玄铁 E907 R1S1 用户手册

2021 年 05 月 17 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
Draft01	第一版临时版发布。	2021.02.06
01	正式发布，调整文档大纲。	2021.04.09
02	修正部分描述错误。	2021.04.21
03	更新模板。	2021.05.14
04	升级版本号。	2021.05.17

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

术语	描述
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。
标识符	以addr这一标识符为例，后面接不同的数字表明不同含义： addr[15:0]表示一组信号，位宽为16位； addr表示单独的一个信号； addr15表示一组信号中的第16个。
LSB	最低有效位。
MSB	最高有效位。
RAW	写后读。
WAW	写后写。

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　概述

1.1.1　简介

E907 是一款基于 RISC-V 指令集的高性能嵌入式微处理处理器，是平头哥 RISC-V MCU 产品线中的最高性能处理器。E907 主要面向语音、MPU、导航、WiFi 等应用领域。

1.1.2　特点

E907 处理器体系结构的主要特点如下：

- 32 位 RISC 处理器；
- 支持 RISC-V RV32IMA[F][D]C[P] 指令集；
- 支持 RISC-V 32/16 位混编指令集；
- 支持 RISC-V 机器模式和用户模式；
- 32 个 32 位整型通用寄存器，32 个 32 位/64 位浮点通用寄存器；
- 整型 5 级/浮点 7 级，单发射，顺序执行流水线；
- 支持 AXI4.0 主设备接口以及 AHB5.0 外设接口；
- 指令 cache，两路组相连结构，2KB-32KB 可配置；
- 数据 cache，两路组相连结构，2KB-32KB 可配置；
- 支持非对齐内存访问；
- 双周期硬件乘法器，基 4 硬件除法器；
- 可选配 BHT 和 BTB；
- 支持平头哥扩展增强指令集；
- 支持平头哥 MCU 特性扩展技术，包括中断处理加速技术、MCU 扩展特性；
- 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；
- 支持可配的性能监测单元；
- 支持 RISC-V Debug 协议标准；
- 频率 >1.0GHz@T28 HPCPlus, 9T SVT(worst case), Coremark > 3.8 coremark/MHz, Dhrystone > 2.0DMIPS/MHz。

1.1.3 可配置选项

E907 可配置选项如 表 1.1 所示。

表 1.1: E907 可配置选项

可配置单元	配置选项	详细
浮点单元	无/单/单 + 双精度浮点	可配置不带硬件浮点单元，或者带硬件浮点单元，硬件浮点单元可进一步配置为只有单精度浮点或者包含单精度和双精度浮点。
DSP 单元	无/有	可选配 RISCV P 指令集扩展，即标量 DSP 单元。
L1 ICache	2/4/8/16/32KB	可以配置 2KB、4KB、8KB、16KB、32KB。典型配置为 32KB。
L1 DCache	2/4/8/16/32KB	可以配置 2KB、4KB、8KB、16KB、32KB。典型配置为 32KB。
AXI Perf & Area	High-Perf/Low-Area	当配置高性能时 AXI 读 Outstanding 为 6，当配置低成本时 AXI 读 Outstanding 为 2。
SYSMAP 软件配置	无/有	选配该组件时可在 E907 硬件集成时固定内存空间属性基础上，通过软件配置寄存器来指定内存空间属性。
PMP 表项数	0/4/8/12/16	PMP 可以不配或者表项数 4/8/12/16 可配。典型配置为 8 个表项。
BHT	0/2/4/8/16Kb	分支历史信息表，可以不配或者配置 2Kb、4Kb、8Kb、16Kb，典型配置为为 8Kb。
BTB	无/有	分支跳转目标缓存表，选配。
中断数量	16-240	CLIC 可配置接入的中断源数量。
中断优先级位宽	2-5	CLIC 可配置接入的中断源数量。
CLIC	Freq/Latency	当配置为 Freq 时 CLIC 固定工作在 CPU CLK 的二分频时钟，当配置为 Latency 时 CLIC 模块时钟与 CPU CLK 时钟同频。
性能监测单元	无/有	E907 可选配硬件监测用于统计如 CACHE 缺失率等用于性能分析。
EMA 接口	无/1-32 比特位宽	可以为核内的 Memory 选配 EMA 接口用来进行额外的访问控制，该组件会在 E907 顶层新增输入信号直通到核内每块 Memory 接口。
调试资源配置	最小配置/典型配置/ 最大配置	E907 实现了 RISC-V Debug 标准兼容的调试单元，硬件可选配最小/典型/最大三种配置，具体信息可查看调试章节。

1.1.4 E907 扩展技术

面向 MCU 类处理器 E907 在 RISC-V 标准上扩展实现了玄铁增强 ISA 和编程模型，方便用户更好的使用玄铁 E907：

- 基于对 Benchmark 的统计分析，玄铁 E907 扩展实现了位操作指令，算术运算指令，内存访问增强指令以及双精度浮点数据传输指令，可以对程序性能进行增强；

- 玄铁 E907 扩展实现了 CACHE 操作指令，同步指令等方便软件人员编程；
- 玄铁 E907 面向实时性的需求扩展实现了中断投机压栈技术和矢量中断咬尾技术，加速中断响应，中断响应延时为 20 个处理器时钟周期；
- 玄铁 E907 同时扩展实现了如 NMI，深浅睡眠低功耗模式，低功耗唤醒事件，锁定等 MCU 领域常见的应用需求
- 玄铁 E907 支持非对齐内存访问，并可软件开关，方便软件人员编程和调试；

1.1.5 可调试性设计

E907 使用 RISC-V Debug 协议，采用标准 5 线 JTAG 硬件调试接口。E907 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，多核异构调试等。具体请详见调试接口 章节。

1.1.6 版本说明

E907 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*
- *RISC-V External Debug Support Version 0.13.2*
- *RISC-V "P" Extension Proposal Version 0.9*

1.2 处理器简介

1.2.1 结构框图

E907 结构框图如 图 1.1 所示。

E907 处理器采用 5 级流水线结构：取指、译码、执行、内存访问、写回。

取指阶段，访问指令 Cache 或者外部总线，获取指令，同时访问 BTB，发起 0 延时跳转。

译码阶段，访问动态分支预测器和返回栈，发起分支的预测跳转，同时进行指令译码，读取寄存器堆，处理数据相关性和数据前馈。

执行阶段，完成单周期整型计算指令和多周期乘除法指令的执行、存储/加载指令地址计算和跳转指令处理。其中，整型计算包括普通的算术指令和逻辑指令。

内存访问阶段，利用执行阶段产生的存储/载入指令的目标地址访问数据 Cache 或者外部总线。

写回阶段，将指令执行结果写回寄存器堆。

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。

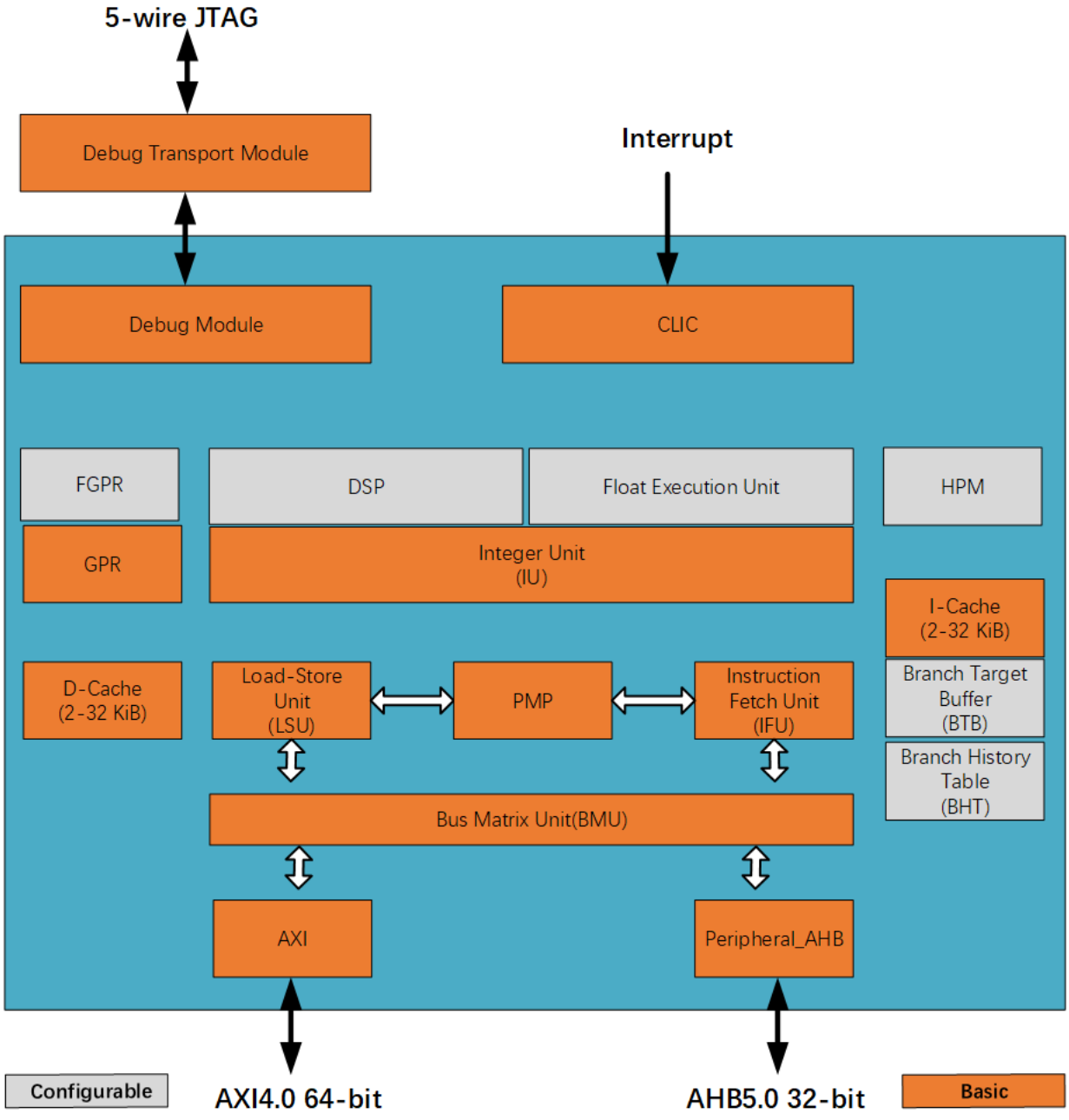


图 1.1: E907 结构图

调试单元（Debug Module，以下简称 DM）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。

E907 设计有片上紧耦合的 IP 接口和两条主设备总线接口。片上紧耦合的 IP 接口集成矢量中断控制器（CLIC），支持中断嵌套。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。

1.2.2 流水线介绍

本节介绍 E907 的指令流水线和指令时序信息。

E907 处理器有 5 级整型流水线：即指令提取、指令译码、指令执行、内存访问和写回。5 级流水线的作用如 表 1.2 所示。

表 1.2: 整型流水线描述

流水线名称	缩写	流水线作用
指令提取	IF	1. 访问指令 Cache，总线； 2. 分支跳转地址预测。
指令译码	ID	1. 指令译码； 2. 分支跳转预测； 3. 寄存器堆访问； 4. 数据冲突检测和处理。
指令执行	EX	1. 算术、逻辑指令执行； 2. 跳转指令处理； 3. 乘法指令执行； 4. Load/Store 指令地址产生。
内存访问	MEM	1. 访问数据 Cache、总线； 2. 除法指令执行； 3. 指令退休。
写回	WB	1. 指令执行结果回写。

当配置硬件浮点单元时，因为浮点指令的执行延时需要 3-4 周期，执行级会变成 3-4 级，因此浮点流水线为 7 级流水。

普通浮点指令需要 3-4 周期的执行延时，而浮点除法类似于整型除法，为长周期指令。为减少对流水线的堵塞，浮点指令也采用乱序回写机制。当指令没有写后写（WAW）数据冲突时，指令可以发射到执行单元，不需要等待前序指令

回写完成。当检测到写后读（RAW）或者写后写（WAW）冲突时，阻塞指令发射。浮点除法指令需要多个周期完成，后续紧挨的没有 RAW 冲突的浮点加法可以先于除法指令回写，对于存在 RAW 冲突的浮点指令需要等待浮点除法指令完成才能发射到执行单元。

1.2.3 紧耦合 IP 架构

为了提高 E907 的系统集成度，方便用户集成与开发，E907 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 16-240 个外部中断源；
- 支持硬件中断嵌套；
- 支持电平中断和沿中断；
- 中断优先级位宽 2-5 比特任意可配；
- 支持矢量中断咬尾。

第二章　指令集与编程模型

2.1　指令集

E907 采用了 16/32 位混合编码的 RV32IMA[F][D]C[P] 指令集，并在此基础上扩展了平头哥自定义指令。E907 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。

2.1.1　RV32IMAFDCP 指令

本章主要介绍了 E907 中实现的 RV32IMA[F][D]C[P] 指令集，包括标准的整型指令集（RV32I），RV 乘除法指令集（RV32M），RV 原子指令集 (RV32A)，RV 单精度浮点指令集 (32F)，RV 双精度浮点指令集 (32D)，和 RV 压缩指令（RVC）。

2.1.1.1　RV32I 整型指令集

本节介绍了 32 位基本整型指令。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 2.1: 整型指令（RV32I）指令列表

指令名称	指令描述	执行延时
加减法指令		
ADD	有符号加法指令	1
ADDI	有符号立即数加法指令	1
SUB	有符号减法指令	1
逻辑操作指令		
AND	按位与指令	1
ANDI	立即数按位与指令	1
OR	按位或指令	1
ORI	立即数按位或指令	1
XOR	按位异或指令	1
XORI	立即数按位异或指令	1
移位指令		
SLL	逻辑左移指令	1
SLLI	立即数逻辑左移指令	1
SRL	逻辑右移指令	1
SRLI	立即数逻辑右移指令	1
SRA	算术右移指令	1
SRAI	立即数算术右移指令	1
比较指令		
SLT	有符号比较小于置位指令	1
SLTU	无符号比较小于置位指令	1
SLTI	有符号立即数比较小于置位指令	1
SLTIU	无符号立即数比较小于置位指令	1
数据传输指令		
LUI	高位立即数装载指令	1
AUIPC	PC 高位立即数加法指令	1
分支跳转指令		
BEQ	相等分支指令	1
BNE	不等分支指令	1
BLT	有符号小于分支指令	1
BGE	有符号大于等于分支指令	1
BLTU	无符号小于分支指令	1
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	

下页继续

表 2.1 – 续上页

SB	字节存储指令	
SH	半字存储指令	
SW	字存储指令	
控制寄存器操作指令		
CSRRW	控制寄存器读写传送指令	阻塞执行
CSRRS	控制寄存器置位传送指令	
CSRRC	控制寄存器清零传送指令	
CSRRWI	控制寄存器立即数读写传送指令	
CSRRSI	控制寄存器立即数置位传送指令	
CSRRCI	控制寄存器立即数清零传送指令	
低功耗指令		
WFI	进入低功耗模式指令	不可预期
异常返回指令		
MRET	机器模式异常返回指令	阻塞执行
特殊功能指令		
FENCE	存储同步指令	不可预期
FENCE.I	指令流同步指令	阻塞执行
ECALL	环境异常调用指令	1
EBREAK	断点指令	1

2.1.1.2　RV32M 乘除法指令集

RV32M 乘除法指令如 表 2.2 所示。

表 2.2: 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	2
MULH	有符号乘法取高位指令	2
MULHSU	有符号与无符号乘法取高位指令	2
MULHU	无符号乘法取高位指令	2
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

2.1.1.3　RV32A 原子指令集

RV32A 原子指令如 表 2.3 所示。

表 2.3: RV32A 原子指令集指令列表

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

2.1.1.4　RV32F 单精度浮点指令

本节主要介绍 32 位单精度浮点指令，按功能可以分为以下类型：

- 运算指令
- 内存存取指令
- 类型转换及数据传输指令
- 比较指令
- 分类指令

表 2.4: RV32F 单精度浮点指令集指令列表

指令名称	指令描述	执行延时
运算指令		
FADD.S	单精度浮点加法指令	3
FSUB.S	单精度浮点减法指令	3
FMADD.S	单精度浮点乘累加指令	4
FMSUB.S	单精度浮点乘累减指令	4
FNMSUB.S	单精度浮点乘累减取反指令	4
FNMADD.S	单精度浮点乘累加取反指令	4
FMUL.S	单精度浮点乘法指令	3
FDIV.S	单精度浮点除法指令	18
FSQRT.S	单精度浮点开方指令	18
FMIN.S	单精度浮点取最小值指令	3
FMAX.S	单精度浮点取最大值指令	3
内存存取指令		

下页继续

表 2.4 – 续上页

指令名称	指令描述	执行延时
FLW	单精度浮点内存加载指令	2（cache 命中）
FSW	单精度浮点内存存储指令	2（cache 命中）
类型转换及数据传输指令		
FSGNJ.S	单精度浮点符号注入指令	1
FSGNJN.S	单精度浮点符号取反注入指令	1
FSGNJX.S	单精度浮点符号异或注入指令	1
FCVT.W.S	单精度浮点转换为有符号整型指令	3
FCVT.WU.S	单精度浮点转换为无符号整型指令	3
FCVT.S.W	有符号整型转换为单精度浮点指令	3
FCVT.S.WU	无符号整型转换为单精度浮点指令	3
FMV.X.W	单精度浮点读传送指令	1
FMV.W.X	单精度浮点写传送指令	1
比较指令		
FEQ.S	单精度相等比较指令	3
FLT.S	单精度浮点小于比较指令	3
FLE.S	单精度浮点小于等于比较指令	3
分类指令		
FCLASS.S	单精度数值类型分类指令	1

2.1.1.5　RV32D 双精度浮点指令

本节主要介绍 32 位双精度浮点指令，按功能可以分为以下类型：

- 运算指令
- 内存存取指令
- 类型转换及数据传输指令
- 比较指令
- 分类指令

表 2.5: RV32D 双精度浮点指令集指令列表

指令名称	指令描述	执行延时
运算指令		
FADD.D	单精度浮点加法指令	3
FSUB.D	双精度浮点减法指令	3
FMADD.D	双精度浮点乘累加指令	4
FMSUB.D	双精度浮点乘累减指令	4
FNMSUB.D	双精度浮点乘累减取反指令	4
FNMADD.D	双精度浮点乘累加取反指令	4
FMUL.D	双精度浮点乘法指令	3
FDIV.D	双精度浮点除法指令	32
FSQRT.D	双精度浮点开方指令	32
FMIN.D	双精度浮点取最小值指令	3
FMAX.D	双精度浮点取最大值指令	3
内存存取指令		
FLD	双精度浮点内存加载指令	2（cache 命中）
FSD	双精度浮点内存存储指令	2（cache 命中）
类型转换及数据传输指令		
FSGNJ.D	双精度浮点符号注入指令	1
FSGNJN.D	双精度浮点符号取反注入指令	1
FSGNJX.D	双精度浮点符号异或注入指令	1
FCVT.W.D	双精度浮点转换为有符号整型指令	3
FCVT.WU.D	双精度浮点转换为无符号整型指令	3
FCVT.D.W	有符号整型转换为双精度浮点指令	3
FCVT.D.WU	无符号整型转换为双精度浮点指令	3
比较指令		
FEQ.D	双精度相等比较指令	3
FLT.D	双精度浮点小于比较指令	3
FLE.D	双精度浮点小于等于比较指令	3
分类指令		
FCLASS.D	双精度数值类型分类指令	1

2.1.1.6　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 2.6: 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(cache 命中)
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

2.1.1.7　RV32P DSP 指令集

本节主要介绍 DSP 指令。P 指令集按功能可以分为以下类型:

- SIMD 指令
- 部分 SIMD 指令

- 64 位运算指令
- 非 SIMD 运算指令

表 2.7: RV32P DSP 指令集指令列表

指令名称	指令描述	执行延时
SIMD 指令		
16 位加减法指令		
add16	16 位加法指令	1
radd16	16 位有符号加法减半指令	1
uradd16	16 位无符号加法减半指令	1
kadd16	16 位有符号加法饱和指令	1
ukadd16	16 位无符号加法饱和指令	1
sub16	16 位减法指令	1
rsub16	16 位有符号减法减半指令	1
ursub16	16 位无符号减法减半指令	1
ksub16	16 位有符号减法饱和指令	1
uksub16	16 位无符号减法饱和指令	1
cras16	16 位交叉加减指令	1
rcras16	16 位有符号交叉加减减半指令	1
urcras16	16 位无符号交叉加减减半指令	1
kcras16	16 位有符号交叉加减饱和指令	1
ukcras16	16 位无符号交叉加减饱和指令	1
crsa16	16 位交叉减加指令	1
rcrsa16	16 位有符号交叉减加减半指令	1
urcrsa16	16 位无符号交叉减加减半指令	1
kcrsa16	16 位有符号交叉减加饱和指令	1
ukcrsa16	16 位无符号交叉减加饱和指令	1
stas16	16 位对应半字加减指令	1
rstas16	16 位有符号对应半字加减减半指令	1
urstas16	16 位无符号对应半字加减减半指令	1
kstas16	16 位有符号对应半字加减饱和指令	1
ukstas16	16 位无符号对应半字加减饱和指令	1
stsa16	16 位对应半字减加指令	1
rstsa16	16 位有符号对应半字减加减半指令	1
urstsa16	16 位无符号对应半字减加减半指令	1
kstsa16	16 位有符号对应半字减加饱和指令	1
ukstsa16	16 位无符号对应半字减加饱和指令	1
8 位加减法指令		
add8	8 位加法指令	1
radd8	8 位有符号加法减半指令	1
uradd8	8 位无符号加法减半指令	1
kadd8	8 位有符号加法饱和指令	1
ukadd8	8 位无符号加法饱和指令	1

下页继续

表 2.7 – 续上页

指令名称	指令描述	执行延时
sub8	8 位减法指令	1
rsub8	8 位有符号减法减半指令	1
ursub8	8 位无符号减法减半指令	1
ksub8	8 位有符号减法饱和指令	1
uksub8	8 位无符号减法饱和指令	1
16 位移位指令		
sra16	16 位寄存器算术右移指令	1
srai16	16 位立即数算术右移指令	1
sra16.u	16 位寄存器算术右移舍入指令	2
srai16.u	16 位立即数算术右移舍入指令	2
srl16	16 位寄存器逻辑右移指令	1
srli16	16 位立即数逻辑右移指令	1
srl16.u	16 位寄存器逻辑右移舍入指令	2
srli16.u	16 位立即数逻辑右移舍入指令	2
sll16	16 位寄存器逻辑左移指令	1
slli16	16 位立即数逻辑左移指令	1
ksll16	16 位寄存器逻辑左移饱和指令	1
kslli16	16 位立即数逻辑左移饱和指令	1
kslra16	16 位寄存器逻辑左移饱和或算术右移指令	1
kslra16.u	16 位寄存器逻辑左移饱和或算术右移舍入指令	2
8 位移位指令		
sra8	8 位寄存器算术右移指令	1
srai8	8 位立即数算术右移指令	1
sra8.u	8 位寄存器算术右移舍入指令	2
srai8.u	8 位立即数算术右移舍入指令	2
srl8	8 位寄存器逻辑右移指令	1
srli8	8 位立即数逻辑右移指令	1
srl8.u	8 位寄存器逻辑右移舍入指令	2
srli8.u	8 位立即数逻辑右移舍入指令	2
sll8	8 位寄存器逻辑左移指令	1
slli8	8 位立即数逻辑左移指令	1
ksll8	8 位寄存器逻辑左移饱和指令	1
kslli8	8 位立即数逻辑左移饱和指令	1
kslra8	8 位寄存器逻辑左移饱和或算术右移指令	1
kslra8.u	8 位寄存器逻辑左移饱和或算术右移舍入指令	2
16 位比较指令		
cmpeq16	16 位相等比较指令	1
scmplt16	16 位有符号小于比较指令	1
scmple16	16 位有符号小于等于比较指令	1
ucmplt16	16 位无符号小于比较指令	1
ucmple16	16 位无符号小于等于比较指令	1

下页继续

表 2.7 – 续上页

指令名称	指令描述	执行延时
8 位比较指令		
cmpeq8	8 位相等比较指令	1
scmplt8	8 位有符号小于比较指令	1
scmple8	8 位有符号小于等于比较指令	1
ucmplt8	8 位无符号小于比较指令	1
ucmple8	8 位无符号小于等于比较指令	1
16 位乘法指令		
smul16	16 位有符号乘法指令	2
smulx16	16 位有符号交叉乘法指令	2
umul16	16 位无符号乘法指令	2
umulx16	16 位无符号交叉乘法指令	2
khm16	Q15 有符号饱和乘法指令	2
khmx16	Q 15 有符号交叉饱和乘法指令	2
8 位乘法指令		
smul8	8 位有符号乘法指令	2
smulx8	8 位有符号交叉乘法指令	2
umul8	8 位无符号乘法指令	2
umulx8	8 位无符号交叉乘法指令	2
khm8	Q7 有符号饱和乘法指令	2
khmx8	Q7 有符号交叉饱和乘法指令	2
16 位杂类指令		
smin16	16 位有符号求最小值指令	1
umin16	16 位无符号求最小值指令	1
smax16	16 位有符号求最大值指令	1
umax16	16 位无符号求最大值指令	1
sclip16	16 位有符号缩减指令	1
uclip16	16 位无符号缩减指令	1
kabs16	16 位求绝对值指令	1
clrs16	16 位求前导符号位位宽指令	1
clz16	16 位求前导零位宽指令	1
clo16	16 位求前导 1 位宽指令	1
swap16	16 位交换指令	1
8 位杂类指令		
smin8	8 位有符号求最小值指令	1
umin8	8 位无符号求最小值指令	1
smax8	8 位有符号求最大值指令	1
umax8	8 位无符号求最大值指令	1
kabs8	8 位求绝对值指令	1
sclip8	8 位有符号缩减指令	1
uclip8	8 位无符号缩减指令	1
clrs8	8 位求前导符号位位宽指令	1

下页继续

表 2.7 – 续上页

指令名称	指令描述	执行延时
clz8	8 位求前导零位宽指令	1
clo8	8 位求前导 1 位宽指令	1
swap8	8 位交换指令	1
8 位拆解指令		
sunpkd810	有符号字节 1，字节 0 拆解指令	1
sunpkd820	有符号字节 2，字节 0 拆解指令	1
sunpkd830	有符号字节 3，字节 0 拆解指令	1
sunpkd831	有符号字节 3，字节 1 拆解指令	1
sunpkd832	有符号字节 3，字节 2 拆解指令	1
zunpkd810	无符号字节 1，字节 0 拆解指令	1
zunpkd820	无符号字节 2，字节 0 拆解指令	1
zunpkd830	无符号字节 3，字节 0 拆解指令	1
zunpkd831	无符号字节 3，字节 1 拆解指令	1
zunpkd832	无符号字节 3，字节 2 拆解指令	1
部分 SIMD 指令		
16 位封装指令		
pkbb16	两个低半字封装指令	1
pkbt16	低半字和高半字封装指令	1
pktb16	高半字和低半字封装指令	1
pktt16	两个高半字封装指令	1
32x32 高位乘累加指令		
smmul	有符号 32x32 取高字指令	2
smmul.u	有符号 32x32 舍入后取高字指令	2
kmmac	有符号 32x32 取高字后饱和加指令	2
kmmac.u	有符号 32x32 舍入后取高字再饱和加指令	2
kmmsb	有符号 32x32 取高字后饱和减指令	2
kmmsb.u	有符号 32x32 舍入后取高字再饱和减指令	2
kwmmul	有符号 3 2x32 倍增后饱和取高字指令	2
kwmmul.u	有符号 32x32 倍增再舍入后饱和取高字指令	2
32x16 高位乘累加指令		
smmwb	有符号 32x 低 16 取高字指令	2
smmwb.u	有符号 32x 低 16 舍入后取高字指令	2
smmwt	有符号 32x 高 16 取高字指令	2
smmwt.u	有符号 32x 高 16 舍入后取高字指令	2
kmmawb	有符号 32x 低 16 取高字饱和加指令	2
kmmawb.u	有符号 32x 低 16 舍入后取高字再饱和加指令	2
kmmawt	有符号 32x 高 16 取高字饱和加指令	2
kmmawt.u	有符号 32x 高 16 舍入后取高字再饱和加指令	2
kmmwb2	有符号 32x 低 16 倍增后取高字指令	2
kmmwb2.u	有符号 32x 低 16 倍增后再舍入取高字指令	2
kmmwt2	有符号 32x 高 16 倍增后取高字指令	2

下页继续

表 2.7 – 续上页

指令名称	指令描述	执行延时
kmmwt2.u	有符号 32x 高 16 倍增后再舍入取高字指令	2
kmmawb2	有符号 32x 低 16 倍增后取高字再饱和加指令	2
kmmawb2.u	有符号 32x 低 16 倍增后再舍入取高字后饱和加指令	2
kmmawt2	有符号 32x 高 16 倍增后取高字再饱和加指令	2
kmmawt2.u	有符号 32x 高 16 倍增后再舍入取高字后饱和加指令	2
16x32 位加减指令		
smbb16	有符号低半字乘法指令	2
smbt16	有符号低半字和高半字乘法指令	2
smtt16	有符号高半字乘法指令	2
kmda	有符号半字乘法后相加指令	2
kmxda	有符号半字交叉乘法后相加指令	2
smds	有符号半字乘法后相减指令	2
smdrs	有符号半字乘法后反向相减指令	2
smxds	有符号半字交叉乘法后相减指令	2
kmabb	有符号低半字乘法后加 32 位运算指令	2
kmabt	有符号低半字和高半字乘法后加 32 位运算指令	2
kmatt	有符号高半字乘法后加 32 位运算指令	2
kmada	有符号半字乘法后相加再加 32 位运算指令	3
kmaxda	有符号半字交叉乘法后相加再加 32 位运算指令	3
kmads	有符号半字乘法后相减再加 32 位运算指令	3
kmadrs	有符号半字乘法后反向相减再加 32 位运算指令	3
kmaxds	有符号半字交叉乘法后相减再加 32 位运算指令	3
kmsda	有符号半字乘法后相减再被 32 位减运算指令	3
kmsxda	有符号半字交叉乘法后相减再被 32 位减运算指令	3
16x64 位加减法指令		
smal	16x16 再加 64 位运算指令	3
杂类指令		
sclip32	32 位有符号缩减指令	1
uclip32	32 位无符号缩减指令	1
clrs32	32 位求前导符号位位宽指令	1
clz32	32 位求前导零位宽指令	1
clo32	32 位求前导 1 位宽指令	1
pbsad	字节差分绝对值求和指令	1
pbsada	字节差分绝对值求和再累加指令	1
8x32 位加法指令		
smaqa	8 位有符号乘法累加再加 32 位运算指令	3
umaqa	8 位无符号乘法累加再加 32 位运算指令	3
smaqa.su	8 位有符号和无符号乘法累加再加 32 位运算指令	3
64 位 Profile 指令		
64 位加减法指令		
add64	64 位加法指令	2

下页继续

表 2.7 – 续上页

指令名称	指令描述	执行延时
radd64	64 位有符号加法后减半指令	2
uradd64	64 位无符号加法后减半指令	2
kadd64	64 位有符号饱和加法指令	2
ukadd64	64 位无符号饱和加法指令	2
sub64	64 位减法指令	2
rsub64	64 位有符号减法后减半指令	2
ursub64	64 位无符号减法后减半指令	2
ksub64	64 位有符号饱和减法指令	2
uksub64	64 位无符号饱和减法指令	2
32 位相乘与 64 位加减法指令		
smar64	有符号 32x32 再加 64 位指令	2
smsr64	有符号 32x32 被 64 位减指令	2
umar64	无符号 32x32 再加 64 位指令	2
umsr64	无符号 32x32 被 64 位减指令	2
kmar64	有符号 32x32 再加 64 位后饱和指令	2
kmsr64	有符号 32x32 被 64 位减后饱和指令	2
ukmar64	无符号 32x32 再加 64 位后饱和指令	2
ukmsr64	无符号 32x32 被 64 位减后饱和指令	2
16 位相乘与 64 位加减法指令		
smalbb	有符号低 16x 低 16 再加 64 位指令	3
smalbt	有符号低 16x 高 16 再加 64 位指令	3
smaltt	有符号高 16x 高 16 再加 64 位指令	3
smalda	有符号半字相乘累加后再加 64 位指令	3
smalxda	有符号半字交叉相乘累加后再加 64 位指令	3
smalds	有符号半字相乘累减后再加 64 位指令	3
smaldrs	有符号半字相乘反向累减后再加 64 位指令	3
smalxds	有符号半字交叉相乘累减后再加 64 位指令	3
smslda	有符号半字相乘累减后再被 64 位减指令	3
smslxda	有符号半字交叉相乘累减后再被 64 位减指令	3
非 SIMD 指令		
Q15 饱和指令		
kaddh	低半字相加后饱和指令	2
ksubh	低半字相减后饱和指令	1
khmbb	低半字相乘后饱和到 Q15 指令	2
khmbt	低半字和高半字相乘后再饱和到 Q15 指令	2
khmtt	高半字相乘后饱和到 Q15 指令	2
ukaddh	无符号低半字相加后饱和指令	2
uksubh	无符号低半字相减后饱和指令	1
Q31 饱和指令		
kaddw	32 位相加饱和到 Q31 指令	1
ukaddw	无符号 32 位相加饱和指令	1

下页继续

表 2.7 – 续上页

指令名称	指令描述	执行延时
ksubw	32 位相减饱和到 Q31 指令	1
uksubw	无符号 32 位相减饱和指令	1
kdmbb	低半字相乘倍增后饱和到 Q31 指令	2
kdmbt	低半字和高半字相乘倍增后饱和到 Q31 指令	2
kdmtt	高半字相乘倍增后饱和到 Q31 指令	2
kslraw	寄存器逻辑左移后饱和或算术右移指令	1
kslraw.u	寄存器逻辑左移后饱和或算术右移后舍入指令	2
ksllw	寄存器逻辑左移后饱和指令	1
kslliw	立即数逻辑左移后饱和指令	1
kdmabb	低半字相乘后倍增再和 32 位饱和加之后再饱和指令	2
kdmabt	低半字和高半字相乘后倍增再和 32 位饱和加之后再饱和指令	2
kdmatt	高半字相乘后倍增再和 32 位饱和加之后再饱和指令	2
kabsw	32 位求绝对值后再饱和指令	1
32 位计算指令		
raddw	32 位有符号加后减半指令	1
uraddw	32 位无符号加后减半指令	1
rsubw	32 位有符号减后减半指令	1
ursubw	32 位无符号减后减半指令	1
maxw	32 位有符号求最大值指令	1
minw	32 位有符号求最小值指令	1
mulr64	无符号 32x32 运算指令	2
mulsr64	有符号 32x32 运算指令	2
msubr32	32x 32 取低字再被 32 位累减指令	2
饱和/溢出状态操作指令		
Rdov	读 vxsat.OV 指令	阻塞执行
clrov	清除 vxsat.OV 指令	阻塞执行
杂类指令		
ave	求平均值后舍入指令	1
sra.u	寄存器算术右移后舍入指令	2
srai.u	立即数算术右移后舍入指令	2
bitrev	按寄存器比特位倒序指令	1
bitrevi	按立即数比特位倒序指令	1
wext	按寄存器从 64 位中抽取 32 位指令	1
wexti	按立即数从 64 位中抽取 32 位指令	1
bpick	按寄存器值选择比特封装指令	1
insb	字节插入指令	1
maddr32	3 2x32 取低字再累加 32 位指令	2
msubr32	32x 32 取低字再被 32 位累减指令	2

2.1.2　平头哥扩展指令集

基于对 Cache 操作编程模型的统一以及大量通用基准测试程序性能的统计分析，E907 上扩展实现了性能增强 ISA。本章节所描述指令需要在 MXSTATUS 寄存器中 THEADISAEE 位为 1 方可正常执行，否则执行本章节所述指令会触发非法指令异常。另外，Cache 操作指令仅在机器模式下可正常执行，用户模式下执行同样会触发非法指令异常。

除 Cache 操作指令和同步指令外，其他扩展指令可采用平头哥发布的编译器由高阶语言编译生成。

2.1.2.1　Cache 操作指令

表 2.8: 扩展 Cache 指令列表

指令名称	指令描述	执行延时	备注
Cache 操作指令			
DCACHE.IPA	DCACHE 无效物理地址匹配表项指令	1	-
DCACHE.CPA	DCACHE 清除物理地址匹配表项指令		-
DCACHE.CIPA	DCACHE 清除并无效物理地址匹配表项指令		-
DCACHE.ISW	DCACHE 无效 way/set 指向表项指令	1	-
DCACHE.CSW	DCACHE 清除 way/set 指向表项指令		-
DCACHE.CISW	DCACHE 清除并无效 way/set 指向表项指令		-
DCACHE.IALL	DCACHE 无效全部表项指令	N/2	N：DCACHE 缓存行数量
DCACHE.CALL	DCACHE 清除全部表项指令	N/2*	
DCACHE.CIALL	DCACHE 清除并无效全部表项指令		
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 2.8 列出了 E907 实现的平头哥自定义扩展的 cache 指令子集。

注解： 当表项全部为 Clean 时约为 N/2 个周期，当表项存在 Dirty 时，>N/2 个周期，延时与 Dirty 的表项数目相关。

具体指令说明和定义，请参考标准指令集 。

2.1.2.2　同步指令

表 2.9: 扩展同步指令列表

指令名称	**指令描述**	**备注**
同步指令		
SYNC	同步指令	
SYNC.I	同步清空指令	

具体指令说明和定义，请参考标准指令集 。

2.1.2.3　算术运算指令

表 2.10: 扩展算术运算指令列表

指令名称	指令描述	执行延时
算术运算指令		
ADDSL	有符号移位加法指令	1
SRRI	循环右移指令	1
MULA	乘累加指令	2
MULAH	低 16 位乘累加指令	2
MULS	乘累减指令	2
MULSH	低 16 位乘累减指令	2
MVEQZ	寄存器为零传递指令	1
MVNEZ	寄存器非零传递指令	1

具体指令说明和定义，请参考标准指令集 。

2.1.2.4　位操作指令

表 2.11: 扩展位操作指令列表

指令名称	指令描述	执行延时
位操作指令		
EXT	寄存器连续位提取符号位扩展指令	1
EXTU	寄存器连续位提取无符号扩展指令	1
FF0	快速找 0 指令	1
FF1	快速找 1 指令	1
REV	字节倒序指令	1
TST	比特为零测试指令	1
TSTNBZ	字节为零测试指令	1

具体指令说明和定义，请参考标准指令集 。

2.1.2.5　存储访问指令

表 2.12: 扩展存储访问指令列表

指令名称	指令描述	执行延时
存储访问指令		
LRB	寄存器移位字节加载符号位扩展指令	2（cache 命中）
LRH	寄存器移位半字加载符号位扩展指令	2（cache 命中）
LRW	寄存器移位字加载指令	2（cache 命中）
LRBU	寄存器移位字节加载无符号扩展指令	2（cache 命中）
LRHU	寄存器移位半字加载无符号扩展指令	2（cache 命中）
LBIA	字节加载符号位扩展基地址自增指令	2（cache 命中）
LBIB	基地址自增字节加载符号位扩展指令	2（cache 命中）
LHIA	半字加载符号位扩展基地址自增指令	2（cache 命中）
LHIB	基地址自增半字加载符号位扩展指令	2（cache 命中）
LWIA	字加载基地址自增指令	2（cache 命中）
LWIB	基地址自增字加载指令	2（cache 命中）
LBUIA	字节加载无符号扩展基地址自增指令	2（cache 命中）
LBUIB	基地址自增字节加载无符号扩展指令	2（cache 命中）
LHUIA	半字加载无符号扩展基地址自增指令	2（cache 命中）
LHUIB	基地址自增半字加载无符号扩展指令	2（cache 命中）
SRB	寄存器移位字节存储指令	2（cache 命中）
SRH	寄存器移位半字存储指令	2（cache 命中）
SRW	寄存器移位字存储指令	2（cache 命中）
SBIA	字节存储基地址自增指令	2（cache 命中）
SBIB	基地址自增字节存储指令	2（cache 命中）
SHIA	半字存储基地址自增指令	2（cache 命中）
SHIB	基地址自增半字存储指令	2（cache 命中）
SWIA	字存储基地址自增指令	2（cache 命中）
SWIB	基地址自增字存储指令	2（cache 命中）

具体指令说明和定义，请参考标准指令集 。

2.1.2.6　双精度浮点高位数据传输指令

本节所列指令仅在 E907 硬件配置实现了双精度浮点时定义，在 E907 仅配置单精度浮点时执行本节所描述指令会产生非法指令异常。

表 2.13: 扩展浮点数据传输指令列表

指令名称	指令描述	执行延时
浮点数据传输指令		
FMV.X.HW	双精度浮点高位读传输指令	1
FMV.HW.X	双精度浮点高位写传输指令	1

具体指令说明和定义，请参考标准指令集 。

2.1.2.7　中断加速指令

表 2.14: 扩展中断加速指令列表

指令名称	指令描述	执行延时
浮点数据传输指令		
IPUSH	中断加速压栈指令	19
IPOP	中断加速弹栈指令	20

具体指令说明和定义，请参考标准指令集 。

2.2　编程模型

2.2.1　工作模式

E907 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，处理器就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 的寄存器访问。普通用户模式只允许访问通用寄存器、浮点寄存器、浮点控制寄存器，DSP 运算状态寄存器以及事件监测寄存器等；机器模式可以访问所有通用寄存器、浮点寄存器、控制寄存器和所有紧耦合 IP 的寄存器。

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 E907 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。

2.2.2　寄存器视图

玄铁 E907 兼容 RISC-V 1.10 版本的特权架构 SPEC，同时玄铁 E907 还扩展实现了面向 MCU 应用领域特点的编程模型。玄铁 E907 内部的寄存器实现如 图 2.2 所示。

2.2.2.1　通用寄存器

表 2.15 列出了普通用户编程模式下的 32 个 32 位通用寄存器（X0~X31），其中 X0 是零值寄存器。

图 2.1: 寄存器视图

表 2.15: 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
x12-17	a2-a7	函数参数
x18-27	s2-s11	保留寄存器
x28-31	t3-t6	临时寄存器

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）。

2.2.2.2　浮点寄存器

当 E907 配置了硬件浮点单元时，按照 RISC-V SPEC 定义，E907 内部会额外实现一组 32 个浮点寄存器（f31~f0）。根据浮点配置的不同，当配置单精度浮点时，f31~f0 每个寄存器位宽为 32 比特，当配置双精度浮点时，f31~f0 每个寄存器位宽为 64 比特。按照 RISC-V 标准定义，配置双精度浮点时必须同时实现单精度浮点，因此单精度浮点使用 f31~f0 中寄存器的低 32 位，高 32 位为全 1。

表 2.16: 浮点通用寄存器

寄存器	ABI 名称	描述
f0~f7	ft0-7	浮点临时寄存器
f8~f9	fs0-1	浮点保留寄存器
f10~f11	fa0-1	浮点参数/返回值寄存器
f12~f17	fa2-7	浮点参数寄存器
f18~f27	fs2-11	浮点保留寄存器
f28~f31	ft8-11	浮点临时寄存器

2.2.2.3　机器模式控制寄存器

E907 中实现的 RISC-V 标准定义的机器模式控制寄存器如 表 2.17 所示。

表 2.17: 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			
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
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

下页继续

表 2.17 – 续上页

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
机器模式性能监测寄存器组			
MCOUNTEREN	机器模式读写	0x306	机器模式计数器授权寄存器
MCOUNTINHIBIT	机器模式读写	0x320	机器模式计数禁止寄存器
MHPMEVENT3	机器模式读写	0x323	机器模式 3 号事件选择寄存器
……			
MHPMEVENT17	机器模式读写	0x331	机器模式 17 号事件选择寄存器
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
MHPMCOUNTER3	机器模式读写	0xB03	机器模式 3 号事件计数器
……			
MHPMCOUNTER17	机器模式读写	0xB11	机器模式 17 号事件计数器
MHPMCOUNTER3H	机器模式读写	0xB83	机器模式 3 号事件计数器高 32 位
……			
MHPMCOUNTER17H	机器模式读写	0xB91	机器模式 17 号事件计数器高 32 位

E907 中扩展的控制寄存器如 表 2.18 所示。

表 2.18: E907 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
MHINT	机器模式读写	0x7C5	隐式操作寄存器
MRADDR	机器模式只读	0x7E0	处理器复位启动地址指示寄存器
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，请参考机器模式控制寄存器 。

2.2.2.4　用户模式控制寄存器

用户模式可访问的控制寄存器如 表 2.19 所示。

表 2.19: 用户模式控制寄存器列表

名称	读写权限	寄存器编号	描述
用户模式浮点控制寄存器组			
FFLAGS	用户模式读写	0x001	浮点异常累积状态寄存器
FRM	用户模式读写	0x002	浮点动态舍入模式寄存器
FCSR	用户模式读写	0x003	浮点控制寄存器
用户模式性能监测计数器			
CYCLE	用户模式只读	0xC00	用户模式运行周期数计数器低 32 位
TIME	用户模式只读	0xC01	用户模式计时器低 32 位
INSTRET	用户模式只读	0xC02	用户模式指令退休计数器低 32 位
CYCLEH	用户模式只读	0xC80	用户模式运行周期数计数器高 32 位
TIMEH	用户模式只读	0xC81	用户模式计时器高 32 位
INSTERTH	用户模式只读	0xC82	用户模式指令退休计数器高 32 位
HPMCOUNTER3	用户模式只读	0xC03	用户模式 3 号事件计数器
……			
HPMCOUNTER17	用户模式只读	0xC11	用户模式 17 号事件计数器
HPMCOUNTERH3	用户模式只读	0xC83	用户模式 3 号事件计数器高 32 位
……			
HPMCOUNTERH17	用户模式只读	0xC91	用户模式 17 号事件计数器高 32 位
定点运算溢出/饱和状态寄存器			
VXSAT	用户模式读写	0x009	保存定点运算溢出或饱和标志

表 2.19 所列用户模式性能监测计数器均为对应机器模式性能监测计数器的只读映射。在用户模式下的访问权限依赖 MCOUNTEREN 寄存器的授权。

表 2.20: 用户模式扩展控制寄存器

名称	读写权限	寄存器编号	描述
用户模式浮点扩展控制寄存器组			
FXCR	机器模式只读	0x800	用户模式浮点扩展控制寄存器

2.2.3　数据格式

2.2.3.1　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 2.2 所示。

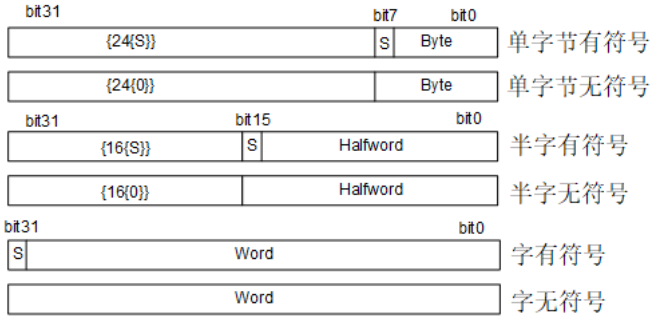


图 2.2: 寄存器中的整型数据组织结构

2.2.3.2　浮点数据格式

E907 浮点单元遵从 RISC-V 标准，兼容 IEEE 754-2008 浮点协议，支持单精度和双精度浮点运算，数据格式如 图 2.3 所示。当 E907 硬件配置了双精度浮点时，单精度数据仅使用 64 位浮点寄存器的低 32 位，高 32 位需要全部为 1，否则会被当做非数处理。当 E907 硬件仅配置单精度浮点时，浮点寄存器位宽为 32 位。

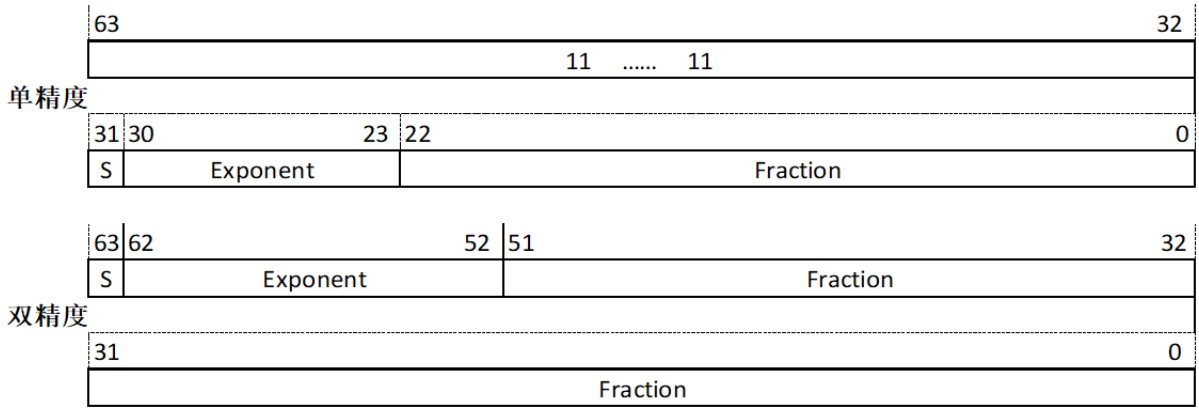


图 2.3: 浮点数据格式

2.2.3.3　大小端

存储器数据有大小端的区分，E907 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 2.4 所示。

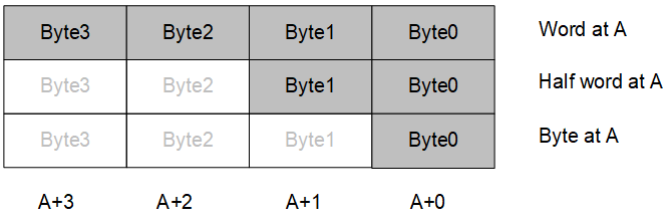


图 2.4: 内存中的数据组织形式

2.3　功耗管理

E907 设计实现了 RV32I 的 WFI 指令用于使处理器从正常工作模式转入低功耗模式，降低功耗水平。在低功耗模式下，E907 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 E907 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。

低功耗模式下，E907 不会向总线发起数据传输请求，内部流水线停顿。

2.3.1　低功耗模式

E907 扩展实现了 MEXSTATUS 寄存器的 LPMD 域来指示通过 WFI 指令进入低功耗模式类型，深睡眠和浅睡眠，并通过 E907 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*）。

E907 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。

除此之外，系统设计人员可以在 SoC 上的功耗管理单元设计更细粒度的功耗管理策略，CPU 通过读写该单元的寄存器来获取或者设置系统低功耗的状态。

2.3.2　低功耗唤醒

E907 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。

E907 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（MEXSTATUS.WFE 为 0 时肯定唤醒中断的优先级高）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。

第三章　异常与中断

3.1　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。E907 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。

E907 兼容 RISC-V 标准的异常向量号，如 表 3.1 所示：

表 3.1: 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

3.1.1 异常

3.1.1.1 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 3.2 所示。E907 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 3.2: 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 3.1 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 3.3 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。

上述第 1 步中，在精确异常模式下保存触发异常的指令 PC 值到 MEPC 寄存器中，在非精确异常模式下，MEPC 不能精确保存触发异常的指令 PC 值，而有可能是指令流执行顺序上触发异常指令后面的指令对应 PC 值，但这种情况下 MTVAL 仍是正确更新。关于精确异常和非精确异常的描述请参考精确与非精确异常 。

上述第 6 步中，响应 NMI 时不会更新 MTVAL 寄存器，可以保证即便在异常服务程序中响应 NMI 请求也不会覆盖 MTVAL 寄存器。关于 NMI 的相关处理描述请参考*NMI* 。

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 E907 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 3.3: 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

3.1.1.2　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 CPU 跳转到该入口执行程序时，软件可依据 MCAUSE 寄存器中的异常向量号来决定是否在异常服务程序中实现再次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。

3.1.1.3　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，精确异常模式下可以保证 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服务程序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）。

需要说明的是在从异常服务程序返回时，硬件不会对 MCAUSE 寄存器里面的 *EXCEPTION.CODE* 域进行清除。

3.1.1.4　锁定

如前文所述，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。E907 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。

需要说明的是，在 CPU 响应异常时，即便再次响应 NMI 请求也不会导致 CPU 的锁定，反之不然（详见锁定 ）。

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。此时，在调试模式下可以正常调试代码，分析 CPU 被锁定的原因。NMI 打断 CPU 锁定状态时，NMI 服务程序也可以正常执行。

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 0xEFFFFFFC，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 0xEFFFFFFC，CPU 在退出调试模式时锁定状态也被清除（包括传递给 SoC 的指示 CPU 处于锁定状态的 LOCKUP 有效信号以及 MEXSTATUS.LOCKUP），以及导致 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。

特殊说明的是，执行 ECALL 或者 EBREAK 主动触发的异常在和任何异常类型（包括 NMI）进行嵌套时都不会触发 CPU 的锁定。

3.1.1.5　非对齐访问异常

在 RISC-V 标准定义中，当取指或者内存数据读写操作地址不对齐时会上报相应的非对齐访问异常。E907 取指令地址在 CPU 复位启动后由硬件维护，不会产生非对齐访问异常。内存数据操作的地址由软件定义，因此当访存的数据位宽和地址不对齐时按照 RISC-V 标准需产生加载指令或者存储指令非对齐访问异常。对于双精度浮点来说，双精度内存加载或者存储地址需要是双字对齐。

对于 MCU 应用场景而言，系统对于内存资源耗费比较敏感，为了节约内存空间，软件编程时会存在非对齐内存访问的情况。为了满足这一应用需求，E907 实现了非对齐访问异常掩码开关，由扩展状态寄存器 MXSTATUS 中的扩展状态寄存器（*MXSTATUS*）中的 *MM* 位来控制非对齐内存访问是按照标准定义触发非对齐访问异常抑或不上报异常，硬件做拆分访问处理。

3.1.1.6　精确与非精确异常

处理器在发生异常时，会把发生异常时处理器的 PC 值存入 MEPC 寄存器中，如果 MEPC 指向的是真正引起异常的指令，则称之为精确异常，反之则是非精确异常。

E907 支持总线返回错误引发的访问错误异常精确响应和非精确响应两种方式。由扩展隐式操作寄存器（*MHINT*）中的精确异常使能位控制，默认模式为总线访问错误非精确响应。当精确异常使能位打开后，总线访问错误异常得到精确响应。对于非精确总线错误异常，存入 MEPC 寄存器的 PC 值不一定指向真正引发总线错误的指令，但是 MTVAL 存放的内存访问地址是引发总线错误的地址。

在非精确异常模式下，除了 Load 和 Store 指令访问内存触发的内存访问错误异常外，其他异常都是精确异常，包括 PMP 权限违反导致的内存访问错误异常。但是因 PMP 权限违反导致的内存加载或存储访问错误异常与外部总线返回错误导致的访问错误异常共享异常向量号，因此 E907 在 MEXSTATUS 寄存器中扩展实现了 BUSERR 指示位，用于区分具体是由哪种情况导致的内存访问错误异常。

对于精确响应的异常，CPU 响应异常时存入 MEPC 和 MTVAL 的值都是触发异常的指令对应的信息。

非精确异常模式可以使得 E907 的 Load/Store 内存访问性能得到进一步的提升。在非精确异常模式下调试内存访问错误异常（5 号和 7 号）时，可以根据异常向量号和 MEPC 回溯到最近的一条 Load 或者 Store 指令，然后根据 MEXSTATUS 中的 *BUSERR* 位和 MTVAL 寄存器来定位分析。

3.1.2　中断

中断作为外部事件请求，处理器在响应中断时需要进行如下判断：

- 中断使能位是否开启以及中断优先级是否足够；
- 处理器需要保存的现场有哪些；
- 处理器跳转执行的中断服务程序入口在哪里；
- 在从中断服务程序返回时处理器现场如何恢复；

E907 设计实现了 RISC-V 标准的 CLINT 中断，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，RISC-V 标准的中断控制器 CLIC。

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 MINTTHRESH 域值寄存器设定的域值（中断优先级非零才可正常参与仲裁）；
- 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。

E907 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。E907 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。

3.1.2.1　矢量中断

可通过将 CLIC 中每个中断的中断属性寄存器（*CLICINTATTR*）中的 shv 域设置为 1 来指示该中断为硬件矢量中断。

中断优先级　E907 支持中断优先级有效位 2-5 位硬件可配，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，该位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 中断控制器* 。

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 以及该中断对应的中断使能寄存器（*CLICINTIE*）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。因为中断优先级比异常高，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 3.1 异常向量号所示。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 (2' b11)，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 域被更新为当前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，需要软件在异常服务程序中清除。

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矢量中断基址寄存器（*MTVT*）。

中断处理　在处理器跳转到中断服务程序执行时，首先需要对处理器之前正常运行的通用寄存器现场进行保存。在运行实时操作系统时，为了减少每个任务的中断现场保存所占空间，会需要一个单独的所有任务共享的中断栈。RISC-V 定义了 MSCRATCHCSWL 寄存器，它的实体寄存器为 MSCRATCH。假如当前中断是 CPU 从正常执行程序流跳转过来响应的，对 MSCRATCHCSWL 寄存器的读写可实现 sp 与 MSCRATCH 寄存器的交换，而 MSCRATCH 寄存器可以保存中断栈的指针值，进而在中断服务程序中使用单独的中断栈进行现场保存和恢复。

在实时操作系统中，E907 所配套的 SDK 采用 3 号机器模式软件中断来进行 task 的切换，在 3 号中断的服务程序中所保存和恢复的 CPU 现场需要存储在各 task 的栈空间中，因此在 3 号中断服务程序中所用的 SP 不能是中断栈，另外 NMI 的服务程序中所用栈指针也不是中断栈指针。

假如需要嵌套响应中断，需要对当前的 MSTATUS，MCAUSE，MINTSTATUS 等寄存器进行保存并重新打开 MSTATUS 中的中断使能位。这些操作均属于真正的中断处理操作之前的必须操作。

为了加速中断的响应，尽快开始真正的中断事务处理，首先在扩展异常状态寄存器 MEXSTATUS 中定义了 SPSWAPEN 位，可由软件控制是否开启上述中断栈的硬件切换功能。另外，E907 上定义实现了扩展中断加速压栈指令 IPUSH，该指令可实现对 ABI 所定义的中断入口所要保存的 GPR（X1，X5-X7，X10-X17，X28-X31，共 16 个 GPR）进行压栈，并且对 MEPC 以及 MCAUSE 两个在中断嵌套时需要保存 CSR 进行压栈。该指令执行的最后一个功能是设置 MSTATUS 寄存器中的 MIE 位，使能中断。

为了进一步加快中断的响应，扩展异常状态寄存器 MEXSTATUS 中定义了 SPUSHEN 位，当该位设置为 1 时，CPU 在响应中断时，会投机的执行一条 IPUSH 指令。执行执行过程中会与从中断服务程序入口取回的指令码进行比对，判断是否投机成功。当投机失败时，即从中断服务程序入口取回的第一条指令不是 IPUSH 指令，投机执行的 IPUSH 指令不产生任何实际效果。

中断服务程序入口的 IPUSH 指令可由编译器编译生成，用户只需在高阶语言实现的中断服务函数上添加指定描述符即可。

如果中断服务程序入口需要对浮点寄存器现场进行压栈保存，可由软件通过浮点 Store 指令完成。

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。

当中断服务程序入口是 IPUSH 指令完成的现场压栈时，中断返回操作的同样可由 E907 定义实现的 IPOP 指令完成。IPOP 指令会先将 MSTATUS 寄存器中的中断使能位关闭，并对 IPUSH 指令压栈的现场进行对应的按序弹栈，最后拆分出 MRET 指令进行中断返回。同 IPUSH 指令一样，IPOP 指令同样可由编译器编译生成，因为 IPOP 最后会拆分出 MRET 指令，因此在中断服务程序的最后无需再加上 MRET 指令。

为了在 IPOP 执行过程中加快对更高优先级中断的响应，一旦 CLIC 仲裁出更高优先级中断会打断 IPOP 的执行，转而响应更高优先级的中断，在高优先级中断返回后重新执行这条 IPOP 指令。

如果中断服务程序中需要对浮点寄存器现场进行弹栈操作，需要在 IPOP 指令之前通过浮点 Load 指令完成。

中断咬尾　中断咬尾是指在中断服务程序中中断事务处理完成后，处理器现场弹栈之前，对 CLIC 中等待处理的较低优先级中断（如果存在的话）进行查询，如果该优先级比 MCAUSE.mpil 寄存器值大的话，会优先响应该等待处理的中断，跳过当前按中断服务程序中的处理器现场弹栈操作，在新响应的中断服务程序中再进行现场的弹栈。

如果采用 IPOP 指令来实现中断的返回，在 IPOP 指令执行之初，会对 CLIC 仲裁出来的等待响应的低优先级中断进行查询响应，并投机的认为新响应中断的入口是 IPUSH 指令，实现中断的咬尾操作。因为处理器现场是在新响应的中断服务程序的最后进行弹栈恢复，这就要求咬尾响应的中断入口必须是 IPUSH 指令，结尾同样通过 IPOP 指令实现中断的返回。

成功进行中断咬尾时，当前中断的 IPOP 和咬尾中断的 IPUSH 指令会立即执行完成，同时不产生任何实际效果。

如果 IPOP 指令执行最后确实需要中断返回，即没有咬尾中断，如果 MCAUSE.MPIL 为 0，代表当前中断为嵌套响应的最外层中断，在扩展异常状态寄存器 MEXSTATUS 中定义了 SPSWAPEN 位为 1 时，硬件会将中断栈和 SP 寄存器进行切换。

3.1.2.2　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。

中断优先级　同中断优先级 节所描述。

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。

中断处理　因为非矢量中断的中断服务入口和异常处理入口相同，都由 MTVEC 寄存器指定，因此非矢量中断服务程序的第一条指令无法使用 IPUSH 指令（因为它同样是异常处理程序的第一条指令）。在中断服务程序的入口，可通过 MCAUSE 寄存器的 INTR 域分辨出异常和中断，进而跳转到指定中断服务程序处理。真正的中断事务处理前的处理器现场可通过标准的 Store 指令完成。

中断返回　与上节的中断处理类似，中断返回前可先通过标准的 Load 指令完成处理器现场的弹栈，接着执行 MRET 指令实现从中断服务程序返回到响应中断之前的执行程序流。

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。

3.1.3　NMI

NMI 作为外部事件，E907 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，E907 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。

3.1.3.1　NMI 响应

CPU 在响应 NMI 时会将当前的状态信息保存下来，主要有：

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，bit[31] 中断标志位设置为 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；
5. 将扩展异常状态寄存器 MEXSTATUS 中的 NMI 域值置为 1，表明处理器正在处理 NMI；
6. 硬件不会更新 MTVAL 寄存器。

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 *4.3.3* 节所述。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。

3.1.3.2　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）。

3.1.3.3　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 节所述，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。

3.2　CLINT 中断

E907 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。

3.2.1　寄存器地址映射

CLINT 中断占据 64KB 内存空间。高 4 位地址空间由系统集成决定，低位地址映射如 表 3.4 所示。所有寄存器仅支持字对齐的访问。

表 3.4: ClINT 寄存器存储器映射地址

地　址 [27:0]	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	-	-	-	-

3.2.2　软件中断

CLINT 可用于软件配置产生软件中断。机器模式软件中断由机器模式软件中断配置寄存器（MSIP）控制。

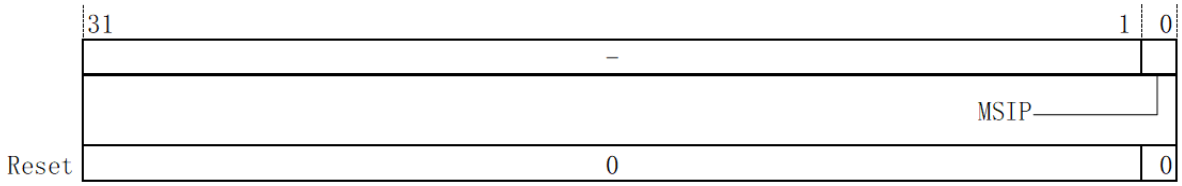


图 3.1: 机器模式软件中断配置寄存器（MSIP）

MSIP-机器模式软件中断等待位:

该位表示机器模式软件中断的中断状态，机器模式下可以对该位进行读写。

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

3.2.3　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 E907 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 E907 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 E907 内部，高 32 位的值可通过 E907 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 E907 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 3.2 所示。

	63 — 32	31 — 0
	MTIMECMPH	MTIMECMPL
Reset	0xffffffff	0xffffffff

图 3.2: 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。

	63 — 32	31 — 0
	MTIMEHI	MTIMELO
Reset	0x0	0x0

图 3.3: 机器模式计时器当前值寄存器（高/低）

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：

- 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。

3.3 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。E907 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。

3.3.1　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 3.5 。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 3.5: CLIC 地址映射

地址 [27:0]	名称	类型	初始值	描述
0x800000	CLICCFG	RW	0x1	CLIC 配置寄存器
0x800004	CLICINFO	RO	详见计时器中断	CLIC 信息寄存器
0x800008	MINTTHRESH	RW	0x0	中断阈值寄存器
0x801000+4*i	CLICINTIP[i]	R or RW	0x0	中断源 i 等待寄存器
0x801001+4*i	CLICINTIE[i]	RW	0x0	中断源 i 使能寄存器
0x801002+4*i	CLICINTATTR[i]	RW	0x0	中断源 i 属性寄存器
0x801003+4*i	CLICINTCTL[i]	RW	0x0	中断源 i 控制寄存器

3.3.2　CLIC 寄存器描述

3.3.2.1　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 3.4 所示。

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

图 3.4: 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 E907 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。

3.3.2.2　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 3.5 所示。

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

图 3.5: 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，外部至多连接 240 个中断源。

3.3.2.3　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 3.6 所示。

	31 24	23 0
	mth	-
Reset	0	0

图 3.6: 中断阈值寄存器 (MINTTHRESH)

mth-机器模式阈值：

机器模式中断的阈值。由于 E907 仅支持机器模式中断，所以仅有一个阈值域。

3.3.2.4　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 3.7 所示。

	7 … 1	0
	–	IP
Reset	0	0

图 3.7: 中断等待寄存器 (CLICINTIP)

IP-中断等待：

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

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

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

3.3.2.5　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 3.8 所示。

	7 … 1	0
	–	IE
Reset	0	0

图 3.8: 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。

3.3.2.6　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 3.9 所示。

mode-中断特权态：

该域用于配置中断的特权态，由于 E907 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：

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

图 3.9: 中断属性寄存器 (CLICINTATTR)

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。

3.3.2.7　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 3.10 所示。

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

图 3.10: 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 3.6 所示。

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

表 3.6: 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。

- nlbits 复位后为 0，如果不配置该寄存器域的话，所有中断的优先级均默认为 8’hFF。

CLICCFG.nlbits 与 CLICINTCTL 划分不完全示例如 表 3.7 所示：

表 3.7: 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。

第四章　存储子系统与接口

4.1　内存子系统

E907 兼容实现了 32 位 RISC-V 指令集架构，可以访问 4GB 全地址空间。E907 对内存地址空间的访问可以通过如下接口实现：

- 顶层 AXI 4.0 总线接口；
- 顶层 AHB 5.0 总线接口；
- 内部 TCIP 总线接口；

4.2　内存模型

玄铁 E907 支持两种内存类型，分别是存储（以下简称 Memory 类型）和外设（以下简称 Device 类型）。Device 类型内存需要配置成 Strong Order 属性，即 CPU 对该片地址空间的访问顺序跟指令序列顺序完全一致，在配置为 Strong Order 后，就不可高缓，对该地址空间的访问不可缓存进指令或者数据 cache。Memory 类型内存需要配置成 Weak Order 属性（Strong Order 配置为 0）。Memory 类型内存支持配置为是否可高缓（cacheable 或者 non-cacheable），对于 cacheable 的内存地址空间，E907 从该区域访问得到的数据可以缓存在 E907 内部的指令或者数据 cache 中。

E907 还支持对内存配置为是否可暂存的内存（bufferable 或者 non-bufferable）。而对于 bufferable 的内存地址空间，E907 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 E907 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，E907 对该地址空间的写操作的响应必须在 E907 访问的 slave 上写操作完成后由该 slave 返回给 E907。

全空间内存属性的配置在 E907 硬件集成时指定默认值并可通过映射在固定地址空间 0xEFFF_F000 起始的 8 组寄存器来动态调整，如下所示：

表 4.1: sysmap 寄存器配置

寄存器	地址	描述
SYSMAP_BASE_ADDR0	0xEFFF_F000	地址段 0 的地址上限（不包含）
SYSMAP_ADDR0_ATTR	0xEFFF_F004	地址段 0 的内存属性
SYSMAP_BASE_ADDR1	0xEFFF_F008	地址段 1 的地址上限（不包含）
SYSMAP_ADDR1_ATTR	0xEFFF_F00C	地址段 1 的内存属性
SYSMAP_BASE_ADDR2	0xEFFF_F010	地址段 2 的地址上限（不包含）
SYSMAP_ADDR2_ATTR	0xEFFF_F014	地址段 2 的内存属性
SYSMAP_BASE_ADDR3	0xEFFF_F018	地址段 3 的地址上限（不包含）
SYSMAP_ADDR3_ATTR	0xEFFF_F01C	地址段 3 的内存属性
SYSMAP_BASE_ADDR4	0xEFFF_F020	地址段 4 的地址上限（不包含）
SYSMAP_ADDR4_ATTR	0xEFFF_F024	地址段 4 的内存属性
SYSMAP_BASE_ADDR5	0xEFFF_F028	地址段 5 的地址上限（不包含）
SYSMAP_ADDR5_ATTR	0xEFFF_F02C	地址段 5 的内存属性
SYSMAP_BASE_ADDR6	0xEFFF_F030	地址段 6 的地址上限（不包含）
SYSMAP_ADDR6_ATTR	0xEFFF_F034	地址段 6 的内存属性
SYSMAP_BASE_ADDR7	0xEFFF_F038	地址段 7 的地址上限（不包含）
SYSMAP_ADDR7_ATTR	0xEFFF_F03C	地址段 7 的内存属性

sysmap 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 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 设定的 8 个地址区间的地址属性默认为 SO，Non-Cacheable 和 Non-Bufferable。每个地址空间上下边界是 4KB 对齐，因此 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 SYSMAP_ADDR**i**_ATTR *(***i** 从 *0* 到 *7)* 定义，具体如 图 4.1 所示：

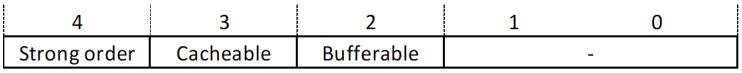


图 4.1: sysmap 地址属性配置

SYSMAP_BASE_ADDR 寄存器配置的是地址段地址上限的高 20 位，亦即需要将真实地址右移 12 位后配置进 SYSMAP_BASE_ADDR 寄存器。

对于 CLINT 和 CLIC 所在的 TCIP 地址空间需要设置为 SO 以及 Non-Cacheable，Non-Bufferable 属性。sysmap 寄存器所在的 0xEFFF_F000-0xEFFF_FFFF 地址空间固定为 SO，Non-Cacheable, Non-Bufferable。外设 AHB 总线接口所连接地址空间属性固定为 Non-Cacheable。

4.3　物理内存保护

4.3.1　PMP 简介

E907 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。

E907 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 128 字节；
- 支持 OFF、TOR（Top of Range）、NAPOT (Naturally Aligned Power-of-Two Region）三种地址匹配模式，不支持 NA4（Naturally Aligned 4-Byte Region）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。

4.3.2　PMP 控制寄存器

4.3.2.1　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 32 位的 PMPCFG 提供 4 个表项的权限设置，整体分布如 图 4.2 所示。

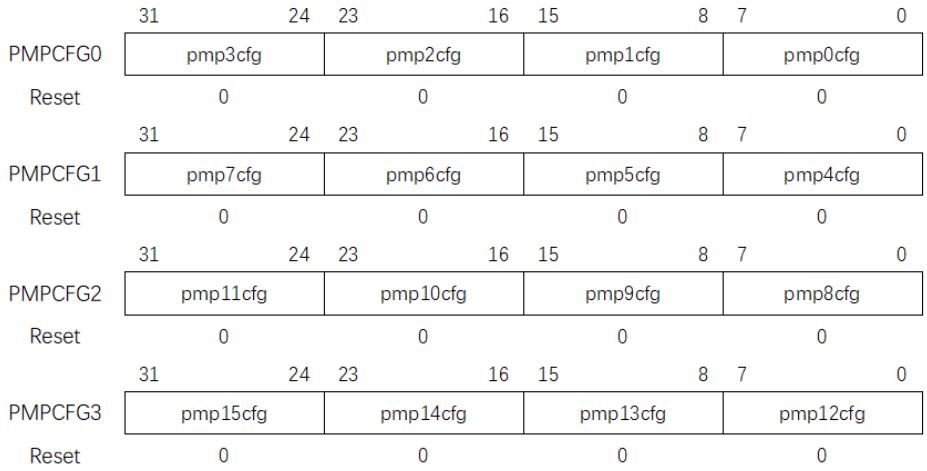


图 4.2: 物理内存保护设置寄存器整体分布图

每个物理内存保护设置寄存器的格式如 图 4.3 所示。

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

图 4.3: 物理内存保护设置寄存器格式

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；

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；由于 E907 地址划分最小粒度为 128 字节，不支持 NA4 匹配模式；

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。

4.3.2.2　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$_{i-1}$[31:5], 7’ b0}, {PMPADDR$_i$[31:5], 7’ b0})，PMPADDR$_i$[4:0] 的值不参与地址匹配逻辑运算。对于表项 0，使用 0x0 作为地址空间下边界，即 [0, {PMPADDR$_0$[31:5], 7’ 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 NAPOT 模式，其地址与区域大小的关系如 表 4.2 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。以 PMPADDR$_i$=32’ baaaa_aaaa_aaaa_aaaa_aaaa_aaaa_aaa0_1111 为例，其基址为 34’ baa_aaaa_aaaa_aaaa_aaaa_aaaa_aaaa_a000_0000，空间大小为 128B。

需要注意的是，E907 PMP 支持的地址划分最小粒度为 128B。因此，在 NAPOT 模式下，PMPADDR$_i$[3:0] 的值不参与地址匹配逻辑运算且视为 4’ b1111，用于 表 4.2 查表的值为 {PMPADDRi[31:4], 4’ b1111}，上文中的 n 将恒大于 4。例如给 PMPADDR$_i$ 写入 32’ baaaa_aaaa_aaaa_aaaa_aaaa_aaaa_0111_0111 时，用于查表的值为 32 ‘baaaa_aaaa_aaaa_aaaa_aaaa_aaaa_0111_1111，此时 n 等于 7，最终该表项的基址为 34 ‘baa_aaaa_aaaa_aaaa_aaaa_aaaa_aa00_0000_0000，空间大小为 1KiB。

PMPADDR 寄存器最大支持 16GB 的物理空间，但 E907 采用 32 位地址，最大支持 4GB。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 4.2: 保护区间编码

PMPADDRi	PMPCFG.A	匹配区域大小
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	1KB
aaaa_aaaa_aaaa_aaaa_aaaa_aaa0_1111_1111	NAPOT	2KB
aaaa_aaaa_aaaa_aaaa_aaaa_aa01_1111_1111	NAPOT	4KB
aaaa_aaaa_aaaa_aaaa_aaaa_a011_1111_1111	NAPOT	8KB
aaaa_aaaa_aaaa_aaaa_aaaa_0111_1111_1111	NAPOT	16KB
aaaa_aaaa_aaaa_aaaa_aaa0_1111_1111_1111	NAPOT	32KB
aaaa_aaaa_aaaa_aaaa_aa01_1111_1111_1111	NAPOT	64KB
aaaa_aaaa_aaaa_aaaa_a011_1111_1111_1111	NAPOT	128KB
aaaa_aaaa_aaaa_aaaa_0111_1111_1111_1111	NAPOT	256KB
aaaa_aaaa_aaaa_aaa0_1111_1111_1111_1111	NAPOT	512KB
aaaa_aaaa_aaaa_aa01_1111_1111_1111_1111	NAPOT	1MB
aaaa_aaaa_aaaa_a011_1111_1111_1111_1111	NAPOT	2MB
aaaa_aaaa_aaaa_0111_1111_1111_1111_1111	NAPOT	4MB
aaaa_aaaa_aaa0_1111_1111_1111_1111_1111	NAPOT	8MB
aaaa_aaaa_aa01_1111_1111_1111_1111_1111	NAPOT	16MB
aaaa_aaaa_a011_1111_1111_1111_1111_1111	NAPOT	32MB
aaaa_aaaa_0111_1111_1111_1111_1111_1111	NAPOT	64MB
aaaa_aaa0_1111_1111_1111_1111_1111_1111	NAPOT	128MB
aaaa_aa01_1111_1111_1111_1111_1111_1111	NAPOT	256MB
aaaa_a011_1111_1111_1111_1111_1111_1111	NAPOT	512MB
aaaa_0111_1111_1111_1111_1111_1111_1111	NAPOT	1GB
aaa0_1111_1111_1111_1111_1111_1111_1111	NAPOT	2GB
aa01_1111_1111_1111_1111_1111_1111_1111	NAPOT	4GB
a011_1111_1111_1111_1111_1111_1111_1111	NAPOT	8GB
0111_1111_1111_1111_1111_1111_1111_1111	NAPOT	16GB

4.3.3　内存保护

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*）。

4.4　高速缓存

E907 采用哈佛结构的一级高速缓存，包含独立的指令高速缓存和数据高速缓存。

4.4.1　指令高速缓存子系统

L1 指令高速缓存的主要特征如下：

- 指令高速缓存大小硬件可配置，支持 2KB/4KB/8KB/16KB/32KB;
- 2 路组相联，缓存行大小为 32 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。

4.4.1.1　分支预测器

E907 采用分支历史表对条件分支的跳转方向进行预测。使用 2-bit 饱和计数预测器，采用 8 路组相连存储结构，每周期支持一条分支结果预测。

分支历史表由分支历史寄存器和分支预测器缓存两部分组成。分支历史寄存器分为两部分，分别记录分支指令跳转历史的预测轨迹和执行轨迹。分支历史表通过分支预测轨迹寄存器对分支预测器缓存进行索引，得到 8 路预测器结果后，结合分支预测轨迹寄存器和分支指令 PC 对结果进行选择，得到最终的预测结果。当发生分支预测失败时，使用分支执行轨迹寄存器覆盖分支预测估计寄存器，并根据预测结果和实际执行跳转情况，训练预测失败对应的 2-bit 饱和计数器。

分支历史表进行预测的条件分支指令包括：

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

4.4.1.2　分支跳转目标缓存器

E907 使用分支跳转目标缓存器（BTB）对分支指令的跳转目标地址进行预测。分支跳转目标缓存器对分支指令历史目标地址进行记录，如果当前分支指令命中分支跳转目标预测器，则将记录目标地址作为当前分支指令预测目标地址，直接在 IF 级流水线发起分支跳转。在 ID 级流水线，处理器使用分支指令的预译码信息和分支预测对 BTB 的预测目标进行判断，如果 BTB 预测失败，则将对应的 BTB 表项进行无效化，同时在 ID 级发起正确的分支跳转。

分支跳转目标预测器主要特征包括：

- 16 个表项，采用全相联形式；
- 只存储发生跳转的分支指令；

- 无需配合分支预测信息，命中 BTB 则即刻发起跳转；
- BTB 命中时立即发起跳转，无额外跳转损失；
- 使用当前分支指令部分 PC 进行索引。

使用分支跳转目标预测器进行预测的分支指令包括：

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

4.4.1.3　返回地址预测器

返回地址预测器用于函数调用结束时，返回地址的快速准确预测。当取指单元译码得到有效的函数调用指令时，将函数返回地址压栈存入返回地址预测器；当取指单元译码得到有效的函数返回指令时，则从返回地址预测器弹栈，获取函数返回目标地址。

返回地址预测器最多支持 6 层函数调用嵌套，超出嵌套次数会导致目标地址预测错误。支持基于指针的 RAS 纠错，在流水线后级发生跳转后通过对应指针恢复 RAS 指针；支持基于表项的 RAS 纠错，在 RAS 预测失败时，通过无效化对应表项的方式纠正。

函数调用指令包括：

- JAL、JALR、C.JALR。

函数返回指令包括：

- JALR、C.JR、C.JALR。

指令功能的具体划分可以参见 表 4.3 。

表 4.3: 指令功能具体划分

指令的目的寄存器 rd	指令的源寄存器 rs1	RAS 行为
rd != x1	rs1 != x1	无操作
rd != x1	rs1 == x1	pop
rd == x1	rs1 != x1	push
rd == x1	rs1 == x1	push and pop

4.4.2　数据高速缓存子系统

数据高速缓存的主要特征如下：

- 数据高速缓存大小硬件可配置，支持 2KB/4KB/8KB/16KB/32KB；
- 2 路组相联，缓存行大小为 32 字节；
- 采用先进先出的替换策略；
- 写策略支持写直-写分配、写直-写不分配、写回-写分配和写回-写不分配四种模式；
- 支持对整个高速缓存的无效和清除操作，支持对单条缓存行的无效和清除操作；
- 支持 exclusive 和 atomic 操作。

4.4.2.1　原子操作

E907 支持 RISC-V 的 A 指令扩展，包括 AMO、LR、SC 指令。用户可以使用这些指令构成原子锁等同步原语实现同一个核不同进程之间或者不同核之间的同步。对于 E907，推荐的方式是使用 AMO 指令构成原子锁操作，并且被操作的原子锁的地址属性必须是 Non-cacheable。如果采用其他方式，则操作结果为 UNPREDICTABLE。

4.4.3　高速缓存操作

在处理器复位后，指令和数据高速缓存会自动进行无效化操作，但是指令和数据高速缓存默认关闭。在 RISC-V 编程模型基础上，E907 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、清脏表项和无效化操作。

4.4.3.1　高速缓存扩展寄存器

E907 支持高速缓存扩展寄存器：机器模式硬件配置寄存器 (mhcr)。可以实现对指令和数据高速缓存的开关以及写策略的配置。

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）*。

4.4.3.2　高速缓存扩展指令

E907 扩展了高速缓存相关操作的指令，包括：按地址进行无效化、无效化全部、按地址清脏表项、清全部脏表项、按地址清脏表项并无效化和清全部脏表项并无效化，具体如 表 4.4 所示。

表 4.4: Cache 扩展指令

指令	描述
DCACHE.IPA	DCACHE 无效物理地址匹配表项指令
DCACHE.CPA	DCACHE 清除物理地址匹配表项指令
DCACHE.CIPA	DCACHE 清除并无效物理地址匹配表项指令
DCACHE.ISW	DCACHE 无效 way/set 指向表项指令
DCACHE.CSW	DCACHE 清除 way/set 指向表项指令
DCACHE.CISW	DCACHE 清除并无效 way/set 指向表项指令
DCACHE.IALL	DCACHE 无效全部表项指令
DCACHE.CALL	DCACHE 清除全部表项指令
DCACHE.CIALL	DCACHE 清除并无效全部表项指令
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

4.5　内存加速访问

面向中高端 MCU 以及 MPU 应用，E907 设计实现了 AXI4.0 总线接口和快速外设访问 AHB5.0 总线接口。为了降低内存访问延时，提高流水线快速获取数据的能力，E907 设计实现了多种机制：

- E907 内部设计实现了哈佛结构的指令和数据 Cache，大小 2KB-32KB 可配置；

- E907 在存储读写单元设计了多种预取策略，并根据需求可选配不同的资源配置：1）E907 数据 CACHE 实现了全局和基于流的预取器，支持顺序和 Stride 预取，最多可预取 6 条缓存行；2）可根据额需求硬件配置实现 2 个或者 6 个 Outstanding 传输；
- E907 支持 AMR 操作，在连续多个缓存行写缺失时关闭 write_allocate；
- 为了支持对外设的快速访问，E907 设计实现了一条 AHB5.0 总线接口，通过该总线接口发出的内存访问请求在核内固定为 Non-Cacheable，可以从核内的存储读写单元不经过数据 CACHE 直接发到总线接口上，提升了外设访问的速度。

通过上述机制，E907 在运行常见的如内存拷贝等应用时表现出了优异的性能。

4.6　内存访问顺序

如上节所述，E907 支持两条总线接口，CPU 执行内存访问操作会根据访问地址信息分配到不同的总线接口发送到核外。同时，E907 为了加速内存访问，不同地址且没有数据相关性的内存访问操作在总线传输顺序上会存在和指令发射顺序不一致的情况。对于 SoC 上存储指令或者数据的 Memory 来说，CPU 基于该机制可以获取较好的整体性能，且软件编程人员无感知。对于需要保证内存访问顺序的内存空间来说，需明确以下特点：

- CPU 发往两条总线的内存访问请求会依赖于核外的不同总线访问延时而形成长短不一的数据返回延时，软件编程人员如果要保证这两条总线上不同访问的次序的话需要加上 SYNC 操作；
- CPU 支持不存在地址相关性的内存访问请求乱序发往总线，且支持 AXI 总线上请求数据的乱序返回操作，如果要保证 AXI 总线上不同内存访问的次序的话，需要将相关内存空间设置为 Strong Order 属性；

4.7　总线接口

4.7.1　简介

E907 实现了多总线接口，分别包括 AXI 主设备接口，快速外设访问 AHB 接口，以及紧耦合 IP 接口。其中快速外设总线接口和紧耦合 IP 接口的地址空间由系统集成时分别指定。

4.7.2　AXI 接口

E907 内部取指请求均从该总线接口发出去，数据读写请求可根据地址发往不同的总线接口。为了加快数据的预取，系统集成者可以根据资源配置情况配置不同的总线访存能力。

- 性能优先配置下 AXI 总线接口可以发出两条数据 CACHE line 的读传输 Outstanding 请求以及 16 笔写传输的 Outstanding 请求；
- 面积优先配置下 AXI 总线接口可以发出六条数据 CACHE line 的读传输 Outstanding 请求以及 8 笔写传输的 Outstanding 请求；

4.7.3　快速外设访问 AHB 接口

E907 设计实现的快速外设访问接口兼容 AHB5.0 总线协议，连接在该总线接口的地址空间属性固定为 Non-Cacheable，通过该总线接口发到核外的请求不经过核内的数据 CACHE。系统设计者可将需要快速访问的外设连接在该总线接口上。

发往该总线接口上的请求支持 Exclusive 传输，不支持 Burst 传输。同时，发往该总线接口上的请求支持非对齐访问。

第五章　调试

5.1　调试接口

5.1.1　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。

E907 采用标准 5 线 JTAG 调试接口，兼容 RISC-V debug V0.13.2 协议标准。

调试接口的主要特性如下：

- 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；
- 可在普通用户模式下进入调试模式；
- 支持 CPU 全速运行时通过调试端口直接访问内存。

E907 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 5.1 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。

5.1.2　DM 寄存器

下表所示为 E907 DM 模块实现的寄存器列表，除了标准定义寄存器外，E907 还在 DMI Address 编码域上扩展实现了部分 DM 寄存器。

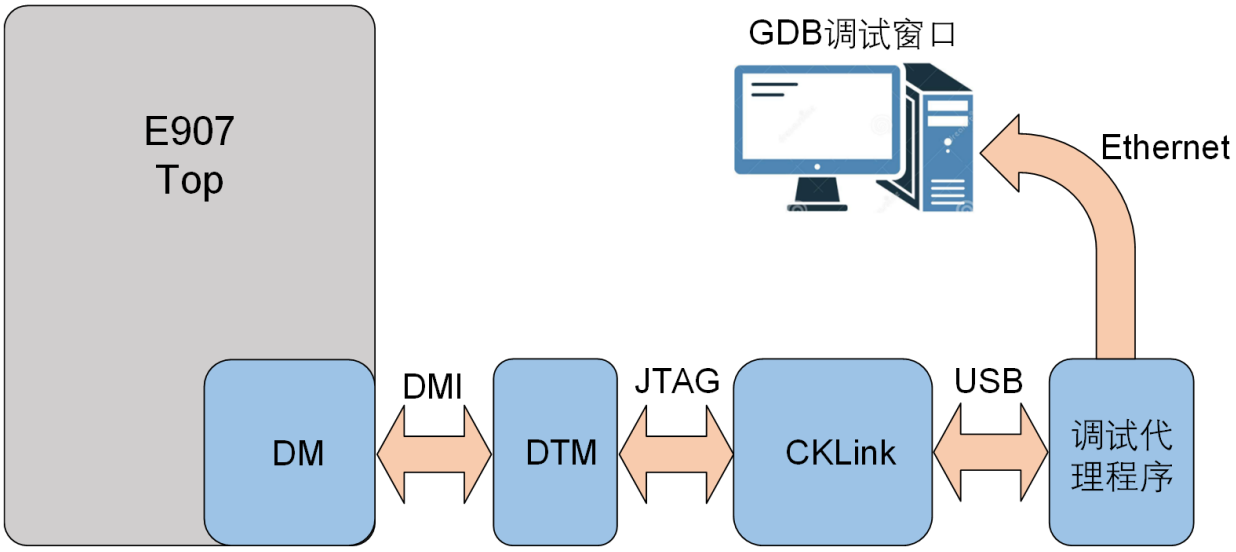


图 5.1: 调试接口在整个 CPU 调试环境中的位置

表 5.1: DM 寄存器映射及描述

地址	寄存器名称	描述
0x4	DATA0	抽象 DATA0
0x5	DATA1	抽象 DATA1
0x10	DMCONTROL	DM 控制寄存器
0x11	DMSTATUS	DM 状态寄存器
0x15	HAWINDOW	Hart 阵列窗口
0x16	ABSTRACTCS	抽象控制和状态寄存器
0x17	COMMAND	抽象命令寄存器
0x18	ABSTRACTAUTO	抽象命令自动执行寄存器
0x1D	NEXTDM	下一个 DM 基址寄存器
0x20-0x2F	PROGBUF0-1F	可编程 Buffer0-15
0x32	DMCS2	DM 控制和状态寄存器 2
0x38	SBCS	SBA 控制和状态寄存器
0x39	SBADDRESS0	SBA 地址 31:0
0x3C	SBDATA0	SBA 数据 31:0
0x40	HALTSUM0	HALT 总览寄存器 0
玄铁扩展寄存器		
0x1F	ITR	指令传输寄存器
0x70	CUSTOMCS	客制控制和状态寄存器
0x71	CUSTOMCMD	客制命令寄存器
0x72-0x79	CUSTOMBUF0-7	客制 Buffer0-7
0x7F	COMPID	组件 ID 寄存器

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

扩展控制和状态寄存器 CUSTOMCS 用于描述玄铁 E907 扩展的抽象命令实现和扩展。寄存器描述如下：

表 5.2: DM CUSTOMCS 寄存器描述

位域	寄存器名称	描述
31:29	custcmderr	在使用 CUSTOMCMD 执行命令时的错误状态：0：没有错误 1：命令不支持
28:25	cusbufcnt	实现的 Buffer 的数
24:28	-	-
17	cuscmdbusy	使用 CUSTOMCMD 执行命令的状态：0：没有执行命令 1：正在执行命令
16	-	-
15:0	coredbginfo	用于表示 E907 支持的核内调试资源情况。

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

表 5.3: DM CUSTOMCMD 寄存器描述

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

扩展组件 ID 寄存器 COMPID 该寄存器用于指明当前 DM 模块实现的内容和实现版本信息。

5.1.3 资源配置

为了方便用户选择，E907 提供了三种调试资源配置组合供用户选择：

- 最小配置 1）1 个 Program Buffer 且实现额外的 EBREAK；2）1 个硬断点；
- 典型配置 1）2 个 Program Buffer 且实现额外的 EBREAK；2）3 个硬断点；3）8 个表项的 PCFIFO 用于记录过往 PC 跳转流；
- 最大配置 1）2 个 Program Buffer 且实现额外的 EBREAK；2）8 个硬断点且可以组成触发链；3）16 个表项的 PCFIFO 用于记录 PC 跳转流; 4）配置单独的调试 AXI 总线接口，用于独立访问内存空间。

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

5.2　性能监测单元

E907 性能监测单元（HPM）遵从 RISC-V 标准，用于统计程序运行中的软件信息和部分硬件信息，供软件开发人员进行程序优化。

性能监测单元统计的软硬件信息分为以下几类：

- 程序运行时间统计；
- 指令信息统计；
- 处理器关键部件信息统计。

鉴于 E907 实现了机器模式和用户模式，因此 HPM 实现了一套机器模式事件相关选择器与计数器，如 MHPMEVENTn 与 MHPMCOUNTERn 等，以及用户模式下相对应的只读镜像计数寄存器（HPMCOUNTERn 等）。用户可通过在机器模式下配置机器模式计数器访问授权寄存器（MCOUNTEREN）来授权用户模式下是否可以正常访问只读镜像计数寄存器（HPMCOUNTERn）。

5.2.1　性能监测控制寄存器

HPM 控制寄存器主要为：机器模式计数器访问授权寄存器（MCOUNTEREN）和机器模式计数器禁止寄存器(MCOUNTINHIBIT)。

5.2.1.1　机器模式计数器访问授权寄存器（MCOUNTEREN）

机器模式计数器访问授权寄存器（MCOUNTEREN），用于授权用户模式是否可以正常访问机器模式计数器。

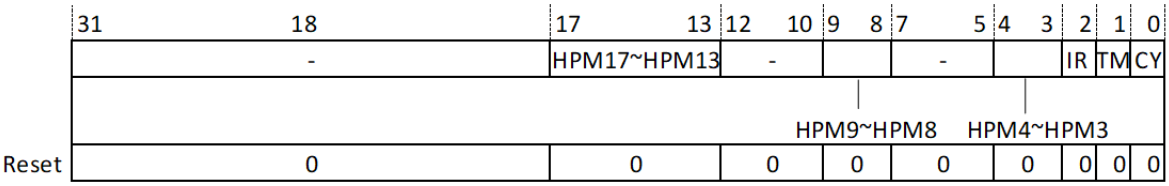


图 5.2: 机器模式计数器访问授权寄存器（MCOUNTEREN）

机器模式计数器访问授权寄存器说明如 表 5.4 所示。

表 5.4: 机器模式计数器访问授权寄存器说明

域	读写	名称	介绍
31:18	NA	NA	保留。
17:13	读写	HPMn（n 为 17~13）	HPMCOUNTERn 寄存器用户模式访问授权位： 0：用户模式访问 HPMCOUNTERn 将发生非法指令异常； 1：用户模式能正常访问 HPMCOUNTERn。
12:10	NA	NA	保留。
9:8	读写	HPMn（n 为 9~8）	HPMCOUNTERn 寄存器用户模式访问授权位： 0：用户模式访问 HPMCOUNTERn 将发生非法指令异常。 1：用户模式能正常访问 HPMCOUNTERn。
7：5	NA	NA	保留。
4:3	读写	HPMn（n 为 4~3）	HPMCOUNTERn 寄存器用户模式访问授权位： 0：用户模式访问 HPMCOUNTERn 将发生非法指令异常。 1：用户模式能正常访问 HPMCOUNTERn。
2	读写	IR	MINSTRET 寄存器用户模式访问授权位： 0：用户模式访问 MINSTRET 寄存器将发生非法指令异常； 1：用户模式能正常访问 MINSTRET 寄存器。
1	读写	TM	MTIMELO 和 MTIMEHI 寄存器用户模式访问授权位： 0：用户模式访问 MTIMELO 和 MTIMEHI 寄存器将发生非法指令异常； 1：用户模式能正常访问 MTIMELO 和 MTIMEHI 寄存器。
0	读写	CY	MCYCLE 寄存器用户模式访问授权位： 0：用户模式访问 MCYCLE 寄存器将发生非法指令异常； 1：用户模式能正常访问 MCYCLE 寄存器。

5.2.1.2　机器模式计数禁止寄存器（MCOUNTINHIBIT）

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

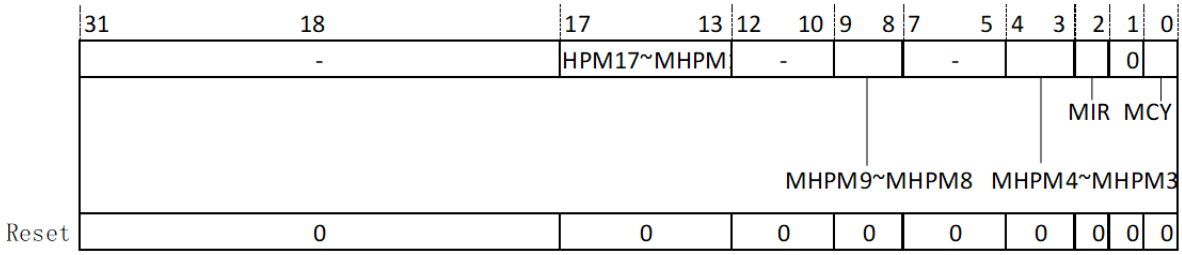


图 5.3: 机器模式计数禁止授权寄存器（MCOUNTINHIBIT）

机器模式禁止计数寄存器说明如 表 5.5 所示。

表 5.5: 机器模式计数禁止授权寄存器寄存器说明

域	读写	名称	介绍
31:18	NA	NA	保留
17:13	读写	MHPM*n*（n 为 17~13）	MHPMCOUNTER*n* 寄存器禁止计数位： 0：正常计数； 1：禁止计数。
12:10	NA	NA	保留。
9:8	读写	MHPM*n*（n 为 9~8）	MHPMCOUNTER*n* 寄存器禁止计数位： 0：正常计数； 1：禁止计数。
7:5	NA	NA	保留。
4:3	NA	MHPM*n*（n 为 4~4）	MHPMCOUNTER*n* 寄存器禁止计数位： 0：正常计数； 1：禁止计数。
2	读写	MIR	MINSTRET 寄存器禁止计数位： 0：正常计数； 1：禁止计数。
1	NA	NA	因为系统计时器位于核外，可供芯片系统中多个模块读取，因此 CPU 内不存在禁止计数位。
0	读写	MCY	MCYCLE 寄存器禁止计数位： 0：正常计数； 1：禁止计数。

5.2.1.3　扩展控制寄存器

因为用户模式计数器为机器模式计数器对应的镜像寄存器，在机器模式计数禁止寄存器的基础上，为了区分机器模式和用户模式的计数使能，在扩展状态寄存器 MXSTATUS 中，E907 实现了机器模式计数器计数开关位 *PMDM* （参见*扩展状态寄存器（MXSTATUS）*）和用户模式计数器计数开关位 *PMDU* （参见*扩展状态寄存器（MXSTATUS）*）。

5.2.2　性能监测事件选择寄存器

性能监测事件选择寄存器（MHPMEVENT*n*,n 为 17~13，9~8 和 4~3），用于设置其唯一对应（MHPMEVENTn 对应 MHPMCOUNTERn）的计数器所计事件的事件号。E907 中，只有当事件选择器 MHPMEVENTn 设置成唯一对应事件的事件号（见 表 5.7 ），并通过 csrw 指令初始化对应事件计数器后，计数器才能正常计数。比如 MHPMEVENT3 只有设置为 0x1，MHPMCOUNTER3 才会正常计数 ICACHE 的访问次数。

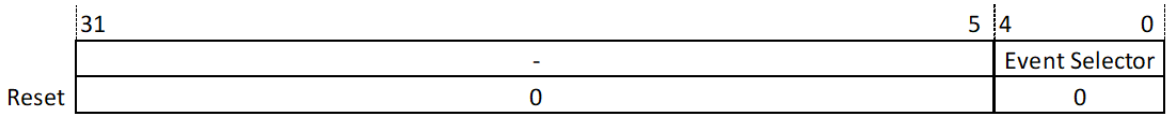


图 5.4: 机器模式性能监测事件选择寄存器（MHPMEVENTn）

机器模式性能监测事件选择寄存器说明如 表 5.6 所示。

表 5.6: 机器模式性能监测事件选择寄存器说明

域	读写	名称	介绍
31:5	NA	NA	保留。
4:0	读写	事件索引	性能监测事件事件号： 0：没有事件； 0x1~0x1A：硬件实现的性能监测事件，具体如 表 5.7 所示。 >0x1A：硬件未定义的性能监测事件，为软件自定义事件使用。

事件选择器和事件以及计数器之间的对应关系如 表 5.7 所示。

表 5.7: 计数器事件事件号及对应计数器寄存器

事件选择器	事件号	事件	计数器
MHPMEVENT3	0x1	L1 ICache Access Counter	MHPMCOUNTER3
MHPMEVENT4	0x2	L1 ICache Miss Counter	MHPMCOUNTER4
MHPMEVENT8	0x6	Conditional Branch Mispredict Counter	MHPMCOUNTER8
MHPMEVENT9	0x7	Conditional Branch Instruction Counter	MHPMCOUNTER9
MHPMEVENT13	0xB	Store Instruction Counter	MHPMCOUNTER13
MHPMEVENT14	0xC	L1 DCache read access Counter	MHPMCOUNTER14
MHPMEVENT15	0xD	L1 DCache read miss Counter	MHPMCOUNTER15
MHPMEVENT16	0xE	L1 DCache write access Counter	MHPMCOUNTER16
MHPMEVENT17	0xF	L1 DCache write miss Counter	MHPMCOUNTER17

只有当机器模式性能监测事件选择寄存器写入上面所述唯一对应的事件号时，相对应的事件计数器才可以正常计数，其计数值含义如 表 5.7 所述。

5.2.3　事件计数器

事件计数器有两组组，分别为：机器模式事件计数器和用户模式事件计数器。具体如 表 5.8 所示：

表 5.8: 机器模式事件计数器列表

名称	索引	读写	初始值	介绍
MCYCLE	0xB00	MRW	0x0	cycle counter
MINSTRET	0xB02	MRW	0x0	instructions-retired counter
MHPMCOUNTER3	0xB03	MRW	0x0	performance-monitoring counter
MHPMCOUNTER4	0xB04	MRW	0x0	performance-monitoring counter
...	...	...	...	...
MHPMCOUNTER17	0xB1F	MRW	0x0	performance-monitoring counter
MCYCLEH	0xB80	MRW	0x0	upper 32bits of cycle counter
MINSTRETH	0xB82	MRW	0x0	upper 32bits of instructions-retired counter
MHPMCOUNTER3H	0xB83	MRW	0x0	upper 32bits of performance-monitoring counter3
MHPMCOUNTER4H	0xB84	MRW	0x0	upper 32bits of performance-monitoring counter4
...	...	...	...	...
MHPMCOUNTER17H	0xB91	MRW	0x0	upper 32bits of performance-monitoring counter17

表 5.9: 用户模式事件计数器列表

名称	索引	读写	初始值	介绍
CYCLE	0xC00	URO	0x0	cycle counter
TIME	0xC01	URO	0x0	timer
INSTRET	0xC02	URO	0x0	instructions-retired counter
HPMCOUNTER3	0xC03	URO	0x0	performance-monitoring counter
HPMCOUNTER4	0xC04	URO	0x0	performance-monitoring counter
...	...	...	...	...
HPMCOUNTER17	0xC1F	URO	0x0	performance-monitoring counter
CYCLEH	0xC80	URO	0x0	cycle counter
TIMEH	0xC81	URO	0x0	timer
INSTRETH	0xC82	URO	0x0	upper 32bits of instructions-retired counter
HPMCOUNTER3H	0xC83	URO	0x0	upper 32bits of performance-monitoring counter3
HPMCOUNTER4H	0xC84	URO	0x0	upper 32bits of performance-monitoring counter4
...	...	...	...	...
HPMCOUNTER17H	0xC91	URO	0x0	upper 32bits of performance-monitoring counter17

其中，用户模式的 CYCLE、INSTRET 和 HPMCOUNTERn 是对应机器模式事件计数器的只读映射，TIME 计数器是 MTIME 寄存器的只读映射，MTIME 为 CLINT 中内存映射的寄存器。

第六章　附录

6.1　标准指令集

E907 兼容标准的 RISC-V 指令集架构，实现了 RV32IMA[F][D]C[P] 指令集，实现的 SPEC 版本如下：

- *The RISC-V Instruction Set Manual, Volume I: RISC-V User-Level ISA, Version 2.2*

P 指令集实现版本为 V0.9，除了实现基准部分指令外，E907 还实现了 Zp64 选配部分指令集。P 指令集实现版本如下：

- *RISC-V "P" Extension Proposal Version 0.9*

标准指令集均可以通过 RISC-V 基金会官网（https://riscv.org/technical/specifications/）下载获取。

6.2　玄铁扩展指令集

6.2.1　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。

6.2.1.1　DCACHE.CALL——DCACHE 清除全部表项指令

语法：

dcache.call

操作：

clear 所有 dcache 表项，将所有 dirty 表项写回到下一级存储

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
0000000	00001	00000	000	00000	0001011

6.2.1.2　DCACHE.CIALL——DCACHE 清除并无效全部表项指令

语法：

dcache.ciall

操作：

将所有 dcache dirty 表项写回到下一级存储后，无效所有表项。

执行权限：

M mode

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
0000000	00011	00000	000	00000	0001011

6.2.1.3　DCACHE.CIPA——DCACHE 清除并无效物理地址匹配表项指令

语法：

dcache.cipa rs1

操作：

将 rs1 中物理地址所属的 dcache 表项写回下级存储并无效该表项。

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：

31 25	24 20	19 15	14 12	11 7	6 0
0000001	01011	rs1	000	00000	0001011

6.2.1.4　DCACHE.CSW——DCACHE 清除 way/set 指向表项指令

语法：

dcache.csw rs1

操作：

按照 rs1 中指定的 way/set 将 dache dirty 表项写回到下一级存储

执行权限：

M mode

异常：

非法指令异常

说明：

E907 dcache 为两 way 组相联，rs1[31] 为 way 编码，rs1[s:6] 为 set 编码。当 dcache 为 32K 时,s 为 13，dcache 为 16KiB 时，s 为 12，依此类推。

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	00001	rs1	000	00000	0001011

6.2.1.5　DCACHE.CISW——DCACHE 清除并无效 way/set 指向表项指令

语法：

dcache.cisw rs1

操作：

按照 rs1 中指定的 way/set 将 dache dirty 表项写回到下一级存储并无效该表项

执行权限：

M mode

异常：

非法指令异常

说明：

E907 dcache 为两 way 组相联，rs1[31] 为 way 编码，rs1[s:6] 为 set 编码。当 dcache 为 32KiB 时,s 为 13，dcache 为 16KiB 时，s 为 12，依此类推。

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	00011	rs1	000	00000	0001011

6.2.1.6　DCACHE.CPA——DCACHE 清除物理地址匹配表项指令

语法：

dcache.cpa rs1

操作：

将 rs1 中物理地址所对应的 dcache 表项写回到下一级存储

执行权限：

M mode

异常：

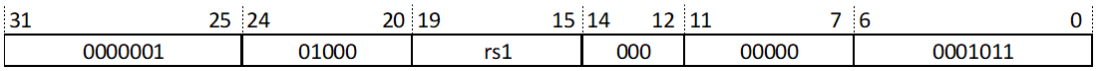
非法指令异常

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	01000	rs1	000	00000	0001011

6.2.1.7　DCACHE.IPA ——DCACHE 无效物理地址匹配表项指令

语法：

dcache.ipa rs1

操作：

将 rs1 中物理地址所对应的 dcache 表项无效

执行权限：

M mode

异常：

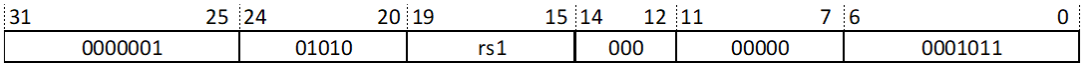
非法指令异常

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	01010	rs1	000	00000	0001011

6.2.1.8　DCACHE.ISW ——DCACHE 无效 way/set 指向表项指令

语法：

dcache.isw rs1

操作：

无效指定 set 和 way 的 dcache 表项

执行权限：

M mode

异常：

非法指令异常

说明：

E907 dcache 为两 way 组相联，rs1[31] 为 way 编码，rs1[s:6] 为 set 编码。当 dcache 为 32K 时,s 为 13，dcache 为 16K 时，s 为 12，依此类推。

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	00010	rs1	000	00000	0001011

6.2.1.9　DCACHE.IALL——DCACHE 无效全部表项指令

语法：

dcache.iall

操作：

无效所有 dcache 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：

31 25	24 20	19 15	14 12	11 7	6 0
0000000	00010	00000	000	00000	0001011

6.2.1.10　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：

31 25	24 20	19 15	14 12	11 7	6 0
0000000	10000	00000	000	00000	0001011

6.2.1.11　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

6.2.2　同步指令术语

同步指令集实现了同步指令的扩展，每条指令位宽为 32 位，以下指令按英文字母顺序排列。

6.2.2.1　SYNC——同步指令

语法：

sync

操作：

该指令保证前序所有指令比该指令早退休，后续所有指令比该指令晚退休

执行权限：

M mode/U mode

异常：

非法指令异常

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
0000000	11000	00000	000	00000	0001011

6.2.2.2　SYNC.I——同步清空指令

语法：

sync.i

操作：

该指令保证前序所有指令比该指令早退休，后续所有指令比该指令晚退休，该指令退休时清空流水线

执行权限：

M mode/U mode

异常：

非法指令异常

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
0000000	11010	00000	000	00000	0001011

6.2.3　算术运算指令术语

算术运算指令针对整型运算进行扩展，每条指令位宽为 32 位，以下指令按英文字母顺序排列。

6.2.3.1　ADDSL——寄存器移位相加指令

语法：

addsl rd rs1, rs2, imm2

操作：

rd ← rs1+ rs2<<imm2

执行权限：

M mode/U mode

异常：

非法指令异常

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00000	imm2	rs2	rs1	001	rd	0001011

6.2.3.2　MULA——乘累加指令

语法：

mula rd, rs1, rs2

操作：

rd ← rd+ (rs1 * rs2)[31:0]

执行权限：

M mode/U mode

异常：

非法指令异常

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00100	00	rs2	rs1	001	rd	0001011

6.2.3.3　MULAH——低 16 位乘累加指令

语法：

mulah rd, rs1, rs2

操作：

rd ← rd+ (rs1[15:0] * rs[15:0])

执行权限：

M mode/U mode

异常：

非法指令异常

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00101	00	rs2	rs1	001	rd	0001011

6.2.3.4　MULS——乘累减指令

语法：

muls rd, rs1, rs2

操作：

rd ← rd- (rs1 * rs2)[31:0]

执行权限：

M mode/U mode

异常：

非法指令异常

指令格式：

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

6.2.3.5　MULSH——低 16 位乘累减指令

语法：

mulsh rd, rs1, rs2

操作：

tmp[31:0] ← rd[31:0]- (rs1[15:0] * rs[15:0])
rd ←sign_extend(tmp[31:0])

执行权限：

M mode/U mode

异常：

非法指令异常

指令格式：

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

6.2.3.6　MVEQZ——寄存器为 0 传送指令

语法：

mveqz rd, rs1, rs2

操作：

if (rs2 == 0)
rd ← rs1

执行权限：

M mode/U mode

异常：

非法指令异常

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
01000	00	rs2	rs1	001	rd	0001011

6.2.3.7　MVNEZ——寄存器非 0 传送指令

语法：

mvnez rd, rs1, rs2

操作：

if (rs2 != 0)
rd ← rs1

执行权限：

M mode/U mode

异常：

非法指令异常

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
01000	01	rs2	rs1	001	rd	0001011

6.2.3.8　SRRI——循环右移指令

语法：

srri rd, rs1, imm5

操作：

rd ← rs1 >>>> imm5
rs1 原值右移，左侧移入右侧移出位

执行权限：

M mode/U mode

异常：

非法指令异常

指令格式：

31 26	25	24 20	19 15	14 12	11 7	6 0
000100	0	imm5	rs1	001	rd	0001011

6.2.4　位操作指令术语

位操作指令针对位运算进行扩展，每条指令位宽为 32 位，以下指令按英文字母顺序排列。

6.2.4.1　EXT——寄存器连续位提取符号位扩展指令

语法：

ext rd, rs1, imm1,imm2

操作：

rd←sign_extend(rs1[imm1:imm2])

执行权限：

M mode/U mode

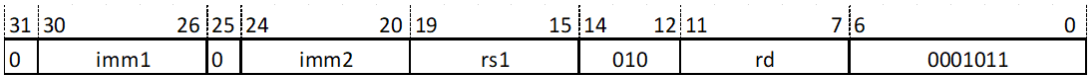
异常：

非法指令异常

说明：

若 imm1<imm2，该指令行为不可预测

指令格式：



6.2.4.2　EXTU——寄存器连续位提取无符号扩展指令

语法：

extu rd, rs1, imm1,imm2

操作：

rd←zero_extend(rs1[imm1:imm2])

执行权限：

M mode/U mode

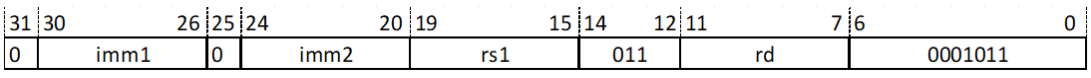
异常：

非法指令异常

说明：

若 imm1<imm2，该指令行为不可预测

指令格式：



6.2.4.3　FF0——快速找 0 指令

语法：

ff0 rd, rs1

操作：

从 rs1 最高位开始查找第一个为 0 的位，结果写回 rd。如果 rs1 的最高位为 0，则结果为 0，如果 rs1 中所有比特位均为 1，结果为 32

执行权限：

M mode/U mode

异常：

非法指令异常

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
10000	10	00000	rs1	001	rd	0001011

6.2.4.4　FF1——快速找 1 指令

语法：

ff1 rd, rs1

操作：

从 rs1 最高位开始查找第一个为 1 的位，结果写回 rd。如果 rs1 的最高位为 1，则结果为 0，如果 rs1 值为 0，指令执行结果为 32，写入 rd。

执行权限：

M mode/U mode

异常：

非法指令异常

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
10000	11	00000	rs1	001	rd	0001011

6.2.4.5　REV——字节倒序指令

语法：

rev rd, rs1

操作：

rd[31:24] ←rs1[7:0]
rd[23:16] ←rs1[15:8]
rd[15:8] ←rs1[23:16]
rd[7:0] ←rs1[31:24]

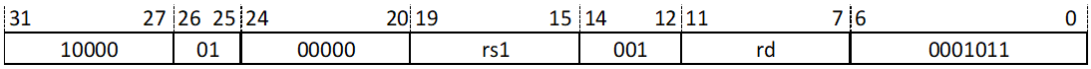
执行权限：

M mode/U mode

异常：

非法指令异常

指令格式：



6.2.4.6　TST——比特为 0 测试指令

语法：

tst rd, rs1, imm5

操作：

if(rs1[imm5] == 1)
　　rd←1
else
　　rd←0

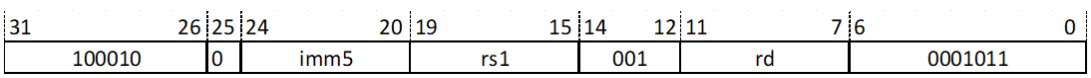
执行权限：

M mode/U mode

异常：

非法指令异常

指令格式：



6.2.4.7　TSTNBZ——字节为 0 测试指令

语法：

tstnbz rd, rs1

操作：

```
for (i =0; i<4;i++)
    if(rs1[8i+7:8i] == 0)
        rd[8i+7:8i] = 8'h ff
    else
        rd[8i+7:8i] = 8'h 0
```

执行权限：

M mode/U mode

异常：

非法指令异常

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
10000	00	00000	rs1	001	rd	0001011

6.2.5　存储指令术语

存储指令实现了对存储操作的扩展定义，每条指令位宽为 32 位，以下指令按英文字母顺序排列。

6.2.5.1　LBIA——字节加载符号位扩展基地址自增指令

语法：

lbia rd, (rs1), imm5,imm2

操作：

rd ←sign_extend(mem[rs1])
rs1←rs1 + sign_extend(imm5 << imm2)

执行权限：

M mode/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、非法指令异常。

说明：

rd 和 rs1 不可相等

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00011	imm2	imm5	rs1	100	rd	0001011

6.2.5.2　LBIB——基地址自增字节加载符号位扩展指令

语法：

lbib rd, (rs1), imm5,imm2

操作：

rs1←rs1 + sign_extend(imm5 << imm2)

rd ←sign_extend(mem[rs1+7:rs1])

执行权限：

M mode/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、非法指令异常。

说明：

rd 和 rs1 不可相等

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00001	imm2	imm5	rs1	100	rd	0001011

6.2.5.3　LBUIA——字节加载无符号扩展基地址自增指令

语法：

lbuia rd, (rs1), imm5,imm2

操作：

rd ←zero_extend(mem[rs1])

rs1←rs1 + sign_extend(imm5 << imm2)

执行权限：

M mode/U mode

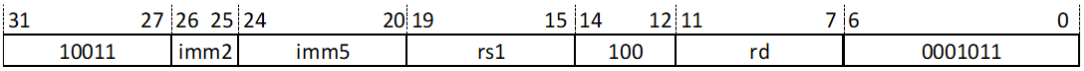
异常：

加载指令非对齐访问异常、加载指令访问错误异常、非法指令异常。

说明：

rd 和 rs1 不可相等

指令格式：



31 27	26 25	24 20	19 15	14 12	11 7	6 0
10011	imm2	imm5	rs1	100	rd	0001011

6.2.5.4　LBUIB——基地址自增字节加载无符号扩展指令

语法：

lbuib rd, (rs1), imm5,imm2

操作：

rs1←rs1 + sign_extend(imm5 << imm2)

rd ←zero_extend(mem[rs1])

执行权限：

M mode/U mode

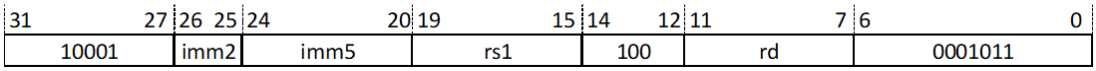
异常：

加载指令非对齐访问异常、加载指令访问错误异常、非法指令异常。

说明：

rd 和 rs1 不可相等

指令格式：



31 27	26 25	24 20	19 15	14 12	11 7	6 0
10001	imm2	imm5	rs1	100	rd	0001011

6.2.5.5　LHIA——符号位扩展半字加载基地址自增指令

语法：

lhia rd, (rs1), imm5,imm2

操作：

rd ←sign_extend(mem[rs1+1:rs1])

rs1←rs1 + sign_extend(imm5 << imm2)

执行权限：

M mode/U mode

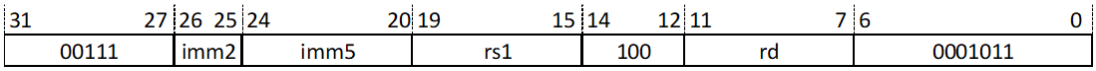
异常：

加载指令非对齐访问异常、加载指令访问错误异常、非法指令异常。

说明：

rd 和 rs1 不可相等

指令格式：



6.2.5.6　LHIB——基地址自增半字加载符号位扩展指令

语法：

lhib rd, (rs1), imm5,imm2

操作：

rs1←rs1 + sign_extend(imm5 << imm2)

rd ←sign_extend(mem[rs1+1:rs1])

执行权限：

M mode/U mode

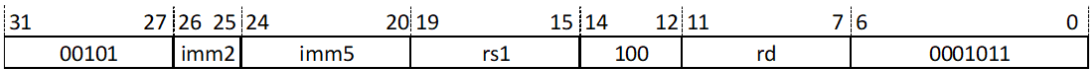
异常：

加载指令非对齐访问异常、加载指令访问错误异常、非法指令异常。

说明：

rd 和 rs1 不可相等

指令格式：



6.2.5.7　LHUIA——半字加载无符号扩展基地址自增指令

语法：

lhuia rd, (rs1), imm5,imm2

操作：

rd ←zero_extend(mem[rs1+1:rs1])

rs1←rs1 + sign_extend(imm5 << imm2)

执行权限：

M mode/U mode

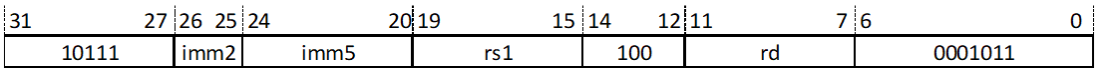
异常：

加载指令非对齐访问异常、加载指令访问错误异常、非法指令异常。

说明：

rd 和 rs1 不可相等

指令格式：



6.2.5.8　LHUIB——基地址自增半字加载无符号扩展指令

语法：

lhuib rd, (rs1), imm5,imm2

操作：

rs1←rs1 + sign_extend(imm5 << imm2)

rd ←zero_extend(mem[rs1+1:rs1])

执行权限：

M mode/U mode

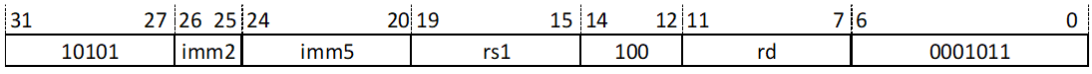
异常：

加载指令非对齐访问异常、加载指令访问错误异常、非法指令异常。

说明：

rd 和 rs1 不可相等

指令格式：



6.2.5.9　LRB——寄存器移位字节加载符号位扩展指令

语法：

lrb rd, rs1, rs2, imm2

操作：

rd ←sign_extend(mem[(rs1+rs2<<imm2)])

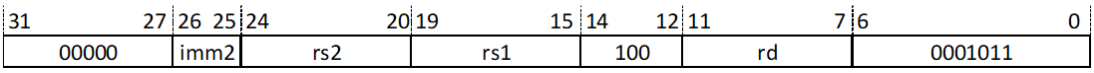
执行权限：

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、非法指令异常

指令格式：



6.2.5.10　LRBU——寄存器移位字节加载无符号扩展指令

语法：

lrbu rd, rs1, rs2, imm2

操作：

rd ←zero_extend(mem[(rs1+rs2<<imm2)])

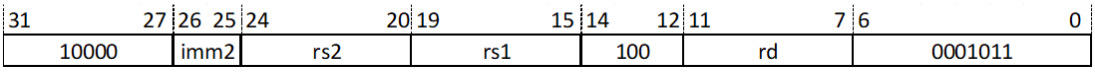
执行权限：

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、非法指令异常

指令格式：



6.2.5.11　LRH——寄存器移位半字加载符号位扩展半字加载指令

语法：

lrh rd, rs1, rs2, imm2

操作：

rd ←sign_extend(mem[(rs1+rs2<<imm2)+1: (rs1+rs2<<imm2)])

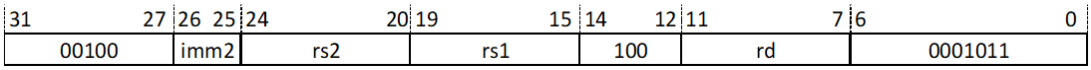
执行权限：

M mode/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、非法指令异常

指令格式：



6.2.5.12　LRHU——寄存器移位零扩展扩展半字加载无符号扩展指令

语法：

lrhu rd, rs1, rs2, imm2

操作：

rd ←zero_extend(mem[(rs1+rs2<<imm2)+1: (rs1+rs2<<imm2)])

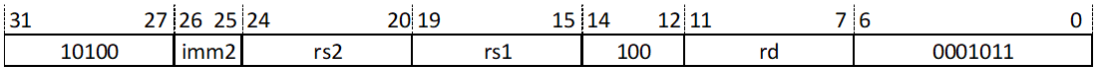
执行权限：

M mode/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、非法指令异常

指令格式：



6.2.5.13　LRW——寄存器移位字加载指令

语法：

lrw rd, rs1, rs2, imm2

操作：

rd ←(mem[(rs1+rs2<<imm2)+3: (rs1+rs2<<imm2)]

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、非法指令异常

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
01000	imm2	rs2	rs1	100	rd	0001011

6.2.5.14　LWIA——字加载基地址自增指令

语法：

lwia rd, (rs1), imm5,imm2

操作：

rd ←(mem[rs1+3:rs1])

rs1←rs1 + sign_extend(imm5 << imm2)

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、非法指令异常。

说明：

rd 和 rs1 不可相等

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
01011	imm2	imm5	rs1	100	rd	0001011

6.2.5.15　LWIB——基地址自增字加载指令

语法：

lwib rd, (rs1), imm5,imm2

操作：

rs1←rs1 + sign_extend(imm5 << imm2)

rd ←(mem[rs1+3:rs1])

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、加载指令页面错误异常、非法指令异常。

说明：

rd 和 rs1 不可相等

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
01001	imm2	imm5	rs1	100	rd	0001011

6.2.5.16　SBIA——字节存储基地址自增指令

语法：

sbia rs2, (rs1), imm5,imm2

操作：

mem[rs1]←rs2[7:0]

rs1←rs1 + sign_extend(imm5 << imm2)

执行权限：

M mode/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、非法指令异常

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00011	imm2	imm5	rs1	101	rs2	0001011

6.2.5.17　SBIB——基地址自增字节存储指令

语法：

sbib rs2, (rs1), imm5,imm2

操作：

rs1←rs1 + sign_extend(imm5 << imm2)

mem[rs1] ←rs2[7:0]

执行权限：

M mode/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、非法指令异常

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00001	imm2	imm5	rs1	101	rs2	0001011

6.2.5.18　SHIA——半字存储基地址自增指令

语法：

shia rs2, (rs1), imm5,imm2

操作：

mem[rs1+1:rs1]←rs2[15:0]

rs1←rs1 + sign_extend(imm5 << imm2)

执行权限：

M mode/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、非法指令异常

指令格式：

31 27	26 25	24 20	19 15	14 12	11 7	6 0
00111	imm2	imm5	rs1	101	rs2	0001011

6.2.5.19　SHIB——基地址自增半字存储指令

语法：

shib rs2, (rs1), imm5,imm2

操作：

rs1←rs1 + sign_extend(imm5 << imm2)
mem[rs1+1:rs1] ←rs2[15:0]

执行权限：

M mode/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、非法指令异常

指令格式：

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

6.2.5.20　SRB——寄存器移位字节存储指令

语法：

srb rd, rs1, rs2, imm2

操作：

mem[(rs1+rs2<<imm2)] ←rd[7:0]

执行权限：

M mode/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、非法指令异常

指令格式：

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

6.2.5.21　SRH——寄存器移位半字存储指令

语法：

srh rd, rs1, rs2, imm2

操作：

mem[(rs1+rs2<<imm2)+1: (rs1+rs2<<imm2)] ←rd[15:0]

执行权限：

M mode/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、非法指令异常

指令格式：

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

6.2.5.22　SRW——寄存器移位字存储指令

语法：

srw rd, rs1, rs2, imm2

操作：

mem[(rs1+rs2<<imm2)+3: (rs1+rs2<<imm2)] ←rd[31:0]

执行权限：

M mode/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、非法指令异常

指令格式：

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

6.2.5.23　SWIA——字存储基地址自增指令

语法：

swia rs2, (rs1), imm5,imm2

操作：

mem[rs1+3:rs1]←rs2
rs1←rs1 + sign_extend(imm5 << imm2)

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、非法指令异常

指令格式：

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

6.2.5.24　SWIB——基地址自增字存储指令

语法：

swib rs2, (rs1), imm5,imm2

操作：

rs1←rs1 + sign_extend(imm5 << imm2)
mem[rs1+3:rs1] ←rs2

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、非法指令异常

指令格式：

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

6.2.6　双精度浮点高位数据传输指令术语

本节指令对双精度浮点和整型间的数据传送操作进行扩展定义，每条指令位宽为 32 位，以下指令按英文字母顺序排列。

6.2.6.1　FMV.X.HW——双精度浮点高位读传输指令

语法：

fmv.x.hw rd, fs1

操作：

rd←fs1[63:32]

执行权限：

M mode/U mode

异常：

非法指令异常

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
1010000	00000	rs1	001	rd	0001011

6.2.6.2　FMV.HW.X——双精度浮点高位写传输指令

语法：

fmv.x.hw rd, fs1

操作：

fs1[63:32]←rd

执行权限：

M mode/U mode

异常：

非法指令异常

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
1100000	00000	rs1	001	rd	0001011

6.2.7　中断加速指令术语

本节对中断加速响应的场景进行指令扩展定义，每条指令位宽为 32 位，以下指令按英文字母顺序排列。

6.2.7.1　IPUSH——中断加速压栈指令

语法：

ipush

操作：

mem[int_sp-4]~mem[int_sp-72] ←{mcause,mepc,Xn};

int_sp=int_sp-72

Xn 依次为 X1，X5-X7，X10-X17，X28-X31

执行权限：

M mode

异常：

内存存储非对齐访问异常、内存存储访问错误异常、非法指令异常

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
0000000	00100	00000	000	00000	0001011

6.2.7.2　IPOP——中断加速弹栈指令

语法：

ipop

操作：

{mcause,mepc,Xn}←mem[int_sp+68]~mem[int_sp];int_sp=int_sp+72;mret

Xn 依次为 X1，X5-X7，X10-X17，X28-X31

执行权限：

M mode

异常：

内存加载非对齐访问异常、内存加载访问错误异常、非法指令异常

指令格式：

31 25	24 20	19 15	14 12	11 7	6 0
0000000	00101	00000	000	00000	0001011

6.3 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。

6.3.1 机器模式信息寄存器组

6.3.1.1 供应商编号寄存器（MVENDORID）

机器模式厂商编号寄存器（MVENDORID）存储了 JEDEC 分配给处理器厂商的编号，平头哥半导体对应编号为

机器模式供应商编号寄存器（MVENDORID）存储了平头哥半导体有限公司的厂商编号信息，E907 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。

6.3.1.2 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，E907 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。

6.3.1.3 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。E907 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。

6.3.1.4 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。E907 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。

6.3.2 机器模式异常设置寄存器组

6.3.2.1 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。

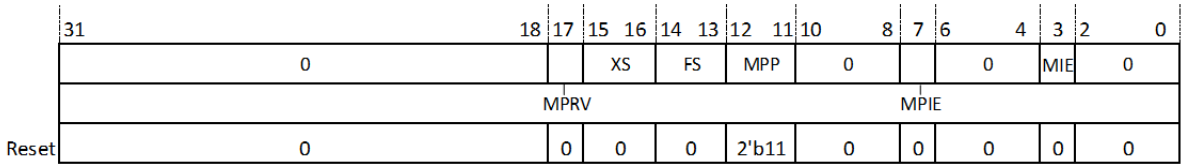


图 6.1: 机器模式处理器状态寄存器（MSTATUS）

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。

FS-浮点寄存器状态位：

该域仅在配置硬件浮点时存在，用于指示浮点控制寄存器以及浮点通用寄存器的状态。

2’b00：无法访问浮点控制寄存器和浮点通用寄存器，访问这类寄存器会触发非法指令异常；

2‘b01：浮点控制寄存器和浮点通用寄存器处于初始化状态；

2‘b10：浮点控制寄存器和浮点通用寄存器是干净的；

2’b11：部分浮点控制寄存器或者浮点通用寄存器被写脏过；

在系统初始化时可将该域设置为 2’b01 或者 2’b10，硬件在程序运行中会维护 FS 域。在异常或中断服务程序入口如果判断 FS 非 2’b11 的话，表明响应中断前没有将浮点相关寄存器现场写脏，因此异常或者中断服务程序的入口也无需对这些现场进行保存。

该域复位值为 2’b00。

XS-用户模式浮点扩展状态位：

该域仅在配置硬件浮点时有意义，E907 将其硬件绑为 0。

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。

6.3.2.2　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。

E907 支持的指令集架构为 RV32IMAC, 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 20190608-Priv-MSU-Ratified)

E907 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。

6.3.2.3　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。

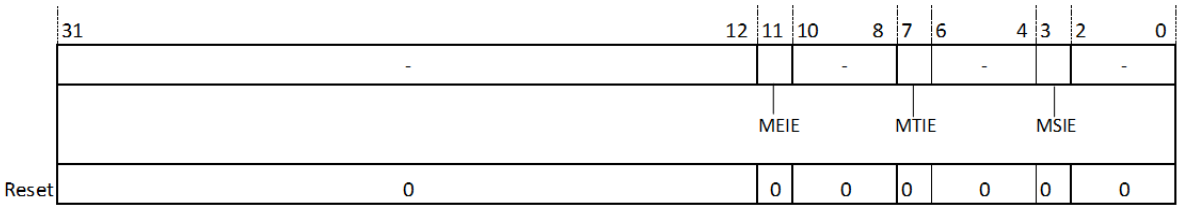


图 6.2: 机器模式中断使能控制寄存器（MIE）

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。E907 固定配置 CLIC 模式。

6.3.2.4　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。

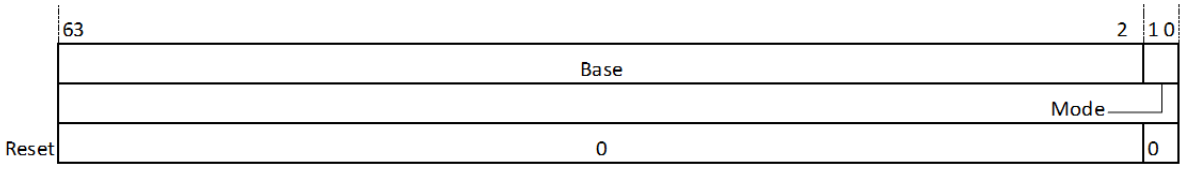


图 6.3: 机器模式异常向量基址寄存器（MTVEC）

BASE-向量基址位：

向量基址位指示了异常服务程序入口地址的高 30 位，将此基址低位拼接 2’b00 即可得到异常服务程序入口地址。

该位复位值为零。

MODE-向量入口模式位：

当 MODE[1:0] 是 2’b00 时，异常和中断都统一使用 BASE 地址作为异常入口地址；

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

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

当 MODE[1:0] 是 2’b10 时，保留。

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

E907 中 MODE[1:0] 硬件固定设置为 3，软件不可设置。

6.3.2.5　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。

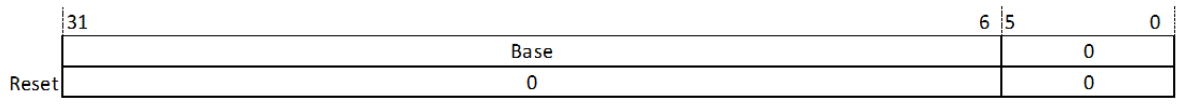


图 6.4: 机器模式矢量中断基址寄存器（MTVT）

BASE-向量基址位：

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

该基址域复位值为零。

6.3.3 机器模式异常处理寄存器组

6.3.3.1 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。

6.3.3.2 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。

6.3.3.3 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。

6.3.3.4 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，E907 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。

6.3.3.5　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）。E907 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。

6.3.3.6　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。

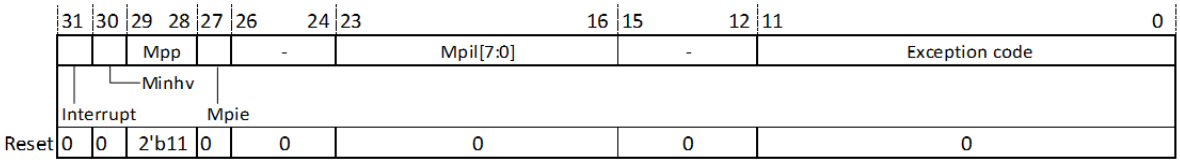


图 6.5: 机器模式异常事件向量寄存器（MCAUSE）

Interrupt-中断标记位：

当 Interrupt 位为 0 时，表示触发异常的来源不是中断，Exception Code 按照普通异常规则解析；

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

该位复位值为零。

MINHV-矢量中断跳转指示位

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

该位为 MSTATUS.MPIE 的镜像。

MPIL-机器模式保留中断优先级位

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。

该位复位值为零。

6.3.3.7　机器模式等待中断向量地址和中断使能寄存器（MNXTI）

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。

6.3.3.8　机器模式中断状态寄存器（MINTSTATUS）

	31 — 24	23 — 0
	Mil[7:0]	-
Reset	0	0

图 6.6: 机器模式中断状态寄存器（MINTSTATUS）

该寄存器在处理器配置 CLIC 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。

6.3.3.9 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。

6.3.3.10 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。

E907 实现了 CLIC 模式，因此该寄存器值为零，软件不可写。

6.3.4 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* 。

6.3.5 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。

6.3.5.1 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。

6.3.5.2 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。

6.3.6 机器模式扩展寄存器组

6.3.6.1 扩展状态寄存器（MXSTATUS）

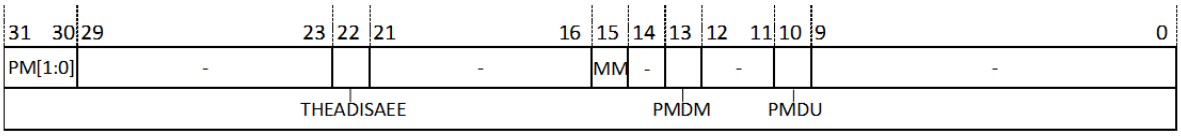


图 6.7: 扩展状态寄存器（MXSTATUS）

PMDU-用户模式事件监测计数器开关位

1 ‘b0: 用户模式下事件监测计数器可以正常计数

1’ b1: 用户模式下事件监测计数器禁止计数

该域复位值为 1’ b0，机器模式读写。

PMDM-机器模式事件检测计数器开关位

1’ b0: 机器模式下事件监测计数器可以正常计数

1’ b1: 机器模式下事件监测计数器禁止计数

该域复位值为 1 ‘b0，机器模式读写。

MM-非对齐访问控制位

1’ b1：当内存访问出现地址非对齐时，不会上报非对齐访问异常，硬件会对非对齐访问请求进行拆分处理；

1’ b0：当内存访问出现地址非对齐时，上报非对齐访问异常；

该域复位值为 1 ‘b1，机器模式读写。

THEADISAEE- THEAD 扩展指令集使能位

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 1’ b0，机器模式读写。

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。
该域复位值为 2’b11，机器模式读写。

6.3.6.2　硬件配置寄存器（MHCR）

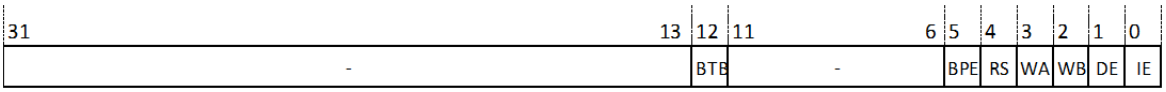


图 6.8: 硬件配置寄存器（MHCR）

BTB-分支目标预测使能位:

当 BTB 值为 0 时，分支目标预测关闭；当 BTB 值为 1 时，分支目标预测开启。
该域复位值为 1’b0，机器模式读写。

BPE-允许预测跳转设置位:

当 BPE 为 0 时，预测跳转关闭；当 BPE 为 1 时，预测跳转开启。
该域复位值为 1’b0，机器模式读写。

RS-地址返回栈设置位:

当 RS 为 0 时，返回栈关闭；当 RS 为 1 时，返回栈开启。
该域复位值为 1’b0，机器模式读写。

WA-高速缓存写分配有效设置:

该位表示处理器执行 Store 指令且数据 cache 缺失时，是否在数据 cache 中分配缺失的缓存行。当 WA 为 0 时，数据 cache 为 write non-allocate 模式，即在数据 cache 缺失时不会将缺失的数据缓存行从内存加载到数据 cache 内；当 WA 为 1 时，数据 cache 为 write allocate 模式，即在 Store 指令导致的数据 cache 缺失时会将该缺失的数据缓存行从内存中加载到数据 cache 中。
该域复位值为 1’b0，机器模式读写。

WB-高速缓存写回设置位:

当 WB 为 0 时，数据 cache 为写直模式；当 WB 为 1 时，数据 cache 为写回模式。
该域复位值为 1’b0，机器模式读写。

DE-数据高速缓存设置位:

当 DE 为 0 时，数据 cache 关闭；当 DE 为 1 时，数据 cache 开启。
该域复位值为 1’b0，机器模式读写。

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。
该域复位值为 1’b0，机器模式读写。

6.3.6.3　隐式操作寄存器（MHINT）

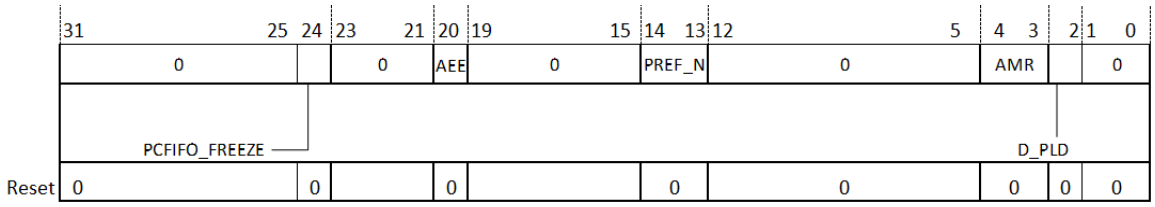


图 6.9: 隐式操作寄存器（MHINT）

PCFIFO_FREEZE PCFIFO 冻结使能位：

当 PCFIFO_FREEZE 为 1 时，调试的 PCFIFO 将会冻结，不会再继续更新，;

当 PCFIFO_FREEZE 为 0 时，调试的 PCFIFO 正常更新。

AEE-精确异常使能位：

当 AEE 为 1 时，处理器处于精确异常工作模式。由于 Load/Store 指令访问总线产生的总线访问错误异常信号返回延迟不可预期，处理器在执行 Load/Store 指令时会堵塞流水线，后续指令不会下发被执行。直到异常信号返回处理器，进入异常服务程序，MEPC 将保存这条产生总线访问异常的 Load/Store 指令的 PC，从而实现精确异常。

当 AEE 为 0 时，处理器处于非精确异常工作模式。处理器在执行 Load/Store 指令时不会堵塞流水线，若后续指令不为 Load/Store 指令（结构竞争）且与该 Load/Store 指令无数据相关性时，可以继续执行。当总线访问错误异常信号传送至处理器，此时进入异常服务程序时，MEPC 将保存流水线 EX 级中正在被执行的指令的 PC，不一定为该出现内存访问错误异常的 Load/Store 指令的 PC。

需要注意的是，即便 AEE 配置为 0，执行 Load/Store 指令因 PMP 权限不匹配导致的内存访问错误异常是精确的。另外，非精确异常工作模式下，MTVAL 寄存器的更新值是精确的，即产生内存访问错误异常的地址被精确保存了下来。

该域复位值为 1’b0，机器模式读写。

PREF_N-在 DCACHE 预取开启时预取缓存行数量：

2’b00: 预取 2 条缓存行。

2’b01: 预取 4 条缓存行。

2’b10: 预取 8 条缓存行。

2’b11: 预取 16 条缓存行。

该域复位值为 2’b0。

AMR-DCACHE 写分配策略自动调整使能位：

当 AMR 为 0 时，写分配策略由 MHCR 寄存器中 WA 域决定。

当 AMR 为 1 时，在出现连续 3 条缓存行的存储操作时后续连续地址的存储操作不再写入 DCACHE。

当 AMR 为 2 时，在出现连续 64 条缓存行的存储操作时后续连续地址的存储操作不再写入 DCACHE。

当 AMR 为 3 时，在出现连续 128 条缓存行的存储操作时后续连续地址的存储操作不再写入 DCACHE。

该域复位值为 1’b0。

D_PLD-DCACHE 预取使能位：

当 D_PLD 为 0 时，CACHE 预取关闭；

当 D_PLD 为 1 时，DCACHE 预取开启。

该域复位值为 1’ b0。

6.3.6.4　扩展异常状态寄存器（MEXSTATUS）

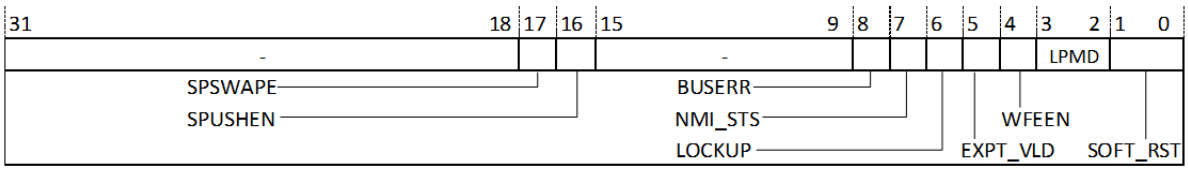


图 6.10: 扩展异常状态寄存器（MEXSTATUS）

RSTMD-软件复位模式：

向 RSTMD 写 2’ b00 时，表示不需要进行软复位；向 RSTMD 写 2’ b01 时，表示需要复位核；向 RSTMD 写 2’ b10 时，表示需要复位整个系统；向 RSTMD 写 2’ b11 为 reserved。

LPMD-lowpower 模式选择：

当 LPMD 为 2’ b00 时，下次使用 WFI 指令进入深睡眠；当 LPMD 为 2’ b01 时，下次使用 WFI 指令进入浅睡眠。复位值为 2’ b00。

WFEEN-wait for event 模式使能：

当 WFEEN 为 1 时，执行 WFI 指令实际为 WFE 功能，event 可以唤醒，中断不需要考虑优先级即可唤醒；当 WFEEN 为 0 时，WFI 指令仅能由中断进行唤醒，而且需要考虑中断优先级。复位值为 1。

EXPT-异常有效：

当 EXPT 为 1 时，表示处理器进入异常处理状态。

LOCKUP-lockup 有效：

当 LOCKUP 为 1 时，表示处理器进入锁定状态。

NMI-NMI 有效：

当 NMI 为 1 时，表示处理器进入 NMI 处理状态。

BUSERR-BUSERR 异常：

当 BUSERR 为 1 时，表示处理器上一次发生的异常为 BUS ERR 异常。

SPUSHEN-中断自动压栈开关位：

该位为 0 时，不使能硬件自动压栈机制；该位为 1 时，使能硬件自动压栈机制，响应中断时，硬件投机执行一条 IPUSH 指令。

SPSWAPEN-自动中断栈切换位：

该位为 0 时，不使能自动中断栈切换机制；该位为 1 时，使能自动中断栈切换机制，当响应第 0 层中断时，自动将 mscratchcswl 寄存器的值写入 sp 寄存器。第 0 层中断判断条件：当前 MIL 为 0。

6.3.6.5　复位地址指示寄存器（MRADDR）

该寄存器用于指示 CPU 的复位启动地址，CPU 复位启动地址由系统集成时指定，该寄存器仅用于软件查询，机器模式只读。

6.3.6.6　NMI 状态寄存器 (MNMICAUSE)

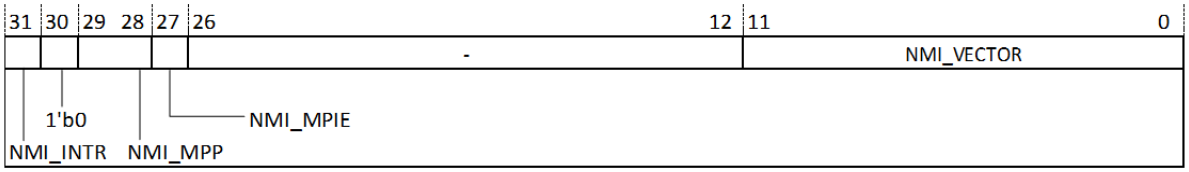


图 6.11: NMI 状态寄存器 (MNMICAUSE)

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

6.3.6.7　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。

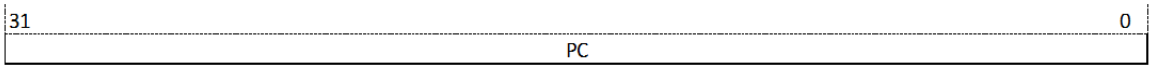


图 6.12: NMI 异常程序计数器（MNMIPC）

6.3.6.8　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，通过连续读取 MCPUID 寄存器可以获取 3 个不同的返回值用于表征产品信息。具体定义如下图所示。

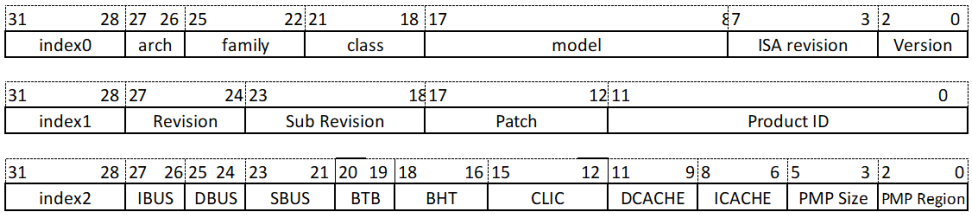


图 6.13: 处理器型号寄存器（MCPUID）

6.4　用户模式控制寄存器

用户模式控制寄存器包括浮点控制寄存器和事件监测寄存器。

6.4.1　用户模式浮点寄存器组

6.4.1.1　浮点异常累积状态寄存器（FFLAGS）

浮点异常累积状态寄存器（FFLAGS）是浮点控制状态寄存器（FCSR）的异常累积域映射，用户模式只读。

6.4.1.2　浮点动态舍入模式寄存器（FRM）

浮点动态舍入寄存器（FRM）是浮点控制状态寄存器（FCSR）的舍入模式域映射，用户模式可以正常读写。

6.4.1.3　浮点控制状态寄存器（FCSR）

浮点控制状态寄存器（FCSR）用于记录浮点的异常累积和舍入模式控制。

该寄存器的位宽是 32 位，该寄存器任何模式都可以读写。

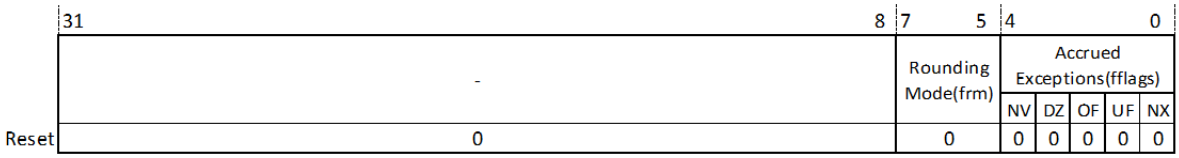


图 6.14: 浮点控制状态寄存器（FCSR）

NX-非精确异常：

当 NX=0 时，没有产生非精确异常。

当 NX=1 时，产生非精确异常。

注意这里的非精确异常指 IEEE754 标准定义的异常，而非 MHINT 寄存器 AEE 位控制使能的非精确异常。

UF-下溢异常：

当 UF=0 时，没有产生下溢异常。

当 UF=1 时，产生下溢异常。

OF-上溢异常：

当 OF=0 时，没有产生上溢异常。

当 OF=1 时，产生上溢异常。

DZ-除 0 异常：

当 DZ=0 时，没有产生除 0 异常。

当 DZ=1 时，产生除 0 异常。

NV-无效操作数异常：

当 NV=0 时，没有产生无效操作数异常。

当 NV=1 时，产生无效操作数异常。

RM-舍入模式：

当 RM=0 时，RNE 舍入模式，向最近偶数舍入。

当 RM=1 时，RTZ 舍入模式，向 0 舍入。

当 RM=2 时，RDN 舍入模式，向负无穷舍入。

当 RM=3 时，RUP 舍入模式，向正无穷舍入。

当 RM=4 时，RMM 舍入模式，向最近舍入。

6.4.2　用户模式事件计数寄存器组

用户模式事件计数寄存器为机器模式事件计数寄存器的只读映射，详细描述可参考 11.3 节事件计数器。

6.4.3　用户模式浮点扩展状态寄存器组

6.4.3.1　用户模式浮点扩展控制寄存器（FXCR）

用户模式浮点扩展控制寄存器（FXCR）用于浮点扩展功能开关和浮点异常累积位，用户模式可读写。

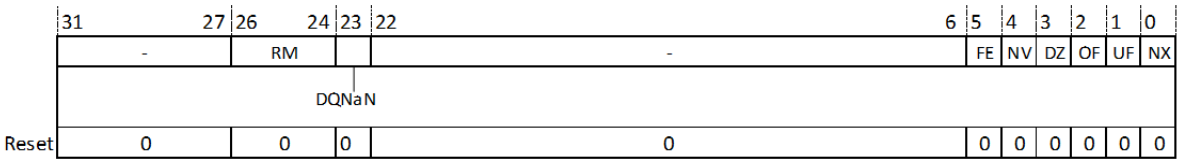


图 6.15: 用户模式浮点扩展控制寄存器（FXCR）

NX-非精确异常：

FCSR 对应位的映射。

UF-下溢异常：

FCSR 对应位的映射。

OF-上溢异常：

FCSR 对应位的映射。

DZ-除 0 异常：

FCSR 对应位的映射。

NV-无效操作数异常：

FCSR 对应位的映射。

FE-浮点异常累积位：

当有任何一个浮点异常发生时，该位将被置为 1。

DQNaN-输出 QNaN 模式位：

当 DQNaN 为 0 时，计算输出的 QNaN 值为 RISC-V 规定的固定值，即 0x7FC0_0000。

当 DQNaN 为 1 时，计算输出的 QNaN 值根 IEEE754 标准一致。

RM-舍入模式：

FCSR 对应位的映射。

对 NX/UF/OF/DZ/NV/RM 位域的写操作会同时体现在 FCSR 寄存器上。

6.4.4　定点运算饱和状态寄存器

定点运算饱和状态寄存器是标准 RISC-V P 指令集扩展下定义的寄存器，用于保存饱和运算的状态。当 bit[0] 为 1 时表明 P 指令运算中发生过结果需要饱和运算。该寄存器用户模式可读可写。

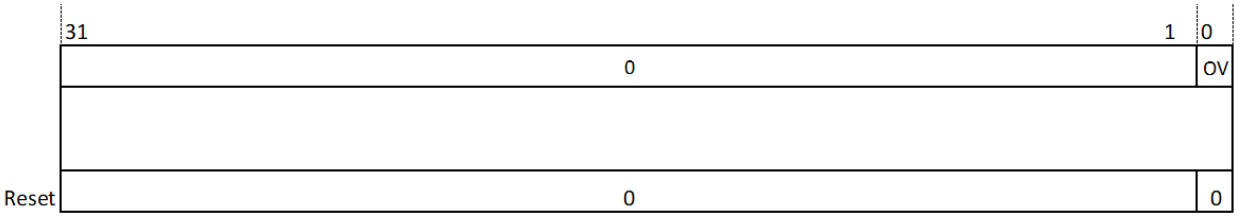


图 6.16: 定点运算饱和状态寄存器（VXSAT）

6.5　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。

6.5.1 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，详见 表 4.2 。

```
/* 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，详见*PMP 控制寄存器* 。

```
/* 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，详见 图 4.2 和 图 4.3 。

```
/* 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
```

6.5.2 高速缓存设置示例

E907 支持高速缓存扩展寄存器：机器模式硬件配置寄存器 (mhcr)。可以实现对指令和数据高速缓存的开关以及写分配和写回模式的配置。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, mxstatus, t1

//invalid icache

ICACHE.IALL

//当 DE 为 0 时，数据高速缓存关闭；当 DE 为 1 时，数据高速缓存开启。

li t1,  2 //设置 dcache enable 打开，数据可高速缓冲
csrrs t3, mhcr, t1

//使用 cache 指令使 dcache 全部无效化
//invalid dcache

DCACHE.IALL

//另外，cache 在 reset 之后会自动 invalid
//当 WA 为 0 时，数据高速缓存为 write non-allocate 模式；当 WA 为 1 时，数据高速缓存为 write allocate 模式。

li t1, 8 //设置高速缓存写分配有效
csrrs t3, mhcr, t1

//当 WB 为 0 时，数据高速缓存为写直模式；当 WB 为 1 时，数据高速缓存为写回模式。

li, t1, 4 //设置高速缓存写回模式
csrrs t3, mhcr, t1
```

6.5.3 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：

```
//打开全局中断使能位 mstatus.mie

li t1, 0x8
csrrs x0, mstatus, t1

//如有需求打开局部中断使能位，例如 clicintie

la t0, 0xe0801000
li t1, 0x100
sw t1, 0x0(t0)
```

6.5.4 通用寄存器初始化示例

```
//初始化通用寄存器 x0~x31。

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
li x16, 0
li x17, 0
li x18, 0
li x19, 0
li x20, 0
li x21, 0
li x22, 0
li x23, 0
li x24, 0
li x25, 0
li x26, 0
li x27, 0
li x28, 0
```

(下页继续)

(续上页)

```
li x29, 0
li x30, 0
li x31, 0
```

6.5.5 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
```

6.5.6 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，E907 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)
```

(下页继续)

(续上页)

```
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)
li t1, 0xfff
and t1, t0, t1 //获取 cause number
srli t0, t0, 0x1b
andi t0, t0, 0x10 //得到 mcause 的 bit[31]，用于判断是否为中断
add t0, t0, t1 //cause+16，空出低 16 个异常向量的空间
slli t0, t0, 0x2
la t1, vector_table //存放异常和中断服务程序的入口地址
add t0, t0, t1
lw t1, 0(t0)
//handle with the exception or non vector int
jalr t1
//recovery the cpu field
lw t2, 4(sp)
csrw mstatus, t2
lw t1, 8(sp)
csrw mepc, t1
lw t0, 12(sp)
csrw mcause, t0
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
……
```

(下页继续)

(续上页)

```
.long MACHINE_ECALL_HANDLER //11 号机器模式环境调用异常处理函数
.rept 4
.long 0x0
.endr
.rept 3
.long 0x0 //0-2 号 clint 中断未实现
.endr
.long MSOFT_INT_HANDLER //3 号机器模式软件中断处理函数
.rept 3
.long 0x0 //4-6 号 clint 中断未实现
.endr
.long MTIME_INT_HANDLER //7 号机器模式计时器中断处理函数
.rept 3
.long 0x0 //8-10 号 clint 中断未实现
.endr
.long 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 16 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。

6.5.7 浮点单元初始化设置示例

```
//设置浮点控制寄存器和浮点通用寄存器的使用状态为 initial

la a0, 0x2000
csrs mstatus, a0
```


6.5.8 性能监测单元设置示例

```
/*1.inhibit counters counting*/
li a0 0xffffffff
csrw mcountinhibit, a0

/*2.you can initial pmu counters manually if necessarily*/
csrw mcycle, x0
csrw minstret, x0
csrw mhpmcounter3, x0
……
csrw mhpmcounter17, x0

/*3.configure mhpmevent*/
li a0, 0x1
csrw mhpmevent3, a0 // mhpmcounter3 count event: L1 ICache Access Counter
li a0, 0x2
csrw mhpmevent4, a0 // mhpmcounter4 count event: L1 ICache Miss Counter
……

/*4. configure mcounteren*/
li a0, 0xffffffff
csrw mcounteren, a0 // enable user mode to read hpmcounter

/*5. enable counters to count when you want*/

csrw mcountinhibit, x0
```

6.5.9 AMO 原子锁操作示例

```
//本示例给出了一种使用 AMO 指令实现原子锁操作的方法，供参考。
//令 t1=<lock_addr>，t2=0x1. <lock_addr> 的值为 0 表示空闲，为 1 表示被占用。

try_lock:
    lw t0, (t1)
    bnez t0, try_lock
    amoswap.w t0, t2, (t1)
    bnez t0, try_lock
// critical section starts
    ...
    ...
    ...
// critical section ends
```

(下页继续)

(续上页)

```
release_lock:
    sw zero, (t1)
```

