玄铁 E902R2S2 用户手册

2020 年 10 月 12 日

平头哥半导体有限公司 T-HEAD Semiconductor Co.,LTD

地址: 杭州市余杭区向往街 1122 号欧美金融城 (EFC) 英国中心西楼 T6

邮编: 311121

网址: www.t-head.cn

版本历史

版本	描述	日期
1.0.0	1. 第一次正式发布。 2. 相比 C20011201-C1 修改第五章关于锁定的描述。修复 MEXSTATUS.LPMD 复位值描述错误。完善十三章中断初始化描述。	2020.06.19
01	更新文档名称和版本号，勘误文档。	2020.07.30
02	优化文档的内容格式。	2020.08.10
03	调整目录结构，第四章第五章调换位置。	2020.08.12
04	修正了 MCAUSE 和 MTVAL 寄存器的描述。	2020.08.14
05	调整部分章节描述。	2020.09.07
06	删除 SYNC 指令描述。	2020.10.12

文档编号

平头哥半导体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

第一章　概述

1.1　适用范围

本文档适用于平头哥半导体有限公司的玄铁系列 CPU E902。

1.2　简介

玄铁 E902 是平头哥半导体有限公司自主研发的极低功耗、极低成本嵌入式 CPU 核，以 8 位 CPU 的成本获得 32 位嵌入式 CPU 的运行效率与性能。E902 兼容 RISC-V 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，具备极低成本、极低功耗和高代码密度等优点。E902 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。

1.3　特点

E902 处理器体系结构的主要特点如下：

- 支持 RISC-V RV32E[M]C 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 机器模式和用户模式；
- 可配置的单周期硬件乘法器，多周期硬件除法器；
- 兼容 RISC-V CLIC 中断标准，支持中断嵌套，外部中断源数量最高可配置 240 个；
- 兼容 RISC-V PMP 内存保护标准，0/4/8/12/16 区域可配置；
- 支持 AHB-Lite 总线协议，支持指令总线，系统总线；
- 支持可配置的指令高速缓存，缓存行 16 字节，容量 2KiB/4KiB/8KiB 可配；
- 支持平头哥扩展编程模型；
- 支持复位启动地址硬件集成时可配置；
- 支持软复位操作。

1.4 可配置选项

E902 可配置选项如 表 1.1 所示。

表 1.1: E902 可配置选项

可配置单元	配置选项	详细
指令 cache	无/2KiB/4KiB/8KiB	可以不配置指令 cache 或者配置为 2KiB、4KiB、8KiB
硬件乘法器	无/有	若配置硬件乘法器，仍可配置单周期快速乘法器或者多周期（3-34）慢速乘法器
内存保护单元	0/4/8/12/16	可以配置为 0/4/8/12/16 个表项，其中 0 表示不实现内存保护单元。
CLIC	中断源：1-240 任意可配 中断优先级有效位：2-5 位任意可配	支持 1-240 中断源任意可配，中断优先级有效位：2-5 位任意可配，对应 4-32 个优先级

1.5 可调试性设计

E902 使用平头哥自定义的两线 JTAG 硬件调试接口。E902 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、程序流跟踪等。具体请详见调试接口 。

1.6 版本说明

E902 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 2.2*
- *The RISC-V Instruction Set Manual, Volume II: RISC-V Privileged Architecture, Version1.10*
- *RISC-V Core-Local Interrupt Controller (CLIC) Version 0.8*

第二章　处理器简介

2.1　结构框图

E902 结构框图如 图 2.1 所示。

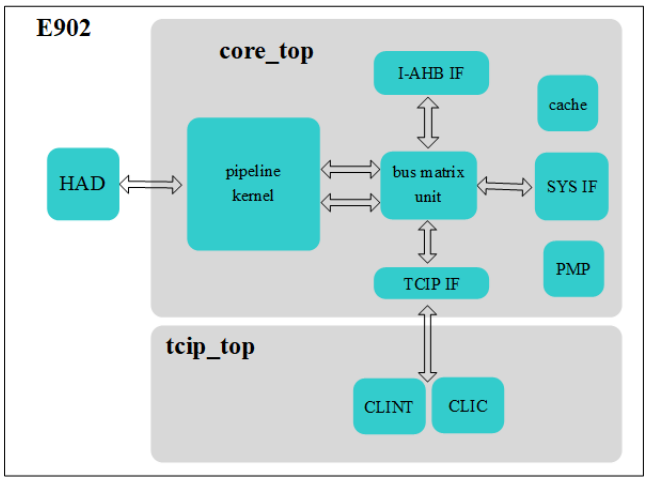


图 2.1: E902 结构图

E902 处理器采用 2 级流水线结构：取指和执行。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。

可配置的物理内存保护单元（Physical Memory Protection，PMP）负责物理地址的权限检查实现内存的保护功能，权限可划分为：不可读写/只读/可读写，可执行/不可执行。

硬件辅助调试单元（Haredware Assistant Debug，HAD）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（GPR）、控制寄存器（CSR）和内存。

E902 设计有片上紧耦合的 IP 接口和多条 AHB-Lite 的总线接口。片上紧耦合的 IP 接口集成处理器中断控制器（CLIC），支持中断嵌套以及处理器核局部中断（CLINT）。外部中断源数量最高可配置 240 个，中断优先级支持 4/8/16/32 级可配置。

2.2 紧耦合 IP 架构

为了提高 E902 的系统集成度，方便用户集成与开发，E902 设置内部总线用于集成紧耦合 IP（Titly 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:

- 支持 1-240 个外部中断源；
- 支持中断嵌套；
- 支持电平中断和脉冲中断；
- 中断优先级 4/8/16/32 任意可配。

2.3 接口概览

E902 接口总览如 图 2.2 所示，具体接口信号描述参考《玄铁 E902 R2S2 集成手册》。

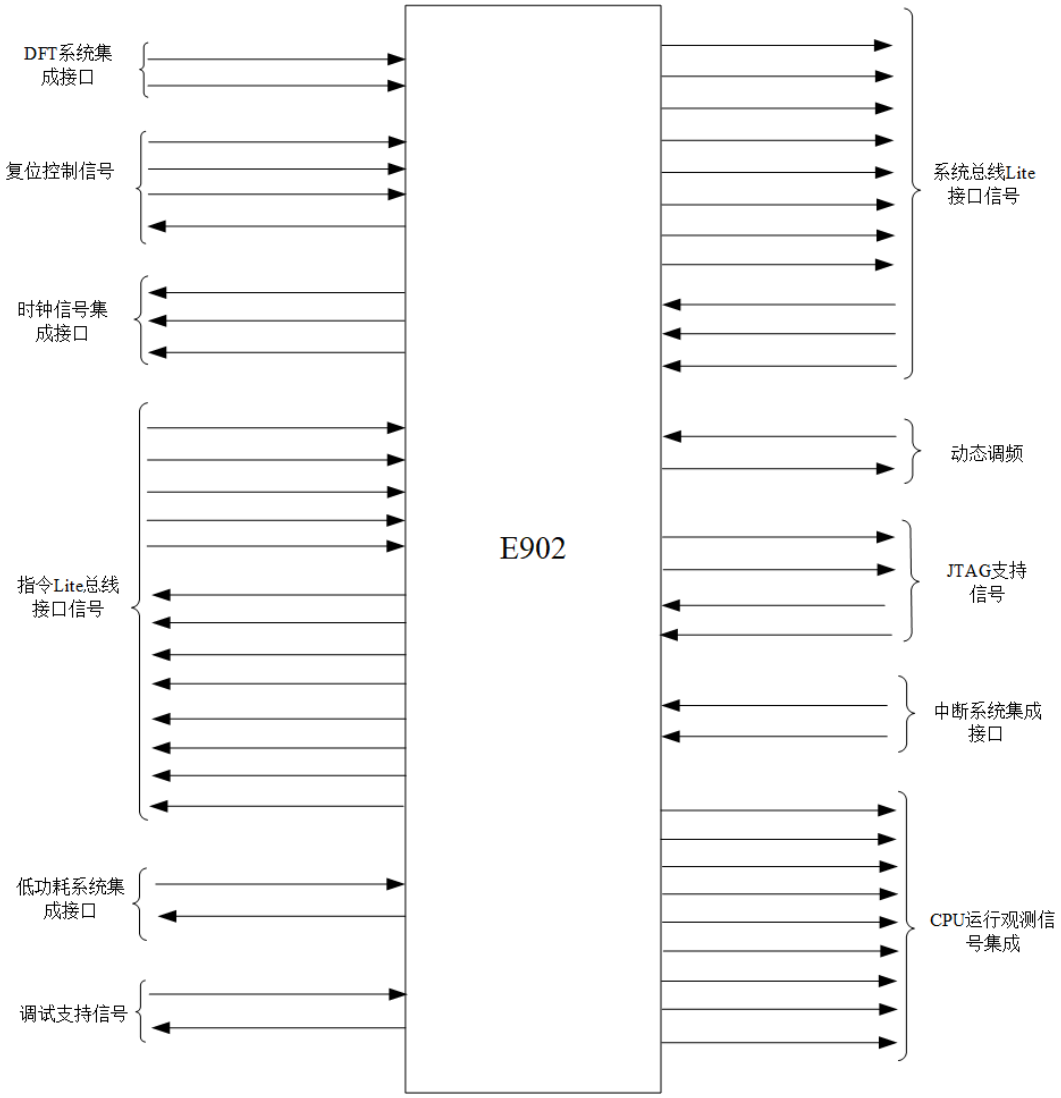


图 2.2: E902 接口总览

第三章　编程模型

3.1　工作模式及寄存器视图

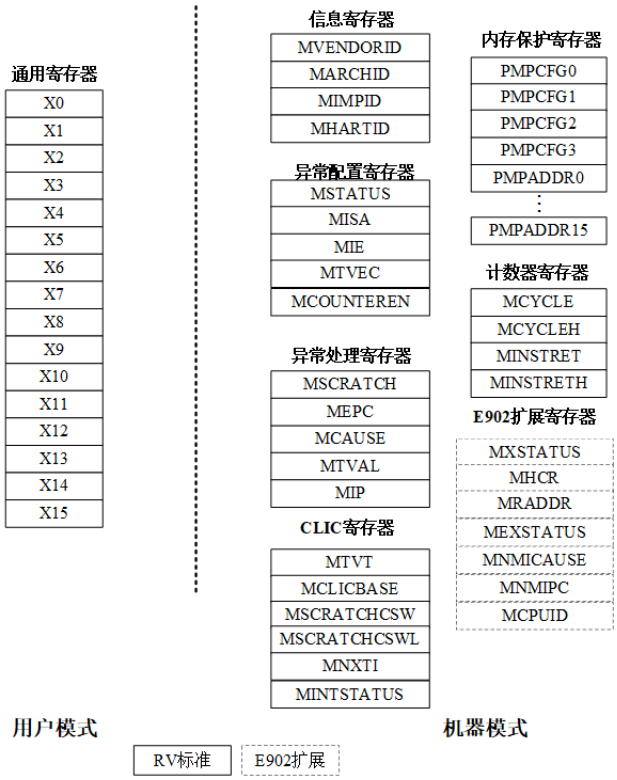


图 3.1: 编程模型

E902 实现了两种 RISC-V 编程模式：机器模式和普通用户模式。当 MXSTATUS 寄存器内 PM 位被置位为 2’ b11 时，处理器即在机器模式下执行程序。处理器复位后，工作在机器模式。

两种编程模式对应不同的操作权限，区别主要体现在以下几个方面：

1. 对控制寄存器的访问；
2. 特权指令的使用；
3. 对紧耦合 IP 的控制寄存器访问。

普通用户模式只允许访问通用寄存器；机器模式可以访问所有通用寄存器、控制寄存器和所有紧耦合 IP 的控制寄存器。

普通用户模式可以使用绝大多数除了对系统产生重大影响的特权指令如 WFI，MRET，CSR 和平头哥扩展的 CACHE 指令之外的指令。机器模式下可以使用 E902 支持的所有指令。普通用户模式通过使用 ECALL 指令进入机器模式。

3.2 通用寄存器

表 3.1 列出了普通用户编程模式下的 16 个 32 位通用寄存器（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。

此外，普通用户编程模式寄存器还包括处理器执行地址寄存器（PC）。

3.3 机器模式控制寄存器

E902 中实现的 RISC-V 标准定义的机器模式控制寄存器如 表 3.2 所示。

表 3.2: 机器编程模式控制寄存器列表

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	微体系结构编号寄存器
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			

下页继续

表 3.2 – 续上页

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
MIE	机器模式读写	0x304	机器模式中断使能控制寄存器
MTVEC	机器模式读写	0x305	机器模式异常向量基址寄存器
MCOUNTEREN	机器模式读写	0x306	机器模式计数器使能寄存器
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
MTVAL	机器模式读写	0x343	机器模式异常向量寄存器
MIP	机器模式读写	0x344	机器模式中断等待寄存器
MNXTI	机器模式读写	0x345	机器模式等待中断向量地址和中断使能寄存器
MINTSTATUS	机器模式只读	0x346	机器模式中断状态寄存器
MSCRATCHCSW	机器模式读写	0x348	机器模式多模式数据备份寄存器
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
机器模式内存保护寄存器组			
PMPCFG0	机器模式读写	0x3A0	物理内存保护配置寄存器 0
PMPCFG1	机器模式读写	0x3A1	物理内存保护配置寄存器 1
PMPCFG2	机器模式读写	0x3A2	物理内存保护配置寄存器 2
PMPCFG3	机器模式读写	0x3A3	物理内存保护配置寄存器 3
PMPADDR0	机器模式读写	0x3B0	物理内存保护基址寄存器 0
……			
PMPADDR15	机器模式读写	0x3BF	物理内存保护基址寄存器 15
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
MCYCLEH	机器模式读写	0xB80	机器模式周期计数器高 32 位
MINSTRETH	机器模式读写	0xB82	机器模式退休指令计数器高 32 位

E902 中扩展的控制寄存器如 表 3.3 所示。

表 3.3: E902 扩展机器模式控制寄存器

名称	读写权限	寄存器编号	描述
机器模式扩展寄存器组			
MXSTATUS	机器模式读写	0x7C0	扩展状态寄存器
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
MNMICAUSE	机器模式读写	0x7E2	NMI 现场保存寄存器
MNMIPC	机器模式读写	0x7E3	NMI 异常程序计数器
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 机器模式控制寄存器 。

3.4 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 。

3.5 数据格式

3.5.1 整型数据格式

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。

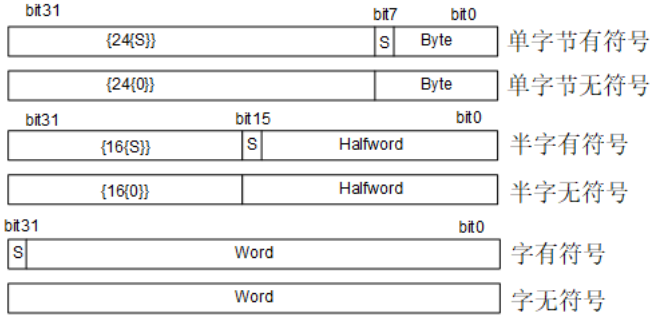


图 3.2: 寄存器中的整型数据组织结构

3.5.2 小端

存储器数据有大小端的区分，E902 仅支持小端模式，即数据高位存放至物理内存的高地址。如 图 3.3 所示。

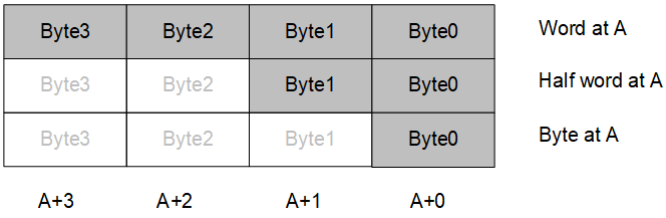


图 3.3: 内存中的数据组织形式

第四章　异常与中断

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。

异常处理是处理器根据内部或外部的异常事件从正常的程序处理转入特定的异常处理程序。引起异常的外部事件包括：外部设备的中断请求、读写访问错误；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）。ECALL 和 EBREAK 指令正常执行时也会产生异常。异常处理利用异常向量表跳转到异常服务程序的入口。

异常处理的关键是在异常发生时，保存 CPU 当前指令运行的状态，在退出异常处理时恢复异常处理前的状态。异常能够在指令流水线的各个阶段被识别，并使后面的指令不会改变 CPU 的状态。异常在指令的边界上被处理，即 CPU 在指令退休时响应异常，并保存退出异常处理时下一条被执行的指令的地址。即使异常指令退休前被识别，异常也要在相应的指令退休时才会被处理。E902 根据异常识别时的指令是否完成决定异常地址寄存器存储哪一条指令的地址。例如，如果异常事件是外部中断服务请求，被中断的指令将正常退休并改变 CPU 的状态，它的下一条指令的地址（PC+2/PC+4，根据当前指令是 16 位或 32 位决定 +2 或者 +4）将被保存在异常保留程序计数器（MEPC）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）将被保存在异常保留程序计数器（MEPC）中，CPU 从异常服务程序返回时继续执行这条访问错误指令。

E902 兼容 RISC-V 标准的异常向量号，如 表 4.1 所示：

表 4.1: 异常向量号

中断标记	向量号	异常中断类型
1	0-2	未实现/保留
1	3	机器模式软件中断
1	4-6	未实现/保留
1	7	机器模式计时器中断
1	8-10	未实现/保留
1	11	机器模式外部中断
1	>=12	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	255	CLIC 外接中断 239（pad_clic_int_vld[239]）
0	0	未实现
0	1	取指令访问错误异常
0	2	非法指令异常
0	3	调试断点异常
0	4	加载指令非对齐访问异常
0	5	加载指令访问错误异常
0	6	存储指令非对齐访问异常
0	7	存储指令访问错误异常
0	8	用户模式环境调用异常
0	9	未实现
0	10	保留
0	11	机器模式环境调用异常
0	12~23	未实现/保留
0	24	NMI
0	>=25	未实现/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 表 4.2 所示。E902 将 NMI 的异常向量号定义为 24，并将其设置为所有中断和异常中优先级最高的异常类型。

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
↓	8	用户模式环境调用异常
↓	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. 将 MXSTATUS 中的 PM 域保存到 MSTATUS 寄存器的 MPP 域，并将 PM 设置为机器模式，即异常响应后 CPU 进入机器模式。
6. 将产生异常事件的原因更新到 MTVAL 寄存器，如 表 4.3 所示。
7. 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 E902 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*）。

表 4.3: MTVAL 寄存器更新值定义

异常向量号	异常类型	MTVAL 更新值
1	取指令访问错误异常	取指访问的地址
2	非法指令异常	指令码
3	调试断点异常	0
4	加载指令非对齐访问异常	加载访问的地址
5	加载指令访问错误异常	加载访问的地址
6	存储指令非对齐访问异常	存储访问的地址
7	存储指令访问错误异常	存储访问的地址
8	用户模式环境调用异常	0
11	机器模式环境调用异常	0

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 域被恢复成 MSTATUS.MPP 的值，MPP 被设置为 2' b00（硬件实现用户模式时）或者 2' b11（硬件仅实现机器模式时）。

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。E902 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。

调试请求和下文描述的 NMI 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 NMI 处理函数返回后仍将处于锁定状态。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 0xfffffffc，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 0xfffffffc，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

E902 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。

E902 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。E902 只实现了 MTVEC.mode[1:0]=3 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。

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 中断优先级

E902 支持中断优先级 2-5 位可配，最多支持 32 个中断优先级。中断优先级的设置分为两步：

1. 设置 CLIC 寄存器 CLICCFG 中的 nlbits 域，该为指示了可配置的中断优先级个数的多少；
2. 设置每个中断所对应寄存器 CLICINTCTL 的 int_ctl 域，指示该中断参与优先级仲裁的有效值及保存在 MINTSTATUS.MIL 寄存器域的有效值

具体寄存器配置请参考*CLIC 寄存器描述* 。

4.2.1.2 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. 将 MXSTATUS 中的 PM 域保存到 MSTATUS 寄存器的 MPP 域，并将 PM 设置为机器模式，即中断响应后 CPU 进入机器模式。
6. 将 MCAUSE 寄存器的 MPIL 域更新为 MINTSTATUS 寄存器的 MIL 域，MINTSTATUS 寄存器的 MIL 域被更新为当先被响应的中断的优先级。
7. 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=3 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）* 。

4.2.1.3 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。同时 CLIC SPEC 还定义了 MINTSTATUS.MIL 域来表征 CPU 当前正在处理的中断的优先级，用于和 CLIC 传给流水线核心的新进中断优先级进行比较，来判断是否可以进行中断的嵌套响应。需要注意的是，如果需要进行中断嵌套响应，还需在中断服务程序入口将 MSTATUS/MCAUSE 等寄存器进行压栈保存。

4.2.1.4 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. 将 MXSTATUS 寄存器中的 PM 域恢复成 MSTATUS.MPP，MPP 被设置为 2’b00（硬件实现用户模式时）或者 2’b11（硬件仅实现机器模式时）；
3. MSTATUS.MIE 恢复成 MSTATUS.MPIE 的值；
4. MINTSTATUS.MIL 被更新成 MCAUSE.MPIL 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 域设置为 0 来指示该中断为非矢量中断。

4.2.2.1 中断优先级

同中断优先级 节所描述。

4.2.2.2 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使用读 MNXTI 寄存器的方式触发 pending 位的清除，相关介绍请参考中断咬尾 。

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 中断返回

同中断返回 所描述。

4.2.2.4 中断咬尾

CLIC SPEC 在非硬件矢量中断模式下定义了中断咬尾处理的功能，同时支持在中断服务程序入口响应晚到且高优先级中断的功能。

因为在非硬件矢量中断模式下，中断服务程序的入口统一由 MTVEC 寄存器指定，因此可以在中断服务程序入口做统一的寄存器压栈操作。在压栈完成之后可以读取 MNXTI 寄存器的值，如果返回非零值即当前处于等待状态的 CLIC 仲裁出来的最高优先级的中断（可能是当前正在被处理的中断也有可能是新近的更高优先级的中断），并且该中断在 CLIC 内部的 pending 位被清除（仅限脉冲中断），同时通过读写 MNXTI 寄存器的操作可以主动设置 MSTATUS.MIE 值为 1。

同样，在非矢量中断本身的中断事务处理完后，在进行现场弹栈及执行 MRET 返回前，可通过执行读写 MNXTI 寄存器的指令，获取 CLIC 仲裁出来的比 MCAUSE.MPIL 优先级还高的中断入口地址，该地址由 MTVT 加中断号指定的偏移量决定。软件可据此地址从内存加载该等待响应的中断的服务程序入口地址并跳转执行。如果返回非零值，即不需要处理 CLIC 仲裁出来的中断（或者 CLIC 根本没有待处理的中断），可以直接进行弹栈操作和中断返回。

4.3 NMI

NMI 作为外部事件，E902 将其异常向量号定义为 24，优先级在所有的中断和异常中最高。因为 NMI 不受全局中断使能位 MSTATUS.MIE 的控制，因此为了使得任何时候响应 NMI 并返回之后，CPU 都可以恢复正常运行程序流，E902 扩展实现了 MNMICAUSE 寄存器和 MNMIPC 寄存器用于保存响应 NMI 时 CPU 的现场。

4.3.1 NMI 响应

1. 将 MEPC 寄存器的值保存在 MNMIPC 寄存器
2. 将响应 NMI 请求时的指令 PC 保存在 MEPC 寄存器，便于 NMI 处理结束后 CPU 返回正常程序轨迹
3. 将 MCAUSE 寄存器域的 Exception Code 以及中断指示位保存在 MNMICAUSE 寄存器，并更新 MCAUSE 寄存器的 Exception Code 为 12 ‘h18，中断指示位 (bit[31]) 设置为 0
4. 将 MSTATUS 寄存器域的 MPP 及 MPIE 域保存在 MNMICAUSE 寄存器，并将 MXSTATUS 寄存器的 PM 域更新到 MPP，将 MSTATUS 的 MIE 域保存在 MPIE

在上述 CPU 状态保存的同时，CPU 会依据 MTVEC 寄存器指定的地址统一异常入口地址，跳转执行 NMI 的服务程序。

在 NMI 被响应直到 CPU 返回正常程序轨迹之前，新的 NMI 请求以及中断请求都无法被响应，当在 NMI 服务程序中触发异常时会使 CPU 进入锁定的状态，如锁定 所述。因为 NMI 状态维护依赖 MCAUSE 寄存器中的 Exception Code 域，因此软件要避免在 NMI 服务程序中改写该寄存器域。

4.3.2 NMI 返回

在 NMI 服务程序返回时需要通过执行 MRET 指令将 CPU 的现场进行恢复，主要有如下操作：

1. 将 PC 恢复成 MEPC 寄存器的值，将 MEPC 寄存器的值恢复成 MNMIPC 寄存器的值；
2. 将 MCAUSE.exception_code，中断指示位，恢复成 MNMICAUSE 中保存的值；
3. MXSTATUS.PM 被恢复成 MSTATUS.MPP 域的值；

4. 将 MSTATUS.MPIE 和 MSTATUS.MPP 的值恢复成 MNMICAUSE 中保存的值;
5. MNMICAUSE 中的 NMI_MPP 被恢复成 2 ‘b00（硬件实现用户模式时）或者 2’ b11（硬件仅实现机器模式时）。

4.3.3 锁定

在 NMI 的服务程序中如果触发异常, CPU 会进入锁定状态。如锁定 所述, CPU 会停止取指和执行, 并设置 MEXSTATUS 寄存器的 LOCKUP 域为 1。这时即便新的 NMI 请求也不能将 CPU 的锁定状态打断。来自 SoC 的输入复位请求信号可以将 CPU 的锁定状态彻底打断，使得 CPU 重新从复位启动地址开始执行程序。

第五章　指令集

E902 采用了 16/32 位混合编码的 RV32E[M]C 指令集，并在此基础上扩展了平头哥自定义指令。E902 扩展指令集需要打开机器模式扩展状态寄存器（MXSTATUS）的扩展指令集使能位（THEADISAEE）才能正常使用，否则出现非法指令异常。

5.1　RV32IMAFDC 指令

本章主要介绍了 E902 中实现的 RV32EMC 指令集，包括标准的整型指令集（RV32E），RV 乘除法指令集（RV32M）和 RV 压缩指令（RVC）。

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
- 低功耗指令
- 异常返回指令
- 特殊功能指令

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

下页继续

表 5.1 – 续上页

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
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（cache 命中）
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	

下页继续

表 5.1 – 续上页

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 *A-1 E 指令术语* 。

5.1.2 RV32M 乘除法指令集

RV32M 乘除法指令如 表 5.2 所示。

表 5.2: RV32M 指令集列表

指令名称	指令描述	执行延时
乘除法指令		
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**
DIV	**有符号除法指令**	**1-33**
DIVU	**无符号除法指令**	**1-33**
REM	**有符号取余指令**	**1-33**
REMU	**无符号取余指令**	**1-33**

具体指令说明和定义，请参考附录 *A-2 M 指令术语* 。

5.1.3 RVC 压缩指令集

本节主要介绍 16 位压缩指令。压缩指令集按功能可以分为以下类型：

- 加减法指令
- 逻辑操作指令

- 移位指令
- 数据传输指令
- 分支跳转指令
- 立即数偏移存取指令

表 5.3: RVC 压缩指令集指令列表

指令名称	指令描述	执行延时
加减法指令		
C.ADD	有符号加法指令	1
C.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	
C.LWSP	字堆栈加载指令	
C.SWSP	字堆栈存储指令	
特殊指令		
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 A-3 C 指令术语* 。

5.2 平头哥扩展指令集

5.2.1 Cache 指令集

表 5.4: Cache 指令列表

指令名称	指令描述	执行延时	备注
ICACHE.IALL	ICACHE 无效全部表项指令	M/2	M：ICACHE 缓存行数量
ICACHE.IPA	ICACHE 无效物理地址匹配表项指令		

表 5.4 列出了 E902 实现的平头哥自定义扩展的 cache 指令子集，Cache 指令在机器模式并且开启 MXSTATUS.theadisaee 寄存器时可以正常执行，否则执行 Cache 指令会触发非法指令异常。

具体指令说明和定义，请参考*附录 B-1 Cache 指令术语* 。

第六章　物理内存保护

6.1　PMP 简介

E902 物理内存保护单元（Physical Memory Protection，PMP）遵从 RISC-V 标准。PMP 主要保护两类系统资源：存储器和外围设备。PMP 主要负责对存储器和外围设备访问的合法性进行检查，判定当前工作模式下 CPU 是否具备对内存地址的读/写/执行访问权限。

PMP 单元支持 0（即不实现 PMP）/4/8/12/16 个表项可配置，在机器模式下 CPU 可以对区域的访问权限进行设置。每个表项通过编号 0-15 来标识和索引。

E902 PMP 单元的主要特征有：

- 硬件可配置 0/4/8/12/16 个 PMP 表项；
- 地址划分最小粒度为 4 字节；
- 支持 OFF、TOR（Top of Range）、NA4（Naturally Aligned 4-Byte Region）、NAPOT (Naturally Aligned Power-of-Two Region）四种地址匹配模式；
- 支持可读、可写、可执行三种权限的配置；
- 支持表项 Lock 功能。

6.2　PMP 控制寄存器

6.2.1　物理内存保护设置寄存器（PMPCFG0~PMPCFG3）

物理内存保护设置寄存器（PMPCFG0~PMPCFG3）用于配置物理内存表项的 Lock 模式、工作模式和访问权限。物理内存保护设置寄存器共 4 个，每个寄存器可以设置 4 个表项，最多可以设置 16 个表项。

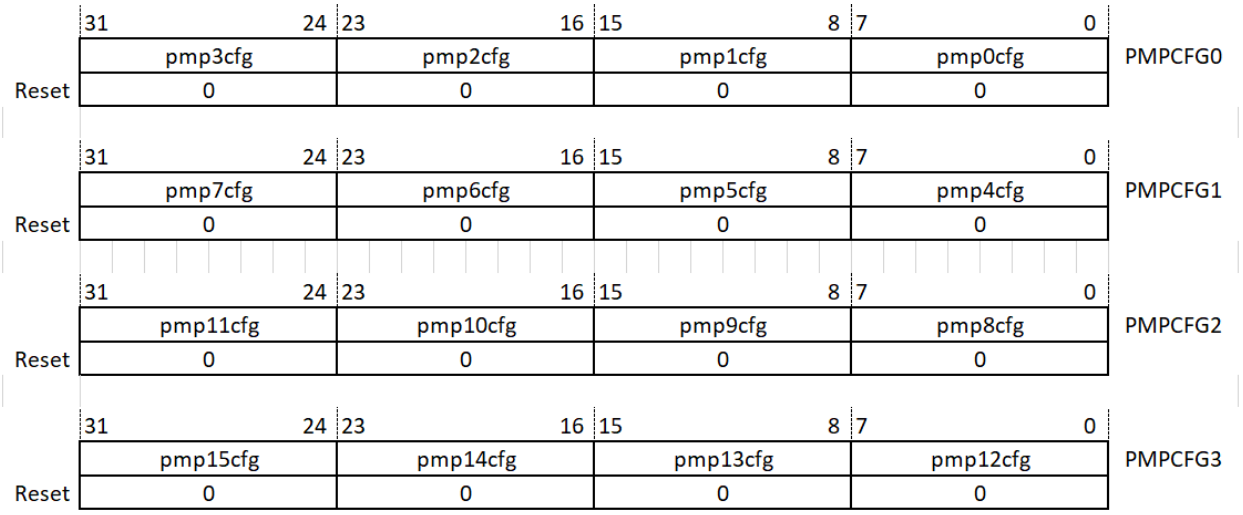
该寄存器位宽为 32 位，仅在机器模式下可读写，非机器模式访问会触发非法指令异常。

每个 64 位的 PMPCFG 提供 8 个表项的权限设置，整体分布如 图 6.1 所示。

每个物理内存保护设置寄存器的格式如 图 6.2 所示。

L-Lock 位：

当 L 为 0 时，该表项的物理内存保护设置寄存器和地址寄存器都可以修改。机器模式下的访问权限不受 R/W/X 位的限制，用户模式下的访问权限受 R/W/X 位的限制；



	31 24	23 16	15 8	7 0	
	pmp3cfg	pmp2cfg	pmp1cfg	pmp0cfg	PMPCFG0
Reset	0	0	0	0	
	31 24	23 16	15 8	7 0	
	pmp7cfg	pmp6cfg	pmp5cfg	pmp4cfg	PMPCFG1
Reset	0	0	0	0	
	31 24	23 16	15 8	7 0	
	pmp11cfg	pmp10cfg	pmp9cfg	pmp8cfg	PMPCFG2
Reset	0	0	0	0	
	31 24	23 16	15 8	7 0	
	pmp15cfg	pmp14cfg	pmp13cfg	pmp12cfg	PMPCFG3
Reset	0	0	0	0	

图 6.1: 物理内存保护设置寄存器整体分布图

	7	6 5	4 3	2	1	0
	L	-	A	X	W	R
Reset	0	0	0	0	0	0

图 6.2: 物理内存保护设置寄存器格式

当 L 为 1 时，该表项所有内容，包括 L 位，在 CPU 复位前都不可以修改。且机器模式下的访问和用户模式下的访问都将受到 R/W/X 位的限制；当该表项的物理内存保护设置寄存器配置为 TOR 地址匹配模式时，其前一表项的物理内存保护地址寄存器也无法修改。

复位值为零。

A-地址匹配模式：

当 A 为 00 时，表项无效；

当 A 为 01 时，TOR（Top Of Range）模式，使用相邻表项的地址作为匹配区间；

当 A 为 10 时，NA4（Naturally Aligned 4-byte）模式，使用 4 字节大小作为匹配区间；

当 A 为 11 时，NAPOT（Naturally Aligned Power Of Two）模式，使用 2 的幂次方大小作为匹配空间；

复位值为零。

X-可执行属性：

当 X 为 0 时，该区域为不可执行；

当 X 为 1 时，该区域为可执行；

复位值为零。

W-可写属性：

当 W 为 0 时，该区域为不可写；

当 W 为 1 时，该区域为可写；

复位值为零。

R-可读属性：

当 R 为 0 时，该区域为不可读；

当 R 为 1 时，该区域为可读；

复位值为零。

6.2.2 物理内存保护地址寄存器（PMPADDR0~PMPADDR15）

物理内存保护地址寄存器（PMPADDR0~PMPADDR15）用于配置物理内存表项的地址和区域大小。物理内存保护地址寄存器共 16 个，每个寄存器对应一个表项。

该寄存器组位宽均为 32 位，仅在机器模式下可读写，非机器模式访问会触发非法指令异常。

物理内存保护地址寄存器配合物理内存保护设置寄存器一起决定表项区域大小。

对于 TOR 模式：表项 i 控制的区域大小为 [{$PMPADDR_{i-1}$[31:0], 2' b0}, {$PMPADDR_i$[31:2], 2' b0})。对于表项 0，使用 0x0 作为地址空间下边界，即 [0, {$PMPADDR_0$[31:0], 5' b0})；若该区域的下边界大于或等于上边界，则该表项视为 OFF，即不使能，所有访问不会命中该表项。

对于 NA4 模式，第 i 个表项所对应的空间大小为 [{$PMPADDR_i$，2 'b0},{PMPADDRi,2' b0})。

对于 NAPOT 模式，其地址与区域大小的关系如 表 6.1 所示。

NAPOT 模式下表项 i 的区域大小和基址由 $PMPADDR_i$ 决定，假设 $PMPADDR_i$ 中从 bit[0] 到 bit[31] 第一次出现比特位值为 0 的是 PMPADDRi 的第 n 位，即 PMPADDRi[n]，则该表项基址为 {$PMPADDR_i$[31:n+1],(n+3){1' b0}}，空间大小为 2^(n+3) 字节（不包括空间大小上限对应地址）。以 $PMPADDR_i$=32' baaaa_aaaa_aaaa_aaaa_aaaa_aaaa_aaa0_1111 为例（其中 a 表示 1' b0 或者 1' b1），其基址为 34' baa_aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_a000_0000，空间大小为 128B。

PMPADDR 寄存器最大支持 16GiB 的物理空间，但 E902 采用 32 位地址，最大支持 4GiB。因此 PMPADDRi 的高两位尽管软件可读写，但不参与地址匹配逻辑运算。

表 6.1: 保护区间编码

PMPADDRi	PMPCFG.A	匹配区域大小
aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_aaaa	NAPOT	*4B*
aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_aaa0	NAPOT	*8B*
aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_aa01	NAPOT	*16B*
aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_a011	NAPOT	*32B*
aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_0111	NAPOT	*64B*
aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_aaa0_1111	NAPOT	128B
aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_aa01_1111	NAPOT	256B
aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_a011_1111	NAPOT	512B
aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_0111_1111	NAPOT	1KiB
aaaa_aaaa_aaaa_aaaa_aaaa_aaa0_1111_1111	NAPOT	2KiB
aaaa_aaaa_aaaa_aaaa_aaaa_aa01_1111_1111	NAPOT	4KiB
aaaa_aaaa_aaaa_aaaa_aaaa_a011_1111_1111	NAPOT	8KiB
aaaa_aaaa_aaaa_aaaa_aaaa_0111_1111_1111	NAPOT	16KiB

下页继续

表 6.1 – 续上页

aaaa_aaaa_aaaa_aaaa_aaa0_1111_1111_1111	NAPOT	32KiB
aaaa_aaaa_aaaa_aaaa_aa01_1111_1111_1111	NAPOT	64KiB
aaaa_aaaa_aaaa_aaaa_a011_1111_1111_1111	NAPOT	128KiB
aaaa_aaaa_aaaa_aaaa_0111_1111_1111_1111	NAPOT	256KiB
aaaa_aaaa_aaaa_aaa0_1111_1111_1111_1111	NAPOT	512KiB
aaaa_aaaa_aaaa_aa01_1111_1111_1111_1111	NAPOT	1MiB
aaaa_aaaa_aaaa_a011_1111_1111_1111_1111	NAPOT	2MiB
aaaa_aaaa_aaaa_0111_1111_1111_1111_1111	NAPOT	4MiB
aaaa_aaaa_aaa0_1111_1111_1111_1111_1111	NAPOT	8MiB
aaaa_aaaa_aa01_1111_1111_1111_1111_1111	NAPOT	16MiB
aaaa_aaaa_a011_1111_1111_1111_1111_1111	NAPOT	32MiB
aaaa_aaaa_0111_1111_1111_1111_1111_1111	NAPOT	64MiB
aaaa_aaa0_1111_1111_1111_1111_1111_1111	NAPOT	128MiB
aaaa_aa01_1111_1111_1111_1111_1111_1111	NAPOT	256MiB
aaaa_a011_1111_1111_1111_1111_1111_1111	NAPOT	512MiB
aaaa_0111_1111_1111_1111_1111_1111_1111	NAPOT	1GiB
aaa0_1111_1111_1111_1111_1111_1111_1111	NAPOT	2GiB
aa01_1111_1111_1111_1111_1111_1111_1111	NAPOT	4GiB
a011_1111_1111_1111_1111_1111_1111_1111	NAPOT	8GiB
0111_1111_1111_1111_1111_1111_1111_1111	NAPOT	16GiB

6.3　内存访问处理

当处理器配置 PMP 表项后，内存访问的地址会经过 PMP 检查，判断当前访问的地址是否在这些保护区内。当访问的地址命中 PMP 表项中的一个或多个内时，优先匹配高索引表项（0 为最高，15 为最低）。

对于命中的表项，L、R、W 和 X 将共同决定访问是否成功。当 L 为 0 时，机器模式下的访问权限不受 R/W/X 位的限制，所有访问都将成功；用户模式下的访问权限受 X/W/R 位的限制，只有当前访问的访问类型与该表项的访问属性相匹配时，访问成功，否则访问失败。当 L 为 1 时，机器模式下的访问和用户模式下的访问都将受到 R/W/X 位的限制，只有当前访问的访问类型与该表项的访问属性相匹配时，访问成功，否则访问失败。

当所有 PMP 表项不使能，即地址匹配模式为 OFF（2 'b00）时，或者至少有一个 PMP 表项使能，且访问的地址不命中这些使能表项中的任何一个时，此次访问是否成功由处理器所处模式决定。如果访问为机器模式权限，则访问成功，如果访问为用户模式权限，则访问失败。

当访问失败时，内存访问会被停止，且处理器会根据访问类型抛出对应的异常，即 Load 访问异常、Store 访问异常和指令访问异常。

需要说明的是，对于取指访问内存操作，其访问权限仅与发送请求时处理器所处模式有关；对于加载和存储内存操作，其访问权限由处理器所处模式和 MSTATUS 寄存器中的 MPRV、MPP 位共同决定，详情见机器模式处理器状态寄存器（*MSTATUS*）。

第七章　内存子系统

E902 实现可配置的指令高速缓存，主要特征如下：

- 指令高速缓存大小硬件可配置，支持 2KiB/4KiB/8KiB；
- 2 路组相联，缓存行大小为 16 字节；
- 访问数据位宽为 32 比特；
- 采用先进先出的替换策略；
- 支持对整个指令高速缓存的无效操作，支持对单条缓存行的无效操作。

7.1　高速缓存操作

在处理器复位后，指令高速缓存默认关闭。在 RISC-V 编程模型基础上，E902 扩展了高速缓存操作相关的控制寄存器和指令，支持高速缓存的开关、无效化等操作。

7.1.1　高速缓存扩展寄存器

E902 支持高速缓存扩展寄存器：机器模式硬件配置寄存器 (mhcr)。可以实现对指令高速缓存的开关配置。

具体控制寄存器说明可以附录 C 机器模式扩展寄存器组*硬件配置寄存器（MHCR）*。

7.1.2　高速缓存扩展指令

E902 扩展了高速缓存相关操作的指令，具体如 表 7.1 所示。

表 7.1: Cache 扩展指令

指令	描述
ICACHE.IALL	ICACHE 无效全部表项指令
ICACHE.IPA	ICACHE 无效物理地址匹配表项指令

7.2　内存模型

玄铁 E902 支持两种内存属性的配置，分别为是否可高缓（cacheable 或者 non-cacheable）的内存以及是否可暂存的内存（bufferable 或者 non-bufferable）。对于 cacheable 的内存地址空间，E902 从该区域访问得到的数据可以缓存

在 E902 内部的指令或者数据 cache 中。而对于 bufferable 的内存地址空间，E902 在对该地址空间进行写操作时，总线写响应（HRESP）可由总线通路的任一节点返回，而不必等待 E902 访问的 slave 上写操作完成后再返回。反之，对于 non-bufferable 的内存地址空间，E902 对该地址空间的写操作的响应必须在 E902 访问的 slave 上写操作完成后由该 slave 返回给 E902。

全空间内存属性的配置在平头哥自定义的 sysmap.h 文件完成。sysmap.h 文件会随着 E902 代码一同交付给客户，客户需要在系统设计阶段完成对内存空间属性的设置，并将该文件放入项目文件列表中，一经设定，软件无法修改。

sysmap.h 支持对 8 个内存地址空间的属性设定，第 **i**（**i** 从 0 到 7）个地址空间地址上限（不包含）由宏 SYSMAP_BASE_ADDR**i**（****i** 从 *0* 到 *7*）定义，地址下限（包含）由 SYSMAP_BASE_ADDR(**i**-1) 定义，具体为 SYSMAP_BASE_ADDR(**i**-1) <= 第 i 个地址空间地址 < SYSMAP_BASE_ADDR**i**。第 0 个地址空间下限是 0x0，内存地址不在 sysmap.h 文件设定的 8 个地址区间的地址属性默认为 cacheable 和 bufferable。每个地址空间上下边界是 4KiB 对齐，因此宏 SYSMAP_BASE_ADDR**i** 定义的是地址的高 20 位。

落在第 **i***(***i** 从 *0* 到 *7)* 个地址空间内的地址的属性由宏 SYSMAP_FLAG**i***(***i** 从 *0* 到 *7)* 定义，具体如 图 7.1 所示：

4	3	2	1	0
-	Cacheable	Bufferable	-	

图 7.1: sysmap.h 地址属性

第八章　总线矩阵与总线接口

8.1　简介

E902 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 8.1 所示。

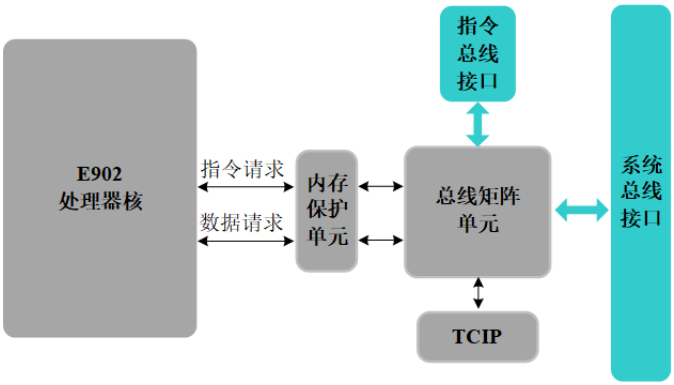


图 8.1: E902 总线矩阵

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，E902 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 1MiB 到 4GiB 可配置，例如设置指令总线地址空间大小为 8MiB，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 8.1 。

表 8.1: 指令总线对基地址和地址对齐的要求

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

E902 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 4GiB 空间。

8.2　系统总线接口

E902 的系统总线接口支持 AMBA3.0 AHB-Lite 协议，请参考 AMBA 3.0 规格说明—AMBA3 AHB-Lite Protocal Specification Rev 1.0。E902 的系统总线接口只实现了 AHB-Lite 协议中的部分内容。

在 AHB-Lite 协议下，作为主设备，总线接口支持的传输类型为：

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

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。

8.3　指令总线接口

E902 的指令总线只支持 AMBA3.0 AHB-Lite 协议，可参考 AMBA 3.0 规格说明- AMBA3 AHB-Lite Protocal Specification Rev 1.0。E902 指令总线接口只实现了 AHB-Lite 协议中的部分内容。

在 AHB-Lite 协议下，作为主设备，总线接口支持的传输类型为：

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

在 AHB-Lite 协议下，总线接口接受从设备的响应类型为：

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

第九章　CLINT 中断

E902 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。

9.1　寄存器地址映射

CLINT 中断占据 64KiB 内存空间。高 16 位为固定分配地址 0xE000，低 16 位地址映射如 表 9.1 所示。所有寄存器仅支持字对齐的访问。

表 9.1: 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	MTIMECMPLO	读/写	0xFFFFFFFF	机器模式计时器： 比较值寄存器 (低 32 位)
0x4004	MTIMECMPHI	读/写	0xFFFFFFFF	机器模式计时器： 比较值寄存器 (高 32 位)。
Reserved	-	-	-	-
0xBFF8	MTIMELO	读	0x00000000	机器模式计时器： 当前值寄存器（低 32 位），该寄存器值为 pad_cpu_sys_cnt[31:0] 信号的值。
0xBFFC	MTIMEHI	读	0x00000000	机器模式计时器 当前值寄存器（高 32 位），该寄存器值为 pad_cpu_sys_cnt[63:32] 信号的值 *
Reserved	-	-	-	-

9.2　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。

MSIP-机器模式软件中断等待位：

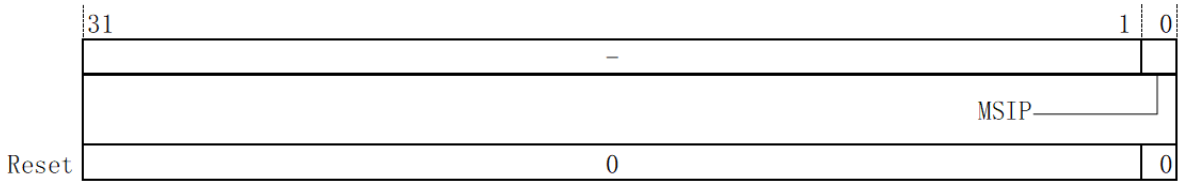


图 9.1: 机器模式软件中断配置寄存器（MSIP）

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。

9.3 计时器中断

CLINT 可用于生成机器模式计时器中断。该计时器中断需要搭配 E902 外部由系统设计实现的 64 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 E902 集成时需要将该 64 位系统计数器的值通过 pad_cpu_sys_cnt[63:0] 信号传入 E902 内部，高 32 位的值可通过 E902 设计实现的 MTIMEHI[31:0] 寄存器读取，低 32 位的值可通过 MTIMELO[31:0] 寄存器读取。

同时 E902 内部设计实现了一个 64 位的机器模式计时器比较值寄存器（*MTIMECMPL, MTIMECMPH*），这两个寄存器可以通过地址字对齐访问的方式读写。

CLINT 通过比较 {MTIMECMPH[31:0],MTIMECMPL[31:0]} 的值与系统计数器的当前值 {MTIMEHI[31:0],MTIMELO[31:0]} 确定是否产生计时器中断。当 {MTIMECMPH[31:0],MTIMECMPL[31:0]} 大于系统计数器的值时不产生中断；当 {MTIMECMPH[31:0],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 MTIMECMPH 和 MTIMECMPL 的值来清除对应的计时器中断。

机器模式下拥有修改或访问所有计时器中断相关寄存器的权限；普通用户模式没有权限。

每组寄存器结构相同，其寄存器位分布和位定义如 图 9.2 所示。

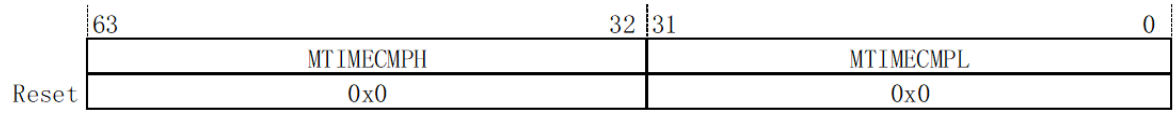


图 9.2: 机器模式计时器中断比较值寄存器（高/低）

MTIMECMPH/MTIMECMPL-机器模式计时器中断比较值寄存器高位/低位：

该寄存器存储了计时器比较值：

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

MTIMEHI/MTIMELO-机器模式计时器所用的系统计数器当前计数值的高位/低位：

该寄存器存储了系统计数器的当前值：

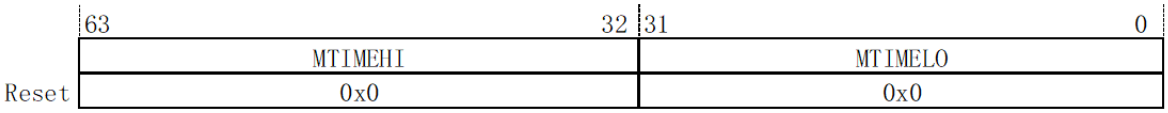


图 9.3: 机器模式计时器当前值寄存器（高/低）

- MTIMEHI：系统计数器当前计数值高 32 位；
- MTIMELO：系统计数器当前计数值低 32 位。

第十章　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。E902 实现的 CLIC 单元基本功能如下：

- 支持 RISC-V 标准 CLIC spec-0.8 版本；
- 最多支持 240 个外部中断源可配，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 256 个中断处理；
- 中断优先级有效位 CLICINTCTLBITS 2-5 可配，最多 32 个级别的中断优先级；
- 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；
- 支持了可选的 MSCRATCHCSW，MSCRATCHCSWL 寄存器，供中断处理函数内快速交换栈指针使用。

10.1　中断处理机制

10.1.1　中断仲裁

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）；
- 中断优先级大于 0；
- 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 16~255。

10.1.2　中断请求与响应

当 CLIC 仲裁产生有效的中断请求，且优先级大于寄存器 MINTTHRESH 设定的中断阈值时，会向 CPU 发起中断请求。

当 CPU 收到有效中断请求，且优先级大于核内正在响应中断的优先级，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，若中断服务程序中对 MNXTI 寄存器进行读操作，该读操作会从 CLIC 获得一个 ID 并生成对应的中断服务程序入口地址，表示当前 CLIC 仲裁出的中断的服务程序入口地址。CPU 根据所获得的地址进行跳转操作。如果从 MNXTI 寄存器读取的值为 0，表示没有有效中断请求。当 CLIC 收到 CPU 对 MNXTI 寄存器有效的读操作后，根据相应 ID 清除对应的中断等待位。

10.2 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 10.1 。CLIC 地址映射所示。其中，CLICCFG, CLICINFO, MINTTHRESH 寄存器仅支持地址字对齐的访问。

表 10.1: 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
0xE0800008	MINTTHRESH	RW	0x0	中断阈值寄存器
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

10.3 CLIC 寄存器描述

10.3.1 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 10.1 所示。

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

图 10.1: 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 E902 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：

CLICINTCTL[i] 中作为中断优先级的位数。CLIC 产生的中断优先级位数为固定 8 位，不足的部分由 1 进行填充。

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+ 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。

10.3.2 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 10.2 所示。

31 25	24 21	20 13	12 0
-	CLICCTLBITS	version	num_interrupt

图 10.2: 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 256 个。

10.3.3 中断阈值寄存器（MINTTHRESH）

阈值寄存器定义了当前处于等待状态的中断请求能够向 CPU 流水线核心发起中断请求的优先级临界值。中断阈值寄存器为每一个特权模式提供了一个 8 位的域作为相应的中断阈值。处于等待状态的中断请求的优先级必须高于 MINTTHRESH 寄存器定义的阈值才能向处理器发起中断。

寄存器位分布和位定义如 图 10.3 所示。

mth-机器模式阈值：

机器模式中断的阈值。由于 E902 仅支持机器模式中断，所以仅有一个阈值域。

	31 24	23 0
	mth	-
Reset	0	0

图 10.3: 中断阈值寄存器 (MINTTHRESH)

	7 1	0
	-	IP
Reset	0	0

图 10.4: 中断等待寄存器 (CLICINTIP)

10.3.4 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 10.4 所示。

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，软件建议通过执行一条 CSR 访问指令同时产生对 MNXTI 寄存器有效的读写操作来获取中断等待状态，同时该操作可以清除在等待响应的对应的中断，CPU 去跳转处理中断。如果不是采用读写 MNXTI 寄存器的方式去获取非硬件矢量模式的中断而是该中断主动将中断信息传递给 CPU 流水线核心，那么 CPU 在响应该中断时硬件无法清除其中断等待状态，需要通过软件清除。

10.3.5 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 10.5 所示。

	7 1	0
	-	IE
Reset	0	0

图 10.5: 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。

10.3.6 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 10.6 所示。

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

图 10.6: 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 E902 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。

10.3.7 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 10.7 所示。

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

图 10.7: 中断控制寄存器 (CLICINTCTL)

int__ctl-参与仲裁优先级：

该域有效位为 CLICINTCTLBITS 位（2-5 硬件可配置）。有效位域内的数值全部作为 CLIC 内部进行中断仲裁的优先级，但仅有高 CLICCFG.nlbits 位用于传递给 CPU 流水线核心表征仲裁出的中断的优先级（该优先级用于更新 MINTSTATUS 寄存器的 mil 域）。

CLICCFG.nlbits 与最终 CLIC 传递给 CPU 的中断优先级编码不完全示例如 表 10.2 所示。

当 nlbits > CLICINTCTLBITS 时，nlbits 被认为等于 CLICINTCTLBITS。

表 10.2: nlbits 与 CPU 保存中断优先级编码示例表

nlbits	CLICINTCTL 编码	可配中断优先级
0	xxxxxxxx	255
1	lxxxxxxx	127, 255
2	llxxxxxx	63, 127, 191, 255
3	lllxxxxx	31, 63, 95, 127, 159, 191, 223, 255
4	**llllxxxx	15, 31, 47, 63, 79, 95, 111,** 127, 143, 159, 175, 191, 207, 223, 239, 255	

注解： CLICINTCTL 编码一栏 “l “代表 CLIC 传递给 CPU 的中断优先级高位有效位，8 位优先级中剩余低位（编码中 “x” 表示的位）值为 1。

CLICCFG.nlbits 与 CLICINTCTL 划分不完全示例如 表 10.3 所示：

表 10.3: nlbits 与 CLICINTCTL 划分示例表

CLICINTCTLBITS	nlbits	编码	可配中断优先级
2	2	llxxxxxx	63, 127, 191, 255
2	1	lpxxxxxx	127, 255
2	0	ppxxxxxx	255
3	2	llpxxxxx	63, 127, 191, 255
4	2	llppxxxx	63, 127, 191, 255
5	1	lppppxxx	127, 255

注解： 编码一栏 “l “代表 CLIC 传递给 CPU 流水线核心的中断优先级高位有效位；” l “和” p “组合起来代表参与 CLIC 内部中断优先级仲裁的位；x 代表实际 CLICINTCTL 寄存器中硬件绑 1 的位。

第十一章　调试接口

11.1　概述

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

E902 调试接口包括两线通信协议，两线接口控制器。E902 支持 T-HEAD 自定义的两线调试接口，调试接口使用自定义两线接口协议与外部的调试器通信。

调试接口的主要特性如下：

- 支持两线制调试接口；
- 非侵入式获取 CPU 状态；
- 支持同步调试和异步调试，保证在极端恶劣情况下使处理器进入调试模式；
- 支持软断点；
- 可以设置多个硬断点；
- 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；
- 可在在普通用户模式下进入调试模式；
- 可使用 CPU 内部 TCIP 接口访问调试寄存器资源，详见*HAD 直接访问内存功能（DDMA）* 节；
- 支持 JTAG 或 TCIP 接口直接操作 HAD 寄存器发起内存访问请求。

E902 的调试工作是调试软件，调试代理服务程序，调试器和调试接口一起配合完成的，调试接口在整个 CPU 调试环境中的位置如 图 11.1 所示。其中，调试软件和调试代理服务程序通过网络互联，调试代理服务程序与调试器通过 USB 连接，调试器与 CPU 的调试接口以 JTAG 接口通信。

11.2　外部接口

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

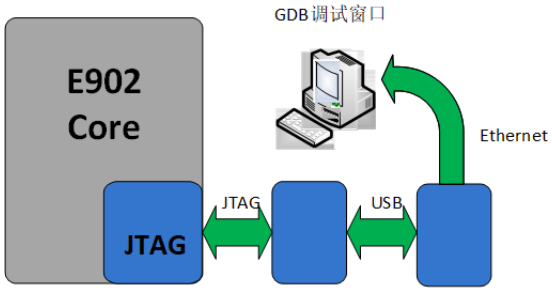


图 11.1: 调试接口在整个 CPU 调试环境中的位置

表 11.1: 调试模块与外部的接口信号

信号名	方向
had_pad_jdb_pm[1:0]	输出
pad_had_jtg_tclk	输入
pad_had_jtg_trst_b	输入
pad_had_jtg_tms_i	输入
had_pad_jtg_tms_o	输出
had_pad_htg_tms_oe	输出

- **had_pad_jdb_pm[1:0]**

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

表 11.2: had_pad_jdb_pm 指示当前 CPU 状态

had_pad_jdb_pm[1:0]	说明
00	普通模式
01	低功耗模式
10	调试模式
11	保留

- **pad_had_jtg_tclk**

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

- **pad_had_jtg_trst_b**

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

- **JTAG_2 相关信号**

pad_had_jtg_tms_i 信号为 2 线制 JTAG 串行数据输入信号，E902 调试接口在 JTAG 时钟信号 TCLK 的上升沿对其采样，而外部调试器在 JTAG 时钟的下降沿设置该信号；

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

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

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

11.3 HAD 直接访问内存功能（DDMA）

11.3.1 简介

为了提高调试效率，E902 在传统的调试单元访问内存方式的基础上，支持 HAD 直接访问内存。如 图 11.2 所示，路线 1 为调试单元访问内存的传统方式，首先需要驱动一条内存访问指令，如 LD、ST 等，由指令功能决定对内存的访问方式。如路线 2 所示，HAD 直接访问内存功能将绕过 CPU 中的取指单元和执行单元，直接向总线发出传输请求，拥有更高的内存访问速度，并且降低了 HAD 内存访问操作对 CPU 状态的影响。此功能的另外一个特点是允许 HAD 在非调试模式下执行内存访问，HAD 寄存器 HCR 的第 29 位 DDAE 打开时，HAD 能发起对内存的直接访问请求。

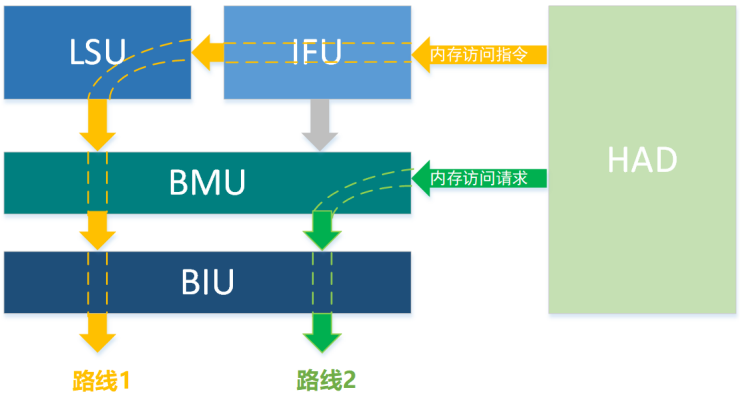


图 11.2: DDMA 功能示意图

11.3.2 DDMA 寄存器定义

1. **DACSR：** DACSR 决定每次传输的类型、size 以及 prot。在每次传输开始前都需要将此寄存器配置完成。

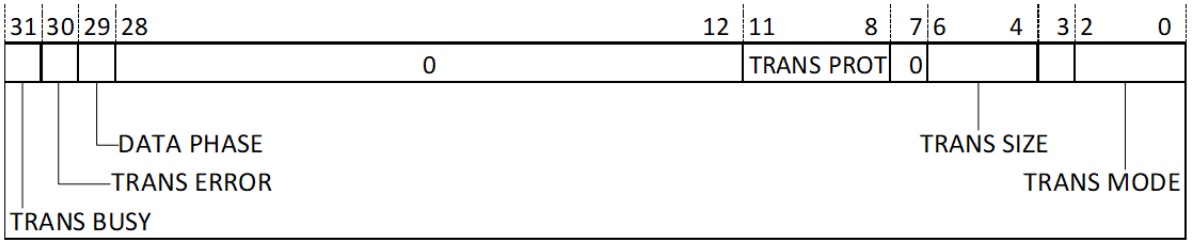


图 11.3: DACSR 寄存器

表 11.3: DACSR 表项说明

域	权限	描述
TRANS MODE[2:0]	R/W	3’b000：single，单次传输。 3’b001：increase，地址递增传输： • Size 为 byte：传输完成后 DATR 加 1。 • Size 为 half word：传输完成后 DATR 加 2。 • Size 为 word：传输完成后 DATR 加 4。 3’b010~3’b111: Reserved。
TRANS SIZE[2:0]	R/W	3’b000：byte； 3’b001：half word； 3’b010：word； 3’b011~3’b111: preserved。
TRANS PROT[3:0]	R/W	***0：取指令； ***1：数据访问； **0*：用户模式访问； **1*：机器模式访问； *0**：Non secure； *1**：Secure； 0***：Non cacheable； 1***：cacheable。 Secure 和 non secure 域不可写，固定为 Non secure。
TRANS BUSY	R	1：前次传输未完成； 0：传输为空。
TRANS ERROR	R	1：前次传输发生异常； 0：前次传输正常完成。
DATA PHASE	R/W	读传输数据阶段位： 1：读 DARWR 将不会发起新的总线读传输； 0：读 DARWR 将会发起新的总线读传输。 硬件置 1：当传输类型为 single 时，由本调试单元发起的一次总线读传输正常完成后，此位将被硬件置为 1。 硬件清 0：读 DARWR 后会被硬件清 0。

2. **DATR：** DATR 直接内存访问目的地址寄存器

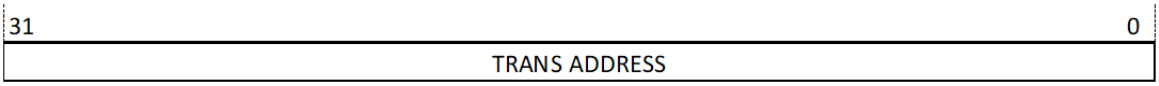


图 11.4: DATR 寄存器

该寄存器由 32 位组成，记录了访问内存地址，下一次传输的地址即为此数值。如果是递增传输，每一次传输完成地址寄存器会自动更新。递增方式参照 TRANS MODE 中的描述。

3. **DARWR：**读写寄存器

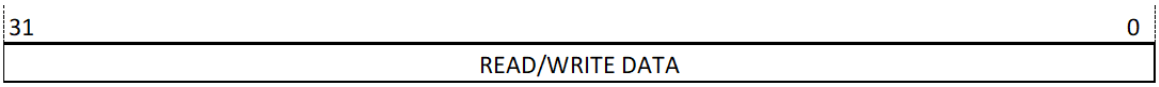


图 11.5: DARWR 寄存器

此寄存器有两个功能：

1). 通过读或写这组寄存器来驱动一次总线的读或写传输，详细情况参见 表 11.4 。

2). 通过读此寄存器，来得到上一次读传输完成后的数据。

表 11.4: DARWR 驱动总线传输

对 DARWR 的操作	DATA PHASE	TRANS MODE	发起的传输
读	0	Single	读
读	0	Increase	读
读	1	- (no care)	无
写	-	-	写

对于写操作来说，此寄存器只有单一的功能：驱动总线发起一次写传输操作。写传输地址为 DATR 的数值，写传输的数据为写入此寄存器的数值。

对于读操作来说，一次读有可能触发一次总线读操作，并且把 DARWR 的数据读出。如果不想发起总线传输，需要把 DATA PHASE 位置起。Single 的读传输正常完成后，DATA PHASE 位将自动被置 1。当 DATA PHASE 位为 1 时，下一次读 DARWR 寄存器时，触发总线读传输的能力将被屏蔽，同时 DATA PHASE 位被清为 0。

11.3.3 DDMA 操作示例

1. 对地址 32’hABCD 发起单次读操作，size：word，prot：4’b0。
    1. 配置 HCR 寄存器，置 DDAE 位为 1；
    2. Set DATR = 32’hABCD；
    3. Set DACSR = 32’h20；(传输类型：single)；
    4. Read DARWR；(总线发起读传输，完成后 read phase 位置 1)；

5. Read DARWR。(读回总线返回数据，read phase 位置 0)；

在第 3 和第 4 步之间，可以增加一步 read DACSR，通过 busy 和 error 两位用来检查之前的传输是否已经正常完成。

2. 对地址 32’hABCD 发起连续 5 次 increase 读操作，size：word，prot：4’b0。
   1. 配置 HCR 寄存器，置 DDAE 位为 1；
   2. Set DATR = 32’hABCD；
   3. Set DACSR = 32’h21；(传输类型：increase)；
   4. Read DARWR；(第一次传输开始)；
   5. Read DARWR；(第二次传输开始，第一个数据得到)；

   # Read DARWR；(第三次传输开始，第二个数据得到)；
   1. Read DARWR；(第四次传输开始，第三个数据得到)；
   2. Read DARWR；(第五次传输开始，第四个数据得到)；
   3. Read DARWR。(第五个数据得到，多发起一次读传输，如要避免，需要在第 8 步之后设置 DACSR 中的 DATA PHASE 位为 1)
3. 对地址 32’hABCD 发起连续 5 次 increase 写操作，size：word，prot：4’b0。
   1. 配置 HCR 寄存器，置 DDAE 位为 1；
   2. Set DATR = 32’hABCD；
   3. Set DACSR = 32’h21；(传输类型：increase)；
   4. write DARWR；(第一次写传输开始)；
   5. write DARWR；(第二次写传输开始)；
   6. write DARWR；(第三次写传输开始)；
   7. write DARWR；(第四次写传输开始)；
   8. write DARWR。(第五次写传输开始)。

第十二章　功耗管理

E902 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，E902 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。WFI 指令只有 E902 处于机器模式下可以被正常执行，用户模式下执行该指令会触发非法指令异常。

在低功耗模式下，E902 不会向总线发起数据传输请求，内部流水线停顿。

12.1　低功耗模式

E902 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 E902 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*）。

E902 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。

12.2　低功耗唤醒

E902 的低功耗唤醒支持如下请求类型：

- 调试请求；
- NMI 请求；
- 中断请求；
- 外部事件。

E902 扩展实现了 MEXSTATUS 寄存器的 WFE 域来指示唤醒 CPU 的请求类型，当该域值为 1 时，上述所有请求都可以唤醒处于低功耗模式的 CPU，当该域值为 0 时，除了外部事件，其他请求均可以唤醒 CPU，但是在使用中断请求进行唤醒时，要求该中断的优先级大于 MINTSTATUS.MIL 的优先级才可以唤醒 CPU。该域复位值为 1。

在 NMI 请求唤醒 CPU 后，CPU 会去响应 NMI 请求进而处理该请求。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，根据唤醒中断的优先级与 MINTSTATUS.MIL 的大小比较结果决定是否响应该中断还是执行 WFI 指令的后续指令。

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。

第十三章　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。

13.1　PMP 设置示例

```
/* 设置区域 0 地址模式，大小和起始地址，地址模式 NAPOT，大小 128KiB
/* 区域 0 控制地址区间 [0x0, 0x00020000)

li t1,0x3fff
csrw pmpaddr0,t1
```

注解： pmpaddr 根据地址大小和起始地址进行计算

```
/* 设置区域 1 地址模式，大小和起始地址，地址模式 NAPOT，大小 128B
/* 区域 1 控制地址区间 [0x01000000, 0x01000080)

li t1,0x40000f
csrw pmpaddr1,t1
```

注解： li t1, 0x400000”也可达到同样的效果

```
/* 设置区域 2 地址模式，大小和起始地址，地址模式 TOR
/* 区域 2 控制地址区间 [0x01000000,0x01100000)。访问 [0x01000000，0x01000080) 会同时命中区域 1 和区域 2，此时区域 1 优先级更高，将以区域 1 的访问属性做出判断。当区域 1 地址匹配模式改为 OFF 时，访问 [0x01000000,0x01100000) 将全部命中区域 2，以区域 2 的访问属性做出判断。

li t1,0x440000
csrw pmpaddr2,t1
```

```
/* 设置区域 3 地址模式，大小和起始地址，地址模式 NAPOT，大小 8KiB
/* 区域 3 控制地址区间 [0x01200000, 0x01202000)

li t1,0x4803ff
csrw pmpaddr3,t1
```

```
/* 设置区域 0~3 的属性
/* 区域 0 lock=1 ，可执行，可写，可读 0x9f (地址模式 NAPOT)
/* 区域 1 lock=0 ，不可执行，可写，可读 0x1b(地址模式 NAPOT)
/* 区域 2 lock=1 ，可执行，可写，不可读 0x8e(地址模式 TOR)
/* 区域 3 lock=0 ，可执行，不可写，可读 0x1d(地址模式 NAPOT)

li t1, 0x1d8e1b9f
csrw pmpcfg0,t1
```

注解： pmpcfg 根据地址属性进行计算

```
/* 设置区域 4 地址模式，大小和起始地址，地址模式 OFF
/* 所有地址不会命中该表项

li t1,0x4c0000
csrw pmpaddr4,t1
```

```
/* 设置区域 5 地址模式，大小和起始地址，地址模式 NAPOT，大小 16KiB
/* 区域 5 控制地址区间 [0x01400000, 0x01404000)

li t1,0x5007ff
csrw pmpaddr5,t1
```

```
/* 设置区域 6 地址模式，大小和起始地址，地址模式 NAPOT，大小 32KiB
/* 区域 6 控制地址区间 [0x01500000, 0x01508000)

li t1,0x540fff
csrw pmpaddr6,t1
```

```
/* 设置区域 7 地址模式，大小和起始地址，地址模式 NAPOT，大小 64KiB
/* 区域 7 控制地址区间 [0x01600000, 0x01610000)

li t1,0x581fff
csrw pmpaddr7,t1
```

```
/* 设置区域 4~7 的属性
/* 区域 4 lock=1 ，不可执行，可写，可读 0x83(地址模式 OFF)
/* 区域 5 lock=0 ，可执行，可写，不可读 0x1e(地址模式 NAPOT)
/* 区域 6 lock=1 ，可执行，不可写，可读 0x9d(地址模式 NAPOT)
/* 区域 7 lock=0 ，不可执行，可写，可读 0x1b(地址模式 NAPOT)

li t1,0x1b9d1e83
csrw pmpcfg1,t1

/* 区域 8~15 配置方式和上述相同，分别对应
/* pmpaddr8, pmpaddr9, pmpaddr10, pmpaddr11, pmpcfg2
/* pmpaddr12, pmpaddr13, pmpaddr14, pmpaddr15, pmpcfg3
```

13.2　高速缓存设置示例

E902 支持高速缓存扩展寄存器：机器模式硬件配置寄存器 (mhcr)。可以实现对指令高速缓存的开关。关于系统中可 Cache 地址空间的设置请参考内存模型 的描述。

```
// 当 IE 为 0 时，指令高速缓存关闭；当 IE 为 1 时，指令高速缓存开启。

li t1, 1 //设置 icache enable 打开，指令可高速缓冲
csrrs t3, mhcr, t1

//设置 mxstatus.theadisaee 为 1，使用 cache 指令使 icache 全部无效化

la t1, 0x400000
csrrs t3, mhcr, t1

//invalid icache

ICACHE.IALL
```

13.3　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
```

(下页继续)

(续上页)

```
//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
```

13.4　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
```

13.5　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
```

13.6　异常和中断服务程序入口地址设置示例

异常和中断服务程序的入口地址设置分两个步骤:

步骤 1:

设置异常向量表基地址和模式，写入 MTVEC，E902 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
```

(下页继续)

(续上页)

```
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
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
```

(下页继续)

(续上页)

```
.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 EXTERNAL_INT_HANDLER
//11 号 clint 外部中断，通过 pad_cpu_ext_int_b 接入
.rept 4
.long 0x0 //12-15 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

 //最多可实现接入 240 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。

第十四章　附录 A 标准指令术语

E902 实现了 RV32EMC 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。

14.1　附录 A-1 E 指令术语

以下是对 E902 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 *A-3 C* 指令术语 。

14.1.1　ADD——有符号加法指令

语法：

add rd, rs1, rs2

操作：

rd ← rs1 + rs2

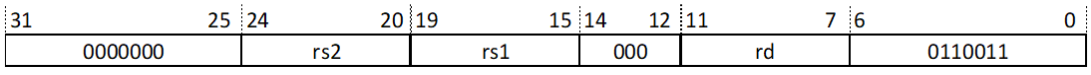
执行权限：

M mode/U mode

异常：

无

指令格式：



14.1.2　ADDI——有符号立即数加法指令

语法：

addi rd, rs1, imm12

操作：

rd ← rs1 + sign_extend(imm12)

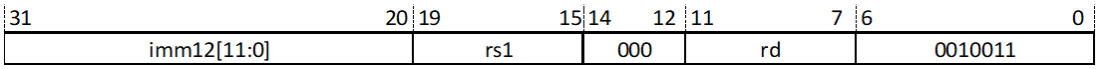
执行权限：

M mode/U mode

异常：

无

指令格式：



14.1.3 AND——按位与指令

语法：

and rd, rs1, rs2

操作：

rd ← rs1 & rs2

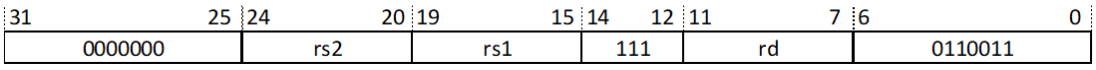
执行权限：

M mode/U mode

异常：

无

指令格式：



14.1.4 ANDI——立即数按位与指令

语法：

andi rd, rs1, imm12

操作：

rd ← rs1 & sign_extend(imm12)

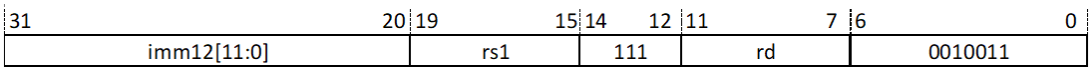
执行权限：

M mode/U mode

异常：

无

指令格式：



14.1.5　AUIPC——PC 高位立即数加法指令

语法：

auipc rd, imm20

操作：

rd ← current pc + imm20<<12

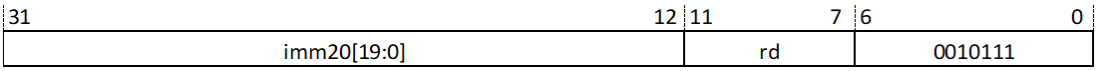
执行权限：

M mode/U mode

异常：

无

指令格式：



14.1.6　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

执行权限：

M mode/U mode

异常：

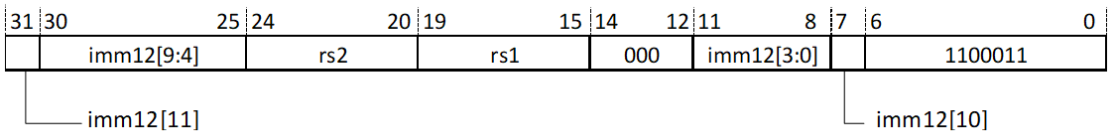
无

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：



14.1.7　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

执行权限：

M mode/U mode

异常：

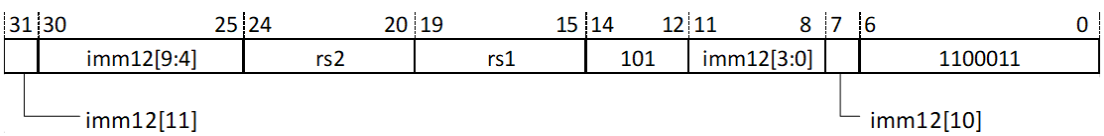
无

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：



14.1.8　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

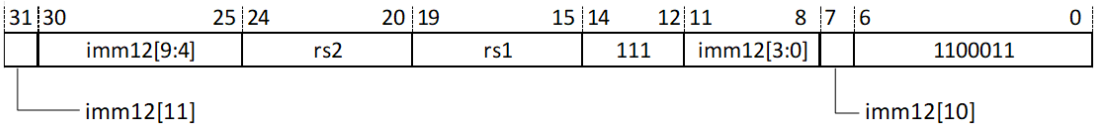
异常：

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：



14.1.9　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

执行权限：

M mode/U mode

异常：

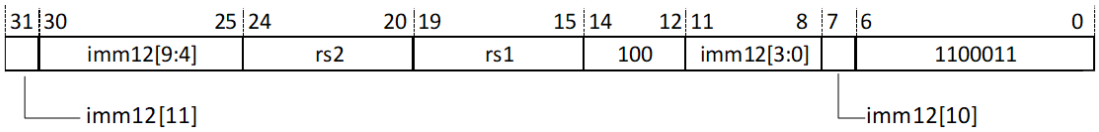
无

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：



14.1.10　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

执行权限：

M mode/U mode

异常：

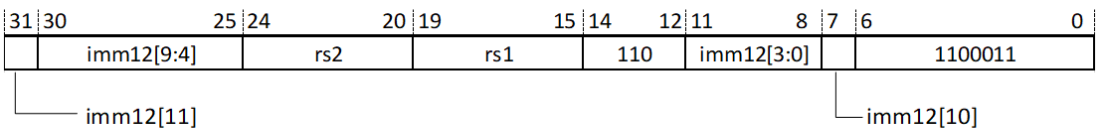
无

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：



14.1.11　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

执行权限：

M mode/U mode

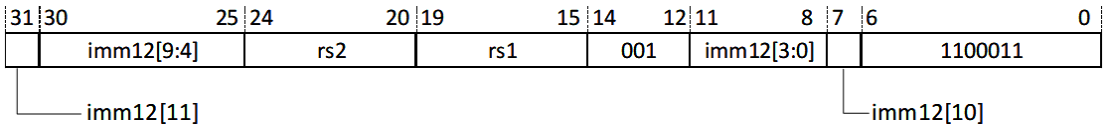
异常：

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：



14.1.12　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)

执行权限：

M mode/U mode

异常：

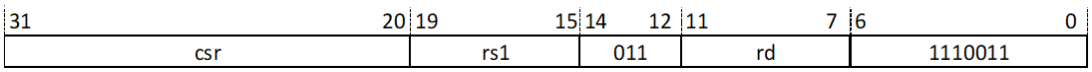
非法指令异常

说明：

各权限下允许访问的控制寄存器不同，具体请参考控制寄存器章节。

当 rs1=x0 时，该指令不产生写操作，不会产生写行为引发的异常。

指令格式：



14.1.13　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)

执行权限：

M mode/U mode

异常：

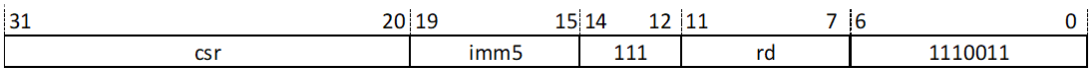
非法指令异常

说明：

各权限下允许访问的控制寄存器不同，具体请参考控制寄存器章节。

当 rs1=x0 时，该指令不产生写操作，不会产生写行为引发的异常。

指令格式：



14.1.14　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

执行权限：

M mode/U mode

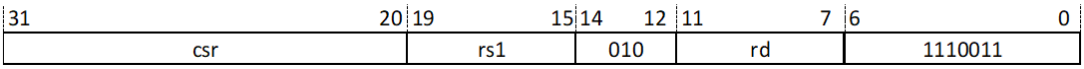
异常：

非法指令异常

说明：

各权限下允许访问的控制寄存器不同，具体请参考控制寄存器章节。
当 rs1=x0 时，该指令不产生写操作，不会产生写行为引发的异常。

指令格式：



14.1.15　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)

执行权限：

M mode/U mode

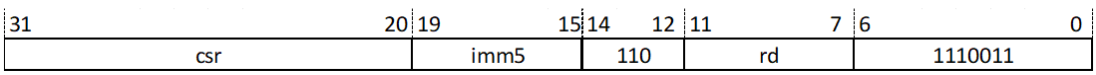
异常：

非法指令异常

说明：

各权限下允许访问的控制寄存器不同，具体请参考控制寄存器章节。
当 rs1=x0 时，该指令不产生写操作，不会产生写行为引发的异常。

指令格式：



14.1.16　CSRRW——控制寄存器读写传送指令

语法：

csrrw rd, csr, rs1

操作：

rd ← csr
csr ← rs1

执行权限：

M mode/U mode

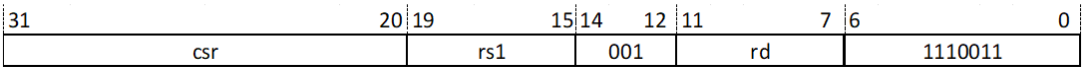
异常：

非法指令异常

说明：

各权限下允许访问的控制寄存器不同，具体请参考控制寄存器章节。
当 rs1=x0 时，该指令不产生写操作，不会产生写行为引发的异常。

指令格式：



14.1.17　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]

执行权限：

M mode/U mode

异常：

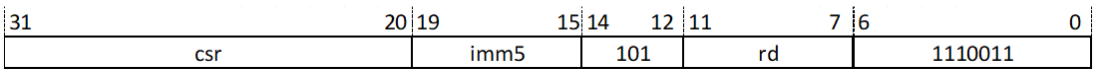
非法指令异常

说明：

各权限下允许访问的控制寄存器不同，具体请参考控制寄存器章节。

当 rs1=x0 时，该指令不产生写操作，不会产生写行为引发的异常。

指令格式：



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

14.1.18 EBREAK——断点指令

语法：

ebreak

操作：

产生断点异常或者进入调试模式

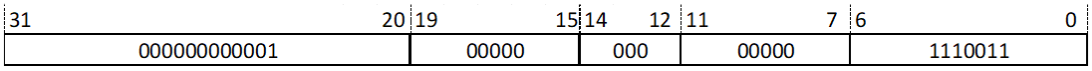
执行权限：

M mode/U mode

异常：

断点异常

指令格式：



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

14.1.19 ECALL——环境异常指令

语法：

ecall

操作：

产生环境异常

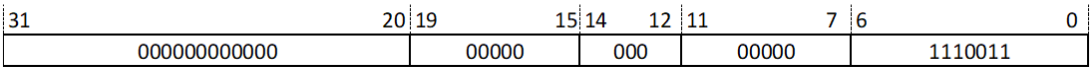
执行权限：

M mode/U mode

异常：

用户模式环境调用异常、机器模式环境调用异常

指令格式：



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

14.1.20　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。

执行权限：

M mode/U mode

异常：

无

说明：

pi=1, so=1，指令语法为 fence i,o，以此类推

指令格式：

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

14.1.21　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。

执行权限：

M mode/U mode

异常：

无

指令格式：

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

14.1.22　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 ← currect pc + 4

执行权限：

M mode/U mode

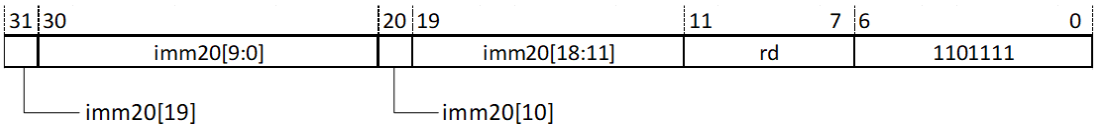
异常：

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：



14.1.23　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 ← currect pc + 4

执行权限：

M mode/U mode

异常：

无

说明：

指令跳转范围为全部 4GiB 地址空间

指令格式：

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

14.1.24　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])

执行权限：

M mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常

指令格式：

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

14.1.25　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])

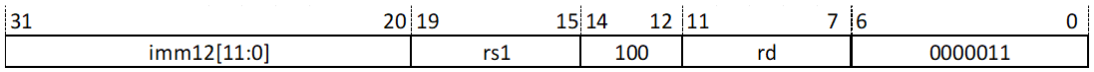
执行权限：

M mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常

指令格式：



14.1.26　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])

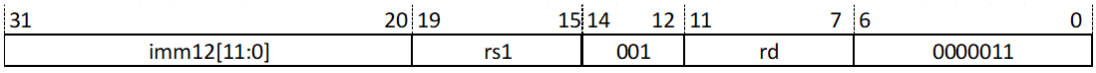
执行权限：

M mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常

指令格式：



14.1.27　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])

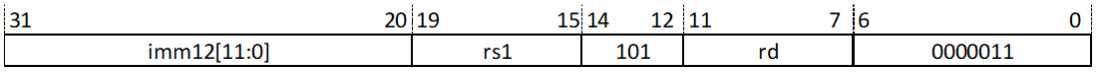
执行权限：

M mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常

指令格式：



14.1.28　LUI——高位立即数装载指令

语法：

lui rd, imm20

操作：

rd←imm20<<12

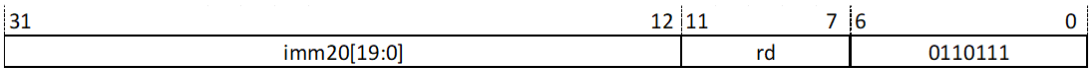
执行权限：

M mode/U mode

异常：

无

指令格式：



14.1.29　LW——字加载指令

语法：

lw rd, imm12(rs1)

操作：

address←rs1+sign_extend(imm12)

rd ← mem[(address+31):address]

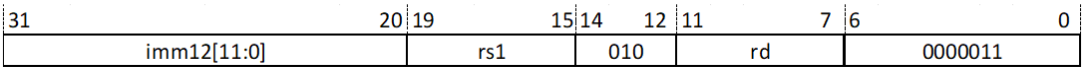
执行权限：

M mode/U mode

异常：

加载指令非对齐访问异常、加载指令访问错误异常

指令格式：



14.1.30　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

执行权限：

M mode

异常：

非法指令异常

指令格式：

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

14.1.31　OR——按位或指令

语法：

or rd, rs1, rs2

操作：

rd ← rs1 | rs2

执行权限：

M mode/U mode

异常：

无

指令格式：

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

14.1.32　ORI——立即数按位或指令

语法：

ori rd, rs1, imm12

操作：

rd ← rs1 | sign_extend(imm12)

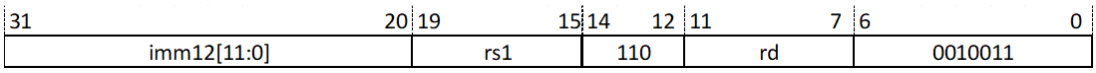
执行权限：

M mode/U mode

异常：

无

指令格式：



14.1.33　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]

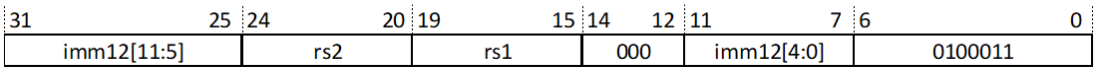
执行权限：

M mode/U mode

异常：

存储指令非对齐访问异常、存储指令访问错误异常

指令格式：



14.1.34　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]

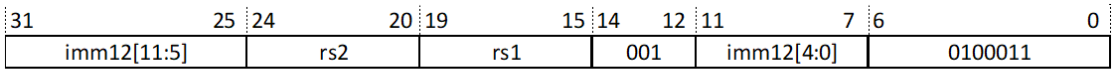
执行权限：

M mode/U mode

异常：

存储指令非对齐访问异常、存储指令访问错误异常

指令格式：



14.1.35　SLL——逻辑左移指令

语法：

sll rd, rs1, rs2

操作：

rd← rs1 << rs2[4:0]

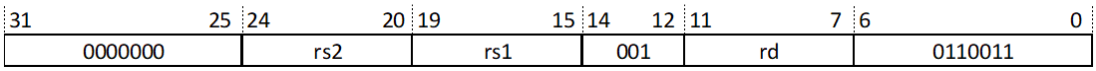
执行权限：

M mode/U mode

异常：

无

指令格式：



14.1.36　SLLI——立即数逻辑左移指令

语法：

slli rd, rs1, shamt5

操作：

rd← rs1 << shamt5

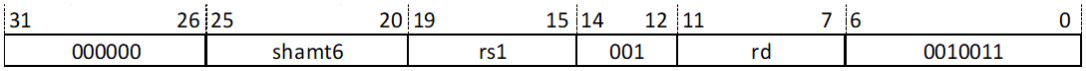
执行权限：

M mode/U mode

异常：

无

指令格式：



14.1.37　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

执行权限：

M mode/U mode

异常：

无

指令格式：

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

14.1.38　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

执行权限：

M mode/U mode

异常：

无

指令格式：

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

14.1.39　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 <zero_extend(imm12))
　　rd←1
else
　　rd←0

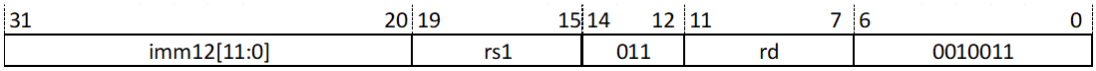
执行权限：

M mode/U mode

异常：

无

指令格式：



14.1.40　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

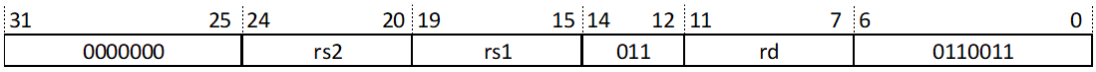
执行权限：

M mode/U mode

异常：

无

指令格式：



14.1.41　SRA——算术右移指令

语法：

sra rd, rs1, rs2

操作：

rd←rs1 >>> rs2[4:0]

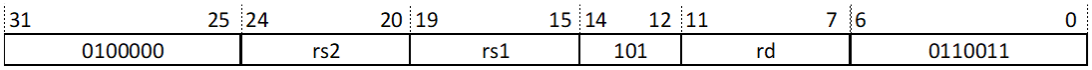
执行权限：

M mode/U mode

异常：

无

指令格式：



14.1.42　SRAI——立即数算术右移指令

语法：

srai rd, rs1, shamt5

操作：

rd← rs1 >>>shamt5

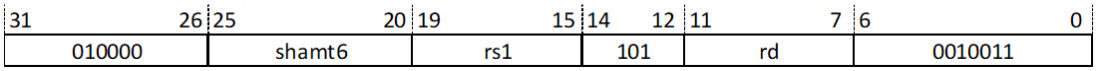
执行权限：

M mode/U mode

异常：

无

指令格式：



14.1.43　SRL——逻辑右移指令

语法：

srl rd, rs1, rs2

操作：

rd←rs1 >> rs2[4:0]

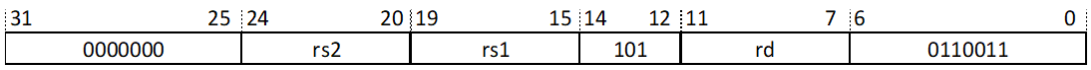
执行权限：

M mode/U mode

异常：

无

指令格式：



14.1.44　SRLI——立即数逻辑右移指令

语法：

srli rd, rs1, shamt5

操作：

rd← rs1 >> shamt5

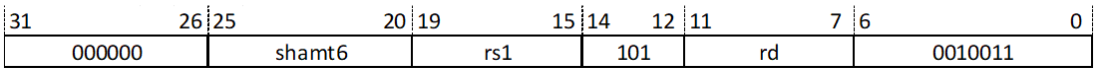
执行权限：

M mode/U mode

异常：

无

指令格式：



指令格式：

14.1.45　SUB——有符号减法指令

语法：

sub rd, rs1, rs2

操作：

rd ← rs1 - rs2

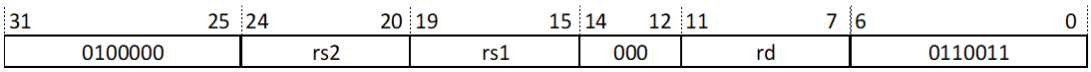
执行权限：

M mode/U mode

异常：

无

指令格式：



14.1.46　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]

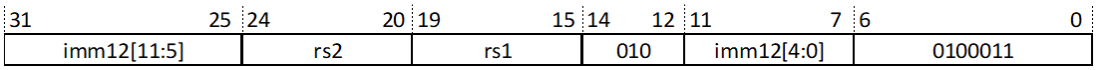
执行权限：

M mode/U mode

异常：

存储指令非对齐访问异常、存储指令访问错误异常

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

14.1.47　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

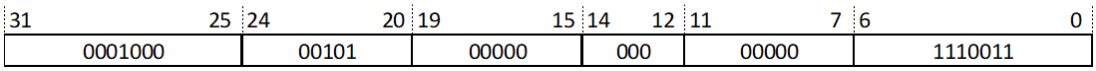
执行权限：

M mode/U mode

异常：

无

指令格式：



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

14.1.48　XOR——按位异或指令

语法：

xor rd, rs1, rs2

操作：

rd ← rs1 ^ rs2

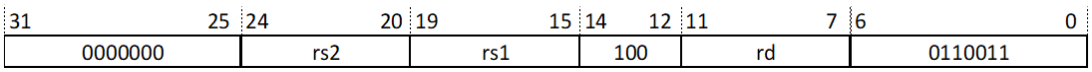
执行权限：

M mode/U mode

异常：

无

指令格式：



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

14.1.49　XORI——立即数按位异或指令

语法：

xori rd, rs1, imm12

操作：

rd ← rs1 & sign_extend(imm12)

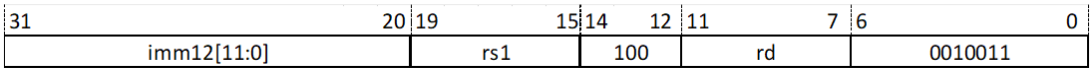
执行权限：

M mode/U mode

异常：

无

指令格式：



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

14.2　附录 A-2 M 指令术语

以下是对 E902 实现的 RV32M 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。

14.2.1　DIV——有符号除法指令

语法：

div rd, rs1, rs2

操作：

rd ← rs1 / rs2

执行权限：

M mode/U mode

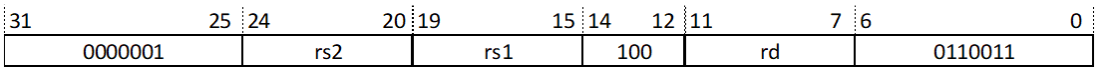
异常：

无

说明：

除数是 0 时，除法结果为 0xffffffff
产生 overflow 时，除法结果为 0x80000000

指令格式：



14.2.2　DIVU——无符号除法指令

语法：

divu rd, rs1, rs2

操作：

rd ← rs1 / rs2

执行权限：

M mode/U mode

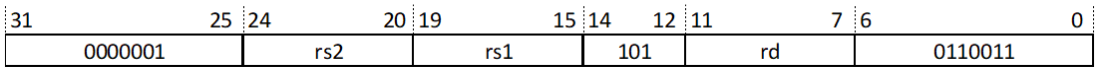
异常：

无

说明：

除数是 0 时，除法结果为 0xffffffff

指令格式：



14.2.3　MUL——有符号乘法指令

语法：

mul rd, rs1, rs2

操作：

rd ← (rs1 * rs2)[31:0]

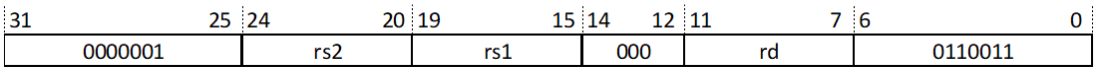
执行权限：

M mode/U mode

异常：

无

指令格式：



14.2.4　MULH——有符号乘法取高位指令

语法：

mulh rd, rs1, rs2

操作：

rd ← (rs1 * rs2)[63:32]

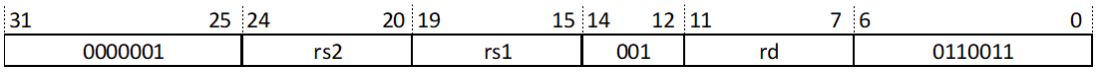
执行权限：

M mode/U mode

异常：

无

指令格式：



14.2.5　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]

执行权限：

M mode/U mode

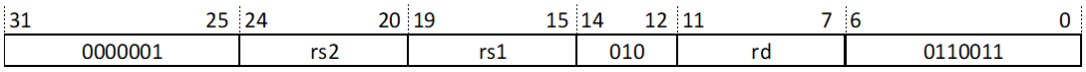
异常：

无

说明：

rs1 有符号数，rs2 无符号数

指令格式：



14.2.6　MULHU——无符号乘法取高位指令

语法：

mulhu rd, rs1, rs2

操作：

rd ← (rs1 * rs2)[63:32]

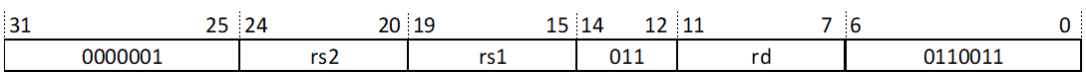
执行权限：

M mode/U mode

异常：

无

指令格式：



14.2.7　REM——有符号取余指令

语法：

rem rd, rs1, rs2

操作：

rd ← rs1 % rs2

执行权限：

M mode/U mode

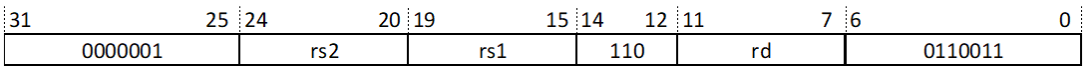
异常：

无

说明：

除数是 0 时，求余结果为被除数
产生 overflow 时，余数结果为 0x0

指令格式：



14.2.8　REMU——无符号取余指令

语法：

remu rd, rs1, rs2

操作：

rd ← rs1 % rs2

执行权限：

M mode/U mode

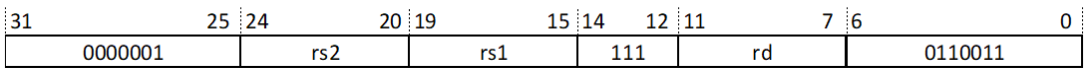
异常：

无

说明：

除数是 0 时，求余结果为被除数

指令格式：



14.3　附录 A-3 C 指令术语

以下是对 E902 实现的 RVC 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。

14.3.1　C.ADD——有符号加法指令

语法：

c.add rd, rs2

操作：

rd ← rs1 + rs2

执行权限：

M mode/U mode

异常：

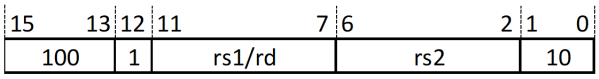
无

说明：

rs1 = rd != 0

rs2 ! = 0

指令格式：



14.3.2　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)

执行权限：

M mode/U mode

异常：

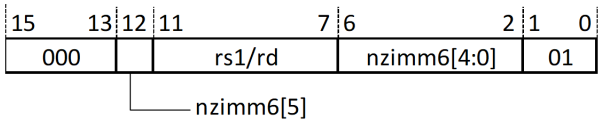
无

说明：

rs1 = rd != 0

nzimm6!=0

指令格式：



14.3.3 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)

执行权限：

M mode/U mode

异常：

无

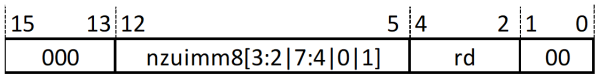
说明：

nzuimm8 != 0

rd 编码代表寄存器如下:

- 000 x8
- 001 x9
- 010 x10
- 011 x11
- 100 x12
- 101 x13
- 110 x14
- 111 x15

指令格式：



14.3.4　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)

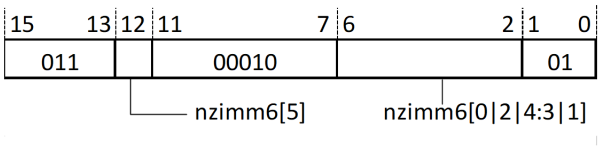
执行权限：

M mode/U mode

异常：

无

指令格式：



14.3.5　C.AND——按位与指令

语法：

c.and rd, rs2

操作：

rd ← rs1 & rs2

执行权限：

M mode/U mode

异常：

无

说明：

rs1 = rd

rd/rs1，rs2 编码代表寄存器如下:

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

- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

指令格式：

15 13	12	11 10	9 7	6 5	4 2	1 0
100	0	11	rs1/rd	11	rs2	01

14.3.6 C.ANDI——立即数按位与指令

语法：

c.andi rd, imm6

操作：

rd ← rs1 & sign_extend(imm6)

执行权限：

M mode/U mode

异常：

无

说明：

rs1 = rd

rd/rs1 编码代表寄存器如下:

- 000: x8
- 001: x9
- 010: x10
- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

指令格式：

15 13	12	11 10	9 7	6 2	1 0
100		10	rs1/rd	imm6[4:0]	01

imm6[5]

14.3.7　C.BEQZ——等于零分支指令

语法：

c.beqz rs1, label

操作：

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

执行权限：

M mode/U mode

异常：

无

说明：

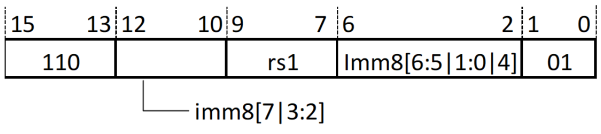
rs1 编码代表寄存器如下:

- 000: x8
- 001: x9
- 010: x10
- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

汇编器根据 label 算出 imm8

指令跳转范围为±128B 地址空间

指令格式：



14.3.8　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 + imm8<<1;

else

next pc = current pc + 2;

执行权限：

M mode/U mode

异常：

无

说明：

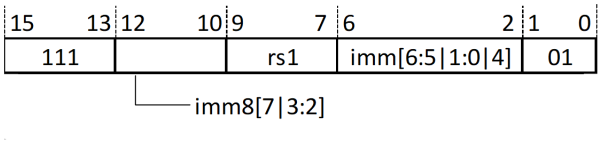
rs1 编码代表寄存器如下:

- 000: x8
- 001: x9
- 010: x10
- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

汇编器根据 label 算出 imm12

指令跳转范围为±256B 地址空间

指令格式：



14.3.9 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

执行权限：

M mode/U mode

异常：

断点异常

指令格式：

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

14.3.10　C.J——无条件跳转指令

语法：

c.j label

操作：

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

执行权限：

M mode/U mode

异常：

无

说明：

汇编器根据 label 算出 imm11
指令跳转范围为±2KB 地址空间

指令格式：

15 13	12 2	1 0
101	imm11[10|3|8:7|9|5|6|2:0|4]	01

14.3.11　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;

执行权限：

M mode/U mode

异常：

无

说明：

汇编器根据 label 算出 imm11

指令跳转范围为±2KB 地址空间

指令格式：

15 13 12 2 1 0

0 0 1	imm11[10|3|8:7|9|5|6|2:0|4]	0 1

14.3.12　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;

执行权限：

M mode/U mode

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

14.3.13　C.JR——寄存器跳转指令

语法：

c.jr rs1

操作：

next pc = rs1;

执行权限：

M mode/U mode

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	0	rs1	00000	10

14.3.14　C.LI——立即数传送指令

语法：

c.li rd, imm6

操作：

rd ←sign_extend(imm6)

执行权限：

M mode/U mode

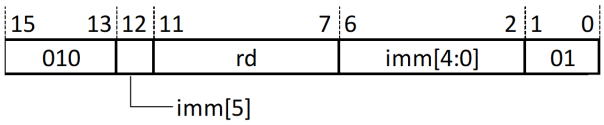
异常：

无

说明：

rd != 0。

指令格式：



14.3.15　C.LUI——高位立即数传送指令

语法：

c.lui rd, nzimm6

操作：

rd ←sign_extend(nzimm6<<12)

执行权限：

M mode/U mode

异常：

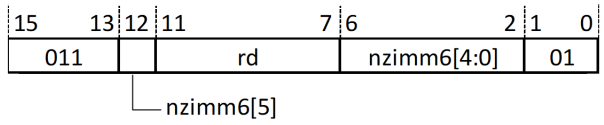
无

说明：

rd != 0。

nzimm6 != 0。

指令格式：



14.3.16　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]

执行权限：

M mode/U mode

异常：

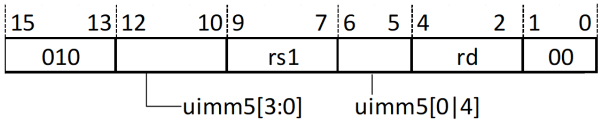
加载指令非对齐访问异常、加载指令访问错误异常

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：



14.3.17　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]

执行权限：

M mode/U mode

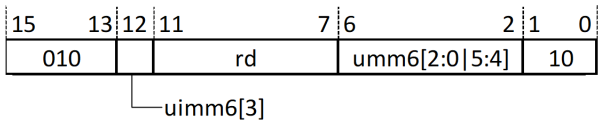
异常：

加载指令非对齐访问异常、加载指令访问错误异常

说明：

rd != 0

指令格式：



14.3.18　C.MV——数据传送指令

语法：

c.mv rd, rs2

操作：

rd ← rs2;

执行权限：

M mode/U mode

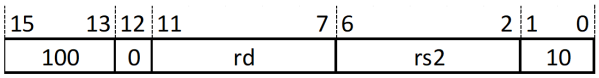
异常：

无

说明：

rs2 != 0, rd !=0。

指令格式：



14.3.19　C.NOP——空指令

语法：

c.nop

操作：

无操作

执行权限：

M mode/U mode

异常：

无

指令格式：

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

14.3.20　C.OR——按位或指令

语法：

c.or rd, rs2

操作：

rd ← rs1 | rs2

执行权限：

M mode/U mode

异常：

无

说明：

rs1 = rd

rd/rs1 编码代表寄存器如下:

- 000: x8
- 001: x9
- 010: x10
- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

指令格式：

15 13	12	11 10	9 7	6 5	4 2	1 0
100	0	11	rs1/rd	10	rs2	01

14.3.21　C.SLLI——立即数逻辑左移指令

语法：

c.slli rd, shamt5

操作：

rd ←rs1 << shamt5

执行权限：

M mode/U mode

异常：

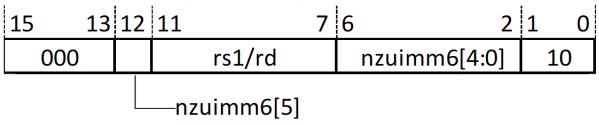
无

说明：

rs1==rd

rd/rs1 != 0，shamt5 != 0，shamt[5] = 0

指令格式：



14.3.22　C.SRAI——立即数算术右移指令

语法：

c.srli rd, shamt5

操作：

rd ←rs1 >>>shamt6

执行权限：

M mode/U mode

异常：

无

说明：

shamt5 != 0，shamt[5] = 0

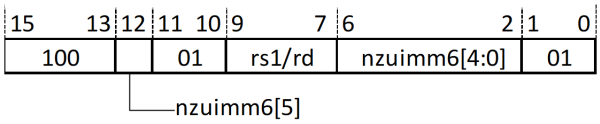
rs1 == rd

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：



14.3.23　C.SRLI——立即数逻辑右移指令

语法：

c.srli rd, shamt5

操作：

rd ←rs1 >> shamt5

执行权限：

M mode/U mode

异常：

无

说明：

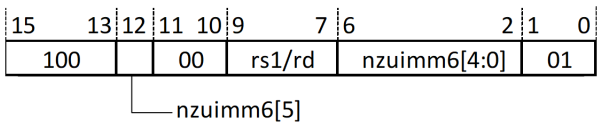
shamt5 != 0，shamt[5] = 0

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：



14.3.24　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

执行权限：

M mode/U mode

异常：

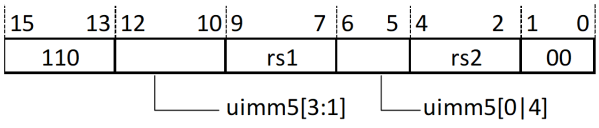
存储指令非对齐访问异常、存储指令访问错误异常

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：



14.3.25　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

执行权限：

M mode/U mode

异常：

存储指令非对齐访问异常、存储指令访问错误异常

指令格式：

15 13	12 7	6 2	1 0
110	uimm6[3:0|5:4]	rs2	10

14.3.26　C.SUB——有符号减法指令

语法：

c.sub rd, rs2

操作：

rd ← rs1 - rs2

执行权限：

M mode/U mode

异常：

无

说明：

rs1 == rd

rs1/rd 编码代表寄存器如下:

- 000: x8
- 001: x9
- 010: x10
- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

指令格式：

15 13	12	11 10	9 7	6 5	4 2	1 0
100	0	11	rs1/rd	00	rs2	01

14.3.27　C.XOR——按位异或指令

语法：

c.xor rd, rs2

操作：

rd ← rs1 ^ rs2

执行权限：

M mode/U mode

异常：

无

说明：

rs1 == rd

rs1/rd 编码代表寄存器如下:

- 000: x8
- 001: x9
- 010: x10
- 011: x11
- 100: x12
- 101: x13
- 110: x14
- 111: x15

指令格式：

15 13	12	11 10	9 7	6 5	4 2	1 0
100	0	11	rs1/rd	01	rs2	01

14.4　附录 A-4 伪指令列表

RISC-V 实现了一系列的伪指令，在此列出仅供参考，按英文字母顺序排列。

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

下页继续

表 14.1 – 续上页

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 6 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 6 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 6 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
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

第十五章　附录 B 平头哥扩展指令术语

除了标准中定义的 RV32EMC 指令集外，E902 扩展实现了自定义的 Cache 指令集，以下对每条指令做具体描述。

当 MXSTATUS.theadisaee 为 0 时，执行本节所有指令将产生非法指令异常。

15.1　附录 B-1 Cache 指令术语

Cache 指令集实现了对 cache 的操作，每条指令位宽为 32 位，以下指令按英文字母顺序排列。

15.1.1　ICACHE.IALL——ICACHE 无效全部表项指令

语法：

icache.iall

操作：

无效所有 icache 表项

执行权限：

M mode

异常：

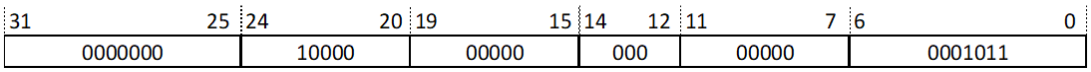
非法指令异常

说明：

mxstatus.theadisaee=0，执行该指令产生非法指令异常。

mxstatus.theadisaee=1，U mode 下执行该指令产生非法指令异常。

指令格式：



15.1.2　ICACHE.IPA——ICACHE 无效物理地址匹配表项指令

语法：

icache.ipa rs1

操作：

将 rs1 中物理地址所对应的 icache 表项无效

执行权限：

M mode

异常：

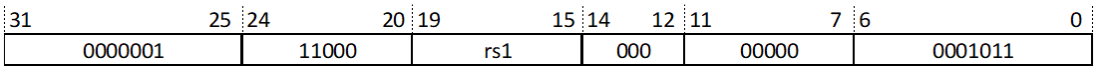
非法指令异常

说明：

mxstatus.theadisaee=0，执行该指令产生非法指令异常。

mxstatus.theadisaee=1，U mode 下执行该指令产生非法指令异常。

指令格式：



31 25	24 20	19 15	14 12	11 7	6 0
0000001	11000	rs1	000	00000	0001011

第十六章　附录 C 机器模式控制寄存器

机器模式控制寄存器按照功能分为：机器模式信息寄存器组、机器模式异常配置寄存器组、机器模式异常处理寄存器组、机器模式内存保护寄存器组、机器模式计数器寄存器组、机器模式扩展寄存器组。

16.1　机器模式信息寄存器组

16.1.1　厂商编号寄存器（MVENDORID）

机器模式厂商编号寄存器（MVENDORID）存储了平头哥半导体有限公司的厂商编号信息，该 ID 由 JEDEC 管理分配，目前 E902 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。

16.1.2　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，E902 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。

16.1.3　微体系架构编号寄存器（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。E902 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。

16.1.4　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。E902 内目前尚未定义，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。

16.2　机器模式异常设置寄存器组

16.2.1　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。

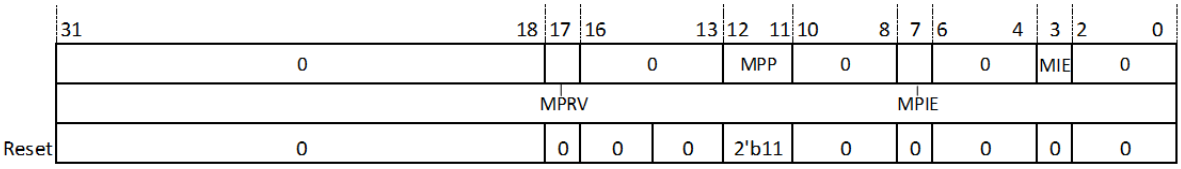


图 16.1: 机器模式处理器状态寄存器（MSTATUS）

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。

当 MPP 为 2’b00 时，表示处理器进入异常服务程序前处于用户模式；

当 MPP 为 2’b11 时，表示处理器进入异常服务程序前处于机器模式；

该域复位值为 2’b11。

MPRV-存储特权位：

当 MPRV 为 0 时，访存地址的转换和保护判断没有特殊要求；

当 MPRV 为 1 时，加载和存储指令（load/store）要根据 MPP 位中存储的特权模式信息进行地址转换和保护判断。

取指的地址转换和保护判断不受该位影响。

该位复位值为零。

在处理器 CLIC 模式下，MPP 位和 MPIE 位可以通过 MCAUSE 寄存器访问得到。

16.2.2　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。

E902 持的指令集架构为 RV32E[M]C，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）

E902 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。

16.2.3　机器模式中断使能控制寄存器（MIE）

机器模式中断使能控制寄存器（MIE）用于控制机器模式下 CLINT 中不同中断类型的局部屏蔽。

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。

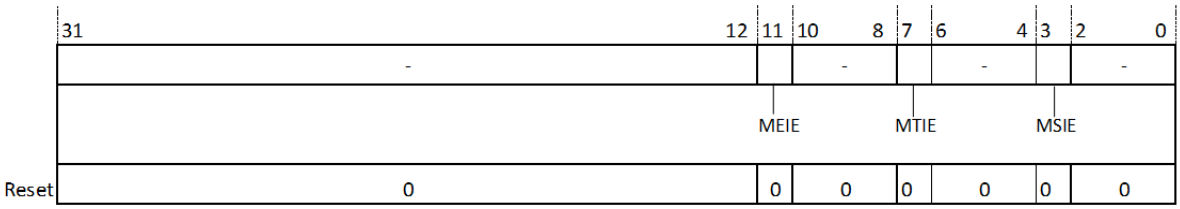


图 16.2: 机器模式中断使能控制寄存器（MIE）

MSIE-机器模式软件中断使能位：

0：机器模式软件中断未使能；
1：机器模式软件中断使能。

MTIE-机器模式计时器中断使能位：

0：机器模式计时器中断未使能；
1：机器模式计时器中断使能。

MEIE-外部中断使能位：

当 MEIE 为 0 时，外部中断无效，处理器在低功耗模式下无法被外部中断唤醒；
当 MEIE 为 1 时，外部中断有效，处理器在低功耗模式下可以被外部中断唤醒；
当处理器配置 CLIC 模式时，且处理器 MTVEC 寄存器配置为 CLIC 模式下（MTVEC.mode[1] 为 1’ b1），该寄存器将会置 0。E902 固定配置 CLIC 模式。

16.2.4　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。

BASE-向量基址位：

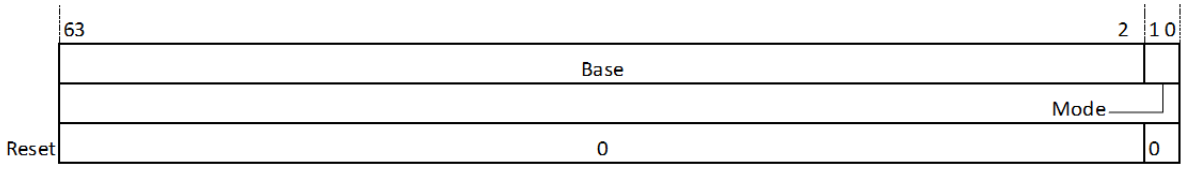


图 16.3: 机器模式异常向量基址寄存器（MTVEC）

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。

E902 中 MODE[1:0] 硬件固定设置为 3，软件不可设置。

16.2.5　机器模式计数器使能寄存器（MCOUNTEREN）

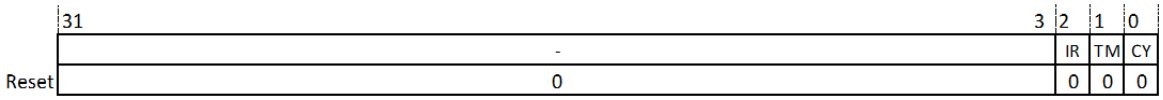


图 16.4: 机器模式计数器使能寄存器（MCOUNTEREN）

CY - MCYCLE 寄存器用户模式访问控制位：

CY 为 0 时用户模式下访问 MCYCLE 寄存器将发生非法指令异常，为 1 时用户模式下能正常访问 MCYCLE 寄存器。

TM - MTIME 寄存器用户模式访问控制位：

TM 为 0 时用户模式下访问计时器当前值寄存器（MTIMEHI 和 MTIMELO）将发生非法指令异常，TM 为 1 时，用户模式下能正常访问计时器当前值寄存器。

IR - MINSTRET 寄存器用户模式访问控制位：

IR 为 0 时用户模式下访问 MINSTRET 寄存器将发生非法指令异常，IR 为 1 时，用户模式下能正常访问 MINSTRET 寄存器。

16.2.6 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 16.5: 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。

16.3 机器模式异常处理寄存器组

16.3.1 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。

16.3.2 机器模式多模式数据备份寄存器（MSCRATCHCSW）

机器模式多模式数据备份寄存器（MSCRATCHCSW）用于加速多特权模式下的中断处理。一般用法为，在不同的特权模式转换时，支持 MSCRATCH 在满足进入中断之前的特权模式不为机器模式时，与栈指针交换数值，否则值保持不变。该指令语法如下：

csrrw rd, mscratchcsw, rs1。

当处理器进入中断之前的特权模式不是机器模式（mcause.mpp!=M-mode）时，执行：

t = rs1; rd = mscratch; mscratch = t;

否则执行：

rd = rs1。

一般用法为：

csrw sp, mscratchcsw, sp。

16.3.3 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。

当处理器进入中断前没有处理中断时（(mcause.pil == 0) != (minstatus.mil == 0)) 执行：

t = rs1; rd = mscratch; mscratch = t;

否则执行 rd = rs1。

一般用法为：

csrw sp, mscratchcswl, sp。

16.3.4　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，E902 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。

16.3.5　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）。E902 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。

16.3.6　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。

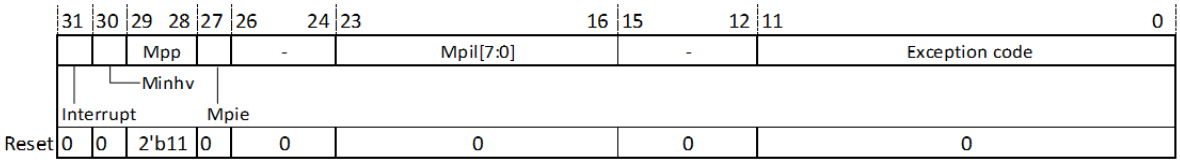


图 16.6: 机器模式异常事件向量寄存器（MCAUSE）

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。

该位为 MSTATUS.MPP[1:0] 的镜像，即读写 MCAUSE.MPP 将会与读写 MSTATUS.MPP 产生相同结果。

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。

该位保存处理器进入中断服务程序前的中断优先级，即将 MINTSTATUS.MIL 位拷贝至该位。执行 MRET 指令从中断返回时，处理器将 MPIL 位拷贝至 MINTSTATUS 寄存器中的 MIL 位。

处理器响应异常时，该位保持不变。

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。

在处理器配置了 CLIC 模式时，该位域扩展为 12 位，支持最多 4096 个中断 ID 号记录。

在处理器没有配置 CLIC 模式时，该位域为 4 位。E902 固定配置 CLIC 模式。

该位复位值为零。

具体的异常向量号列表请参考 表 4.1 。

16.3.7 机器模式等待中断向量地址和中断使能寄存器（MNXTI）

该寄存器在处理器配置了 CLIC 模式时存在。

软件使用该寄存器加速中断响应。在机器模式下，当前处于等待状态中断的优先级高于保留中断优先级（MCAUSE.mpil）时，软件通过读该寄存器可以获取下一中断入口地址。

通过 CSRRSI、CSRRCI 指令操作 MNXTI 寄存器，处理器可以获取下一中断向量表地址，并同时将值写入 MSTATUS 寄存器：

处理器可以通过写 MNXTI 寄存器的 bit[3] 为 1 来实现设置 MSTATUS 寄存器的 MIE 位，进而使能中断。

当读 MNXTI 寄存器返回非零值时，CLIC 标准将其定义为等待处理的有效中断的入口地址，即 MTVT[31:6]<<6 + 4* 中断 ID，同时处理器硬件更新中断现场，如下所述：

MINTSTATUS.mil 位设置为该等待处理的有效中断的中断优先级；

MCAUSE.exception_code 将会设置为该等待处理的有效中断的 ID。

读 MNXTI 寄存器返回非零值，获取有效中断的条件如下：

1. 处理器处于机器模式；
2. 当前等待中断优先级高于保留中断优先级（MCAUSE.mpil）；
3. 该中断不是硬件矢量中断。

当读 MNXTI 寄存器为零时，表征没有有效等待中断需要处理，包括如下情况之一：

1. 确实 CLIC 内没有等待被处理的中断；
2. 有等待被处理的中断优先级大于 MCAUSE.mpil，但该中断是硬件矢量模式。

当读 MNXTI 寄存器返回零时，不会对 MINTSTATUS.mil 域和 MCAUSE.exception_code 域进行更新。

16.3.8　机器模式中断状态寄存器（MINTSTATUS）

	31 24	23 0
	Mil[7:0]	-
Reset	0	0

图 16.7: 机器模式中断状态寄存器（MINTSTATUS）

该寄存器在处理器配置 CLIC 模式时存在。软件只读。

MIL-当前中断优先级位

该域为处理器当前处理的中断的优先级。

当处理器响应中断时，该域被更新为当前被响应中断的优先级。当处理器执行 MRET 指令从中断服务程序返回时，处理器将 MCAUSE.mpil 拷贝至该位。

当处理器响应异常时，该位不发生改变。

该位复位值为零。

16.3.9　机器模式异常原因寄存器（MTVAL）

机器模式异常原因寄存器（MTVAL）用于保存触发异常事件的具体原因，比如访问错误异常的错误访问地址等。

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。

16.3.10　机器模式中断等待寄存器（MIP）

机器模式中断等待寄存器（MIP）用于保存处理器的中断等待状态，该寄存器仅在非 CLIC 模式下有意义。当中断处于等待状态时，MIP 寄存器中的对应位会被置位。

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。

E902 实现了 CLIC 模式，因此该寄存器值为零，软件不可写。

16.4　机器模式内存保护寄存器组

机器模式内存保护寄存器组是和内存保护单元设置相关的控制寄存器，在 CPU 配置内存保护单元时有效。

机器模式物理内存保护配置寄存器（PMPCFG）

机器模式物理内存保护配置寄存器（PMPCFG）用于配置物理内存的访问权限、地址匹配模式。

该寄存器的位宽是 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。

具体的控制寄存器定义请参考*PMP 控制寄存器* 。

机器模式物理内存地址寄存器（PMPADDR）

机器模式物理内存地址寄存器（PMPADDR）用于配置物理内存的每个表项的地址区间。

该寄存器的位宽是 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。

具体的控制寄存器定义请参考*PMP 控制寄存器* 。

16.5 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。

16.5.1 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。

周期计数器为 64 位宽，在 RV32 位架构下，分为 MCYCLEH 和 MCYCLE 两个控制寄存器实现。这两个寄存器分别保存周期计数器的高 32 位和低 32 位数据。

MCYCLEH 和 MCYCLE 两个寄存器的位宽都是 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。

周期计数器复位值为零。

16.5.2 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。

退休指令计数器为 64 位宽，在 RV32 位架构下，分为 MINSTRETH 和 MINSTRET 两个控制寄存器实现。这两个寄存器分别保存退休指令计数器的高 32 位和低 32 位数据。

MINSTRETH 和 MINSTRET 两个寄存器的位宽都是 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。

退休指令计数器复位值为零。

16.6 机器模式扩展寄存器组

16.6.1 扩展状态寄存器（MXSTATUS）

THEADISAEE- THEAD 扩展指令集使能位

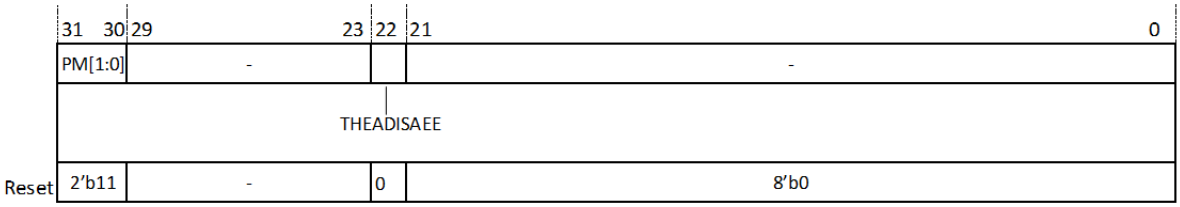


图 16.8: 扩展状态寄存器（MXSTATUS）

当 THEADISAEE 为 0 时，执行所有 T-Head 自定义扩展指令产 * 生非法指令异常；当 THEADISAEE 为 1 时，处理器可以正常执行 T-Head 自定义扩展指令集。
该域复位值为零。

PM-当前中断特权模式位:

表征当前处理器的所处的特权模式，该域为 2’b11 时处理器为机器模式，2’b00 时为用户模式。

16.6.2　硬件配置寄存器（MHCR）

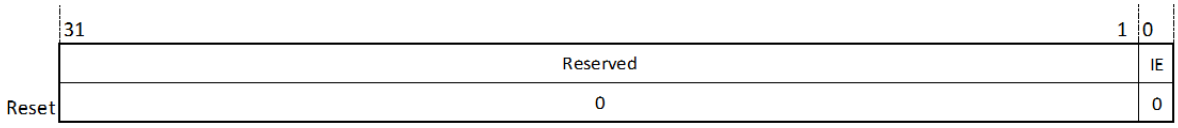


图 16.9: 硬件配置寄存器（MHCR）

IE-指令高速缓存设置位:

当 IE 为 0 时，指令 cache 关闭；当 IE 为 1 时，指令 cache 开启。

16.6.3　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。用户模式访问该寄存器会触发非法指令异常。

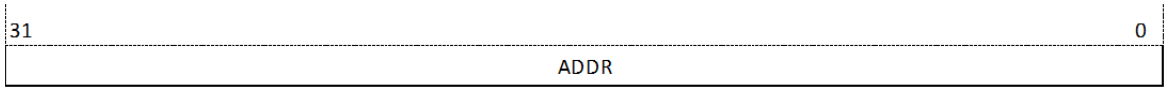


图 16.10: 处理器复位启动地址寄存器（MRADDR）

ADDR-复位启动地址

该域的低两位值为常 0。

16.6.4　扩展异常状态寄存器（MEXSTATUS）

SOFT_RST-软复位指示

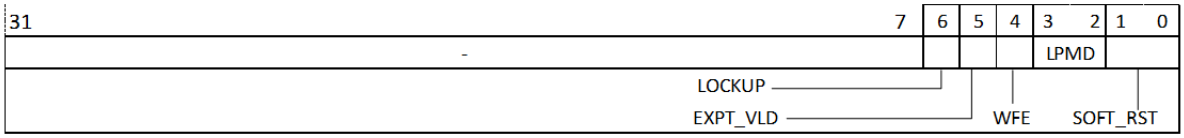


图 16.11: 扩展异常状态寄存器（MEXSTATUS）

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，E902 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 时要求中断优先级大于 MINTSTATUS.MIL 才可以唤醒 CPU；

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。

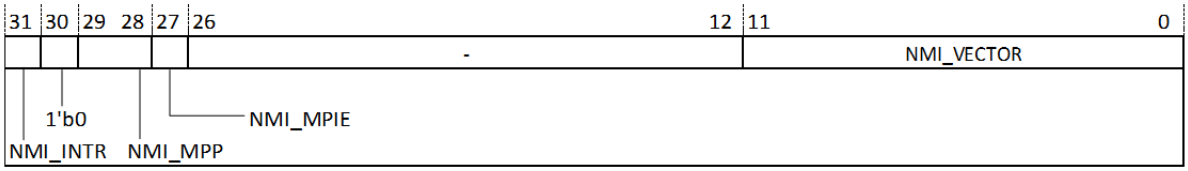


图 16.12: NMI 状态寄存器 (MNMICAUSE)

16.6.5 NMI 状态寄存器 (MNMICAUSE)

NMI_VECTOR:

保存 NMI 响应时 MCAUSE 内 exception code 寄存器的值

NMI_MPIE:

保存 NMI 响应时 MSTATUS 内 MPIE 寄存器的值

NMI_MPP:

保存 NMI 响应时 MSTATUS 内 MPP 寄存器的值

NMI_INTR:

保存 NMI 响应时 MCAUSE 内 INTR 寄存器的值

16.6.6 NMI 异常程序计数器（MNMIPC）

该寄存器位宽为 32 位，仅在机器模式下可读写，用户模式下访问该寄存器会触发非法指令异常。

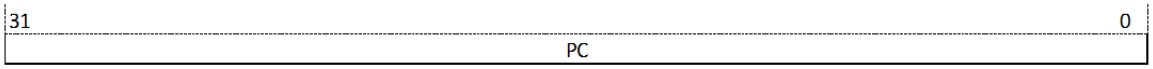


图 16.13: NMI 异常程序计数器（MNMIPC）

PC-异常程序计数器

该域最低 bite 值为常 0。

16.6.7 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，具体定义遵循《平头哥产品 ID 定义规范（5.0 版）》。