似，获取指定内存的数据，并对该地址后序的写操作进行监视。CPU 对获取的数据进行 ALU 计算后，通过 SC 指令，把新的值写入之前 LR 操作的内存地址。如果该内存地址未发生任何 CPU 的写入操作，那么 SC 会像普通 store 那样把新值写入内存，并设置 rd 为 0 值，表示成功。否则不会写入，并设置 rd 为非 0 值，表示失败。

之所以选择 LR/SC，RISC-V 给出了如下解释：

1. CAS 有 ABA 问题，即只关心结果，不关心过程，只要上一次 load 的值和 cas 获取的值一致，那么新值就成功写入。即使有人写过该地址，或者写过两次，第二次又改回来。这有时候不符合程序员的预期，破坏了原子性。而 LR/SC 会监视任何写入操作，即使写入相同值也会破坏 SC 指令。
2. CAS 硬件实现过于复杂，它需要三个源寄存器，和一个目的寄存器 (结果)。

3. 为了解决 ABA 问题，有些系统提供了 DW-CAS 指令，这个指令的实现更加复杂，需要 5 个源寄存器和 2 个目的寄存器。
4. LR/SC 的效率优于 CAS，因为 CAS 总是要多一条 load 指令，即 load + CAS 指令，而 LR + SC 只有一条 load 指令。

以上就是 RISC-V 给出的选择 LR/SC 而放弃 CAS 指令的原因。然而实际情况是，软件 API 并不能很好地支持 LR/SC，譬如 Linux 只用 cmpxchg 原语直接对应 CAS 指令，而没有 load_reserved/store_conditional 原语。

这导致实际使用是用 LR/SC 去实现 cmpxchg：

```
# a0 holds address of memory location
# a1 holds expected value
# a2 holds desired value
# a0 holds return value, 0 if successful, !0 otherwise
cas:
lr.w t0, (a0) # Load original value.
bne t0, a1, fail # Doesn't match, so fail.
sc.w t0, a2, (a0) # Try to update.
bnez t0, cas # Retry if store-conditional failed.
li a0, 0 # Set return to success.
jr ra # Return.
fail:
li a0, 1 # Set return to failure.
jr ra # Return.
```

加上 cmpxchg 使用本身的循环结构，实际上构成了双循环实现：

```
c = v->counter;
while ((old = cmpxchg(&v->counter, c, c c_op i)) != c)
        c = old;
```

如果这样使用，那么 RISC-V 的 4 条理由中，1) 3) 4) 的好处都不存在了，所以放弃 CAS 对软件兼容性产生负面影响。arm64 目前支持了 CAS 指令，就是很好的例证。

LR/SC 的活锁问题就更复杂了，对于 > 128 harts 的 NUMA 系统，会遇到更多问题。(本文不展开)

相比 arm64，RISC-V LR/SC 还缺少 LR/wfe 的配对使用方式，无法实现 load_cond 原语（该指令在单 core 含有多 threads 时是必备的，以让出流水线）。

在 non-cacheable 地址区间使用 lr/sc 指令时，会置位 AXI 总线的 axlock 位。若需在 non-cacheable 地址区间使用 lr/sc 指令，需要 slave 端拥有大于等于核心数量的 monitor。

21.3　XuanTie 增强指令

21.3.1　sync.is

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。该指令退休时清空流水线，并将该请求广播给其他核，也可以 sync.s (仅 flush)。

21.3.2　dcache.cipa rs1

将 rs1 中物理地址所属的 dcache/L2cache 表项写回下级存储并无效该表项，也可以 dcache.cpa (仅 flush), dcache.ipa(仅 invalid) 分别使用。

21.3.3　icache.iva rs1

将 rs1 中虚拟地址所对应的 icache 表项无效。

21.4　软件示例

以 Linux RISC-V 架构对 MMU 和 CACHE 维护实现举例，给出相关操作在 OS 中的软件示例。

21.4.1　TLB 维护

21.4.1.1　全刷 TLB

```
static inline void local_flush_tlb(unsigned long asid)
{
       __asm__ __volatile__ ("sfence.vma" : : : "memory");
}
```

21.4.1.2　根据 ASID 刷进程相关 TLB

```
static inline void local_flush_tlb(unsigned long asid)
{
       __asm__ __volatile__ ("sfence.vma , %0" : : "r" (asid) : "memory");
}
```

21.4.1.3　根据 VA 刷 TLB 表项

```
static inline void local_flush_tlb_range(unsigned long start, unsigned long size)
{
        unsigned long page_add = PAGE_DOWN(start);
        unsigned long page_end = PAGE_UP(start + size);

        while(page_add < page_end) {
                __asm__ __volatile__ ("sfence.vma %0, zero"
                                      :
                                      : "r" (page_add), "r" (asid)
                                      : "memory");
                page_add += PAGE_SIZE;
        }
}
```

21.4.1.4　根据 VA 和 ASID 刷 TLB 表项

```
static inline void local_flush_tlb_range_asid(unsigned long start, unsigned long␣
↪size, unsigned long asid)
{
        unsigned long page_add = PAGE_DOWN(start);
        unsigned long page_end = PAGE_UP(start + size);

        while(page_add < page_end) {
                __asm__ __volatile__ ("sfence.vma %0, %1"
                                      :
                                      : "r" (page_add), "r" (asid)
                                      : "memory");
                page_add += PAGE_SIZE;
        }
}
```

21.4.2　指令区同步

21.4.2.1　本核全局指令区同步

```
static inline void local_flush_icache_all(void)
{
         asm volatile ("fence.i" ::: "memory");
}
```

21.4.2.2 多核全局指令区同步

```
static void ipi_remote_fence_i(void *info)
{
        asm volatile ("fence.i" ::: "memory");
}

void flush_icache_all(void)
{
        on_each_cpu(ipi_remote_fence_i, NULL, 1);
}
```

21.4.2.3 XuanTie 多核精确指令区同步

```
static inline void flush_icache_range(unsigned long va_start, unsigned long size)
{
       register unsigned long i asm("a0") = va_start & ~(L1_CACHE_BYTES - 1);

       for (; i < (start + size); i += L1_CACHE_BYTES)
               __asm__ __volatile__ ("icache.iva" : : "r" (asid) : "memory");

       __asm__ __volatile__("sync.is");
}
```

21.4.3 DMA 同步

21.4.3.1 XuanTie 多核精确 DMA 同步，含 3 个方向

```
void dma_sync_from_cpu_to_dev(unsigned long pa_start, unsigned long size)
{
       register unsigned long i asm("a0") = pa_start & ~(L1_CACHE_BYTES - 1);

       for (; i < (start + size); i += L1_CACHE_BYTES)
               __asm__ __volatile__ ("dcache.cpa" : : "r" (asid) : "memory");

       __asm__ __volatile__("sync.s");
}

void dma_sync_from_dev_to_cpu(unsigned long pa_start, unsigned long size)
{
```

(续下页)

(接上页)

```
        register unsigned long i asm("a0") = pa_start & ~(L1_CACHE_BYTES - 1);

        for (; i < (start + size); i += L1_CACHE_BYTES)
                __asm__ __volatile__ ("dcache.ipa" : : "r" (asid) : "memory");

        __asm__ __volatile__("sync.s");
}

void dma_sync_all(unsigned long pa_start, unsigned long size)
{
        register unsigned long i asm("a0") = pa_start & ~(L1_CACHE_BYTES - 1);

        for (; i < (start + size); i += L1_CACHE_BYTES)
                __asm__ __volatile__ ("dcache.cipa" : : "r" (asid) : "memory");

        __asm__ __volatile__("sync.s");
}
```

21.4.4　AMO 参考实现

以下内容来自 Linux 官方 riscv arch_atomic 和 cmpxchg 实现。

```
/*
 * First, the atomic ops that have no ordering constraints and therefor don't
 * have the AQ or RL bits set.  These don't return anything, so there's only
 * one version to worry about.
 */
#define ATOMIC_OP(op, asm_op, I, asm_type, c_type, prefix)              \
static __always_inline                                                  \
void atomic##prefix##_##op(c_type i, atomic##prefix##_t *v)             \
{                                                                       \
        __asm__ __volatile__ (                                          \
                "       amo" #asm_op "." #asm_type " zero, %1, %0"      \
                : "+A" (v->counter)                                     \
                : "r" (I)                                               \
                : "memory");                                            \
}                                                                       \

#ifdef CONFIG_GENERIC_ATOMIC64
#define ATOMIC_OPS(op, asm_op, I)                                       \
        ATOMIC_OP (op, asm_op, I, w, int,   )
```

(续下页)

(接上页)

```
#else
#define ATOMIC_OPS(op, asm_op, I)                                       \
        ATOMIC_OP (op, asm_op, I, w, int,   )                           \
        ATOMIC_OP (op, asm_op, I, d, s64, 64)
#endif

ATOMIC_OPS(add, add,  i)
ATOMIC_OPS(sub, add, -i)
ATOMIC_OPS(and, and,  i)
ATOMIC_OPS( or,  or,  i)
ATOMIC_OPS(xor, xor,  i)

#undef ATOMIC_OP
#undef ATOMIC_OPS

/*
 * Atomic ops that have ordered, relaxed, acquire, and release variants.
 * There's two flavors of these: the arithmatic ops have both fetch and return
 * versions, while the logical ops only have fetch versions.
 */
#define ATOMIC_FETCH_OP(op, asm_op, I, asm_type, c_type, prefix)        \
static __always_inline                                                  \
c_type atomic##prefix##_fetch_##op##_relaxed(c_type i,                  \
                                             atomic##prefix##_t *v)     \
{                                                                       \
        register c_type ret;                                            \
        __asm__ __volatile__ (                                          \
                "       amo" #asm_op "." #asm_type " %1, %2, %0"        \
                : "+A" (v->counter), "=r" (ret)                         \
                : "r" (I)                                               \
                : "memory");                                            \
        return ret;                                                     \
}                                                                       \
static __always_inline                                                  \
c_type atomic##prefix##_fetch_##op(c_type i, atomic##prefix##_t *v)      \
{                                                                       \
        register c_type ret;                                            \
        __asm__ __volatile__ (                                          \
                "       amo" #asm_op "." #asm_type ".aqrl  %1, %2, %0"  \
                : "+A" (v->counter), "=r" (ret)                         \
                : "r" (I)                                               \
                : "memory");                                            \
```

(续下页)

(接上页)

```
        return ret;                                                     \
}

#define ATOMIC_OP_RETURN(op, asm_op, c_op, I, asm_type, c_type, prefix)  \
static __always_inline                                                  \
c_type atomic##prefix##_##op##_return_relaxed(c_type i,                 \
                                              atomic##prefix##_t *v)    \
{                                                                       \
        return atomic##prefix##_fetch_##op##_relaxed(i, v) c_op I;      \
}                                                                       \
static __always_inline                                                  \
c_type atomic##prefix##_##op##_return(c_type i, atomic##prefix##_t *v)  \
{                                                                       \
        return atomic##prefix##_fetch_##op(i, v) c_op I;                \
}

#ifdef CONFIG_GENERIC_ATOMIC64
#define ATOMIC_OPS(op, asm_op, c_op, I)                                 \
        ATOMIC_FETCH_OP( op, asm_op,       I, w, int,   )               \
        ATOMIC_OP_RETURN(op, asm_op, c_op, I, w, int,   )
#else
#define ATOMIC_OPS(op, asm_op, c_op, I)                                 \
        ATOMIC_FETCH_OP( op, asm_op,       I, w, int,   )               \
        ATOMIC_OP_RETURN(op, asm_op, c_op, I, w, int,   )               \
        ATOMIC_FETCH_OP( op, asm_op,       I, d, s64, 64)               \
        ATOMIC_OP_RETURN(op, asm_op, c_op, I, d, s64, 64)
#endif

ATOMIC_OPS(add, add, +,  i)
ATOMIC_OPS(sub, add, +, -i)

#define atomic_add_return_relaxed       atomic_add_return_relaxed
#define atomic_sub_return_relaxed       atomic_sub_return_relaxed
#define atomic_add_return               atomic_add_return
#define atomic_sub_return               atomic_sub_return

#define atomic_fetch_add_relaxed        atomic_fetch_add_relaxed
#define atomic_fetch_sub_relaxed        atomic_fetch_sub_relaxed
#define atomic_fetch_add                atomic_fetch_add
#define atomic_fetch_sub                atomic_fetch_sub

#ifndef CONFIG_GENERIC_ATOMIC64
```

(续下页)

(接上页)

```
#define atomic64_add_return_relaxed     atomic64_add_return_relaxed
#define atomic64_sub_return_relaxed     atomic64_sub_return_relaxed
#define atomic64_add_return             atomic64_add_return
#define atomic64_sub_return             atomic64_sub_return

#define atomic64_fetch_add_relaxed      atomic64_fetch_add_relaxed
#define atomic64_fetch_sub_relaxed      atomic64_fetch_sub_relaxed
#define atomic64_fetch_add              atomic64_fetch_add
#define atomic64_fetch_sub              atomic64_fetch_sub
#endif

#undef ATOMIC_OPS

#ifdef CONFIG_GENERIC_ATOMIC64
#define ATOMIC_OPS(op, asm_op, I)                                       \
        ATOMIC_FETCH_OP(op, asm_op, I, w, int,   )
#else
#define ATOMIC_OPS(op, asm_op, I)                                       \
        ATOMIC_FETCH_OP(op, asm_op, I, w, int,   )                      \
        ATOMIC_FETCH_OP(op, asm_op, I, d, s64, 64)
#endif

ATOMIC_OPS(and, and, i)
ATOMIC_OPS( or,  or, i)
ATOMIC_OPS(xor, xor, i)

#define atomic_fetch_and_relaxed        atomic_fetch_and_relaxed
#define atomic_fetch_or_relaxed         atomic_fetch_or_relaxed
#define atomic_fetch_xor_relaxed        atomic_fetch_xor_relaxed
#define atomic_fetch_and                atomic_fetch_and
#define atomic_fetch_or                 atomic_fetch_or
#define atomic_fetch_xor                atomic_fetch_xor

#ifndef CONFIG_GENERIC_ATOMIC64
#define atomic64_fetch_and_relaxed      atomic64_fetch_and_relaxed
#define atomic64_fetch_or_relaxed       atomic64_fetch_or_relaxed
#define atomic64_fetch_xor_relaxed      atomic64_fetch_xor_relaxed
#define atomic64_fetch_and              atomic64_fetch_and
#define atomic64_fetch_or               atomic64_fetch_or
#define atomic64_fetch_xor              atomic64_fetch_xor
#endif
```

(续下页)

(接上页)

```
#undef ATOMIC_OPS

#undef ATOMIC_FETCH_OP
#undef ATOMIC_OP_RETURN

/* This is required to provide a full barrier on success. */
static __always_inline int atomic_fetch_add_unless(atomic_t *v, int a, int u)
{
       int prev, rc;

        __asm__ __volatile__ (
                "0:     lr.w     %[p],  %[c]\n"
                "       beq      %[p],  %[u], 1f\n"
                "       add      %[rc], %[p], %[a]\n"
                "       sc.w.rl  %[rc], %[rc], %[c]\n"
                "       bnez     %[rc], 0b\n"
                "       fence    rw, rw\n"
                "1:\n"
                : [p]"=&r" (prev), [rc]"=&r" (rc), [c]"+A" (v->counter)
                : [a]"r" (a), [u]"r" (u)
                : "memory");
        return prev;
}
#define atomic_fetch_add_unless atomic_fetch_add_unless

#ifndef CONFIG_GENERIC_ATOMIC64
static __always_inline s64 atomic64_fetch_add_unless(atomic64_t *v, s64 a, s64 u)
{
       s64 prev;
       long rc;

        __asm__ __volatile__ (
                "0:     lr.d     %[p],  %[c]\n"
                "       beq      %[p],  %[u], 1f\n"
                "       add      %[rc], %[p], %[a]\n"
                "       sc.d.rl  %[rc], %[rc], %[c]\n"
                "       bnez     %[rc], 0b\n"
                "       fence    rw, rw\n"
                "1:\n"
                : [p]"=&r" (prev), [rc]"=&r" (rc), [c]"+A" (v->counter)
                : [a]"r" (a), [u]"r" (u)
                : "memory");
```

(续下页)

(接上页)

```
        return prev;
}
#define atomic64_fetch_add_unless atomic64_fetch_add_unless
#endif

/*
 * atomic_{cmp,}xchg is required to have exactly the same ordering semantics as
 * {cmp,}xchg and the operations that return, so they need a full barrier.
 */
#define ATOMIC_OP(c_t, prefix, size)                                    \
static __always_inline                                                  \
c_t atomic##prefix##_xchg_relaxed(atomic##prefix##_t *v, c_t n)          \
{                                                                       \
        return __xchg_relaxed(&(v->counter), n, size);                  \
}                                                                       \
static __always_inline                                                  \
c_t atomic##prefix##_xchg_acquire(atomic##prefix##_t *v, c_t n)          \
{                                                                       \
        return __xchg_acquire(&(v->counter), n, size);                  \
}                                                                       \
static __always_inline                                                  \
c_t atomic##prefix##_xchg_release(atomic##prefix##_t *v, c_t n)          \
{                                                                       \
        return __xchg_release(&(v->counter), n, size);                  \
}                                                                       \
static __always_inline                                                  \
c_t atomic##prefix##_xchg(atomic##prefix##_t *v, c_t n)                  \
{                                                                       \
        return __xchg(&(v->counter), n, size);                          \
}                                                                       \
static __always_inline                                                  \
c_t atomic##prefix##_cmpxchg_relaxed(atomic##prefix##_t *v,              \
                                     c_t o, c_t n)                      \
{                                                                       \
        return __cmpxchg_relaxed(&(v->counter), o, n, size);            \
}                                                                       \
static __always_inline                                                  \
c_t atomic##prefix##_cmpxchg_acquire(atomic##prefix##_t *v,              \
                                     c_t o, c_t n)                      \
{                                                                       \
        return __cmpxchg_acquire(&(v->counter), o, n, size);            \
}                                                                       \
```

(续下页)

(接上页)

```
static __always_inline                                                  \
c_t atomic##prefix##_cmpxchg_release(atomic##prefix##_t *v,             \
                                     c_t o, c_t n)                      \
{                                                                       \
        return __cmpxchg_release(&(v->counter), o, n, size);            \
}                                                                       \
static __always_inline                                                  \
c_t atomic##prefix##_cmpxchg(atomic##prefix##_t *v, c_t o, c_t n)        \
{                                                                       \
        return __cmpxchg(&(v->counter), o, n, size);                    \
}

#ifdef CONFIG_GENERIC_ATOMIC64
#define ATOMIC_OPS()                                                    \
        ATOMIC_OP(int,   , 4)
#else
#define ATOMIC_OPS()                                                    \
        ATOMIC_OP(int,   , 4)                                           \
        ATOMIC_OP(s64, 64, 8)
#endif

ATOMIC_OPS()

#define atomic_xchg_relaxed atomic_xchg_relaxed
#define atomic_xchg_acquire atomic_xchg_acquire
#define atomic_xchg_release atomic_xchg_release
#define atomic_xchg atomic_xchg
#define atomic_cmpxchg_relaxed atomic_cmpxchg_relaxed
#define atomic_cmpxchg_acquire atomic_cmpxchg_acquire
#define atomic_cmpxchg_release atomic_cmpxchg_release
#define atomic_cmpxchg atomic_cmpxchg

#undef ATOMIC_OPS
#undef ATOMIC_OP

static __always_inline int atomic_sub_if_positive(atomic_t *v, int offset)
{
       int prev, rc;

        __asm__ __volatile__ (
                "0:     lr.w     %[p],  %[c]\n"
                "       sub      %[rc], %[p], %[o]\n"
```

(续下页)

(接上页)

```
                "       bltz    %[rc], 1f\n"
                "       sc.w.rl  %[rc], %[rc], %[c]\n"
                "       bnez    %[rc], 0b\n"
                "       fence   rw, rw\n"
                "1:\n"
                : [p]"=&r" (prev), [rc]"=&r" (rc), [c]"+A" (v->counter)
                : [o]"r" (offset)
                : "memory");
        return prev - offset;
}

#define atomic_dec_if_positive(v)       atomic_sub_if_positive(v, 1)

#ifndef CONFIG_GENERIC_ATOMIC64
static __always_inline s64 atomic64_sub_if_positive(atomic64_t *v, s64 offset)
{
       s64 prev;
       long rc;

        __asm__ __volatile__ (
                "0:     lr.d    %[p],  %[c]\n"
                "       sub     %[rc], %[p], %[o]\n"
                "       bltz    %[rc], 1f\n"
                "       sc.d.rl  %[rc], %[rc], %[c]\n"
                "       bnez    %[rc], 0b\n"
                "       fence   rw, rw\n"
                "1:\n"
                : [p]"=&r" (prev), [rc]"=&r" (rc), [c]"+A" (v->counter)
                : [o]"r" (offset)
                : "memory");
        return prev - offset;
}

#define __xchg_relaxed(ptr, new, size)                                  \
({                                                                      \
        __typeof__(ptr) __ptr = (ptr);                                  \
        __typeof__(new) __new = (new);                                  \
        __typeof__(*(ptr)) __ret;                                       \
        switch (size) {                                                 \
        case 4:                                                         \
                __asm__ __volatile__ (                                  \
                        "       amoswap.w %0, %2, %1\n"                 \
```

(续下页)

(接上页)

```
                        : "=r" (__ret), "+A" (*__ptr)                   \
                        : "r" (__new)                                   \
                        : "memory");                                    \
                break;                                                  \
        case 8:                                                         \
                __asm__ __volatile__ (                                  \
                        "       amoswap.d %0, %2, %1\n"                 \
                        : "=r" (__ret), "+A" (*__ptr)                   \
                        : "r" (__new)                                   \
                        : "memory");                                    \
                break;                                                  \
        default:                                                        \
                BUILD_BUG();                                            \
        }                                                               \
        __ret;                                                          \
})

#define xchg_relaxed(ptr, x)                                            \
({                                                                      \
        __typeof__(*(ptr)) _x_ = (x);                                   \
        (__typeof__(*(ptr))) __xchg_relaxed((ptr),                      \
                                            _x_, sizeof(*(ptr)));       \
})

#define __xchg_acquire(ptr, new, size)                                  \
({                                                                      \
        __typeof__(ptr) __ptr = (ptr);                                  \
        __typeof__(new) __new = (new);                                  \
        __typeof__(*(ptr)) __ret;                                       \
        switch (size) {                                                 \
        case 4:                                                         \
                __asm__ __volatile__ (                                  \
                        "       amoswap.w %0, %2, %1\n"                 \
                        RISCV_ACQUIRE_BARRIER                           \
                        : "=r" (__ret), "+A" (*__ptr)                   \
                        : "r" (__new)                                   \
                        : "memory");                                    \
                break;                                                  \
        case 8:                                                         \
                __asm__ __volatile__ (                                  \
                        "       amoswap.d %0, %2, %1\n"                 \
                        RISCV_ACQUIRE_BARRIER                           \
```

(续下页)

(接上页)

```
                        : "=r" (__ret), "+A" (*__ptr)                   \
                        : "r" (__new)                                   \
                        : "memory");                                    \
                break;                                                  \
        default:                                                        \
                BUILD_BUG();                                            \
        }                                                               \
        __ret;                                                          \
})

#define xchg_acquire(ptr, x)                                            \
({                                                                      \
        __typeof__(*(ptr)) _x_ = (x);                                   \
        (__typeof__(*(ptr))) __xchg_acquire((ptr),                      \
                                            _x_, sizeof(*(ptr)));       \
})

#define __xchg_release(ptr, new, size)                                  \
({                                                                      \
        __typeof__(ptr) __ptr = (ptr);                                  \
        __typeof__(new) __new = (new);                                  \
        __typeof__(*(ptr)) __ret;                                       \
        switch (size) {                                                 \
        case 4:                                                         \
                __asm__ __volatile__ (                                  \
                        RISCV_RELEASE_BARRIER                           \
                        "       amoswap.w %0, %2, %1\n"                 \
                        : "=r" (__ret), "+A" (*__ptr)                   \
                        : "r" (__new)                                   \
                        : "memory");                                    \
                break;                                                  \
        case 8:                                                         \
                __asm__ __volatile__ (                                  \
                        RISCV_RELEASE_BARRIER                           \
                        "       amoswap.d %0, %2, %1\n"                 \
                        : "=r" (__ret), "+A" (*__ptr)                   \
                        : "r" (__new)                                   \
                        : "memory");                                    \
                break;                                                  \
        default:                                                        \
                BUILD_BUG();                                            \
        }                                                               \
```

(续下页)

(接上页)

```
        __ret;                                                          \
})

#define xchg_release(ptr, x)                                            \
({                                                                      \
        __typeof__(*(ptr)) _x_ = (x);                                   \
        (__typeof__(*(ptr))) __xchg_release((ptr),                      \
                                            _x_, sizeof(*(ptr)));       \
})

#define __xchg(ptr, new, size)                                          \
({                                                                      \
        __typeof__(ptr) __ptr = (ptr);                                  \
        __typeof__(new) __new = (new);                                  \
        __typeof__(*(ptr)) __ret;                                       \
        switch (size) {                                                 \
        case 4:                                                         \
                __asm__ __volatile__ (                                  \
                        "       amoswap.w.aqrl %0, %2, %1\n"            \
                        : "=r" (__ret), "+A" (*__ptr)                   \
                        : "r" (__new)                                   \
                        : "memory");                                    \
                break;                                                  \
        case 8:                                                         \
                __asm__ __volatile__ (                                  \
                        "       amoswap.d.aqrl %0, %2, %1\n"            \
                        : "=r" (__ret), "+A" (*__ptr)                   \
                        : "r" (__new)                                   \
                        : "memory");                                    \
                break;                                                  \
        default:                                                        \
                BUILD_BUG();                                            \
        }                                                               \
        __ret;                                                          \
})

#define xchg(ptr, x)                                                    \
({                                                                      \
        __typeof__(*(ptr)) _x_ = (x);                                   \
        (__typeof__(*(ptr))) __xchg((ptr), _x_, sizeof(*(ptr)));        \
})
```

(续下页)

(接上页)

```
#define xchg32(ptr, x)                                          \
({                                                              \
        BUILD_BUG_ON(sizeof(*(ptr)) != 4);                      \
        xchg((ptr), (x));                                       \
})

#define xchg64(ptr, x)                                          \
({                                                              \
        BUILD_BUG_ON(sizeof(*(ptr)) != 8);                      \
        xchg((ptr), (x));                                       \
})

/*
 * Atomic compare and exchange.  Compare OLD with MEM, if identical,
 * store NEW in MEM.  Return the initial value in MEM.  Success is
 * indicated by comparing RETURN with OLD.
 */
#define __cmpxchg_relaxed(ptr, old, new, size)                  \
({                                                              \
        __typeof__(ptr) __ptr = (ptr);                          \
        __typeof__(*(ptr)) __old = (old);                       \
        __typeof__(*(ptr)) __new = (new);                       \
        __typeof__(*(ptr)) __ret;                               \
        register unsigned int __rc;                             \
        switch (size) {                                         \
        case 4:                                                 \
                __asm__ __volatile__ (                          \
                        "0:     lr.w %0, %2\n"                  \
                        "       bne  %0, %z3, 1f\n"             \
                        "       sc.w %1, %z4, %2\n"             \
                        "       bnez %1, 0b\n"                  \
                        "1:\n"                                  \
                        : "=&r" (__ret), "=&r" (__rc), "+A" (*__ptr)     \
                        : "rJ" ((long)__old), "rJ" (__new)      \
                        : "memory");                            \
                break;                                          \
        case 8:                                                 \
                __asm__ __volatile__ (                          \
                        "0:     lr.d %0, %2\n"                  \
                        "       bne %0, %z3, 1f\n"              \
                        "       sc.d %1, %z4, %2\n"             \
                        "       bnez %1, 0b\n"                  \
```

(续下页)

(接上页)

```
                        "1:\n"                                          \
                        : "=&r" (__ret), "=&r" (__rc), "+A" (*__ptr)    \
                        : "rJ" (__old), "rJ" (__new)                    \
                        : "memory");                                    \
                break;                                                  \
        default:                                                        \
                BUILD_BUG();                                            \
        }                                                               \
        __ret;                                                          \
})

#define cmpxchg_relaxed(ptr, o, n)                                      \
({                                                                      \
        __typeof__(*(ptr)) _o_ = (o);                                   \
        __typeof__(*(ptr)) _n_ = (n);                                   \
        (__typeof__(*(ptr))) __cmpxchg_relaxed((ptr),                   \
                                        _o_, _n_, sizeof(*(ptr)));      \
})

#define __cmpxchg_acquire(ptr, old, new, size)                          \
({                                                                      \
        __typeof__(ptr) __ptr = (ptr);                                  \
        __typeof__(*(ptr)) __old = (old);                               \
        __typeof__(*(ptr)) __new = (new);                               \
        __typeof__(*(ptr)) __ret;                                       \
        register unsigned int __rc;                                     \
        switch (size) {                                                 \
        case 4:                                                         \
                __asm__ __volatile__ (                                  \
                        "0:     lr.w %0, %2\n"                          \
                        "       bne  %0, %z3, 1f\n"                     \
                        "       sc.w %1, %z4, %2\n"                     \
                        "       bnez %1, 0b\n"                          \
                        RISCV_ACQUIRE_BARRIER                           \
                        "1:\n"                                          \
                        : "=&r" (__ret), "=&r" (__rc), "+A" (*__ptr)    \
                        : "rJ" ((long)__old), "rJ" (__new)              \
                        : "memory");                                    \
                break;                                                  \
        case 8:                                                         \
                __asm__ __volatile__ (                                  \
                        "0:     lr.d %0, %2\n"                          \
```

(续下页)

(接上页)

```
                        "       bne %0, %z3, 1f\n"                      \
                        "       sc.d %1, %z4, %2\n"                     \
                        "       bnez %1, 0b\n"                          \
                        RISCV_ACQUIRE_BARRIER                           \
                        "1:\n"                                          \
                        : "=&r" (__ret), "=&r" (__rc), "+A" (*__ptr)     \
                        : "rJ" (__old), "rJ" (__new)                    \
                        : "memory");                                    \
                break;                                                  \
        default:                                                        \
                BUILD_BUG();                                            \
        }                                                               \
        __ret;                                                          \
})

#define cmpxchg_acquire(ptr, o, n)                                      \
({                                                                      \
        __typeof__(*(ptr)) _o_ = (o);                                   \
        __typeof__(*(ptr)) _n_ = (n);                                   \
        (__typeof__(*(ptr))) __cmpxchg_acquire((ptr),                   \
                                        _o_, _n_, sizeof(*(ptr)));      \
})

#define __cmpxchg_release(ptr, old, new, size)                          \
({                                                                      \
        __typeof__(ptr) __ptr = (ptr);                                  \
        __typeof__(*(ptr)) __old = (old);                               \
        __typeof__(*(ptr)) __new = (new);                               \
        __typeof__(*(ptr)) __ret;                                       \
        register unsigned int __rc;                                     \
        switch (size) {                                                 \
        case 4:                                                         \
                __asm__ __volatile__ (                                  \
                        RISCV_RELEASE_BARRIER                           \
                        "0:     lr.w %0, %2\n"                          \
                        "       bne  %0, %z3, 1f\n"                     \
                        "       sc.w %1, %z4, %2\n"                     \
                        "       bnez %1, 0b\n"                          \
                        "1:\n"                                          \
                        : "=&r" (__ret), "=&r" (__rc), "+A" (*__ptr)     \
                        : "rJ" ((long)__old), "rJ" (__new)              \
                        : "memory");                                    \
```

(续下页)

(接上页)

```
                break;                                                  \
        case 8:                                                         \
                __asm__ __volatile__ (                                  \
                        RISCV_RELEASE_BARRIER                           \
                        "0:     lr.d %0, %2\n"                          \
                        "       bne %0, %z3, 1f\n"                      \
                        "       sc.d %1, %z4, %2\n"                     \
                        "       bnez %1, 0b\n"                          \
                        "1:\n"                                          \
                        : "=&r" (__ret), "=&r" (__rc), "+A" (*__ptr)     \
                        : "rJ" (__old), "rJ" (__new)                    \
                        : "memory");                                    \
                break;                                                  \
        default:                                                        \
                BUILD_BUG();                                            \
        }                                                               \
        __ret;                                                          \
})

#define cmpxchg_release(ptr, o, n)                                      \
({                                                                      \
        __typeof__(*(ptr)) _o_ = (o);                                   \
        __typeof__(*(ptr)) _n_ = (n);                                   \
        (__typeof__(*(ptr))) __cmpxchg_release((ptr),                   \
                                       _o_, _n_, sizeof(*(ptr)));       \
})

#define __cmpxchg(ptr, old, new, size)                                  \
({                                                                      \
        __typeof__(ptr) __ptr = (ptr);                                  \
        __typeof__(*(ptr)) __old = (old);                               \
        __typeof__(*(ptr)) __new = (new);                               \
        __typeof__(*(ptr)) __ret;                                       \
        register unsigned int __rc;                                     \
        switch (size) {                                                 \
        case 4:                                                         \
                __asm__ __volatile__ (                                  \
                        "0:     lr.w %0, %2\n"                          \
                        "       bne  %0, %z3, 1f\n"                     \
                        "       sc.w.rl %1, %z4, %2\n"                  \
                        "       bnez %1, 0b\n"                          \
                        "       fence rw, rw\n"                         \
```

(续下页)

(接上页)

```
                        "1:\n"                                          \
                        : "=&r" (__ret), "=&r" (__rc), "+A" (*__ptr)    \
                        : "rJ" ((long)__old), "rJ" (__new)              \
                        : "memory");                                    \
                break;                                                  \
        case 8:                                                         \
                __asm__ __volatile__ (                                  \
                        "0:     lr.d %0, %2\n"                          \
                        "       bne %0, %z3, 1f\n"                      \
                        "       sc.d.rl %1, %z4, %2\n"                  \
                        "       bnez %1, 0b\n"                          \
                        "       fence rw, rw\n"                         \
                        "1:\n"                                          \
                        : "=&r" (__ret), "=&r" (__rc), "+A" (*__ptr)    \
                        : "rJ" (__old), "rJ" (__new)                    \
                        : "memory");                                    \
                break;                                                  \
        default:                                                        \
                BUILD_BUG();                                            \
        }                                                               \
        __ret;                                                          \
})
```