XUANTIE

E901

用户手册

此手册支持的 CPU 版本为 R1S0

修订版 03

2025-10-10

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

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

Website: www.xrvm.cn

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

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

网址: www.xrvm.cn

版本历史

版本	描述	日期
01	初版	2025-07-02
02	R1S0P1 版内容更新：	2025-08-29
	- R1S0P1 支持指令总线，文档新增指令总线相关内容	
	正文其它内容更新：	
	- 删除不支持的 DM 寄存器映射信息	
	- 删除不支持的 mxstatus.xuantieisaee 相关信息	
	- 修正 mxstatus、dcsr 寄存器示意图	
	- 修正协处理器指令描述	
	- 优化可选配置的描述	
	- 补充 B 扩展和 Zc 扩展的指令延时	
03	增加关于 CSR 及 Zicsr 的解释说明	2025-10-10

文档编号

玄铁 CPU 技术文档类编号采用如下所示规则：

产品名称-产品型号-产品版本号-文档类型。

术语

逻辑 1：指对应于布尔逻辑真的电平值。

逻辑 0：指对应于布尔逻辑伪的电平值。

置位：指使得某个或某几个位达到逻辑 1 对应的电平值。

清除：指使得某个或某几个位达到逻辑 0 对应的电平值。

保留位：为功能的扩展而预留的，没有特殊说明时其值为 0。

信号：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：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：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：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：最低有效位。

MSB：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]：表示一组地址总线；
* addr[4] 表示最高位是 addr[0] 表示最低位是。

单个的标识符就表示单个信号，比如：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。

RAW 表示写后读操作，WAW 表示写后写操作。

符号

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

目录

图目录

表目录

1 概述

1.1 简介

玄铁 E901 是基于 RISC-V 指令架构的低成本、高能效的 32 位嵌入式 CPU 处理器，用户可以以近似 8 位 CPU 的成本获得 32 位嵌入式处理器的效率与性能。

E901 处理器兼容 RV32E[B][_Zmmul]_Zc 指令架构，采用 16/32 位混合编码系统，指令系统与流水线硬件结构精简高效。同时支持配置协处理器接口，用于满足用户 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。

E901 主要针对智能卡、智能电网、低成本微控制器、无线传感网络等嵌入式应用。

1.2 特点

E901 处理器体系结构的主要特点如下：

- 支持 RV32E[B][_Zmmul]_Zc 指令集
- 16 个 32 位通用寄存器
- 两级顺序执行流水线
- 支持 RISC-V 机器模式
- 支持 RISC-V Debug 架构，支持标准五线 JTAG 调试接口，支持 CJTAG 两线调试接口
- 支持以下硬件乘法器配置：
  - 不配置硬件乘法器
  - 配置单周期快速硬件乘法器
  - 配置多周期（3-33）慢速硬件乘法器
- 兼容 RISC-V CLIC 中断标准，中断优先级固定
- 外部中断源数量最高可配置为 112 个
- 支持系统总线，系统总线协议支持 AHB 2.0 和 AHB-Lite（即 AHB 3.0）
- 可选配指令总线，支持 AHB-Lite 协议
- 支持玄铁扩展编程模型
- 支持复位启动地址硬件集成时可配置

- 支持软复位操作
- 支持协处理器接口可配置
- 最大支持 2MB 的寻址和访存空间

1.3 可配置选项

E901 处理器支持以下配置选项。

表 1.1: E901 可配置选项

配置	选项名称	可选设置
RISC-V B 扩展	B Extension	No/Yes
硬件乘法器（Zmmul）	ZMMUL Extension	No/Small/Fast
指令总线	I-AHB-Lite	No/Yes
系统总线	SYSTEM BUS	AHB-Lite/AHB
协处理器接口	Co-Processor	No/Yes
调试配置	Debug Resources	No/Minimal/Typical
外部中断源量	External Interrupt Num	1~112
mcycle/minstret/mtime/mtimecmp 计数器	MCYCLE/MINSTRET/MTIME/MTIMECMP (32bits)	No/Yes
mtvec/mtvt/jvt 寄存器只读	MTVEC MTVT JVT FIXED	No/Yes
JTAG 接口	JTAG TYPE	2-Wire/5-Wire
RTL 类型	RTL Type	ASIC/FPGA

备注

当在 GUI 进行配置时：

- **ZMMUL Extension**：
  - Fast 表示配置单周期快速乘法器。
  - Small 表示配置多周期（3-33）慢速乘法器。
- **Debug Resources** 的 Minimal/Typical 配置详情，可查看 *调试接口* 章节。
- **MTVEC/MTVT/JVT FIXED**：
  - 选配为 Yes 时，代表 mtvec/mtvt/jvt 寄存器为软件只读，即只读的值由对应硬件接口确定，软件不再可写，同时需要将链接脚本 ld 中异常/中断入口地址及 jump table 地址链接到硬件接口指定的地址。
  - 选配为 No 时，代表 mtvec/mtvt/jvt 寄存器为软件可写。

- **MCYCLE/MINSTRET/MTIME/MTIMECMP** 选配为 No 时，代表硬件不再支持 mcycle/minstret/mtime/mtimecmp 寄存器，核内不再支持 TIMER；针对 RTOS 的延时函数，需要由核外 TIMER 实现。

1.4 可调试性设计

E901 支持标准 2 线或 5 线 JTAG 调试接口。E901 支持所有常见的调试功能，包括软断点、内存断点，寄存器检查和修改、存储器检查和修改，指令单步跟踪与多步跟踪等。

具体请详见*调试接口*章节。

1.5 版本说明

E901 兼容 RISC-V 标准，具体版本为：

- *The RISC-V Instruction Set Manual, Volume I: RISC-V User-Level ISA, Version 20240411* *
- *The RISC-V Instruction Set Manual, Volume II: RISC-V Privileged Architecture, Version 20240411* *
- *RISC-V Core-Local Interrupt Controller (CLIC) Version 0.8* *
- *RISC-V External Debug Support Version 0.13.2*

备注

* E901 对于该标准的实现与原标准定义存在差异。

2 处理器简介

2.1　结构框图

E901 结构框图如 图 2.1 所示。

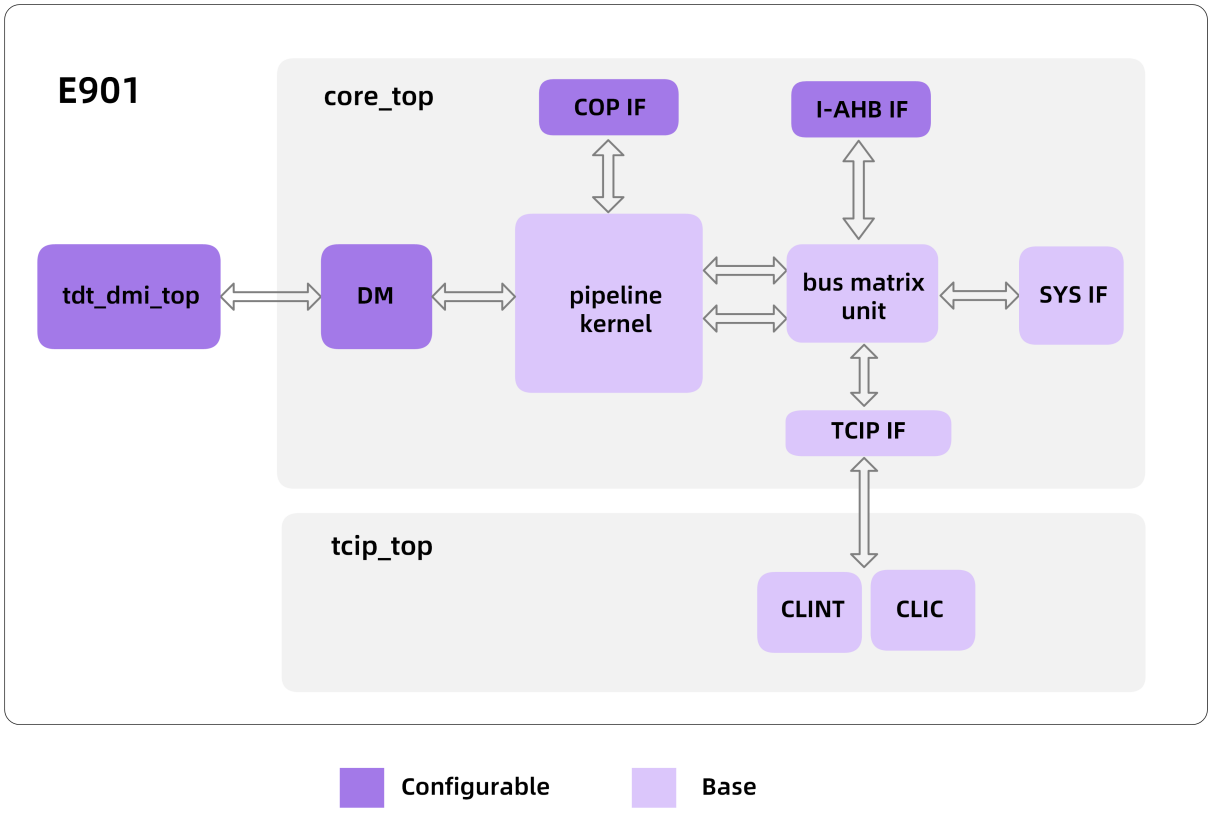


图 2.1: 结构图

E901 处理器采用 2 级流水线结构：取指（IF）和执行（EX）。指令取指阶段主要负责从内存中获取指令；指令执行阶段主要负责指令译码、执行和回写。

TDT 调试单元（XuanTie Debug Trace）支持各种调试方式，包括软件断点、内存断点、单步和多步的指令跟踪等多种方式，可在线调试 CPU、通用寄存器（General-purpose Register, GPR）、控制状态寄存器（Control and Status Register, CSR）和内存。

支持可配置的协处理器单元（Coprocessor），支持用户自扩展指令执行，由 CPU 提供 DSA 指令和操作数信息到用户自扩展执行单元，并接收执行结果。

E901 设计有片上紧耦合的 IP 接口，提供一条 AHB-Lite（AHB 3.0）/AHB 2.0 的系统总线接口，一条支持 AHB-Lite（即 AHB 3.0）协议的指令总线接口。

2.2　紧耦合 IP 架构

为了提高 E901 的系统集成度，方便用户集成与开发，E901 设置内部总线用于集成紧耦合 IP（Tightly Coupled IP，TCIP）。这些紧耦合 IP 包括兼容 RISC-V 标准的 CLINT 和矢量中断控制器 CLIC。

矢量中断控制器的主要特征包括:

- 中断源数量最高可配置 128 个，外部中断源数量最高可配置为 112 个
- 固定中断优先级
- 不支持中断嵌套或中断咬尾，若出现嵌套情况，行为不可预知
- 支持电平中断和脉冲中断

2.3　接口概览

E901 具体接口信号描述参考《玄铁 E901 集成手册》。

3 编程模型

3.1　工作模式及寄存器视图

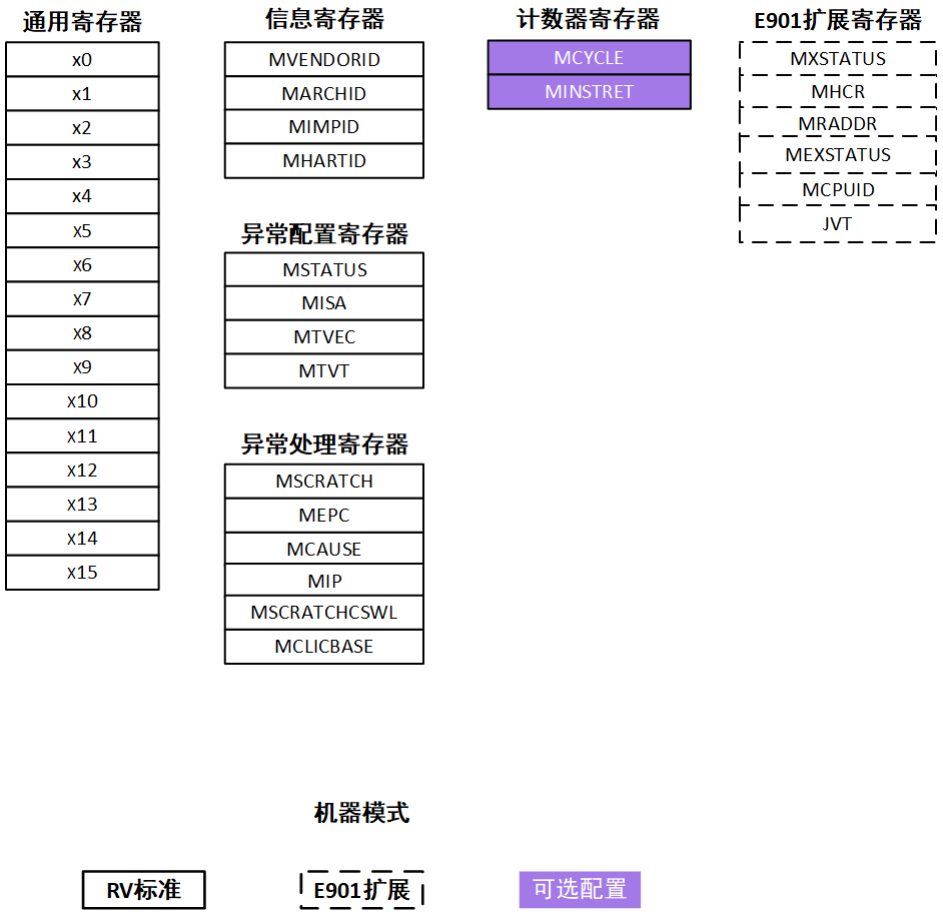


图 3.1: 编程模型

E901 支持运行 32 位操作系统，同时为了极低功耗和极低面积，支持特性如下：

1. 仅支持机器模式。
2. 不支持中断嵌套或中断咬尾。
3. 除了对 CLINT 和 CLIC 的访问，系统对寻址操作及其他加载/存储单元（LSU）访问的地址空间限制为最大 21 位。当地址值超过 21 位时，超出的高位比特（bit）将被丢弃，仅保留低位 21 位用于实际寻址。

4. 对 CLINT 和 CLIC 的访问按照 32 位进行，base address 见 clic_base_address。

3.2　通用寄存器

表 3.1 列出了支持的 16 个 32 位通用寄存器（General-purpose Register, GPR），x0~x15，其中 x0 是零值寄存器。

表 3.1: 通用寄存器

寄存器	ABI 名称	描述
x0	zero	零值
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 ~ a1	函数参数/返回值
x12 ~ 15	a2 ~ a5	函数参数

通用寄存器通常用于存储指令操作数和结果以及地址信息。软硬件上约定这些通用寄存器作为子程序的链接调用、参数传递以及堆栈指针等功能。

3.3　机器模式控制状态寄存器

E901 中实现的 RISC-V 标准定义的机器模式控制状态寄存器（Control and Status Register, CSR）如下表所示。

表 3.2: E901 机器模式编程控制状态寄存器列表

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	厂商编号寄存器
marchid	机器模式只读	0xF12	架构编号寄存器
mimpid	机器模式只读	0xF13	硬件实现编号寄存器

续下页

表 3.2 - 接上页

名称	读写权限	寄存器编号	描述
mhartid	机器模式只读	0xF14	线程编号寄存器
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信息寄存器
mtvec *	机器模式读写	0x305	机器模式异常向量基址寄存器
mtvt *	机器模式读写	0x307	机器模式矢量中断基址寄存器
机器模式异常处理寄存器组			
mscratch	机器模式读写	0x340	机器模式数据备份寄存器
mepc	机器模式读写	0x341	机器模式异常程序计数器
mcause	机器模式读写	0x342	机器模式异常原因寄存器
mip	机器模式只读	0x344	机器模式中断等待寄存器
mscratchcswl	机器模式读写	0x349	机器模式中断数据备份寄存器
mclicbase	机器模式只读	0x350	机器模式中断控制器基址寄存器
机器模式性能监测控制寄存器			
mcounteren **	机器模式只读	0x306	机器模式计数器使能寄存器
mcountinhibit **	机器模式读写	0x320	机器模式计数禁止寄存器
机器模式计数器/计时器组			
mcycle **	机器模式读写	0xB00	机器模式周期计数器
minstret **	机器模式读写	0xB02	机器模式退休指令计数器

备注

* 选配为只读时，读写权限为机器模式只读，软件写会上报异常。

** 计数器选配为无时，硬件不再实现 mcounteren、mcountinhibit、mcycle、minstret，软件读写会上报异常。

具体寄存器的定义和功能，请参考*附录 C 机器模式控制状态寄存器*。

E901 中扩展的机器模式控制状态寄存器如下表所示。

表 3.3: E901 扩展机器模式控制状态寄存器

名称	读写权限	寄存器编号	描述
机器模式扩展寄存器组			
mxstatus	机器模式读写	0x7C0	扩展状态寄存器

续下页

表 3.3 - 接上页

名称	读写权限	寄存器编号	描述
mhcr	机器模式读写	0x7C1	硬件控制寄存器
mraddr	机器模式只读	0x7E0	处理器复位启动地址寄存器
mexstatus	机器模式读写	0x7E1	扩展异常状态寄存器
机器模式处理器型号扩展寄存器组			
mcpuid	机器模式只读	0xFC0	处理器型号寄存器

具体寄存器的定义和功能，请参考*附录 C 机器模式控制状态寄存器*。

3.4 jvt 控制状态寄存器

E901 支持 Zc 扩展，其中的 Zcmt 扩展添加了表跳转指令，并添加了 jvt 控制状态寄存器。

Jump Vector Table（跳转向量表），jvt 寄存器用于配置跳转表的基地址和模式。它允许通过单条指令实现复杂的跳转逻辑，从而减少代码大小。

表 3.4: E901 jvt 控制状态寄存器

名称	读写权限	寄存器编号	描述
jvt *	机器模式读写	0x017	选配为只读时，从 top port 接口定义 jvt 寄存器的值

备注

* 选配为只读时，读写权限为机器模式只读。

具体寄存器的定义和功能，请参考*Zcmt jvt 控制寄存器*。

3.5 调试/追踪寄存器组（与调试模式共享）

以下控制状态寄存器用于支持调试/追踪功能。

表 3.5: 调试/追踪寄存器组（与调试模式共享）

名称	读写权限	寄存器编号	描述
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

续下页

表 3.5 - 接上页

名称	读写权限	寄存器编号	描述
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

具体寄存器的定义和功能，请参考*调试/追踪寄存器组（与调试模式共享）*。

3.6　调试模式寄存器组

以下控制状态寄存器用于支持调试模式，包括 RISC-V 官方定义和玄铁扩展定义的控制状态寄存器。

表 3.6: 调试模式寄存器组

名称	读写权限	寄存器编号	描述
调试模式寄存器组			
dcsr	调试模式读写	0x7B0	调试模式控制与状态寄存器
dpc	调试模式读写	0x7B1	调试模式程序计数器
dscratch0	调试模式读写	0x7B2	调试模式临时数据备份寄存器 0
dscratch1	调试模式读写	0x7B3	调试模式临时数据备份寄存器 1
玄铁调试扩展寄存器			
mhaltcause	机器模式读写	0xFE0	玄铁调试原因寄存器
mdbginfo	机器模式只读	0xFE1	玄铁调试信息寄存器
mpcfifo	机器模式只读	0xFE2	玄铁分支目标地址记录寄存器

具体寄存器的定义和功能，请参考*调试模式寄存器组*。

3.7　异常处理

异常处理（包括指令异常和外部中断）是处理器的一项重要技术，在异常事件产生时，用来使处理器转入对异常事件的处理。

详细的异常与中断的处理过程请参考*异常与中断*章节。

3.8　数据格式

3.8.1　整型数据格式

E901 寄存器内部的数值存在有符号和无符号的区别。其格式均为从右至左表示逻辑低位到高位的排布，如 图 3.2 所示。

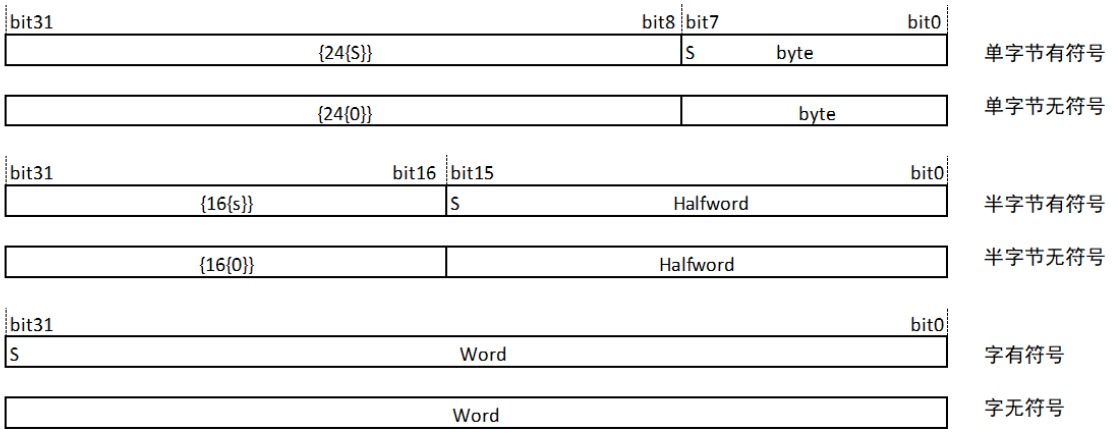


图 3.2: 寄存器中的整型数据组织结构

3.8.2　小端

存储器数据有大小端的区分，E901 仅支持小端模式，内存中的数据组织形式如 图 3.3 所示，即数据高位存放至物理内存的高地址。

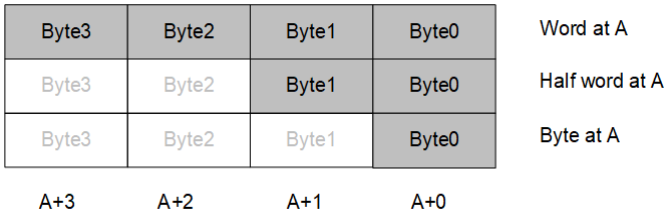


图 3.3: 内存中的数据组织形式

4 异常与中断

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

对应于 RISC-V 标准，定义如下：

- 异常（exception）：狭义而言，指代当前硬件线程（hart）中与指令相关联的运行时异常情况。
- 中断（interrupt）：指代外部的异步事件，它可能导致硬件线程发生意外的控制转移。
- 陷阱（trap）或异常处理：指代由异常或中断引发的控制转移到陷阱处理程序的过程。

注：在“异常处理”的语境下，“异常”泛指（指令）异常（exception）和（外部）中断（interrupt）。

异常处理是处理器根据内部或外部的异常事件从正常的程序处理转入特定的异常处理程序，异常事件可分类如下：

- 引起异常的外部事件包括：外部设备的中断请求、读写访问错误等。
- 引起异常的内部事件包括：非法指令和非对齐访问错误（misaligned error）等。
- 某些特殊功能指令正常执行时也会产生异常，如 ECALL 和 EBREAK 指令。

E901 兼容 RISC-V 标准的异常向量号，如 表 4.1 所示。

异常处理的关键是在异常发生时，保存 CPU 当前指令运行的状态，在退出异常处理时恢复异常处理前的状态，具体如下：

- 识别异常：异常能够在指令流水线的各个阶段被识别，并使后面的指令不会改变 CPU 的状态。
- 处理异常：异常在指令的边界上被处理，即 CPU 在指令退休时响应异常，使用异常保留程序计数器（mepc）保存下一条待执行的指令的地址，以便用于退出异常处理后执行。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。异常处理利用异常向量表跳转到异常服务程序的入口。
- 存储指令地址：E901 根据异常识别时的指令是否完成决定异常保留程序计数器（mepc）应当存储哪一条指令的地址。具体如下：
  - 如果异常事件是外部中断服务请求，被中断的指令将正常退休并改变 CPU 的状态，它的下一条指令的地址（pc+2 或者 pc+4，根据当前指令是 16 位或 32 位决定 +2 或者 +4）将被保存在 mepc 寄存器中作为中断返回时指令的入口。
  - 如果异常事件是由访问错误指令产生的，因为这条指令不能完成，它将异常退休但不改变 CPU 的状态（即不改变寄存器的值），这条访问错误地址指令的地址（pc 的值）将被保存在 mepc 寄存器中，CPU 从异常服务程序返回时继续执行这条访问错误指令。

表 4.1: 异常向量号

中断标记	向量号	异常中断类型
1	0 ~ 2	保留
1	3	机器模式软件中断
1	4 ~ 6	保留
1	7	机器模式计时器中断（不选配计数器资源的情况下不支持）
1	8 ~ 15	保留
1	16	CLIC 外接中断 0（pad_clic_int_vld[0]）
1	……	……
1	16+i	CLIC 外接中断 i（pad_clic_int_vld[i]）
1	……	……
1	127	CLIC 外接中断 111（pad_clic_int_vld[111]）
0	0	保留
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	存储指令非对齐访问异常
0	7	存储指令访问错误异常
0	8	保留
0	9	保留
0	10	保留
0	11	机器模式环境调用异常
0	>=12	保留

4.1　异常

4.1.1　异常响应

RISC-V 编程模型中没有异常使能寄存器，因此一旦触发异常即可进行响应。按照 RISC-V 标准定义，中断优先级高于异常，异常内部优先级定义如下表所示。

表 4.2: 异常优先级定义

优先级	向量号	异常类型
最高	3	调试断点异常
↓	1	取指令访问错误异常
↓	2	非法指令异常
↓	11	机器模式环境调用异常
↓	6	存储指令非对齐访问异常
↓	4	加载指令非对齐访问异常
↓	7	存储指令访问错误异常
↓	5	加载指令访问错误异常

异常响应会打断 CPU 正常的程序执行轨迹，转而处理该异常事件，因此在异常响应时需要对 CPU 现场状态进行保存，并在 CPU 退出异常服务程序时恢复现场并执行异常响应前的程序流。异常响应按如下步骤进行现场保存：

异常按以下步骤被处理，所有步骤在一个处理器时钟周期完成：

1. 处理器保存 pc 到异常保留程序计数器（mepc）中。
2. 将 mcause 中的异常向量号 Exception Code 域更新为当前发生的异常向量号，标识异常类别，具体向量号如 表 4.1 所示。
3. 将 mstatus 中的中断使能位 MIE 保存到 MPIE 中。
4. 将 mstatus 中的中断使能位 MIE 位清零，禁止响应中断。
5. 处理器根据向量基址寄存器 mtvec 中的基址得到异常服务程序入口地址进行跳转执行。

所有异常的跳转入口均由 mtvec 寄存器定义，从该地址取回的第一笔数据即为异常服务程序的第一条指令。因为只实现 mtvec.MODE = 3 这一模式，因此要求异常服务程序的入口地址为 64 字节对齐。具体请参考*机器模式异常向量基址寄存器（mtvec）*。

硬件选配 mtvt、mtvec 只读后，寄存器 mtvt、mtvec 由硬件接口指定地址，软件不可写。

4.1.2 异常处理

如上节所述，所有异常响应时的跳转执行入口均由 mtvec 寄存器指定，因此在跳转到该入口之后，软件可依据 mcause 寄存器中的异常向量号来决定是否实现跳转到各自对应的服务程序进行处理。需要注意的是，除了在异常响应时对必要的 CSR 寄存器及 pc 等现场状态进行保存外，在异常服务程序入口需要软件对 GPR 等需要用到的寄存器进行压栈处理。

4.1.3 异常返回

异常服务程序的返回需要通过执行 MRET 指令实现。MRET 指令执行时会将异常响应时保存的 CPU 现场进行恢复，主要包括如下方面：

1. 将 pc 恢复成 mepc 寄存器的值，保证 CPU 从异常服务程序返回后可以从触发异常的地方重新执行指令。这就要求上节的异常处理过程中将触发该异常的事件进行修复，避免再次触发异常。
2. mstatus.MIE 被恢复成 mstatus.MPIE 的值，MPIE 被设置为 1。
3. mxstatus.PM 及 mstatus.MPP 硬件固定为 2'b11，表示机器模式。

4.1.4 锁定

如前文所示，RISC-V 编程模型中没有定义异常的使能寄存器，在从异常服务程序返回到正常程序流时也没有对 mcause 寄存器中的异常向量号进行清除，因此软件开发人员不能方便的从编程模型所定义寄存器中得知 CPU 当前正在处理异常还是运行正常程序轨迹。E901 中实现了*扩展异常状态寄存器（mexstatus）*，当 CPU 响应异常时会将该寄存器的 EXPT_VLD 域置为 1，通过 MRET 指令从异常服务程序返回后该域被清零。

当 CPU 响应异常时会判断当前程序流是否处于异常服务程序中（通过 mexstatus 的 EXPT_VLD 域），如果 CPU 正在处理异常，还未从异常服务程序返回时又触发新的异常，将会导致 CPU 被锁定。CPU 被锁定时会置位 CPU 顶层输出信号（cpu_pad_lockup）为高，告知 SoC CPU 处于锁定状态，同时 CPU 停止指令取指，执行并保持 pc 为触发 CPU 锁定的指令 pc，mexstatus 寄存器中的 LOCKUP 域被设置为 1。

调试请求可以将 CPU 的锁定状态打断，CPU 传递给 SoC 的锁定指示信号被拉低。

在 CPU 从调试模式退出后有所差异，在调试模式下如果软件将 CPU 退出调试模式跳转执行的 pc 设为 0x1FFFFC，CPU 在退出调试后仍将处于锁定状态。如果软件不将 CPU 退出调试模式跳转执行的 pc 设为 0x1FFFFC，CPU 在退出调试模式时锁定状态也被清除（传递给 SoC 的指示 CPU 处于锁定状态的 LOCKUP 有效信号以及 mexstatus.LOCKUP），包括导致 CPU 进入锁定状态的异常信息（mexstatus.EXPT_VLD），CPU 继续执行指令。

当 CPU 处于锁定状态时，建议系统设计人员将 CPU 核进行复位。

特殊的是，执行 ECALL 或者 EBREAK 主动触发的异常在和其他类型的异常嵌套时不会触发 CPU 的锁定。

4.2 中断

E901 设计实现了兼容 RISC-V 标准的处理器核局部中断（以下简称 CLINT）和核内局部中断控制器（以下简称 CLIC）。CLINT 包括机器模式软件中断，机器模式计时器中断以及机器模式外部中断。CLIC 负责对中断源进行采样，优先级仲裁和分发。

E901 内部设计实现的 CLIC 兼容 CLIC SPEC-0.8 版本，按照 SPEC 定义，硬件实现 CLIC 时，mtvec.MODE[1:0] 扩展出 2'b11 这一模式，支持硬件矢量中断和非矢量中断，中断服务程序入口可由 mtvt 寄存器指定。E901 只实现了 mtvec.MODE[1:0] = 2'b11 这一模式，本节主要对该模式下矢量中断和非矢量中断的处理进行描述。

中断优先级固定为对应的中断向量号，不支持中断咬尾、中断嵌套。硬件选配寄存器 mtvt、mtvec 只读时，寄存器 mtvt、mtvec 由硬件接口指定地址，软件不可写配置，软件写会上报异常。

4.2.1 矢量中断

可通过将 CLIC 中每个中断的配置寄存器 clicintattr 中的 SHV 域设置为 1 来指示该中断为硬件矢量中断。

4.2.1.1　矢量中断优先级

中断优先级固定为对应的中断向量号。

4.2.1.2　矢量中断响应

为了保证中断被正常响应，必须保证全局中断使能位 mstatus.MIE 为 1 以及该中断对应的 clicintie 寄存器中的使能位为 1。

中断响应按以下步骤被处理，所有步骤在一个处理器时钟周期完成：

1. 处理器保存 pc 到异常保留程序计数器（mepc）中，如果响应中断的指令本身同时触发异常，保存在 mepc 寄存器中的值会是响应中断的指令本身，否则为响应中断指令的下一条指令。
2. 将 mcause 中的异常向量号 Exception Code 域更新为当前有效的中断号，具体向量号如 表 4.1 所示。并且将 mcause 寄存器最高位置为 1，表示 CPU 响应的是中断。
3. 将 mstatus 中的中断使能位 MIE 保存到 MPIE 中。
4. 将 mstatus 中的中断使能位 MIE 位清零，禁止响应中断。
5. 对于脉冲中断而言，CPU 会自动清除其对应 clicintip 寄存器中的 pending 位，而对于电平中断而言则不会清除。

在 mtvec.MODE=2'b11 时，硬件矢量中断的服务程序入口是由 mtvt 寄存器指定的基址加中断号所指定的偏移量决定的，具体请参考 *机器模式异常向量基址寄存器（mtvec）* 。

中断优先级固定为对应的中断向量号，硬件选配只读后，寄存器 mtvt、mtvec 由硬件接口指定地址，软件不可读写配置。

4.2.1.3　矢量中断处理

在中断服务程序入口需要视中断服务程序内所用的 GPR 数量对其进行压栈处理，压栈完成后可以设置 mstatus.MIE 域为 1 来使能中断，达到进一步响应中断的目的。

E901 不支持中断嵌套或中断咬尾。

4.2.1.4　矢量中断返回

中断服务程序的退出必须使用 MRET 指令完成，在执行 MRET 指令之前需要软件将中断服务程序入口压栈保存的现场进行弹栈处理。通过执行 MRET 指令将响应中断之前的 CPU 现场进行恢复，主要有如下操作：

1. 将 pc 恢复成 mepc 寄存器的值。
2. mstatus.MIE 恢复成 mstatus.MPIE 的值。

4.2.2　非矢量中断

可通过将 CLIC 中每个中断的配置寄存器 CLICINTATTR 中的 shv（Selective Hardware Vectoring）域设置为 0 来指示该中断为非矢量中断。

4.2.2.1　非矢量中断优先级

同*矢量中断优先级*节所描述。

4.2.2.2　非矢量中断响应

跟矢量中断的中断响应类似，CPU 会保存响应中断时的处理器现场。

所不同的是 CPU 在响应中断时不会主动清除 CLIC 内对应中断的 pending 位，无论脉冲中断还是电平中断。需要软件在中断服务程序中写 CLIC 对应 PENDING 寄存器 0 触发 pending 位的清除。

另外，对于非矢量中断而言，中断服务程序入口都是统一由 mtvec 寄存器指定，不同于硬件矢量中断的由 mtvt 加中断号偏移量指定的模式。另外，从该地址取回的数据即为中断服务程序的第一条指令。

4.2.2.3　非矢量中断返回

同*矢量中断返回*所描述。

4.2.2.4　非矢量中断咬尾

E901 不支持中断嵌套或中断咬尾。

5 指令集

E901 采用了 16/32 位混合编码的 RV32E[B][_Zmmul]_Zc 指令集，并在此基础上扩展了玄铁自定义指令。

5.1 RV32E[B][_Zmmul]_Zc 指令

本章主要介绍了 E901 中实现的 RISC-V(RV) 指令集，包括：

- 标准的 RV 整型指令集（RV32E）
- RV 乘法指令集（Zmmul）
- RV B 扩展指令集
- RV Zc 扩展指令集

5.1.1 RV32E 整型指令集

本节介绍了 32 位基本整型指令 RV32E。作为必备基础指令集，RV32E 相比 RV32I 而言，实现的指令相同，但 RV32E 仅实现 16 个 GPR，访问高 16 个 GPR 时会触发非法指令异常。基本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[1]
- 低功耗指令
- 异常返回指令
- 特殊功能指令

[1] 控制寄存器操作指令在 The RISC-V Instruction Set Manual Volume I: Unprivileged Architecture 中被归类为 Zicsr 扩展，代表 Extension for Control and Status Register (CSR) Instructions。

表 5.1: 整型指令（RV32E）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
ADDI	有符号立即数加法指令	1
SUB	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
SLLI	立即数逻辑左移指令	1
SRL	逻辑右移指令	1
SRLI	立即数逻辑右移指令	1
SRA	算术右移指令	1
SRAI	立即数算术右移指令	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
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

续下页

表 5.1 - 接上页

指令名称	指令描述	执行延时
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	2（总线零延时）
LBU	无符号扩展字节加载指令	
LH	有符号扩展半字加载指令	
LHU	无符号扩展半字加载指令	
LW	有符号扩展字加载指令	
SB	字节存储指令	
SH	半字存储指令	
SW	字存储指令	
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
CSRRS	控制寄存器置位传送指令	
CSRRC	控制寄存器清零传送指令	
CSRRWI	控制寄存器立即数读写传送指令	
CSRRSI	控制寄存器立即数置位传送指令	
CSRRCI	控制寄存器立即数清零传送指令	
低功耗指令		
WFI	进入低功耗模式指令	不可预期
异常返回指令		
MRET	机器模式异常返回指令	阻塞执行
特殊功能指令		
FENCE	存储同步指令	不可预期
FENCE.I	指令流同步指令	阻塞执行
ECALL	环境异常调用指令	1
EBREAK	断点指令	1

具体指令说明和定义，请参考*附录 E 指令术语*。

5.1.2 RV32_Zmmul 指令集

RV32_Zmmul 指令如 表 5.2 所示。

表 5.2: RV32_Zmmul 指令集列表

指令名称	指令描述	执行延时
乘法指令		
MUL	有符号乘法指令	1/3-33
MULH	有符号乘法取高位指令	1/3-33
MULHSU	有符号与无符号乘法取高位指令	1/3-33
MULHU	无符号乘法取高位指令	1/3-33

具体指令说明和定义，请参考*附录 Zmmul 扩展指令术语*。

5.1.3 RV32B 指令集

E901 支持 Zba, Zbb 和 Zbs 扩展。分别提供如下指令：

Zba：地址生成（Address generation）

表 5.3: RISC-V Zba 指令集

指令名称	指令描述	指令延时
SH1ADD	Shift left by 1 and add	1
SH2ADD	Shift left by 2 and add	1
SH3ADD	Shift left by 3 and add	1

Zbb：基本位操作（Basic bit-manipulation）

表 5.4: RISC-V Zbb 指令集

指令名称	指令描述	指令延时
ANDN	AND with inverted operand	1
ORN	OR with inverted operand	1
XNOR	Exclusive NOR	1
CLZ	Count leading zero bits	1
CTZ	Count trailing zeros	1
CPOP	Count set bits	1
MAX	Maximum	1
MAXU	Unsigned maximum	1
MIN	Minimum	1
MINU	Unsigned minimum	1
SEXT.B	Sign-extend byte	1

续下页

表 5.4 - 接上页

指令名称	指令描述	指令延时
SEXT.H	Sign-extend halfword	1
ZEXT.H	Zero-extend halfword	1
ROL	Rotate Left (Register)	1
ROR	Rotate Right	1
RORI	Rotate Right (Immediate)	1
ORC.B	Bitwise OR-Combine, byte granule	1
REV8	Byte-reverse register	1

Zbs：单比特操作（Single-bit）

表 5.5: RISC-V Zbs 指令集

指令名称	指令描述	指令延时
BCLR	Single-Bit Clear (Register)	1
BCLRI	Single-Bit Clear (Immediate)	1
BEXT	Single-Bit Extract (Register)	1
BEXTI	Single-Bit Extract (Immediate)	1
BINV	Single-Bit Extract (Immediate)	1
BINVI	Single-Bit Invert (Immediate)	1
BSET	Single-Bit Set (Register)	1
BSETI	Single-Bit Set (Immediate)	1

具体指令说明和定义，请参考 *The RISC-V Instruction Set Manual, Volume I: RISC-V User-Level ISA, Version 20240411*。

5.1.4 RV32 Zc 指令集

E901 支持 Zca、Zcb、Zcmp 和 Zcmt 扩展。

Zca 指令集指不包含浮点相关的 C 扩展。Zca 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 5.6: Zca 指令

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.ADDI	有符号立即数加法指令	1
C.SUB	有符号减法指令	1
C.ADDI16SP	堆栈指针有符号自加指令	1
C.ADDI4SPN	堆栈指针无符号加法指令	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.JAL	无条件跳转子程序指令	1
C.JALR	寄存器跳转子程序指令	1
内存存取指令		
C.LW	字加载指令	2（总线零延时）
C.SW	字存储指令	2（总线零延时）
C.LWSP	字堆栈加载指令	2（总线零延时）
C.SWSP	字堆栈存储指令	2（总线零延时）
特殊指令		

续下页

表 5.6 - 接上页

指令名称	指令描述	执行延时
C.NOP	空操作指令	1
C.EBREAK	调试断点指令	1

具体指令说明和定义，请参考*附录 Zca 指令术语*。

Zcb：Zcb 提供简单的代码大小节省指令。

表 5.7: Zcb 指令

指令名称	指令描述	指令延时
C.LBU	Load unsigned byte	2（总线零延时）
C.LHU	Load unsigned halfword	2（总线零延时）
C.LH	Load signed halfword	2（总线零延时）
C.SB	Store byte	2（总线零延时）
C.SH	Store halfword	2（总线零延时）
C.ZEXT.B	Zero extend byte	1
C.SEXT.B	Sign extend byte	1
C.ZEXT.H	Zero extend halfword	1
C.SEXT.H	Sign extend halfword	1
C.NOT	Bitwise not	1
C.MUL	Multiply	1/3-33

Zcmp：Zcmp 包括部分基于堆栈指针的加载和存储（PUSH/POP and double move）。

表 5.8: Zcmp 指令

指令名称	指令描述	指令延时
CM.PUSH	Push register to stack	3~7
CM.POP	Pop register from stack	3~7
CM.POPRETZ	Pop data from the stack and return with zero flag set	5~9
CM.POPRET	Pop data from the stack and return	4~8
CM.MVA01S	Move two s0-s7 registers into a0-a1	2
CM.MVSA01	Move a0-a1 into two different s0-s7 registers	2

Zcmt：Zcmt 添加了表跳转指令，并添加了 jvt 控制状态寄存器。

表 5.9: Zcmt 指令

指令名称	指令描述	指令延时
CM.JT	Jump via table	3（总线零延时）
CM.JALT	Jump and link via table	3（总线零延时）

具体指令说明和定义，请参考 *The RISC-V Instruction Set Manual, Volume I: RISC-V User-Level ISA, Version 20240411*。

5.2 玄铁扩展指令集

5.2.1 玄铁协处理器指令集

为方便部分用户快速使用协处理器指令，玄铁 CPU 预设了部分用户可能常用的协处理器扩展指令，该部分指令已经在编译工具中进行了支持，指令编码使用 custom2 域。

使用玄铁扩展的用户自扩展协处理器指令集（包括玄铁预设的协处理器扩展指令），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 5.10: 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
	CPX9
	CPX10

具体指令说明和定义，请参考*协处理器接口*和*附录玄铁扩展协处理器指令术语*。

6 协处理器接口

6.1　概述

E901 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。E901 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。

使用玄铁扩展的用户自扩展协处理器指令集（包括玄铁预设的协处理器扩展指令），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。

E901 协处理器结构图如下 图 6.1 所示：

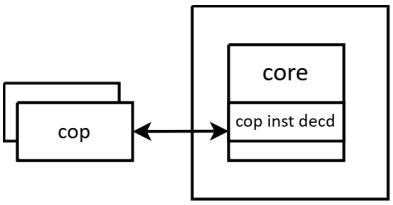


图 6.1: 协处理器结构示意图

E901 协处理器基本特性如下：

- 开放自定义译码接口
- 开放 GPR 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
- 支持玄铁预设扩展指令（custom2）及客户自定义扩展指令（custom1-custom3）
  - 当使用玄铁预设扩展指令（custom2）时，支持配置最多 4 个协处理器扩展
  - 当使用客户自定义扩展指令（custom1-custom3）时，支持配置最多 32 个协处理器扩展

提示

- custom1 对应 [6:0]:0101011

- custom2 对应 [6:0]:1011011
- custom3 对应 [6:0]:1111011

6.2　协处理器指令扩展

6.2.1　协处理器指令接口特性

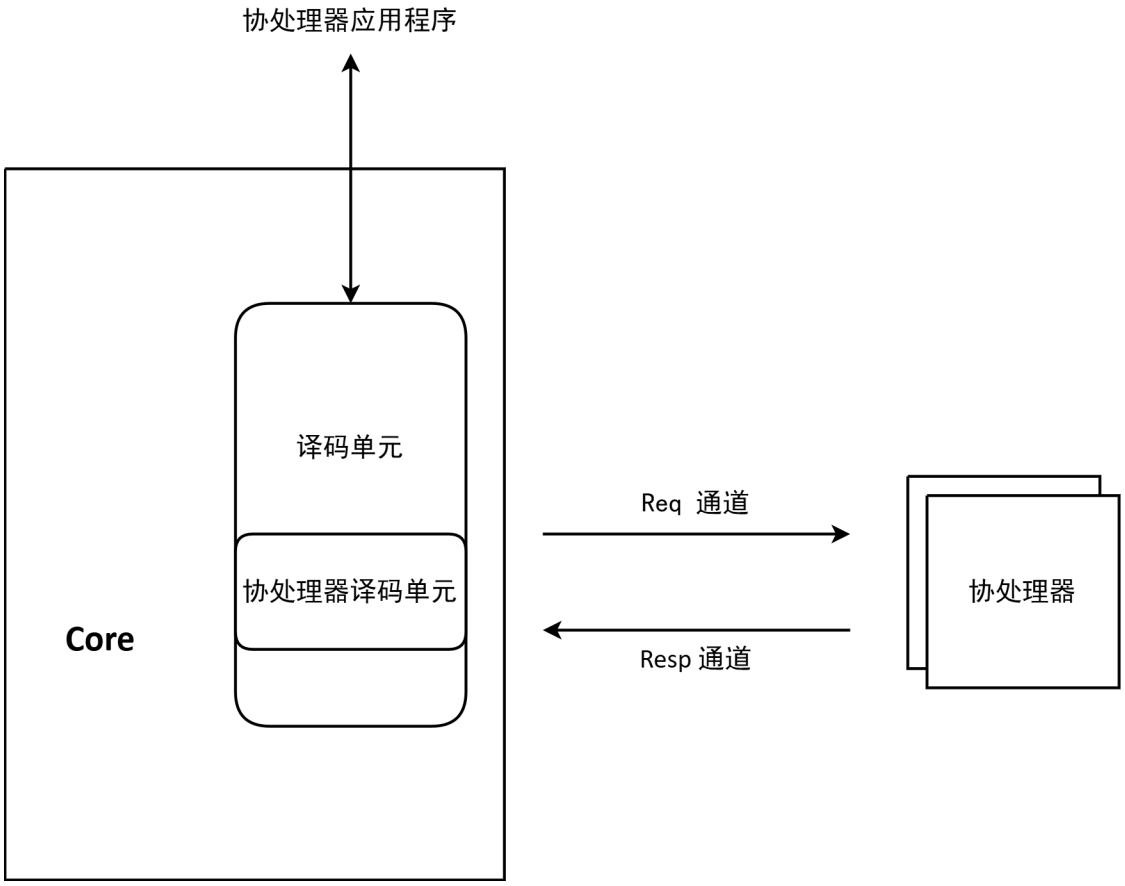


图 6.2: 协处理器指令接口示意图

玄铁协处理器扩展指令支持使用 custom 域进行指令扩展，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。

E901 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。

6.2.2　协处理器扩展指令支持

用户可以通过使用 custom2 域自定义扩展指令，可定义的指令类型为 scalar 整型类型。支持将 GPR 作为指令操作数，并通过协处理器接口发送至协处理器。

- 当使用玄铁预设扩展指令（custom2）时，玄铁预设自定义扩展指令下协处理器数量固定为最大 4 个。
- 当使用客户自定义扩展指令（custom1-custom3）时，支持自定义协处理器个数，即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 整型扩展指令最多支持 2 个寄存器作为 source 数据发送给协处理器。index 约定最大为 5bit，即最多可以控制 32 个协处理器。

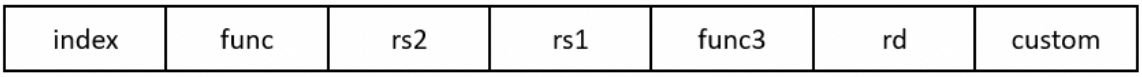


图 6.3: 协处理器指令编码示例

图 6.3 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 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，只需要提供所编码里的寄存器信息即可。

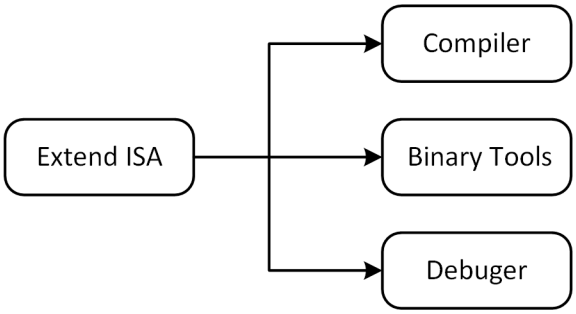


图 6.4: 协处理器工具支持

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。

6.2.3　协处理器指令译码

E901 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 sc_iu_decd_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：

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*附录玄铁扩展协处理器指令术语*。

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

- 源操作数和目的操作数索引
- 指令非法信息标记

当 E901 直接将指令发送给协处理器，等待协处理器指令响应完成后才会退休。

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 E901 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 E901 卡死。

6.3　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 E901 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。主要的使用场景如下 图 6.5 。

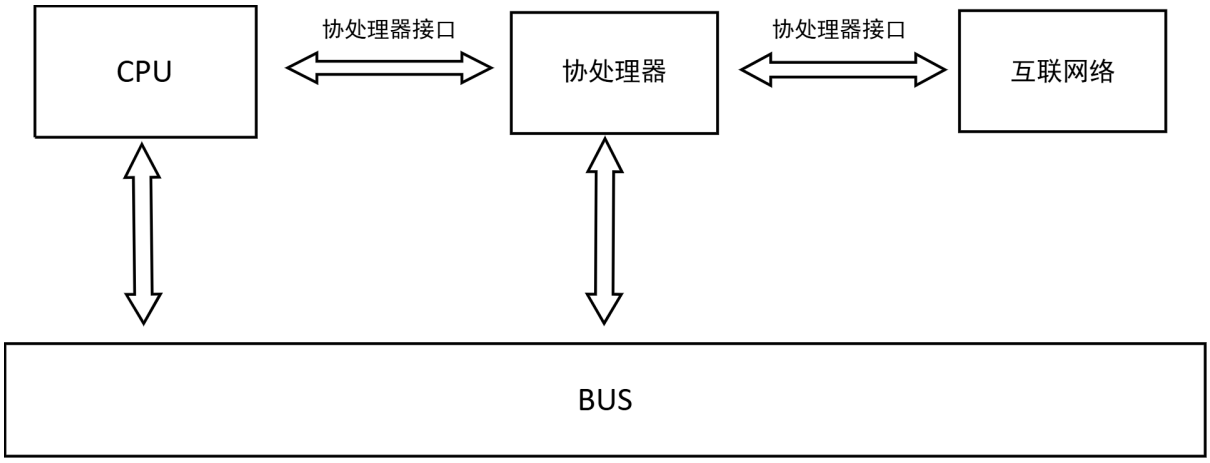


图 6.5: 协处理器接口互联场景

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 bus，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。

7 总线矩阵与总线接口

7.1 简介

E901 实现了多总线接口，分别包括指令总线和系统总线，以及紧耦合 IP 接口。其中指令总线地址空间可由用户根据实际的系统需要进行配置，紧耦合 IP 接口的地址空间固定为 0xE0000000 ~ 0xEFFFFFFF，剩余地址空间对应系统总线。

总线矩阵为处理器内部请求访问外部总线接口提供了互联功能。总线矩阵与 CPU 内部请求及总线接口的连接关系如 图 7.1 所示。

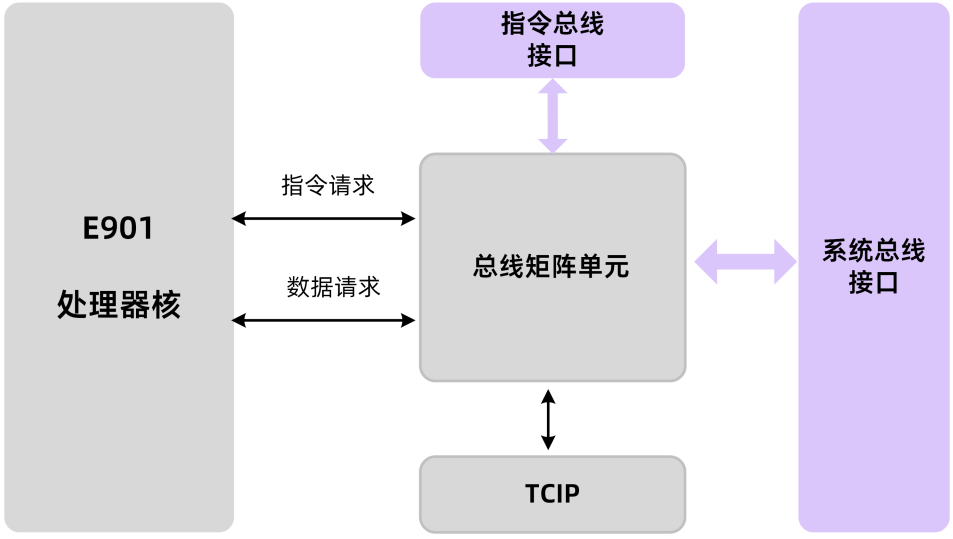


图 7.1: 总线矩阵

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

另外，E901 通过提供一组接口信号（pad_bmu_iahbl_base 和 pad_bmu_iahbl_mask ），支持指令总线基地址和空间大小硬件集成时可配置。其中，pad_bmu_iahbl_base 指定了指令总线的基地址；pad_bmu_iahbl_mask 指定了不同地址空间下对地址对齐的需求。

指令总线的地址空间 512B 到 2MiB 可配置，例如设置指令总线地址空间大小为 4KiB，pad_bmu_iahbl_base[2:0] 必须为 3'b0，pad_bmu_iahbl_mask[11:0] 必须为 12'b1111 1111 1000。不同大小的地址空间具体要求见 表 7.1 。

表 7.1: 指令总线对基地址和地址对齐的要求

地址空间大小	对 pad_bmu_iahbl_base 的要求	对 pad_bmu_iahbl_mask 的要求
512B	没有要求	bit[11:0]=12'b1111 1111 1111
1KiB	bit[0] =0	bit[11:0]=12'b1111 1111 1110
2KiB	bit[1:0] =2'b0	bit[11:0]=12'b1111 1111 1100
4KiB	bit[2:0] =3'b0	bit[11:0]=12'b1111 1111 1000
8KiB	bit[3:0] =4'b0	bit[11:0]=12'b1111 1111 0000
16KiB	bit[4:0] =5'b0	bit[11:0]=12'b1111 1110 0000
32KiB	bit[5:0] =6'b0	bit[11:0]=12'b1111 1100 0000
64KiB	bit[6:0] =7'b0	bit[11:0]=12'b1111 1000 0000
128KiB	bit[7:0] =8'b0	bit[11:0]=12'b1111 0000 0000
256KiB	bit[8:0] =9'b0	bit[11:0]=12'b1110 0000 0000
512KiB	bit[9:0] =10'b0	bit[11:0]=12'b1100 0000 0000
1MiB	bit[10:0] =11'b0	bit[11:0]=12'b1000 0000 0000
2MiB	bit[11:0] =12'b0	bit[11:0]=12'b0000 0000 0000

注意

使用时不能把数据放到 IAHB 区域。

E901 不允许指令总线地址空间与 TCIP 接口相互交叠。当总线接口配置错误，导致一个地址请求同时命中指令总线地址空间和 TCIP 接口中的多个时，CPU 行为将不可预测。

因此在硬件配置有多条总线接口时，不要将指令总线配置成 2MiB 空间。

7.2　系统总线接口

E901 的系统总线接口支持 AMBA3.0 AHB-Lite 和 AMBA2.0 AHB 协议，请参考 AMBA 3.0 规格说明—AMBA3 AHB-Lite Protocol Specification Rev 1.0。E901 的系统总线接口只实现了 AHB-Lite 协议中的部分内容。

E901 支持 21 位地址和 32 位数据。

在 AHB-Lite 协议下，作为主设备，总线接口支持的传输类型为：

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

在 AHB 及 AHB-Lite 协议下，总线接口接受从设备的响应类型为：

- HREADY 支持 Ready 和 Not Ready

- HRESP 支持 OKAY 和 ERROR，其它响应类型不支持

7.3　指令总线接口

E901 的指令总线支持 AMBA3.0 AHB-Lite 协议，可参考 AMBA 3.0 规格说明- AMBA3 AHB-Lite Protocol Specification Rev1.0。E901 指令总线接口只实现了 AHB-Lite 协议中的部分内容。

E901 支持 21 位地址和 32 位数据。

在 AHB-Lite 协议下，作为主设备，指令总线接口支持的传输类型为：

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

在 AHB-Lite 协议下，指令总线接口接受从设备的响应类型为：

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

8 CLINT 中断

E901 实现了处理器核局部中断（以下简称 CLINT），包括软件中断和计时器中断，该模块寄存器映射在紧耦合 IP 的地址空间。

8.1 寄存器地址映射

CLINT 中断占据 32KiB 内存空间。

高 16 位为固定分配地址 0xE000，低 16 位地址映射如 表 8.1 所示。所有寄存器仅支持字对齐的访问。

没有选配计数器资源时，不支持 mtimecmpl 或 mtimel 寄存器。

表 8.1: CLINT 寄存器存储器映射地址

地址 [15:0]	名称	类型	初始值	描述
0x0000	msip	读/写	0x00000000	机器模式软件中断配置寄存器： 高位硬件固定为 0，bit[0] 有效。
Reserved	-	-	-	-
0x4000	mtimecmpl	读/写	0xFFFFFFFF	机器模式计时器： 比较值寄存器
Reserved	-	-	-	-
0xBFF8	mtimel	读	0x00000000	机器模式计时器： 当前值寄存器，该寄存器值为 pad_cpu_sys_cnt[31:0] 信号的值。
Reserved	-	-	-	-

8.2 软件中断

CLINT 可用于软件配置产生软件中断。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。

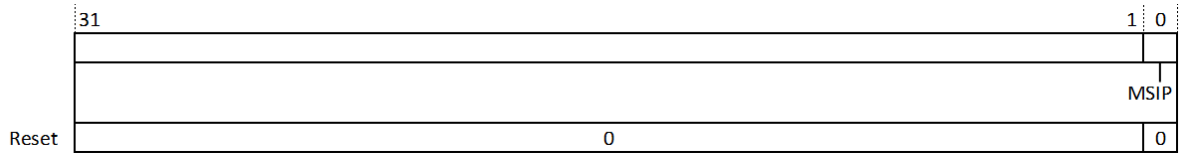


图 8.1: 机器模式软件中断配置寄存器（msip）

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，机器模式下可以对该位进行读写。

- 当 MSIP 位置 1，当前存在有效的机器模式软件中断请求；
- 当 MSIP 位置 0，当前不存在有效的机器模式软件中断请求。

8.3 计时器中断

备注

本小节适用于选配计数器资源的场景。

CLINT 可用于生成机器模式计时器中断。

E901 可选配 mtimel、mtimecmpl 寄存器。

该计时器中断需要搭配 E901 外部由系统设计实现的 32 位计数器使用。该系统计数器需要工作在 always-on 的电压域，复位后在每个时钟周期进行计数。在 E901 集成时需要将该 32 位系统计数器的值通过 pad_cpu_sys_cnt[31:0] 信号传入 E901 内部，值可通过 mtimel[31:0] 寄存器读取。

同时 E901 内部设计实现了一个 32 位的*机器模式计时器比较值寄存器（mtimecmpl）*，该寄存器可以通过地址字对齐访问的方式读写。

CLINT 通过比较 mtimecmpl[31:0] 与系统计数器 mtimel[31:0] 的当前值，确定是否产生计时器中断。

- 当 mtimecmpl[31:0] 大于系统计数器的值时不产生中断。
- 当 mtimecmpl[31:0] 小于或等于系统计数器的值时 CLINT 产生计时器中断。

软件可通过改写 mtimecmpl 的值来清除对应的计时器中断。

每组寄存器结构相同，其寄存器位分布和位定义如 图 8.2 和 图 8.3 所示。

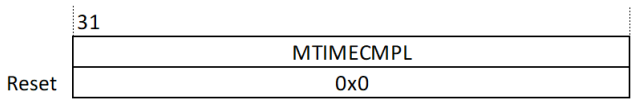


图 8.2: mtimecmpl-机器模式计时器中断比较值寄存器

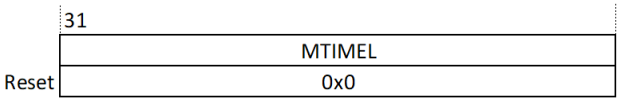


图 8.3: mtimel-机器模式计时器所用的系统计数器当前计数值

9 CLIC 中断控制器

核内局部中断控制器（以下简称 CLIC），仅用于对中断源进行采样，优先级仲裁和分发。CLIC 仲裁来源包括处理器各个模式下触发的中断。E901 实现的 CLIC 单元基本功能如下：

- 支持 RISC-V 标准 CLIC spec-0.8 版本
- 外部中断源数量最高可配置为 112 个，支持电平中断，脉冲中断，加上兼容 CLINT 的至多 16 个中断（目前仅实现机器模式软件中断和机器模式计时器中断），CLIC 共支持 128 个中断处理
- 中断优先级固定
- 每个中断目标拥有 4 个 memory-mapped 的控制寄存器
- 通过写相应中断源的控制寄存器可以配置此中断源的各个属性
- 支持 mscratchcswl 寄存器，不支持 mscratchcsw 寄存器

9.1 中断处理机制

9.1.1 中断仲裁

E901 实现固定优先级仲裁，中断的优先级固定为对应的中断向量号，向量号越大，优先级越高。

在 CLIC 中只有符合条件的中断源才会参与仲裁。需满足的条件如下：

- 中断源处于等待状态（IP =1）
- CLIC 中该中断使能位为 1（IE=1）

当 CLIC 中有多个中断处于等待状态时，CLIC 仲裁出优先级最高的中断。CLIC 中断优先级配置寄存器 (clicintctl) 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，则中断 ID 较大的优先处理。

CLIC 会将仲裁结果包括中断 ID，优先级，特权态，是否为矢量中断的信息传递给 CPU 流水线核心。其中，中断 ID 作为中断号进行处理，CLINT 中断对应中断号为 0~15，CLIC 外接的中断源中断号为 16~127。

9.1.2 中断请求与响应

当 CLIC 仲裁产生有效的中断请求，会向 CPU 发起中断请求。当 CPU 收到有效中断请求，根据不同的中断类型情况，CPU 会向 CLIC 发送中断响应消息。中断响应机制如下：

- 中断为电平中断时，无中断响应信号，需要中断服务程序里通过软件清除外部中断源。

- 中断为矢量模式边缘中断时，CPU 响应中断后，会发出一个响应信号，CLIC 收到该信号后，清除对应中断的中断等待位。
- 中断为非矢量模式边缘中断时，需要软件写 CLIC 对应 PENDING 寄存器 0 来清除 PENDING 中断。

9.2 CLIC 寄存器地址映射

CLIC 中断控制器占据 20KiB 内存空间，其地址映射如 表 9.1 所示。

其中，cliccfg 和 clicinfo 寄存器仅支持地址字对齐的访问。

表 9.1: CLIC 地址映射

地址	名称	类型	初始值	描述
0xE0800000	cliccfg	RW	0x1	CLIC 配置寄存器
0xE0800004	clicinfo	RO	详见*计时器中断*	CLIC 信息寄存器
0xE0801000+4*i	clicintip[i]	R or RW	0x0	中断源 i 等待寄存器
0xE0801001+4*i	clicintie[i]	RW	0x0	中断源 i 使能寄存器
0xE0801002+4*i	clicintattr[i]	RW	0x0	中断源 i 属性寄存器
0xE0801003+4*i	clicintctrl[i]	RW	0x0	中断源 i 控制寄存器

此处，i 为中断向量号。比如，使能软件中断（向量号为 3）和计时器中断（向量号为 7）时，需要分别使用 clicintie[3] 和 clicintie[7]。

9.3 CLIC 寄存器描述

9.3.1 CLIC 配置寄存器（cliccfg）

CLIC 有一个 8-bit 的全局配置寄存器 cliccfg，其中定义了支持的中断响应特权态，clicintctl[i] 的划分，以及是否支持硬件矢量中断。寄存器位分布和位定义如 图 9.1 所示。

	7	6 5	4 1	0
	-	nmbits	nlbits	nvbits
Reset	0	0	0	1

图 9.1: CLIC 配置寄存器 (cliccfg)

NMBITS-特权态有效位数：

clicintattr[i].MODE 中有效的位数。由于 E901 仅支持机器模式下处理中断，所以该位绑为 0。clicintattr[i].MODE 域中的值无论为何值均认为是机器模式响应中断。

NLBITS-中断优先级有效位数：

NLBITS 由硬件固定为 0x7。

NVBITS-硬件矢量中断实现标志位：

代表 CLIC 控制器是否支持矢量模式中断，该位恒为 1，表示支持硬件矢量模式中断。开启中断硬件矢量模式需要将中断对应的 clicintattr.SHV 置 1。

- 硬件矢量中断的服务程序入口地址采用硬件两级跳转的方式获取，首先在保存完处理器现场后 CPU 从 mtvt 加中断号偏移量（即中断 ID*4）的地址处取数据，该数据被认为是该中断 ID 的中断服务程序的入口地址，CPU 跳转到该地址去执行中断服务程序。
- 非硬件矢量中断的服务程序入口是 mtvec[31:6] << 6，CPU 跳转到该地址去处理中断。

9.3.2 CLIC 信息寄存器（clicinfo）

clicinfo 为一个只读寄存器，里面提供了 CLIC 的部分信息。寄存器位分布和位定义如 图 9.2 所示。

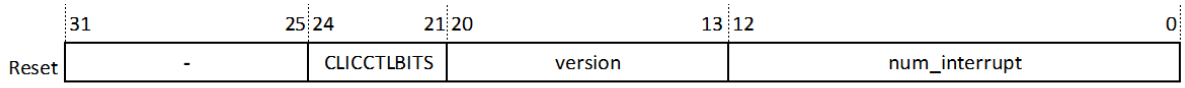


图 9.2: CLIC 信息寄存器 (clicinfo)

CLICCTLBITS-CLICINTCTL 有效位数：

clicintctl 寄存器内优先级有效位数。实现的有效位数在 clicintctl[i] 中左对齐。

VERSION-版本信息：

其中低 4 位为硬件实现的修改版本；高四位为 CLIC 架构版本信息。

num_interrupt-中断源数量：

代表该 CLIC 控制器硬件支持的中断源个数，至多 128 个。

9.3.3 中断等待寄存器（clicintip）

该寄存器最低位被置起表示对应的中断源有中断等待被处理。寄存器位分布和位定义如 图 9.3 所示。

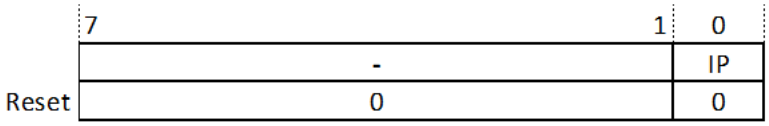


图 9.3: 中断等待寄存器 (clicintip)

IP-中断等待：

中断源是否有中断等待响应。该位在电平中断和边缘中断情况下有不同的置位与清除逻辑。

电平中断模式下，clicintip 为只读寄存器。更改 clicintip 的值需要通过直接对外部中断源进行操作来实现，外部中断源为高则 IP 为 1，外部中断源为低则 IP 为 0。

边缘中断模式下，clicintip 为可读可写寄存器，且带有自动清除 IP 位的功能。当中断配置为硬件矢量模式时，CPU 响应中断的同时会自动清除该中断的 IP 位；对于非硬件矢量模式的中断，建议软件通过写该位 0 来清除中断。

9.3.4 中断使能寄存器（clicintie）

该寄存器最低位被置起表示对应的中断源被使能，在满足条件的情况下 CPU 可以响应该中断。寄存器位分布和位定义如 图 9.4 所示。

	7 … 1	0
	-	IE
Reset	0	0

图 9.4: 中断使能配置寄存器 (clicintie)

IE-中断使能：

1：对应中断被使能；

0：对应中断未使能。

9.3.5 中断属性寄存器（clicintattr）

该寄存器用于配置不同的中断源的属性，包括了可响应中断的 CPU 特权态、中断的触发模式以及中断是否为硬件矢量模式。寄存器位分布和位定义如 图 9.5 所示。

	7 … 6	5 … 3	2 … 1	0
	mode	-	trig	shv
Reset	2'b11	0	0	0

图 9.5: 中断属性寄存器 (clicintattr)

MODE-中断特权态：

该域用于配置中断的特权态，由于 E901 仅支持机器模式下响应中断，所以该域被绑为 2’b11，代表机器模式中断。

TRIG-中断触发方式：

该域用于区分脉冲中断和电平中断。

当 TRIG[0] 为 0 时，代表为电平中断。

当 TRIG[0] 为 1 时，TRIG[1] 为 0 代表上升沿中断，TRIG[1] 为 1 代表下降沿中断。

SHV-矢量中断使能：

代表该中断是否为硬件矢量中断（Selective Hardware Vectoring）。

9.3.6 中断控制寄存器（clicintctl）

该寄存器用于表示每一个中断源参与仲裁的优先级，同时配合 cliccfg.NLBITS 产生给 CPU 的中断优先级。给 CPU 的中断优先级固定为 8 位。寄存器位分布和位定义如 图 9.6 所示。

E901 只支持固定优先级，此时寄存器 clicintctl 只读为对应的中断向量号，且向量号越大中断优先级越高，软件读该寄存器返回对应中断向量号的值。

	7 8-CLICINTCTLBITS	7-CLICINTCTLBITS 0
	int_ctl	hardware tied to 1
Reset	0	{(CLICINTCTLBITS'{1b'1}

图 9.6: 中断控制寄存器 (clicintctl)

10 调试接口

10.1 概述

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

E901 兼容 RISC-V External Debug Support Version 0.13.2 协议标准。对外调试接口支持两种配置：5 线 JTAG 接口（标准 JTAG5 协议）和 2 线 CJTAG 接口。

调试接口的主要特性如下：

- 支持 5 线 JTAG 和 2 线 CJTAG 两种配置
- 支持同步调试和异步调试，保证在极端恶劣情况下使处理器进入调试模式
- 支持软断点
- 可以设置多个内存断点
- 检查和设置 CPU 寄存器的值
- 检查和改变内存值
- 可进行指令单步执行或多步执行
- 快速下载程序
- 可在 CPU 复位之后进入调试模式

E901 的调试工作是调试软件，调试代理服务程序，调试器和调试接口一起配合完成的，调试接口在整个 CPU 调试环境中的位置如 图 10.1 所示。其中，调试软件和调试代理服务程序通过网络互联，调试代理服务程序与调试器通过 USB 连接，调试器与 CPU 的调试接口以 CJTAG 模式或 JTAG 模式通信。

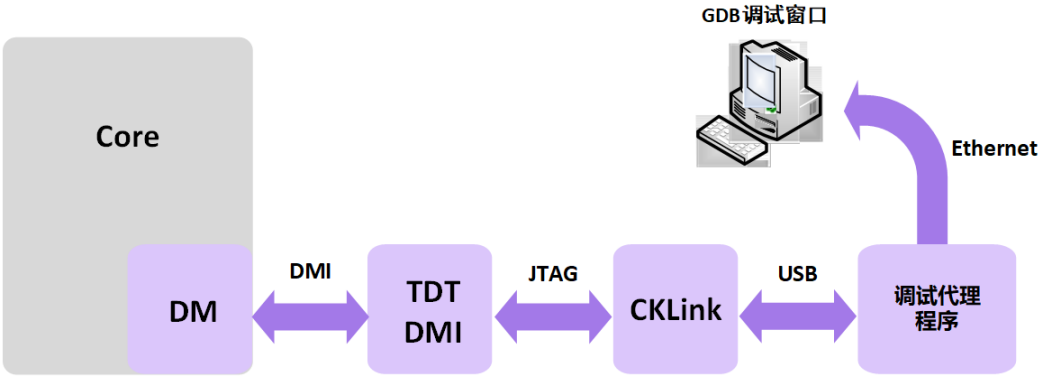


图 10.1: 调试接口在整个 CPU 调试环境中的位置

10.2　DM 寄存器

下表所示为 E901 DM 模块实现的寄存器列表，除了标准定义寄存器外，E901 还在 DMI Address 编码域上扩展实现了部分 DM 寄存器。

表 10.1: DM 寄存器映射及描述

地址	寄存器名称	描述
0x4	data0	抽象 DATA0
0x10	dmcontrol	DM 控制寄存器
0x11	dmstatus	DM 状态寄存器
0x15	hawindow	Hart 阵列窗口
0x16	abstractcs	抽象控制和状态寄存器
0x17	command	抽象命令寄存器
0x18	abstractauto	抽象命令自动执行寄存器
0x1D	nextdm	下一个 DM 基址寄存器
0x20-0x2F	progbuf0-1F	可编程 Buffer0-15
0x32	dmcs2	DM 控制和状态寄存器 2
0x40	haltsum0	HALT 总览寄存器 0
玄铁扩展寄存器		
0x1F	itr	指令传输寄存器
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
0x7F	compid	组件 ID 寄存器

备注

dmcs2 是新版 RISC-V Debug（Version 1.0.0-STABLE）的功能。

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 LSU 执行，省去取址等操作，更具鲁棒性。

扩展控制和状态寄存器 customcs

用于描述玄铁 E901 扩展的抽象命令实现和扩展。寄存器描述如下：

表 10.2: customcs 寄存器描述

位域	名称	描述
31:29	custcmderr	在使用 CUSTOMCMD 执行命令时的错误状态： 0：没有错误 1：命令不支持
28:25	cusbufcnt	实现的 Buffer 的数
24:28	-	-
17	cuscmdbusy	使用 CUSTOMCMD 执行命令的状态： 0：没有执行命令 1：正在执行命令
16	-	-
15:0	coredbginfo	用于表示支持的核内调试资源情况。

扩展命令寄存器 customcmd

该寄存器用于执行扩展命令的寄存器，如果输入的命令不支持，则 CUSTOMCS.custcmderr 置为 1。

表 10.3: customcmd 寄存器描述

位域	名称	描述
31:24	type	指明 CUSTOM 的命令类型，支持的有： 0：向当前选中的核发起异步调试请求； 1：寄存器内部移动； 2：PC 采样。命令执行后，CPU 执行的最近一次跳转指令的下一条 PC 的值被拷贝到 DATA 寄存器； 其他：保留。
23：0	-	-

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。

10.3　资源配置

为了方便用户选择，E901 提供了以下调试资源配置组合供用户选择：

- 最小配置（Minimal）：2 个 Program Buffer 且实现隐式的 EBREAK；1 个硬断点。
- 典型配置（Typical）：2 个 Program Buffer 且实现隐式的 EBREAK；4 个硬断点。

10.4　同步调试与异步调试

10.4.1　同步调试

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

10.4.2　异步调试

调试器向 DM 模块中的 custom command 寄存器（属于玄铁调试系统自定义寄存器）的 type 域写 0 时，表示 debugger 向所选的核发出异步调试请求，则 DM 会向核内发出拉高的异步调试请求信号。与同步调试不同，当核内 RTU 模块接收到异步调试请求后，会立即进入调试模式。该功能为玄铁处理器调试系统扩展功能。调试器发起同步调试请求后，会等当前指令退休后再进入调试模式。当 CPU 卡死（当前指令一直无法退休时），就无法响应同步调试请求，调试器等待几秒发现同步调试请求无法被响应时，就会自动发起异步调试请求，尝试在 CPU 卡死时也能让 CPU 进入调试模式。关于上述调试相关寄存器，请参考 RISC-V 官方文档 RISC-V External Debug Support V0.13.2 。

11 功耗管理

E901 设计实现了 RV32E 的 WFI 指令用于使处理器从正常工作模式转入低功耗模式。在低功耗模式下，E901 的内部门控时钟管理单元会将绝大多数的寄存器时钟关闭，而跟处理器唤醒功能相关的逻辑部分的时钟不会被关闭。

在低功耗模式下，E901 不会向总线发起数据传输请求，内部流水线停顿。

11.1 低功耗模式

E901 扩展实现了 mexstatus 寄存器的 LPMD 域来指示通过 WFI 指令进入的低功耗模式类型，深睡眠和浅睡眠，并通过 E901 顶层的输出信号 sysio_pad_lpmd_b[1:0] 指示给 SoC。具体寄存器定义请参考*扩展异常状态寄存器（mexstatus）*。

E901 虽然扩展实现了深睡眠和浅睡眠两种睡眠模式，但是处理器核对两种睡眠模式的低功耗处理相同。SoC 设计人员可根据 CPU 顶层的指示信号来决定是否实现不同的低功耗策略。

11.2 低功耗唤醒

E901 的低功耗唤醒支持如下请求类型：

- 调试请求
- 中断请求
- 外部事件

E901 扩展实现了 mexstatus 寄存器的 WFE 域来指示唤醒 CPU 的请求类型。

- 当 WFE = 1 时，上述所有请求都可以唤醒处于低功耗模式的 CPU。
- 当 WFE=0 时，除了外部事件，其他请求均可以唤醒 CPU。
- E901 不支持 NMI 中断，只要出现中断即可唤醒 CPU。

在调试请求唤醒 CPU 后，CPU 会进入调试模式。在使用中断请求唤醒 CPU 后，CPU 会响应中断。

在 CPU 被唤醒后，会驱动它的顶层输出信号 sysio_pad_lpmd_b[1:0] 从 2'b00 变为 2'b11。

12 程序示例

本章主要介绍多种程序示例，包含：PMP 设置示例、高速缓存设置示例、中断使能初始化示例、通用寄存器初始化示例、堆栈指针初始化示例和异常与中断服务程序入口地址设置示例。

12.1 中断使能初始化设置示例

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

```
//配置中断优先级有效位

li t1, 0x6 //nlbits = 3
li t2, 0xe0800000 //cliccfg 寄存器地址
sw t1, 0(t2) //具体中断优先级位宽最大值可查看 CLICINFO.CLICCTLBITS 域

//设置中断类型

li t1, 0x7f010100

//[31:24] 为 CLICINTCTL，优先级为 0x7f,[23:16] 为 CLICINTATTR，矢量电平中断
//[15:8] 为 CLICINTIE，使能中断，[7:0] 为 CLICINTIP

li t2, 0xe0801000 //0 号中断配置基址
sw t1, 64(t2) //配置 16 号中断

//打开全局中断使能位 mstatus.mie

li t1, 0x8
csrrs t3, mstatus, x1
```

12.2 通用寄存器初始化示例

```
//初始化通用寄存器 x0~x15。

li x1, 0
li x2, 0
li x3, 0
li x4, 0
li x5, 0
li x6, 0
li x7, 0
li x8, 0
li x9, 0
li x10, 0
li x11, 0
li x12, 0
li x13, 0
li x14, 0
li x15, 0
```

12.3 堆栈指针初始化示例

堆栈指针的设置如下所示。

```
li x2,  0x01000000 //设置堆栈指针
```

12.4 异常和中断服务程序入口地址设置示例

硬件选配 mtvec/mtvt/jvt 只读时，需要由链接脚本将程序中对应 table 地址指定为硬件固定的对应的值。

异常和中断服务程序的入口地址设置分两个步骤：

步骤 1：

设置异常向量表基地址和模式，写入 mtvec，E901 模式位固定为 2'b11

步骤 2：

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

// if clicintattr[i].shv = 0 clic direct mode（非矢量模式）：

异常和中断入口地址为 mtvec[31:6] << 6

//if clicintattr[i].shv = 1 clic vector mode （矢量模式）：

异常入口地址为 mtvec[31:6] << 6, 中断入口地址 MEM[mtvt[31:0]+ 4* 中断 ID]

示例如下：

```
li t3, (trap_handler) //trap_handler 为 2^6 地址对齐
csrw mtvec, t3 //初始化 mtvec
li t3, (vector_table) //vector_table 为 2^6 地址对齐
addi t3, t3, 64
csrw mtvt, t3 //初始化 mtvt

.align 6
trap_handler:
addi sp, sp, -48
sw t0, 44(sp)
sw t1, 40(sp)
sw t2, 36(sp)
sw t3, 32(sp)
sw t4, 28(sp)
sw t5, 24(sp)
sw t6, 20(sp)
sw ra, 16(sp)
csrr t0, mcause
sw t0, 12(sp)
csrr t1, mepc
sw t1, 8(sp)
csrr t2, mstatus
sw t2, 4(sp)
andi t1, t0, 0xfff //获取 cause number
srli t0, t0, 0x1b
andi t0, t0, 0x10 //得到 mcause 的 bit[31], 用于判断是否为中断
add t0, t0, t1 //cause+16, 空出低 16 个异常向量的空间
slli t0, t0, 0x2
la t1, vector_table //存放异常和中断服务程序的入口地址
add t0, t0, t1
lw t1, 0(t0)
//handle with the exception or non vector int
jalr t1
//recovery the cpu field
lw t2, 4(sp)
csrw mstatus, t2
lw t1, 8(sp)
csrw mepc, t1
lw t0, 12(sp)
csrw mcause, t0
```

(续下页)

(接上页)

```
lw ra, 16(sp)
lw t6, 20(sp)
lw t5, 24(sp)
lw t4, 28(sp)
lw t3, 32(sp)
lw t2, 36(sp)
lw t1, 40(sp)
lw t0, 44(sp)
addi sp, sp, 48
mret

.align 6
vector_table:
.long 0x0 //reserved
.long INST_FETCH_ERROR_HANDLER //1 号取指令访问异常处理函数
.long ILLEGAL_INST_ERROR_HANDLER //2 号非法指令异常处理函数
......
.long MACHINE_ECALL_HANDLER //11 号机器模式环境调用异常处理函数
.rept 5
.long 0x0
.endr
.rept 3
.long 0x0 //0-2 号 clint 中断未实现
.endr
.long MSOFT_INT_HANDLER //3 号机器模式软件中断处理函数
.rept 3
.long 0x0 //4-6 号 clint 中断未实现
.endr
.long MTIME_INT_HANDLER //7 号机器模式计时器中断处理函数
.rept 3
.long 0x0 //8-10 号 clint 中断未实现
.endr

.long CLIC_INT0_HANDLER

//通过 clic 接入的 0 号中断，即 pad_clic_int_vld[0] 接入
//其在 mcause 中 cause 的值为 16。

.long CLIC_INT1_HANDLER

 //通过 clic 接入的 1 号中断，所以通过 pad_clic_int_vld
```

(续下页)

(接上页)

```
//最多可实现接入 112 个外部中断。
```

在上述初始化中，对于同一中断而言，矢量和非矢量的中断服务程序地址共用了同一入口并跳转执行，所不同的是矢量中断模式下是 CPU 硬件自动获取该中断服务程序入口并跳转执行，而在非矢量中断模式下是由 CPU 执行指令通过软件方式模拟这一行为。

当然，中断服务程序的现场保存和恢复可以在各个中断服务程序内部完成，在中断服务程序声明时可以添加 _attribute_((isr)) 来修饰，这样编译器会在编译时自动插入现场保存和恢复，以及中断返回的指令。

13 附录 A 标准指令术语

E901 实现了 RV32EC[_Zmmul] 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。

13.1　附录 E 指令术语

以下是对 E901 实现的 RV32E 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 Zca 指令术语*。

13.1.1　ADD—有符号加法指令

语法：

add rd, rs1, rs2

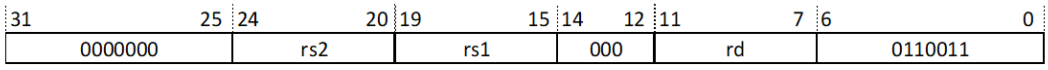
操作：

rd ← rs1 + rs2

异常：

无

指令格式：



13.1.2　ADDI—有符号立即数加法指令

语法：

addi rd, rs1, imm12

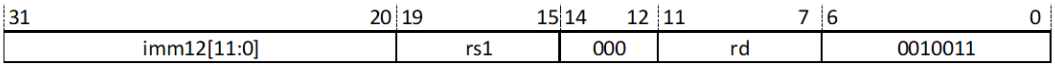
操作：

rd ← rs1 + sign_extend(imm12)

异常：

无

指令格式：



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

13.1.3　AND——按位与指令

语法：

and rd, rs1, rs2

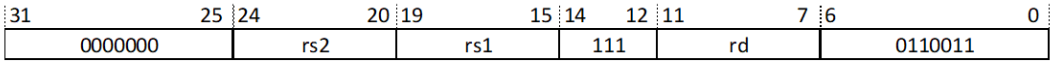
操作：

rd ← rs1 & rs2

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

13.1.4　ANDI——立即数按位与指令

语法：

andi rd, rs1, imm12

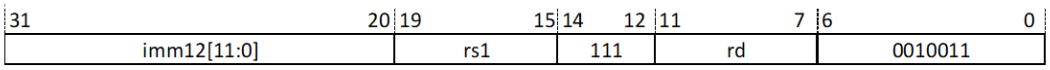
操作：

rd ← rs1 & sign_extend(imm12)

异常：

无

指令格式：



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

13.1.5　AUIPC——PC 高位立即数加法指令

语法：

auipc rd, imm20

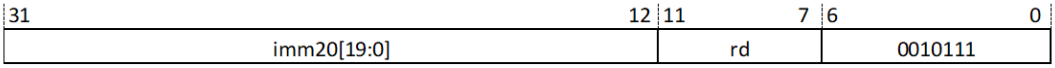
操作：

rd ← current pc + imm20 << 12

异常：

无

指令格式：



13.1.6　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

异常：

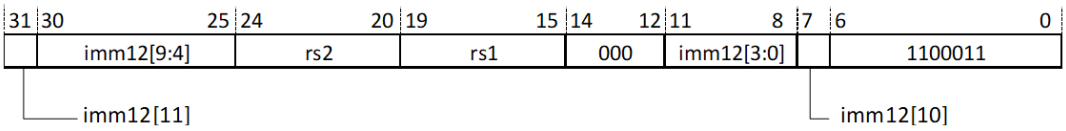
无

说明：

汇编器根据 label 算出 imm12

指令跳转范围为 ±4KiB 地址空间

指令格式：



13.1.7　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

异常：

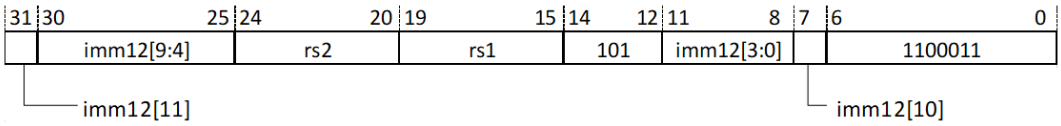
无

说明：

汇编器根据 label 算出 imm12

指令跳转范围为 ±4KiB 地址空间

指令格式：



13.1.8　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/U mode

异常：

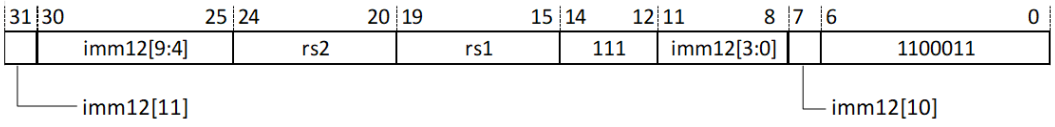
无

说明：

汇编器根据 label 算出 imm12

指令跳转范围为 ±4KiB 地址空间

指令格式：



13.1.9　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

异常：

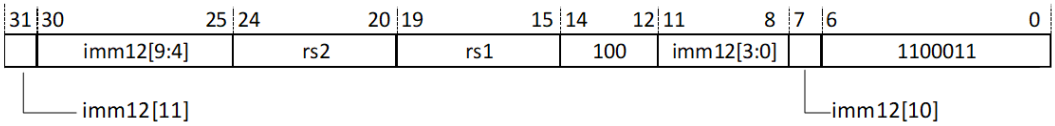
无

说明：

汇编器根据 label 算出 imm12

指令跳转范围为 ±4KiB 地址空间

指令格式：



13.1.10　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

异常：

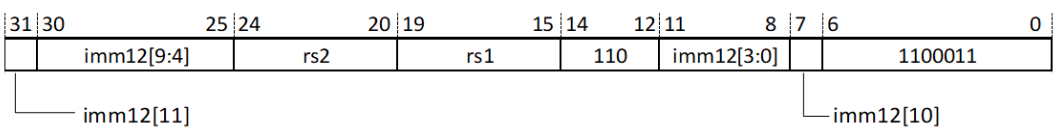
无

说明：

汇编器根据 label 算出 imm12

指令跳转范围为 ±4KiB 地址空间

指令格式：



13.1.11　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

异常：

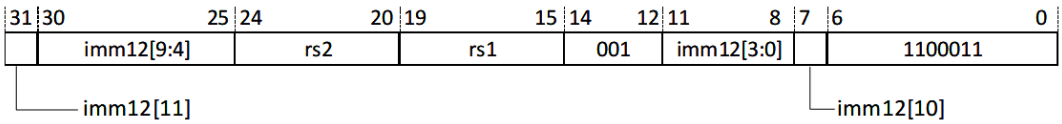
无

说明：

汇编器根据 label 算出 imm12

指令跳转范围为 ±4KiB 地址空间

指令格式：



13.1.12　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)

异常：

非法指令异常

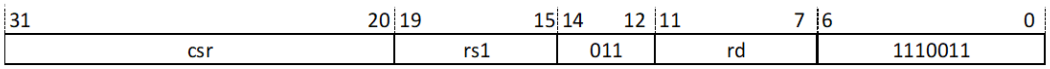
说明：

各权限下允许访问的控制寄存器不同，具体请参考控制寄存器章节。

当 rs1=x0 时，该指令不产生写操作，不会产生写行为引发的异常。

该指令在 The RISC-V Instruction Set Manual Volume I: Unprivileged Architecture 中被归类为 Zicsr 扩展。

指令格式：



13.1.13　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)

异常：

非法指令异常

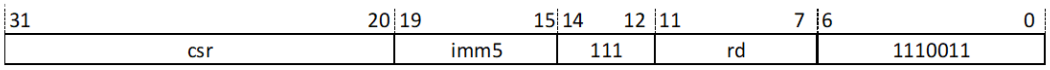
说明：

各权限下允许访问的控制寄存器不同，具体请参考控制寄存器章节。

当 rs1=x0 时，该指令不产生写操作，不会产生写行为引发的异常。

该指令在 The RISC-V Instruction Set Manual Volume I: Unprivileged Architecture 中被归类为 Zicsr 扩展。

指令格式：



13.1.14　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

异常：

非法指令异常

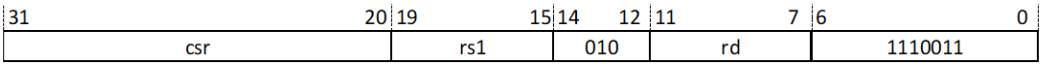
说明：

各权限下允许访问的控制寄存器不同，具体请参考控制寄存器章节。

当 rs1=x0 时，该指令不产生写操作，不会产生写行为引发的异常。

该指令在 The RISC-V Instruction Set Manual Volume I: Unprivileged Architecture 中被归类为 Zicsr 扩展。

指令格式：



13.1.15　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)

异常：

非法指令异常

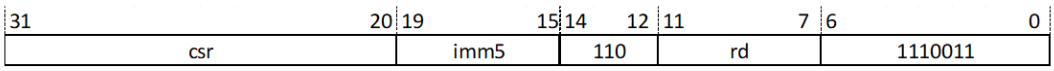
说明：

各权限下允许访问的控制寄存器不同，具体请参考控制寄存器章节。

当 rs1=x0 时，该指令不产生写操作，不会产生写行为引发的异常。

该指令在 The RISC-V Instruction Set Manual Volume I: Unprivileged Architecture 中被归类为 Zicsr 扩展。

指令格式：



13.1.16　CSRRW—控制寄存器读写传送指令

语法：

csrrw rd, csr, rs1

操作：

rd ← csr

csr ← rs1

异常：

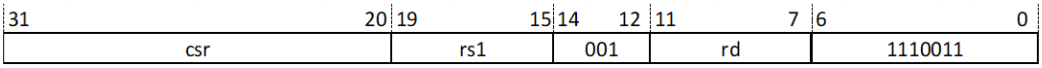
非法指令异常

说明：

各权限下允许访问的控制寄存器不同，具体请参考控制寄存器章节。

该指令在 The RISC-V Instruction Set Manual Volume I: Unprivileged Architecture 中被归类为 Zicsr 扩展。

指令格式：



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

13.1.17　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[4:0] ← imm5

csr[63:5] ← csr[63:5]

异常：

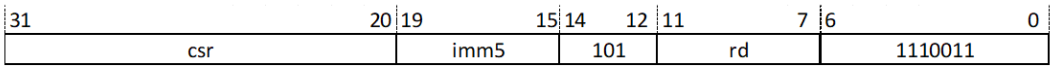
非法指令异常

说明：

各权限下允许访问的控制寄存器不同，具体请参考控制寄存器章节。

该指令在 The RISC-V Instruction Set Manual Volume I: Unprivileged Architecture 中被归类为 Zicsr 扩展。

指令格式：



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

13.1.18　EBREAK——断点指令

语法：

ebreak

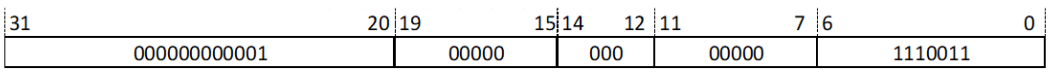
操作：

产生断点异常或者进入调试模式

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

13.1.19　ECALL——环境异常指令

语法：

ecall

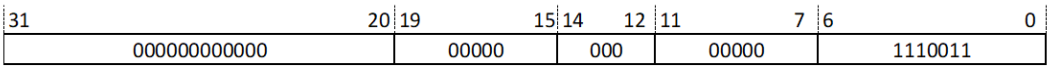
操作：

产生环境异常

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

13.1.20　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。

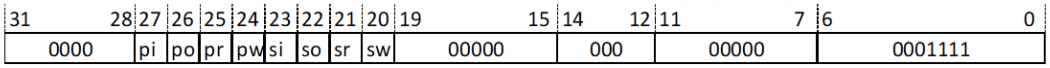
异常：

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

13.1.21　FENCE.I—指令流同步指令

语法：

fence.i

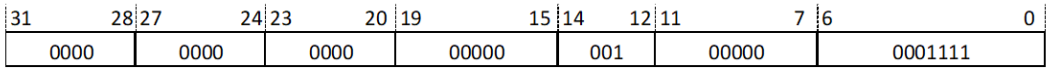
操作：

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。

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

13.1.22　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 ← current pc + 4

异常：

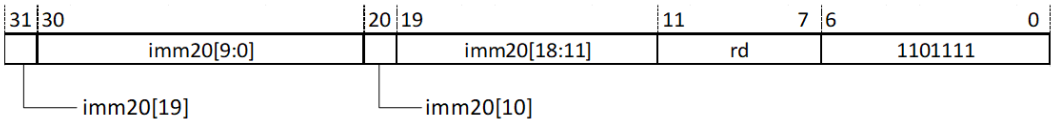
无

说明：

汇编器根据 label 算出 imm20

指令跳转范围为 ±1MiB 地址空间

指令格式：



13.1.23　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)) & 32'hfffffffe

rd ← current pc + 4

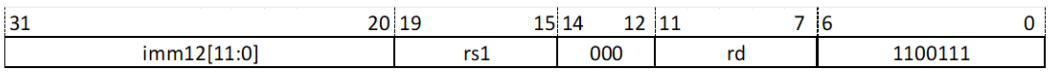
异常：

无

说明：

指令跳转范围为全部 4GiB 地址空间

指令格式：



13.1.24　LB——有符号扩展字节加载指令

语法：

lb rd, imm12(rs1)

操作：

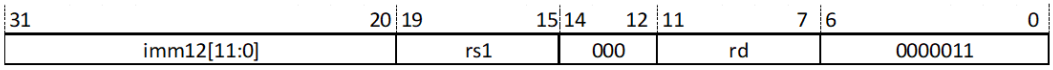
address ← rs1+sign_extend(imm12)

rd ← sign_extend(mem[(address+7):address])

异常：

加载指令非对齐访问异常、加载指令访问错误异常

指令格式：



13.1.25　LBU——无符号扩展字节加载指令

语法：

lbu rd, imm12(rs1)

操作：

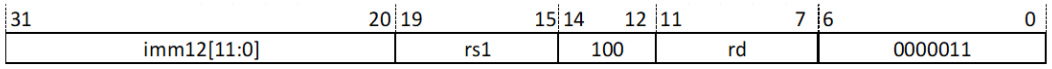
address ← rs1+sign_extend(imm12)

rd ← zero_extend(mem[(address+7):address])

异常：

加载指令非对齐访问异常、加载指令访问错误异常

指令格式：



13.1.26　LH——有符号扩展半字加载指令

语法：

lh rd, imm12(rs1)

操作：

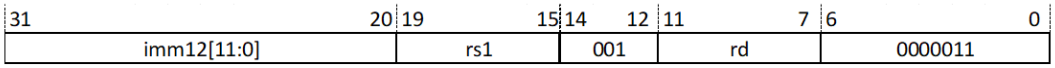
address ← rs1+sign_extend(imm12)

rd ← sign_extend(mem[(address+15):address])

异常：

加载指令非对齐访问异常、加载指令访问错误异常

指令格式：



13.1.27　LHU——无符号扩展半字加载指令

语法：

lhu rd, imm12(rs1)

操作：

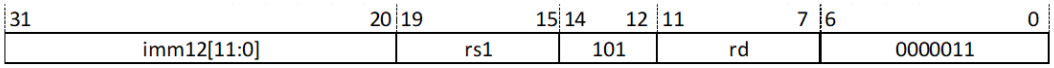
address ← rs1+sign_extend(imm12)

rd ← zero_extend(mem[(address+15):address])

异常：

加载指令非对齐访问异常、加载指令访问错误异常

指令格式：



13.1.28　LUI——高位立即数装载指令

语法：

lui rd, imm20

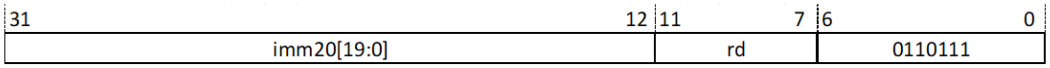
操作：

rd ← imm20 << 12

异常：

无

指令格式：



13.1.29　LW——字加载指令

语法：

lw rd, imm12(rs1)

操作：

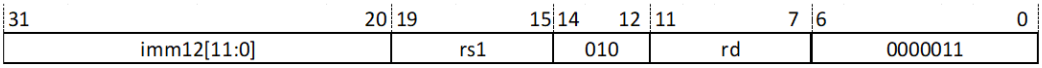
address ← rs1+sign_extend(imm12)

rd ← mem[(address+31):address]

异常：

加载指令非对齐访问异常、加载指令访问错误异常

指令格式：



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

13.1.30　MRET——机器模式异常返回指令

语法：

mret

操作：

next pc ← mepc

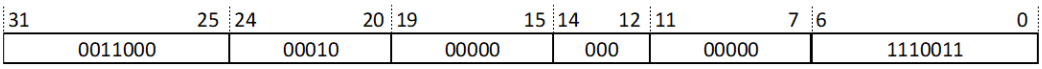
mstatus.mie ← mstatus.mpie

mstatus.mpie ← 1

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

13.1.31　OR——按位或指令

语法：

or rd, rs1, rs2

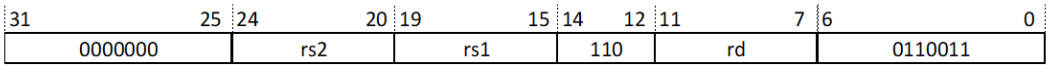
操作：

rd ← rs1 | rs2

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

13.1.32　ORI——立即数按位或指令

语法：

ori rd, rs1, imm12

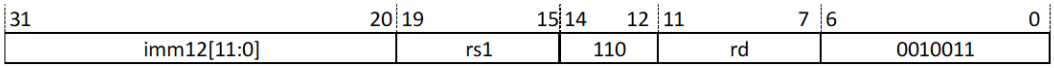
操作：

rd ← rs1 | sign_extend(imm12)

异常：

无

指令格式：



13.1.33　SB——字节存储指令

语法：

sb rs2, imm12(rs1)

操作：

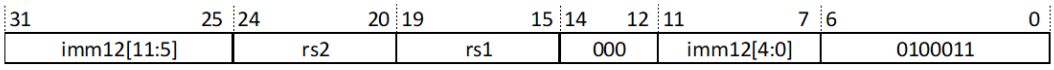
address ← rs1+sign_extend(imm12)

mem[(address+7):address] ← rs2[7:0]

异常：

存储指令非对齐访问异常、存储指令访问错误异常

指令格式：



13.1.34　SH——半字存储指令

语法：

sh rs2, imm12(rs1)

操作：

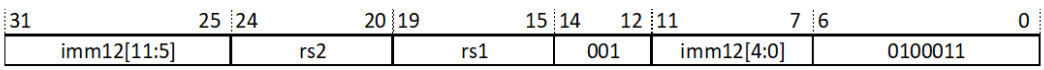
address ← rs1+sign_extend(imm12)

mem[(address+15):address] ← rs2[15:0]

异常：

存储指令非对齐访问异常、存储指令访问错误异常

指令格式：



13.1.35　SLL——逻辑左移指令

语法：

sll rd, rs1, rs2

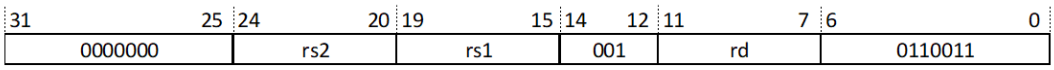
操作：

rd ← rs1 << rs2[4:0]

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

13.1.36　SLLI—立即数逻辑左移指令

语法：

slli rd, rs1, shamt5

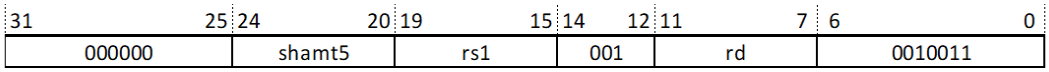
操作：

rd ← rs1 << shamt5

异常：

无

指令格式：



31 25	24 20	19 15	14 12	11 7	6 0
000000	shamt5	rs1	001	rd	0010011

13.1.37　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

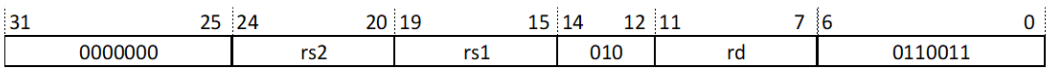
else

　　rd ← 0

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

13.1.38　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

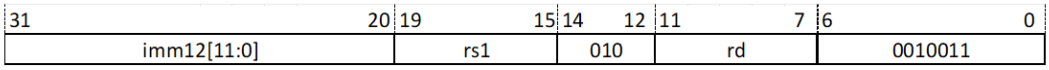
else

rd ← 0

异常：

无

指令格式：



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

13.1.39　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

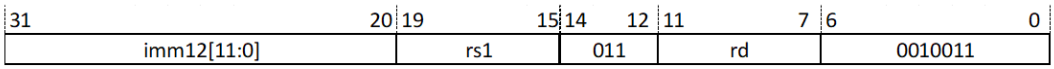
else

rd ← 0

异常：

无

指令格式：



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

13.1.40　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

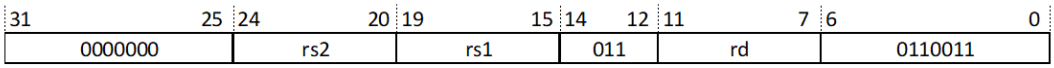
else

rd ← 0

异常：

无

指令格式：



13.1.41　SRA—算术右移指令

语法：

sra rd, rs1, rs2

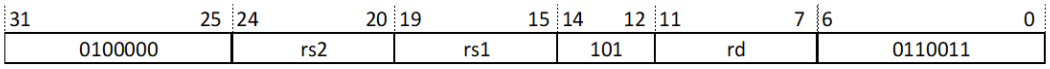
操作：

rd ← rs1 >>> rs2[4:0]

异常：

无

指令格式：



13.1.42　SRAI—立即数算术右移指令

语法：

srai rd, rs1, shamt5

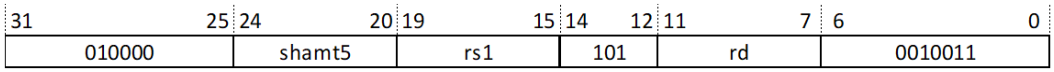
操作：

rd ← rs1 >>> shamt5

异常：

无

指令格式：



13.1.43　SRL—逻辑右移指令

语法：

srl rd, rs1, rs2

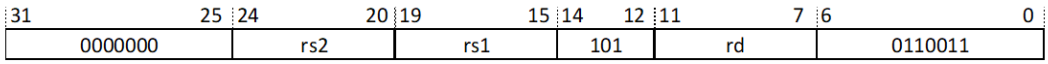
操作：

rd ← rs1 >> rs2[4:0]

异常：

无

指令格式：



13.1.44　SRLI—立即数逻辑右移指令

语法：

srli rd, rs1, shamt5

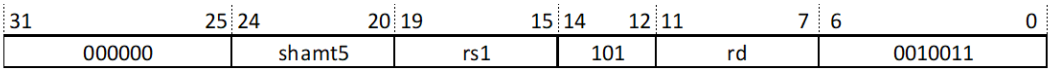
操作：

rd ← rs1 >> shamt5

异常：

无

指令格式：



13.1.45　SUB—有符号减法指令

语法：

sub rd, rs1, rs2

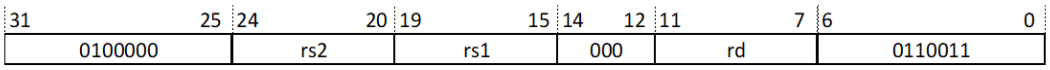
操作：

rd ← rs1 - rs2

异常：

无

指令格式：



13.1.46　SW—字存储指令

语法：

sw rs2, imm12(rs1)

操作：

address ← rs1+sign_extend(imm12)

mem[(address+31):address] ← rs2[31:0]

异常：

存储指令非对齐访问异常、存储指令访问错误异常

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

13.1.47　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

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

13.1.48　XOR——按位异或指令

语法：

xor rd, rs1, rs2

操作：

rd ← rs1 ^ rs2

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

13.1.49　XORI——立即数按位异或指令

语法：

xori rd, rs1, imm12

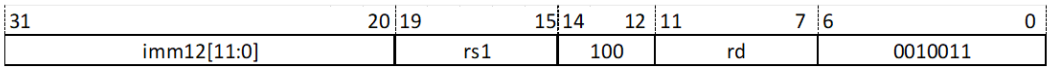
操作：

rd ← rs1 & sign_extend(imm12)

异常：

无

指令格式：



13.2　附录 Zmmul 扩展指令术语

以下是对 E901 实现的 RV32 Zmmul 扩展指令集的具体描述，本节指令位宽为 32 位，指令按英文字母顺序排列。

13.2.1　MUL—有符号乘法指令

语法：

mul rd, rs1, rs2

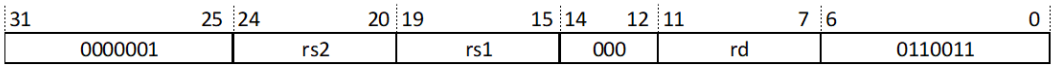
操作：

rd ← (rs1 * rs2)[31:0]

异常：

无

指令格式：



13.2.2　MULH—有符号乘法取高位指令

语法：

mulh rd, rs1, rs2

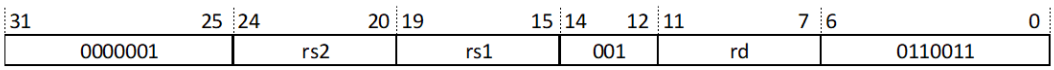
操作：

rd ← (rs1 * rs2)[63:32]

异常：

无

指令格式：



13.2.3　MULHSU—有符号无符号乘法取高位指令

语法：

mulusu rd, rs1, rs2

操作：

rd ← (rs1 * rs2)[63:32]

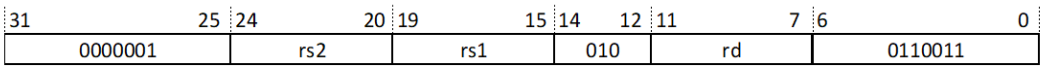
异常：

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

13.2.4　MULHU—无符号乘法取高位指令

语法：

mulhu rd, rs1, rs2

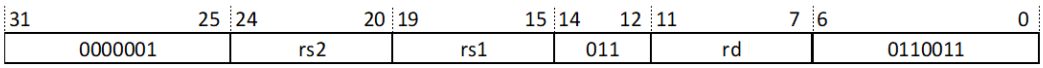
操作：

rd ← (rs1 * rs2)[63:32]

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

13.3　附录 Zca 指令术语

以下是对 E901 实现的 Zca 指令集的具体描述，每条指令位宽为 16 位，指令按英文字母顺序排列。

13.3.1　C.ADD—有符号加法指令

语法：

c.add rd, rs2

操作：

rd ← rs1 + rs2

异常：

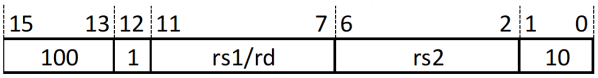
无

说明：

rs1 = rd != 0

rs2 ! = 0

指令格式：



13.3.2　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)

异常：

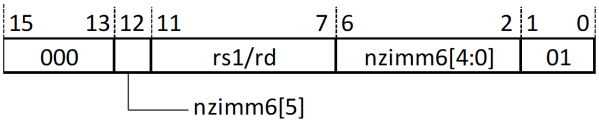
无

说明：

rs1 = rd != 0

nzimm6!=0

指令格式：



13.3.3　C.ADDI4SPN—堆栈指针有符号加法指令

语法：

c.addi4spn rd, sp, nzuimm8 << 2

操作：

rd ← sp + zero_extend(nzuimm8 << 2)

异常：

无

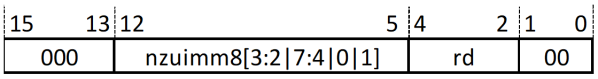
说明：

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:



13.3.4　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)

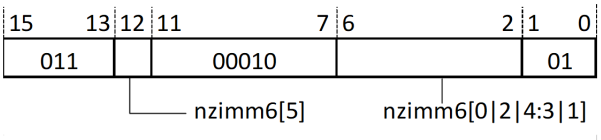
异常:

无

说明:

nzimm6 != 0

指令格式:



13.3.5　C.AND—按位与指令

语法:

c.and rd, rs2

操作:

rd ← rs1 & rs2

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

13.3.6　C.ANDI—立即数按位与指令

语法：

c.andi rd, imm6

操作：

rd ← rs1 & sign_extend(imm6)

异常：

无

说明：

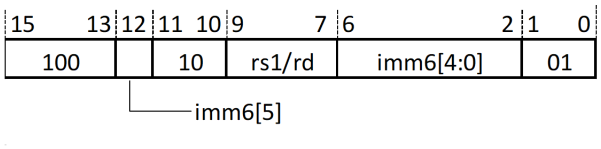
rs1 = rd

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：



13.3.7　C.BEQZ—等于零分支指令

语法：

c.beqz rs1, label

操作：

```
if (rs1 == 0)
        next pc = current pc + sign_extended(imm8) << 1;
else
        next pc = current pc + 2;
```

异常：

无

说明：

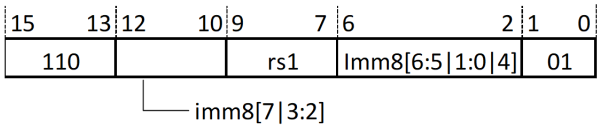
rs1 编码代表寄存器如下:

- 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：



13.3.8　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 + sign_extended(imm8) << 1;
else
        next pc = current pc + 2;
```

异常：

无

说明：

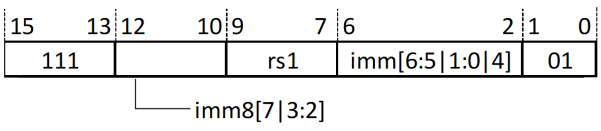
rs1 编码代表寄存器如下:

- 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：



13.3.9　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

异常：

断点异常

指令格式：

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

13.3.10　C.J—无条件跳转指令

语法：

c.j label

操作：

next pc ← current pc + sign_extend(imm << 1);

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

13.3.11　C.JAL—无条件跳转子程序指令

语法：

c.jal label

操作：

next pc ← current pc + sign_extend(imm << 1);

x1 ← current pc + 2;

异常：

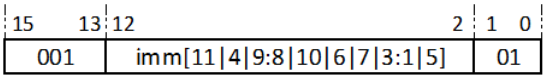
无

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
001	imm[11|4|9:8|10|6|7|3:1|5]	01

13.3.12　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;

异常：

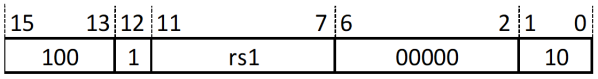
无

说明：

rs1 != 0

指令跳转范围是全部 4GB 地址空间。

指令格式：



15 13	12	11 7	6 2	1 0
100	1	rs1	00000	10

13.3.13　C.JR——寄存器跳转指令

语法：

c.jr rs1

操作：

next pc = rs1;

异常：

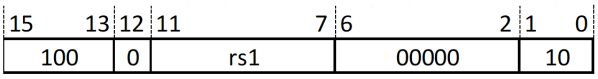
无

说明：

rs1 != 0

指令跳转范围是全部 4GB 地址空间。

指令格式：



13.3.14　C.LI—立即数传送指令

语法：

c.li rd, imm6

操作：

rd ← sign_extend(imm6)

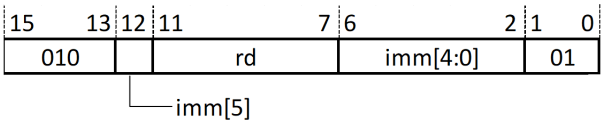
异常：

无

说明：

rd != 0

指令格式：



13.3.15　C.LUI—高位立即数传送指令

语法：

c.lui rd, nzimm6

操作：

rd ← sign_extend(nzimm6 << 12)

异常：

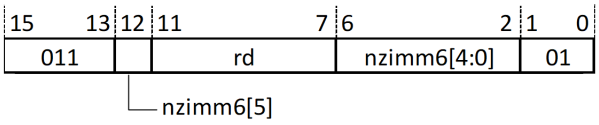
无

说明：

rd != 0 & rd != 2

nzimm6 != 0

指令格式：



13.3.16　C.LW—字加载指令

语法：

c.lw rd, uimm5 << 2(rs1)

操作：

address ← rs1+ zero_extend(uimm5 << 2)

rd ← mem[address+31:address]

异常：

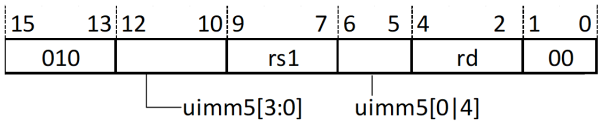
加载指令非对齐访问异常、加载指令访问错误异常

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：



13.3.17　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)

rd ← mem[address+31:address]

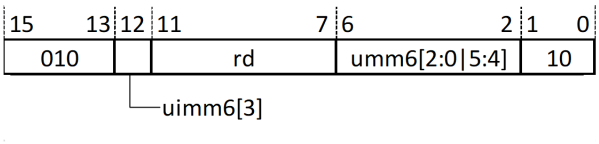
异常：

加载指令非对齐访问异常、加载指令访问错误异常

说明：

rd != 0

指令格式：



13.3.18　C.MV——数据传送指令

语法：

c.mv rd, rs2

操作：

rd ← rs2;

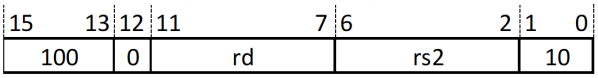
异常：

无

说明：

rs2 != 0, rd !=0

指令格式：



13.3.19　C.NOP——空指令

语法：

c.nop

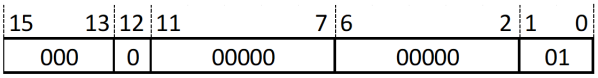
操作：

无操作

异常：

无

指令格式：



13.3.20　C.OR——按位或指令

语法：

c.or rd, rs2

操作：

rd ← rs1 | rs2

异常：

无

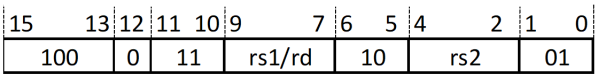
说明：

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：



13.3.21　C.SLLI—立即数逻辑左移指令

语法：

c.slli rd, shamt6

操作：

rd ← rs1 << shamt6

异常：

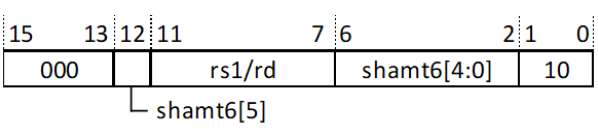
无

说明：

rs1==rd

rd/rs1 != 0，shamt6 != 0，shamt6[5] = 0

指令格式：



13.3.22　C.SRAI—立即数算术右移指令

语法：

c.srli rd, shamt6

操作：

rd ← rs1 >> shamt6

异常：

无

说明：

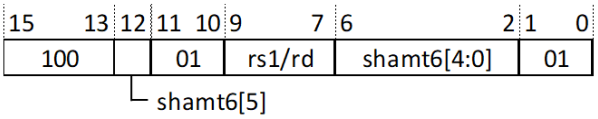
shamt6 != 0，shamt6[5] = 0

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：



13.3.23　C.SRLI—立即数逻辑右移指令

语法：

c.srli rd, shamt6

操作：

rd ← rs1 >> shamt6

异常：

无

说明：

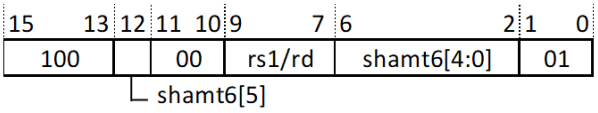
shamt6 != 0，shamt6[5] = 0

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：



13.3.24　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+31:address] ← rs2

异常：

存储指令非对齐访问异常、存储指令访问错误异常

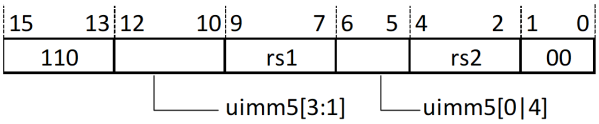
说明：

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：



13.3.25　C.SWSP—字堆栈存储指令

语法：

c.swsp rs2, uimm6 << 2(sp)

操作：

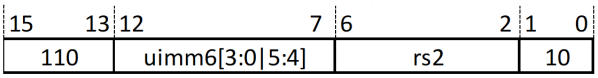
address ← sp+ zero_extend(uimm6 << 2)

mem[address+31:address] ← rs2

异常：

存储指令非对齐访问异常、存储指令访问错误异常

指令格式：



13.3.26　C.SUB—有符号减法指令

语法：

c.sub rd, rs2

操作：

rd ← rs1 - rs2

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

13.3.27　C.XOR—按位异或指令

语法：

c.xor rd, rs2

操作：

rd ← rs1 ^ rs2

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

13.4　附录伪指令列表

RISC-V 实现了一系列的伪指令，在此 表 13.1 列出仅供参考，按英文字母顺序排列。

表 13.1: 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 4KiB-4GiB 空间的函数
csrc csr, rs	csrrc x0, csr, rs	清除控制寄存器中对应比特
csrci csr, imm	csrrci x0, csr, imm	清除控制寄存器低 5 位中对应比特
csrs csr, rs	csrrs x0, csr, rs	置位控制寄存器中对应比特
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
fence	fence iorw, iorw	存储和外设同步指令
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} rd, symbol, rt	auipc rt, symbol[31:12] l{b|h|w} 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
nop	addi x0,x0,0	空指令
not rd, rs	xori rd, rs, -1	寄存器取反指令
ret	jalr x0, x1,0	子程序返回指令

续下页

表 13.1 - 接上页

伪指令	基础指令	含义
s{b|h|w} rd, symbol, rt	auipc rt, symbol[31:12] s{b|h|w} rd, symbol[11:0](rt)	4GiB 地址空间存储指令
seqz rd, rs	sltiu rd, rs, 1	寄存器为 0 置 1 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

14 附录 B 玄铁扩展指令术语

除了 RISC-V 标准中定义的指令集外，E901 扩展实现了玄铁协处理器指令，以下对每条指令做具体描述。

14.1　附录玄铁扩展协处理器指令术语

为方便部分用户快速使用协处理器指令，E901 预设了部分用户可能常用的协处理器扩展指令，该部分指令已经在编译工具中进行了支持，指令编码使用 custom2 域。

备注

如需使用玄铁扩展的用户自扩展协处理器指令集（包括本节提到的玄铁预设的协处理器扩展指令），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。

E901 预设玄铁扩展协处理器指令支持功能如下：

- 支持部分整型指令扩展。
- 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 custom2 域，均为 [6:0]:1011011。
- 玄铁协处理器扩展指令 reserved 域当作 0 处理。

详细指令描述如下各小节所述，示意图中的 type1 表示整型指令：

14.1.1　整型指令

14.1.1.1　cpx0

语法：

cpx0 index，rs1，uimm

异常：

非法指令异常

说明：

source 来源 rs1、uimm，其中 uimm 数据共 10 bit，高 5 bit 是 uimm1，低 5 bit 是 uimm2，由两个编码拼接而成，该指令不产生回写

index 指示发往哪一个协处理器, 位宽为 2 bit，值可以为 0 ~ 3

指令格式：

31 30	29	28	27	26	25	24 20	19 15	14	13	12	11 7	6	5	4	3	2	1	0	
index	0	0	0	0	0	uimm1	rs1	0	0	0	uimm2	1	0	1	1	0	1	1	type1

14.1.1.2　cpx1

语法：

cpx1 index，rs1

异常：

非法指令异常

说明：

source 来源 rs1，该指令不产生回写

index 指示发往哪一个协处理器, 位宽为 2 bit，值可以为 0 ~ 3

指令格式：

31 30	29	28	27	26	25	24 20	19 15	14	13	12	11 7	6	5	4	3	2	1	0	
index	0	0	0	0	1	reserved	rs1	0	0	0	reserved	1	0	1	1	0	1	1	type1

14.1.1.3　cpx2

语法：

cpx2 index，rd，rs1，uimm

异常：

非法指令异常

说明：

source 来源 rs1，uimm，该指令需要回写 rd 寄存器

index 指示发往哪一个协处理器, 位宽为 2 bit，值可以为 0 ~ 3

指令格式：

31 30	29	28	27	26	25	24 20	19 15	14	13	12	11 7	6	5	4	3	2	1	0	
index	0	0	0	0	0	uimm	rs1	0	0	1	rd	1	0	1	1	0	1	1	type1

14.1.1.4　cpx3

语法：

cpx3 index，rd，rs1

异常：

非法指令异常

说明：

source 来源 rs1，该指令需要回写 rd 寄存器

index 指示发往哪一个协处理器, 位宽为 2 bit，值可以为 0 ~ 3

指令格式：

31 30	29	28	27	26	25	24 20	19 15	14	13	12	11 7	6	5	4	3	2	1	0	
index	0	0	0	0	1	reserved	rs1	0	0	1	rd	1	0	1	1	0	1	1	type1

14.1.1.5　cpx4

语法：

cpx4 index，rs1，rs2，uimm

异常：

非法指令异常

说明：

source 来源 rs1, rs2，uimm，该指令不产生回写

index 指示发往哪一个协处理器, 位宽为 2 bit，值可以为 0 ~ 3

指令格式：

31 30	29	28	27	26	25	24 20	19 15	14	13	12	11 7	6	5	4	3	2	1	0	
index	0	0	0	1	0	rs2	rs1	0	0	0	uimm	1	0	1	1	0	1	1	type1

14.1.1.6　cpx5

语法：

cpx5 index，rs1，rs2

异常：

非法指令异常

说明：

source 来源 rs1, rs2，该指令不产生回写

index 指示发往哪一个协处理器, 位宽为 2 bit，值可以为 0 ~ 3

指令格式：

31 30	29	28	27	26	25	24 20	19 15	14	13	12	11 7	6	5	4	3	2	1	0	
index	0	0	0	1	1	rs2	rs1	0	0	0	reserved	1	0	1	1	0	1	1	type1

14.1.1.7　cpx6

语法：

cpx6 index，rd，rs1，rs2

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	1	0	1	1	0	1	1	type1

14.1.1.8　cpx9

语法：

cpx9 index，rd，rs1，uimm

异常：

非法指令异常

说明：

source 来源 rs1，rd，uimm，该指令同时需要回写 rd 寄存器，rd 既是 source 又是目的寄存器

index 指示发往哪一个协处理器, 位宽为 2 bit，值可以为 0 ~ 3

指令格式：

31 30	29 20	19 15	14	13	12	11 7	6	5	4	3	2	1	0	
index	uimm	rs1	1	1	1	rd	1	0	1	1	0	1	1	type1

14.1.1.9　cpx10

语法：

cpx10 index，rd，uimm

异常：

非法指令异常

说明：

source 来源 uimm，该指令同时需要回写 rd 寄存器

index 指示发往哪一个协处理器, 位宽为 2 bit，值可以为 0 ~ 3

指令格式：

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	uimm	0	0	1	rd	1	0	1	1	0	1	1	type1

15 附录 C 机器模式控制状态寄存器

机器模式控制状态寄存器（CSR）按照功能分为：机器模式信息寄存器组、机器模式异常配置寄存器组、机器模式异常处理寄存器组、机器模式计数器寄存器组、机器模式扩展寄存器组。

15.1 机器模式信息寄存器组

15.1.1 厂商编号寄存器（mvendorid）

机器模式厂商编号寄存器（mvendorid）存储了玄铁 CPU 的厂商编号信息，该 ID 由 JEDEC 管理分配，目前 E901 内该寄存器值固定为 0x5B7。

该寄存器的位宽是 32 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。

15.1.2 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，E901 目前尚未定义，值为全零。

该寄存器的位宽是 32 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。

15.1.3 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。E901 内目前未定义，值为全零。

该寄存器的位宽是 32 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。

15.1.4 线程编号寄存器（mhartid）

机器模式线程编号寄存器（mhartid）存储了处理器核的线程编号。E901 内开放低 3 bits 到接口由客户自定义，高 29 bits 值为全零。

该寄存器的位宽是 32 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。

15.2　机器模式异常设置寄存器组

15.2.1　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。

该寄存器的位宽是 32 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。该寄存器的复位值为 0x1800。

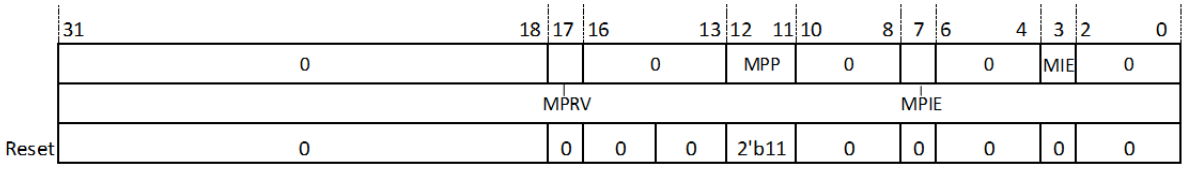


图 15.1: 机器模式处理器状态寄存器（mstatus）

MIE-机器模式全局中断使能位：

当 MIE 为 0 时，中断无效；

当 MIE 为 1 时，中断有效。

该位复位值为零，也在处理器响应异常时被清零；在处理器退出异常时被置为 MPIE 的值。

MPIE-机器模式保留中断使能位：

该位用于保存处理器进入异常服务程序前 MIE 位的值。

该位复位值为零，在处理器退出异常服务程序时被置 1。

MPP-机器模式保留特权状态位：

该位用于保存处理器进入异常服务程序前的特权状态。

MPP 固定为 2’b11，表示处理器进入异常服务程序前处于机器模式。

MPRV-存储特权位：

MPRV 硬件连线为 1’b0，访存地址的转换和保护判断没有特殊要求。

15.2.2　机器模式处理器指令集信息寄存器（misa）

机器模式处理器指令集寄存器（misa）存储了处理器所支持的指令集架构信息。

该寄存器的位宽是 32 位，寄存器的读写权限是机器模式只可读写，即非机器模式访问都会导致非法指令异常。

E901 的指令集架构为 RV32E[B][_Zmmul]_Zc，对应的 misa 寄存器复位值为 0x40901105。具体的赋值规则请参考 RISC-V 官方文档《riscv-privileged》。（Document Version 20190208-Priv-MSU-Ratified）

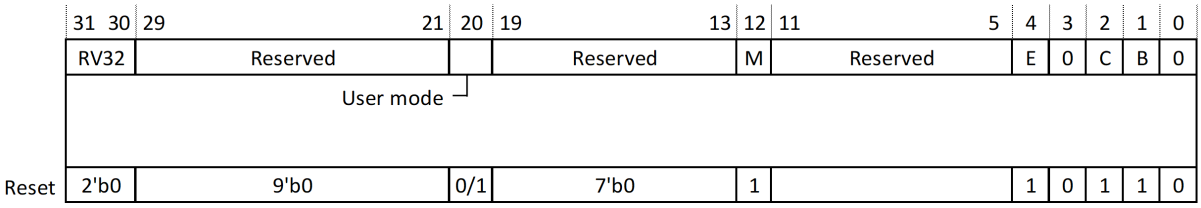


图 15.2: 机器模式处理器指令集信息寄存器（misa）

B: RISC-V B 扩展指令集

用户选配。

C: RISC-V C 扩展指令集

固定为 1。

E：RISC-V E 扩展指令集

固定支持 RV32E。

User Mode: 是否支持用户模式

固定为 0。

RV32:

固定为 1。

E901 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。

15.2.3　机器模式异常向量基址寄存器（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以及中断与异常服务程序的入口地址寻址模式。

该寄存器的位宽是 32 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。

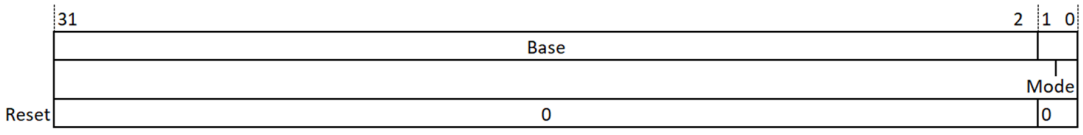


图 15.3: 机器模式异常向量基址寄存器（mtvec）

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 30 位，将此基址低位拼接 2'b00 即可得到异常服务程序入口地址。

该位复位值为零。

MODE-向量入口模式位：

当 MODE[1:0] 是 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] 是 2'b01 时，异常仍使用 BASE 地址作为入口地址，中断使用（BASE << 2 + 4 * 中断 ID）的值作为入口地址，中断 ID 为中断的向量号。

硬件配置 CLIC 模式时，MODE[1:0] 新增 2'b10 和 2'b11 配置。处理器在该两种模式下，异常入口地址会被约束为 64 字节对齐。

当 MODE[1:0] 是 2'b10 时，保留。

当 MODE[1:0] 是 2'b11 时，CPU 使用 mtvec[31:6] << 6 作为异常的服务程序入口地址并跳转执行。硬件矢量中断模式下，CPU 首先使用 mtvt + 4* 中断 ID 为地址，取出中断服务程序入口地址，并跳转到该入口地址执行中断服务程序。非硬件矢量中断模式下 CPU 使用 mtvec[31:6] << 6 作为中断服务程序入口地址并跳转执行。mtvt 为矢量中断基址寄存器，在 CLIC 配置下存在。

E901 中 MODE[1:0] 硬件固定设置为 2'b11，软件不可设置。

备注

- mtvec 硬件选配为只读时，mtvec 地址由硬件接口指定，写 mtvec 会上报异常。
- mtvec 硬件选配为软件可读可写时，软件可配置 base 地址。

15.2.4　机器模式矢量中断基址寄存器（mtvt）

机器模式矢量中断基址寄存器（mtvt）用于配置矢量中断服务程序的入口地址。

该寄存器的位宽是 32 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。

	31 … 6	5 … 0
	Base	0
Reset	0	0

图 15.4: 机器模式矢量中断基址寄存器（mtvt）

BASE-向量基址位：

向量基址位指示了矢量中断向量表基址，处理器通过计算该基址加上每个中断的地址偏移量（mtvt + 4* 中断 ID）得到矢量中断向量表中每个中断服务程序的入口地址并跳转执行。

该基址域复位值为零。

备注

- mtvt 硬件选配为只读时，mtvt 地址由硬件接口指定，写 mtvt 会上报异常。
- mtvt 硬件选配为软件可读可写时，软件可配置 base 地址。

15.3　机器模式异常处理寄存器组

15.3.1　机器模式数据备份寄存器（mscratch）

机器模式数据备份寄存器（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。

该寄存器的位宽是 32 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。

15.3.2　机器模式中断数据备份寄存器（mscratchcswl）

机器模式中断数据备份寄存器（mscratchcswl）用于加速处理器前后两个状态不同时都为中断处理的情况。可用于 mscratch 与栈指针交换数值，该指令语法如下：

```
csrrw rd, mscratchcswl, rs1
```

无条件支持，即该指令直接可以交换临时寄存器和 sp 寄存器的值。

一般用法为：

```
csrw sp, mscratchcswl, sp
```

15.3.3　机器模式中断控制器基址寄存器（mclicbase）

机器模式中断控制器基址寄存器（mclicbase）用于向软件指示 CLIC 内存映射寄存器的地址，E901 中硬件固定设置为 32'hE0800000，软件不可改写。

该寄存器的位宽是 32 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。

15.3.4　机器模式异常程序计数器（mepc）

机器模式异常程序计数器（mepc）用于存储程序从异常服务程序退出时要返回的程序计数器值（即 pc 值）。E901 支持 16 位宽指令，pc 值以半字对齐，因此 mepc 的最低位为常零，剩余高 31 位为写有效区域。

该寄存器的有效位宽是 21 位，寄存器的读写权限是机器模式可读写。

15.3.5　机器模式异常向量寄存器（mcause）

机器模式异常向量寄存器（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。

该寄存器的位宽是 32 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。

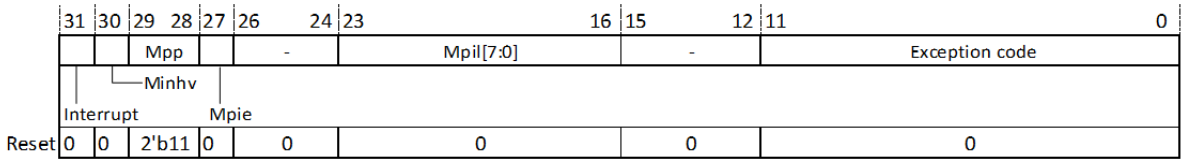


图 15.5: 机器模式异常事件向量寄存器（mcause）

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 按照普通异常规则解析；

当 Interrupt 位为 1 时，表示触发异常的来源是中断，Exception Code 按照中断规则解析。

该位复位值为零。

MINHV-矢量中断跳转指示位

该位仅在处理器配置了 CLIC 模式下存在。

用于指示处理器是否正在取矢量中断入口地址，在处理器响应矢量中断时，该位会被置高。成功获取矢量中断服务程序入口地址后清 0。

该位复位值为零。

MPP-机器模式保留特权状态位

该位仅在处理器配置了 CLIC 模式下存在。

读取时固定为 2'b11。

MPIE-机器模式保留中断使能位

该位仅在处理器配置了 CLIC 模式下存在。

该位为 mstatus.MPIE 的镜像。

MPIL-机器模式保留中断优先级位

该位仅在处理器配置 CLIC 模式下存在。

读取时固定为 0。CLIC 为中断固定优先级，优先级为中断异常向量号，不支持中断嵌套。

Exception Code-异常向量号位

在处理器进入异常时，异常向量号域会被更新为异常来源的向量号。

由于 E901 固定配置 CLIC 模式，该位域为 12 位，但只提供 8 位可读可写，[11:9] 固定为只读 0。

该位复位值为零。

具体的异常向量号列表请参考 表 4.1 。

15.4　机器模式性能监测控制寄存器

15.4.1　机器模式计数器使能寄存器（mcounteren）

E901 不支持用户模式，所以 mcounteren 寄存器无实际作用。

备注

计数器选配为无时，硬件不再实现 mcounteren，软件读写会上报异常。

15.4.2　机器模式计数禁止寄存器（mcountinhibit）

机器模式计数禁止寄存器（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，可以降低处理器功耗。

	31 … 3	2	1	0
	Reserved	MIR	0	MCR
Reset	0	0	0	0

图 15.6: 机器模式计数禁止授权寄存器（mcountinhibit）

MCY - mcycle 寄存器禁止计数位

0: 正常计数；

1: 停止计数。

MIR - minstret 寄存器禁止计数位

0: 正常计数；

1: 停止计数。

备注

计数器选配为无时，硬件不再实现 mcountinhibit，软件读写会上报异常。

15.5　机器模式异常处理寄存器组

机器模式计数器寄存器组属于性能监测方面的寄存器，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。

15.5.1　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 寄存器就会在每个处理器执行周期自增计数，每个周期加 1。

周期计数器复位值为零。

备注

计数器选配为无时，硬件不再实现 mcycle，软件读写会上报异常。

15.5.2　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，每退休一条指令该寄存器加 1。

退休指令计数器复位值为零。

备注

计数器选配为无时，硬件不再实现 minstret，软件读写会上报异常。

15.6　玄铁机器模式扩展寄存器组

15.6.1　扩展状态寄存器（mxstatus）

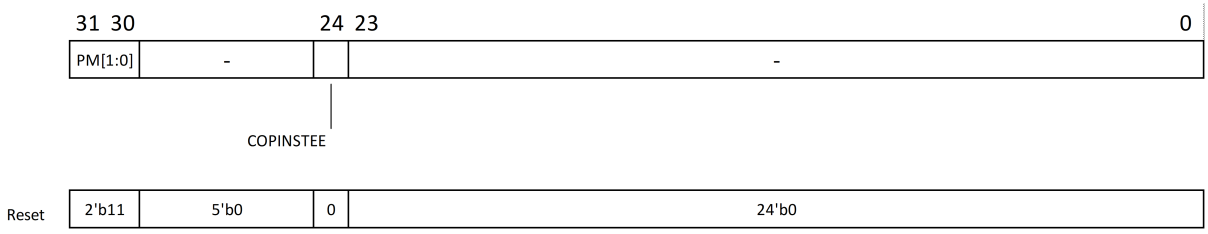


图 15.7: 扩展状态寄存器（mxstatus）

COPINSTEE-用户自扩展协处理器指令集使能位：

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。

PM-当前中断特权模式位:

硬件连线为 2'b11，表示机器模式。

15.6.2　处理器复位启动地址寄存器（mraddr）

该寄存器仅用于标识 CPU 硬件集成时 SoC 所指定的 CPU 复位启动地址值的大小，机器模式只读，写无效。

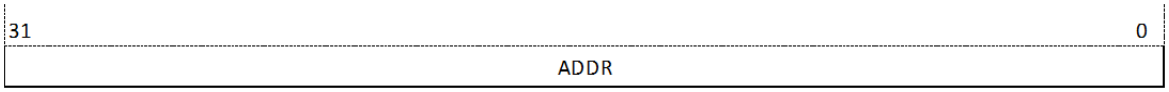


图 15.8: 处理器复位启动地址寄存器（mraddr）

ADDR-复位启动地址

该域的低两位值为常 0。

由于寻址和访存空间为 21 bits，因此外部接口的 mraddr 也需要满足该地址空间需求。

15.6.3 扩展异常状态寄存器（mexstatus）

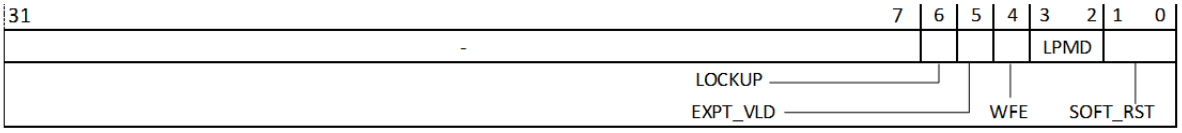


图 15.9: 扩展异常状态寄存器（mexstatus）

SOFT_RST-软复位指示

2'b00: 无复位请求；

2'b01: 仅复位内核；

2'b10: 复位系统；

2'b11: 未实现。

该域复位为 2'b00，通过写该域可以使得 CPU 顶层输出信号 cpu_pad_srst[1:0] 指示相应的值。SoC 设计人员需要依据该值来实现相应的复位操作，E901 内部不会自主复位。

LPMD-低功耗模式

2'b00: 深度睡眠；

2'b01: 浅度睡眠；

2'b10: 未实现，写无效；

2'b11: 正常工作模式，写无效。

该域复位为 2'b00，通过设置该域可以指示 SoC 在 CPU 执行 WFI 指令后系统应进入何等水平的低功耗模式，CPU 以顶层输出信号 sysio_pad_lpmd_b[1:0] 指示。对于 CPU 而言，不管该域设置为 2'b00 还是 2'b01，再通过 WFI 进入低功耗模式后所采取的低功耗策略一致，即关闭绝大多数跟唤醒逻辑无关的寄存器时钟。

当 CPU 被唤醒后，该值为 2'b11。

WFE-低功耗唤醒模式指示

1'b0: 中断唤醒 CPU 时不要求唤醒中断的优先级；

1'b1: 中断唤醒 CPU 时不要求唤醒中断的优先级，同时外部事件也可以唤醒 CPU；

该域复位值为 1'b1。

EXPT_VLD-异常状态指示位

1'b0: 处理器没有在处理异常；

1'b1: 处理器正在处理异常。

该域复位值为 1'b0，仅在调试模式下可以对该寄存器进行写操作，非调试模式下，机器模式只读。

LOCKUP-锁定指示位

1'b0: CPU 没有被锁定；

1'b1: CPU 被锁定。

该位复位值为 1'b0，机器模式只读。

15.6.4　处理器型号寄存器（mcpuid）

处理器型号寄存器（mcpuid）存储了处理器型号信号，其复位值由产品本身决定。

15.7　Zcmt jvt 控制寄存器

Zcmt 扩展中的 jvt 寄存器是跳转向量基地址和控制寄存器，用于支持基于跳转表的指令优化。它通过配置跳转表的基地址和模式，允许程序高效实现复杂的跳转逻辑，从而减少代码大小并提高执行效率。

31 6	5 0
base	mode

图 15.10: Zcmt JVT 寄存器（jvt）

mode:

固定为 6'b000000;

base:

- jvt 硬件选配为只读时，jvt 地址由接口指定。
- jvt 硬件选配为软件可读可写时，base 地址软件可配置。

15.8　调试/追踪寄存器组（与调试模式共享）

15.8.1　调试/追踪触发器选择寄存器（tselect）

调试/追踪触发器选择寄存器（tselect）用于在多个触发器（trigger）之间选中一个，以进行下一步对该触发器的寄存器读写。

	31 4	3 0
		SELECT
Reset	0	0

图 15.11: 调试/追踪触发器选择寄存器（tselect）

SELECT - 调试/追踪触发器选择

记录目前选中的调试/追踪触发器编号。例如，要配置 2 号触发器时，SELECT 写入 0x2。

15.8.2　调试/追踪触发器数据寄存器 1（tdata1）

	31 — 28	27	26 — 0
	TYPE	DEMODE	DATA
Reset	0x2 (mcontrol) 0x5 (itrigger/etrigger/icount)	0	0

图 15.12: 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。

E901 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 1/2/3（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 v0.13.2 中 5.2 小节）。

15.8.3　调试/追踪触发器数据寄存器 2（tdata2）

	31 — 0
	DATA
Reset	0

图 15.13: 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。

15.8.4　调试/追踪触发器数据寄存器 3（tdata3）

	31 26	25	24 18	17 0
	MVALUE	MSELECT	0	reserved
Reset	0	0	0	0

图 15.14: 调试/追踪触发器数据寄存器 3（tdata3）

MVALUE - 触发器机器模式内容匹配数据

用于设置希望匹配的机器模式内容数值。

MSELECT - 触发器机器模式内容匹配控制

- 当 MSELECT 为 0 时：关闭触发器机器模式内容匹配
- 当 MSELECT 为 1 时：当机器模式内容寄存器（mcontext）与触发器机器模式内容匹配数据（MVALUE）相等时，该触发器可以对处理器信息进行匹配。

15.8.5　调试/追踪触发器信息寄存器（tinfo）

	31 16	15 0
	0	INFO
Reset	0	0x100 (mcontrol) 0x111000 (itrigger/etrigger/icount)

图 15.15: 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。

E901 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。

15.8.6　调试/追踪触发器控制寄存器（tcontrol）

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

图 15.16: 调试/追踪触发器控制寄存器（tcontrol）

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。

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。

15.8.7　机器模式内容寄存器（mcontext）

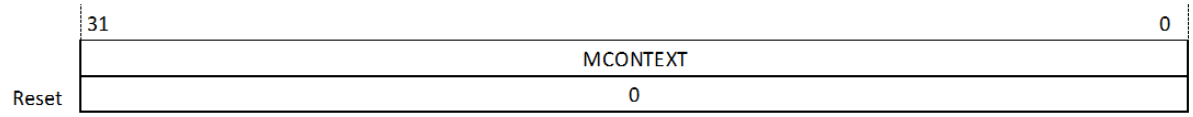


图 15.17: 机器模式内容寄存器（mcontext）

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。

15.9　调试模式寄存器组

15.9.1　调试模式控制与状态寄存器（dcsr）

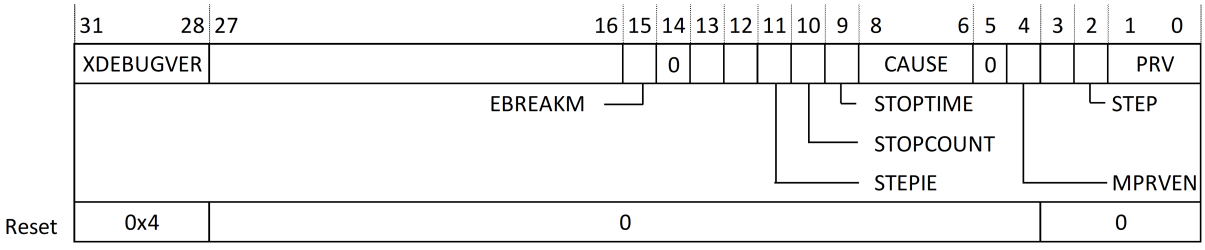


图 15.18: 调试模式控制与状态寄存器（dcsr）

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

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（包括 mcycle 和 minstret 寄存器）不计数

STOPTIME：

- 保留

CAUSE：

- 1: 表示进入调试模式的原因是 ebreak 指令的执行
- 2: 表示进入调试模式的原因是触发器的触发
- 3: 表示进入调试模式的原因是同步调试请求
- 4: 表示进入调试模式的原因是单步调试请求
- 5: 表示进入调试模式的原因是复位调试请求

MPRVEN：

- 0：mstatus 寄存器中 MPRV 字段在调试模式失效
- 1：mstatus 寄存器中 MPRV 字段在调试模式有效，处理器根据 MPRV 字段和 MPP 字段的设置处理访存指令的地址翻译与保护

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

15.9.2　调试模式程序计数器（dpc）

DPC[31:0]：

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。

15.9.3　调试模式临时数据备份寄存器 0（dscratch0）

DSCRATCH0[31:0]：

硬件上用于调试系统与处理器核之间的数据交换。

15.9.4　调试模式临时数据备份寄存器 1（dscratch1）

DSCRATCH1[31:0]：

硬件上用于调试系统与处理器核之间的数据交换。

15.10　调试扩展寄存器组

15.10.1　玄铁调试原因寄存器（mhaltcause）

	31 … 4	3 … 0
		MHALTCAUSE
Reset	0	0

图 15.19: 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

15.10.2　玄铁调试信息寄存器（mdbginfo）

MDBGINFO[31:0]：

用于异步调试时记录处理器核的调试信息。

15.10.3　玄铁分支目标地址记录寄存器（mpcfifo）

MPCFIFO[31:0]：

记录分支/跳转指令的目标地址。