C920

用户手册 (xrvm)

此手册支持的 CPU 版本为 R3S1

修订版 11

2025-11-05

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

修订版本	描述	日期
01	初版	2024-09-05
02	更新寄存器 MCCR2 描述并修改手册版本号为 R3S1	2024-11-15
03	更新玄铁扩展指令集、协处理器指令集相关描述	2024-12-17
04	新增同步调试与异步调试小节	2025-02-25
05	更新 TLB 缺失时地址转换步骤描述 更新寄存器 scountovf bit 描述	2025-03-13
06	更新寄存器 pmpaddr 相关描述 修改异常发生时寄存器 mtval 的更新值相关描述	2025-06-05
07	更新文档模板	2025-08-11
08	更新计数器事件对应列表 更新寄存器 mcer 描述	2025-09-04
09	更新寄存器 mcdata0/1 描述	2025-09-10
10	更新调试接口的主要特性相关描述	2025-10-15
11	新增玄铁扩展调试寄存器相关描述	2025-10-21

目录

图目录

表目录

1 概述

1.1 简介

C920V3 是基于 RISC-V 指令架构的 64 位高性能多核心处理器，主要面向对性能要求严格的边缘计算领域，如边缘服务器、边缘计算卡、高端机器视觉、高端视频监控、自动驾驶、移动智能终端、5G 基站等。C920V3 采用同构多核架构，支持 1~4 个 C920V3 核心可配置。每个 C920V3 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，引入 4 译码 10 执行的超标量架构和多通道的数据预取等高性能技术。此外，C920V3 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 C920V3 处理器体系结构的主要特点

- 同构多核架构，支持 1~4 个 C920V3 核心可配置；
- 支持各个核心独立下电以及 cluster 下电；
- 支持 1 个 AMBA 4.0 AXI/ACE Master 接口，128 比特的总线宽度；地址位宽 40 bit 和 48 bit 可配。
- 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；
- 两级高缓结构，哈佛结构一级高缓和共享的二级高缓；
- 一级缓存大小可配置，指令/数据缓存分别支持 32KB/64KB，缓存行 SIZE 为 64B；
- 一级缓存支持 MESI 的一致性协议，二级缓存支持 MESI 的一致性协议；
- 二级高缓支持 16 路组相联，可配置 ECC 校验机制；
- 二级高缓大小可配置，支持 256KB/512KB/1MB/2MB/4MB/8MB，缓存行 SIZE 为 64B；
- 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；
- 支持计时器功能；
- 支持 RISC-V 调试框架，支持多核多 cluster 调试；
- 支持协处理器。

1.2.2 C920V3 核心的主要特点

- RISC-V 64GC[V] 指令架构；
- 支持小端模式；
- 9~12 级深流水架构；
- 4 译码 10 执行的超标量架构，对软件完全透明；
- 按序取指，乱序发射，乱序完成和按序退休；
- 两级 TLB 内存管理单元，实现虚实地址转换与内存管理；
- 指令高缓和数据高缓大小可配置，支持 32KB、64KB，缓存行为 64B；
- 指令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；
- 短循环缓存的低功耗执行技术；
- 64Kb 的两级多路并行分支预测器；
- 1024/2048 表项可配置的分支目标缓存器；
- 支持 12 层的硬件返回地址堆栈；
- 256 表项的间接跳转分支预测器；
- 非阻塞发射，投机猜测执行；
- 基于物理寄存器的重命名技术；
- 支持 0 延时 move 指令；
- 双发射、全乱序执行的 load、store 指令；
- 支持读写各 8 路并发的总线访问；
- 支持写合并；
- 支持 8 个通道的数据缓存硬件预取，支持 stride 的预取方式；
- 浮点执行单元固定配置，支持半精度、单精度；

1.2.3 矢量计算单元的主要特点

- 遵循 RISC-V V 扩展；
- 在 4 核、2GHz 下，算力可达 512GOPS (@int8)/256GFlops (@FP16)；
- 矢量执行单元支持 FP16/BF16/FP32 浮点和 INT8/INT16/INT32/INT64 整型的矢量运算；
- 支持 128 位矢量寄存器位宽 VLEN；
- 支持两条矢量执行单元运算和数据存储流水线；
- 支持 128 位矢量数据存储访问位宽 VLEN；

- 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

C920V3 的可配置选项如 表 1.1 所示。

表 1.1: C920 可配置选项

可配置单元	配置选项	详细
CPU Arch	RV64GC/ RV64GCV	矢量执行单元可配
C920V3 核心数量	1/2/3/4	C920V3 提供 1 ~ 4 个核心可配。
BTB Entry	1024/2048	分支目标缓存器支持 1024/2048 表项可配
PMP region	8/16/32/64	PMP 区域个数可配
ePMP	无/有	Enhanced PMP 功能可配置
MMU	SV39/SV48	支持 SV39 或 SV48 模式，配置为 SV48 可以同时支持 SV39。
L1 ICache	32K/64K	可以配置 32KB、64KB。
L1 DCache	32K/64K	可以配置 32KB、64KB。
L1 ECC/Parity	无/有	L1 Icache 的 Parity 校验 L1 Dcache 的 ECC 校验
L2 Cache	Size: 256K/512K/1M 2M/4M/8M	可以配置 256KB~8MB
L2 ECC	无/有	L2 Tag/Data RAM 的 ECC 校验。
L2 RAM 延时	支持配置	L2 Tag RAM Setup: No flop/Flop L2 Data RAM Setup: No flop/Flop L2 Tag RAM Access: 1~5 cycles L2 Data RAM Access: 1~8 cycles
Master Addr Width	PA40/PA48	总线地址位宽支持 40 或 48 bit 可配
Master 接口协议	AXI/ACE	Master 接口支持 AXI 或 ACE 协议。
LLP	无/有	低延时外设端口可选。
DCP	无/有	用于外设对片上高速缓存的访问，实现数据一致性，可用于连接 DMA。
Co-Processor Interface	有/无	协处理器接口可配
TEE	无/有	TEE 扩展可选
Debug Resources	Min/Typical /Max	Debug 资源可配

续下页

表 1.1 - 接上页

可配置单元	配置选项	详细
System Bus Access	无/有	Debug 是否支持 System Bus Access 功能。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	64-1024， step 32	中断数量
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	无/有	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	无/有	Sys APB Access 允许 CPU 通过 Master Port 和系统总线访问调试寄存器。

1.4 玄铁架构扩展技术

C920V3 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

C920V3 兼容 RISC-V 标准，具体信息如下 表 1.2 和 表 1.3 所示。

表 1.2: C920V3 编程模型及对应版本

Specification	C920V3
RISC-V Profile	RVA23
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

续下页

表 1.2 - 接上页

Specification	C920V3
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	Version 0.13.2
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
RISC-V Code Size Reduction (Zc)	v1.0
RISC-V WAIT-on-Reservation-Set(Zawrs)	Version 1.0 11/2022
玄铁扩展指令集	支持

表 1.3: C920V3 指令集及对应版本

Specification	Modules	C920V3
RISC-V Instruction Set Maunal Volume I: User-Level ISA	Width of an integer register in bits (XLEN)	RV64
	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 bfloat16 Specification	Version 1.0.0-rc1, 27 October 2023: Frozen
	Misc. basic Scalar FP (Zfa)	支持
	Non-Temporal Locality Hints (Zihintntl)	支持
	RISC-V Wait-on-Reservation-Set (Zawrs)	支持
	ZiCond	Version 1.0, 2023-02-22
RISC-V Instruction Set Manual Volume II: Privileged Architecture	Virtual Memory System	SV39 + SV48
	NAPOT Translation Contiguity (Svnapot)	Version 1.0

续下页

表 1.3 - 接上页

Specification	Modules	C920V3
	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
	Count Overflow and Mode-Based Filtering Extension (Sscofpmf)	支持
	Base Cache Management Operation ISA Extensions (Zicbom, Zicboz, Zicbop)	Version 1.0.1-b34ea8a, 2022-05-13: Ratified
RISC-V "V" Vector Extension	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
RISC-V Code Size Reduction (Zc)	Zca, Zcf, Zcb, Zcmb, Zcmp, Zcmpe, Zcmt	Zca, Zcd, Zcb

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。

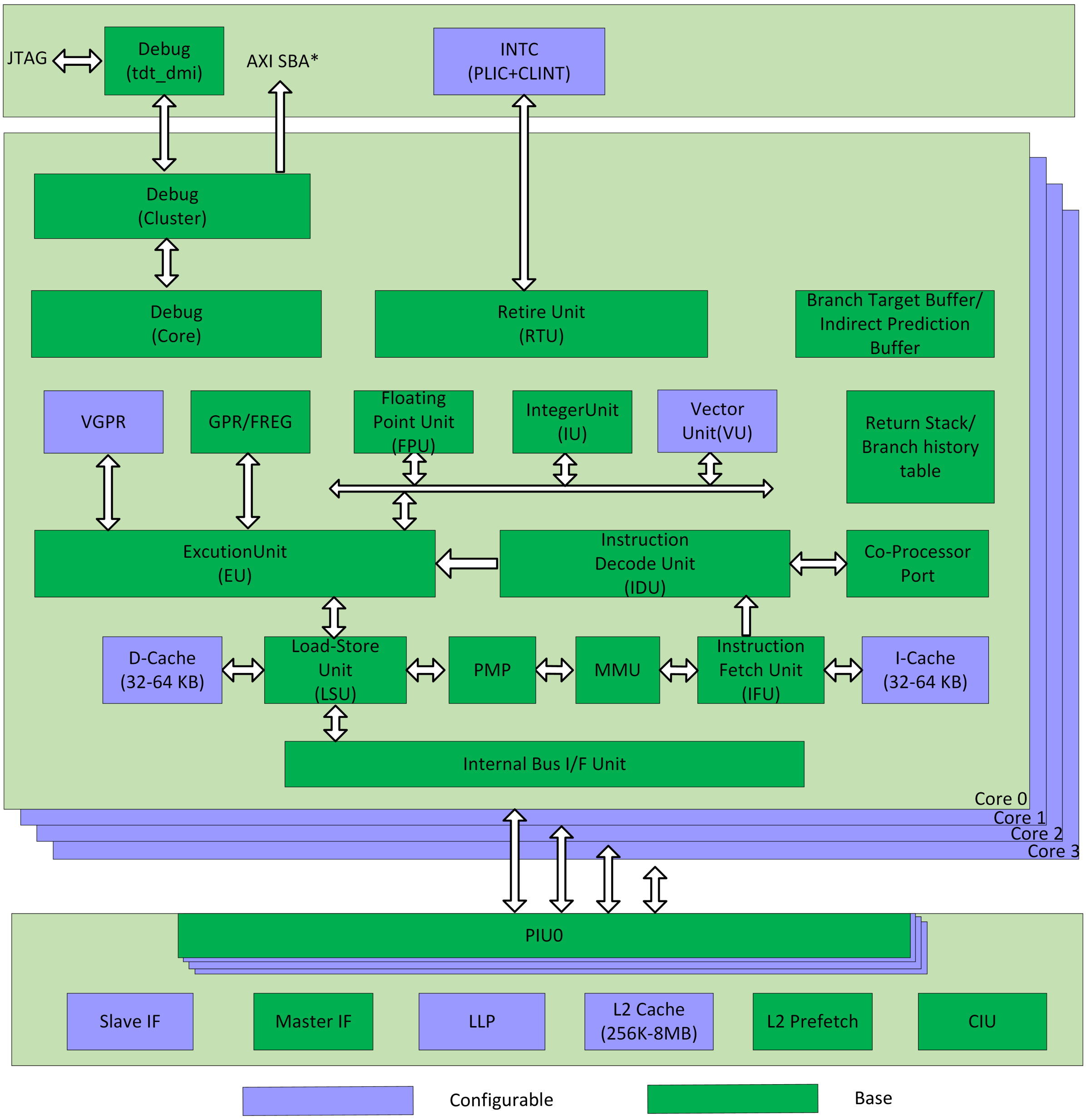
  例如"pad_cpu_rst_b"就表示单独的一个信号。

  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

C920V3 结构框图如 图 2.1 所示。



＊ ： 该特性可配。

图 2.1: C920V3 微体系结构图

2.2 核内子系统

C920V3 核内子系统主要包含：指令提取单元（IFU）、指令译码单元（IDU）、整型执行单元（IU）、浮点单元（FPU）、矢量执行单元（VU）、存储载入单元（LSU）、指令退休单元（RTU）、虚拟内存管理单元（MMU）和物理内存保护单元（PMP）。

2.2.1　指令提取单元

指令提取单元（IFU)，一次可最多提取八条指令并对其并行处理。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，循环加速缓存器，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　指令译码单元

指令译码单元（IDU）可以同时对四条指令进行译码并检测数据相关性。利用物理寄存器重命名技术解决指令间的数据相关性，并将指令乱序发送至下级流水线执行。指令译码单元支持指令的乱序调度分发，并通过投机性的发射缓解因数据相关性造成的性能损失。

2.2.3　执行单元

执行单元包含整型单元（IU）、浮点单元（FPU）以及矢量执行单元（VU）。

整型单元包含算术逻辑单元（ALU）、乘法单元（MULT）、除法单元（DIV）和跳转单元（BJU）。ALU 执行 64 位整数操作。MULT 支持 16*16、32*32、64*64 整数乘法。除法器的设计采用了基 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。

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。

矢量执行单元在浮点单元的基础上扩展而成。浮点单元在支持原有标量浮点计算的基础上，扩展成为矢量浮点单元。矢量浮点单元包括矢量浮点算数逻辑单元（VFALU）、矢量浮点乘累加单元（VFMAU）和矢量浮点除法开方单元（VFDSU），支持不同位宽的矢量浮点运算。

另外，新增了矢量整型单元。矢量整型单元包含矢量加法单元（VALU）、矢量移位单元 (VSHIFT)、矢量乘法单元（VMUL）、矢量除法单元（VDIVU）、矢量置换单元（VPERM）、矢量缩减单元（VREDU）以及矢量逻辑操作单元（VMISC）。

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/加载指令的单发射以及所有存储/加载指令的全乱序执行，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　指令退休单元

指令退休单元（RTU）包括一个重排序缓冲器与一个物理寄存器堆。其中，重排序缓冲器负责指令的乱序回收与按序退休，物理寄存器堆负责结果的乱序回收和传递。通过支持指令并行回收与快速退休提高指令退休效率。指令退休单元每个时钟周期并行退休三条指令，支持精确异常。

2.2.6 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 40/48 位物理地址。C920V3 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*。

2.2.7 物理内存保护单元

C920V3 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。

具体信息参考*内存模型*。

2.3 多核子系统

C920V3 多核子系统包含：数据一致性接口单元（CIU）、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）、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。采用高效的数据旁路机制，当监听请求命中被监听的 L1 数据高缓时，直接将数据旁路给请求发起核心；另外，CIU 单元还支持 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；采用分块的流水线架构，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 C920V3 相同的工作频率，TAG RAM 和 DATA RAM 的访问延时可以由软件配置。

2.3.3 主设备接口

主设备接口单元支持 ACE/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

设备一致性接口单元支持 AXI4.0 协议，可用于外设对片上高速数据缓存的访问，硬件实现数据一致性，可以用来连接外部 DMA。

2.3.5 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。由于具有单独的数据通路，LLP 不会受到主接口带宽的影响。

2.4 多 cluster 子系统

C920V3 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。C920V3 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 调试系统

C920V3 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 。

具体信息请参考 *调试* 章节。

2.5 接口概览

C920V3 的接口按照功能主要分为：时钟复位信号、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。C920V3 的主要接口如 图 2.2 所示。

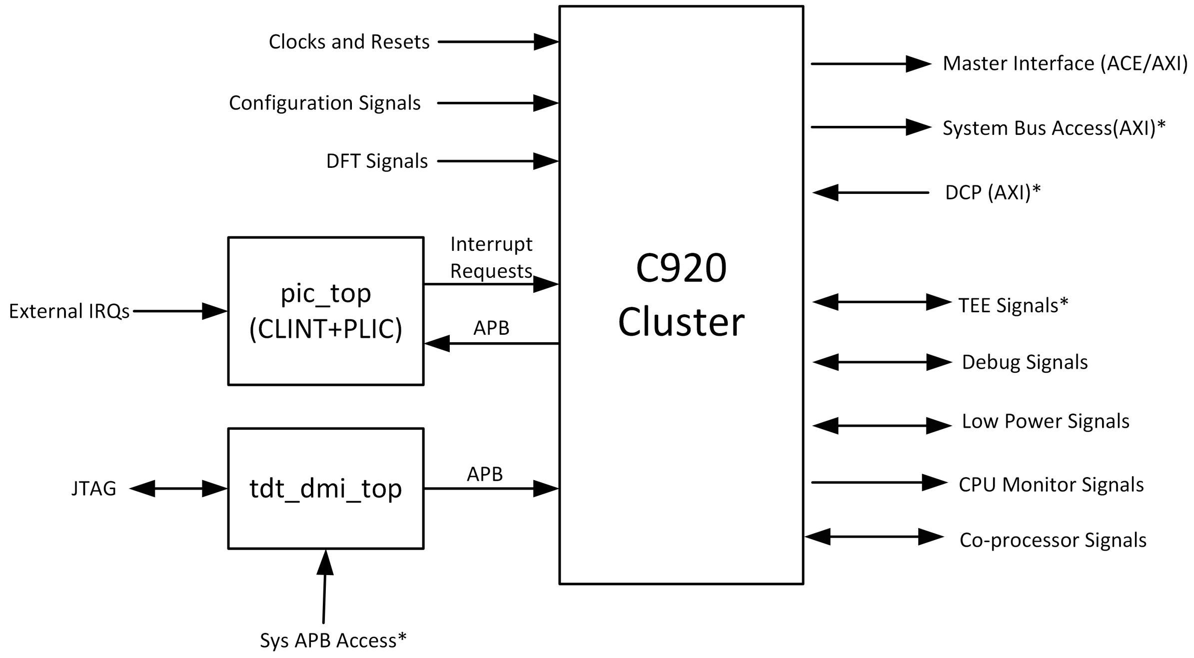


图 2.2: C920V3 接口概览

3 指令集

本章主要介绍 C920V3 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
控制寄存器操作指令		
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	3
FMADD.S	单精度浮点乘累加指令	4
FMSUB.S	单精度浮点乘累减指令	4
FNMADD.S	单精度浮点乘累加取负指令	4
FNMSUB.S	单精度浮点乘累减取负指令	4
FDIV.S	单精度浮点除法指令	8-14
FSQRT.S	单精度浮点开方指令	8-14
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	不拆分，3
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 1+1
FLT.S	单精度浮点比较小于指令	拆分执行 1+1
FLE.S	单精度浮点比较小于等于指令	拆分执行 1+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	不拆分，3
FCVT.S.WU	无符号整型转换成单精度浮点指令	不拆分，3
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	不拆分，3
FCVT.S.LU	无符号长整型转换成单精度浮点指令	不拆分，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 。

3.2　玄铁扩展指令集

C920V3 除了支持 RV64GCV 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、Cache 指令子集、多核同步指令子集、半精度浮点指令集、玄铁协处理器扩展指令。

C920V3 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、AI 扩展指令仅在 mxstatus.XUANTIEISAEE == 1、mxstatus.COPINSTEE == 0 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、mxstatus.COPINSTEE == 0 或 mxstatus.XUANTIEISAEE == 0、mxstatus.COPINSTEE == 1 时可以正常执行；
- 玄铁协处理器扩展指令需要在 mxstatus.XUANTIEISAEE == 0、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*。

以下按照不同扩展指令子集对每条指令做具体描述。

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-1 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-2 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

存储指令	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	寄存器移位无符号位扩展半字加载指令	
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	
LWUIA	零拓展字加载地址自增指令	

续下页

表 3.8 - 接上页

存储指令	指令描述	执行延时
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-3 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-4 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
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-5 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	8-11
FSQRT.H	半精度浮点开方指令	8-11
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	拆分执行 1+1
FMV.H.X	半精度浮点写传送指令	3
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 1+1
FLT.H	半精度浮点比较小于指令	拆分执行 1+1
FLE.H	半精度浮点比较小于等于指令	拆分执行 1+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	不拆分，3
FCVT.H.WU	无符号整型转换成半精度浮点指令	不拆分，3
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	不拆分，3
FCVT.H.LU	无符号长整型转换成半精度浮点指令	不拆分，3
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	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 B-6 浮点半精度指令术语*。

3.2.7 玄铁协处理器扩展指令

3.2.7.1 指令编码

ISA 设计遵循 RISC-V 指令集规范，使用 RISC-V 扩展指令和 Custom CSR 扩展。

- 选择 custom-1（RISC-V Spec Chapter 24）自定义指令扩展码。

inst[4:2] inst[6:5]	000	001	010	011	100	101	110	111 (> 32b)
00	LOAD	LOAD-FP	*custom-0*	MISC-MEM	OP-IMM	AUIPC	OP-IMM-32	48b
01	STORE	STORE-FP	*custom-1*	AMO	OP	LUI	OP-32	64b
10	MADD	MSUB	NMSUB	NMADD	OP-FP	*reserved*	*custom-2/rv128*	48b
11	BRANCH	JALR	*reserved*	JAL	SYSTEM	*reserved*	*custom-3/rv128*	$\geq$ 80b

图 3.1: RISC-V base opcode map, ins[1:0]=11

- 使用 Unprivileged and User-Level CSRs - Custom read/write CSRs（RISC-V Privileged Spec Chapter 2.2）作

为自定义扩展寄存器。

CSR Address			Hex	Use and Accessibility
[11:10]	[9:8]	[7:4]		
Unprivileged and User-Level CSRs				
00	00	XXXX	0x000-0x0FF	Standard read/write
01	00	XXXX	0x400-0x4FF	Standard read/write
10	00	XXXX	0x800-0x8FF	Custom read/write
11	00	0XXX	0xC00-0xC7F	Standard read-only
11	00	10XX	0xC80-0xCBF	Standard read-only
11	00	11XX	0xCC0-0xCFF	Custom read-only

图 3.2: RISC-V base opcode map, ins[1:0]=11

3.2.7.2 玄铁协处理器扩展指令列表

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 custom1 域。C920V3 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 3.12: 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
vector 类型指令	VCPX0
	VCPX1
	VCPX2
	VCPX3
	VCPX4
	VCPX5

续下页

表 3.12 - 接上页

分类	指令名称
	VCPX6
	VCPX7
	VCPX8
	VCPX9
	VCPX10
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-7 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

C920V3 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

C920V3 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

C920V3 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
x10-11	a0-1	函数参数/返回值
x12-17	a2-7	函数参数

续下页

表 4.1 - 接上页

寄存器	ABI 名称	描述
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

C920V3 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

C920V3 拥有 32 个独立的矢量架构寄存器，位宽为 128 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

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 C920V3 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。

C920V3 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	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
机器模式环境配置寄存器组			
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
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
机器模式计时和计数寄存器组			
mcycle	机器模式读写	0xB00	机器模式周期计数器

续下页

表 4.3 - 接上页

名称	读写权限	寄存器编号	描述
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	机器模式性能监测事件选择寄存器 3
……			
mhpmevent31	机器模式读写	0x33F	机器模式性能监测事件选择寄存器 31
调试/追踪寄存器组/Trace 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/Trace 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

C920V3 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
超级用户模式处理器控制和状态扩展寄存器组			
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
scountovf	超级用户模式只读	0xDA0	超级用户模式计数器溢出寄存器

续下页

表 4.4 - 接上页

名称	读写权限	寄存器编号	描述
超级用户模式环境配置寄存器组			
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
超级用户模式地址保护寄存器组			
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
超级用户模式调试寄存器组			
scontext	超级用户模式读写	0x5A8	超级用户模式场景寄存器
超级用户模式计时和计数寄存器组			
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

C920V3 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
用户模式计时和计数器寄存器组			
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
vxsat	用户模式读写	0x009	定点溢出标志位寄存器

续下页

表 4.5 - 接上页

名称	读写权限	寄存器编号	描述
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	矢量寄存器字节数寄存器

4.6.2 扩展控制寄存器

本章节描述 C920V3 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。

C920V3 中扩展的机器模式控制寄存器如 表 4.6 所示。

表 4.6: C920V3 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 ECC 寄存器
mhint	机器模式读写	0x7C5	机器模式隐式操作寄存器
mrmr	机器模式读写	0x7C6	机器模式复位寄存器
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
机器模式扩展寄存器组 2			
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
msmpr	机器模式读写	0x7F3	Snoop 监听使能寄存器
mzoneid	机器模式读写	0x7F5	zone id 寄存器
mllcpid	机器模式读写	0x7F6	细粒度回填 ID 寄存器

续下页

表 4.6 - 接上页

名称	读写权限	寄存器编号	描述
mllwp	机器模式读写	0x7F7	L2 细粒度配置寄存器
msbepa	机器模式读写	0x7FB	机器模式 L1 Cache ECC 单比特错误物理地址寄存器
msbepa2	机器模式读写	0x7FC	机器模式 L2 Cache ECC 单比特错误物理地址寄存器
机器模式 Cache 访问扩展寄存器组			
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	机器模式读写	0x7D4	机器模式 Cache 数据寄存器 0
mcdata1	机器模式读写	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	机器模式读写	0x7D8	L1 LD BUS ERROR 地址寄存器
mcper	机器模式读写	0x7D9	TEE 下 cache 权限控制寄存器
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
mapbaddr2	机器模式只读	0xFC3	片上系统互联寄存器基地址
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

注意：

mrmr 寄存器在 C920V3（R1S4 及以上）中已经被删除。软件仍然可以访问该寄存器，结果是读为零、写无效，不会触发异常。

具体寄存器的定义和功能，请参考 *附录 C-1 RISC-V 标准机器模式控制寄存器* 。

C920V3 中扩展的超级用户模式控制寄存器如 表 4.7 所示。

表 4.7: C920V3 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

续下页

表 4.7 - 接上页

名称	读写权限	寄存器编号	描述
scer2	超级用户模式只读	0x5C2	超级用户模式 L2 Cache ECC 寄存器
scer	超级用户模式只读	0x5C3	超级用户模式 L1 Cache ECC 寄存器
shint	超级用户模式读写	0x5C6	超级用户模式隐式寄存器
shint2	超级用户模式读写	0x5C7	超级用户模式隐式寄存器 2
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	超级用户模式性能监测控制起始触发寄存器
shpmer	超级用户模式读写	0x5CB	超级用户模式性能监测控制起始终止寄存器
sllcpid	超级用户模式读写	0x5CC	细粒度回填 ID 寄存器
sllwp	超级用户模式读写	0x5CD	L2 细粒度配置寄存器
sbeaddr	超级用户模式读写	0x5D0	S 态 BUS error 寄存器
ssbepa	超级用户模式读写	0x5D1	S 态 L1 Cache ECC 单比特错误物理地址寄存器
ssbepa2	超级用户模式读写	0x5D2	S 态 L2 Cache ECC 单比特错误物理地址寄存器
scycle	超级用户模式读写	0x5E0	超级用户模式周期计数器
sinstret	超级用户模式读写	0x5E2	超级用户模式指令退休计数器
shpmcounter3	超级用户模式读写	0x5F3	超级用户模式计数器 3
……			
shpmcounter31	超级用户模式读写	0x5FF	超级用户模式计数器 31

具体寄存器的定义和功能，请参考 *附录 C-2 RISC-V 标准超级用户模式控制寄存器*

C920V3 中扩展的用户模式控制寄存器如 表 4.8 所示。

表 4.8: C920V3 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 RISC-V 标准用户模式控制寄存器* 。

4.7 数据格式

4.7.1 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。

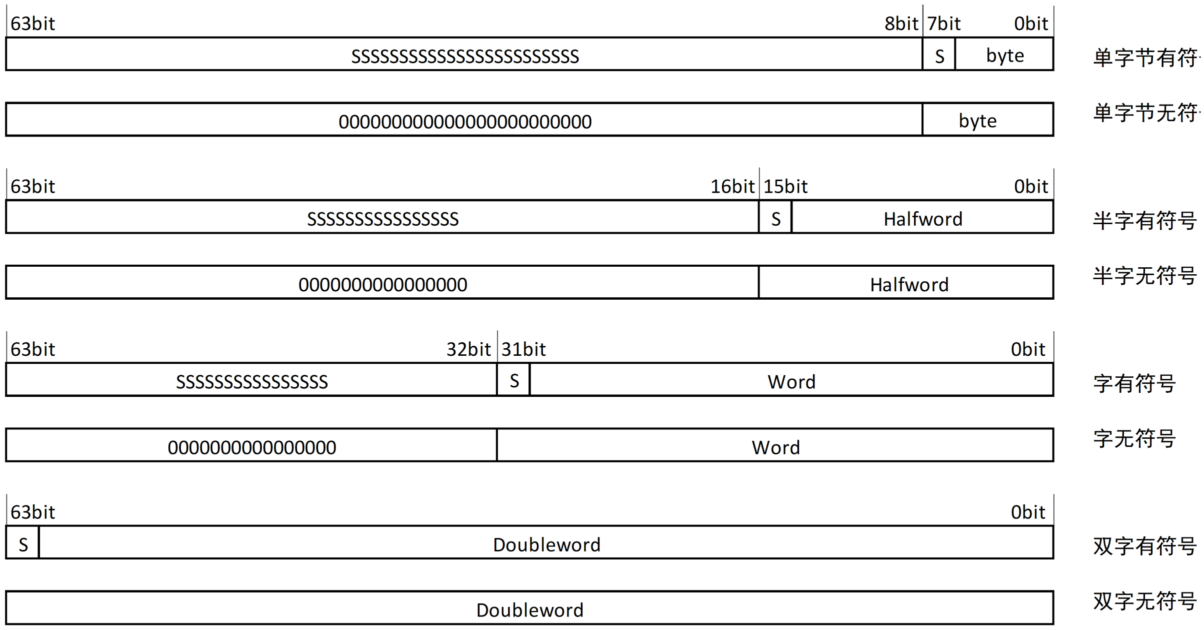


图 4.2: 寄存器中的整型数据组织结构

4.7.2　浮点数据格式

C920V3 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。

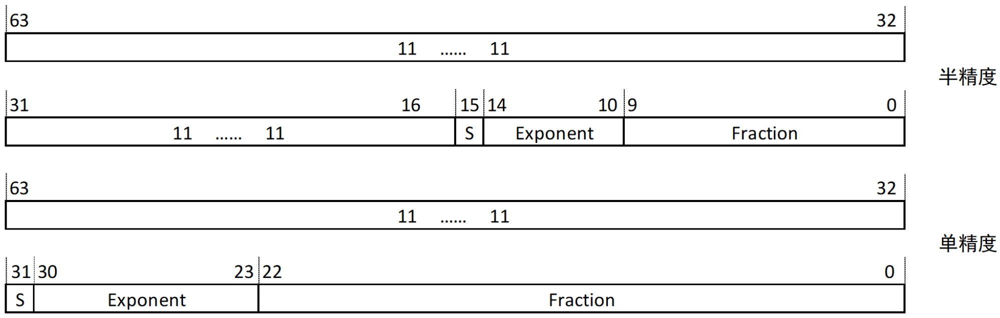


图 4.3: 寄存器中的浮点数据组织结构

4.7.3　矢量数据格式

矢量寄存器的宽度为 128 位，矢量寄存器中包含的元素个数由当前矢量元素位宽决定。C920V3 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。

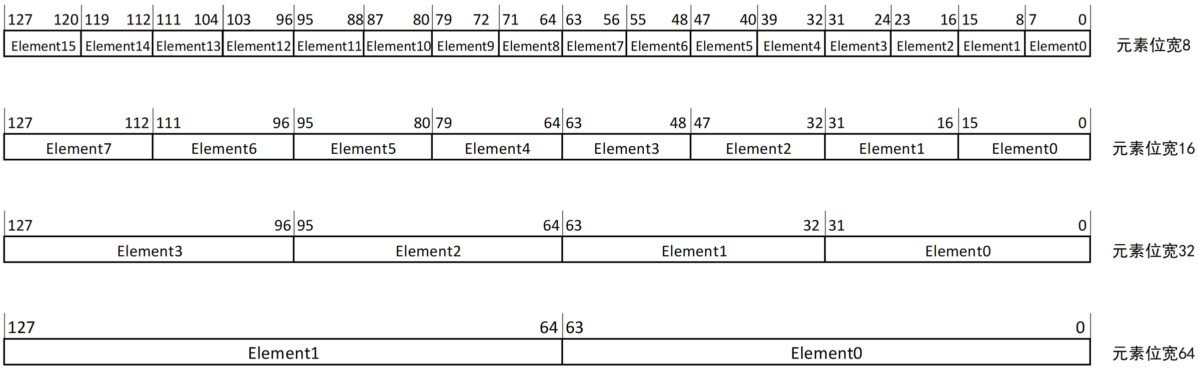


图 4.4: 寄存器中的矢量数据组织结构

4.8　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。

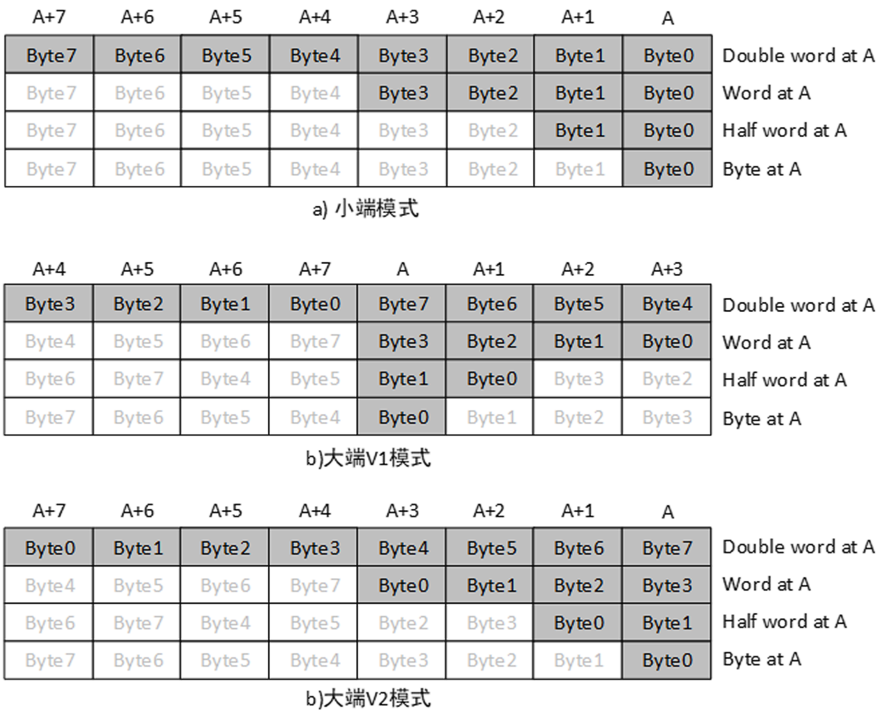


图 4.5: 内存中的数据组织形式

C920V3 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。

C920V3 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）
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

C920V3 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。

5.2.3 非精确异常

在极少数情况下，处理器可能表现出“非精确异常”的行为。非精确异常，是指发生异常时，mepc 没有指向触发该异常的指令。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，当总线返回 Error 时，load 指令已经退休，所以 mepc 指向的是后续的某条指令，而不是 load 指令本身。

值得注意的是，非精确异常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。

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。

6 内存模型

6.1 内存模型概述

6.1.1 内存属性

C920V3 支持两种内存类型，分别是内存（Memory）和外设（Device），由 SO 位区分。其中，Memory 类型的内存需要配置成 WeakOrder 属性（SO 比特位配置为 0），特点是支持投机执行和乱序执行，根据是否可高缓（Cacheable，C）进一步分为可高缓内存（Cacheable memory）和不可高缓内存（Non-cacheable memory）。

注意

SO 区域不能存放指令和页表。

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
可高缓内存	0	1
不可高缓内存	0	0
外设	1	0

地址的页面属性由 sysmap.h 决定。sysmap.h 是 C920V3 扩展的配置文件，对用户开放，用户可以根据自身需求，定义不同地址段的页面属性。

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 $SYSMAP\_BASE\_ADDR_i$ （i=0~7）定义，地址下限（包含）由 $SYSMAP\_BASE\_ADDR_{i-1}$ 定义，即：

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

第 0 个地址空间的下限是 0x0。内存地址不在 sysmap.h 文件设定的 8 个地址区间的地址类型默认为 Device（SO = 1）。每个地址空间上下边界是 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，属性的排布如 图 6.1 所示：

1	0
Strong order	Cacheable

图 6.1: sysmap.h 地址属性格式

6.1.2　内存一致性模型

C920V3 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，C920V3 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

6.1.3　SYSMAP 配值参考

- 第 0 段地址空间属性定义：40’h0 <= addr0[39:0] <40’h100_0000, flg0 = 2’b01。该段地址空间包括 SRAM，配置成 cacheable 属性。属性定义如下：

```
`define CT_SYSMAP_BASE_ADDR0    28'h1000

`define CT_SYSMAP_FLG0          2'b01
```

- 第 1 段地址空间属性定义：40’h100_0000 <= addr1[39:0] < 40’h200_0000, flg1 = 2’b10。该段地址空间主要对应特殊函数写地址（如 print 函数）配置成 strong_order 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 CT_SYSMAP_BASE_ADDR1    28'h2000
```

(续下页)

(接上页)

```
`define CT_SYSMAP_FLG1          2'b10
```

- 第 2 段地址空间属性定义：40’h200_0000 <= addr2[39:0] < 40’h8000_0000, flg2 = 2’b10。该段地址空间主要对应 APB 接口，配置成 strong ordered 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 CT_SYSMAP_BASE_ADDR2    28'h8_0000

`define CT_SYSMAP_FLG2          2'b10
```

- 第 3 段地址空间属性定义：40’h8000_0000 <= addr3[39:0] < 40’hb000_0000, flg3 = 2’b01。属性定义如下:

```
`define CT_SYSMAP_BASE_ADDR3    28'hb_0000

`define CT_SYSMAP_FLG3          2'b01
```

- 第 4 段地址空间属性定义：40’hb000_0000 <= addr4[39:0] < 40’hffff_f000, flg4 = 2’b10。属性定义如下:

```
`define CT_SYSMAP_BASE_ADDR4    28'hf_ffff

`define CT_SYSMAP_FLG4          2'b10
```

- 第 5 段地址空间属性定义：40’hffff_f000 <= addr5[39:0] < 40’h40_0000_0000, flg5 = 2’b01。属性定义如下:

```
`define CT_SYSMAP_BASE_ADDR5    28'h400_0000

`define CT_SYSMAP_FLG5          2'b01
```

- 第 6 段地址空间属性定义：40’h 40_0000_0000 <= addr6[39:0] <40’h50_0000_0000, flg6 = 2’b10。属性定义如下:

```
`define CT_SYSMAP_BASE_ADDR6    28'h500_0000

`define CT_SYSMAP_FLG6          2'b10
```

- 第 7 段地址空间属性定义：40’h50_0000_0000, <= addr7[39:0] < 40’hff_ffff_f000, flg7 = 2’b01。属性定义如下:

```
`define CT_SYSMAP_BASE_ADDR7    28'hfff_ffff

`define CT_SYSMAP_FLG7          2'b01
```

6.2　虚拟内存管理

6.2.1　MMU 概述

C920V3 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（40/48 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。

C920V3 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。

C920V3 支持 U Mode 配置为 32 位或者 64 位。当 UXL 配置为 32 位时：

- MMU 按照 32 位虚拟地址进行 SV39/SV48 的地址转换；
- 虚拟地址的高 32 位不允许出现非 0 值，否则产生 page fault 异常。

6.2.2　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。

C920V3 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 16 个全相联表项，可以混合存储 4K、2M 和 1G 三种大小的页面，取指请求命中 I-uTLB 时，当拍可以得到物理地址和相应权限属性。

D-uTLB 有 16 个全相联表项，可以混合存储 4K、2M 和 1G 三种大小的页面，加载和存储请求命中 D-uTLB 时，当拍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，可以混合存储 4K、2M 和 1G 三种大小页面。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。

6.2.3　地址转换流程

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。

在 Sv48 硬件配置下，C920V3 采用最多四级页表索引的方式实现地址转换。在 Sv39 硬件配置下，C920V3 采用最多三级页表索引的方式实现地址转换。此处以三级页表的地址转换为例，介绍 C920V3 地址转换流程：

1. 访问第一级页表得到第二级页表的基地址和相应的权限属性；
2. 访问第二级页表得到第三级页表的基地址和相应的权限属性；
3. 访问第三级页表得到最终物理地址和相应的权限属性。

每一级访问都有可能得到最终的物理地址，即叶子表项。虚拟页面号 VPN 有 27-bit，等分为三个 9-bit 的 VPN[i]，每次访问使用一部分 VPN 进行索引。叶子表项的内容（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。若 uTLB 失配则访问 jTLB，若 jTLB 进一步失配，则 MMU 会启动 Hardware Page Table Walk，访问内存得到最终的地址转换结果。

C920V3 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，其结构如 图 6.2 所示。

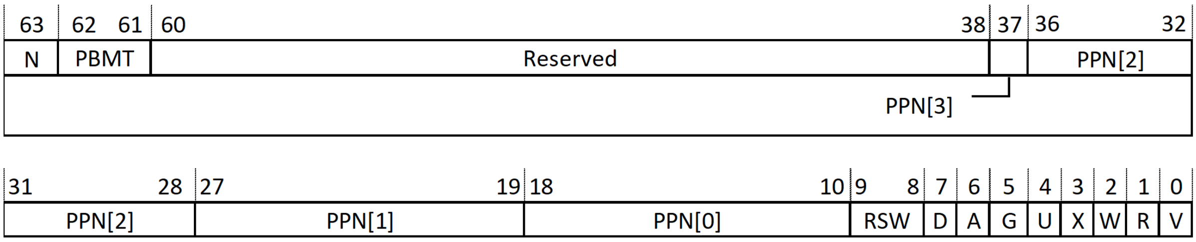


图 6.2: 页表结构说明（PA40）

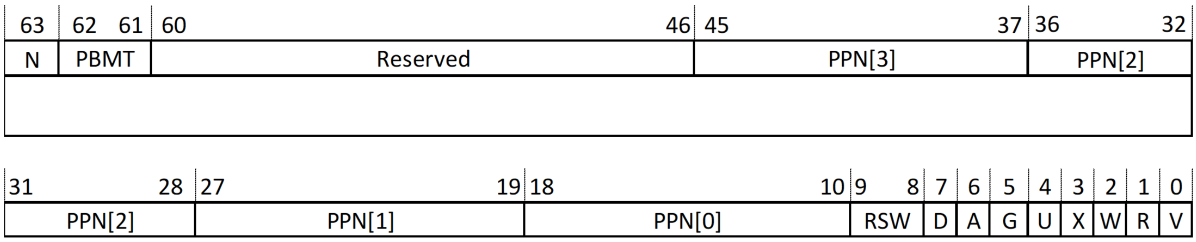


图 6.3: 页表结构说明（PA48）

C920V3 页表各 bit 属性：

N (NAPOT, Naturally Aligned Power of Two):

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 表 6.3 定义。

表 6.3: 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	-

续下页

表 6.3 - 接上页

i	pte.ppn[i]	Description	pte.napot bits
>=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。详细说明请查看 *RISC-V SVNAPOT 标准扩展说明*。

PBMT(Page-Based Memory Types):

PBMT 位表示虚拟地址的 PMA 属性，具体定义按场景划分，如 表 6.4 所示。（注：此功能需要开启 menvcfg 寄存器的 PBMTE 位）

表 6.4: PBMT 的定义

Mode	Value	Requested Memory Attributes
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

说明：

- 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。

详细说明请查看 *RISC-V SVPBMT 标准扩展说明*。

RSW (Reserved for use by Supervisor SoftWare): 给特权态软件保留的页表属性位，硬件不做处理。

D (Dirty bit): 表示页面可写。

A (Access bit): 表示页面可访问。

G (Global bit): 表示页面作用于全局。

U (User bit): 表示页面为 User 态专用。

X (Executable bit): 表示页面可执行。

W (Writeable bit): 表示页面可写。

R (Readable bit): 表示页面可读。

V (Valid bit): 表示页面有效。

C920V3 页表 XWR bit 组合的含义规定如 表 6.5 所示。

表 6.5: C920V3 页表 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

SV39/48 标准规定的地址转换的详细流程描述如下：

CPU 要访问某个虚拟地址，若 TLB 命中，则从 TLB 中直接获取物理地址及相关属性。若 TLB 缺失，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]，则地址的转换具体步骤为：

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 DCache/内存，得到 64-bit 一级页表 PTE;
2. 当 PTE V 位有效时，根据 表 6.5 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. 找到了叶子页表，结合 PMP 中的 X/W/R/L 位和 PTE 中的 X/W/R 位得到两者的最小权限进行权限检查，并将 PTE 的内容回填到 jTLB 中；
4. 在任何一步的 PMP 检查中，如果有权限违反，则根据访问类型产生对应的 access error 异常；
5. 若得到叶子页表，但访问类型违反 A/D/X/W/R/U-bit 的设置，产生对应的 page fault 异常；若访问 Dcache/内存过程中得到 access error 响应，产生 access fault 异常。
6. 若得到叶子页表，但访问次数少于 3 次，则说明得到了大页表。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；
7. 若使用 VPN[min] 访问结束仍未得到叶子页表，则产生对应的 page fault 异常。

6.2.4 系统控制寄存器

6.2.4.1 MMU 地址转换寄存器 (SATP)

SATP 是 SV39 规范的 MMU 控制寄存器。

63 60	59 44	43 32
Mode	ASID	-

31 28	27 0
-	PPN

图 6.4: SATP 寄存器说明

Mode - MMU 地址翻译模式

表 6.6: MMU 地址翻译模式

RV64		
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
12-15	-	Reserved

ASID –当前 ASID

表示当前程序的 ASID 号。

PPN –硬件回填根 PPN

第一级硬件回填使用的 PPN。

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*调试/追踪寄存器组/Trace 寄存器组（与调试模式共享）*中有关 MCER/SCER 的描述。

1 bit 以上的错误无法检测或纠正。

C920V3 MMU 支持软件注入错误功能，具体控制寄存器说明可以参考*调试/追踪寄存器组/Trace 寄存器组（与调试模式共享）*中有关 MEICR 的描述。

6.4 物理内存保护

6.4.1 PMP 概述

C920V3 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。

C920V3 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）

物理内存保护设置寄存器提供 8 个表项的权限设置。

63 56	55 48	47 40	39 32	31 24	23 16	15 8	7 0	
entry7_cfg	entry6_cfg	entry5_cfg	entry4_cfg	entry3_cfg	entry2_cfg	entry1_cfg	entry0_cfg	**pmpcfg0**
8	8	8	8	8	8	8	8	

63 56	55 48	47 40	39 32	31 24	23 16	15 8	7 0	
entry15_cfg	entry14_cfg	entry13_cfg	entry12_cfg	entry11_cfg	entry10_cfg	entry9_cfg	entry8_cfg	**pmpcfg2**
8	8	8	8	8	8	8	8	

图 6.5: 物理内存保护设置寄存器整体分布

7	6 5	4 3	2	1	0
L(WARL)	0(WARL)	A(WARL)	X(WARL)	W(WARL)	R(WARL)
1	2	0	1	1	1

图 6.6: 物理内存保护设置寄存器

PMP 控制寄存器描述其具体的描述如 表 6.7 所示。

表 6.7: 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 6.8 所示。

表 6.8: 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

续下页

表 6.8 - 接上页

pmpaddr[45:9]	pmpcfg.A	保护区大小	备注
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
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，C920V3 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 C920V3 PMP 表项粒度最低支持 4 KB，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 6.7: PMP 地址寄存器

6.5 内存访问顺序

在不同的场景下，C920V3 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。

7 内存子系统

7.1　内存子系统概述

C920V3 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

L1 指令高速缓存的主要特征如下：

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

C920V3 指令高速缓存采用两路组相联结构，为了减少并行访问两路缓存的功耗，C920V3 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。

7.2.3 循环加速缓存器

针对程序中存在的大量短循环，C920V3 设置了一个 32B 的循环加速缓存器。当检测到短循环指令序列时，该循环体被加载到循环加速缓存器中。当后续取指命中该缓存器时，直接从缓存器中获取指令以及跳转的目标地址，关闭对指令高速缓存和分支历史表、分支跳转目标预测器的访问，从而降低了取指的动态功耗。

用户可以通过配置隐式操作寄存器 MHINT.LPE 使能短循环加速功能。

7.2.4 分支历史表

C920V3 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 BI-MODE 预测器作为预测机制，每周期支持一条分支结果预测。

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

分支历史表进行预测的条件分支指令包括：

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

7.2.5 分支跳转目标预测器

C920V3 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

C920V3 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，

C920V3 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，C920V3 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

7.3　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 32KB/64KB；
- 2 路组相联，缓存行大小为 64B；
- 物理地址索引，物理地址标记 (PIPT)；
- 每次读访问的最大宽度为 128 比特，支持字节/半字/字/双字/四字访问；
- 每次写访问的最大宽度为 256 比特，支持任意字节组合的访问；
- 写策略支持写回-写分配模式和写回-写不分配模式；
- 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　L1 D-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。

C920V3 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

C920V3 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。C920V3 为每个核心上设置了一个位于 L1 数据高缓的局部监测器和一个位于二级高缓的全局监测器。每个监测器由一个状态机和一个地址缓存器组成，其中，状态机包含两个状态：IDLE 和 EXCLUSIVE。

对于属性设置为可高缓的页面，通过局部监测器就能够实现独占式访问。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 并且地址和 Size 完全匹配，那么执行该写操作，返回写成功，并清除状态机回到 IDLE 态；否则如果状态或者地址/Size 有一项不满足条件，或者数据高速缓存未使能时，不执行该写操作，返回写失败，并清除状态机回到 IDLE 态。其他核的写操作在相同 cacheline 地址匹配局部监测器时，也会将状态机清回到 IDLE 态；本核的写操作或不同地址的独占访问不影响局部监测器。此外，在进程切换时需要清除局部监测器。

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

在基于 C920V3 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 256KB/512KB/1MB/2MB/4MB/8MB；
- 16 路组相联，缓存行大小为 64B；
- L2 Cache 与 L1 D-Cache 是严格的 Inclusive 关系。L2 cache 与 L1 I-Cache 是非严格的 Inclusive 关系；
- 物理地址索引，物理地址标记（PIPT）；
- 每次访问的最大宽度为 64B；
- 支持写回-写分配和写回-写不分配的写策略；
- 采用先进先出的替换策略；
- 可编程的 RAM 访问延时；
- 可选的 ECC 校验机制；
- 支持指令预取和 TLB 预取机制；
- 采用分块的流水线技术。

7.4.2 L2 D-Cache 一致性

C920V3 二级高速缓存采用 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)

7.4.3 组织形式

C920V3 L2 高速缓存在组织形式上采用了分块的流水线架构，将访问地址离散在两个不同的块中，允许多个访问的并行处理，从而提高访问效率。

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。

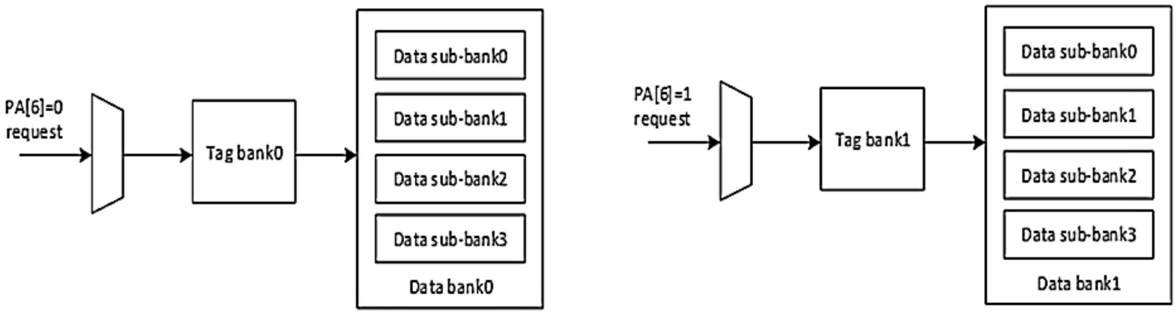


图 7.1: L2 Cache 组织形式

7.4.4 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。C920V3 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 所示。

表 7.2: RAM 访问延迟配置

配置选项	功能	说明
L2 TAG setup	L2 Cache Tag RAM setup： **1b0** 0 cycle. **1b1** 1 cycle.	L2 Cache Tag RAM 的配置只影响 TAG RAM 的访问
L2 TAG latency	L2 Cache Tag RAM latency： **3b000** 1 cycle. **3b001** 2 cycle. **3b010** 3 cycle. **3b011** 4 cycle. **3b1xx** 5 cycle.	

续下页

表 7.2 - 接上页

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

用户应该根据所用 RAM 的访问时长，配置 setup/latency 参数。

setup/latency 默认值与硬件配置值相同。当 latency 硬件配置值 >= 2 cycle 时，RAM clock 采用 2 分频时钟。

在配置以上选项后，所得的访问周期数如 表 7.3 所示。

表 7.3: TAG RAM 有效访问延迟

TAG Latency	TAG setup = 0	TAG setup = 1
000	1	2
001	2	3
010	3	4
011	4	5
1xx	5	5

表 7.4: DATA RAM 有效访问延迟

DATA Latency	DATA setup = 0	DATA setup = 1
000	1	2
001	2	3
010	3	4
011	4	5
100	5	6
101	6	7
110	7	8

续下页

表 7.4 - 接上页

DATA Latency	DATA setup = 0	DATA setup = 1
111	8	9

备注

- L2 Tag latency 最大有效延迟为 5 周期；
- TAG setup 设置为 1 时增加 1 周期访问，访问 SRAM 之前会对 SRAM 输入端信号进行 flop；
- L2 Data latency 最大有效延迟为 9 周期；
- DATA setup 设置为 1 时增加 1 周期访问，访问 SRAM 之前会对 SRAM 输入端信号进行 flop。

7.5 内存加速访问

本小节集中描述 C920V3 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，C920V3 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。

C920V3 最多支持 8 个通道的数据预取，并实现了连续预取和间隔预取（stride<=32 个缓存行）这 2 种不同的预取方式。

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

在处理器执行数据高速缓存无效和清除操作时，停止数据预取功能。

用户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

C920V3 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（0，1，2，3），所有预取都会回填 L2 高速缓存；
- TLB 预取数固定为 1；
- 预取机制以 4KB 页表为边界，对于预取时发生跨 4KB 边界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 在缺失时仍会发起回填操作。

7.6.1 L1 高速缓存扩展寄存器

C920V3 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

C920V3 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

C920V3 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 7.5 所示。

表 7.5: 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

续下页

表 7.5 - 接上页

指令名称	指令描述
DCACHE.IPA	DCACHE 按物理地址无效
DCACHE.ISW	DCACHE 按 set/way 无效
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 算术运算指令术语*

7.7 L1/L2 Cache 的保护机制

C920V3 实现的 Cache 保护机制包括：L1 I-Cache 奇偶校验、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 7.6 所示。

表 7.6: 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	无法检测 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 28 bit；检查 data array，校验粒度为 32 bit。

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* / *超级用户模式处理器控制和状态扩展寄存器组*中有关*机器模式 L1 Cache ECC 寄存器（MCER）* / *超级用户模式 L1 Cache ECC 寄存器（SCER）*的描述。1 bit 以上错误无法检测或纠正。

C920V3 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 Cache 访问扩展寄存器组*中有关*机器模式 L1 Cache 硬件错误注入寄存器（MEICR）*的描述。

此外，还实现了对 jTLB 的奇偶校验，能够检测 1 bit 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 7.7 所示。

表 7.7: 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 1 bit ECC 错误时，能够自动纠正错误，返回正确数据；当发生 2 bit 错误时，能够检测出错误，发起校验错误中断，并需要软件主动根据寄存器信息无效掉 L1 D-cache 中出错的缓存行。2 bit 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 2 bit 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*机器模式 L1 Cache ECC 寄存器（MCER）*/ *超级用户模式 L1 Cache ECC 寄存器（SCER）* 的描述。

L1 数据缓存发生 2 bit 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*机器模式异常处理寄存器组*中有关*机器模式中断等待状态寄存器（MIP）* 的描述。

C920V3 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 Cache 访问扩展寄存器组*中有关*机器模式 L1 Cache 硬件错误注入寄存器（MEICR）* 的描述。

7.7.3 L2 ECC 校验

L2 高速缓存支持可配置的 ECC 校验，可以对 Tag RAM 和 Data RAM 进行校验。校验粒度如 表 7.8 所示。

表 7.8: L2 ECC 校验粒度

RAM	校验粒度	校验比特
TAG	23/24/25/26/27	7（ECC）
Dirty	8 bit（fifo 除外）	5（ECC）
DATA	64	8（ECC）

L2 CACHE ECC 由机器模式 L2CAHCE 控制寄存器（MCCR2）中的 bit 1 使能。开启后，L2 高速缓存在写入时对数据进行 ECC 编码，在读取时进行校验。当检测到 1 bit ECC 错误时，能够纠正错误，返回正确的数据；当发生 2 bit 错误时，能够检测出数据错误，上报 ECC 校验错误中断，返回错误的数据，并无效掉 L2 Cache 中出错的缓存行。2 bit 以上错误不能准确检测或纠正。

软件可以查询 MCER2/SCER2 寄存器获取错误的相关信息，例如是否产生 2 bit 错误以及错误在 L2 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 L2 Cache ECC 寄存器（MCER2）的方式清除。具体控制寄存器说明可以参考*机器模式 Cache 访问扩展寄存器组*中有关*机器模式 L2 Cache ECC 控制寄存器（MCER2）*/ *超级用户模式 L2 Cache ECC 寄存器（SCER2）* 的描述。

L2 ECC 中断作为一个中断源输入 PLIC，中断 ID 固定为 1 号（PLIC 内 ID）。CPU 可通过响应外部中断后，查询中断响应/完成寄存器（PLIC_CLAIM）获得该 ID。中断的配置、维护及触发过程描述可以参考*中断控制器*。

C920V3 L2 内存子系统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 Cache 访问扩展寄存器组*中有关*机器模式 L2 Cache 硬件错误注入寄存器（MEICR2）*的描述。

8 矢量计算

C920V3 可选配支持 RVV 官方矢量扩展，该矢量系列指令主要应用于多媒体计算领域，为多媒体计算领域提供额外的算力支撑。

8.1 版本支持

C920V3 兼容 RISC-V “V” Vector Extension, Version 1.0-rc1-20210608 。

8.2 矢量编程模型

C920V3 的矢量扩展支持以下特性：

- C920V3 拥有 32 个独立的矢量架构寄存器 v0-v31。矢量寄存器的位宽为 128 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 BF16、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 属性值不能为 1。

8.3 矢量控制寄存器

C920V3 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 C920V3 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。

- vlenb

  矢量位宽寄存器，以字节为单位表示 C920V3 的矢量位宽。

此外，C920V3 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时，是否需要保存矢量相关的寄存器。

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

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。

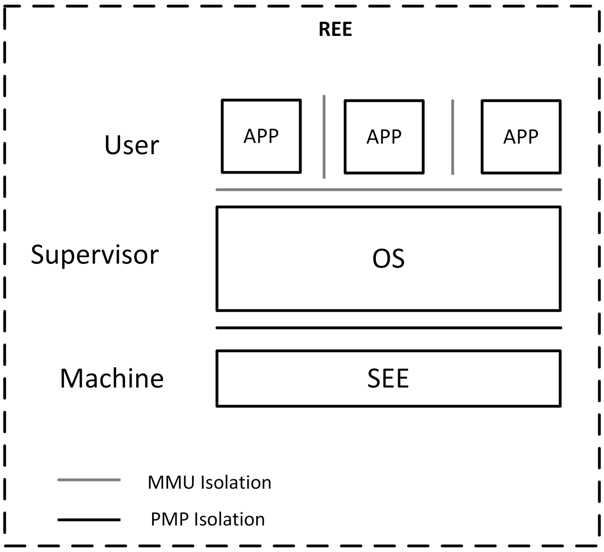


图 9.1: RISC-V 特权模型

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。

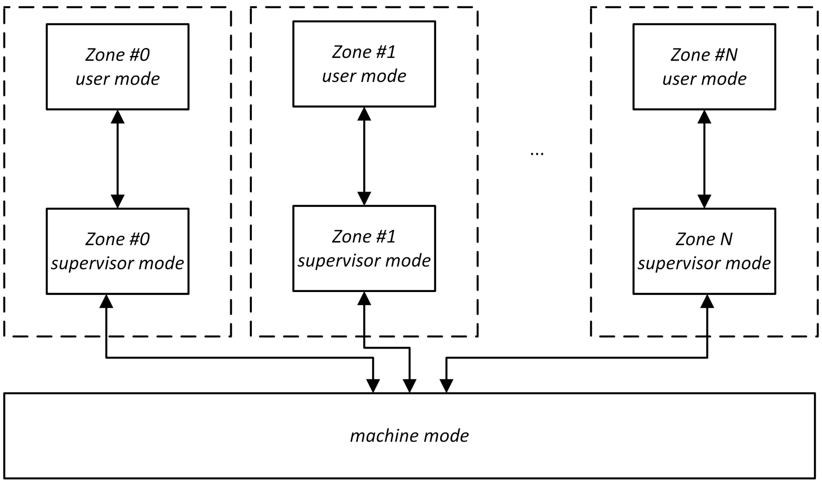


图 9.2: 玄铁 RISC-V 处理器的 Zones 和特权模式

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。

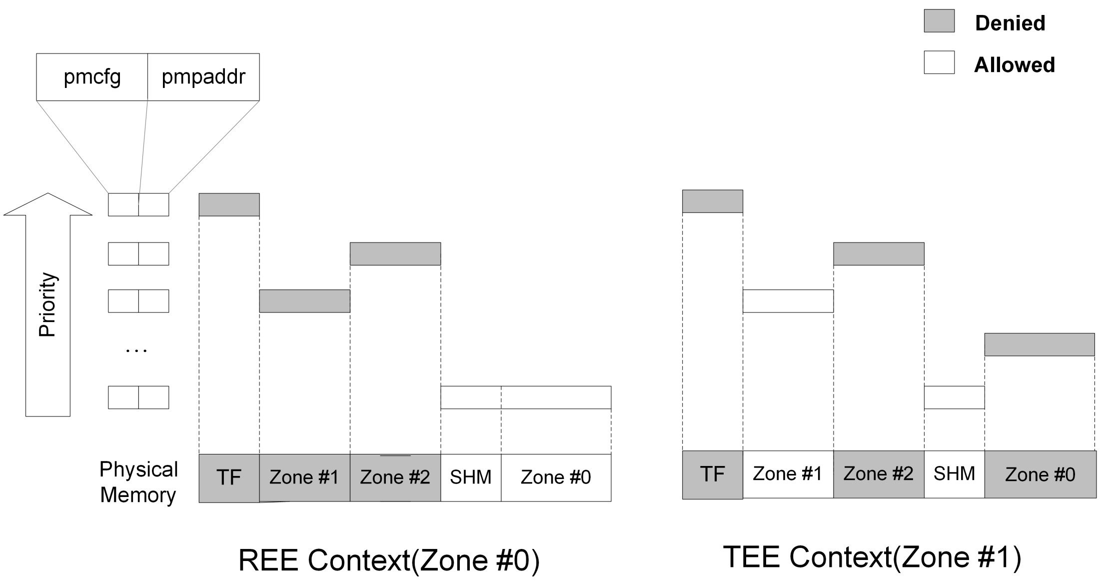


图 9.3: 不同 Zone 的 PMP 配置

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:

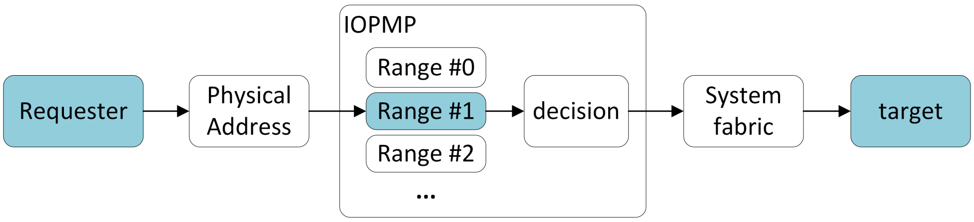


图 9.4: 请求端连接 IOPMP

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:

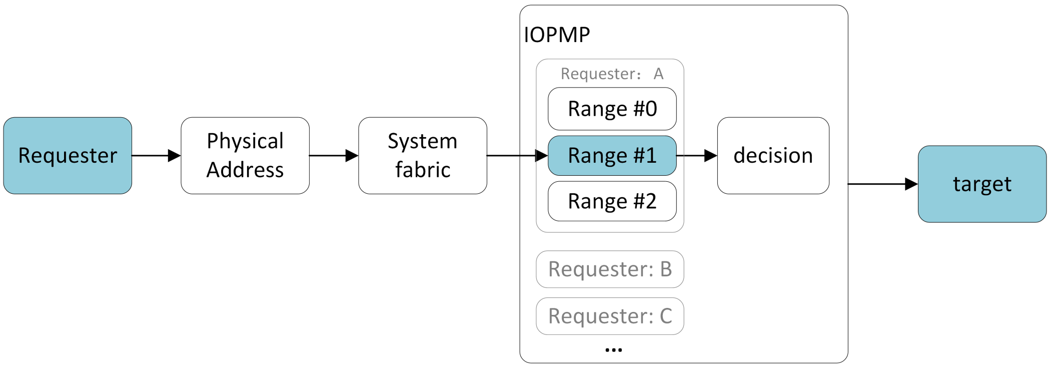


图 9.5: 目标端连接 IOPMP

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。

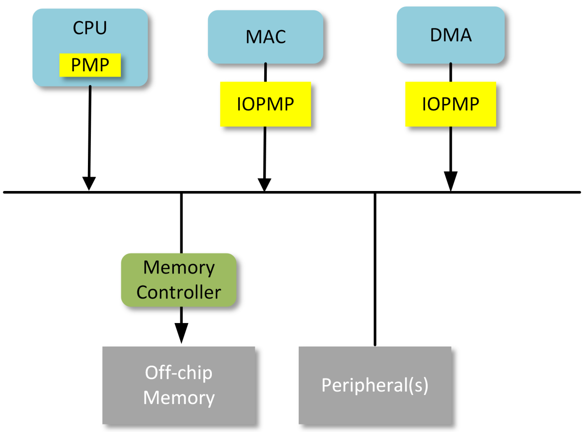


图 9.6: 基于 PMP 和 IOPMP 隔离的 SoC 架构

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 C920V3 提供了 DCP 端口。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 C920V3 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。

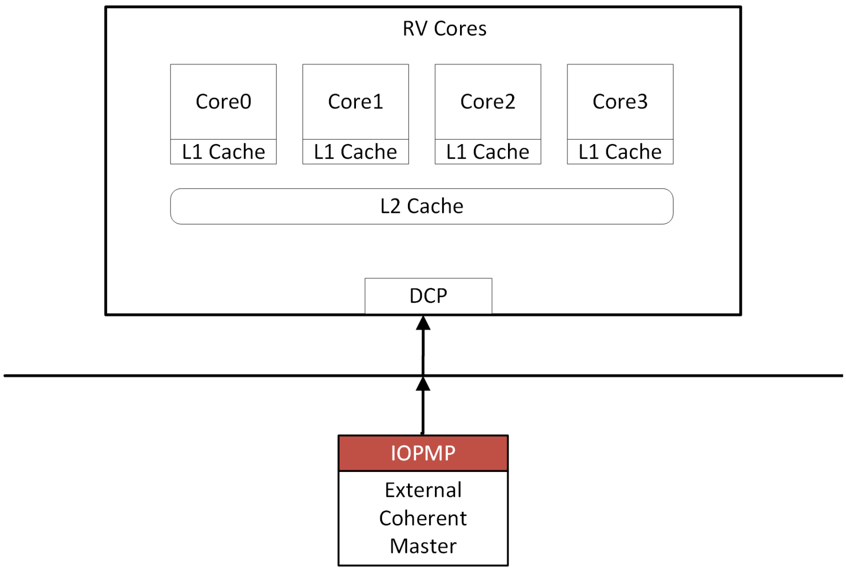


图 9.7: DCP 端口保护

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

如 图 9.8，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。

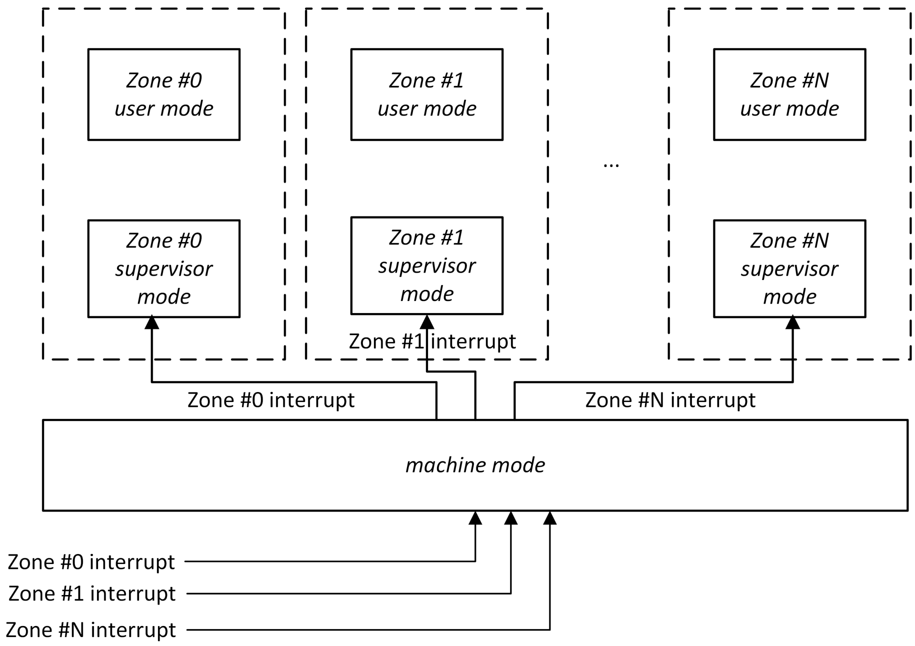


图 9.8: 玄铁 RISC-V 处理器 M 模式中断分发

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

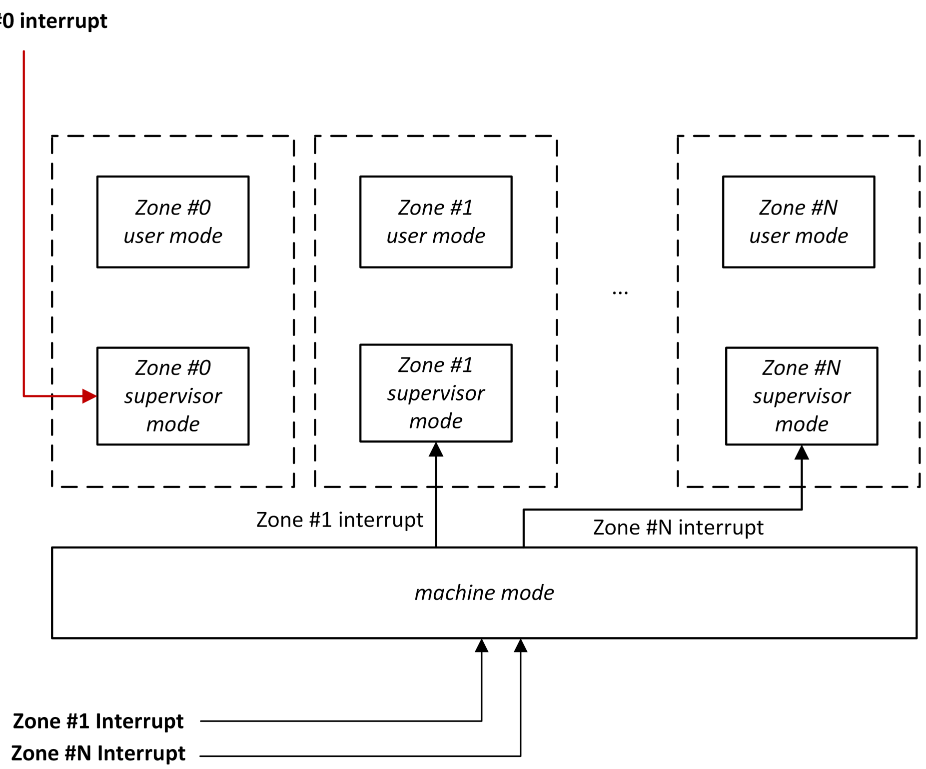


图 9.9: 处理器运行在 Zone #0 时的中断处理规则

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

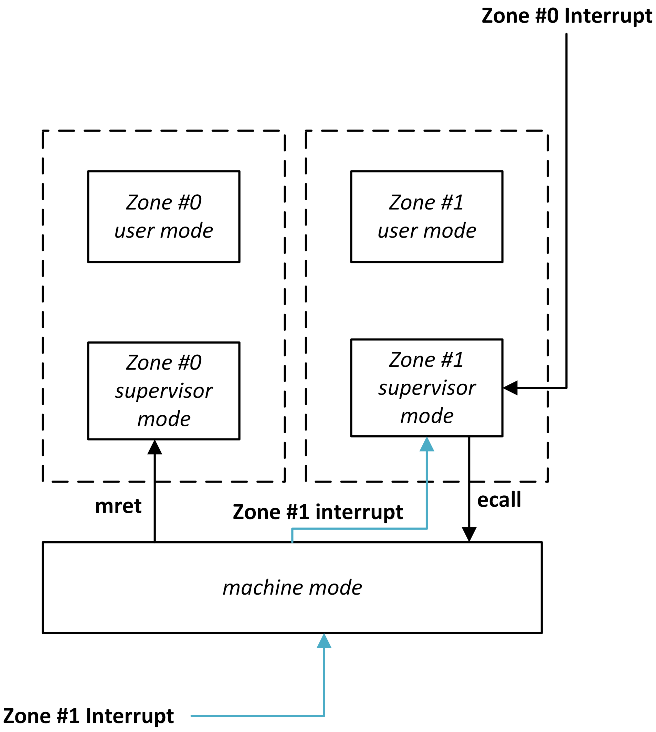


图 9.10: 处理器运行在 Zone #1 时的中断处理规则

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

C920V3 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 和 图 10.2 所示。

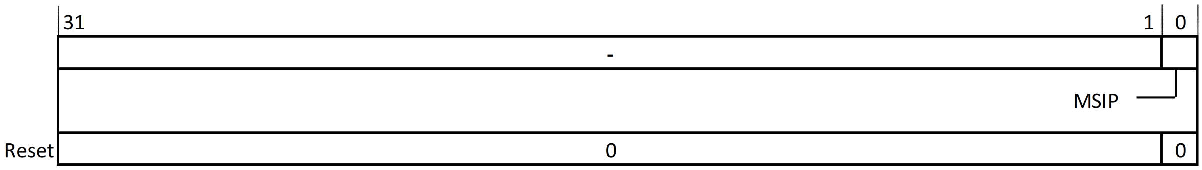


图 10.1: 机器模式软件中断配置寄存器（MSIP）

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。

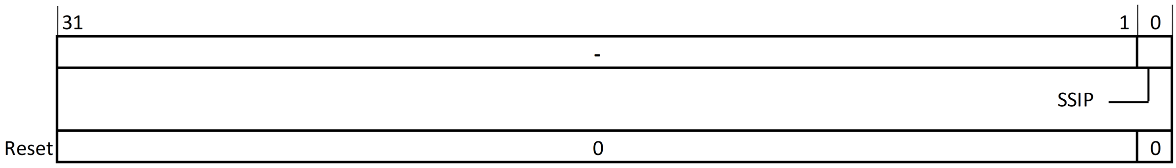


图 10.2: 超级用户模式软件中断配置寄存器（SSIP）

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；

	63	32	31	0
	MTIMEH		MTIMEL	
Reset	0x0		0x0	

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；

	63	32	31	0
	STIMEH		STIMEL	
Reset	0x0		0x0	

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。C920V3 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。每组寄存器结构相同，其寄存器位分布和位定义如 图 10.5 和 图 10.6 所示。

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。

	63	32	31	0
	MTIMECMPH		MTIMECMPL	
Reset	0xffffffff		0xffffffff	

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

- 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

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

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。

C920V3 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。C920V3 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 表 10.2 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
	0x0001004	PLIC_IP1	R/W	0x0	32~63 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
	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
	…	…	…	…	…
	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

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
	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	-	-	-	-
	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

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
	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 128MB, 这段空间的基地址由 pad_cpu_apb_base（输入端口，请参照 C920V3 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。

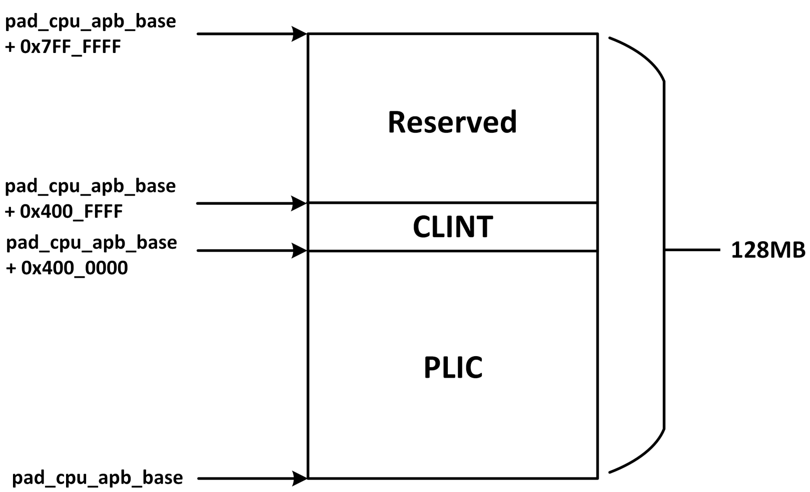


图 10.7: PLIC&CLINT 地址空间

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。

	31 … 5	4 … 0
	-	PRIO
Reset	0	0

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。

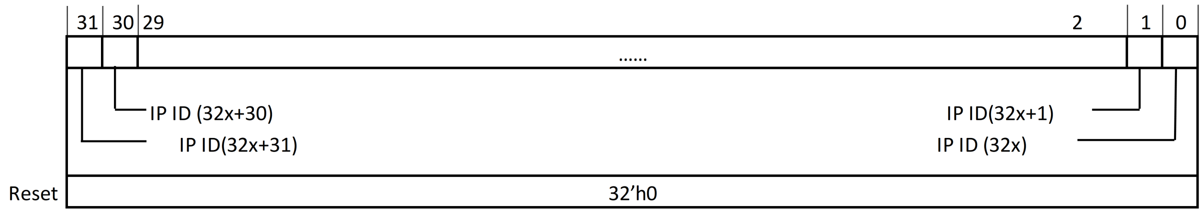


图 10.9: PLIC_IP x 中断等待寄存器 (PLIC_IP)

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。

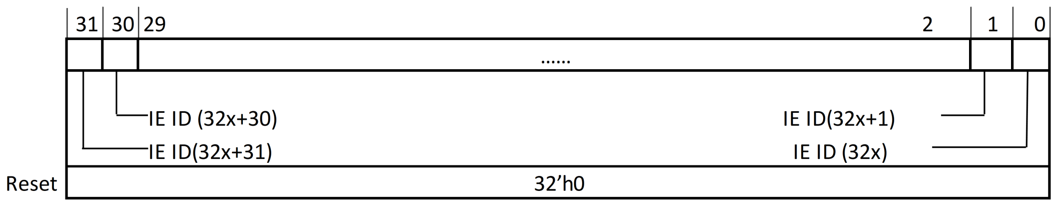


图 10.10: PLIC_IE x 中断使能寄存器 (PLIC_IE)

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。

	31 … 1	0
	-	S_PER
Reset	0	0

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。

	31 … 5	4 … 0
	-	PRIOTHRESHOLD
Reset	0	0

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。

	31 ... 10	9 ... 0
	-	CLAIM_ID
Reset	0	0

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，并且此时 CPU 配置为 4 核：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

C920V3 的主设备接口支持 AMBA 4.0 ACE 或 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

主设备接口负责 C920V3 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 AMBA 4.0 ACE/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 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 C920V3 主设备接口的 Outstanding 能力，具体如下：

表 11.1: 主设备接口 Outstanding 能力

参数	数值	说明
Read Issuing Capability	7n+42 n= 核心数量	每个核心最多发出 7 个 Non-cacheable 和 Device 读请求。 全局最多 42 个 Cacheable 读请求，其中 28 个来自于读、写数据/TLB/ 取指回填 L1 及 L2 预取，14 个来自于写数据回填 L2。
Write Issuing Capability	12n+32 n= 核心数量	每个核心最多发出 12 个 Non-cacheable 和 Device 写请求，其中 Device 写请求最多 8 个。 全局最多 32 个 Cacheable 写请求，32 个 Device 或 Non-cacheable 写请求。

表 11.2: 主设备接口 ARID 编码

ARID[7:0]	适用场景	每个 ID 的 Outstanding
{2'b10, 6'b??????}	Cacheable 读请求	每个 ID 无 outstanding，所有 cacheable 读请求 outstanding 总共 42 个。
{1'b0, 2'b(coreid), 5'b00???}	Non-cacheable weak-ordered 读请求	每个 ID 无 outstanding，每个核心不同 ID Non-cacheable weak-ordered 读请求 outstanding 数量为 7 个，总共 28 个。
{1'b0, 2'b(coreid), 5'h1e}	Non-cacheable weak-ordered 抢锁请求	每个 ID 无 outstanding
{1'b0, 2'b(coreid), 5'h1d}	Non-cacheable strong-ordered 读请求	每个 ID 的 outstanding 数量为 7 个

表 11.3: 主设备接口 AWID 编码

AWID[7:0]	适用场景	每个 ID 的 Outstanding
{3'b111, 5'b?????}	Cacheable 写请求	每个 ID 无 outstandin g，所有 cacheable 写请求 outstanding 总共 32 个。
{2'b00, 1'b?, 1'b0, 4'h????}	Non-cacheable weak-ordered 写请求	每个 ID 无 outstanding，每个核心不同 ID outstanding 数量最多为 12 个，全局最多 32 个。 所有 Non-cacheable weak-ordered 写请求 outstanding 总共 32 个。
{1'b0, 2'b(coreid), 5'h1e}	Non-cacheable weak-ordered 抢锁写请求	每个 ID 无 outstanding
{1'b0, 2'b(coreid), 5'h1d}	Non-cacheable strong-ordered 写请求	每个 ID 的 outstanding 数量为 8，全局最多 32 个。 Non-cacheable strong-ordered 写请求 outstanding 为 32 个。

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/ACE 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 和 WRAP 传输，其它突发类型均不支持；
- LEN 支持传输长度为 1 或者 4，其他传输长度均不支持；

- 支持独占式访问；
- SIZE 支持四字、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：C920V3 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/ACE 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，写传输忽略此错误；

11.1.6　主设备接口信号

主设备的接口信号为 表 11.5 中所有信号。

表 11.5: AXI/ACE 总线主接口信号

信号名	I/O	初始值	时钟域	功能描述
AXI/ACE 总线主接口信号				

续下页

表 11.5 - 接上页

信号名	I/O	初始值	时钟域	功能描述
biu_pad_araddr[39:0]/[43:0]/[47:0]	O	40'b0 44'b0 48'b0	SYS	读地址总线： 40/44/48 位地址总线。根据配置，地址位宽不同，且三个信号不会同时存在。 40 位地址总线： 物理地址位宽配置为 40 bit 时，主接口配置为 AXI 或 MMU 配置为 SV39； 44 位地址总线： 物理地址位宽配置为 40 bit 时，主接口配置为 ACE 且 MMU 配置为 SV48； 48 位地址总线： 物理地址位宽配置为 48 bit。
biu_pad_arburst[1:0]	O	2'b0	SYS	突发传输指示信号： 指示传输是一次突发传输的一部分： 01：INCR； 10：WRAP4。
biu_pad_arcache[3:0]	O	4'b0	SYS	读请求对应的 cache 属性： [3]:Other Allocate； [2]:Allocate； [1]:Modifiable； [0]:Bufferable。
biu_pad_arid[7:0]	O	8'b0	SYS	读地址 ID。
biu_pad_arlen[7:0]	O	8'b0	SYS	突发传输长度： 00000000：1 拍； 00000011：4 拍。
biu_pad_arlock	O	1'b0	SYS	读请求对应的访问方式： 0：normal access； 1：exclusive access。
biu_pad_arprot[2:0]	O	3'b0	SYS	读请求的保护类型： 0 | 1 [2]: Data | Instruction； [1]: Secure | Non-secure； [0]: User | Privileged。

续下页

表 11.5 - 接上页

信号名	I/O	初始值	时钟域	功能描述
biu_pad_arsize[2:0]	O	3'b0	SYS	读请求每拍数据位宽： 000: 8bits; 001:16bits; 010:32bits; 011:64bits; 100: 128bits。
biu_pad_arvalid	O	1'b0	SYS	读地址有效信号。
pad_biu_arready	I	-	SYS	读地址通道 ready 信号。
pad_biu_rdata[127:0]	I	-	SYS	读数据总线： 128 位数据总线。
pad_biu_rid[7:0]	I	-	SYS	读数据 ID。
pad_biu_rresp[3:0]	I	-	SYS	读响应信号： [1:0]： 00：OKAY； 01：EXOKAY； 10：SLVERR； 11：DECERR。
pad_biu_rlast	I	-	SYS	读数据最后一拍指示信号。
pad_biu_rvalid	I	-	SYS	读数据有效信号。
biu_pad_rready	O	1'b1	SYS	读数据通道 ready 信号。
biu_pad_awaddr[39:0]/[43:0]/[47:0]	O	40'b0 44'b0 48'b0	SYS	写地址总线：40/44/48 位地址总线。 40 位地址总线： 物理地址位宽配置为 40 bit 时，主接口配置为 AXI 或 MMU 配置为 SV39； 44 位地址总线： 物理地址位宽配置为 40 bit 时，主接口配置为 ACE 且 MMU 配置为 SV48； 48 位地址总线： 物理地址位宽配置为 48 bit。
biu_pad_awburst[1:0]	O	2'b0	SYS	突发传输指示信号： 指示传输是一次突发传输的一部分： 01：INCR；

续下页

表 11.5 - 接上页

信号名	I/O	初始值	时钟域	功能描述
biu_pad_awcache[3:0]	O	4'b0	SYS	写请求对应的 cache 属性： [3]:Other Allocate； [2]:Allocate； [1]:Modifiable； [0]:Bufferable。
biu_pad_awid[7:0]	O	8'b0	SYS	写地址 ID。
biu_pad_awlen[7:0]	O	8'b0	SYS	突发传输长度： 00000000：1 拍; 00000011：4 拍。
biu_pad_awlock	O	1'b0	SYS	写请求对应的访问方式： 0：normal access； 1：exclusive access。
biu_pad_awprot[2:0]	O	3'b0	SYS	写请求的保护类型： 0 | 1 [2]: Data | Instruction； [1]: Secure | Non-secure； [0]: User | Privileged。
biu_pad_awsize[2:0]	O	3'b0	SYS	写请求每拍数据位宽： 000: 8bits; 001:16bits; 010:32bits; 011:64bits; 100: 128bits。
biu_pad_awvalid	O	1'b0	SYS	写地址有效信号。
pad_biu_awready	I	-	SYS	写地址通道 ready 信号。
biu_pad_wvalid	O	1'b0	SYS	写数据有效信号。
pad_biu_wready	I	-	SYS	写数据通道 ready 信号。
biu_pad_wdata[127:0]	O	128'b0	SYS	写数据总线：128 写数据总线
biu_pad_wstrb[15:0]	O	16'b0	SYS	写数据字节有效信号。
biu_pad_wlast	O	1'b0	SYS	写数据最后一拍指示信号。
pad_biu_bid[7:0]	I	-	SYS	写响应 ID。
pad_biu_bresp[1:0]	I	-	SYS	写响应信号： 00：OKAY； 01：EXOKAY； 10：SLVERR； 11：DECERR。
pad_biu_bvalid	I	-	SYS	写响应有效信号。

续下页

表 11.5 - 接上页

信号名	I/O	初始值	时钟域	功能描述
biu_pad_bready	O	1’b1	SYS	写响应通道 ready 信号。
biu_pad_cactive	O	1’b1	-	固定为 1。
biu_pad_csysack	O	1’b1	SYS	低功耗通道请求响应信号。
pad_biu_csysreq	I	-	SYS	低功耗通道请求信号。
AXI/ACE 总线主接口信号（仅限 ACE 模式）				
biu_pad_arbar[1:0]	O	2’b0	SYS	AR channel 扩展信号。
biu_pad_ardomain[1:0]	O	2’b0	SYS	
biu_pad_arsnoop[3:0]	O	4’b0	SYS	
biu_pad_awbar[1:0]	O	2’b0	SYS	AW channel 扩展信号。
biu_pad_awdomain[1:0]	O	2’b0	SYS	
biu_pad_awsnoop[2:0]	O	3’b0	SYS	
biu_pad_rack	O	1’b0	SYS	RACK 信号。
biu_pad_wack	O	1’b0	SYS	WACK 信号。
pad_biu_acaddr[39:0]	I	-	SYS	AC channel 地址。
pad_biu_acprot[2:0]	I	-	SYS	AC channel 信号。
pad_biu_acsnoop[3:0]	I	-	SYS	
pad_biu_acvalid	I	-	SYS	
biu_pad_acready	O	1’b0	SYS	
biu_pad_crresp[4:0]	O	5’b0	SYS	CR channel 信号。
biu_pad_crvalid	O	1’b0	SYS	
pad_biu_crready	I	-	SYS	
biu_pad_cddata[127:0]	O	128’b0	SYS	CD channel 信号。
biu_pad_cdlast	O	1’b0	SYS	
biu_pad_cdvalid	O	1’b0	SYS	
pad_biu_cdready	I	-	SYS	

11.1.7　支持的 Coherency Transaction 类型

当主设备接口被选择为 ACE 总线时，支持的 Coherency Transaction 类型如 表 11.6 所示

表 11.6: ACE 协议支持的 coherency transaction 类型

请求类型	来源
ARSNOOP	

续下页

表 11.6 - 接上页

请求类型	来源
ReadNoSnoop (ARDOMAIN=2’b11， ARLEN=0)	non-cacheable weak-ordered 读请求 non-cacheable strong-ordered 读请求
ReadOnce (ARDOMAIN=2’b01， ARLEN=0/3)	指令、数据以及 TLB 的 no allocate 读请求 当 L2 内相应 cacheline 状态为 miss 时的 DCP no allocate 读请求
ReadNotSharedDirty (ARDOMAIN=2’b01， ARLEN=3)	指令、数据以及 TLB 的 allocate 读请求 指令、数据以及 TLB 的预取请求 prefetch.r*/prefetch.i* 指令 当 L2 内相应 cacheline 状态为 miss 时的 DCP allocate 读请求
ReadUnique (ARDOMAIN=2’b01， ARLEN=3)	当 L2 cache 内相应 cache line miss 时的写请求 allocate partial 写请求 prefetch.w* 指令 当 L2 内相应 cacheline 状态为 miss 时的 DCP allocate partial 写请求 当 L2 内相应 cacheline 状态为 miss 或 shared 时的 DCP no allocate partial 写请求
CleanUnique (ARDOMAIN=2’b01， ARLEN=3)	当 L2 cache 内相应 cacheline 状态为 shared 时的 allocate partial 写请求 当 L2 内相应 cacheline 状态为 shared 时的 DCP allocate 写请求
MakeUnique (ARDOMAIN=2’b01， ARLEN=3)	当 L2 cacheline miss/shared 时， full cache line 写请求 当 L2 内相应 cacheline 状态为 miss 时的 DCP full cacheline 写请求 当 L2 内相应 cacheline 状态为 shared 时的 DCP no allocate full cacheline 写请求
CleanShared (ARDOMAIN=2’b01， ARLEN=3)	cbo.clean* /dcache.cva/dcache.cpa 指令
CleanInvalid (ARDOMAIN=2’b01， ARLEN=3)	cbo.flush*/dcache.civa/dcache.cipa 指令
MakeInvalid (ARDOMAIN=2’b01， ARLEN=3)	cbo.inval* /dcache.iva/dcache.ipa 指令
DVM Message (ARDOMAIN=2’b01， ARLEN=0)	icache.ialls/icache.ipa/icache.iva/fence.i/sfence/sync 指令，其中 sync 指令只用于传输 DVM Sync
DVM Complete (ARDOMAIN=2’b01， ARLEN=0)	DVM 握手
AWSNOOP	

续下页

表 11.6 - 接上页

请求类型	来源
WriteNoSnoop (AWDOMAIN=2'b11, AWLEN=0， WSTRB 不固定)	non-cacheable weak-order 写请求 non-cacheable strong-order 写请求
WriteClean (AWDOMAIN=2'b01, AWLEN=3， WSTRB 全为 1)	当 L1/L2 cache 相应 cache line 状态为 dirty 时，执行 cbo.clean*/dcache.cva/dcache.cpa 指令
WriteBack (AWDOMAIN=2'b01, AWLEN=3， WSTRB 全为 1)	当 L1/L2 cache 相应 cache line 状态为 dirty 时，执行 cbo.flush*/dcache.civa/dcache.cipa 指令 no allocate 写请求 当 L1/L2 对应 cache line 的状态为 dirty 时， L2 cacheline 替换行为 当 L2 对应 cacheline 的状态为 miss 时， DCP no allocate 写请求
Evict (AWDOMAIN=2'b01, AWLEN=3)	当 L1/L2 对应 cacheline line 的状态为 clean 时，L2 cache line 替换行为

*：该指令为 CMO 扩展指令，具体内容请详见 *RISC-V Base Cache Management Operation ISA Extensions*

C920V3 支持的 DVM message type 包括：

- TLB Invalidate
- Physical Instruction Cache Invalidate
- Synchronization

对应 ARADDR 如下 表 11.7 所示：

表 11.7: TLB Invalidate First Part

ARADDR bits	Name	Function
[47:43][1]	Reserved	5'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

续下页

表 11.7 - 接上页

ARADDR bits	Name	Function
[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 11.8: TLB Invalidate Second Part

ARADDR bits	Description
[47:44]	4’b0
[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 11.9: Physical Instruction Cache Invalidate First Part

ARADDR bits	Name	Function
[47:40][1]	Reserved	8’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.

续下页

表 11.9 - 接上页

ARADDR bits	Name	Function
[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 11.10: Physical Instruction Cache Invalidate Second Part

ARADDR bits	Description
[47:40][1]	8'b0
[39:4]	PA[39:4]
[3:0]	4'b0

1:Present when Sv48 selected.

表 11.11: Synchronization

ARADDR bits	Name	Function
[47:40][1]	Reserved	8'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

C920V3 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（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。

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：

- 仅支持 INCR 的传输模式，且 LEN 只支持 0 和 3；
- 要求 CACHE[3:0] 必须为 4’b1111、4’b1011、4’b0111，否则返回 SLVERR；
- 要求 SIZE[2:0] 只能为 3’b100，否则返回 SLVERR；
- 不支持独占式访问；
- WSTRB 在 LEN 为 0 时支持任意字节有效，在 LEN 为 3 时要求全为 1；
- AxADDR 在 LEN 为 0 时 16B 边界对齐，在 LEN 为 3 时 64B 边界对齐；
- 支持 5 bit 的 AxID；
- 支持读和写操作。

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 11.12 所示。

表 11.12: 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；

11.2.6 设备一致性接口信号

表 11.13: 设备一致性接口信号

信号名	I/O	Reset	定义
读地址通道相关接口			
pad_slvif_araddr[39:0]/ [47:0]	I	-	读地址总线： 40/48 位地址总线。
pad_slvif_arburst[1:0]	I	-	突发传输指示信号： 指示传输是一次突发传输的一部分： 01： INCR；
pad_slvif_arcache[3:0]	I	-	读请求对应的 cache 属性： [3]:Other Allocate [2]:Allocate [1]:Modifiable [0]:Bufferable
pad_slvif_arid[4:0]	I	-	读地址 ID。
pad_slvif_arlen[7:0]	I	-	突发传输长度： 00000000： 1 拍; 00000011： 4 拍。
pad_slvif_arlock	I	-	读请求对应的访问方式： 0： normal access 1： exclusive access
pad_slvif_arprot[2:0]	I	-	读请求的保护类型： 0 | 1 [2]: Data | Instruction [1]: Secure | Non-Secure [0]: User | Privileged
pad_slvif_arsize[2:0]	I	-	读请求每拍数据位宽： 100: 128bits。

续下页

表 11.13 - 接上页

信号名	I/O	Reset	定义
pad_slvif_arvalid	I	-	读地址有效信号。
slvif_pad_arready	O	1’b1	读地址通道 ready 信号。
读数据通道相关接口			
slvif_pad_rdata[127:0]	O	128’b0	读数据总线：128 位数据总线。
slvif_pad_rid[4:0]	O	5’b0	读数据 ID。
slvif_pad_rresp[1:0]	O	4’b0	读响应信号： [1:0]： 00：OKAY 10：SLVERR
slvif_pad_rlast	O	1’b0	读数据最后一拍指示信号。
slvif_pad_rvalid	O	1’b0	读数据有效信号。
pad_slvif_rready	I	-	读数据通道 ready 信号。
写地址通道相关接口			
pad_slvif_awaddr[39:0]/ [47:0]	I	-	写地址总线： 40/48 位地址总线。
pad_slvif_awburst[1:0]	I	-	突发传输指示信号： 指示传输是一次突发传输的一部分： 01：INCR；
pad_slvif_awcache[3:0]	I	-	写请求对应的 cache 属性： [3]:Other Allocate [2]:Allocate [1]:Modifiable [0]:Bufferable
pad_slvif_awid[4:0]	I	-	写地址 ID。
pad_slvif_awlen[7:0]	I	-	突发传输长度： 00000000：1 拍; 00000011：4 拍。
pad_slvif_awlock	I	-	写请求对应的访问方式： 0：normal access 1：exclusive access
pad_slvif_awprot[2:0]	I	-	写请求的保护类型： 0 | 1 [2]: Data | Instruction [1]: Secure | Non-Secure [0]: User | Privileged

续下页

表 11.13 - 接上页

信号名	I/O	Reset	定义
pad_slvif_awsize[2:0]	I	-	写请求每拍数据位宽： 100：128bits。
pad_slvif_awvalid	I	-	写地址有效信号。
slvif_pad_awready	O	1'b1	写地址通道 ready 信号。
写数据通道相关接口			
pad_slvif_wdata[127:0]	I	-	写数据总线： 128 位写数据总线。
pad_slvif_wstrb[15:0]	I	-	写数据字节有效信号。
pad_slvif_wlast	I	-	写数据最后一拍指示信号。
pad_slvif_wvalid	I	-	写数据有效信号。
slvif_pad_wready	O	1'b1	写数据通道 ready 信号。
写响应通道相关信号			
slvif_pad_bid[4:0]	O	5'b0	写响应 ID。
slvif_pad_bresp[1:0]	O	2'b0	写响应信号： 00：OKAY； 10：SLVERR。
slvif_pad_bvalid	O	1'b0	写响应有效信号。
pad_slvif_bready	I	-	写响应通道 ready 信号。

11.3　低延时外设接口

C920V3 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（请参考 *AMBA 规格说明–AMBA® AXI™ and ACE™ Protocol Specification*）。

11.3.1　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 128bits，地址总线位宽 40/48 bits 可配；
- 支持每个核心 7 个读 outstanding 和 12 个写 outstanding；四个核心最多 28 个读 outstanding 和 32 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。

11.3.2 低延时外设接口的 Outstanding 能力

本小节列出了 C920V3 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	7n，n= 核心数量	每个核心最多发出 7 个，四个核心最多 28 个读请求。
Write Issuing Capability	12n，n= 核心数量	每个核心最多发出 12 个，四个核心最多 32 个写请求。

表 11.15: AXI LLP ARID 编码

ARID[7:0]	适用场景	每个 ID 的 Outstanding
{1'b0, 2'b(coreid), 5'b00???}	Non-cacheable weak-ordered 读请求	所有 Non-cacheable 读请求 outstanding 共 28 个。
{1'b0, 2'b(coreid), 5'h10}	Non-cacheable weak-ordered 取指请求	
{1'b0, 2'b(coreid), 5'h1e}	Non-cacheable weak-ordered 抢锁请求	
{1'b0, 2'b(coreid), 5'h1d}	Non-cacheable strong-ordered 读请求	

- AR 通道同一个 ID 不会同时在主设备接口与 LLP 出现

表 11.16: AXI LLP AWID 编码

AWID[7:0]	适用场景	每个 ID 的 Outstanding
{2'b00, 1'b?, 1'b0, 4'h????}	Non-cacheable weak-ordered 写请求	每个 ID 无 outstanding，所有 Non-cacheable weak-ordered 写请求 outstanding 总共 32 个。
{1'b0, 2'b(coreid), 5'h1e}	Non-cacheable weak-ordered 抢锁写请求	每个 ID 核无 outstanding。
{1'b0, 2'b(coreid), 5'h1d}	Non-cacheable strong-ordered 写请求	Non-cacheable strong-ordered 写请求 outstanding 为 32 个。

- AW 通道同一个 ID 不会同时在主设备接口与 LLP 出现

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

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：C920V3 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。

11.3.4 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

C920V3 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。C920V3 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。C920V3 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。

C920V3 协处理器结构图如下 图 12.1 所示：

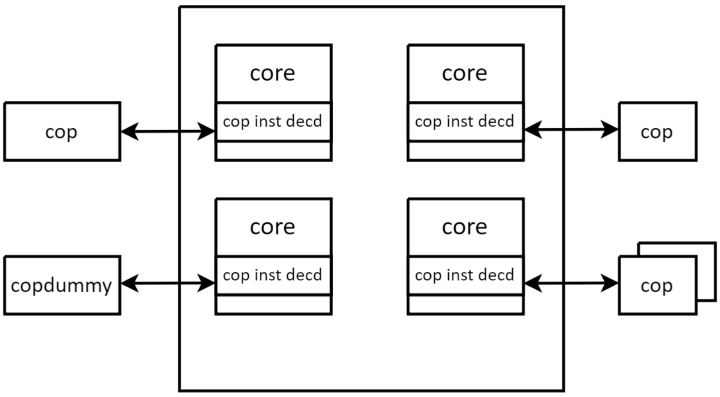


图 12.1: C920V3 协处理器结构示意图

C920V3 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、VGPR 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 C920V3 协处理器指令扩展

12.2.1 协处理器指令接口特性

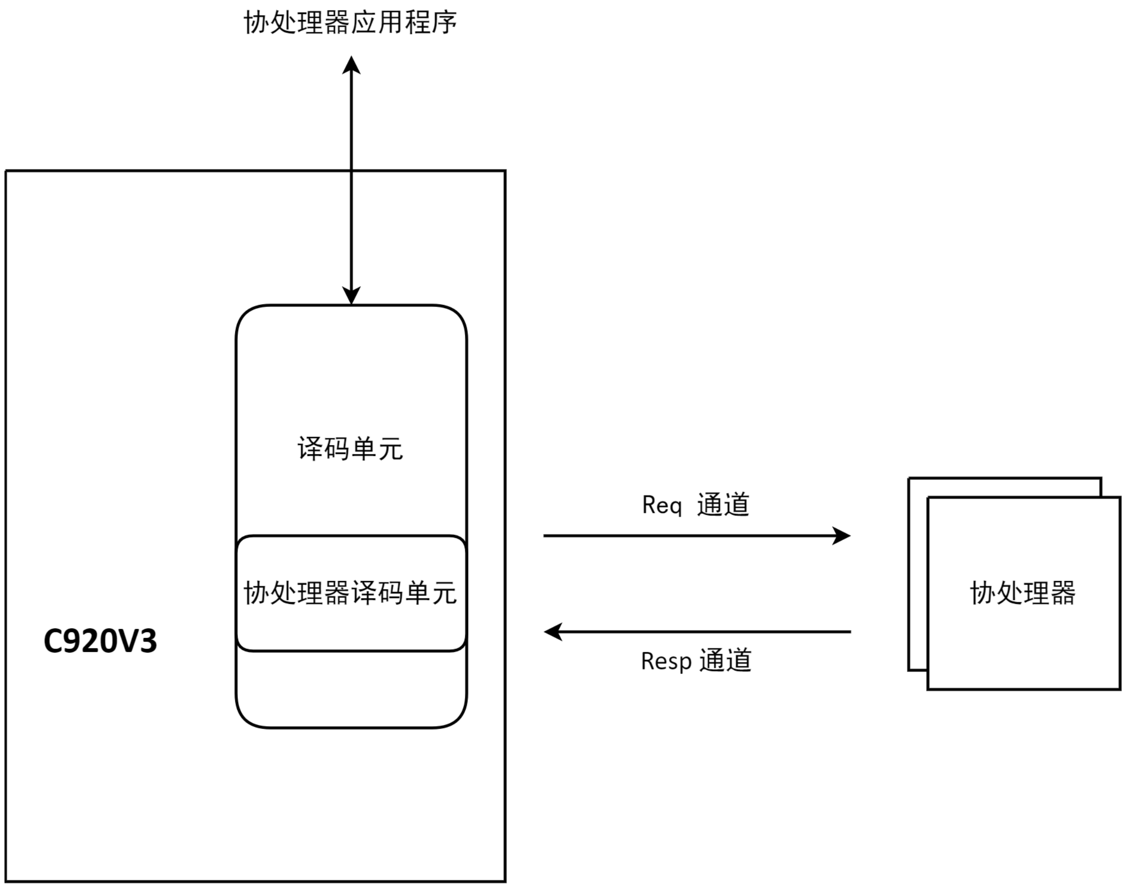


图 12.2: C920V3 协处理器指令接口示意图

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、vector 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。

C920V3 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、vector 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、vector 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；协处理器 vector 扩展指令最多支持 4 个 vector 寄存器作为 source 发送给协处理器，其中包含 vm 寄存器，在 vector 标准编码中 vm 使用单 bit 编码表示，总是指向 v0 寄存器。

index	func	rs2/vs2/fs2	rs1/vs1/fs1	func3	rd/vd/fd	custom

图 12.3: C920V3 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、vgpr 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 5bit，最多可以控制 32 个协处理器。

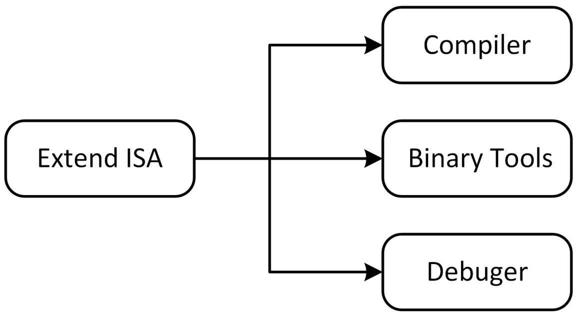


图 12.4: C920V3 协处理器工具支持

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

C920V3 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 ct_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-7 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。
3. 协处理器的 Vector 指令默认 lmul = 1， vlmax = 1 * vlenb/vsew。

协处理器译码模块（ct_idu_xxx/ct_idu_id_cop_decd.v）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 C920V3 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 C920V3 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 C920V3 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。

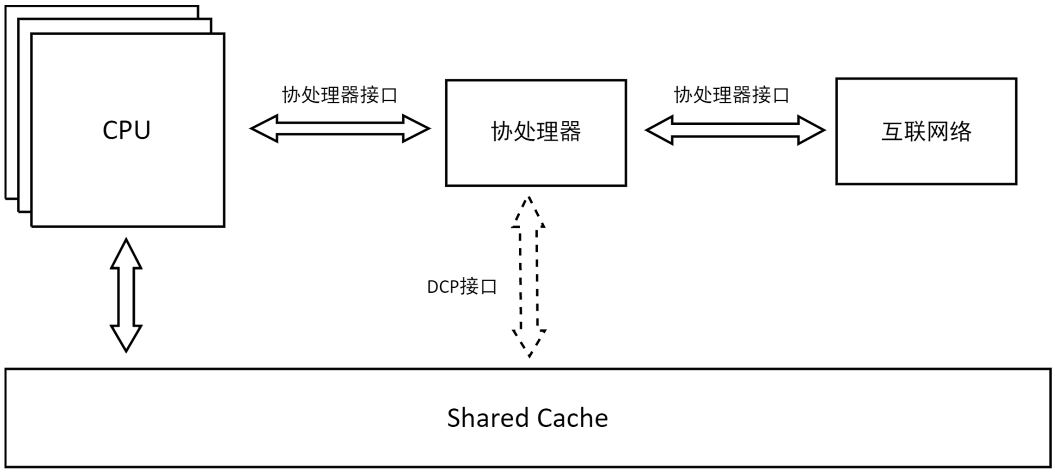


图 12.5: C920V3 协处理器接口互联场景

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

C920V3 兼容 RISC-V Debug V0.13.2 协议标准 。对外调试接口支持 5 线 JTAG（标准 JTAG5）模式。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。

调试接口的主要特性如下：

- 支持 5 线 JTAG 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。

C920V3 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。

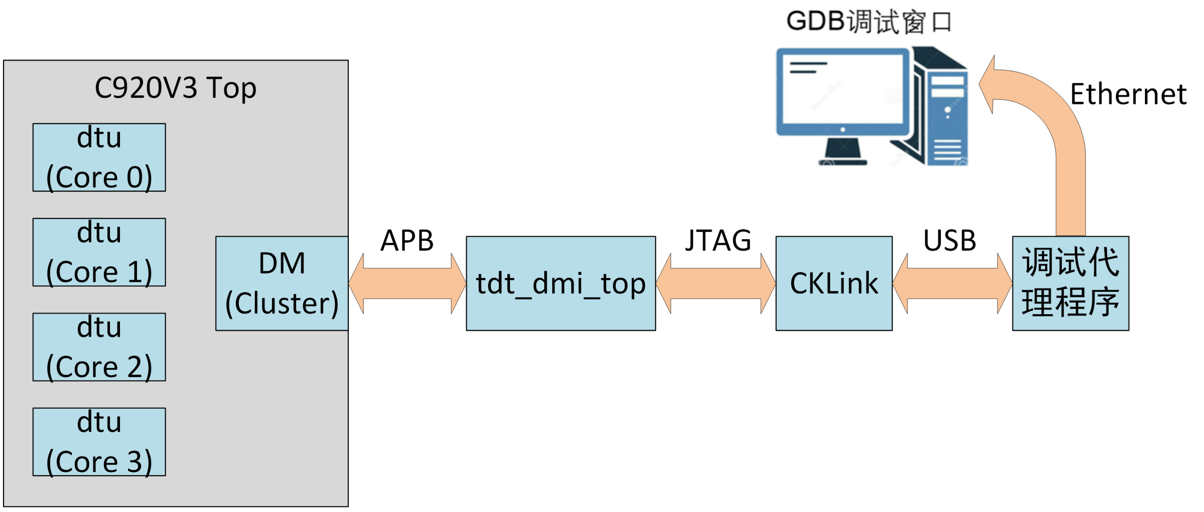


图 13.1: 调试接口在整个 CPU 调试环境中的位置

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，C920V3 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 version 0.13.2 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 同步调试与异步调试

13.3.1 同步调试

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

13.3.2 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 。

13.4 玄铁扩展调试寄存器

表 13.1 所示为 C920 在 DMI Address 编码域上实现的玄铁扩展 DM 寄存器列表。

表 13.1: 玄铁扩展 DM 寄存器映射及描述

地址	寄存器名称	描述
玄铁扩展寄存器		
0x1F	ITR	指令传输寄存器

续下页

表 13.1 - 接上页

地址	寄存器名称	描述
0x70	CUSTOMCS	客制控制和状态寄存器
0x71	CUSTOMCMD	客制命令寄存器
0x72 ~ 0x79	CUSTOMBUF0 ~ 7	客制 Buffer0~7（未实现）
0x7F	COMPID	组件 ID 寄存器

13.4.1 指令传输寄存器（ITR）

相较于 Program Buffer，向 ITR（Instruction Transfer Register）写入的指令会被直接送到 IFU 执行，省去取址等操作，更具鲁棒性。

表 13.2: ITR 寄存器描述

位域	位域名称	读写	描述	Reset
0:31	instruction	读写	DM 向 core 内发送的待执行的指令码	0

13.4.2 扩展控制和状态寄存器（CUSTOMCS）

用于描述玄铁 C920 扩展的抽象命令实现和扩展。寄存器描述如下：

表 13.3: CUSTOMCS 寄存器描述

位域	位域名称	读写	描述	Reset
0:15	coredbginfo	只读	用于表示 C920 支持的核内调试资源情况。每一位分别对应一个配置功能，若为 1 则表示实现对应功能； bit0：debug info 硬件绑定为 1 bit1：pcfifo 硬件绑定为 1 bit2：pipelineinfo 硬件绑定为 0	16’b0000 0000 0000 0011
17	cuscmdbusy	只读	如果 pc/context sample 正在执行，则该位为 1	0
25:28	cusbufcnt	只读	custom buffer 的数量，硬件绑定为 0	0
29:31	cuscmderr	只读	0：no error 1：busy error（在 cuscmdbusy 为 1 时写 CUSTOM-CMD 寄存器） 2：not support（即当前向 CUSTOMCMD 寄存器中写入的 type 类型是不支持的）	0

13.4.3 扩展命令寄存器（CUSTOMCMD）

用于执行玄铁 C920 扩展命令的寄存器。寄存器描述如下：

表 13.4: CUSTOMCMD 寄存器描述

位	名称	读写	描述	Reset
0:23	-	-	-	-
24:31	type	WARZ	Custom command 类型 0: async halt req 发起异步调试请求 1: reserved 2: pc sample 3: context sample Others: reserved	0

13.4.4　扩展组件 ID 寄存器（COMPID）

该寄存器用于指明当前 DM 模块实现的内容和实现版本信息。

表 13.5: CUSTOMCMD 寄存器描述

位	名称	读写	描述	Reset
0:7	version	只读	版本，硬件绑定为 8'b0000001	8'b0000001
8:19	comptype	只读	组件类型，硬件绑定为全 0	0
20:31	vendorID	只读	JEP106 vendor id，硬件绑定为 12'b101101101111	12'b101101101111

14 功耗管理

从 R1S4 版（也称作“1.4.x 版”）开始，C920V3 在功耗管理方面做了大幅的提升，包括支持多个 power domain，支持单个 core 下电，支持 cluster 下电，支持通过外部硬件接口清空 L2 cache 等。本章将详细介绍 C920V3 的功耗管理特性。

14.1 Power Domain

如 图 14.1 所示，C920V3 最多可以划分为 6 个 Power Domain，分别是：

1. PD_CORE(x): 每个核心都具有单独的 power domain，每个核心可以单独掉电。
2. PD_TDT: 包含 tdt_mp_top 模块和外部的 tdt_dmi_top 模块，PD_TDT 和 CPU 电源相互独立：

- 用于支持 cluster 掉电时，PD_TDT 仍然上电；
- 支持核心正常工作时，把 PD_TDT 掉电以节省功耗；

3. PD_CPU: 覆盖 cluster 内部除了以上电源域之外的部分。

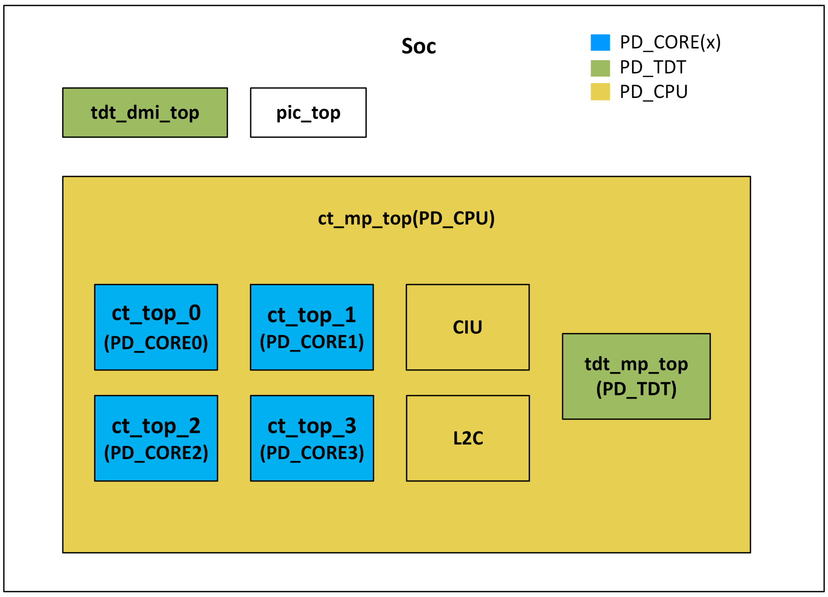


图 14.1: C920V3 Power Domain

14.2 低功耗模式概要

C920V3 支持下列几种低功耗模式：

- 正常模式：各个核心与 L2 都处于正常运行的状态。
- 核心 WFI 模式：个别核心处于 WFI 模式。
- 单个核心下电：个别核心处于下电状态。
- Cluster 下电：整个 cluster，包括 4 个核心和 L2，都处于下电状态。

14.3 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。

14.4 单个核心下电流程

系统可通过关断核心电源的方式完全关断核心的静态功耗。核心电源关断流程如下：

C920V3 被关断核心执行的操作：

1. 通知 SoC 即将执行单个核心下电流程，具体方式取决于 SoC 的设计。
2. 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 寄存器的中断使能位。如果下电流程执行在 M 态，则关闭 MSTATUS 和 MIE 的中断使能；如果下电流程执行在 S 态，则关闭 SSTATUS 和 SIE 的中断使能。
3. 关闭数据预取。
4. 核心执行 D-Cache INV&CLR ALL，将 dirty line 写回 L2 Cache。
5. 核心关闭 D-Cache（注意：在清 cache 和关 cache 之间不能有 store 指令）。
6. 关闭核心的 SMPEN 位，屏蔽对该核心的 snoop 请求。
7. 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。
8. 核心执行低功耗指令 WFI，核心进入低功耗模式。

系统执行的操作：

1. 系统检测到核心的低功耗输出信号 core(x)_pad_lpmd_b 有效。
2. 系统将 pad_tdt_dm_core_unavail[x] 置高，以屏蔽针对待下电核心的调试请求。
3. 激活待下电核心的输出信号钳位。
4. 系统拉低待下电核心的复位信号 pad_core(x)_rst_b。
5. 关闭核心的电源。

核心在断电的状态下，只有通过复位才能重新启动。核心的重新上电流程如下：

1. 系统检测到特定的事件，决定对核心重新上电（也可以称为“唤醒”）。
2. 系统设置被唤醒核心的复位地址。
3. 拉低核心的复位信号。
4. 打开电源，保持复位信号不释放。
5. 释放核心的输出信号钳位。
6. 释放核心复位的信号。
7. 被唤醒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。

14.5 Cluster 下电流程（硬件清空 L2）

首先，确保 Cluster 内除了主核以外，其余三个核心的电源均已关闭。这里的“主核”是指最后一个下电的核心，它可以是 4 个核心中的任意一个。

主核执行的操作：

1. 通知 SoC 即将执行 Cluster 下电流程。具体方式取决于 SoC 的设计。
2. 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 寄存器的中断使能位。
3. 关闭数据预取。
4. 执行 D-Cache INV&CLR ALL 操作。
5. 关闭 D-Cache（注意：在清 cache 和关 cache 之间不能有 store 指令）。
6. 关闭核心的 SMPEN 位。
7. 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。
8. 执行低功耗指令 WFI，进入低功耗模式。

系统执行的操作：

1. 系统检测到主核的低功耗输出信号 core(x)_pad_lpmd_b 有效。
2. 系统将 pad_tdt_dm_core_unavail[x] 置高，以屏蔽针对主核的调试请求。
3. 激活主核的输出信号钳位。
4. 系统拉低主核的复位信号 pad_core(x)_rst_b。

5. 关闭主核的电源。
6. 系统拉高 pad_cpu_l2cache_flush_req，开始清空 L2 cache 的过程。
7. 等待 C920V3 返回 cpu_pad_l2cache_flush_done=1。
8. 系统拉低 pad_cpu_l2cache_flush_req。（随后 C920V3 将拉低 cpu_pad_l2cache_flush_done）
9. 系统确保 DCP（如果已配置）没有新的请求。
10. 等待 C920V3 返回 cpu_pad_no_op=1。
11. 激活顶层的输出信号钳位。
12. 拉低 L2 复位信号 pad_cpu_rst_b。
13. 关闭顶层电源。

Cluster 通过复位重新上电，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. 释放各核心和顶层的输出信号钳位。
4. 释放各核心和顶层的复位信号。
5. 执行复位异常服务程序，恢复 CPU 状态。

14.6 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 C920V3 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. 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。
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. 拉高 pad_cpu_l2cache_flush_req，开始硬件清空 L2 cache 的过程。
4. 等待 C920V3 返回 cpu_pad_l2cache_flush_done=1，代表 L2 已经清空。
5. 拉低 pad_cpu_l2cache_flush_req。（随后 C920V3 将拉低 cpu_pad_l2cache_flush_done。）
6. 等待 cpu_pad_no_op==1'b1，即 L2 进入空闲状态。（此时所有的 CPU 也仍然处于低功耗状态）
7. 激活 cluster 输出信号的 clamp。
8. Assert 全部复位信号。
9. 将整个 cluster 掉电。

14.7 低功耗相关的编程模型和接口信号

14.7.1 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。

机器模式侦听使能寄存器（MSMPR）

该寄存器为新增的寄存器，宽度为 64 位，只有 bit[0] 有定义（=SMPEN），默认值为 0。该寄存器的功能是控制核心能否接受侦听（snoop）请求。

- MSMPR.SMPEN = 0 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。
- MSMPR.SMPEN = 1 时，核心能够处理侦听请求，顶层发送侦听请求给核心。

核心下电前，要求设置核心对应的 SMPEN=0；核心上电后，软件在打开 D-Cache 和 MMU 之前需要设置 SMPEN=1。核心在正常工作模式下，必须保持 SMPEN=1。

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。

14.7.2 接口信号

C920V3 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）。

- pad_cpu_l2cache_flush_req 与 cpu_pad_l2cache_flush_done：

  这一组信号用于在 SoC 的控制下清空 L2 cache，应用场景是 Cluster 下电。"req"信号由 SoC 驱动，"done"信号由 C920V3 驱动。操作顺序是：SoC 首先拉高并保持 "req" 以开始清空 L2 的过程；C920V3 完成清空 L2 的动作以后，返回 "done" =1；SoC 拉低 "req" 信号；随后 C920V3 拉低 "done" 信号。

15 性能监测单元

15.1　PMU 简介

C920V3 性能监测单元（PMU）遵从 RISC-V 标准 sstc 扩展和 sscofpmf 扩展, 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）

15.2　PMU 的编程模型

15.2.1　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。C920V3 中每个计数器可以配置任意一个事件。将事件索引值写入机器模式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*

15.2.2　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* 。

15.3 PMU 相关的控制寄存器

15.3.1 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超级用户模式/用户模式计数器。

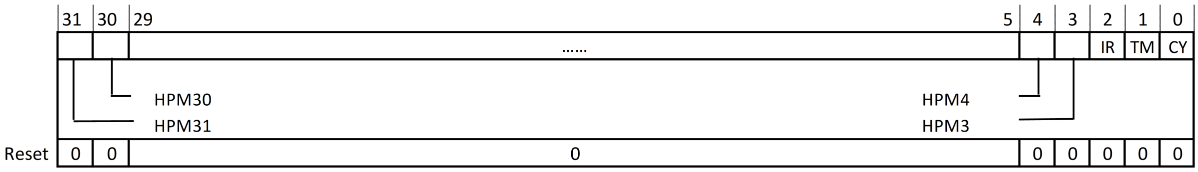


图 15.1: 机器模式计数器访问授权寄存器（MCOUNTEREN）

表 15.1: 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	HPM*n*	shpmcounter*n* /hpmcounter*n* 寄存器 S-mode 访问位： 0：S-mode 访问 shpmcounter*n* / hpmcounter*n* 将发生非法指令异常 1：S-mode 能正常访问 shpmcounter*n* /hpmcounter*n*
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：当 mcounteren 对应位为 1，S-mode 能正常访问 time/stimecmp 寄存器，否则将发生非法指令异常
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

15.3.2 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。

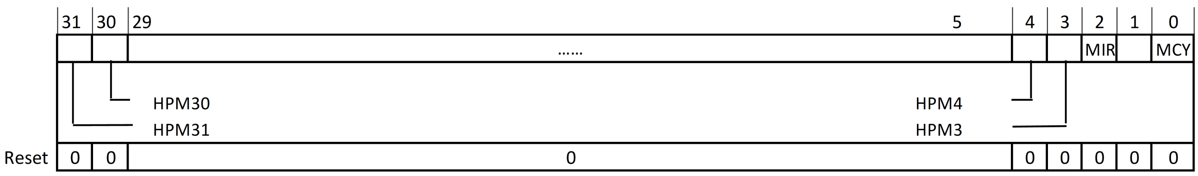


图 15.2: 机器模式计数禁止授权寄存器（MCOUNTINHIBIT）

表 15.2: 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
1	-	-	-
0	读写	MCY	mcycle 寄存器禁止计数位： 0：正常计数 1：禁止计数

15.3.3 机器模式性能监测控制寄存器（MHPMCR）

机器模式性能监测控制寄存器，此寄存器为玄铁自扩展寄存器。

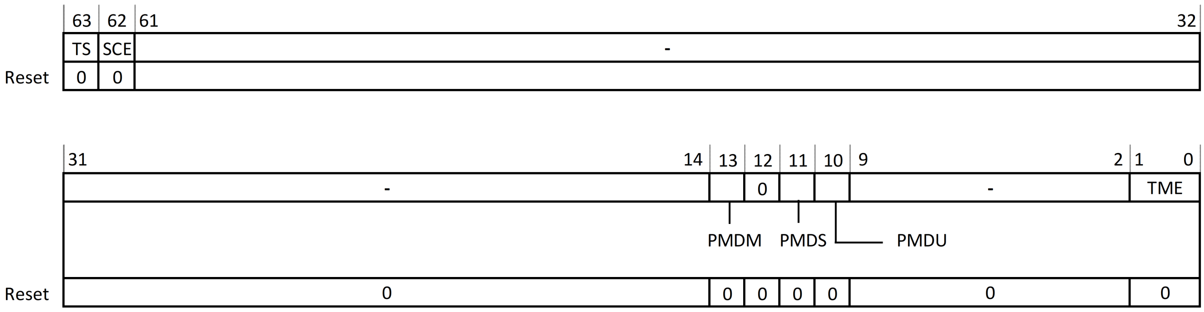


图 15.3: 机器模式性能监测控制寄存器 (MHPMCR)

表 15.3: 机器模式性能监测控制寄存器（MHPMCR）说明

位	读写	名称	介绍
63	读写	TS	触发状态位，指示性能监测是否处于触发态，该位不建议手动修改。 1’b0：未处于触发状态 1’b1：处于触发状态
62	读写	SCE	S-mode 控制使能位： 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
13	读写	PMDM	Performance monitor disable machine mode counting： 停止 M-mode 计数控制位。该位是 mxstatus 中的映射 1’b0：M-mode 正常计数 1’b1：禁止 M-mode 计数

续下页

表 15.3 - 接上页

位	读写	名称	介绍
11	读写	PMDS	Performance monitor disable supervisor mode counting： 停止 S-mode 计数控制位。该位是 mxstatus 中的映射 1’b0：S-mode 正常计数 1’b1：禁止 S-mode 计数
10	读写	PMDU	Performance monitor disable user mode counting： 停止 U-mode 计数控制位。该位是 mxstatus 中的映射 1’b0：U-mode 正常计数 1’b1：禁止 U-mode 计数
1：0	读写	TME	Trigger Mode 使能位： 2’b00：触发模式未使能，正常计数 2’b01：Trigger/Stop 触发模式使能，当程序地址与起始触发寄存器 (mhpmsp) 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 (mhpmep) 匹配，则触发功能关闭，结束计数 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 2’b11：保留状态，无意义，将无法计数

15.3.4 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。

注意：机器模式写相应事件计数器权限，必须同时满足 MCOUNTEREN 和 MCOUNTERWEN 的相应位都使能。在 MCOUNTEREN 的相应位没有使能的时候，使能 MCOUNTERWEN 的相应位，依然没有写权限。

该寄存器为机器模式扩展寄存器，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。

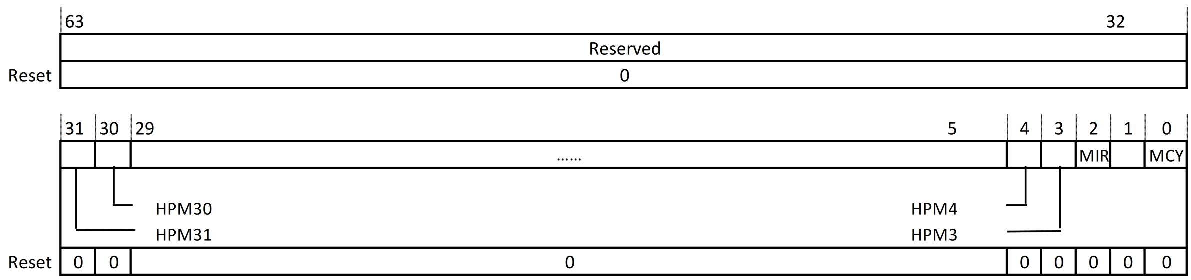


图 15.4: 机器模式计数器写使能寄存器（MCOUNTERWEN）

表 15.4: 机器模式写使能寄存器说明

位	读写	名称	介绍
31：3	读写	HPM*n*	scounter*n* 寄存器超级用户模式写使能位： HPM*n* 为 0 时，表示 scounter*n* 在超级用户模式下不可写，写会产生非法异常 HPM*n* 为 1 时，表示 scounter*n* 在超级用户模式下可写
2	读写	MIR	sinst 寄存器超级用户模式写使能位 0：表示指令计数在超级用户模式不可写，写会产生非法异常 1：表示指令计数在超级用户模式可写
1	-	-	-
0	读写	MCY	scycle 寄存器超级用户模式写使能位 0：表示相应计数器在超级用户模式下不可写，写会产生非法异常 1：表示相应计数器在超级用户模式下可写

15.3.5 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。

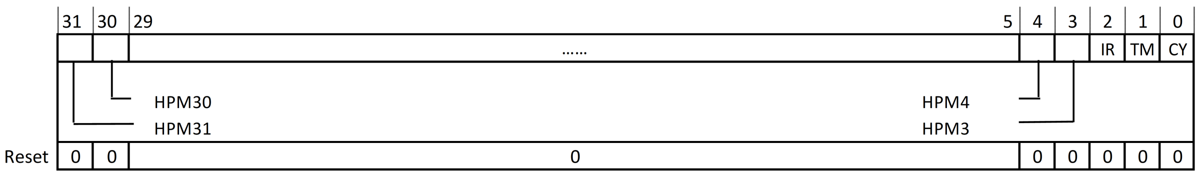


图 15.5: 超级用户模式计数器访问授权寄存器（SCOUNTEREN）

表 15.5: 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 hpmcounter*n* 将发生非法指令异常 1：当 mcounteren 和 scounteren 对应位均为 1，则 U-mode 能正常访问 hpm-counter*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 和 scounteren 对应位均为 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 和 scounteren 对应位均为 1，U-mode 能正常访问 time 寄存器，否则将发生非法指令异常

续下页

表 15.5 - 接上页

位	读写	名称	介绍
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 和 scounteren 对应位均为 1，U-mode 能正常访问 cycle 寄存器，否则将发生非法指令异常

15.3.6 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]。

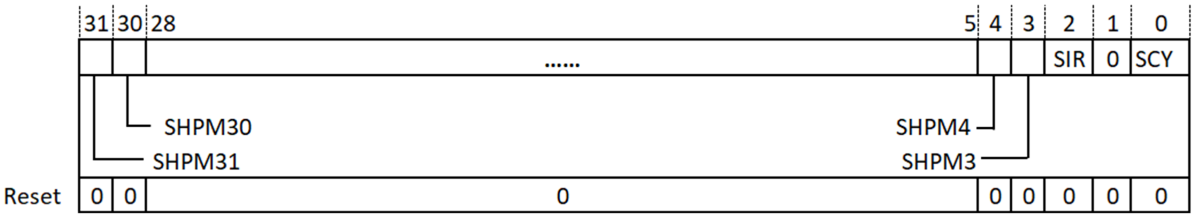


图 15.6: 超级用户模式禁止计数寄存器（scountinhibit）

表 15.6: 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

15.3.7 超级用户模式性能监测控制寄存器（SHPMCR）

超级用户模式性能监测控制寄存器，此寄存器为玄铁自扩展寄存器，是 mhpmcr 的读写映射，不包括 sce。在 S32 时，只映射 TS 到 BIT[31]。

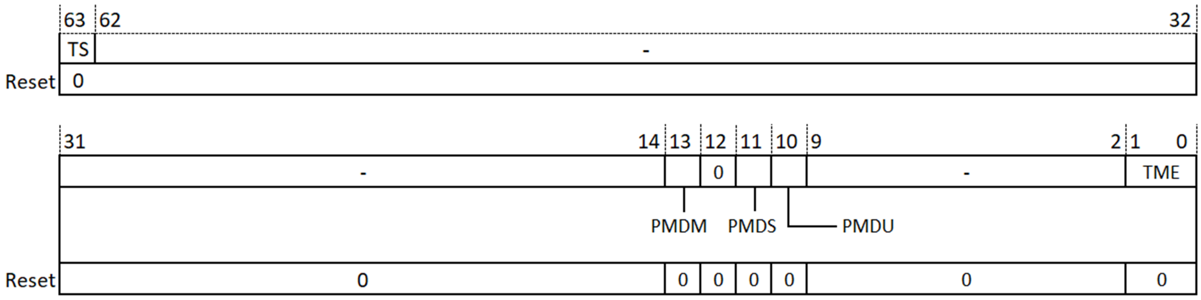
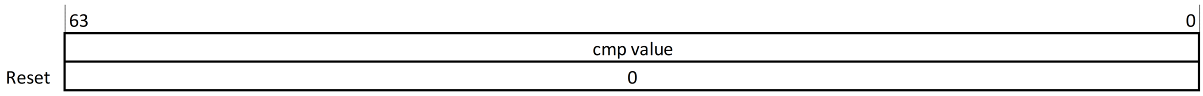


图 15.7: 超级用户模式性能监测控制寄存器

15.3.8 超级用户模式时间比较寄存器（STIMECMP）

超级用户模式时间比较寄存器, 来自 ssct 扩展。



STIMECMP 寄存器说明：

- 复位值为 64'hffff_ffff_ffff_ffff
- 当 menvcfg.STCE 为 0 时，访问 stimecmp csr 产生非法指令异常，并且即使 time 值大于 stimecmp csr 也不产生 stimer 中断。
- 当 STCE 为 1 时，stimecmp csr 作为 stip 的产生源头；
- 当 STCE 为 0 时，memory map 的 stimecmp 作为 stip 的产生源头。

15.3.9 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。

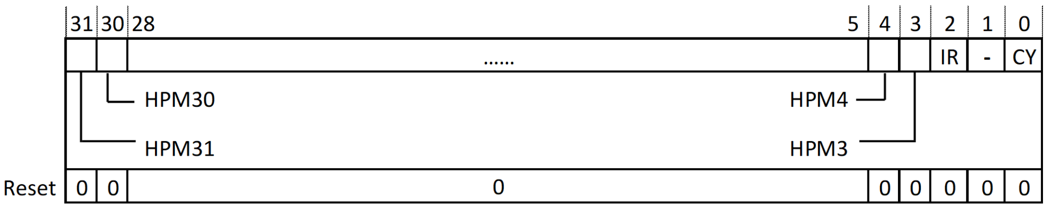


图 15.8: 超级用户模式计数器溢出寄存器（SCOUNTOVF）

表 15.7: 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	只读	IR	1'b0：sinstret 没有发生上溢 1'b1：sinstret 发生上溢
1	-	-	-
0	只读	CY	1'b0：scycle 没有发生上溢 1'b1：scycle 发生上溢

- 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。

15.3.10 机器模式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。C920V3 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。

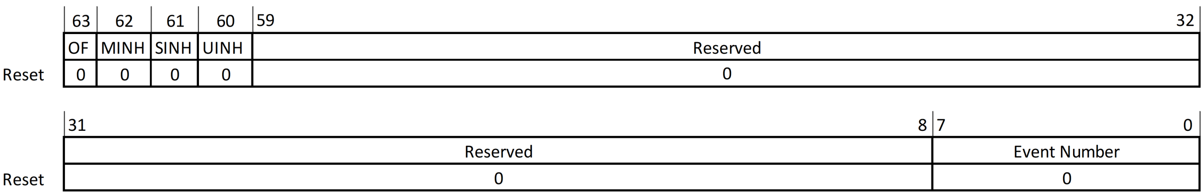


图 15.9: 机器模式性能监测事件选择寄存器（MHPMEVENT3~31）

表 15.8 为机器模式性能监测事件选择寄存器说明。

表 15.8: 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

续下页

表 15.8 - 接上页

位	读写	名称	介绍
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 15.9 为事件选择器和事件以及计数器之间的对应关系。

表 15.9: 计数器事件对应列表

索引	事件
0x1	L1 ICache Access Counter
0x2	L1 ICache Miss Counter
0x3	I-UTLB Miss Counter
0x4	D-UTLB Miss Counter
0x5	JTLB Miss Counter
0x6	Conditional Branch Mispredict Counter
0x7	Conditional Branch Counter
0x8	Indirect Branch Mispredict Counter
0x9	Indirect Branch Counter
0xA	LSU Spec Fail Counter
0xB	Store Instruction Counter
0xC	L1 DCache load access Counter
0xD	L1 DCache load miss Counter
0xE	L1 DCache store access Counter
0xF	L1 DCache store miss Counter
0x10	未实现定义
0x11	未实现定义
0x12	未实现定义
0x13	未实现定义
0x14	RF Launch Fail Counter
0x15	RF Reg Launch Fail Counter
0x16	RF Instruction Counter
0x17	LSU Cross 4K Stall Counter
0x18	LSU Other Stall Counter
0x19	LSU SQ Discard Counter

续下页

表 15.9 – 接上页

索引	事件
0x1A	LSU SQ Data Discard Counter
0x1B	IFU Branch Target Mispred Counter
0x1C	IFU Branch Target Instruction Counter
0x1D	ALU Instruction Counter
0x1E	LDST Instruction Counter
0x1F	Vector SIMD Instruction Counter
0x20	CSR Instruction Counter
0x21	Sync Instruction Counter
0x22	LDST Unaligned Access Counter
0x23	Interrupt Number Counter
0x24	Interrupt Off Cycle Counter
0x25	Environment Call Counter
0x26	Long Jump Counter
0x27	Stalled Cycles Frontend Counter
0x28	Stalled Cycles Backend Counter
0x29	Sync Stall Counter
0x2A	Floating Point Instruction Counter
>= 0x2B	暂未实现定义

15.3.11 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

	63	62	61	60	59 – 32
	OF	MINH	SINH	UINH	Reserved
Reset	0	0	0	0	0

	31 – 8	7 – 0
	Reserved	Event Number
Reset	0	0

图 15.10: MHPMEVENT0/MHPMEVENT2

C920V3 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。

15.4 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、C920V3 扩展的超级用户模式事件计数器和用户模式事件计数器。

机器模式事件计数器具体如 表 15.10 所示。

表 15.10: 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

C920V3 扩展的超级用户模式事件计数器具体如 表 15.11 所示。

表 15.11: 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

用户模式事件计数器列表如 表 15.12 所示。

表 15.12: 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

续下页

表 15.12 – 接上页

名称	索引	读写	初始值	介绍
HPMCOUNTER4	0xC04	URO	0x0	performance-monitoring counter
…	…	…	…	…
HPMCOUNTER31	0xC1F	URO	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 寄存器的只读映射；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。

15.5 触发寄存器

C920V3 触发寄存器如下表所示：

表 15.13: 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：

15.5.1 起始触发寄存器

	63 … 0
Reset	START PC

图 15.11: 起始触发寄存器

15.5.2 终止触发寄存器

	63 … 0
Reset	END PC

图 15.12: 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。

16 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、多核启动示例、同步原语示例、PLIC 设置示例和 PMU 设置示例。

16.1 处理器最优性能配置

采用下列配置，可以发挥 C920V3 的最优性能：

- MHCR = 0x11FF
- MHINT = 0x31EA32C
- MCCR2 = 0xE249000B （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, 0x11ff
  csrs mhcr,x3

# mhint
  li x3, 0x31ea32c
  csrs mhint,x3

# mxstatus
  li   x3, 0x638000
  csrs mxstatus,x3

# msmpr
  csrsi msmpr,0x1

# mccr2
  li x3, 0xe249000b
  csrs mccr2,x3
```

16.2 MMU 设置示例

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

* Function: An example of setting C920V3 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}
*
*************************************************************************/

  /*C920V3 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)
```

16.3 PMP 设置示例

```
/*******************************************************************************
* Function: An example of setting C920V3 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 ~ 0xffffffffff, TOR 模式, 无任何权限
  li x3, (0xffffffffff  >>  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 ~ 0xffffffffff, TOR 模式, 0x100000000 <= addr <
  0xffffffffff 时都会命中 pmpaddr5, 但是 0xffffff000 ~
  0xffffffffff 地址区间无法命中 pmpaddr5 (C920V3 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。
```

16.4 高速缓存示例

16.4.1 高速缓存的开启示例

```
/*C920V3 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

/*C920V3 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
```

16.4.2　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
```

16.4.3　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
```

16.4.4 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 SLLWP 和 SLLCPID。

16.5 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.
```

16.6 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*/
```


16.7　PMU 设置示例

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

/*2.C920V3 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
```

17 附录 A 标准指令术语

C920V3 实现了 RV64IMAFDCV 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。

17.1　附录 A-1 I 指令术语

以下是对 C920V3 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*。

17.1.1　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：

31　25	24　20	19　15	14　12	11　7	6　0
0000000	rs2	rs1	000	rd	0110011

17.1.2　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：

31　　20	19　　15	14　　12	11　　7	6　　0
imm12[11:0]	rs1	000	rd	0010011

17.1.3 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：

31　　20	19　　15	14　　12	11　　7	6　　0
imm12[11:0]	rs1	000	rd	0011011

17.1.4 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

17.1.5 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

17.1.6 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

17.1.7 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：

31　　12	11　　7	6　　0
imm20[19:0]	rd	0010111

17.1.8　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：

31	30　　25	24　　20	19　　15	14　　12	11　　8	7	6　　0
	imm12[9:4]	rs2	rs1	000	imm12[3:0]		1100011

imm12[11]　　imm12[10]

17.1.9　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：

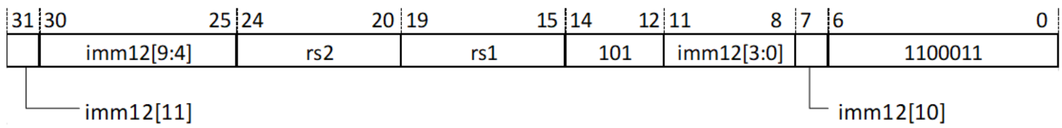
无

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：



17.1.10　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：

31	30 25	24 20	19 15	14 12	11 8	7	6 0
	imm12[9:4]	rs2	rs1	111	imm12[3:0]		1100011
imm12[11]						imm12[10]	

17.1.11 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：

31	30 25	24 20	19 15	14 12	11 8	7	6 0
	imm12[9:4]	rs2	rs1	100	imm12[3:0]		1100011
imm12[11]						imm12[10]	

17.1.12 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：

31	30 25	24 20	19 15	14 12	11 8	7	6 0
imm12[11]	imm12[9:4]	rs2	rs1	110	imm12[3:0]	imm12[10]	1100011

17.1.13 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：

31	30 25	24 20	19 15	14 12	11 8	7	6 0
	imm12[9:4]	rs2	rs1	001	imm12[3:0]		1100011
imm12[11]						imm12[10]	

17.1.14 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*附录 C 系统控制寄存器*。

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	rs1	011	rd	1110011

17.1.15 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*附录 C 系统控制寄存器*。

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	111	rd	1110011

17.1.16　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*附录 C 系统控制寄存器*。

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	rs1	010	rd	1110011

17.1.17　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*附录 C 系统控制寄存器*。

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

17.1.18 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*附录 C 系统控制寄存器*。

指令格式：

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

17.1.19 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*附录 C 系统控制寄存器*。

指令格式：

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

17.1.20 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

17.1.21 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

17.1.22　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

17.1.23　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

17.1.24　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：

31	30 20	20	19 12	11 7	6 0
	imm20[9:0]		imm20[18:11]	rd	1101111

imm20[19]　　imm20[10]

17.1.25 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：

31 20	19 15	14 12	11 7	6 0
imm12[11:0]	rs1	000	rd	1100111

17.1.26 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

17.1.27 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

17.1.28 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：

31　　20	19　　15	14　　12	11　　7	6　　0
imm12[11:0]	rs1	011	rd	0000011

17.1.29　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：

31　　20	19　　15	14　　12	11　　7	6　　0
imm12[11:0]	rs1	001	rd	0000011

17.1.30　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：

31　　20	19　　15	14　　12	11　　7	6　　0
imm12[11:0]	rs1	101	rd	0000011

17.1.31 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:

31 — 12	11 — 7	6 — 0
imm20[19:0]	rd	0110111

17.1.32 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:

31 — 20	19 — 15	14 — 12	11 — 7	6 — 0
imm12[11:0]	rs1	010	rd	0000011

17.1.33 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：

31　　20	19　　15	14　　12	11　　7	6　　0
imm12[11:0]	rs1	110	rd	0000011

17.1.34 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：

31　　25	24　　20	19　　15	14　　12	11　　7	6　　0
0011000	00010	00000	000	00000	1110011

17.1.35 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：

31 25	24 20	19 15	14 12	11 7	6 0
0000000	rs2	rs1	110	rd	0110011

17.1.36 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：

31 20	19 15	14 12	11 7	6 0
imm12[11:0]	rs1	110	rd	0010011

17.1.37 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：

31 25	24 20	19 15	14 12	11 7	6 0
imm12[11:5]	rs2	rs1	000	imm12[4:0]	0100011

17.1.38 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

17.1.39　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

17.1.40　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:

31 25	24 20	19 15	14 12	11 7	6 0
imm12[11:5]	rs2	rs1	001	imm12[4:0]	0100011

17.1.41 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:

31 25	24 20	19 15	14 12	11 7	6 0
0000000	rs2	rs1	001	rd	0110011

17.1.42 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

17.1.43 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

17.1.44 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

17.1.45 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

17.1.46 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

17.1.47　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：

31 20	19 15	14 12	11 7	6 0
imm12[11:0]	rs1	011	rd	0010011

17.1.48　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：

31 25	24 20	19 15	14 12	11 7	6 0
0000000	rs2	rs1	011	rd	0110011

17.1.49 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：

31 25	24 20	19 15	14 12	11 7	6 0
0100000	rs2	rs1	101	rd	0110011

17.1.50 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：

31 26	25 20	19 15	14 12	11 7	6 0
010000	shamt6	rs1	101	rd	0010011

17.1.51 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：

31 25	24 20	19 15	14 12	11 7	6 0
0100000	shamt5	rs1	101	rd	0011011

17.1.52 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：

31 25	24 20	19 15	14 12	11 7	6 0
0100000	rs2	rs1	101	rd	0111011

17.1.53 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

17.1.54 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

17.1.55 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

17.1.56 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：

31 25	24 20	19 15	14 12	11 7	6 0
0000000	shamt5	rs1	101	rd	0011011

17.1.57 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：

31 25	24 20	19 15	14 12	11 7	6 0
0000000	rs2	rs1	101	rd	0111011

17.1.58 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

17.1.59 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

17.1.60 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

17.1.61 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：

31-25	24-20	19-15	14-12	11-7	6-0
0001000	00101	00000	000	00000	1110011

17.1.62 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：

31-25	24-20	19-15	14-12	11-7	6-0
0000000	rs2	rs1	100	rd	0110011

17.1.63 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

17.2 附录 A-2 M 指令术语

以下是对 C920V3 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，

17.2.1 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

17.2.2 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：

31 25	24 20	19 15	14 12	11 7	6 0
0000001	rs2	rs1	101	rd	0110011

17.2.3 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：

31 25	24 20	19 15	14 12	11 7	6 0
0000001	rs2	rs1	101	rd	0111011

17.2.4 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：

31 25	24 20	19 15	14 12	11 7	6 0
0000001	rs2	rs1	100	rd	0111011

17.2.5 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：

31 25	24 20	19 15	14 12	11 7	6 0
0000001	rs2	rs1	000	rd	0110011

17.2.6 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：

31 25	24 20	19 15	14 12	11 7	6 0
0000001	rs2	rs1	001	rd	0110011

17.2.7 MULHSU——有符号无符号乘法取高位指令

语法：

mulhsu 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：

31 25	24 20	19 15	14 12	11 7	6 0
0000001	rs2	rs1	010	rd	0110011

17.2.8 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：

31 25	24 20	19 15	14 12	11 7	6 0
0000001	rs2	rs1	011	rd	0110011

17.2.9 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：

31 25	24 20	19 15	14 12	11 7	6 0
0000001	rs2	rs1	000	rd	0111011

17.2.10 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：

31 25	24 20	19 15	14 12	11 7	6 0
0000001	rs2	rs1	110	rd	0110011

17.2.11 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：

31 25	24 20	19 15	14 12	11 7	6 0
0000001	rs2	rs1	111	rd	0110011

17.2.12 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：

31 25	24 20	19 15	14 12	11 7	6 0
0000001	rs2	rs1	111	rd	0111011

17.2.13 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

17.3 附录 A-3 A 指令术语

以下是对 C920V3 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。

17.3.1 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：

31-27	26	25	24-20	19-15	14-12	11-7	6-0
00000	aq	rl	rs2	rs1	011	rd	0101111

17.3.2 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：

31-27	26	25	24-20	19-15	14-12	11-7	6-0
00000	aq	rl	rs2	rs1	010	rd	0101111

17.3.3　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：

31　27	26	25	24　20	19　15	14　12	11　7	6　0
01100	aq	rl	rs2	rs1	011	rd	0101111

17.3.4　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

17.3.5 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

17.3.6 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：

31 27	26	25	24 20	19 15	14 12	11 7	6 0
10100	aq	rl	rs2	rs1	010	rd	0101111

17.3.7 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

17.3.8 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：

31 27	26	25	24 20	19 15	14 12	11 7	6 0
11100	aq	rl	rs2	rs1	010	rd	0101111

17.3.9 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：

31　27	26	25	24　20	19　15	14　12	11　7	6　0
10000	aq	rl	rs2	rs1	011	rd	0101111

17.3.10　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

17.3.11 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

17.3.12 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

17.3.13 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

17.3.14 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：

31 27	26	25	24 20	19 15	14 12	11 7	6 0
01000	aq	rl	rs2	rs1	010	rd	0101111

17.3.15　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

17.3.16　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

17.3.17　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

17.3.18 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：

31 27	26	25	24 20	19 15	14 12	11 7	6 0
00100	aq	rl	rs2	rs1	010	rd	0101111

17.3.19 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

17.3.20 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

17.3.21 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

17.3.22 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

17.4 附录 A-4 F 指令术语

以下是对 C920V3 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 cNaN 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。

17.4.1 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：

31 25	24 20	19 15	14 12	11 7	6 0
0000000	fs2	fs1	rm	fd	1010011

17.4.2 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：

31–25	24–20	19–15	14–12	11–7	6–0
1110000	00000	fs1	001	rd	1010011

17.4.3 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

17.4.4 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

17.4.5 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

17.4.6 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

17.4.7 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：

31 25	24 20	19 15	14 12	11 7	6 0
1101000	00000	rs1	rm	fd	1010011

17.4.8 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：

31 25	24 20	19 15	14 12	11 7	6 0
1101000	00001	rs1	rm	fd	1010011

17.4.9 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:

31 25	24 20	19 15	14 12	11 7	6 0
1100000	00000	fs1	rm	rd	1010011

17.4.10 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

17.4.11 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

17.4.12 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：

31 25	24 20	19 15	14 12	11 7	6 0
1010000	fs2	fs1	010	rd	1010011

17.4.13 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

17.4.14 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

17.4.15 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：

31 20	19 15	14 12	11 7	6 0
imm12[11:0]	rs1	010	fd	0000111

17.4.16 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：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
fs3	00	fs2	fs1	rm	fd	1000011

17.4.17 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

17.4.18　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

17.4.19　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

17.4.20 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

17.4.21 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

17.4.22 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：

31 25	24 20	19 15	14 12	11 7	6 0
1110000	00000	fs1	000	rd	1010011

17.4.23 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

17.4.24 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

17.4.25 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

17.4.26 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

17.4.27 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：

31 25	24 20	19 15	14 12	11 7	6 0
0010000	fs2	fs1	010	fd	1010011

17.4.28 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

17.4.29 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

17.4.30 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

17.5　附录 A-6 C 指令术语

以下是对 C920V3 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。

17.5.1　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：

15　13	12	11　7	6　2	1　0
100	1	rs1/rd	rs2	10

17.5.2　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：

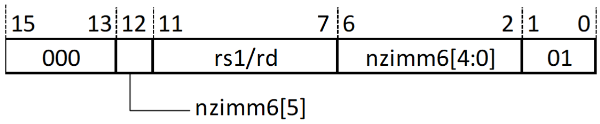
无

说明：

rs1 = rd != 0

nzimm6!=0

指令格式：



17.5.3　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

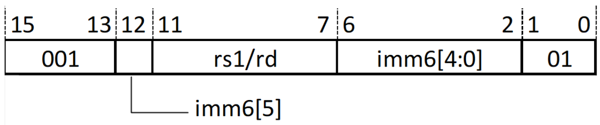
异常：

无

说明：

rs1 = rd != 0

指令格式：



17.5.4　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：

15 13	12 5	4 2	1 0
000	nzuimm8[3:2|7:4|0|1]	rd	00

17.5.5 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：

15 13	12	11 7	6 2	1 0
011		00010		01
	nzimm6[5]		nzimm6[0|2|4:3|1]	

17.5.6 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

17.5.7 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

17.5.8 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：

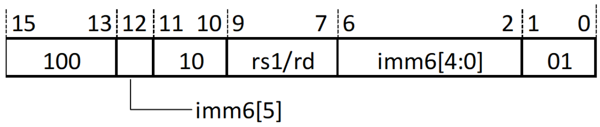
rs1 = rd

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：



17.5.9 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：

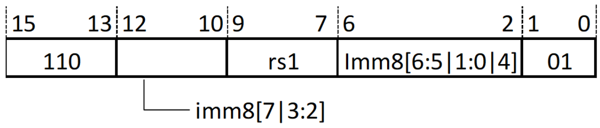
rs1 编码代表寄存器如下:

- 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:



17.5.10 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:

15 13	12 10	9 7	6 2	1 0
111		rs1	imm[6:5|1:0|4]	01

imm8[7|3:2]

17.5.11 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：

15 13	12	11 7	6 2	1 0
100	1	00000	00000	10

17.5.12 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：

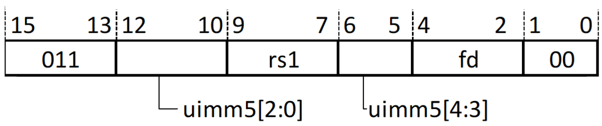
- 000: x8
- 001: x9
- 010: x10

- 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:



17.5.13 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:

15 13	12	11 7	6 2	1 0
001		fd	uimm6[1:0|5:3]	10

uimm6[2]

17.5.14 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：

15　13	12　10	9　7	6　5	4　2	1　0
101		rs1		fs2	00
	uimm5[2:0]		uimm5[4:3]		

17.5.15　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：

15　13	12　7	6　2	1　0
101	uimm6[2:0|5:3]	fs2	10

17.5.16　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

17.5.17 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

17.5.18 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

17.5.19 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：

15 13	12 10	9 7	6 5	4 2	1 0
011		rs1		rd	00
	uimm5[2:0]		uimm5[4:3]		

17.5.20 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：

15 13	12	11 7	6 2	1 0
011	uimm6[2]	rd	uimm6[1:0|5:3]	10

17.5.21 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：

15 13	12	11 7	6 2	1 0
010	imm[5]	rd	imm[4:0]	01

17.5.22 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：

15 13	12	11 7	6 2	1 0
011		rd	nzimm6[4:0]	01

nzimm6[5]

17.5.23 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

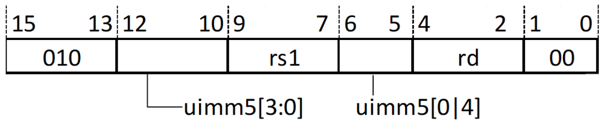
说明：

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：



17.5.24　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

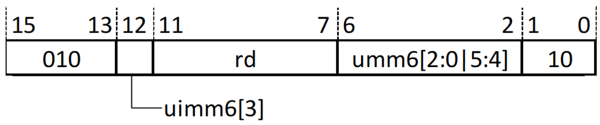
异常：

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：



17.5.25　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：

15–13	12	11–7	6–2	1–0
100	0	rd	rs2	10

17.5.26 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：

15–13	12	11–7	6–2	1–0
000	0	00000	00000	01

17.5.27 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

17.5.28　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:

15 13	12 10	9 7	6 5	4 2	1 0
111		rs1		rd	00
	uimm5[2:0]		uimm5[4:3]		

17.5.29 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:

15 13	12 7	6 2	1 0
111	uimm6[2:0|5:3]	rs2	10

17.5.30 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：

15　13	12	11　7	6　2	1　0
000		rs1/rd	nzuimm6[4:0]	10

nzuimm6[5]

17.5.31　C.SRAI——立即数算数右移指令

语法：

c.srai 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：

15　13	12	11　10	9　7	6　2	1　0
100		01	rs1/rd	nzuimm6[4:0]	01

nzuimm6[5]

17.5.32 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:

15 13	12	11 10	9 7	6 2	1 0
100	nzuimm6[5]	00	rs1/rd	nzuimm6[4:0]	01

17.5.33 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：

15 13	12 10	9 7	6 5	4 2	1 0
110		rs1		rs2	00
	uimm5[3:1]		uimm5[0|4]		

17.5.34 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

17.5.35 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

17.5.36 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

17.5.37　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

17.6 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 17.1: 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
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
csrr rd, csr	csrrs rd, csr, x0	读控制寄存器中对应比特
csrs csr, rs	csrrs x0, csr, rs	置位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

续下页

表 17.1 - 接上页

伪指令	基础指令	含义
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
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

续下页

表 17.1 - 接上页

伪指令	基础指令	含义
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

18 附录 B 玄铁扩展指令术语

C920V3 除了支持 RV64GCV 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、Cache 指令子集、多核同步指令子集、半精度浮点指令集、玄铁协处理器扩展指令。这些扩展指令集中，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、AI 扩展指令仅在 mxstatus.XUANTIEISAEE == 1、mxstatus.COPINSTEE == 0 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、mxstatus.COPINSTEE == 0 或 mxstatus.XUANTIEISAEE == 0、mxstatus.COPINSTEE == 1 时可以正常执行；
- 玄铁协处理器扩展指令需要在 mxstatus.XUANTIEISAEE == 0、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*。

备注

- mxstatus.COPINSTEE 位是协处理器指令的软件控制位，与 mxstatus.XUANTIEISAEE 互斥，硬件阻止同时设置为 1 的行为。如果软件试图同时设为 1，原有值保持不变。
- 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。

18.1 附录 B-1 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。

18.1.1 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

18.1.2 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

18.1.3 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

18.1.4 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

18.1.5 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：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00100	01	rs2	rs1	001	rd	0001011

18.1.6 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：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00101	01	rs2	rs1	001	rd	0001011

18.1.7 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：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00100	11	rs2	rs1	001	rd	0001011

18.1.8 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

18.1.9　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

18.1.10　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：

31　26	25　20	19　15	14　12	11　7	6　0
000100	imm6	rs1	001	rd	0001011

18.1.11　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：

31　25	24　20	19　15	14　12	11　7	6　0
0001010	imm5	rs1	001	rd	0001011

18.2　附录 B-2 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。

18.2.1　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：

31 26	25 20	19 15	14 12	11 7	6 0
imm1	imm2	rs1	010	rd	0001011

18.2.2 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：

31 26	25 20	19 15	14 12	11 7	6 0
imm1	imm2	rs1	011	rd	0001011

18.2.3 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

18.2.4 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

18.2.5 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

18.2.6 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

18.2.7 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

18.2.8 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

18.3 附录 B-3 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。

18.3.1 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

18.3.2 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

18.3.3 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

18.3.4 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

18.3.5 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

18.3.6 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

18.3.7 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

18.3.8 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

18.3.9 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

18.3.10 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

18.3.11 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

18.3.12 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

18.3.13 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：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
11111	imm2	rd2	rs1	100	rd1	0001011

18.3.14 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：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
01111	imm2	imm5	rs1	100	rd	0001011

18.3.15 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

18.3.16 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

18.3.17 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

18.3.18　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

18.3.19　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

18.3.20　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

18.3.21　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

18.3.22　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

18.3.23 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

18.3.24 LRHU—寄存器移位无符号扩展半字加载指令

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

18.3.25 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

18.3.26 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

18.3.27 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

18.3.28 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

18.3.29 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

18.3.30　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

18.3.31　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

18.3.32 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

18.3.33 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

18.3.34 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

18.3.35 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

18.3.36 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

18.3.37 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

18.3.38 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

18.3.39 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

18.3.40 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

18.3.41 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

18.3.42 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

18.3.43　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

18.3.44　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

18.3.45 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

18.3.46 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：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00101	imm2	imm5	rs1	101	rs2	0001011

18.3.47 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:

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00000	imm2	imm5	rs1	101	rd	0001011

18.3.48 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:

31 27	26 25	24 20	19 15	14 12	11 7	6 0
01100	imm2	rs2	rs1	101	rd	0001011

18.3.49 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：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00100	imm2	rs2	rs1	101	rd	0001011

18.3.50 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：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
01000	imm2	rs2	rs1	101	rd	0001011

18.3.51 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

18.3.52 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

18.3.53 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

18.3.54 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

18.3.55 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：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
01011	imm2	imm5	rs1	101	rs2	0001011

18.3.56 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：

31　　27	26　25	24　　20	19　　15	14　12	11　　7	6　　0
01001	imm2	imm5	rs1	101	rs2	0001011

18.3.57　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

18.4　附录 B-4 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。

18.4.1　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 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

18.4.2 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 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

18.4.3 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 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

18.4.4 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：

C920V3 dache 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 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

18.4.5 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 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	00111	rs1	000	00000	0001011

18.4.6 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 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

18.4.7 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 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

18.4.8 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 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	00101	rs1	000	00000	0001011

18.4.9 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 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

18.4.10 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 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

18.4.11 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：

C920V3 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 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

18.4.12 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 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

18.4.13 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 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

18.4.14 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 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

18.4.15 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 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

18.4.16 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 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

18.4.17 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 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

18.4.18　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:

C920V3 dache 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 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 18.1: DCACHE.CSW

18.5　附录 B-5 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。

18.5.1 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 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

18.5.2 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 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

18.5.3 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 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

18.5.4　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 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

18.6　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。

18.6.1 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

18.6.2 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

18.6.3　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

18.6.4 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

18.6.5 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:

31 25	24 20	19 15	14 12	11 7	6 0
0100010	00000	fs1	rm	fd	1010011

18.6.6 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

18.6.7　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

18.6.8　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:

31 25	24 20	19 15	14 12	11 7	6 0
1100010	00010	fs1	rm	rd	1010011

18.6.9 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

18.6.10 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

18.6.11 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

18.6.12　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

18.6.13 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

18.6.14 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

18.6.15 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

18.6.16 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：

31 20	19 15	14 12	11 7	6 0
imm12[11:0]	rs1	001	rd	0000111

18.6.17 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：

31 25	24 20	19 15	14 12	11 7	6 0
1010010	fs2	fs1	001	rd	1010011

18.6.18 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

18.6.19 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

18.6.20 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

18.6.21 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：

31–27	26–25	24–20	19–15	14–12	11–7	6–0
fs3	10	fs2	fs1	rm	fd	1000111

18.6.22 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:

31–25	24–20	19–15	14–12	11–7	6–0
0001010	fs2	fs1	rm	fd	1010011

18.6.23 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：

31 25	24 20	19 15	14 12	11 7	6 0
1111010	00000	rs1	000	fd	1010011

18.6.24 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：

31 25	24 20	19 15	14 12	11 7	6 0
1110010	00000	fs1	000	rd	1010011

18.6.25 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：

31　27	26　25	24　20	19　15	14　12	11　7	6　0
fs3	10	fs2	fs1	rm	fd	1001111

18.6.26 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：

31　27	26　25	24　20	19　15	14　12	11　7	6　0
fs3	10	fs2	fs1	rm	fd	1001011

18.6.27 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

18.6.28 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

18.6.29 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

18.6.30 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

18.6.31 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：

31 25	24 20	19 15	14 12	11 7	6 0
0101110	00000	fs1	rm	fd	1010011

18.6.32 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：

31 25	24 20	19 15	14 12	11 7	6 0
0000110	fs2	fs1	rm	fd	1010011

18.7 附录 B-7 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 custom1 域。支持功能如下：

- 支持部分整型指令扩展
- 支持部分 vector 类型指令扩展
- 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 4 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 custom1 域，均为 [6:0]:0101011
- 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：

18.7.1 整型指令

18.7.1.1 cpx0

语法：

cpx0 index，rs1，imm

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

说明：

source 来源 rs1、imm，其中 imm 数据共 10 bit，高 5 bit 是 imm1，低 5 bit 是 imm2，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	imm1	rs1	0	0	0	imm2	0	1	0	1	0	1	1

18.7.1.2　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	0	1	0	1	0	1	1

18.7.1.3　cpx2

语法：

cpx2 index， rd， rs1， imm

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

说明：

source 来源 rs1， imm， 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	uimm5	rs1	0	0	1	rd	0	1	0	1	0	1	1

18.7.1.4　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	0	1	0	1	0	1	1

18.7.1.5 cpx4

语法：

cpx4 index， rs1， rs2， imm

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， imm， 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	uimm5	0	1	0	1	0	1	1

18.7.1.6 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	0	1	0	1	0	1	1

18.7.1.7 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：

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	1	rd	0	1	0	1	0	1	1

18.7.1.8 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：

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	1	0	0	rs2	rs1	0	0	0	rd	0	1	0	1	0	1	1

18.7.1.9 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：

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	1	rd	0	1	0	1	0	1	1

18.7.1.10 cpx9

语法：

cpx9 index，rd，rs1，imm

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，imm，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：

31 30	29 20	19 15	14	13	12	11 7	6	5	4	3	2	1	0
index	imm10	rs1	1	1	1	rd	0	1	0	1	0	1	1

18.7.1.11 cpx10

语法：

cpx10 index，rd，imm

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

说明：

source 来源 imm，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：

31 30	29	28	27	26	25	24 15	14	13	12	11 7	6	5	4	3	2	1	0
index	0	0	1	0	1	imm	0	0	1	rd	0	1	0	1	0	1	1

18.7.2 vector 类型指令

vector 指令默认按照 lmul=1 且作为 whole register 进行计算，即 vl 不起作用，sew、rm 信息通过协处理器接口 req_hint 域伴随指令传递出去，用户可以使用该信息做判断。

18.7.2.1 vcpx0

语法：

vcpx0 index， vs2, vm

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

说明：

source 来源 vs2， vm， 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

该指令在 vs 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	vm	vs2	reserved	0	1	0	reserved	0	1	0	1	0	1	1

18.7.2.2 vcpx1

语法：

vcpx1 index， vd, vs2, vm

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

说明：

source 来源 vs2, vd， vm，该指令需要回写 vd 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 vs 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	vm	vs2	reserved	0	1	1	vd	0	1	0	1	0	1	1

18.7.2.3 vcpx2

语法：

vcpx2 index， vs2, imm, vm

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

说明：

source 来源 vs2， vm， imm，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

该指令在 vs 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	vm	vs2	uimm5	0	1	1	vd	0	1	0	1	0	1	1

18.7.2.4 vcpx3

语法：

vcpx3 index， vd, vs2, imm, vm

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

说明：

source 来源 vs2, vd，vm，imm，该指令需要回写 vd 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 vs 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	vm	vs2	uimm5	0	1	1	vd	0	1	0	1	0	1	1

18.7.2.5　vcpx4

语法：

vcpx4 index，vd, vs2, vs1, vm

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

说明：

source 来源 vs1，vs2，vd，vm，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

该指令在 vs 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	1	0	vm	vs2	vs1	0	1	0	vd	0	1	0	1	0	1	1

18.7.2.6　vcpx5

语法：

vcpx5 index，vd, vs2, vs1, vm

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

说明：

source 来源 vs1，vs2，vd，vm，该指令需要回写 vd 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 vs 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	1	0	vm	vs2	vs1	0	1	1	vd	0	1	0	1	0	1	1

18.7.2.7 vcpx6

语法：

vcpx6 index， vd, vs2, rs1, vm

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

说明：

source 来源 rs1， vs2， vd， vm， 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 vs 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	1	1	vm	vs2	rs1	0	1	0	vd	0	1	0	1	0	1	1

18.7.2.8 vcpx7

语法：

vcpx7 index， vd, vs2, rs1, vm

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

说明：

source 来源 rs1， vs2， vd， vm， 该指令需要回写 vd 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 vs 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	1	1	vm	vs2	rs1	0	1	1	vd	0	1	0	1	0	1	1

18.7.2.9　vcpx8

语法：

vcpx8 index， vd, vs2, imm, vm

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

说明：

source 来源 vs2， vd， vm， imm， 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 vs 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	1	0	0	vm	vs2	uimm5	0	1	0	vd	0	1	0	1	0	1	1

18.7.2.10　vcpx9

语法：

vcpx9 index， vd, vs2, fs1, vm

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

说明：

source 来源 vs2， vd， vm， 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 vs 和 fs 均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	1	0	1	vm	vs2	fs1	0	1	0	vd	0	1	0	1	0	1	1

18.7.2.11　vcpx10

语法：

vcpx10 index， vd, vs2, fs1, vm

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

说明：

source 来源 vs2，vd，vm，fs1，该指令需要回写 vd 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 vs 和 fs 均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	1	0	1	vm	vs2	fs1	0	1	1	vd	0	1	0	1	0	1	1

18.7.3 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。

18.7.3.1 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	0	1	0	1	0	1	1

18.7.3.2 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	0	1	0	1	0	1	1

18.7.3.3 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	0	1	0	1	0	1	1

18.7.3.4 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	0	1	0	1	0	1	1

18.7.3.5 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：

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	fs2	fs1	1	0	0	fd	0	1	0	1	0	1	1

18.7.3.6 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：

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	fs2	fs1	1	0	1	fd	0	1	0	1	0	1	1

18.7.3.7 fcpx6

语法：

fcpx6 index， fd， fs2, imm

执行权限：

M mode/S mode/U mode

异常：

非法指令异常

说明：

source 来源 fs2, imm 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	imm	1	0	1	fd	0	1	0	1	0	1	1

19 附录 C 系统控制寄存器

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。

19.1　附录 C-1 RISC-V 标准机器模式控制寄存器

19.1.1　机器模式信息寄存器组

19.1.1.1　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。

19.1.1.2　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。

19.1.1.3　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。其中 MIMPID[27:24] 为主版本号，MIMPID[23:18] 为子版本号，MIMPID[17:12] 为补丁版本号。

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。

19.1.1.4　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。

19.1.1.5 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。

19.1.2 机器模式异常配置寄存器组

19.1.2.1 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。

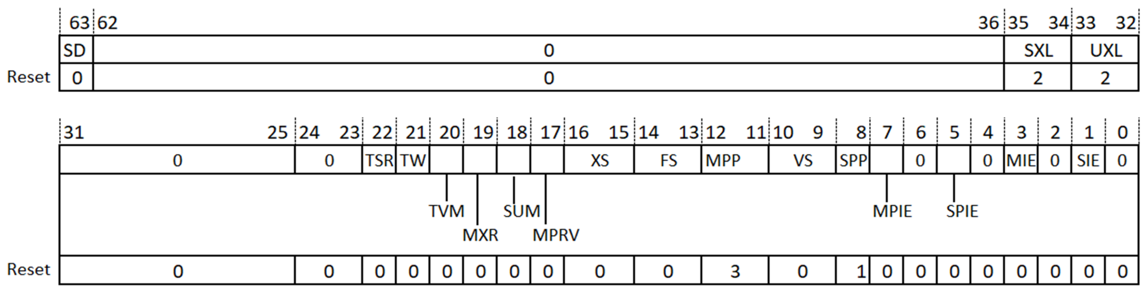


图 19.1: 机器模式处理器状态寄存器（MSTATUS）

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 1'b0 时，表示处理器进入异常服务程序前处于用户模式；

当 SPP 为 1'b1 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 1'b1

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：

C920V3 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。

19.1.2.2　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。

C920V3 支持的指令集架构为 RV64GC，对应的 MISA 寄存器复位值为 0x8000000000b4112f。具体的赋值规则请参考 RISCV 官方文档《riscv-privileged》。

C920V3 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。

19.1.2.3　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。

19.1.2.4 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。

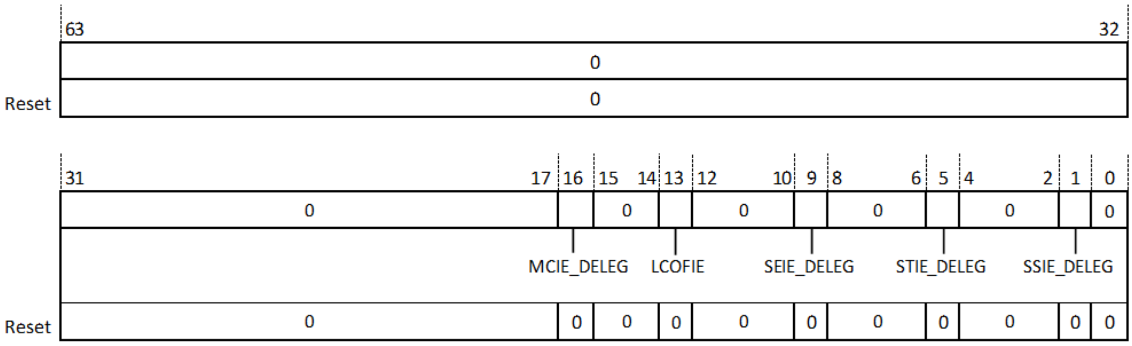


图 19.2: 机器模式中断降级控制寄存器（MIDELEG）

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。

MCIE_DELEG-机器模式 ECC 中断：

- 当 MCIE_DELEG=1 时，表明 ECC 中断可以在处理器被降级到超级用户模式下处理。
- 当 MCIE_DELEG=0 时，表示 ECC 中断只能在处理器切换到机器模式下被处理。

LCOFIE-性能监测单元事件计数器溢出中断：

- 当 LCOFIE=1 时，表明性能监测单元事件计数器溢出中断可以在处理器被降级到超级用户模式下处理。
- 当 LCOFIE=0 时，表明性能监测单元事件计数器溢出中断只能在处理器切换到机器模式下被处理。

SEIE_DELEG-超级用户模式外部中断：

- 当 SEIE_DELEG=1 时，表明超级用户模式外部中断可以在处理器被降级到超级用户模式下处理。
- 当 SEIE_DELEG=0 时，表明超级用户模式外部中断只能在处理器切换到机器模式下被处理。

STIE_DELEG-超级用户模式计时器中断：

- 当 STIE_DELEG=1 时，表明超级用户模式计时器中断可以在处理器被降级到超级用户模式下处理。
- 当 STIE_DELEG=0 时，表明超级用户模式计时器中断只能在处理器切换到机器模式下被处理。

SSIE_DELEG-超级用户模式软件中断：

- 当 SSIE_DELEG=1 时，表明超级用户模式软件中断可以在处理器被降级到超级用户模式下处理。
- 当 SSIE_DELEG=0 时，表明超级用户模式软件中断只能在处理器切换到机器模式下被处理。

19.1.2.5 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。

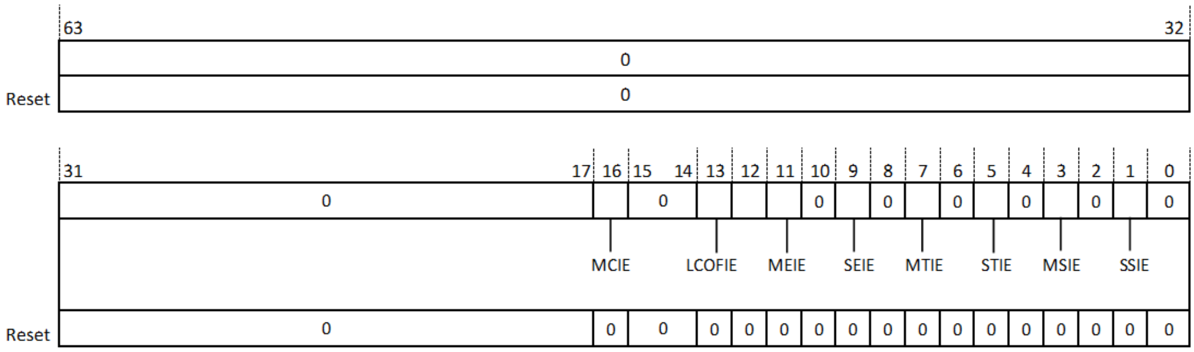


图 19.3: 机器模式中断使能控制寄存器（MIE）

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 中断使能位：

- 当 MCIE 为 0 时，机器模式 ECC 中断无效。
- 当 MCIE 为 1 时，机器模式 ECC 中断有效。

19.1.2.6 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。

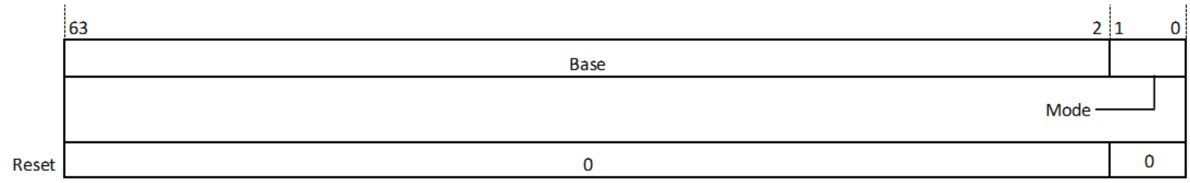


图 19.4: 机器模式向量基址寄存器（MTVEC）

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。

19.1.2.7 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）*。

19.1.3 机器模式异常处理寄存器组

19.1.3.1 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。

19.1.3.2 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）。C920V3 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。

19.1.3.3 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。

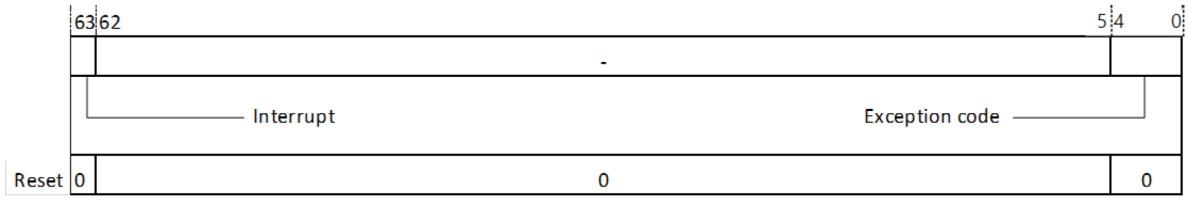


图 19.5: 机器模式异常事件向量寄存器（MCAUSE）

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。

19.1.3.4 机器模式异常事件信息寄存器（MTVAL）

机器模式异常事件信息寄存器（MTVAL）用于保存触发异常的异常事件的原因，例如出异常的地址、指令码等，用于在异常服务程序中处理对应事件。

该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。该寄存器位宽为 MXLEN。

19.1.3.5 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 寄存器可以触发对应的中断，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。

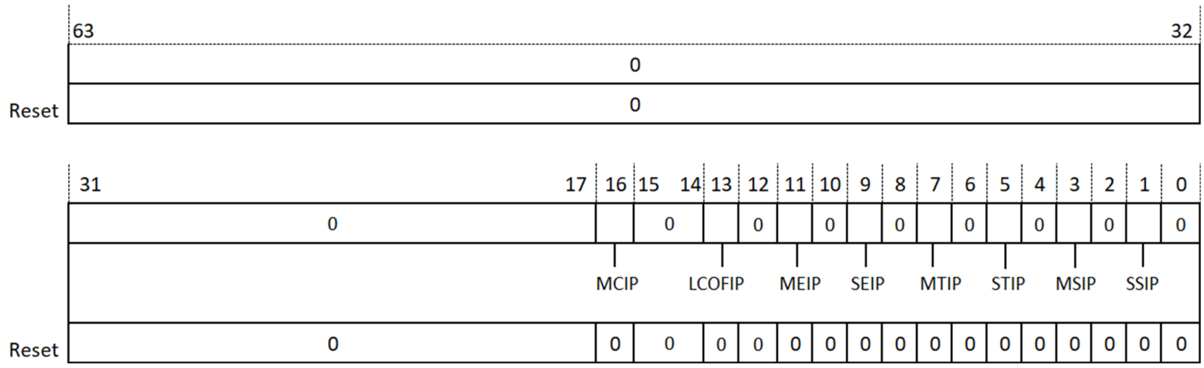


图 19.6: 机器模式中断等待状态寄存器（MIP）

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 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 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 中断。

19.1.4　机器模式环境配置寄存器组

19.1.4.1　机器模式环境配置寄存器（MENVCFG）

	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 19.7: 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。

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=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/S-TIMECMPL）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=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 C920V3 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。

19.1.4.2 机器模式安全配置寄存器（MSECCFG）

	XLEN-1 … 3	2	1	0
	-	RLB		MML
	XLEN-3		MMWP	
Reset	0	1	0	0

图 19.8: 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 19.1 所示:

表 19.1: 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)	

续下页

表 19.1 - 接上页

Bits on pmpcfg register				Result	
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) 。

19.1.5　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。

19.1.5.1　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）*。

19.1.5.2　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）*。

19.1.6 机器模式计时和计数器寄存器组

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。

19.1.6.1 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*。

19.1.6.2 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*。

19.1.6.3 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*。

19.1.7 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。

19.1.7.1 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。

具体信息请参考*机器模式计数禁止寄存器（mcountinhibit）*。

19.1.7.2 机器模式性能监测事件选择寄存器（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。C920V3 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 *机器模式性能监测事件选择寄存器* 。

19.1.8 调试/追踪寄存器组/Trace 寄存器组（与调试模式共享）

19.1.8.1 调试/追踪触发选择寄存器（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。

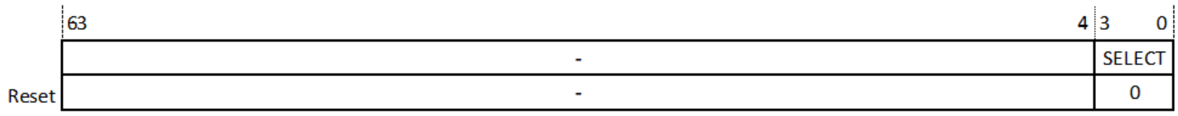


图 19.9: 调试/追踪触发选择寄存器（TSELECT）

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。

19.1.8.2 调试/追踪触发器数据寄存器 1（TDATA1）

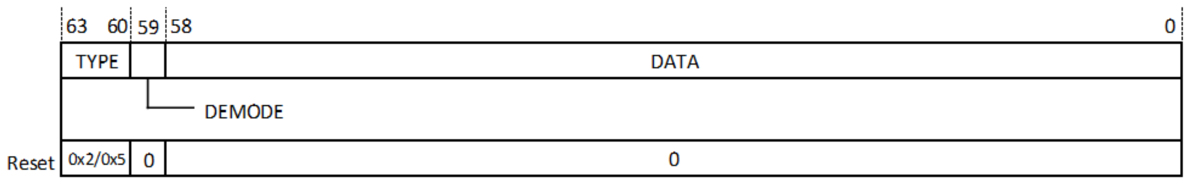


图 19.10: 调试/追踪触发器数据寄存器 1（TDATA1）

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。

C920V3 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/追踪触发器数据寄存器 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/追踪触发器数据寄存器；
- 当 DEMODE 为 1 时，仅调试模式可以写入调试/追踪触发器数据寄存器。

DATA - 调试/追踪触发器数据寄存器 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 External Debug Support version 0.13.2 中 5.2 小节）。

19.1.8.3 调试/追踪触发器数据寄存器 2（TDATA2）

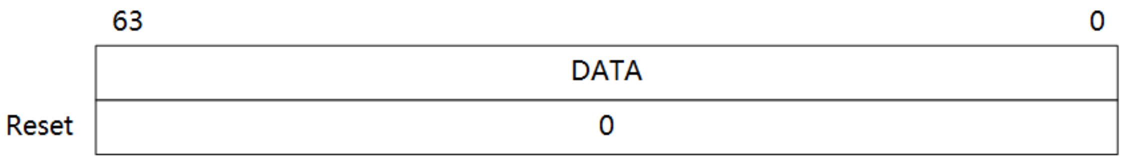


图 19.11: 调试/追踪触发器数据寄存器 2（TDATA2）

DATA - 调试/追踪触发器数据寄存器 2 数据

用于设置触发值，具体含义由 TDATA1 中 TYPE 字段决定。

19.1.8.4 调试/追踪触发器数据寄存器 3（TDATA3）

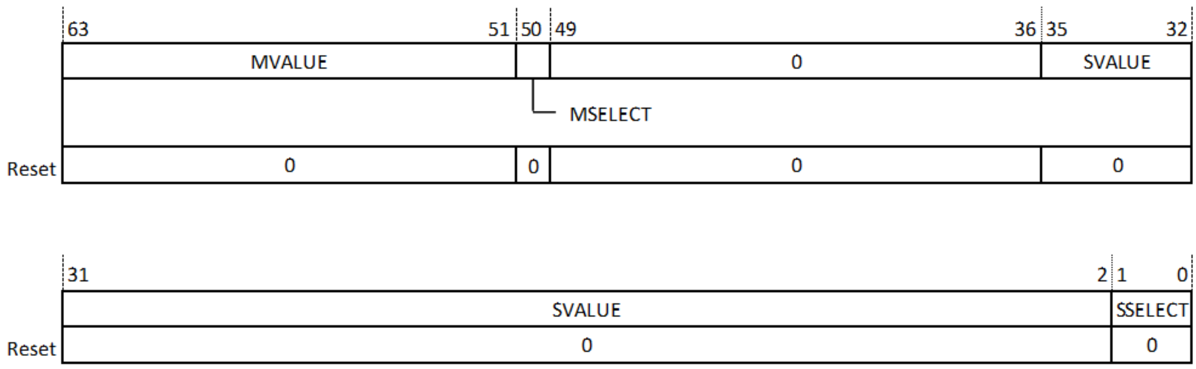


图 19.12: 调试/追踪触发器数据寄存器 3（TDATA3）

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。

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。

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。

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。

19.1.8.5 调试/追踪触发器信息寄存器（TINFO）

	63–16	15–0
	0	INFO
Reset	0	0x100/0x111000

图 19.13: 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。

C920V3 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。

19.1.8.6 调试/追踪触发器控制寄存器（TCONTROL）

	63–32
	0
Reset	0

	31–9	8–7	6–4	3	2–0
		MPTE	0	MTE	0
Reset	0	0	0	0	0

图 19.14: 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。

19.1.8.7 机器模式内容寄存器（MCONTEXT）

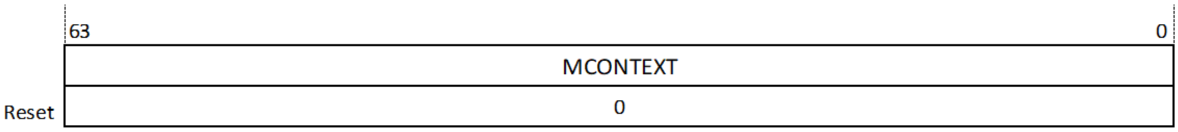


图 19.15: 机器模式内容寄存器（MCONTEXT）

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/追踪触发器数据寄存器 3（TDATA3）中 MSELECT 与 MVALUE，可以控制触发仅在特定机器模式 context 下触发。

19.1.9 调试模式寄存器组/Trace 寄存器组

19.1.9.1 调试模式控制与状态寄存器（DCSR）

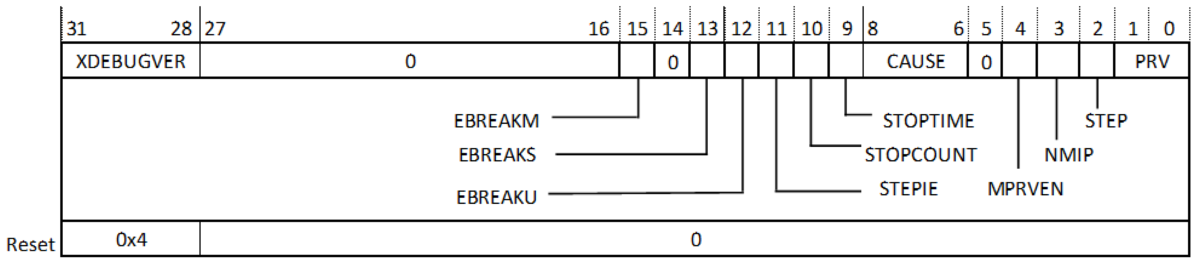


图 19.16: 调试模式控制与状态寄存器（DCSR）

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 和 MPP 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

19.1.9.2　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。

19.1.9.3　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。

19.1.9.4　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。

19.2　附录 C-2 RISC-V 标准超级用户模式控制寄存器

19.2.1　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。

19.2.1.1　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机器模式和超级用户模式可读写，用户模式不可访问，用户模式访问将导致非法指令异常。

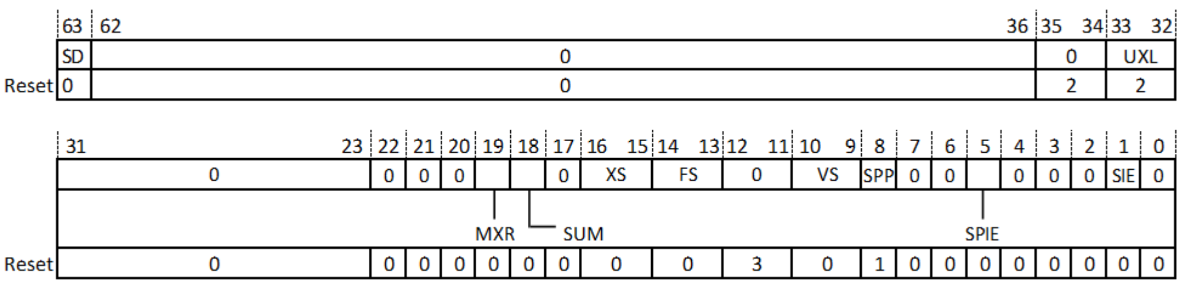


图 19.17: 超级用户模式处理器状态寄存器（SSTATUS）

具体信息请参考*机器模式处理器状态寄存器（MSTATUS）*。

19.2.1.2　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。

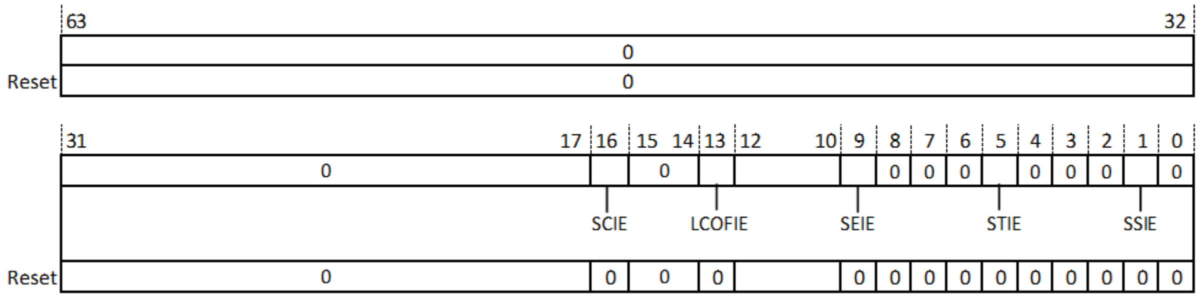


图 19.18: 超级用户模式中断使能控制寄存器（SIE）

具体信息请参考*机器模式中断使能控制寄存器（MIE）*。

19.2.1.3　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。

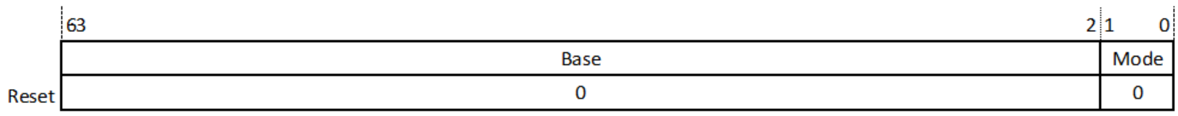


图 19.19: 超级用户模式向量基址寄存器（STVEC）

具体信息请参考*机器模式向量基址寄存器（MTVEC）*。

19.2.1.4　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）*。

19.2.1.5　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）*。

19.2.2 超级用户模式环境配置寄存器组

19.2.2.1 超级用户模式环境配置寄存器（SENVCFG）

	63 … 8	7	6	5 … 4	3 … 1	0
	0	CBZE	CBCFE	CBIE	0	FIOM
Reset	0	0	0	0	0	0

图 19.20: 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。

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

无论该位为何值，对 IO 的 fence 同步均会包括 RW 同步。

注：玄铁 920V3 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。

19.2.3 超级用户模式异常处理寄存器组

19.2.3.1 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。

19.2.3.2 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）。C920V3 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。

19.2.3.3 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。

19.2.3.4 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。

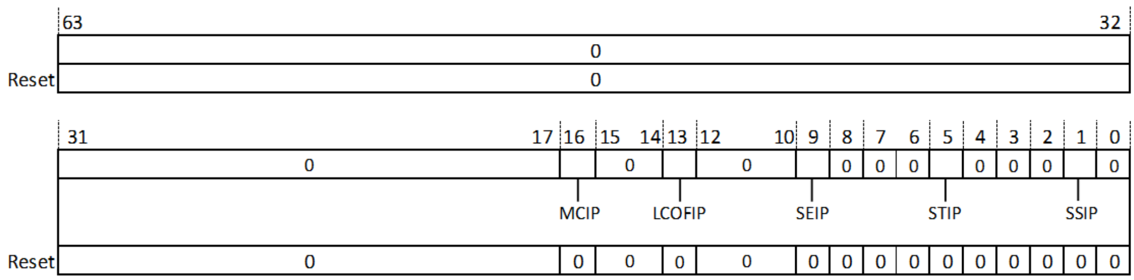


图 19.21: 超级用户模式中断等待状态寄存器（SIP）

具体信息请参考*机器模式中断等待状态寄存器（MIP）*。

19.2.4 超级用户模式地址保护寄存器组

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。

19.2.4.1 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)* 。

19.2.5 超级用户模式调试寄存器组

19.2.5.1 超级用户模式内容寄存器（SCONTEXT）

	63 … 0
	SCONTEXT
Reset	0

图 19.22: 超级用户模式内容寄存器（SCONTEXT）

SCONTEXT - 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/追踪触发器数据寄存器 3（TDATD3）中 SSELECT 与 SVALUE，可以控制触发仅在特定的超级用户模式 context 下触发 trigger。

19.2.6 超级用户模式计时和计数寄存器组

19.2.6.1 超级用户模式计时器中断比较值寄存器（STIMECMP）

	63 … 0
	STIMECMP
Reset	0xFFFF_FFFF_FFFF_FFFF

图 19.23: 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。

19.3 附录 C-3 RISC-V 标准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。

19.3.1 用户模式浮点控制寄存器组

19.3.1.1 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）*。

19.3.1.2 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）*。

19.3.1.3 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。

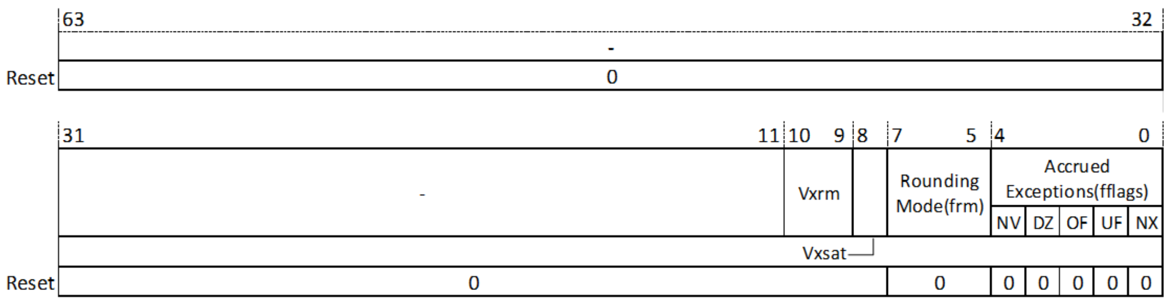


图 19.24: 浮点控制状态寄存器（FCSR）

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。

19.3.2　用户模式计时和计数寄存器组

19.3.2.1　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*。

19.3.2.2　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*。

19.3.2.3　用户模式退休指令计数器（INSTRET）

用户模式退休指令计数器（INSTRET）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*。

19.3.2.4 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*。

19.3.3 矢量扩展寄存器组

19.3.3.1 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。

19.3.3.2 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。

19.3.3.3 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 19.25: 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。

19.3.3.4 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。

19.3.3.5 矢量控制和状态寄存器（VCSR）

	63–32
	Reserved
Reset	0

	31–3	2–1	0
	Reserved	VXRM	VXSAT
Reset	0	0	0

图 19.26: 矢量控制和状态寄存器（VCSR）

VXRM/VXSAT 信息分别为对应寄存器的镜像

19.3.3.6 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 19.27: 矢量数据类型寄存器（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 C920V3 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），C920V3 支持的矢量元素位宽如 表 19.2 所示。

表 19.2: 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，C920V3 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 19.3 所示。

表 19.3: 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

19.3.3.7 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。

C920V3 的矢量位宽为 128 位（即 VLEN=128），所以 VLENB = 128 / 8 = 16。

19.4 附录 C-4 C920V3 扩展机器模式控制寄存器

19.4.1 机器模式处理器控制和状态扩展寄存器组

19.4.1.1 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 C920V3 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。

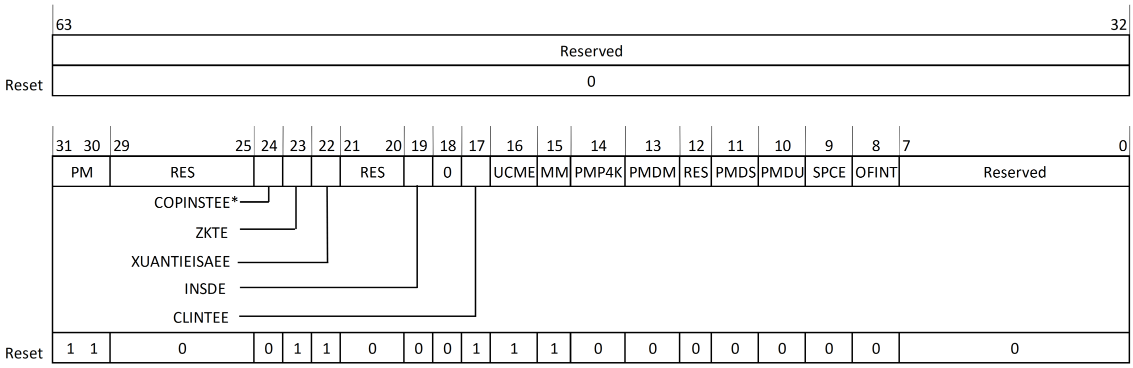


图 19.28: 机器模式扩展状态寄存器（MXSTATUS）

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-超级用户模式 Cache Partition 控制使能:

当 SPCE=0 时，超级用户模式下不能配置 L2 Cache 的 partition。

当 SPCE=1 时，超级用户模式下可以配置 L2 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：

C920V3 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。（C920V3 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。

INSDE-关闭 Icache snoop Dcache 功能：

当 INSDE 为 0 时，Icache 缺失后，会 snoop Dcache。

当 INSDE 为 1 时，Icache 缺失后，不会 snoop Dcache。

XUANTIEISAEE-使能玄铁扩展指令集：

当 XUANTIEISAEE 为 0 时，使用 C920V3 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 C920V3 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。

ZKTE-zkt 使能开关：

当 ZKTE 为 0 时，zkt 相关指令执行延时不固定。

当 ZKTE 为 1 时，zkt 相关指令执行延时固定。

C920V3 中固定为 1。

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

- 该位是协处理器指令的软件控制位，与 XUANTIEISAEE 互斥，硬件阻止同时设置为 1 的行为。如果软件试图同时设为 1，原有值保持不变。
- 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）

19.4.1.2 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。

	63–32
	Reserved
Reset	0

	31–19	18–16	15–13	12	11–9	8	7	6	5	4	3	2	1	0
	Reserved	SCK	Reserved	L0BTB	Reserved	WBR	IBPE	BTB	BPE	RS	WB	WA	DE	IE
Reset	0	0	0	0	0	0	0	0	0	0	1	0	0	0

图 19.29: 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。

C920V3 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。

C920V3 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。

SCK-系统和处理器的时钟比：

C920V3 的 SCK 域固定为 0，不表示时钟比信息。

19.4.1.3　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。

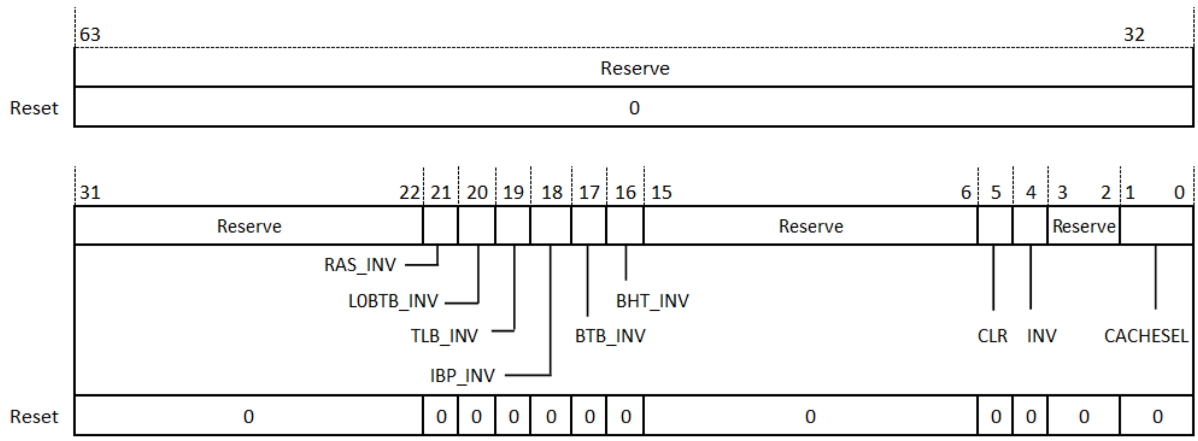


图 19.30: 机器模式硬件操作寄存器（MCOR）

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。

TLB_INV-inv tlb 设置位

当 TLB_INV=0 时，TLB 内的数据不进行无效化。

当 TLB_INV=1 时，TLB 内的数据进行无效化。

L0BTB_INV-L0_BTB 无效设置位：

当 L0_BTB=0 时，L0_BTB 的数据不进行无效化。

当 L0_BTB=1 时，L0_BTB 的数据进行无效化。

RAS_INV-RAS 无效设置位：

当 RAS_INV=0 时，RAS 的数据不进行无效化。

当 RAS_INV=1 时，RAS 的数据进行无效化。

19.4.1.4 机器模式 L2Cache 控制寄存器（MCCR2）

机器模式 L2 Cache 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。

位	63–33	32
字段	-	pae
Reset	0	0

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

图 19.31: 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 L2 Cache 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 L2 Cache 之间为 inclusive 包含性关系（C920V3 固定为 1）。

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，在 L1 CACHE 关闭时禁止分配 L2 CACHE。

当 L2EN=1 时，在 L1 CACHE 关闭时允许分配 L2 CACHE。

备注

若 L1 DCACHE 开启，且该位为 0，由于 L1 DCACHE 与 L2 CACHE 为完全包含性关系，所以分配 L1 的行为依然会分配 L2。建议在 L1 DCACHE 开启前使能该 bit。

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。

该 bit 为只读。

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。

该 bit 为只读。

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 3 条缓存行。

TPRF-L2Cache TLB 预取使能：

当 TPRF 为 0 时，L2 Cache TLB 预取功能关闭；

当 TPRF 为 1 时，L2 Cache TLB 预取功能开启。

pae-Partition 访问使能位：

当 pae 为 0 时，表示 L2 cache 不支持 partition 访问。

当 pae 为 1 时，表示 L2 cache 支持 partition 访问。

19.4.1.5 机器模式 L2 Cache ECC 控制寄存器（MCER2）

机器模式 L2 Cache ECC 控制寄存器（MCER2）用于对 L2 Cache ECC 进行配置。二级高速缓存支持可配置的 ECC，实现 1 比特错误可纠正、2 比特错误可检测的功能。当发现是 2 比特错误时，硬件自动设置 MCER2 寄存器的 ERR_VLD 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。

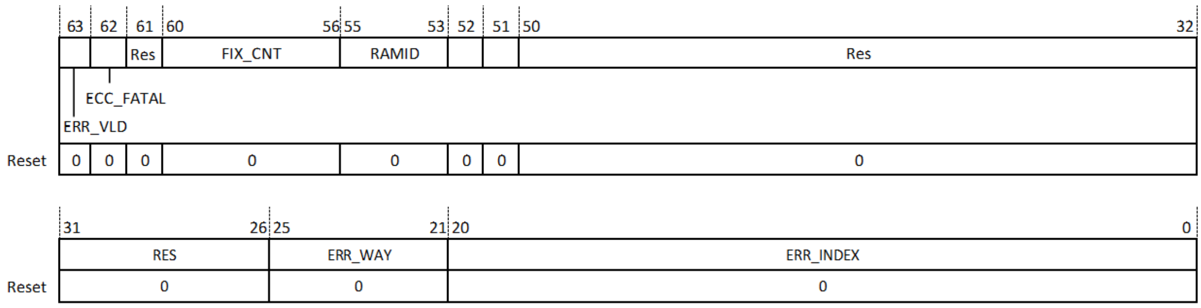


图 19.32: 机器模式 L2Cache ECC 控制寄存器（MCER2）

ERR_VLD-L2 CACHE 校验错误位：

当 ERR_VLD 为 0，L2 CACHE 无异常发生。

当 ERR_VLD 为 1，L2 CACHE 存在 ECC 错误或者奇偶校验错误。

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 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。

FIX_CNT[4:0]-已修复的 ECC Error 数量：

记录已修复的 ECC Error 数量。

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 CORE0；

ID=4：SNOOP FILTER CORE1；

ID=5：SNOOP FILTER CORE2；

ID=6：SNOOP FILTER CORE3；

ERR_WAY-L2 CACHE 校验错误的路位置信息：

记录 L2 CACHE 第一次出现 2 比特校验错误的路位置。

ERR_INDEX-L2 CACHE 校验错误的索引信息：

记录 L2 CACHE 第一次出现 2 比特校验错误的索引位置。

19.4.1.6 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。

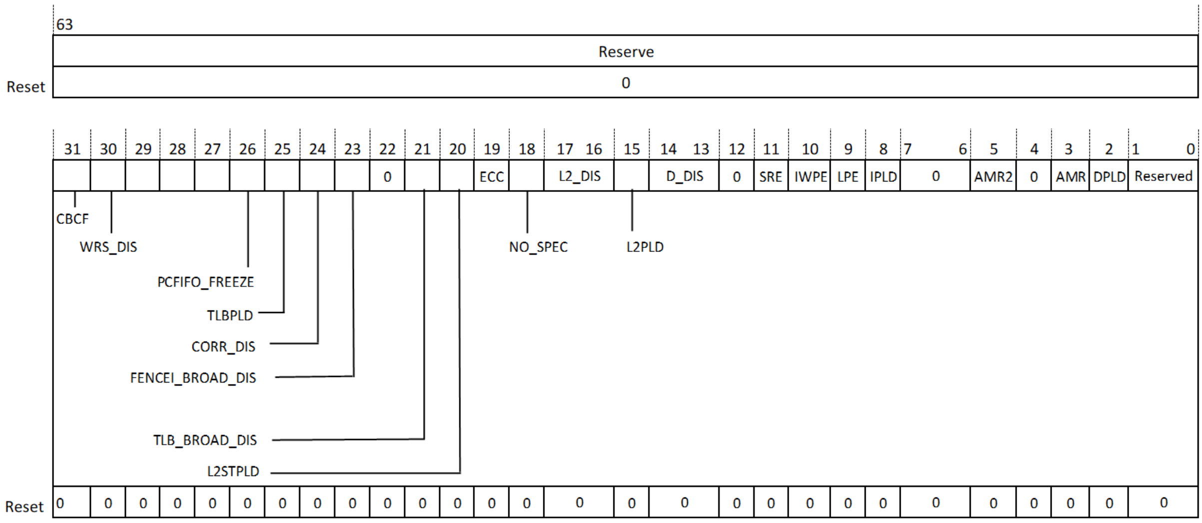


图 19.33: 机器模式隐式操作寄存器（MHINT）

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，在出现连续多条缓存行的存储操作时后续连续地址的存储操作不再写入 L1 Cache。

AMR2-L2Cache 写分配策略自动调整使能位

当 AMR2 为 0 时，写分配策略由访问地址的页面属性 WA 决定。

当 AMR2 为 1 时，在出现连续多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。

LPE-循环加速使能位：

当 LPE 为 0 时，循环加速关闭。

当 LPE 为 1 时，循环加速开启。

IWPE-ICACHE 路预测使能位：

当 IWPE 为 0 时，ICACHE 路预测关闭；

当 IWPE 为 1 时，ICACHE 路预测开启。

SRE-单退休模式位：

当 SRE 为 0 时，单退休模式关闭；

当 SRE 为 1 时，单退休模式开启。

D_DIS-DCACHE 预取缓存行数量：

当 D_DIS 为 0 时，预取 2 条缓存行。

当 D_DIS 为 1 时，预取 4 条缓存行。

当 D_DIS 为 2 时，预取 8 条缓存行。

当 D_DIS 为 3 时，预取 16 条缓存行。

默认为 0。

L2PLD-L2CACHE 预取使能位：

当 L2PLD 为 0 时，L2CACHE 预取关闭；

当 L2PLD 为 1 时，L2CACHE 预取开启。

L2_DIS-L2CACHE 预取缓存行数量：

当 L2_DIS 为 0 时，预取 8 条缓存行；

当 L2_DIS 为 1 时，预取 16 条缓存行；

当 L2_DIS 为 2 时，预取 32 条缓存行；

当 L2_DIS 为 3 时，预取 64 条缓存行；

L2Cache 的预取是在 L1 Cache 预取的基础上再次进行预取。

NO_SPEC-SPEC FAIL 预测功能使能位：

当 NO_SPEC 为 0 时，spec fail 预测功能关闭；

当 NO_SPEC 为 1 时，spec fail 预测功能开启。

ECC-L1 CACHE ECC 校验使能位：

当 ECC 为 0 时，L1 Cache 校验功能关闭；

当 ECC 为 1 时，L1 Cache 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。

CORR_DIS-LSU 内 RAR 乱序纠错方式

当 CORR_DIS 为 0 时，更保守的乱序 RAR 检测方式，只要出现 RAR 乱序时便启动纠错处理;

当 CORR_DIS 为 1 时，性能更优的乱序 RAR 检测方式，仅当 RAR 乱序出现数据错误时才启动纠错处理。

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；

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；

WRS_DIS-用于控制 WRS 指令正常执行或当做 NOP 执行:

当 WRS_DIS 为 0 时，WRS 指令正常执行

当 WRS_DIS 为 1 时，WRS 指令当作 nop 执行

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）。

受该位影响的指令包括：cbo.clean、dcache.call、dcache.cpa、dcache.cpal1、dcache.cva、dcache.cval1、dcache.csw。

19.4.1.7 机器模式复位寄存器（MRMR）

【注意】mrmr 寄存器在 C920V3（R1S4 及以上）中已经被删除，相应的功能也不复存在。软件仍然可以访问该寄存器，结果是读为零、写无效，不会触发异常。

机器模式复位寄存器用于多核启动时各个 C920V3 核心的复位释放使能。各个处理器核心共享一个 MRMR 寄存器，因此某个处理器核心可以通过设置 MRMR 将其它处理器核心从复位状态释放。

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。

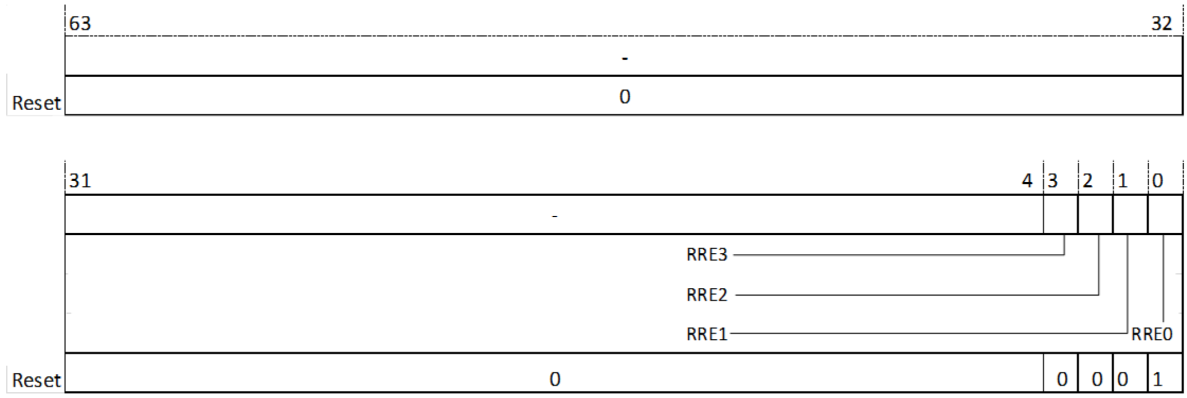


图 19.34: 机器模式复位寄存器（MRMR）

RRE3/2/1/0-复位释放使能位：

控制各个核心复位释放使能位，

当 RREx 为 0 时，对应 C920V3 核心处于复位状态。

当 RREx 为 1 时，对应 C920V3 核心复位释放。

19.4.1.8 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 C920V3 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。

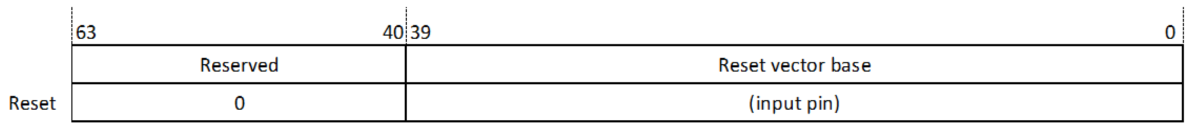


图 19.35: 机器模式复位向量基址寄存器（MRVBR）

Reset vector base-复位基址：

控制核心的复位基址。

19.4.1.9 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，软件可以通过向 ERR_VLD 位写 0 的方式清除 ERR_FATAL 比特并清除 16 号中断，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。

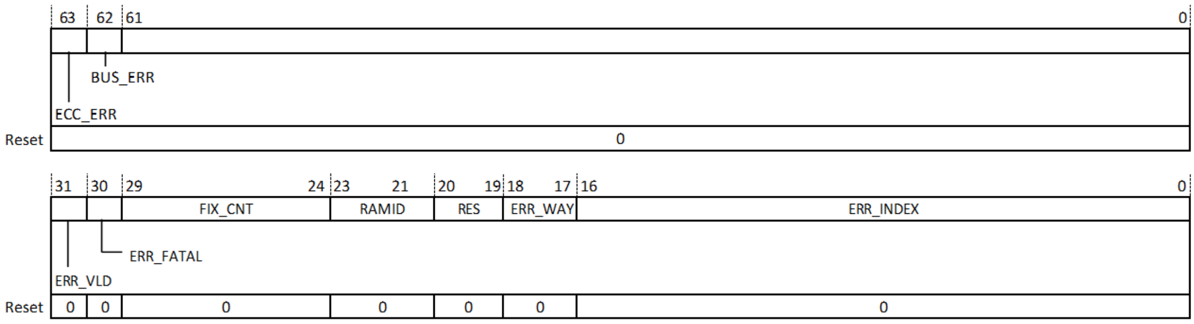


图 19.36: L1 Cache ECC 寄存器（MCER）

ECC_ERR-ECC 信息有效位，ERR_VLD 为高有效

当 ECC_err 为 0 时，L1 CACHE ECC 信息无效；

当 ECC_err 为 1 时，L1 CACHE ECC 信息有效

BUS_ERR-总线异常有效位

ERR_VLD 为高位时有效

当 bus_err 为 0 时，表示没有发生 ld 总线异常；

当 bus_err 为 1 时，表示发生 ld 总线异常。

ERR_VLD- ECC 信息有效位：

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 ECC_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- 出现 ECC 错误时的路位置：

记录出错 RAM 出现 ECC ERROR 时的路位置。发生 FATAL ERR 时，此域将不再更新。

ERR_INDEX- 出现 ECC 错误时的索引位置：

记录出错 RAM 出现 ECC ERROR 时的索引位置。发生 FATAL ERR 时，此域将不再更新。

19.4.1.10　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。详情信息请见*机器模式计数器写使能寄存器（mcounterwen）*。

19.4.2　机器模式扩展寄存器组 2

19.4.2.1　机器模式性能监测控制寄存器（MHPMCR）

M-mode 性能监测控制寄存器，此寄存器为玄铁自扩展寄存器。详细信息请见*机器模式性能监测控制寄存器（MHPMCR）*。

19.4.2.2　机器模式性能监控起始触发寄存器（MHPMSR）

M-mode 性能监控起始触发寄存器，此寄存器为玄铁自扩展寄存器。详情信息请见*起始触发寄存器*。

19.4.2.3　机器模式性能监控终止触发寄存器（MHPMER）

M-mode 性能监控终止触发寄存器，此寄存器为玄铁自扩展寄存器。详情信息请见*终止触发寄存器*。

19.4.2.4　机器模式侦听使能寄存器（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，只有 bit 0 有定义，其余是 Reserved。

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。

19.4.2.5　处理器 ZONE ID 寄存器（MZONEID）

	63 … 4	3 … 0
	Reserved	Zone ID
Reset	0	FIOM

图 19.37: 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。

19.4.2.6　处理器末级 Cache partition ID 寄存器（MLLCPID）

	63 … 3	2 … 0
	Reserved	PARTID
Reset	0	0

图 19.38: 处理器末级 Cache partition 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。

19.4.2.7　处理器 L2 Cache partition 访问配置寄存器（MLLWP）

65 … 56	55 … 48	47 … 40	39 … 32	31 … 24	23 … 16	15 … 8	7 … 0
group7	group6	group5	group4	group3	group2	group1	group0
ff	ff	ff	ff	ff	ff	ff	ff
RW	RW	RW	RW	RW	RW	RW	RW

图 19.39: 处理器 L2 Cache partition 访问配置寄存器（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。

19.4.2.8 机器模式 L1 Cache ECC 单比特错误物理地址寄存器（MSBEPA）

	63 … 40	39 … 32
	Reserved	SBE_PA
Reset	0	0

	31 … 0
	SBE_PA
Reset	0

图 19.40: 机器模式 L1 Cache ECC 单比特错误物理地址寄存器（MSBEPA）

SBE_PA - L1 Cache ECC 错误地址

当发生 DCACHE SBE 上报 L1 Cache ECC 中断时，记录产生该 SBE 错误的物理地址。

19.4.2.9 机器模式 L2 Cache ECC 单比特错误物理地址寄存器（MSBEPA2）

	63 … 40	39 … 32
	Reserved	SBE_PA
Reset	0	0

	31 … 0
	SBE_PA
Reset	0

图 19.41: 机器模式 L2 Cache ECC 单比特错误物理地址寄存器（MSBEPA）

SBE_PA - L2 Cache ECC 错误地址

L2 Cache ECC 错误地址，当发生 L2 Cache SBE 上报 L2Cache ECC 中断时，记录产生该 SBE 错误的物理地址。

19.4.3 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。

19.4.3.1 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。

	63 - 1	0
	-	R
Reset	0	0

图 19.42: 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。

19.4.3.2 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。

	63 - 33	32
	0	RID
Reset	0	0

	31 - 28	27 - 26	25 - 21	20 - 0
	RID	-	WAY	INDEX
Reset	0	0	0	0

图 19.43: 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 L2 Cache TAG RAM。

当 RID 为 5 时，表示访问的是 L2 Cache 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 L2 Cache TAG ECC RAM。

当 RID 为 11 时，表示访问的是 L2 Cache 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。

当 RID 为 22 时，表示访问的是 TLB TAG RAM。

当 RID 为 23 时，表示访问的是 TLB DATA RAM。

当 RID 为 24 时，表示访问的是 TLB TAG RAM ECC RAM。

当 RID 为 25 时，表示访问的是 TLB DATA RAM 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。在访问 TAG RAM 时，保证 INDEX[5:0] 值为 0。

19.4.3.3 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。

	63 … 0
	DATA
Reset	0

图 19.44: 机器模式 Cache 访问数据寄存器（MCDATA0/1）

表 19.4: 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
DCACHE ST TAG	CDATA0[39:14]: 26 bit tag CDATA0[13:12] : cindex[13:12] CDATA0[3:0]: {page share, dirty, share, vld}
DCACHE DATA	CDATA0~CDATA1: 128bit DATA
L2CACHE TAG	CDATA0[40]: PAGE_SHARE CDATA0[39:12]: TAG+INDEX CDATA0[7:4]: CP CDATA0[3:0]: {valid, share, dirty, pend}
L2CACHE DATA	CDATA0~CDATA1: 128bit DATA
ICACHE TAG ECC	CDATA0[0]: ECC
ICACHE DATA ECC	CDATA0[3:0]:ECC
DCACHE ST TAG ECC	CDATA0[7:0]: 7bit st tag&dirty ecc info
DCACHE DATA ECC	CDATA0[27:0]: 4 bank * 7bit ecc info
L2CACHE TAG ECC	CDATA0[11:5]: tag ecc CDATA0[4:0]: dirty ecc
L2CACHE DATA ECC	CDATA0[63:0]
DCACHE LD TAG	CDATA0[39:14]: 26 bit tag CDATA0[13:12]:cindex[13:12] CDATA0[0]:VALID
DCACHE LD TAG ECC	CDATA0[6:0]:7 bit ecc info(1bit parity + 6 bit ham code)
ICACHE PREDECODE	CDATA0[31:0]:PREDECD
SNOOP FILTER	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

续下页

表 19.4 - 接上页

RAM 类型	CDATA 内容
TLB TAG	SV39 　　CDATA0[27:0]:VPN 　　CDATA0[31:28]:Page Size 　　CDATA0[47:32]:ASID 　　CDATA0[48]:G SV48 　　CDATA0[35:0]:VPN 　　CDATA0[39:36]:Page Size 　　CDATA0[55:40]:ASID 　　CDATA0[56]:G
TLB DATA	CDATA0[27:0]:PPN CDATA0[40:28]:FLAG 使能并读 Parity:CDATA0[42:41] Parity
TLB TAG RAM ECC	SV39 　　CDATA0[27:0]:VPN 　　CDATA0[31:28]:Page Size 　　CDATA0[47:32]:ASID 　　CDATA0[48]:G 　　CDATA0[52:49]:4’b0 　　使能并读 Parity:CDATA0[54:53] Parity SV48 　　CDATA0[35:0]:VPN 　　CDATA0[39:36]:Page Size 　　CDATA0[55:40]:ASID 　　CDATA0[56]:G 　　CDATA0[60:57]:4’b0 　　使能并读 Parity:CDATA0[62:61]:Parity
TLB DATA RAM ECC	CDATA0[27:0] PPN CDATA0[40:28] FLAG 使能并读 Parity:CDATA0[42:41]:Parity

备注

MCINDEX[20:19] = 2’b00 表示 jTLB;

MCINDEX[20:19] = 2’b01 表示 iuTLB;

MCINDEX[20:19] = 2’b10 表示 duTLB。

19.4.3.4 机器模式 L1 Cache 硬件错误注入寄存器（MEICR）

机器模式 L1 Cache 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。

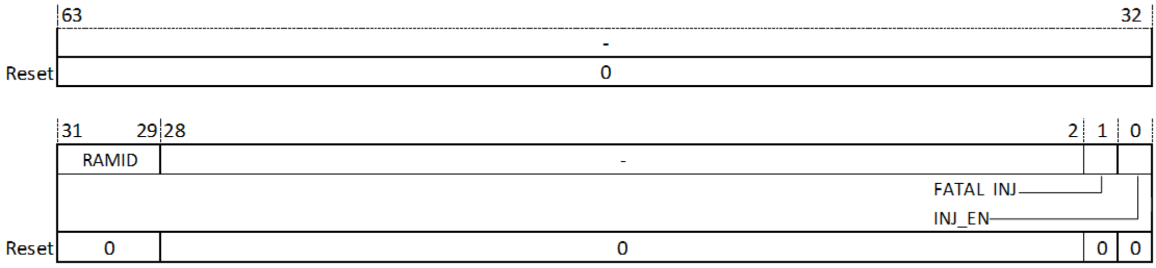


图 19.45: 机器模式 L1 Cache 硬件错误注入寄存器（MEICR）

INJ_EN-ECC error 注入使能位：

当 INJ_EN 为 1 时，L1 Cache ECC error 注入开启；

当 INJ_EN 为 0 时，L1 Cache 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，注入 DCACHE TAG RAM；

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。

19.4.3.5 机器模式 L2 Cache 硬件错误注入寄存器（MEICR2）

机器模式 L2 Cache 硬件错误注入寄存器（MEICR2）用于向 L2 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。

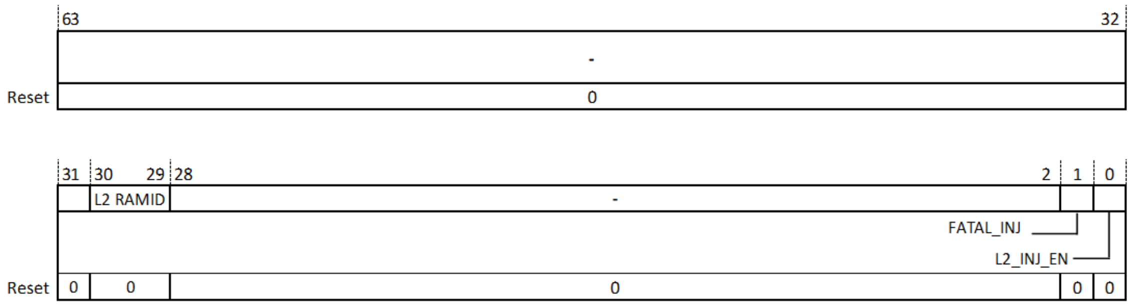


图 19.46: 机器模式 L2 Cache 硬件错误注入寄存器（MEICR2）

L2_INJ_EN-L2 ECC ERROR 注入使能位：

当 L2_INJ_EN 为 1 时，L2 Cache ECC error 注入开启；

当 L2_INJ_EN 为 0 时，L2 Cache 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。

19.4.3.6 L1 LD BUS ERR 地址寄存器（MBEADDR）

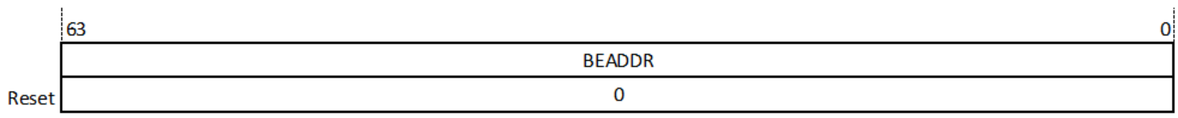


图 19.47: L1 LD BUS ERR 地址寄存器（MBEADDR）

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。

19.4.3.7　Cache 权限控制寄存器（MCPER）

	31 … 1	0
	Reserved	sper
Reset	0	0

图 19.48: Cache 权限控制寄存器（MCPER）

SPER:

当 SPER 为 0 时，仅机器模式拥有执行 表 19.5 所示的 Cache 扩展指令的权限。超级用户模式没有执行 表 19.5 所示的 Cache 扩展指令的权限。

当 SPER 为 1 时，机器模式和超级用户模式都拥有执行所有 Cache 扩展指令的权限。

表 19.5: Cache 扩展指令

指令名称	指令描述
DCACHE.CALL	DCACHE 写回指令
DCACHE.CIALL	DCACHE 写回无效指令
DCACHE.IALL	DCACHE 全部无效指令
DCACHE.CSW	DCACHE 指定 way/set 写回指令
DCACHE.CISW	DCACHE 指定 way/set 写回并无效指令
DCACHE.ISW	DCACHE 指定 way/set 无效指令
ICACHE.IALL	ICACHE 全部无效指令
ICACHE.IALLS	ICACHE 广播全部无效指令

19.4.4　机器模式处理器型号寄存器组

19.4.4.1　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。

19.4.4.2　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。该寄存器的值由端口 pad_cpu_apb_base 决定。

19.4.4.3　片上系统互联寄存器基地址（MAPBADDR2）

该寄存器非 ACE 配置下全 0，ACE 配置下该寄存器的值由端口 pad_cpu_apb_base2 决定。

19.4.5　调试扩展寄存器组

19.4.5.1　玄铁调试原因寄存器（MHALTCAUSE）

	63　　　　4	3　　　　0
	0	MHALTCAUSE
Reset	0	0

图 19.49: 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

19.4.5.2　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。

19.4.5.3　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。

19.4.5.4　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。

19.5　附录 C-5 C920V3 扩展超级用户模式控制寄存器

本附录对 C920V3 扩展超级用户模式控制寄存器进行详细说明。

19.5.1　超级用户模式处理器控制和状态扩展寄存器组

19.5.1.1　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，PMDS 位，PMDU 位可写，用户模式访问会导致非法指令异常。

19.5.1.2 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。

19.5.1.3 超级用户模式 L2 Cache ECC 寄存器（SCER2）

超级用户模式 L2 Cache ECC 寄存器（SCER2）是机器模式 L2 Cache ECC 寄存器（MCER2）的映射，具体信息请参考*机器模式 L2 Cache ECC 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。

19.5.1.4 超级用户模式 L1 Cache ECC 寄存器（SCER）

超级用户模式 L1 Cache 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。

19.5.1.5 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。

机器模式下可读写 scountinhibit，不受 mcounterwen 控制。超级用户模式下，若 mcounterwen.bit[n] 为高，可读写 scountinhibit.bit[n]，否则写无效，读为 0

19.5.1.6 超级用户模式性能监测控制寄存器（SHPMCR）

超级用户模式性能监测控制寄存器，此寄存器为玄铁自扩展寄存器，是 mhpmcr 的读写映射，不包括 sce。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。在 S32 时，只映射 TS 到 BIT[31]。具体详见*超级用户模式性能监测控制寄存器（SHPMCR）*。

19.5.1.7 超级用户模式性能监控起始触发寄存器（SHPMSR）

S-mode 性能监控起始触发寄存器，此寄存器为为 MHPMSR 的映射，当 MHPMCR.sce 为高时，超级用户模式可以通过改组寄存器控制触发的起始地址。详情信息请见*触发寄存器*。

19.5.1.8 超级用户模式性能监控终止触发寄存器（SHPMER）

S-mode 性能监控终止触发寄存器，此寄存器为为 MHPMER 的映射，当 MHPMCR.sce 为高时，超级用户模式可以通过改组寄存器控制触发的终止地址。详情信息请见*触发寄存器*。

19.5.1.9　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 partition 访问 id 号，否则 S 态读写访问非法。

19.5.1.10　超级用户处理器 L2 Cache partition 访问配置寄存器（SLLWP）

当 mxstatus.bit[SPCE] 为 1，超级用户态可以通过配置 SLLWP 来配置 L2 cache 的 partition 访问，否则读写访问非法。

19.5.1.11　超级用户模式 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。

19.5.1.12　超级用户模式 L1 Cache ECC 单比特错误物理地址寄存器（SSBEPA）

超级用户模式 L1 Cache ECC 单比特错误物理地址寄存器（SSBEPA）是机器模式 L1 Cache ECC 单比特错误物理地址寄存器（MSBEPA）的映射，具体信息请参考*机器模式 L1 Cache ECC 单比特错误物理地址寄存器（MSBEPA）*。

该寄存器位长是 64bits，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。

19.5.1.13　超级用户模式 L2 Cache ECC 单比特错误物理地址寄存器（SSBEPA2）

超级用户模式 L2 Cache ECC 单比特错误物理地址寄存器（SSBEPA2）是机器模式 L2 Cache ECC 单比特错误物理地址寄存器（MSBEPA2）的映射，具体信息请参考*机器模式 L2 Cache ECC 单比特错误物理地址寄存器（MSBEPA2）*。

该寄存器位长是 64bits，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。

19.5.1.14　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*。

19.5.1.15　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* 。

19.5.1.16　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* 。

19.6　附录 C-6 C920V3 扩展用户模式控制寄存器

本附录对 C920V3 扩展用户模式控制寄存器进行详细说明。

19.6.1　用户模式扩展浮点控制寄存器组

19.6.1.1　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。

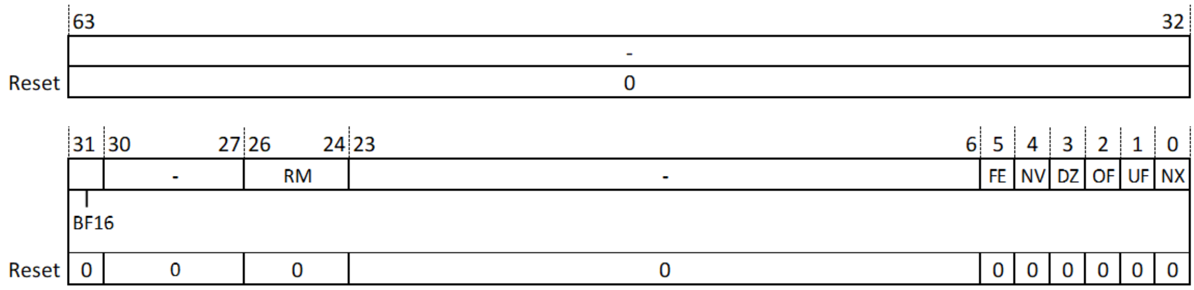


图 19.50: 浮点扩展控制寄存器（FXCR）

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。

RM-舍入模式：

FCSR 对应位的映射。

BF16-Bfloat16 切换位：

当该位为高时，处理器不再支持 half precesion 浮点数，而将 16 位的浮点数按照 BFloat 16 的格式进行处理。

20 附录 D 玄铁 C900 多核同步相关指令和程序实现

20.1　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。

20.2　RISC-V 规范指令

20.2.1　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。

20.2.2　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。

20.2.3　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

20.2.4　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。RV32 只支持 word，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。

20.2.5　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）。

20.3　玄铁增强指令

20.3.1　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)。

20.3.2 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。

20.3.3 icache.iva rs1

将 rs1 中虚拟地址所对应的 icache 表项无效。

20.4 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。

20.4.1 TLB 维护

20.4.1.1 全刷 TLB

```
static inline void local_flush_tlb(unsigned long asid)
{
        __asm__ __volatile__ ("sfence.vma" : : : "memory");
}
```

20.4.1.2 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");
}
```

20.4.1.3 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"
```

(续下页)

(接上页)

```
                                :
                                : "r" (page_add), "r" (asid)
                                : "memory");
                page_add += PAGE_SIZE;
        }
}
```

20.4.1.4 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;
        }
}
```

20.4.2 指令区同步

20.4.2.1 本核全局指令区同步

```
static inline void local_flush_icache_all(void)
{
         asm volatile ("fence.i" ::: "memory");
}
```

20.4.2.2 多核全局指令区同步

```
static void ipi_remote_fence_i(void *info)
{
```

(续下页)

(接上页)

```
        asm volatile ("fence.i" ::: "memory");
}

void flush_icache_all(void)
{
        on_each_cpu(ipi_remote_fence_i, NULL, 1);
}
```

20.4.2.3　玄铁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");
}
```

20.4.3　DMA 同步

20.4.3.1　玄铁多核精确 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)
{
       register unsigned long i asm("a0") = pa_start & ~(L1_CACHE_BYTES - 1);

       for (; i < (start + size); i += L1_CACHE_BYTES)
```

(续下页)

(接上页)

```
                __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");
}
```

20.4.4　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,   )
#else
#define ATOMIC_OPS(op, asm_op, I)                                       \
        ATOMIC_OP (op, asm_op, I, w, int,   )                           \
```

(续下页)

(接上页)

```
        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");                                            \
        return ret;                                                     \
}
```

(续下页)

(接上页)

```
#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
#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
```

(续下页)

(接上页)

```
#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

#undef ATOMIC_OPS

#undef ATOMIC_FETCH_OP
```

(续下页)

(接上页)

```
#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");
         return prev;
}
#define atomic64_fetch_add_unless atomic64_fetch_add_unless
```

(续下页)

(接上页)

```
#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);                  \
}                                                                             \
static __always_inline                                                        \
c_t atomic##prefix##_cmpxchg_release(atomic##prefix##_t *v,                  \
                                     c_t o, c_t n)                            \
```

(续下页)

(接上页)

```
{                                                                          \
        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"
                "       bltz     %[rc], 1f\n"
                "       sc.w.rl  %[rc], %[rc], %[c]\n"
                "       bnez     %[rc], 0b\n"
```

(续下页)

(接上页)

```
                "       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"                 \
                        : "=r" (__ret), "+A" (*__ptr)                   \
                        : "r" (__new)                                   \
                        : "memory");                                    \
```

(续下页)

(接上页)

```
                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                       \
                        : "=r" (__ret), "+A" (*__ptr)               \
                        : "r" (__new)                               \
                        : "memory");                                \
```

(续下页)

(接上页)

```
                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();                                                    \
        }                                                                       \
        __ret;                                                                  \
})
```

(续下页)

(接上页)

```
#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)));        \
})

#define xchg32(ptr, x)                                                  \
({                                                                      \
        BUILD_BUG_ON(sizeof(*(ptr)) != 4);                              \
```

(续下页)

(接上页)

```
        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"                            \
                        "1:\n"                                            \
                        : "=&r" (__ret), "=&r" (__rc), "+A" (*__ptr)      \
                        : "rJ" (__old), "rJ" (__new)                      \
```

(续下页)

(接上页)

```
                                        : "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"                              \
                                        "       bne %0, %z3, 1f\n"                          \
                                        "       sc.d %1, %z4, %2\n"                         \
                                        "       bnez %1, 0b\n"                              \
```

(续下页)

(接上页)

```
                        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");                                          \
                break;                                                        \
        case 8:                                                               \
                __asm__ __volatile__ (                                        \
```

(续下页)

(接上页)

```
                        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"                                 \
                        "1:\n"                                                  \
                        : "=&r" (__ret), "=&r" (__rc), "+A" (*__ptr)           \
                        : "rJ" ((long)__old), "rJ" (__new)                     \
```

(续下页)

(接上页)

```
                        : "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;                                                                \
})
```