芯来科技
NUCLEI

Bumblebee 处理器

内核指令架构手册

www.riscv-mcu.com

修订历史

版本号	修订日期	修订的章节	修订的内容
1.0	2019/6/21	N/A	1. 初始版本

目录

表格清单

图片清单

1. Bumblebee 内核指令集与 CSR 介绍

Bumblebee 处理器内核（Processor Core），简称 Bumblebee 内核，是由芯来科技（Nuclei System Technology）联合兆易创新（Gigadevice）针对其面向 IoT 或其他超低功耗场景的通用 MCU 产品定制的一款商用 RISC-V 处理器内核，专用于型号为 GD32VF103 的 MCU 产品。

有关 Bumblebee 内核的硬件特性介绍，请参见《Bumblebee 内核简明数据手册》，本文主要就其支持的指令架构进行详细介绍。

注意：针对该 MCU 所使用的 Bumblebee 内核为芯来科技（Nuclei System Technology）与台湾晶心科技（Andes Technology）联合开发，由芯来科技（Nuclei System Technology）提供授权以及技术支持等服务。

目前芯来科技（Nuclei System Technology）可授权完全国产自主可控的 N200 系列超低功耗商用处理器内核 IP，以及多个系列（300/600/900 系列）的 32 位架构和 64 位架构高性能嵌入式处理器内核 IP，并为客户提供处理器 IP 定制化服务。

注意：欲了解有关 RISC-V MCU 芯片、开发板以及解决方案的更多综合信息，请访问 www.riscv-mcu.com

1.1. RISC-V 指令集介绍

Bumblebee 内核遵循的标准 RISC-V 指令集文档版本为：“指令集文档版本 2.2”（riscv-spec-v2.2.pdf）。用户可以在 RISC-V 基金会的网站上需注册便可关注并免费下载其完整原文（https://riscv.org/specifications/）。

除了 RISC-V “指令集文档版本 2.2”英文原文之外，用户还可以参阅中文书籍《手把手教你设计 CPU——RISC-V 处理器篇》的附录 A、附录 C~G 部分，其使用通俗易懂的中文对 RISC-V 指令集标准进行了系统讲解。

1.2. Bumblebee 内核支持指令集

RISC-V 指令集基于模块化设计，可以根据配置进行灵活组合。Bumblebee 内核支持的是如下模块化指令集：

- RV32 架构：32 位地址空间，通用寄存器宽度 32 位。
- I：支持 32 个通用整数寄存器。
- M：支持整数乘法与除法指令
- C：支持编码长度为 16 位的压缩指令，提高代码密度。
- A：支持原子操作指令。

按照 RISC-V 架构命名规则，以上指令子集的组合可表示为 RV32IMAC。

1.3. CSR 寄存器

RISC-V 的架构中定义了一些控制和状态寄存器（Control and Status Register，CSR），用于配置或记录一些处理器核的运行状态。CSR 寄存器是处理器核内部的寄存器，使用其专有的 12 位地址编码空间。详情请参见第 7 章。

2. Bumblebee 内核特权架构介绍

2.1. 总体介绍

Bumblebee 内核遵循的标准 RISC-V 特权架构文档版本为：“特权架构文档版本 1.10”（riscv-privileged-v1.10.pdf）。用户可以在 RISC-V 基金会的网站上需注册便可关注并免费下载其完整原文（https://riscv.org/specifications/）。

除了 RISC-V “特权架构文档版本 1.10”英文原文之外，用户还可以参阅中文书籍《手把手教你设计 CPU——RISC-V 处理器篇》的附录 A、附录 C~G 部分，其使用通俗易懂的中文对 RISC-V 特权架构标准进行了系统讲解。

2.2. 特权模式（Privilege Modes）

Bumblebee 内核支持两个特权模式（Privilege Modes）：

- 机器模式（Machine Mode）是必须的模式，该 Privilege Mode 的编码是 0x3。
- 用户模式（User Mode）是可配置的模式，该 Privilege Mode 的编码是 0x0。

2.2.1. 机器模式（Machine Mode）

Bumblebee 内核有关 Machine Mode 的关键要点如下：

- 处理器内核被复位后，默认处于 Machine Mode。
- 在 Machine Mode 下，程序能够访问所有的 CSR 寄存器。

2.2.2. 用户模式（User Mode）

Bumblebee 内核有关 User Mode 的关键要点如下：

- 在 User Mode 下只能够访问 User Mode 限定的 CSR 寄存器，请参见第 7.3 节了解详情。

2.2.3. 机器子模式（Machine Sub-Mode）

Bumblebee 内核的 Machine Mode 可能处于四种不同的状态下，将之称之为机器子模式（Machine Sub-Mode）：

- 正常机器模式（该 Machine Sub-Mode 的编码是 0x0）：
  - 处理器内核被复位之后，处于此子模式之下。处理器复位后如果不产生异常、NMI、中断，则一直正常运行于此模式之下。
- 异常处理模式（该 Machine Sub-Mode 的编码是 0x2）：
  - 响应异常后处理器内核处于此状态。
  - 有关异常机制的详情，请参见第 0 章。
- NMI 处理模式（该 Machine Sub-Mode 的编码是 0x3）：
  - 响应 NMI 后处理器内核处于此状态。
  - 有关 NMI 机制的详情，请参见第 4 章。
- 中断处理模式（该 Machine Sub-Mode 的编码是 0x1）：
  - 响应中断后处理器内核处于此状态。
  - 有关中断机制的详情，请参见第 5 章。

处理器内核当前处于的 Machine Sub-Mode 反映在 CSR 寄存器 msubm 的 TYP 域中，因此软件可以通过读取此 CSR 寄存器查看当前处于的 Machine Sub-Mode。有关 msubm 寄存器的详情，请参见第 7.5.3 节。

注意：在 RISC-V 架构中，进入异常、NMI 或者中断也被统称为 Trap。

2.2.4. 模式（Mode）的查看

处理器模式（Mode）查看的关键要点如下：

- 根据 RISC-V 的架构定义，处理器当前的 Machine Mode 或者 User Mode 并没有反映在任何软件可见的寄存器中（处理器内核会维护一个对软件不可见的硬件寄存器），因此软件

程序无法通过读取任何寄存器而查看当前自己所处的 Machine Mode 或者 User Mode。

- Bumblebee 内核的四种机器子模式（Machine Sub-Mode）反映在 CSR 寄存器 msubm 的 TYP 域中，因此软件可以通过读取此 CSR 寄存器查看当前处于的 Machine Sub-Mode。

2.2.5. Machine Mode 到 User Mode 的切换

在 Machine Mode 下可以直接执行 mret 指令。从 Machine Mode 切换到 User Mode 只能通过执行 mret 指令发生。由于如第 2.2.3 节中所述，Machine Mode 可能处于四种不同的状态下，分别介绍如下：

- 如果是在正常机器模式下，执行 mret 指令的硬件行为与异常处理模式下执行 mret 指令的行为相同，请参见第 3.5 节了解其详情。
  - 因此，如果在正常机器模式下，希望从 Machine Mode 切换到 User Mode，那么需要软件先修改 mstatus 的 MPP 域的值，然后执行 mret 指令达到模式切换的效果。典型的程序代码片段如下所示：

```
/* Switch Machine sub-mode to User mode */
li t0,  MSTATUS_MPP  // MSTATUS_MPP 的值为 0x00001800，即对应 mstatus 的 MPP 位域，请参
                     // 见第 7.4.7 节了解 mstatus 的位域详情。
csrc mstatus,  t0    // 将 mstatus 寄存器的 MPP 位域清为 0
   la t0,  1f        // 将前面的标签 1 所在的 PC 地址赋值给 t0
   csrw mepc,  t0    // 将 t0 的值赋值给 CSR 寄存器 mepc
   mret              // 执行 mret 指令，则会将模式切换到 User Mode，并且从前的标签 1 处开始执行
                     // 程序（标签 1 即为 mret 的下一条指令的位置）
1:                   // 标签 1 的位置
```

- 如果是在异常处理模式下，执行 mret 指令的硬件行为，请参见第 3.5 节了解其详情。
  - 通常来说，mret 指令用于从异常处理模式下退出至进入异常之前的模式。
  - 如果明确希望从 Machine Mode 退出至 User Mode（或者正常机器模式），那么需要软件先修改 mstatus 的 MPP 域的值，然后执行 mret 指令达到模式切换的效果。
- 如果是在中断处理模式下，执行 mret 指令的硬件行为，请参见第 5.7 节了解其详情。
  - 通常来说，mret 指令用于从中断处理模式下退出至进入中断之前的模式。
  - 如果明确希望从 Machine Mode 退出至 User Mode（或者正常机器模式），那么需要

软件先修改 mstatus 的 MPP 域的值，然后执行 mret 指令达到模式切换的效果。

- 如果是在 NMI 处理模式下，执行 mret 指令的硬件行为，请参见第 4.4 节了解其详情。
  - 通常来说，mret 指令用于从 NMI 处理模式下退出至正常机器模式。
  - 如果明确希望从 Machine Mode 退出至 User Mode（或者正常机器模式），那么需要软件先修改 mstatus 的 MPP 域的值，然后执行 mret 指令达到模式切换的效果。

注意：

- 如果在 User Mode 下直接执行 mret 指令会产生非法指令（Illegal Instruction）异常。

2.2.6. User Mode 到 Machine Mode 的切换

Bumblebee 内核从 User Mode 切换到 Machine Mode 只能通过异常、响应中断或者 NMI 的方式发生：

- 响应异常进入异常处理模式。请参见第 3.4 节了解其详情。
  - 注意：软件可以通过调用 ecall 指令强行进入 ecall 异常处理函数。
- 响应中断进入中断处理模式。请参见第 5.6 节了解其详情。
- 响应 NMI 进入 NMI 处理模式。请参见第 4.3 节了解其详情。

2.2.7. 中断、异常、NMI 的嵌套

中断和异常能自我发生嵌套，NMI 无法自我嵌套：

- 在 NMI 处理模式下，如果再次发生 NMI，新来的 NMI 会被屏蔽掉，因此，NMI 无法自我嵌套，请参见第 4.6 节了解详情。
- 在异常处理模式下，如果再次发生异常，这属于异常嵌套情形，请参见第 3.7 节了解详情。
- 在中断处理模式下，如果再次发生中断，这属于中断嵌套情形，请参见第 5.11 节了解详情。

中断、异常和 NMI 彼此之间也可能会发生嵌套，存在如下情形：

- 在中断处理模式下发生了异常，则进入异常处理模式。
- 在 NMI 处理模式下发生了异常，则进入异常处理模式。
- 在中断处理模式下发生了 NMI，则进入 NMI 处理模式。
- 在异常处理模式下发生了 NMI，则进入 NMI 处理模式。
- 注意：在 NMI 和异常模式下默认由于全局中断位被硬件自动关闭，因此不会再响应中断。

为了能够保证异常和 NMI 彼此之间发生嵌套后还能够恢复到之前的状态（Recoverable），Bumblebee 内核实现了一种“两级 NMI/异常状态堆栈（Two Levels of NMI/Exception State Save Stacks）”技术，请参见第 4.6 节了解更多详情。

2.3. 物理存储器保护（PMP）

由于 Bumblebee 内核是面向微控制器领域的低功耗内核，其不支持虚拟地址管理单元（Memory Management Unit），因此所有的地址访问操作都是使用的物理地址。为了根据不同的存储器物理地址区间和不同的 Privilege Mode 进行权限隔离和保护，RISC-V 架构标准定义了物理存储保护机制，即（Physical Memory Protection，PMP）单元。

注意：Bumblebee 内核并不支持 PMP 单元。

3. Bumblebee 内核异常机制介绍

3.1. 异常概述

异常（Exception）机制，即处理器核在顺序执行程序指令流的过程中突然遇到了异常的事情而中止执行当前的程序，转而去处理该异常，其要点如下：

- 处理器遇到的“异常的事情”称为异常（Exception）。异常是由处理器内部事件或程序执行中的事件引起的，譬如本身硬件故障、程序故障，或者执行特殊的系统服务指令而引起的，简而言之是一种内因。
- 异常发生后，处理器会进入异常服务处理程序。

3.2. 异常屏蔽

RISC-V 架构中规定异常是不可以被屏蔽的，也就是说一旦发生了异常，处理器一定会停止当前操作转而进入异常处理模式。

3.3. 异常的优先级

处理器内核可能存在多个异常同时发生的情形，因此异常也有优先级。异常的优先级如表 3-1 中所示，异常编号数字越小的异常优先级越高。

3.4. 进入异常处理模式

进入异常时，Bumblebee 内核的硬件行为可以简述如下。注意，下列硬件行为在一个时钟周期内同时完成：

- 停止执行当前程序流，转而从 CSR 寄存器 mtvec 定义的 PC 地址开始执行。
- 更新相关 CSR 寄存器，分别是以下几个寄存器：
  - mcause（Machine Cause Register）
  - mepc（Machine Exception Program Counter）

- mtval（Machine Trap Value Register ）
- mstatus（Machine Status Register）

- 更新处理器内核的 Privilege Mode 以及 Machine Sub-Mode。

异常响应总体过程如图 3-1 所示。

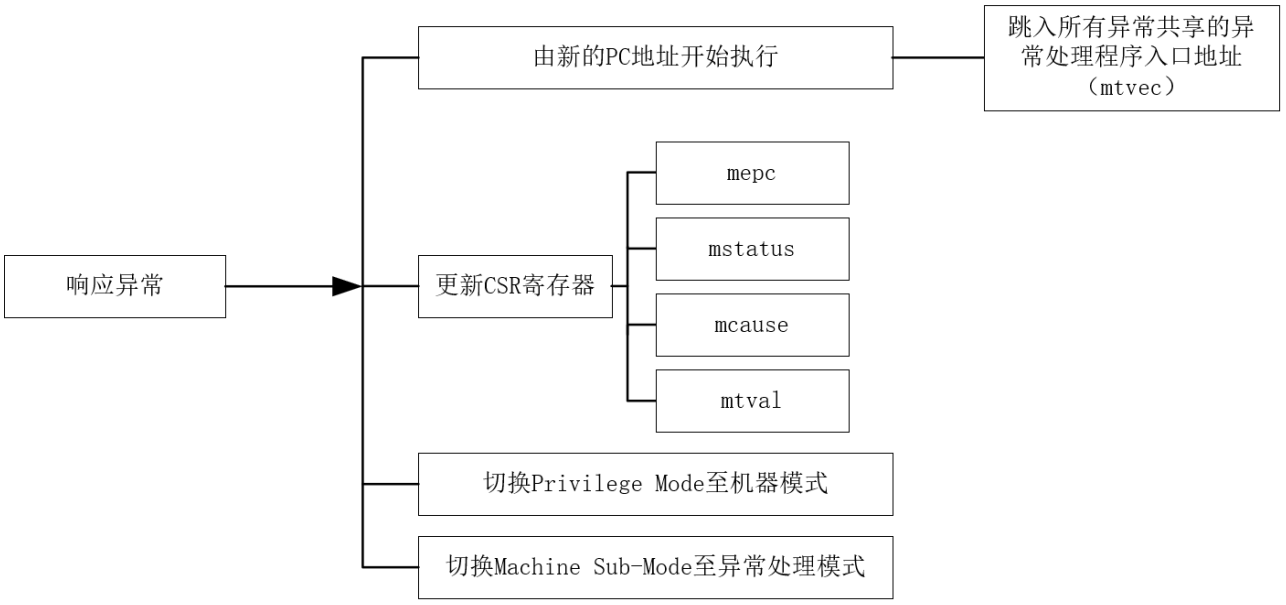

图 3-1 异常响应总体过程

下文将分别予以详述。

3.4.1. 从 mtvec 定义的 PC 地址开始执行

Bumblebee 内核遇到异常后跳入的 PC 地址由 CSR 寄存器 mtvec 指定。

mtvec 寄存器是一个可读可写的 CSR 寄存器，因此软件可以编程更改其中的值。mtvec 寄存器的详细格式如表 7-3 所示。

3.4.2. 更新 CSR 寄存器 mcause

Bumblebee 内核在进入异常时，CSR 寄存器 mcause 被同时（硬件自动）更新，以反映当前的异常种类，软件可以通过读此寄存器查询造成异常的具体原因。

mcause 寄存器的详细格式如表 7-6 所示，其中低 5 位为异常编号域，用于指示各种不同的异常类型，如表 3-1 所示。

表 3-1 mcause 寄存器中的 Exception Code

异常编号（Exception Code）	异常和中断类型	同步/异步	描述
0	指令地址非对齐（Instruction address misaligned）	同步	指令 PC 地址非对齐。 注意：该异常类型在配置了“C”扩展指令子集的处理器中不可能发生。
1	指令访问错误（Instruction access fault）	同步	取指令访存错误。
2	非法指令（Illegal instruction）	同步	非法指令。
3	断点（Breakpoint）	同步	RISC-V 架构定义了 EBREAK 指令，当处理器执行到该指令时，会发生异常进入异常服务程序。该指令往往用于调试器（Debugger）使用，譬如设置断点
4	读存储器地址非对齐（Load address misaligned）	同步	Load 指令访存地址非对齐。 注意：Bumblebee 内核不支持地址非对齐的数据存储器读写操作，因此当访问地址非对齐时会产生此异常。
5	读存储器访问错误（Load access fault）	非精确异步	Load 指令访存错误。
6	写存储器和 AMO 地址非对齐（Store/AMO address misaligned）	同步	Store 或者 AMO 指令访存地址非对齐。注意：Bumblebee 内核不支持地址非对齐的数据存储器读写操作，因此当访问地址非对齐时会产生此异常。
7	写存储器和 AMO 访问错误（Store/AMO access fault）	非精确异步	Store 或者 AMO 指令访存错误。
8	用户模式环境调用（Environment call from U-mode）	同步	User Mode 下执行 ecall 指令。 RISC-V 架构定义了 ecall 指令，当处理器执行到该指令时，会发生异常进入异常服务程序。该指令往往供软件使用，强行进入异常模式。
11	机器模式环境调用（Environment call from M-mode）	同步	Machine Mode 下执行 ecall 指令。 RISC-V 架构定义了 ecall 指令，当处理器执行到该指令时，会发生异常进入异常服务程序。该指令往往供软件使用，强行进入异常模式。

3.4.3. 更新 CSR 寄存器 mepc

Bumblebee 内核退出异常时的返回地址由 CSR 寄存器 mepc（Machine Exception Program Counter）保存。在进入异常时，硬件将自动更新 mepc 寄存器的值，该寄存器将作为退出异常的返回地址，在异常结束之后，能够使用它保存的 PC 值回到之前被异常停止执行的程序点。

注意：

- 出现异常时，异常返回地址 mepc 的值被更新为当前发生异常的指令 PC。
- 虽然 mepc 寄存器会在异常发生时自动被硬件更新，但是 mepc 寄存器本身也是一个可读可写的寄存器，因此软件也可以直接写该寄存器以修改其值。

3.4.4. 更新 CSR 寄存器 mtval

Bumblebee 内核在进入异常时，硬件将自动更新 CSR 寄存器 mtval（Machine Trap Value Register ），以反映引起当前异常的存储器访问地址或者指令编码：

- 如果是由存储器访问造成的异常，譬如遭遇硬件断点、取指令、存储器读写造成的异常，则将存储器访问的地址更新到 mtval 寄存器中。
- 如果是由非法指令造成的异常，则将该指令的指令编码更新到 mtval 寄存器中。

3.4.5. 更新 CSR 寄存器 mstatus

mstatus 寄存器的详细格式如表 7-2 所示，Bumblebee 内核在进入异常时，硬件将自动更新 CSR 寄存器 mstatus（Machine Status Register）的某些域：

- mstatus.MPIE 域的值被更新为异常发生前 mstatus.MIE 域的值，如节和第 8.2 所示。mstatus.MPIE 域的作用是在异常结束之后，能够使用 mstatus.MPIE 的值恢复出异常发生之前的 mstatus.MIE 值。
- mstatus.MIE 域的值则被更新成为 0（意味着进入异常服务程序后中断被全局关闭，所有的中断都将被屏蔽不响应）。

- mstatus.MPP 域的值被更新为异常发生前的 Privilege Mode，如节和第 8.2 所示。mstatus.MPP 域的作用是在异常结束之后，能够使用 mstatus.MPP 的值恢复出异常发生之前的 Privilege Mode。

3.4.6. 更新 Privilege Mode

异常需要在机器模式（Machine Mode）下处理，在进入异常时，处理器内核的 Privilege Mode 被更新为机器模式。

3.4.7. 更新 Machine Sub-Mode

Bumblebee 内核的 Machine Sub-Mode 实时反映在 CSR 寄存器 msubm.TYP 域中。在进入异常时，处理器内核的 Machine Sub-Mode 被更新为异常处理模式，因此：

- CSR 寄存器 msubm.PTYP 域的值被更新为异常发生前的 Machine Sub-Mode（msubm.TYP 域的值），如节和第 8.2 所示。msubm.PTYP 域的作用是在异常结束之后，能够使用 msubm.PTYP 的值恢复出异常发生之前的 Machine Sub-Mode 值。
- CSR 寄存器 msubm.TYP 域的值则被更新为“异常处理模式”如节和第 8.2 所示，以实时反映当前的模式已经是“异常处理模式”。

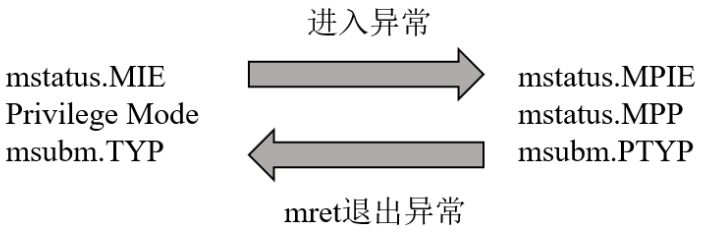

图 3-2 进入/退出异常时 CSR 寄存器的变化

3.5. 退出异常处理模式

当程序完成异常处理之后，最终需要从异常服务程序中退出。

由于异常处理处于 Machine Mode 下，所以退出异常时，软件必须使用 mret 指令。处理器执行 mret 指令后的硬件行为如下。注意，下列硬件行为在一个时钟周期内同时完成：

- 停止执行当前程序流，转而从 CSR 寄存器 mepc 定义的 PC 地址开始执行。
- 更新 CSR 寄存器 mstatus（Machine Status Register）如节和第 8.2 所示，并更新处理器内核的 Privilege Mode 以及 Machine Sub-Mode。

退出异常的总体过程如所图 3-3 示。

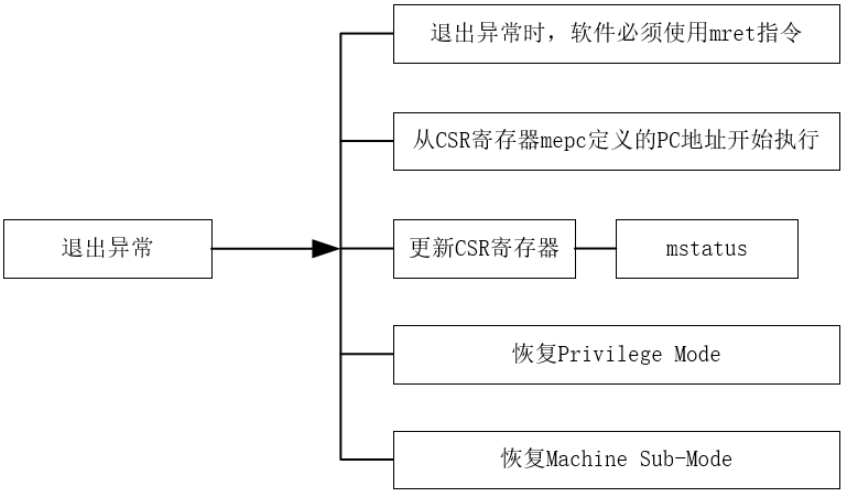

图 3-3 退出异常总体过程

下文将分别予以详述。

3.5.1. 从 mepc 定义的 PC 地址开始执行

在进入异常时，mepc 寄存器被同时更新，以反映当时遇到异常的指令 PC 值。通过这个机制，意味着 mret 指令执行后处理器回到了当时遇到异常的指令的 PC 地址，从而可以继续执行之前被中止的程序流。

注意：退出异常之前可能需要使用软件更新 mepc 的值。例如，如果异常由 ecall 或 ebreak 产

生，由于 mepc 的值被更新为 ecall 或 ebreak 指令自己的 PC。因此在异常返回时，如果直接使用 mepc 保存的 PC 值作为返回地址，则会再次跳回 ecall 或者 ebreak 指令，从而造成死循环（执行 ecall 或者 ebreak 指令导致重新进入异常）。正确的做法是在异常处理程序中软件改变 mepc 指向下一条指令，由于现在 ecall/ebreak 都是 4 字节指令，因此改写设定 mepc=mepc+4 即可。

3.5.2. 更新 CSR 寄存器 mstatus

mstatus 寄存器的详细格式如表 7-2 所示。在执行 mret 指令后，硬件将自动更新 CSR 寄存器 mstatus 的某些域：

- mstatus.MIE 域的值被恢复为当前 mstatus.MPIE 的值。
- 当前 mstatus.MPIE 域的值则被更新为 1。
- mstatus.MPP 域的更新值分为以下两种情形：
  - 配置了用户模式 U-mode 时，mstatus.MPP 被更新为 0x0。
  - 没有配置用户模式 U-mode 时，mstatus.MPP 被更新为 0x11。

在进入异常时，mstatus.MPIE 的值曾经被更新为异常发生前的 mstatus.MIE 值，如节和第 8.2 所示。而 mret 指令执行后，将 mstatus.MIE 域的值恢复为 mstatus.MPIE 的值。通过这个机制，则意味着 mret 指令执行后，处理器的 mstatus.MIE 值被恢复成异常发生之前的值（假设之前的 mstatus.MIE 值为 1，则意味着中断被重新全局打开）。

3.5.3. 更新 Privilege Mode

在进入异常时，mstatus.MPP 的值曾经被更新为异常发生前的 Privilege Mode，而在执行 mret 指令后，处理器的 Privilege Mode 被恢复为 mstatus.MPP 的值，如节和第 8.2 所示。通过这个机制，保证了处理器回到了异常发生前的处理器的 Privilege Mode。

3.5.4. 更新 Machine Sub-Mode

Bumblebee 内核的 Machine Sub-Mode 实时反映在 CSR 寄存器 msubm.TYP 域中。在执行

mret 指令后，硬件将自动恢复处理器的 Machine Sub-Mode 为 msubm.PTYP 域的值：

- 在进入异常时，msubm.PTYP 域的值曾经被更新为异常发生前的 Machine Sub-Mode 值。而使用 mret 指令退出异常后，硬件将处理器 Machine Sub-Mode 的值恢复为 msubm.PTYP 域的值，如图 4-2 所示。通过这个机制，则意味着退出异常后，处理器的 Machine Sub-Mode 被恢复成异常发生之前的 Machine Sub-Mode。

3.6. 异常服务程序

当处理器进入异常后，即开始从 mtvec 寄存器定义的 PC 地址执行新的程序，该程序通常为异常服务程序，并且程序还可以通过查询 mcause 中的异常编号（Exception Code）决定进一步跳转到更具体的异常服务程序。譬如当程序查询 mcause 中的值为 0x2，则得知该异常是非法指令错误（Illegal Instruction）引起的，因此可以进一步跳转到非法指令错误异常服务子程序中去。

注意：由于进入异常和退出异常机制中没有硬件自动保存和恢复上下文的操作，因此需要软件明确地使用（汇编语言编写的）指令进行上下文的保存和恢复。请结合 MCU 芯片的一个完整的异常服务程序代码示例对其进行理解。

3.7. 异常嵌套

Bumblebee 内核支持两级 NMI/异常状态堆栈（Two Levels of NMI/Exception State Save Stacks），更多细节请参见 4.6 节。

4. Bumblebee 内核 NMI 机制介绍

4.1. NMI 概述

NMI（Non-Maskable Interrupt）是处理器内核的一根特殊的输入信号，往往用于指示系统层面的紧急错误（譬如外部的硬件故障等）。在遇到 NMI 之后，处理器内核应该立即中止执行当前的程序，转而去处理该 NMI 错误。

4.2. NMI 屏蔽

Bumblebee 内核中 NMI 是不可以被屏蔽的，也就是说一旦发生了 NMI，处理器一定会停止当前操作转而处理 NMI。

4.3. 进入 NMI 处理模式

进入 NMI 处理模式时，Bumblebee 内核的硬件行为可以简述如下。注意，下列硬件行为在一个时钟周期内同时完成：

- 停止执行当前程序流，转而从 CSR 寄存器 mnvec 定义的 PC 地址开始执行。
- 更新相关 CSR 寄存器，分别是以下几个寄存器：
  - mepc（Machine Exception Program Counter ）
  - mstatus（Machine Status Register）
  - mcause（Machine Cause Register）
- 更新处理器内核的 Privilege Mode 以及 Machine Sub-Mode。

NMI 响应总体过程如图 4-1 所示。

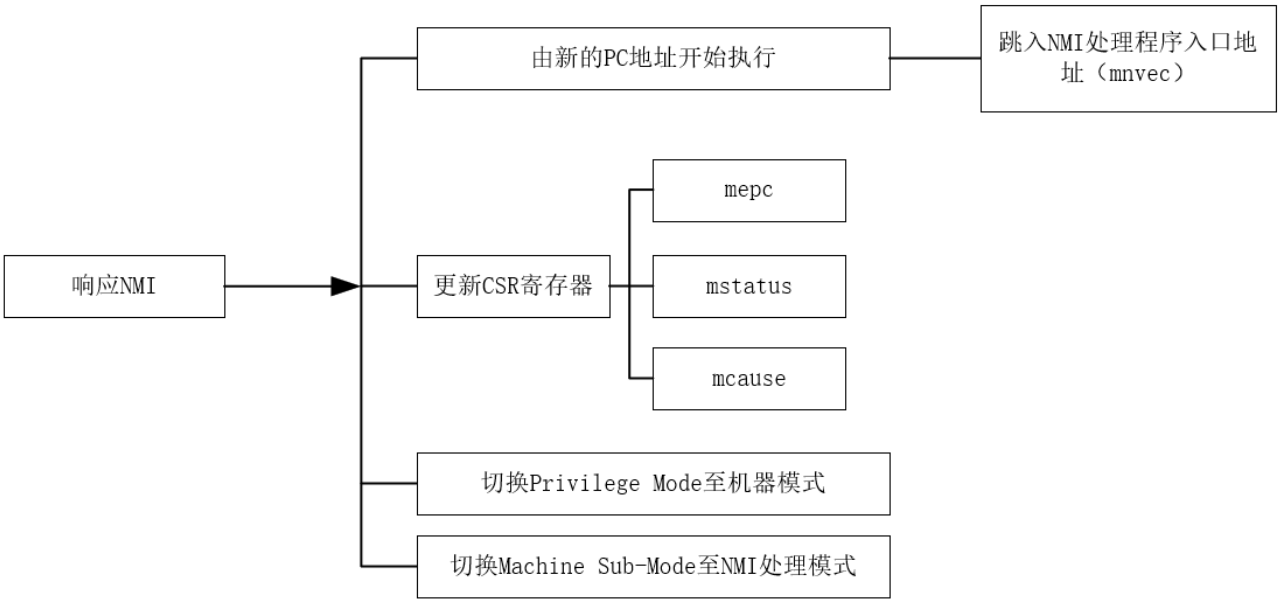

图 4-1 NMI 响应总体过程

下文将分别予以详述。

4.3.1. 从 mnvec 定义的 PC 地址开始执行

Bumblebee 内核遇到 NMI 后跳入的 PC 地址由 CSR 寄存器 mnvec 指定。mnvec 寄存器的值有以下两种情况：

- 当 mmisc_ctl[9]=1 时，mnvec 寄存器的值等于 mtvec，即 NMI 与异常拥有相同的 Trap 入口地址。
- 当 mmisc_ctl[9]=0 时， mnvec 寄存器的值等于 reset_vector，reset_vector 为处理器被 reset 后的 PC 的值。

4.3.2. 更新 CSR 寄存器 mepc

Bumblebee 内核退出 NMI 时的返回地址由 CSR 寄存器 mepc（Machine Exception Program Counter）保存。在进入 NMI 时，硬件将自动更新 mepc 寄存器的值，该寄存器将作为退出 NMI 的返回地址，在 NMI 结束之后，能够使用它保存的 PC 值回到之前被停止执行的程序点。

注意：

- 出现 NMI 时，NMI 返回地址 mepc 被指向下一条尚未执行的指令（因为 NMI 时的指令已经被正确执行）。那么在退出 NMI 后，程序便会回到之前的程序点，从下一条指令开始重新执行。
- 虽然 mepc 寄存器会在 NMI 发生时自动被硬件更新，但是 mepc 寄存器本身也是一个可读可写的寄存器，因此软件也可以直接写该寄存器以修改其值。

4.3.3. 更新 CSR 寄存器 mcause

mcause 寄存器的详细格式如表 7-6 所示。Bumblebee 内核在进入 NMI 时，硬件自动保存当前 Trap 的 ID 到 mcause，以表明 Trap 的原因。中断、异常以及 NMI 都有各自特殊的 Trap ID。NMI 的 Trap ID 有以下两种值：

- 当 mmisc_ctl[9]=1 时，NMI Trap ID 为 0xfff。
- 当 mmisc_ctl[9]=0 时，NMI Trap ID 为 0x1。

通过给每个 Trap 分配特定的 Trap ID，可以识别 Trap 的原因，软件可以根据 Trap 的原因来设计特定的处理程序处理 Trap。

4.3.4. 更新 CSR 寄存器 mstatus

mstatus 寄存器的详细格式如表 7-2 所示， Bumblebee 内核在进入 NMI 时，硬件将自动更新 CSR 寄存器 mstatus 的某些域：

- mstatus.MPIE 域的值被更新为 NMI 发生前 mstatus.MIE 域的值，如图 4-2 所示。mstatus.MPIE 域的作用是在 NMI 结束之后，能够使用 mstatus.MPIE 的值恢复出 NMI 发生之前的 mstatus.MIE 值。
- mstatus.MIE 域的值则被更新成为 0（意味着进入 NMI 服务程序后中断被全局关闭，所有的中断都将被屏蔽不响应）。
- mstatus.MPP 域的值被更新为 NMI 发生前的 Privilege Mode。保存 mstatus.MPP 域的作用时在 NMI 结束之后，能够使用 mstatus.MPP 的值恢复出 NMI 发生前的 Privilege Mode。

4.3.5. 更新 Privilege Mode

NMI 处理是在机器模式（Machine Mode）下完成的，所以在进入 NMI 时，处理器内核的特权模式（Privilege Mode）切换成机器模式。

4.3.6. 更新 Machine Sub-Mode

Bumblebee 内核的 Machine Sub-Mode 实时反映在 CSR 寄存器 msubm.TYP 域中。在进入 NMI 时，处理器内核的 Machine Sub-Mode 被更新为 NMI 处理模式，因此：

- CSR 寄存器 msubm.PTYP 域的值被更新为 NMI 发生前的 Machine Sub-Mode（msubm.TYP 域的值），如图 4-2 所示。msubm.PTYP 域的作用是在 NMI 结束之后，能够使用 msubm.PTYP 的值恢复出 NMI 发生之前的 Machine Sub-Mode 值。
- CSR 寄存器 msubm.TYP 域的值则被更新为“NMI 处理模式”如图 4-2 所示，以实时反映当前的模式已经是“NMI 处理模式”。

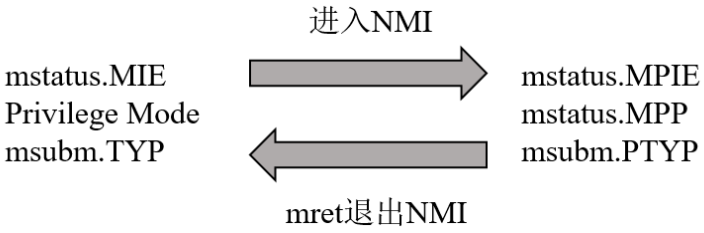

图 4-2 进入/退出 NMI 时 CSR 寄存器的变化

4.4. 退出 NMI 处理模式

当程序完成 NMI 处理之后，最终需要从 NMI 服务程序中退出，并返回主程序。

由于 NMI 处理处于 Machine Mode 下，所以在退出 NMI 时，软件必须使用 mret 指令。处理器执行 mret 指令后的硬件行为如下。注意，下列硬件行为在一个时钟周期内同时完成：

- 停止执行当前程序流，转而从 CSR 寄存器 mepc 定义的 PC 地址开始执行。
- 更新 CSR 寄存器 mstatus。
- 更新 Privilege Mode 以及 Machine Sub-Mode。

退出 NMI 的总体过程如图 4-3 所示。

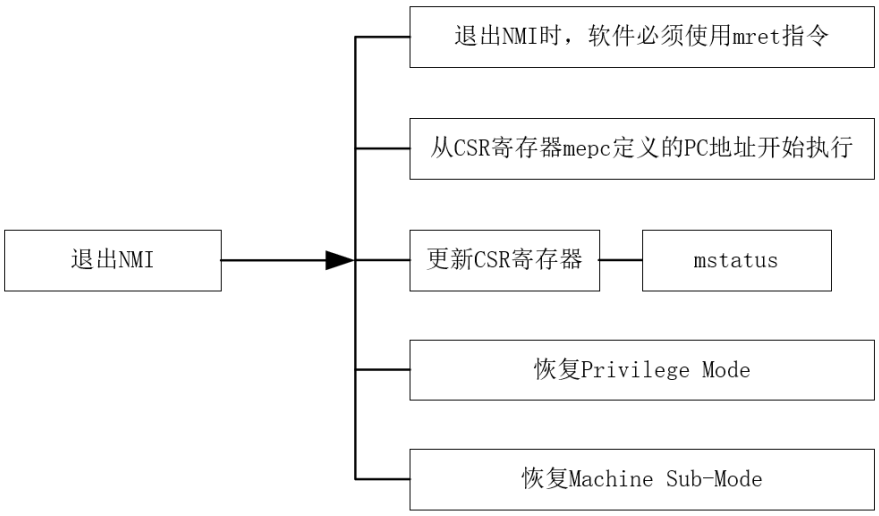

图 4-3 退出 NMI 总体过程

下文将分别予以详述。

4.4.1. 从 mepc 定义的 PC 地址开始执行

在进入 NMI 时，mepc 寄存器被同时更新，以反映当时遇到 NMI 的下一条指令的 PC 值。通过这个机制，意味着 mret 指令执行后处理器回到了当时遇到 NMI 的下一条指令的 PC 地址，从而可以继续执行之前被中止的程序流。

4.4.2. 更新 CSR 寄存器 mstatus

mstatus 寄存器的详细格式如表 7-2 所示，在执行 mret 指令后，硬件将自动更新 CSR 寄存器 mstatus 某些域：

- mstatus.MIE 域的值被恢复为当前 mstatus.MPIE 的值。
- mstatus.MPIE 域的值则被更新为 1。
- mstatus.MPP 域的更新值分为以下两种情形：
  - 配置了用户模式 U-mode 时，mstatus.MPP 被更新为 0x0。

- 没有配置用户模式 U-mode 时，mstatus.MPP 被更新为 0x11。

在进入 NMI 时，mstatus.MPIE 的值曾经被更新为 NMI 发生前的 mstatus.MIE 值。而 mret 指令执行后，将 mstatus.MIE 的值恢复为 mstatus.MPIE 的值，如图 4-2 所示。通过这个机制，则意味着 mret 指令执行后，处理器的 mstatus.MIE 值被恢复成 NMI 发生之前的值（假设之前的 mstatus.MIE 值为 1，则意味着中断被重新全局打开）。

4.4.3. 更新 Privilege Mode

在进入 NMI 时，mstatus.MPP 的值曾经被更新为 NMI 发生前的 Privilege Mode，而在执行 mret 指令后，处理器的 Privilege Mode 被恢复为 mstatus.MPP 的值，如图 4-2 所示。通过这个机制，保证了处理器回到了 NMI 发生前的处理器的 Privilege Mode。

4.4.4. 更新 Machine Sub-Mode

Bumblebee 内核的 Machine Sub-Mode 实时反映在 CSR 寄存器 msubm.TYP 域中。在执行 mret 指令后，硬件将自动恢复处理器的 Machine Sub-Mode 为 msubm.PTYP 域的值：

- 在进入 NMI 时，msubm.PTYP 域的值曾经被更新为 NMI 发生前的 Machine Sub-Mode 值。而使用 mret 指令退出 NMI 后，硬件将处理器 Machine Sub-Mode 的值恢复为 msubm.PTYP 域的值，如图 4-2 所示。通过这个机制，则意味着退出 NMI 后，处理器的 Machine Sub-Mode 被恢复成 NMI 发生之前的 Machine Sub-Mode。

4.5. NMI 服务程序

当处理器进入 NMI 后，即开始从 mnvec 寄存器定义的 PC 地址执行新的程序，该程序通常为 NMI 服务程序。

注意：由于进入 NMI 和退出 NMI 机制中没有硬件自动保存和恢复上下文的操作，因此需要软件明确地使用（汇编语言编写的）指令进行上下文的保存和恢复。请结合 MCU 芯片的一个完整的 NMI 服务程序代码示例对其进行理解。

4.6. NMI/异常嵌套

Bumblebee 内核自定义了如图 4-4 所示的两级 NMI/异常状态堆栈（Two Levels of NMI/Exception State Save Stacks），至多保存三级 NMI/异常的处理器状态，可以实现二级可恢复的 NMI/异常嵌套。

注意：由于处理器器处于 NMI 状态时，NMI 的响应在硬件上被屏蔽掉了，因此 NMI 无法实现自我嵌套。Bumblebee 内核的 NMI/异常嵌套只支持以下 3 种嵌套：

- NMI 嵌套异常
- 异常嵌套异常
- 异常嵌套 NMI

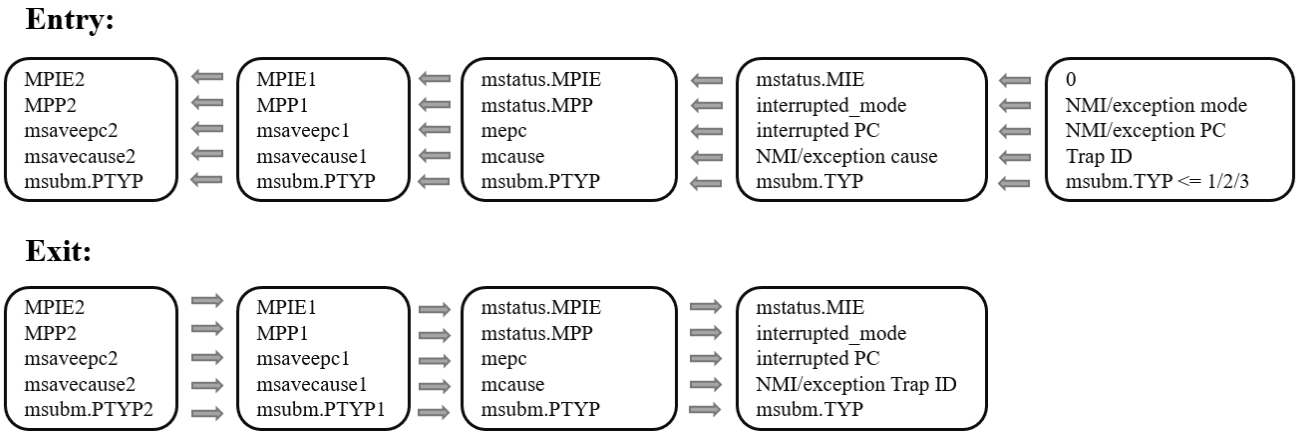

图 4-4 Bumblebee 内核两级 NMI/异常状态堆栈机制示意图

4.6.1. 进入 NMI/异常嵌套

响应 NMI 与异常时，Bumblebee 内核的硬件行为如图 4-4 所示，可以简述如下。

- 停止执行当前程序流，转而从新的 PC 地址开始执行。
  - 如果响应的是异常则从 mtvec 所存储的 PC 地址开始执行。

- 如果响应的是 NMI 则从 mnvec 所存储的 PC 地址开始执行。

- 更新相关 CSR 寄存器，分别是以下几个寄存器及其相关的域：

  - mepc: 记录当前 NMI/异常发生前的 PC，退出 NMI/异常时可从 mepc 恢复 NMI/异常发生前的 PC。

  - msaveepc1：第一级 NMI/异常状态堆栈记录第一级嵌套 NMI/异常（被当前 NMI/异常嵌套的 NMI/异常）发生前的 PC，亦即当前 NMI/异常发生前的 mepc 值，退出 NMI/异常时可从 msaveepc1 恢复 mepc 的值。

  - msaveepc2: 第二级 NMI/异常状态堆栈记录第二级嵌套 NMI/异常（被第一级嵌套 NMI/异常嵌套的 NMI/异常）发生前的 PC，亦即当前 NMI/异常发生前的 msaveepc1 的值，退出 NMI/异常时可从 msaveepc2 恢复 msaveepc1 的值。

  - mstatus:

    - MPIE：记录当前 NMI/异常发生之前的 MIE。

    - MPP: 记录当前 NMI/异常发生之前的 Privilege Mode。

  - msavestatus:

    - MPIE1：第一级 NMI/异常状态堆栈记录第一级嵌套 NMI/异常发生前的 MIE，亦即当前 NMI/异常发生前的 MPIE，退出 NMI/异常时可从 MPIE1 恢复 MPIE 的值。

    - MPIE2：第二级 NMI/异常状态堆栈记录第二级嵌套 NMI/异常发生前的 MIE，亦即当前 NMI/异常发生前的 MPIE1，退出 NMI/异常时可从 MPIE2 恢复 MPIE1 的值。

    - MPP1：第一级 NMI/异常状态堆栈记录第一级嵌套 NMI/异常发生前的 Privilege Mode，亦即当前 NMI/异常发生前的 MPP，退出 NMI/异常时可从 MPP1 恢复 MPP 的值。

    - MPP2：第二级 NMI/异常状态堆栈记录第二级嵌套 NMI/异常发生前的 Privilege Mode，亦即当前 NMI/异常发生前的 MPP1，退出 NMI/异常时可从 MPP2 恢复 MPP1 的值。

- mcause： 记录发生当前 NMI/异常的原因。
- msavecause1:第一级 NMI/异常状态堆栈记录第一级嵌套 NMI/异常原因。
- msavecause2:第二级 NMI/异常状态堆栈记录第二级嵌套 NMI/异常原因。
- msubm:
  - TYP：记录当前 NMI/异常的 Trap 类型。
  - PTYP: 记录当前 NMI/异常发生前处理器所处 Trap 的类型
  - PTYP1: 第一级 NMI/异常状态堆栈记录第一级嵌套 NMI/异常发生前的 Machine Sub Mode，亦即当前 NMI/异常发生前的 PTYP，退出 NMI/异常时可从 PTYP1 恢复 PTYP 的值。
  - PTYP2：第二级 NMI/异常状态堆栈记录第二级嵌套 NMI/异常发生前的 Machine Sub Mode，亦即当前 NMI/异常发生前的 PTYP1，退出 NMI/异常时可从 PTYP2 恢复 PTYP1 的值。

- NMI/异常处理是在机器模式（Machine Mode）下完成的，所以在进入 NMI/异常时，处理器内核的特权模式（Privilege Mode）切换成机器模式。

4.6.2. 退出 NMI/异常嵌套

当程序完成 NMI/异常处理之后，最终需要从 NMI/异常服务程序退出，返回上级 NMI/异常或者主程序，退出之前需要从相关寄存器恢复处理器状态，这是通过 mret 指令完成的，处理器执行 mret 指令后的硬件行为如图 4-4，可以简述如下。

- 停止执行当前程序流，转而从 CSR 寄存器 mepc 定义的 PC 地址开始执行。
- 更新相关 CSR 寄存器，分别是以下几个寄存器及其相关的域：
  - mepc（Machine Exception Program Counter）：恢复为存储在 msaveepc1 中第一级嵌套 NMI/异常发生前的 PC。
  - msaveepc1：第一级 NMI/异常状态堆栈，mret 发生时从第二级 NMI/异常状态堆栈 msaveepc2 恢复寄存器 msaveepc1 值，即恢复为存储在 msaveepc2 中的第二级嵌套

NMI/异常发生前的 PC。

- mstatus（Machine Status Register）
  - MPIE：恢复为存储在 MPIE1 中的第一级嵌套 NMI/异常发生前的 MIE 。
  - MPP: 恢复为存储在 MPP1 中的第一级嵌套 NMI/异常发生前的 Privilege Mode。
- msavestatus:
  - MPIE1：第一级 NMI/异常状态堆栈，mret 发生时从第二级 NMI/异常状态堆栈 MPIE2 恢复寄存器域 msavestatus.MPIE1 的值，即恢复为存储在 MPIE2 中的第二级嵌套 NMI/异常发生前的 MIE。
  - MPP1：第一级 NMI/异常状态堆栈，mret 发生时从第二级 NMI/异常状态堆栈 MPP2 恢复寄存器域 msavestatus.MPP1 的值，即恢复为存储在 MPP2 中的第二级嵌套 NMI/异常发生前的 Privilege Mode。
- mcause（Machine Cause Register）： 恢复为存储在 msavecause1 中的第一级嵌套 NMI/异常的原因。
- msavecause1：第一级 NMI/异常状态堆栈，mret 发生时从第二级 NMI/异常状态堆栈 msavecause2 恢复寄存器 msavecause1 的值，即恢复为存储在 msavecause2 中的第二级嵌套 NMI/异常的原因。
- msubm（Machine Sub-Mode Register）
  - TYP：恢复为存储在 msubm.PTYP 中的当前 NMI/异常发生前处理器的 Trap 类型。
  - PTYP: 恢复为存储在 msubm.PTYP1 中第一级嵌套 NMI/异常发生前处理器的 Trap 类型。
  - PTYP1: 第一级 NMI/异常状态堆栈，mret 发生时从第二级 NMI/异常状态堆栈 PTYP2 恢复寄存器域 msubm.PTYP1 的值，即恢复为存储在 msubm.PTYP2 中的第二级嵌套 NMI/异常发生前处理器的 Trap 类型。

- 根据 mstatus.MPP 域的值更新处理器的 Privilege Mode。

5. Bumblebee 内核中断机制介绍

5.1. 中断概述

中断（Interrupt）机制，即处理器内核在顺序执行程序指令流的过程中突然被别的请求打断而中止执行当前的程序，转而去处理别的事情，待其处理完了别的事情，然后重新回到之前程序中断的点继续执行之前的程序指令流。

中断的若干基本知识要点如下：

- 打断处理器执行的“别的请求”便称之为中断请求（Interrupt Request），“别的请求”的来源便称之为中断源（Interrupt Source），中断源通常来自于内核外部（称之为外部中断源），也可以来自于内核内部（成为内部中断源）。
- 处理器转而去处理的“别的事情”便称之为中断服务程序（Interrupt Service Routine，ISR）。
- 中断处理是一种正常的机制，而非一种错误情形。处理器收到中断请求之后，需要保存当前程序的现场，简称为“保存现场”。等到处理完中断服务程序后，处理器需要恢复之前的现场，从而继续执行之前被打断的程序，简称为“恢复现场”。
- 可能存在多个中断源同时向处理器发起请求的情形，需要对这些中断源进行仲裁，从而选择哪个中断源被优先处理。此种情况称为“中断仲裁”，同时可以给不同的中断分配级别和优先级以便于仲裁，因此中断存在着“中断级别”和“中断优先级”的概念。

5.2. 中断控制器 ECLIC

如第 7.4.13 节中所述，通过软件的不同配置，Bumblebee 内核支持“默认中断模式”和“ECLIC 中断模式”，推荐使用“ECLIC 中断模式”，本文仅对“ECLIC 中断模式”进行介绍。

Bumblebee 内核实现了一个“改进型内核中断控制器（Enhanced Core Local Interrupt Controller，ECLIC）”，可用于多个中断源的管理。Bumblebee 内核中的所有类型（除了调试中断之外）的中断都由 ECLIC 统一进行管理，有关 ECLIC 的详情请参见第 6.2 节。有关 Bumblebee 内核支持的所有中断类型的介绍请参见第 5.3 节。

5.3. 中断类型

Bumblebee 内核支持的中断类型如图 5-1 中所示。

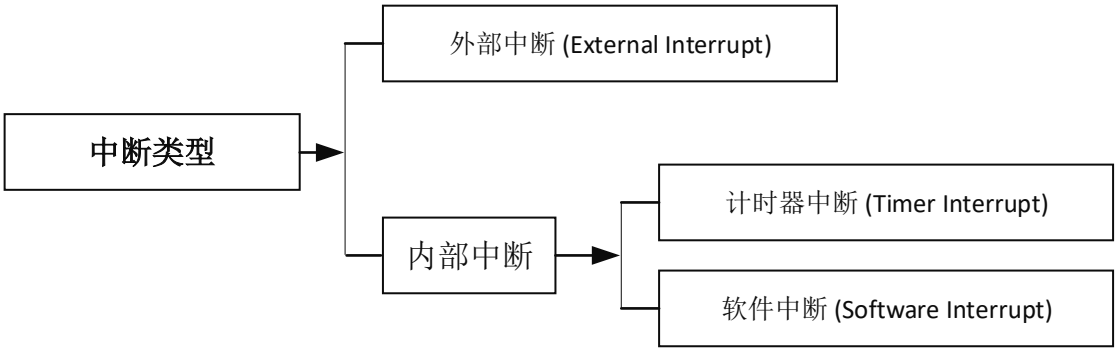

图 5-1 中断类型示意图

下文将分别予以详述。

5.3.1. 外部中断

外部中断是指来自于处理器核外部的中断。外部中断可供用户连接外部中断源，譬如外部设备 UART、GPIO 等产生的中断。

注意：Bumblebee 内核支持多个外部中断源，所有外部中断都由 ECLIC 进行统一管理。

5.3.2. 内部中断

Bumblebee 内核有几种内核私有的内部中断，分别为：

- 软件中断（Software Interrupt）
- 计时器中断（Timer Interrupt）

注意：Bumblebee 内核的内部中断也都由 ECLIC 进行统一管理。

5.3.2.1 软件中断

软件中断要点如下：

- Bumblebee 内核实现了一个 TIMER 单元，TIMER 单元里定义了一个 msip 寄存器，通过其可以产生软件中断，请参见第 6.1.6 节了解其详情。
- 注意：软件中断也由 ECLIC 进行统一管理。

5.3.2.2 计时器中断

计时器中断要点如下：

- Bumblebee 内核实现了一个 TIMER 单元，TIMER 单元里定义了一个计时器，通过其可以产生计时器中断，请参见第 6.1.5 节了解其详情。
- 注意：计时器中断也由 ECLIC 进行统一管理。

5.3.2.3 存储器访问错误中断

“存储器访问错误异常”转化的中断要点如下：

- 当 Bumblebee 内核遭遇“存储器访问错误异常”时，并不会产生异常，而是会被转化成为相应的内部中断，当做一种中断来处理。

5.4. 中断屏蔽

5.4.1. 中断全局屏蔽

Bumblebee 内核的中断可以被屏蔽掉，CSR 寄存器 mstatus 的 MIE 域控制中断的全局使能。请参见第 7.4.8 节了解详情。

5.4.2. 中断源单独屏蔽

对于不同的中断源而言，ECLIC 为每个中断源分配了各自的中断使能寄存器，用户可以通过配置 ECLIC 寄存器来管理各个中断源的屏蔽，请参见第 6.2.6 节了解其详情。

5.5. 中断级别、优先级与仲裁

当多个中断同时出现时，需要进行仲裁。对于 Bumblebee 内核处理器而言，ECLIC 统一管理所有的中断。ECLIC 为每个中断源分配了各自的中断级别和优先级寄存器，用户可以通过配置 ECLIC 寄存器来管理各个中断源的级别和优先级，当多个中断同时发生时，ECLIC 会仲裁出级别和优先级最高的中断，如图 5-2 中所示。请参见第 6.2.9 节了解其详情。

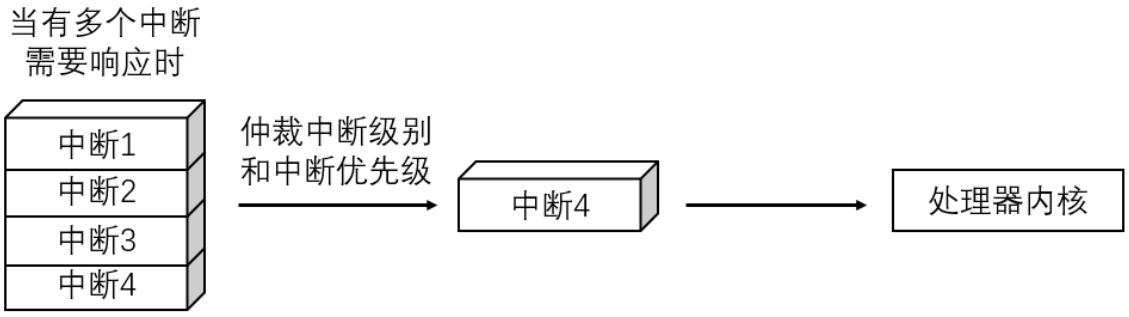

图 5-2 中断仲裁示意图

5.6. 进入中断处理模式

响应中断时，Bumblebee 内核的硬件行为可以简述如下。注意，下列硬件行为在一个时钟周期内同时完成：

- 停止执行当前程序流，转而从新的 PC 地址开始执行。
- 进入中断不仅会让处理器跳转到上述的 PC 地址开始执行，还会让硬件同时更新其他几个 CSR 寄存器，如图 5-4 所示，分别是以下几个寄存器：
  - mepc（Machine Exception Program Counter）
  - mstatus（Machine Status Register）
  - mcause（Machine Cause Register）
  - mintstatus （Machine Interrupt Status Register）

- 除此之外，进入中断还会更新处理器内核的 Privilege Mode 以及 Machine Sub-Mode。
- 总体过程如图 5-3 中所示。

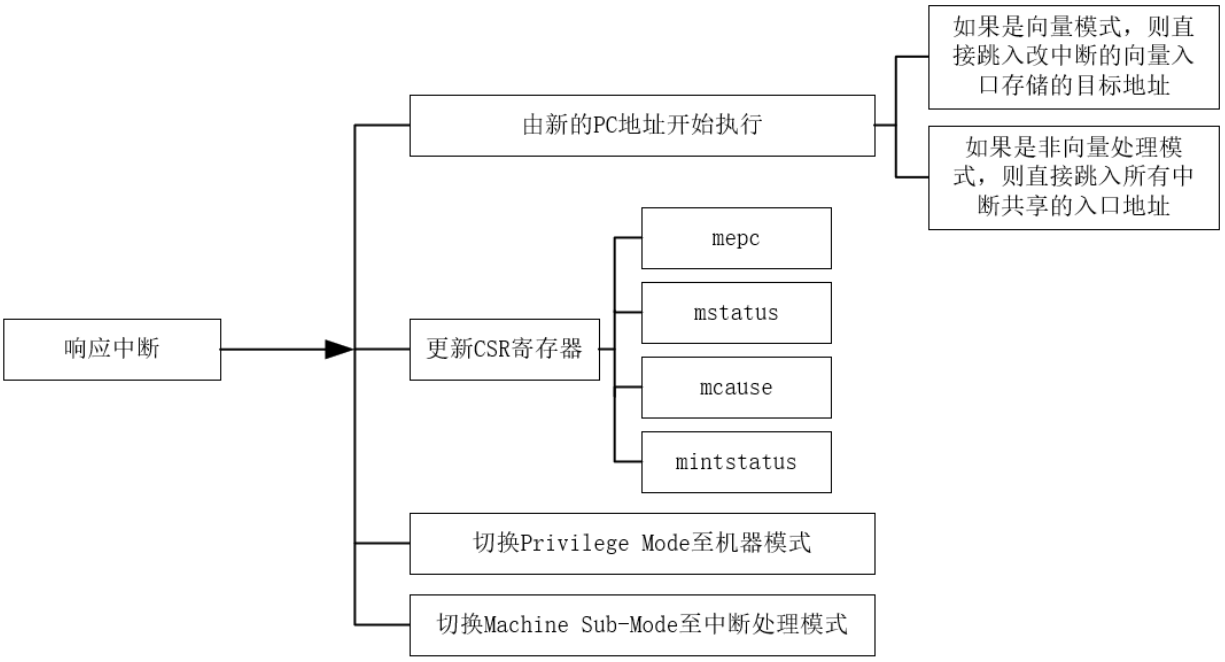

图 5-3 响应中断总体过程

下文将分别予以详述。

5.6.1. 从新的 PC 地址开始执行

ECLIC 的每个中断源均可以设置成向量或者非向量处理（通过寄存器 clicintattr[i]的 shv 域），其要点如下：

- 如果被配置成为向量处理模式，则该中断被处理器内核响应后，处理器直接跳入该中断的向量入口（Vector Table Entry）存储的目标地址。有关中断向量表的详细介绍，请参见第 5.8 节，有关向量处理模式的详细介绍，请参见第 5.13.2 节。
- 如果被配置成为非向量处理模式，则该中断被处理器内核响应后，处理器直接跳入所有

中断共享的入口地址。有关中断非向量处理模式的详细介绍，请参见第 5.13.1 节。

5.6.2. 更新 Privilege Mode

在进入中断时，处理器内核的 Privilege Mode 被更新为 Machine Mode。

5.6.3. 更新 Machine Sub-Mode

Bumblebee 内核的 Machine Sub-Mode 实时反映在 CSR 寄存器 msubm.TYP 域中。在进入中断时，处理器内核的 Machine Sub-Mode 被更新为中断处理模式，因此：

- CSR 寄存器 msubm.PTYP 域的值被更新为中断发生前的 Machine Sub-Mode（msubm.TYP 域的值），如图 5-4 所示。msubm.PTYP 域的作用是在中断结束之后，能够使用 msubm.PTYP 的值恢复出中断发生之前的 Machine Sub-Mode 值。
- CSR 寄存器 msubm.TYP 域的值则被更新为“中断处理模式”，如图 5-4 所示，以实时反映当前的模式已经是“中断处理模式”。

5.6.4. 更新 CSR 寄存器 mepc

Bumblebee 内核退出中断时的返回地址由 CSR 寄存器 mepc 指定。在进入中断时，硬件将自动更新 mepc 寄存器的值，该寄存器将作为退出中断的返回地址，在中断结束之后，能够使用它保存的 PC 值回到之前被停止执行的程序点。

注意：

- 出现中断时，中断返回地址 mepc 被指向一条指令，此指令因为中断的出现而未能完成执行。那么在退出中断后，程序便会回到之前的程序点，从 mepc 所存储的未执行完的指令开始重新执行。
- 虽然 mepc 寄存器会在中断发生时自动被硬件更新，但是 mepc 寄存器本身也是一个可读可写的寄存器，因此软件也可以直接写该寄存器以修改其值。

5.6.5. 更新 CSR 寄存器 mcause 和 mstatus

mcause 寄存器的详细格式如表 7-6 所示。Bumblebee 内核在进入中断时，CSR 寄存器 mcause

被同时（硬件自动）更新，如图 5-4 所示，详情如下：

- 当前的中断被响应后，需要有一种机制能够记录当前这个中断源的 ID 编号。
  - Bumblebee 内核在进入中断时，CSR 寄存器 mcause.EXCCODE 域被更新以反映当前响应的 ECLIC 中断源的 ID 编号，因此软件可以通过读此寄存器查询中断源的具体 ID。
- 当前的中断被响应，有可能是打断了之前正在处理的中断（中断级别相对低，因此可以被打断），需要有一种机制能够记录被打断中断的中断级别（Interrupt Levels）。
  - Bumblebee 内核在进入中断时，CSR 寄存器 mcause.MPIL 域被更新以反映被打断的中断级别（mintstatus.MIL 域的值）。mcause.MPIL 域的作用是在中断结束之后，能够使用 mcause.MPIL 的值恢复出中断发生之前的 mintstatus.MIL 值。
- 当前的中断被响应后，需要有一种机制能够记录响应中断之前的中断全局使能状态和特权模式。
  - Bumblebee 内核在进入中断时，CSR 寄存器 mstatus.MPIE 域的值被更新为中断发生前中断的全局使能状态（mstatus.MIE 域的值）。mstatus.MIE 域的值则被更新成为 0（意味着进入中断服务程序后中断被全局关闭，所有的中断都将被屏蔽不响应）。
  - Bumblebee 内核在进入中断时，处理器的当前特权模式（Privilege Mode）切换到机器模式（Machine Mode），而 CSR 寄存器 mstatus.MPP 域的值被更新为中断发生前特权模式（Privilege Mode）。
- 当前响应的中断如果是向量处理模式，则处理器响应中断后会直接跳入该中断的向量入口（Vector Table Entry）存储的目标地址。有关中断向量处理模式的详细介绍，请参见第 5.13.2 节。从硬件实现上来说，处理器需要分“两步走”，第一步从中断向量表中取出存储的目标地址，然后第二步再跳转到目标地址中去。那么，在第一步“从中断向量表中取出存储的目标地址”这个存储器访问操作的过程中有可能会发生存储器访问错误，需要有一种机制能够记录这种特殊的存储器访问错误。
  - Bumblebee 内核在进入中断时，如果该中断是向量处理模式，CSR 寄存器 mcause.minhv 域的值被更新为 1，直到上述“两步走”操作彻底成功完成后 mcause.minhv 域的值清除为 0。假设中途发生了存储器访问错误，则最终处理器会发生指令访问错误（Instruction access fault），且 mcause.minhv 域的值为 1（没有被清除）。
- 注意：mstatus.MPIE 域和 mstatus.MPP 域的值与 mcause.MPIE 域和 mcause.MPP 域的值是镜像关系，即，在正常情况下，mstatus.MPIE 域的值与 mcause.MPIE 域的值总是完全一

样，mstatus.MPP 域的值与 mcause.MPP 域的值总是完全一样。

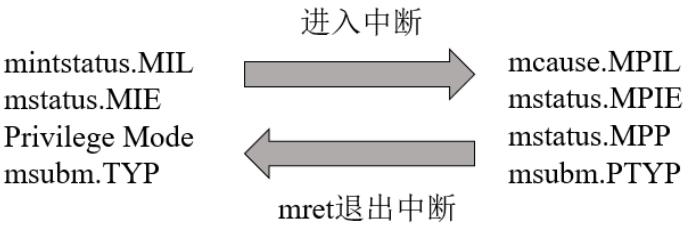

图 5-4 进入/退出中断时 CSR 寄存器的变化

5.7. 退出中断处理模式

当程序完成中断处理之后，最终需要从中断服务程序中退出，并返回主程序。由于中断处理处于 Machine Mode 下，所以退出中断时，软件必须使用 mret 指令。处理器执行 mret 指令后的硬件行为如下。注意，下列硬件行为在一个时钟周期内同时完成：

- 停止执行当前程序流，转而从 CSR 寄存器 mepc 定义的 PC 地址开始执行。
- 执行 mret 指令不仅会让处理器跳转到上述的 PC 地址开始执行，还会让硬件同时更新其他几个 CSR 寄存器，如图 5-4 所示，分别是以下几个寄存器：
  - mstatus（Machine Status Register）
  - mcause（Machine Cause Register）
  - mintstatus（Machine Interrupt Status Register）
- 除此之外，进入中断还会更新处理器内核的 Privilege Mode 以及 Machine Sub-Mode。

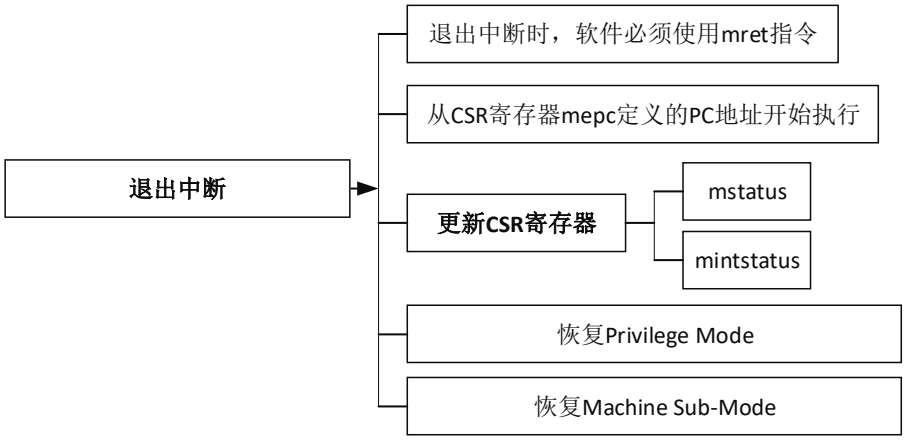

图 5-5 退出中断总体过程

下文将分别予以详述。

5.7.1. 从 mepc 定义的 PC 地址开始执行

在进入中断时，mepc 寄存器被同时更新，以反映当时遇到中断时的 PC 值。软件必须使用 mret 指令退出中断，执行 mret 指令后处理器将从 mepc 定义的 pc 地址重新开始执行。通过这个机制，意味着 mret 指令执行后处理器回到了当时遇到中断时的 PC 地址，从而可以继续执行之前被中止的程序流。

5.7.2. 更新 CSR 寄存器 mcause 和 mstatus

mcause 寄存器的详细格式如表 7-6 所示，执行 mret 指令后，硬件将自动更新 CSR 寄存器 mcause 的某些域：

- 在进入中断时，mcause.MPIL 的值曾经被更新为中断发生前的 mintstatus.MIL 值。而使用 mret 指令退出中断后，硬件将 mintstatus.MIL 的值恢复为 mcause.MPIL 的值。通过这个机制，则意味着退出中断后，处理器的 mintstatus.MIL 值被恢复成中断发生之前的值。
- 在进入中断时，mcause.MPIE 的值曾经被更新为中断发生前的 mstatus.MIE 值。而使用 mret 指令退出中断后，硬件将 mret 指令执行后，将 mstatus.MIE 的值恢复为 mcause.MPIE 的值，如图 5-4 所示。通过这个机制，则意味着退出中断后，处理器的 mstatus.MIE 值

被恢复成中断发生之前的值。

- 在进入中断时，mcause.MPP 的值曾经被更新为中断发生前的特权模式（Privilege Mode）。而使用 mret 指令退出中断后，硬件将处理器特权模式（Privilege Mode）恢复为 mcause.MPP 的值，如图 5-4 所示。通过这个机制，则意味着退出中断后，处理器的特权模式（Privilege Mode）被恢复成中断发生之前的模式。
- 注意：mstatus.MPIE 域和 mstatus.MPP 域的值与 mcause.MPIE 域和 mcause.MPP 域的值是镜像关系，即，在正常情况下，mstatus.MPIE 域的值与 mcause.MPIE 域的值总是完全一样，mstatus.MPP 域的值与 mcause.MPP 域的值总是完全一样。

5.7.3. 更新 Privilege Mode

在执行 mret 指令后，硬件将自动更新处理器的 Privilege Mode 为 mcause.MPP 域的值：

- 在进入中断时，mcause.MPP 的值曾经被更新为中断发生前的特权模式（Privilege Mode）。而使用 mret 指令退出中断后，硬件将处理器特权模式（Privilege Mode）恢复为 mcause.MPP 的值。通过这个机制，则意味着退出中断后，处理器的特权模式（Privilege Mode）被恢复成中断发生之前的模式。

5.7.4. 更新 Machine Sub-Mode

Bumblebee 内核的 Machine Sub-Mode 实时反映在 CSR 寄存器 msubm.TYP 域中。在执行 mret 指令后，硬件将自动恢复处理器的 Machine Sub-Mode 为 msubm.PTYP 域的值：

- 在进入中断时，msubm.PTYP 域的值曾经被更新为中断发生前的 Machine Sub-Mode 值。而使用 mret 指令退出中断后，硬件将处理器 Machine Sub-Mode 的值恢复为 msubm.PTYP 域的值，如图 5-4 所示。通过这个机制，则意味着退出中断后，处理器的 Machine Sub-Mode 被恢复成中断发生之前的 Machine Sub-Mode。

5.8. 中断向量表

如图 5-6 中所示，中断向量表是指在存储器里面开辟的一段连续的地址空间，该地址空间的每个字（Word）用于存储 ECLIC 每个中断源对应的中断服务程序（Interrupt Service Routine，ISR）

函数的 PC 地址。

中断向量表的起始地址由 CSR 寄存器 mtvt 指定，通常可以将 mtvt 寄存器设置为整个代码段的起始位置。

中断向量表的作用非常重要，当处理器响应某个中断源后，无论中断是向量处理模式还是非向量处理模式，硬件最终都将通过查询中断向量表中存储的 PC 地址跳转到其对应的中断服务程序函数中去，请参见第 5.13 节了解更多详细介绍。

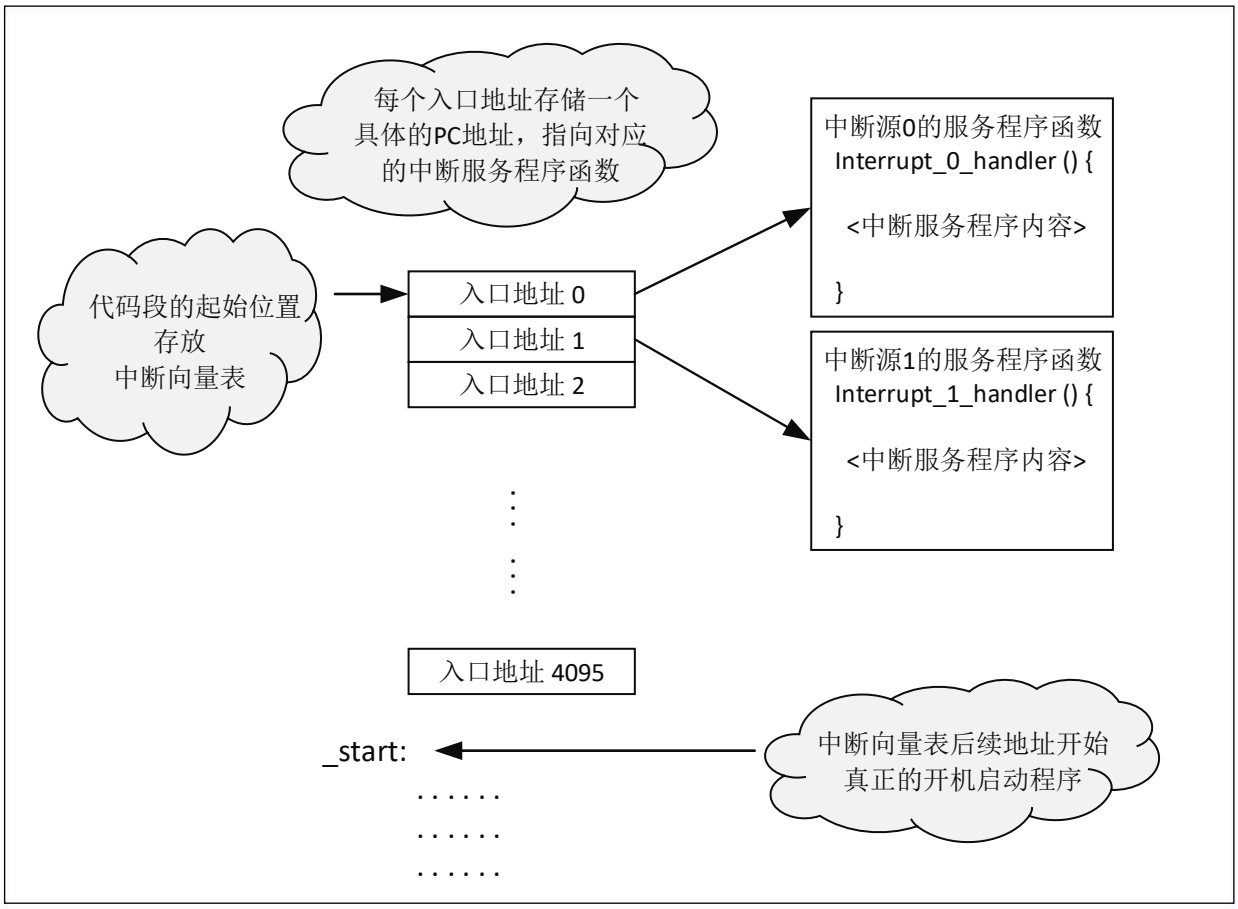

图 5-6 中断向量表示意图

5.9. 进出中断的上下文保存和恢复

RISC-V 架构的处理器在进入和退出中断处理模式时没有硬件自动保存和恢复上下文（通用寄存器）的操作，因此需要软件明确地使用（汇编语言编写的）指令进行上下文的保存和恢复。根据中断是向量处理模式还是非向量处理模式，上下文的保存和恢复涉及到的内容会有所差异，请参见

第 5.13 节了解更多详细介绍。

5.10. 中断响应延迟

中断响应延迟的概念通常是指，从“外部中断源拉高”到“处理器真正开始执行该中断源对应的中断服务程序（Interrupt Service Routine，ISR）中的第一条指令”所消耗的指令周期数。因此，中断响应延迟通常会包含如下几个方面的周期开销：

- 处理器内核响应中断后进行跳转的开销
- 处理器内核保存上下文所花费的周期开销
- 处理器内核跳转到中断服务程序（Interrupt Service Routine，ISR）中去的开销。

取决于中断是向量处理模式还是非向量处理模式，中断响应延迟会有所差异，请参见第 5.13 节了解更多详细介绍。

5.11. 中断嵌套

处理器内核正在处理某个中断的过程中，可能有一个级别更高的新中断请求到来，处理器可以中止当前的中断服务程序，转而开始响应新的中断，并执行其“中断服务程序”，如此便形成了中断嵌套（即前一个中断还没响应完，又开始响应新的中断），并且嵌套的层次可以有很多层。

以图 5-7 中的示例为例：

- 假设处理器正在处理计时器中断，突然有另外一个按键 1 中断到来（级别比计时器中断高），那么处理器会暂停处理计时器中断，开始处理按键 1 中断。
- 但是突然又有另外一个按键 2 中断到来（级别比按键 1 中断更高），那么处理器会暂停处理按键 1 的中断，开始处理按键 2 中断。
- 之后再没有其他更高级别的中断到来，则按键 2 中断不会再被打断，处理器能够顺利处理完毕按键 2 的中断，然后重新回到按键 1 中断的处理程序中去，完成按键 1 中断的处理。
- 完成按键 1 中断的处理之后，处理器会重新回到计时器中断的处理程序中去，完成计时器中断的处理。

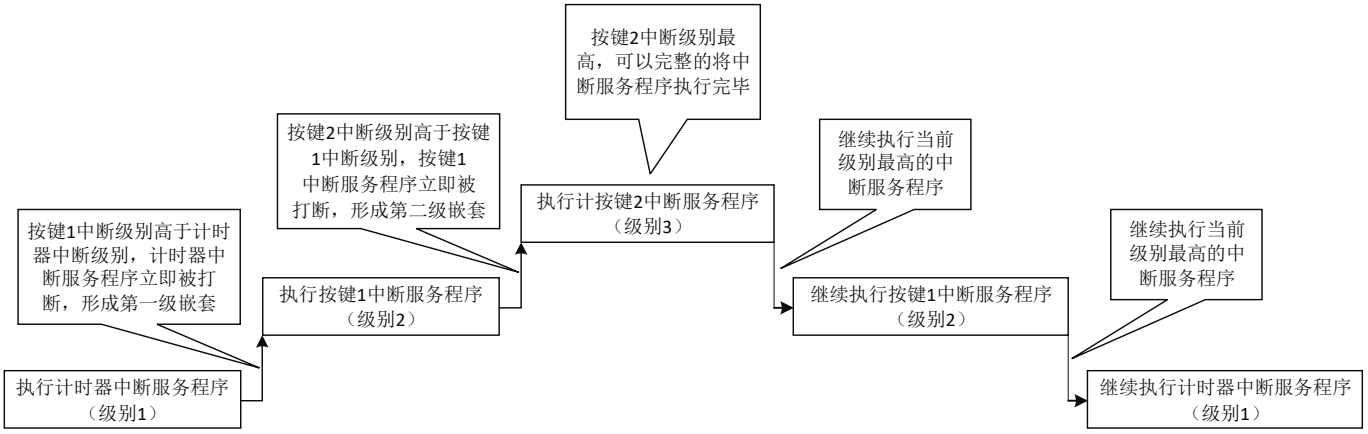

图 5-7 中断嵌套示意图

注意：假设新来的中断请求的优先级比正在处理的中断级别低（或者相同），则处理器不应该响应这个新的中断请求，处理器必须完成当前的中断服务程序之后才考虑响应新的中断请求（因为新中断请求的级别并不比当前正在处理的中断级别高）。有关中断级别的设定请参见第 6.2.9 节了解更多信息。

在 Bumblebee 内核中，取决于中断是向量处理模式还是非向量处理模式，中断嵌套的支持方法会有所差异，请参见第 5.13 节了解更多详细介绍。

5.12. 中断咬尾

处理器内核正在处理某个中断的过程中，可能有新中断请求到来，但是“新中断的级别”低于或者等于“当前正在处理的中断级别”，因此，新中断不能够打断当前正在处理的中断（因此不会形成嵌套）。

当处理器完成当前中断之后，理论上需要恢复上下文，然后退出中断回到主应用程序，然后重新响应新的中断，响应新的中断又需要再次保存上下文。因此，存在着一次背靠背的“恢复上下文”和“保存上下文”操作，如果将此背靠背的“恢复上下文”和“保存上下文”省略掉，则称之为“中断咬尾”，如图 5-8 中所示，显而易见，中断咬尾可以加快多个中断的背靠背处理速度。

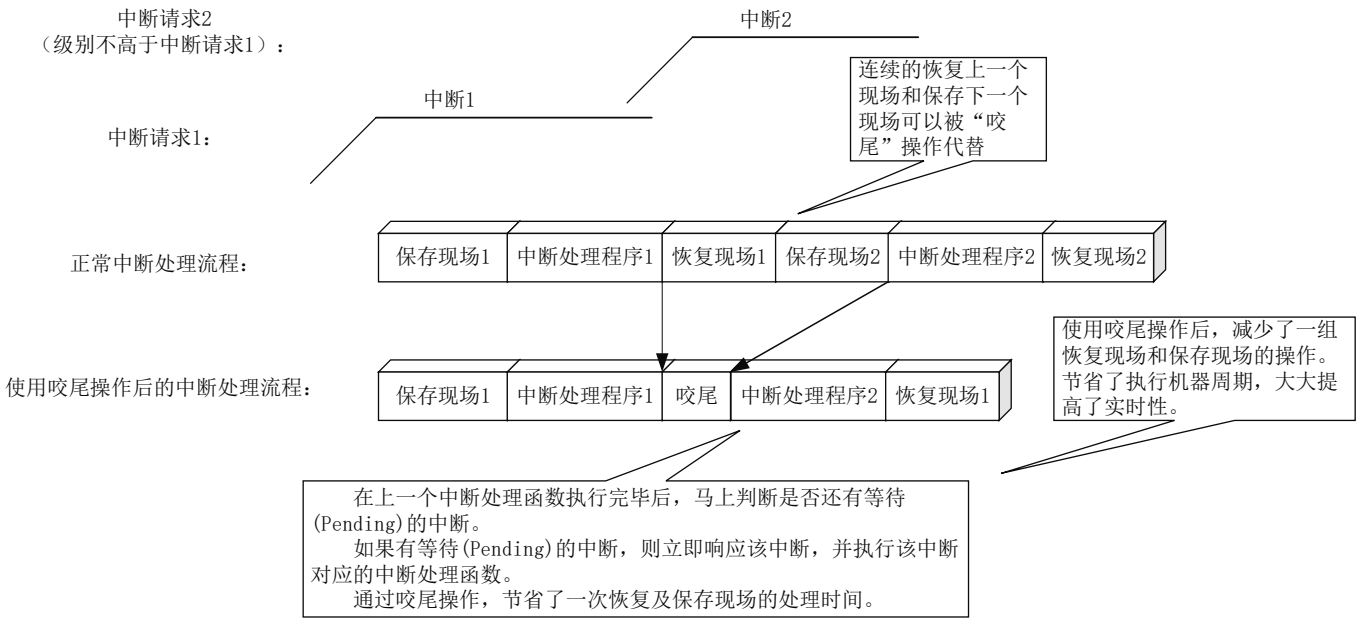

图 5-8 中断咬尾示意图

在 Bumblebee 内核中，只有非向量处理模式才支持中断咬尾，请参见第 5.13.1.3 节了解更多详细介绍。

5.13. 中断的向量处理模式和非向量处理模式

如第 6.2.10 节中所述，ECLIC 的每个中断源均可以设置成向量或者非向量处理（通过寄存器 clicintattr[i]的 shv 域），向量处理模式和非向量处理模式二者有较大的差别，分别介绍如下。

5.13.1. 非向量处理模式

5.13.1.1 非向量处理模式的特点和延迟

如果被配置成为非向量处理模式，则该中断被处理器内核响应后，处理器会直接跳入到所有非向量中断共享的入口地址，该入口地址可以通过软件进行设置：

- 如果配置 CSR 寄存器 mtvt2 的最低位为 0（上电复位默认值），则所有非向量中断共享的入口地址由 CSR 寄存器 mtvec 的值（忽略最低 2 位的值）指定。由于 mtvec 寄存器的值也指定异常的入口地址，因此，意味着在这种情况下，异常和所有非向量中断共享入口地

址。

- 如果配置 CSR 寄存器 mtvt2 的最低位为 1，则所有非向量中断共享的入口地址由 CSR 寄存器 mtvt2 的值（忽略最低 2 位的值）指定。为了让中断以尽可能快的速度被响应和处理，推荐将 CSR 寄存器 mtvt2 的最低位设置为 1，即，由 mtvt2 指定一个独立的入口地址供所有非向量中断专用，和异常的入口地址（由 mtvec 的值指定）彻底分开。

在进入所有非向量中断共享的入口地址之后，处理器会开始执行一段共有的软件代码，如图 5-9 中所示的例子，这段软件代码内容通常如下：

- 首先保存 CSR 寄存器 mepc、mcause、msubm 入堆栈。保存这几个 CSR 寄存器是为了保证后续的中断嵌套能够功能正确，因为新的中断响应会重新覆盖 mepc、mcause、msubm 的值，因此需要将它们先保存入堆栈。
- 保存若干通用寄存器（处理器的上下文）入堆栈。
- 然后执行一条特殊的指令“csrrw ra， CSR_JALMNXTI， ra”。如果没有中断在等待（Pending），则该指令相当于是个 Nop 指令不做任何操作；如果有中断在等待（Pending），执行该指令后处理器会：
  - 直接跳入该中断的向量入口（Vector Table Entry）存储的目标地址，即该中断源的中断服务程序（Interrupt Service Routine，ISR）中去。
  - 在跳入中断服务程序的同时，硬件也会同时打开中断的全局使能，即，设置 mstatus 寄存器的 MIE 域为 1。打开中断全局使能后，新的中断便可以被响应，从而达到中断嵌套的效果。
  - 在跳入中断服务程序的同时，“csrrw ra， CSR_JALMNXTI， ra”指令还会达到 JAL（Jump and Link）的效果，硬件同时更新 Link 寄存器的值为该指令的 PC 自身作为函数调用的返回地址。因此，从中断服务程序函数返回后会回到该“csrrw ra， CSR_JALMNXTI， ra”指令重新执行，重新判断是否还有中断在等待（Pending），从而达到中断咬尾的效果。
  - 在中断服务程序的结尾处同样需要添加对应的恢复上下文出栈操作。并且在 CSR 寄存器 mepc、mcause、msubm 出堆栈之前，需要将中断全局使能再次关闭，以保证 mepc、mcause、msubm 恢复操作的原子性（不被新的中断所打断）。

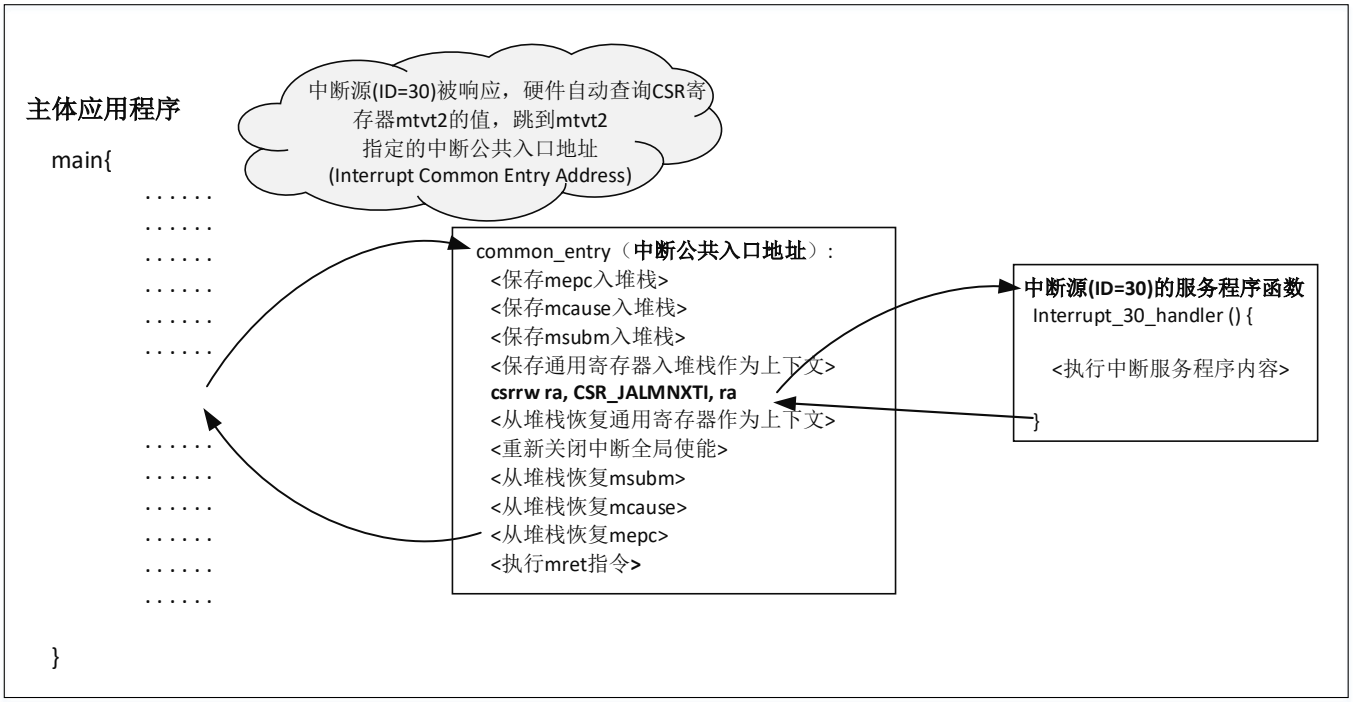

图 5-9 中断的非向量处理模式示例（总是支持嵌套）

由于非向量处理模式时处理器在跳到中断服务程序之前需要先执行一段共有的软件代码进行上下文的保存，因此，从中断源拉高到处理器开始执行中断服务程序中的第一条指令，需要经历以下几个方面的时钟周期开销：

- 处理器内核响应中断后进行跳转的开销。理想情况下约 4 个时钟周期。
- 处理器内核保存 CSR 寄存器 mepc、mcause、msubm 入堆栈的开销。
- 处理器内核保存上下文所花费的周期开销。如果是 RV32E 的架构，则需要保存 8 个通用寄存器，如果是 RV32I 的架构，则需要保存 16 个通用寄存器。
- 处理器内核跳转到中断服务程序（Interrupt Service Routine，ISR）中去的开销。理想情况下约需要 5 个时钟周期。

5.13.1.2 非向量处理模式的中断嵌套

如上文所述，非向量处理模式总是能够支持中断嵌套，如图 5-10 中所示的示例：假设中断源 30、31、32 这三个中断源先后到来，且“中断源 32 的级别” > “中断源 31 的级别”> “中断源 30 的级别”，那么后来的中断便会打断之前正在处理的中断形成中断嵌套。

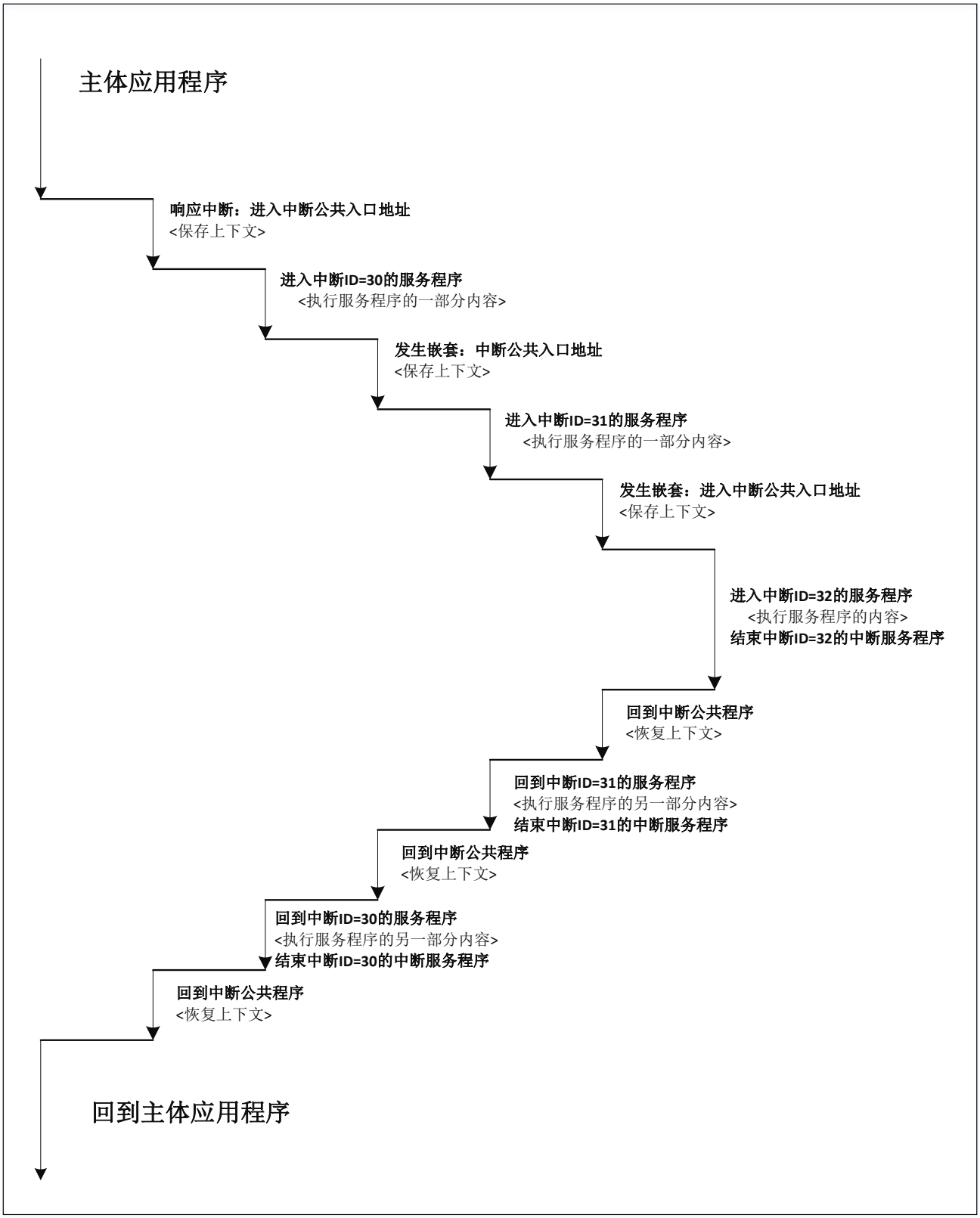

图 5-10 三个先后到来的（非向量处理模式）中断形成嵌套

5.13.1.3 非向量处理模式的中断咬尾

对于非向量处理模式的中断而言，由于在跳入和退出中断服务程序之前，处理器要进行上下文的保存和恢复，因此进行“中断咬尾”能够节省显著的时间（节省一次背靠背的保存上下文和恢复上下文）。

如上文所述，在所有非向量中断共享的共有代码段中，在跳入中断服务程序的同时，“csrrw ra，CSR_JALMNXTI， ra”指令还会达到 JAL（Jump and Link）的效果，硬件同时更新 Link 寄存器的值为该指令的 PC 自身作为函数调用的返回地址。因此，从中断服务程序函数返回后会回到该“csrrw ra， CSR_JALMNXTI， ra”指令重新执行，重新判断是否还有中断在等待（Pending），从而达到中断咬尾的效果。

如图 5-11 中所示的示例：假设中断源 30、29、28 这三个中断源先后到来，且“中断源 30 的级别” >= “中断源 29 的级别”>= “中断源 28 的级别”，那么后来的中断不会打断之前正在处理的中断（不会形成中断嵌套），但是会被置于等待（Pending）状态。当中断源 30 完成处理后，将会直接开始中断源 29 的中断处理，省掉中间的“恢复上下文”和“保存上下文”过程。

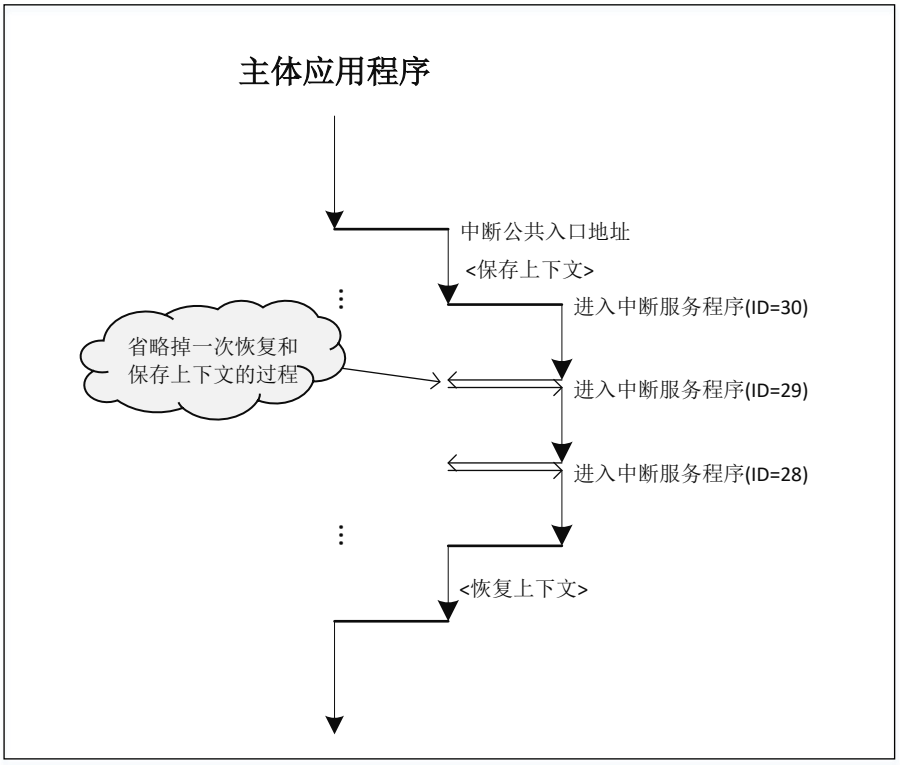

图 5-11 中断咬尾示意图

5.13.2. 向量处理模式

5.13.2.1 向量处理模式的特点和延迟

如果被配置成为向量处理模式，则该中断被处理器内核响应后，处理器会直接跳入该中断的向量入口（Vector Table Entry）存储的目标地址，即该中断源的中断服务程序（Interrupt Service Routine，ISR），如图 5-12 中所示的例子。

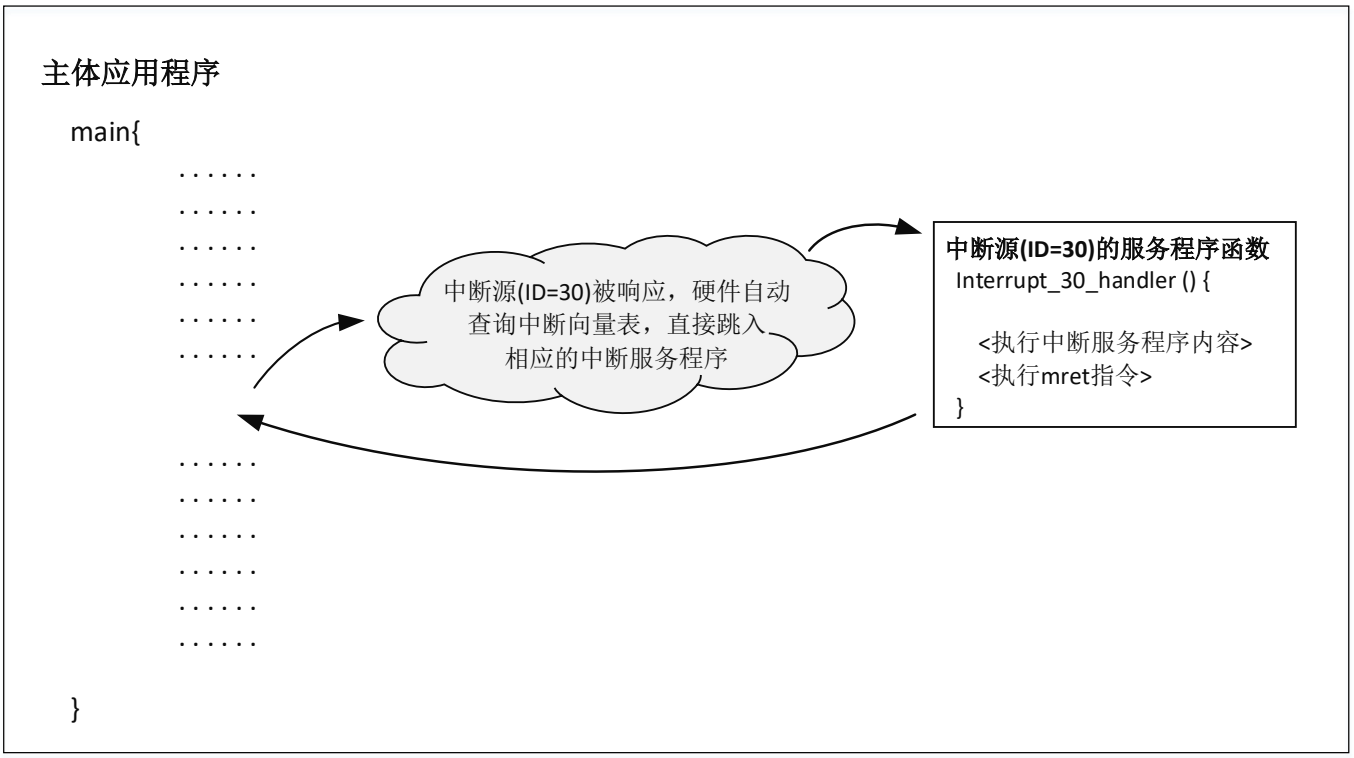

图 5-12 中断的向量处理模式示例

向量处理模式具有如下特点：

- 向量处理模式时处理器会直接跳到中断服务程序，并没有进行上下文的保存，因此，中断响应延迟非常之短，从中断源拉高到处理器开始执行中断服务程序中的第一条指令，基本上只需要硬件进行查表和跳转的时间开销，理想情况下约 6 个时钟周期。
- 对于向量处理模式的中断服务程序函数，一定要使用特殊的__attribute__((interrupt))来修饰中断服务程序函数。

- 向量处理模式时，由于在跳入中断服务程序之前，处理器并没有进行上下文的保存，因此，理论上中断服务程序函数本身不能够进行子函数的调用（即，必须是 Leaf Function）。
  - 如果中断服务程序函数不小心调用了其他的子函数（不是 Leaf Function），如果不加处理则会造成功能的错误。为了规避这种不小心造成的错误情形，只要使用了特殊的 __attribute__ （（interrupt）） 来修饰该中断服务程序函数，那么编译器会自动的进行判断，当编译器发现该函数调用了其他子函数时，便会自动的插入一段代码进行上下文的保存。注意：这种情况下虽然保证了功能的正确性，但是由于保存上下文造成的开销，又会事实上还是增大中断的响应延迟（与非向量模式相当）并且造成代码尺寸（Code Size）的膨胀。因此，在实践中，如果使用向量处理模式，那么不推荐在向量处理模式的中断服务程序函数中调用其他的子函数。
- 向量处理模式时，由于在跳入中断服务程序之前，处理器并没有进行任何特殊的处理，且由于处理器内核在响应中断后，mstatus 寄存器中的 MIE 域将会被硬件自动更新成为 0（意味着中断被全局关闭，从而无法响应新的中断）。因此向量处理模式默认是不支持中断嵌套的，为了达到向量处理模式且又能够中断嵌套的效果，如图 5-13 中所示，需要在中断服务程序的开头处添加特殊的入栈操作：
  - 首先保存 CSR 寄存器 mepc、mcause、msubm 入堆栈。保存这几个 CSR 寄存器是为了保证后续的中断嵌套能够功能正确，因为新的中断响应会重新覆盖 mepc、mcause、msubm 的值，因此需要将它们先保存入堆栈。
  - 重新打开中断的全局使能，即，设置 mstatus 寄存器的 MIE 域为 1。打开中断全局使能后，新的中断便可以被响应，从而达到中断嵌套的效果。
  - 在中断服务程序的结尾处同样需要添加对应的恢复上下文出栈操作。并且在 CSR 寄存器 mepc、mcause、msubm 出堆栈之前，需要将中断全局使能再次关闭，以保证 mepc、mcause、msubm 恢复操作的原子性（不被新的中断所打断）。

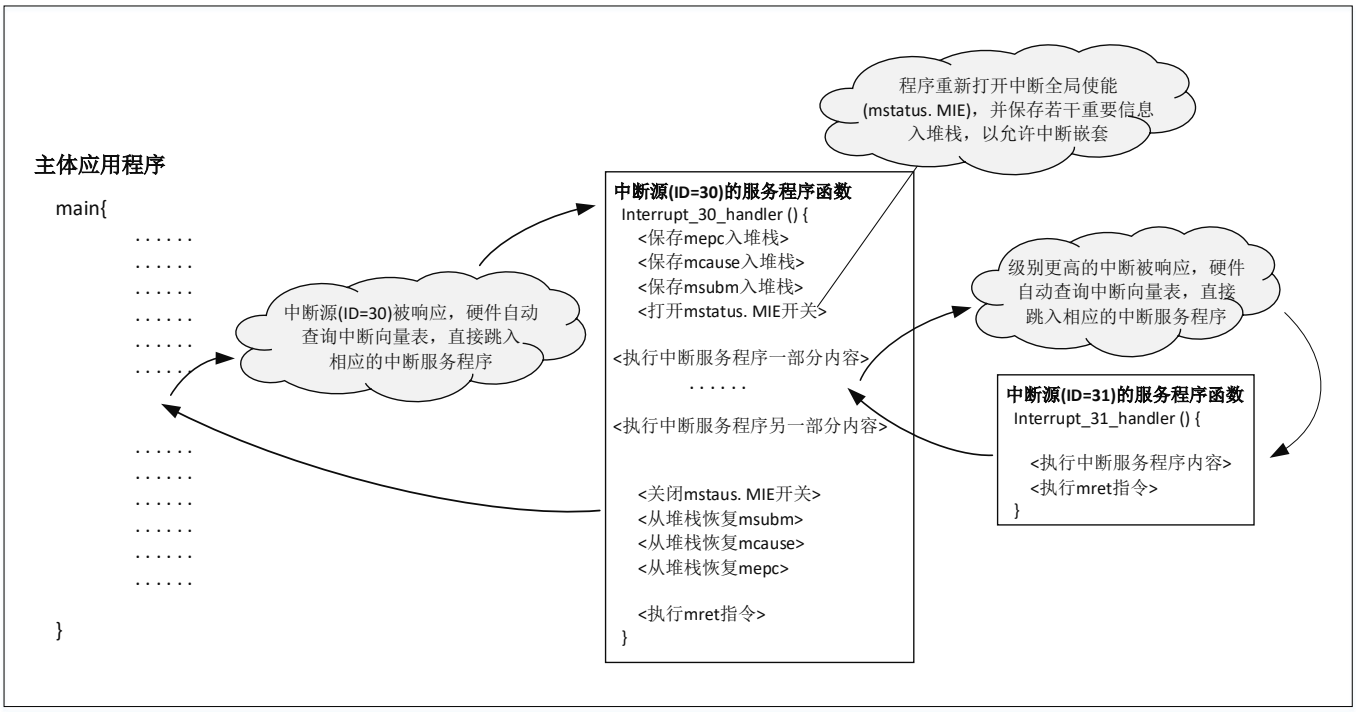

图 5-13 中断的向量处理模式示例（支持中断嵌套）

5.13.2.2 向量处理模式的中断嵌套

如上文所述，向量处理模式的中断经过特殊处理也可以支持中断嵌套，如图 5-14 中所示的示例：假设中断源 30、31、32 这三个中断源先后到来，且“中断源 32 的级别” > “中断源 31 的级别”> “中断源 30 的级别”，那么后来的中断便会打断之前正在处理的中断形成中断嵌套。

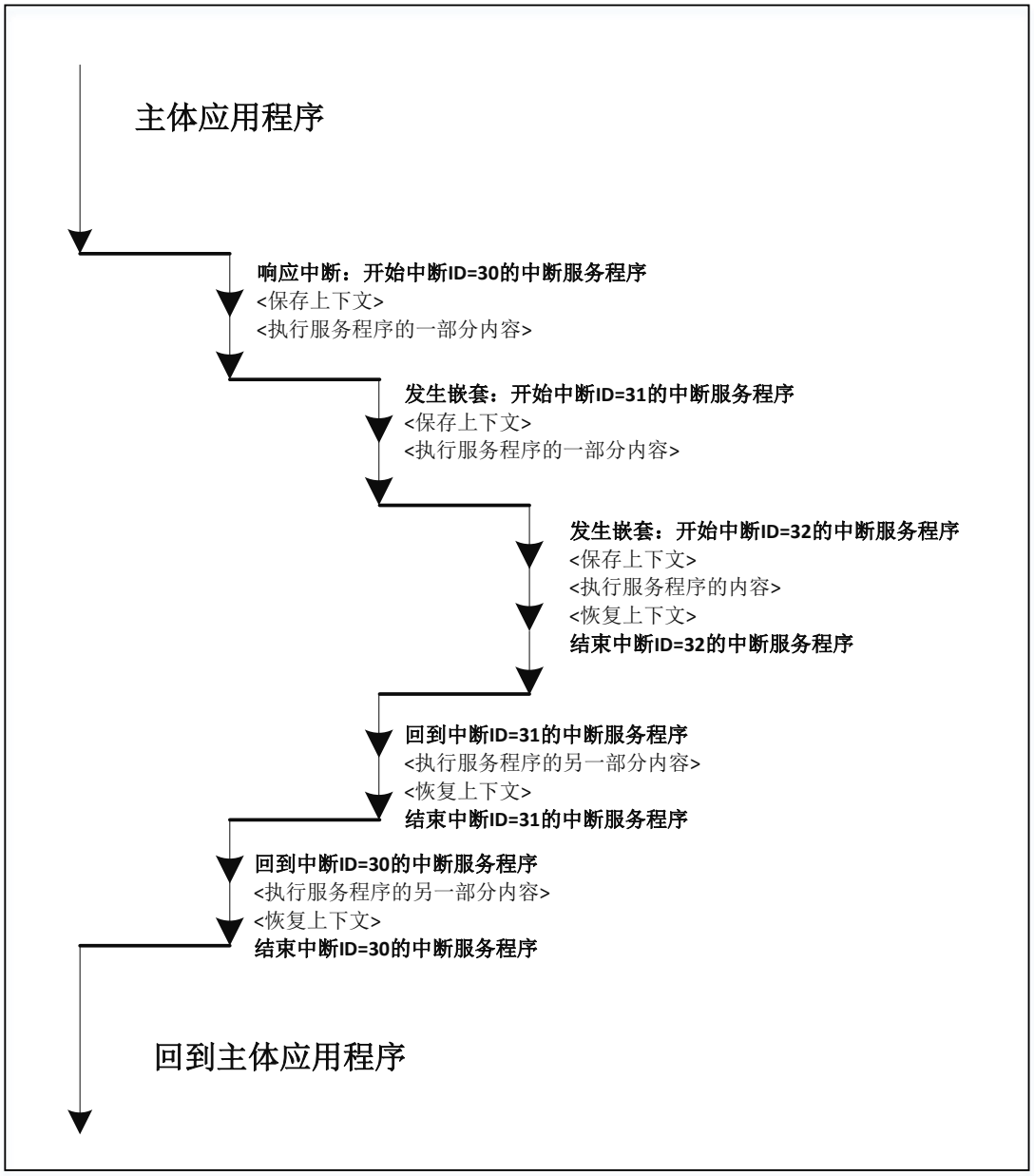

图 5-14 三个先后到来的（向量处理模式）中断形成嵌套

5.13.2.3 向量处理模式的中断咬尾

对于向量处理模式的中断而言，由于在跳入中断服务程序之前，处理器并没有进行上下文的保存，因此进行“中断咬尾”的意义不大，因此，向量处理模式的中断，没有“中断咬尾”处理能力。

6. Bumblebee 内核 TIMER 和 ECLIC 介绍

6.1. TIMER 介绍

6.1.1. TIMER 简介

计时器单元（Timer Unit，TIMER），在 Bumblebee 内核中主要用于产生计时器中断（Timer Interrupt）和软件中断（Software Interrupt）。请参见第 5.3.2.1 节和第 5.3.2.2 节了解计时器中断与软件中断的详细信息。

6.1.2. TIMER 寄存器

TIMER 是一个存储器地址映射的单元：

- TIMER 单元在 Bumblebee 内核中的基地址请参见《Bumblebee 内核简明数据手册》中的介绍。
- TIMER 单元内寄存器和地址偏移量如表 6-1 中所示。

表 6-1 TIMER 寄存器的存储器映射地址

模块内偏移地址	读写属性	寄存器名称	复位默认值	功能描述
0x0	可读可写	mtime_lo	0x00000000	反映计时器 mtime 的低 32 位值，参见第 6.1.3 节了解其详细介绍。
0x4	可读可写	mtime_hi	0x00000000	反映计时器 mtime 的高 32 位值，参见第 6.1.3 节了解其详细介绍。
0x8	可读可写	mtimecmp_lo	0xFFFFFFFF	配置计时器的比较值 mtimecmp 低 32 位，参见第 6.1.5 节了解其详细介绍。
0xC	可读可写	mtimecmp_hi	0xFFFFFFFF	配置计时器的比较值 mtimecmp 高 32 位，参见第 6.1.5 节了解其详细介绍。
0xFF8	可读可写	mstop	0x00000000	控制计时器的暂停，参见第 6.1.4 节了解其详细介绍。
0xFFC	可读可写	msip	0x00000000	生成软件中断，参见第 6.1.6 节了解其详细介绍。
注意： ■ TIMER 的寄存器只支持操作尺寸（Size）为 word 的对齐读写访问。 ■ TIMER 的寄存器区间为 0x00 ~ 0xFF，除了上表中列出的寄存器之外的其他地址内的值为常数 0。				

下文对各寄存器的功能和使用进行详细描述。

6.1.3. 通过 mtime 寄存器进行计时

TIMER 可以用于实时计时，要点如下：

- TIMER 中实现了一个 64 位的 mtime 寄存器，由{mtime_hi，mtime_lo}拼接而成，该寄存器反映了 64 位计时器的值。计时器根据低速的输入节拍信号进行自增计数，计时器默认是打开的，因此会一直进行计数。
- 在 Bumblebee 内核中，此计数器的自增频率由处理器的输入信号 mtime_toggle_a 控制，请参见文档《Bumblebee 内核简明数据手册》了解该输入信号的详情。

6.1.4. 通过 mstop 寄存器暂停计时器

由于 TIMER 的计时器上电后默认会一直进行自增计数，为了在某些特殊情况下关闭此计时器计数，TIMER 中实现了一个 mstop 寄存器。如表 6-2 中所示，mstop 寄存器只有最低位为有效位，该有效位直接作为计时器的暂停控制信号，因此，软件可以通过将 mstop 寄存器设置成 1 将计时器暂停计数。

表 6-2 寄存器 mstop 的比特域

域名	比特位	属性	复位值	描述
Reserved	7:1	只读，写忽略	N/A	未使用的域，值为常数 0
TIMESTOP	0	可读可写	0	控制计时器运行或者暂停。如果该域的值为 1，则计时器暂停计数，否则正常自增计数。

6.1.5. 通过 mtime 和 mtimecmp 寄存器生成计时器中断

TIMER 可以用于生成计时器中断，要点如下：

- TIMER 中实现了一个 64 位的 mtimecmp 寄存器，由{mtimecmp_hi，mtimecmp_lo}拼接而成，该寄存器作为计时器的比较值，假设计时器的值 mtime 大于或者等于 mtimecmp 的值，则产生计时器中断。软件可以通过改写 mtimecmp 或者 mtime 的值（使得 mtimecmp 大于 mtime 的值）来清除计时器中断。

注意：计时器中断是连接到 ECLIC 单元进行统一管理，有关 ECLIC 的详情请参见第 6.2 节。

6.1.6. 通过 msip 寄存器生成软件中断

TIMER 可以用于生成软件中断。TIMER 中实现了一个 msip 寄存器，如表 6-3 中所示，msip 寄存器只有最低位为有效位，该有效位直接作为软件中断，因此：

- 软件写通过写 1 至 msip 寄存器产生软件中断；
- 软件可通过写 0 至 msip 寄存器来清除该软件中断。

注意：软件中断是连接到 ECLIC 单元进行统一管理，有关 ECLIC 的详情请参见第 6.2 节。

表 6-3 寄存器 msip 的比特域

域名	比特位	属性	复位值	描述
Reserved	7:1	只读，写忽略	N/A	未使用的域，值为常数 0
MSIP	0	可读可写	0	该域用于产生软件中断

6.2. ECLIC 介绍

Bumblebee 内核支持在 RISC-V 标准 CLIC 基础上优化而来的“改进型内核中断控制器（Enhanced Core Local Interrupt Controller，ECLIC）”，用于管理所有的中断源。

注意：

- ECLIC 只服务于一个处理器内核，为该处理器内核私有。
- ECLIC 的软件编程模型也向后兼容标准的 CLIC。

6.2.1. ECLIC 简介

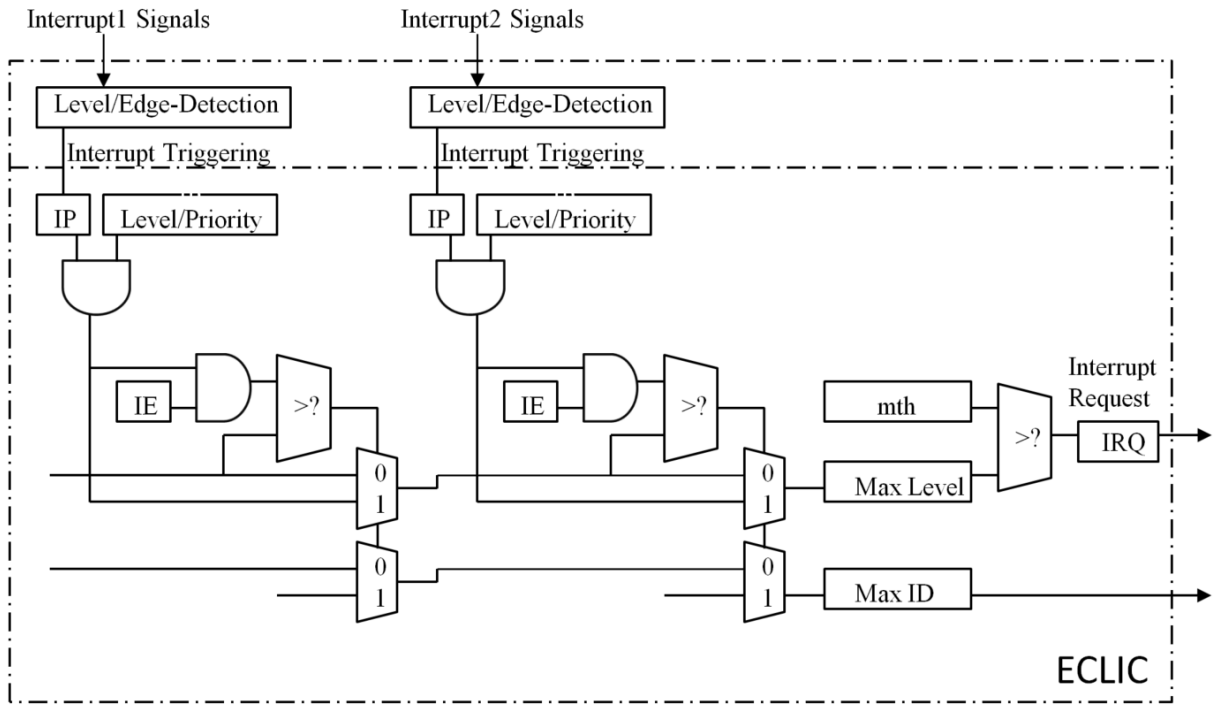

图 6-1 ECLIC 逻辑结构示意图

ECLIC 用于对多个内部和外部中断源进行仲裁、发送请求并支持中断嵌套。ECLIC 的寄存器如表 6-5 所述，逻辑结构如图 6-1 所示，相关概念如下：

- ECLIC 中断目标
- ECLIC 中断源
- ECLIC 中断源的编号
- ECLIC 的寄存器
- ECLIC 中断源的使能位
- ECLIC 中断源的等待标志位
- ECLIC 中断源的电平或边沿属性
- ECLIC 中断源的级别和优先级

- ECLIC 中断源的向量或非向量处理
- ECLIC 中断目标的阈值级别
- ECLIC 中断的仲裁机制
- ECLIC 中断的响应、嵌套、咬尾机制

下文将分别予以详述。

6.2.2. ECLIC 中断目标

ECLIC 单元生成一根中断线，发送给处理器内核（作为中断目标），其关系结构如图 6-2 所示。

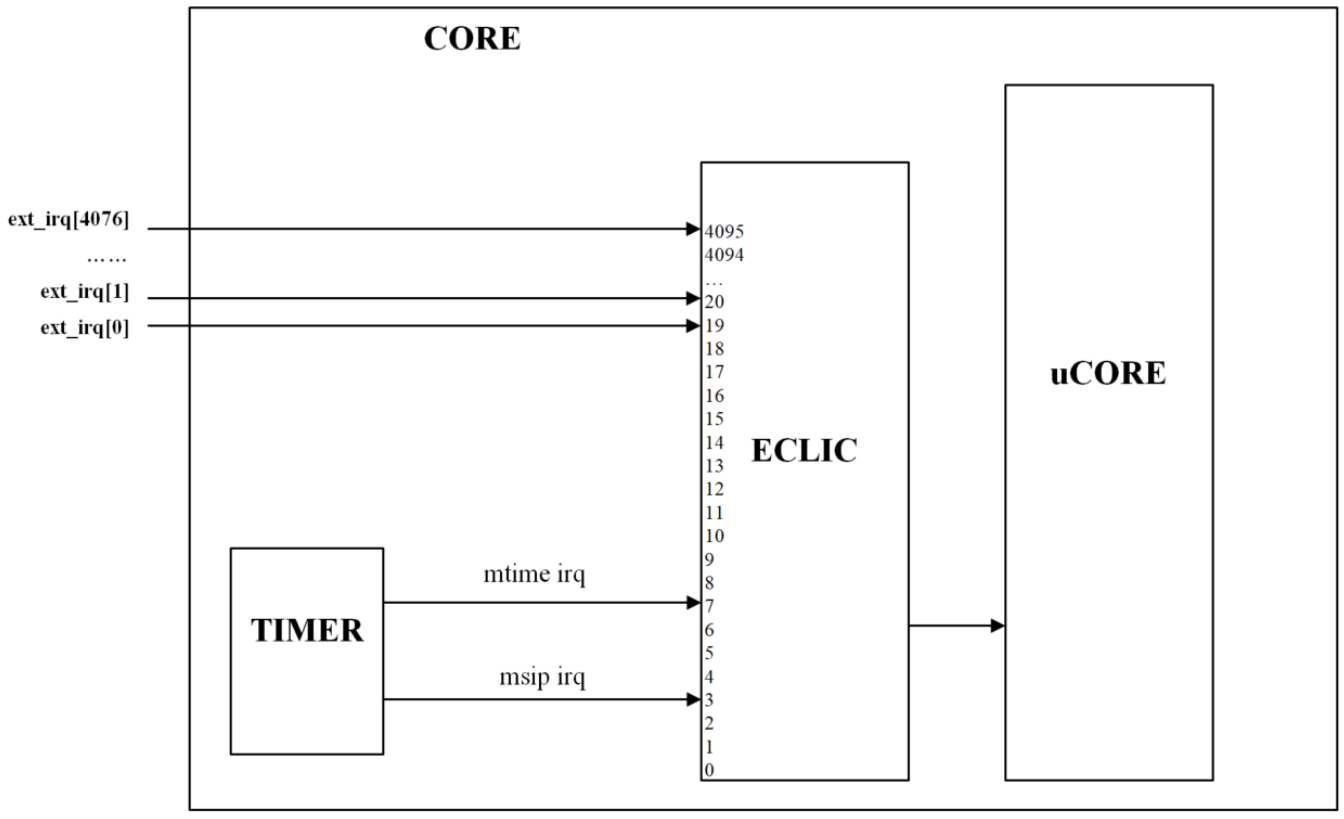

图 6-2 ECLIC 关系结构图

6.2.3. ECLIC 中断源

如图 6-2 所示，ECLIC 理论上从编程模型上可以支持多达 4096 个中断源（Interrupt Source）。ECLIC 为每个中断源定义了如下特性和参数：

- 编号（ID）
- 使能位（IE）
- 等待标志位（IP）
- 电平或边沿属性（Level or Edge-Triggered）
- 级别和优先级（Level and Priority）
- 向量或非向量处理（Vector or Non-Vector Mode）

下文分别予以介绍。

6.2.4. ECLIC 中断源的编号（ID）

ECLIC 为每个中断源分配了一个独一无二的编号（ID）。譬如，假设某 ECLIC 的硬件实现真正支持 4096 个 ID，则 ID 应为 0 至 4095。注意：

- 在 Bumblebee 内核中，中断 ID 编号 0 至 18 的中断被预留作为了内核特殊的内部中断。
- 普通外部中断分配的中断源 ID 从 19 开始，用户可以用于连接外部中断源。

详细介绍如表 6-4 中所示。

表 6-4 ECLIC 中断源编号和分配

ECLIC 中断编号	功能	中断源介绍
0	*预留*	*Bumblebee 内核没有使用该中断。*
1	*预留*	*Bumblebee 内核没有使用该中断。*
2	*预留*	*Bumblebee 内核没有使用该中断。*
3	软件中断	Bumblebee 内核的 TIMER 单元生成的软件中断。
4	*预留*	*Bumblebee 内核没有使用该中断。*
5	*预留*	*Bumblebee 内核没有使用该中断。*
6	*预留*	*Bumblebee 内核没有使用该中断。*
7	计时器中断	Bumblebee 内核的 TIMER 单元生成的计时器中断。
8	*预留*	*Bumblebee 内核没有使用该中断。*

9	*预留*	*Bumblebee 内核没有使用该中断。*
10	*预留*	*Bumblebee 内核没有使用该中断。*
11	*预留*	*Bumblebee 内核没有使用该中断。*
12	*预留*	*Bumblebee 内核没有使用该中断。*
13	*预留*	*Bumblebee 内核没有使用该中断。*
14	*预留*	*Bumblebee 内核没有使用该中断。*
15	*预留*	*Bumblebee 内核没有使用该中断。*
16	*预留*	*Bumblebee 内核没有使用该中断。*
17	存储器访问错误中断	Bumblebee 内核存储器访问错误转化成为的内部中断。
18	*预留*	*Bumblebee 内核没有使用该中断。*
19 ~ 4095	外部中断	普通外部中断供用户连接使用。注意： ■ 虽然 ECLIC 从编程模型上支持最多 4096 个中断源，但是实际硬件支持的中断源数目反映在信息寄存器 clicinfo.NUM_INTERRUPT 中。

6.2.5. ECLIC 的寄存器

ECLIC 是一个存储器地址映射的单元：

- ECLIC 单元在 Bumblebee 内核中的基地址请参见《Bumblebee 内核简明数据手册》中的介绍。
- ECLIC 单元内的寄存器和地址偏移量如表 6-5 中所示。

表 6-5 ECLIC 寄存器的单元内地址偏移量

	属性	名称	宽度
0x0000	可读可写	cliccfg	8 位
0x0004	只读，写忽略	clicinfo	32 位
0x000b	可读可写	mth	8 位
0x1000+4*i	可读可写	clicintip[i]	8 位
0x1001+4*i	可读可写	clicintie[i]	8 位
0x1002+4*i	可读可写	clicintattr[i]	8 位
0x1003+4*i	可读可写	clicintctl[i]	8 位
注意： ■ 上述的 i 表示中断的 ID 编号，带有[i]后缀的寄存器表示这是针对每个中断源会有一份独立的寄存器。 ■ ECLIC 的寄存器支持操作尺寸（Size）为 byte、half-word、或 word 的对齐读写访问。 ■ 向上述“只读”寄存器进行写操作会被忽略，但是不会产生总线出错异常。			

- 实际的 ECLIC 可能没有配置 4096 个中断源，那么不存在的中断源对应寄存器的值为常数 0。
- ECLIC 单元内寄存器区间为 0x0000 ~ 0xFFFF，除了上表中列出的寄存器之外的其他地址内的值为常数 0。

下文对各个寄存器进行详细介绍。

6.2.5.1 寄存器 cliccfg

cliccfg 寄存器是全局性的配置寄存器，软件可以通过改写该寄存器配置若干全局性的参数，其具体比特域的信息请参见表 6-6 中所示。

表 6-6 寄存器 cliccfg 的比特域

域名	比特位	属性	复位值	描述
Reserved	7:5	只读，写忽略	N/A	未使用的域，值为常数 0
nlbits	4:1	可读可写	0	用于指定 clicintctl[i]寄存器中 Level 域的比特数，参见第 6.2.9 节了解其详细介绍。
Reserved	0	只读，写忽略	N/A	未使用的域，值为常数 1

6.2.5.2 寄存器 clicinfo

clicinfo 寄存器是全局性的信息寄存器，软件可以通过读该寄存器查看若干全局性的参数，其具体比特域的信息请参见表 6-7 中所示。

表 6-7 寄存器 clicinfo 的比特域

域名	比特位	属性	复位值	描述
Reserved	31:25	只读，写忽略	N/A	未使用的域，值为常数 0
CLICINTCTLBITS	24:21	只读，写忽略	N/A	用于指定 clicintctl[i]寄存器中有效位的比特数，参见第 6.2.9 节了解其详细介绍。
VERSION	20:13	只读，写忽略	N/A	硬件实现的版本号
NUM_INTERRUPT	12:0	只读，写忽略	N/A	硬件支持的中断源数目

6.2.5.1 寄存器 mth

mth 寄存器是中断目标的阈值级别寄存器，软件可以通过改写该寄存器配置中断目标的阈值级别，其具体比特域的信息请参见表 6-8 中所示。

表 6-8 寄存器 mth 的比特域

域名	比特位	属性	复位值	描述
mth	7:0	可读可写	N/A	中断目标的阈值级别寄存器，参见第 6.2.11 节了解其详细介绍。

6.2.5.2 寄存器 clicintip[i]

clicintip[i]寄存器是中断源的等待标志寄存器，其具体比特域的信息请参见表 6-9 中所示。

表 6-9 寄存器 clicintip[i]的比特域

域名	比特位	属性	复位值	描述
Reserved	7:1	只读，写忽略	N/A	未使用的域，值为常数 0
IP	0	可读可写	0	中断源的等待标志位，参见第 6.2.7 节了解其详细介绍。

6.2.5.3 寄存器 clicintie[i]

clicintie[i]寄存器是中断源的使能寄存器，其具体比特域的信息请参见表 6-10 中所示。

表 6-10 寄存器 clicintip[i]的比特域

域名	比特位	属性	复位值	描述
Reserved	7:1	只读，写忽略	N/A	未使用的域，值为常数 0
IE	0	可读可写	0	中断源的使能位，参见第 6.2.6 节了解其详细介绍。

6.2.5.4 寄存器 clicintattr[i]

clicintattr[i]寄存器是中断源的属性寄存器，软件可以通过改写该寄存器配置中断源的若干属性，其具体比特域的信息请参见表 6-11 中所示。

表 6-11 寄存器 clicintattr[i]的比特域

域名	比特位	属性	复位值	描述
Reserved	7:6	只读，写忽略	N/A	未使用的域，值为常数 3
Reserved	5:3	只读，写忽略	N/A	未使用的域，值为常数 0

trig	2:1	可读可写	0	指定该中断源的电平或边沿属性，参见第 6.2.8 节了解其详细介绍。
shv	0	可读可写	0	指定该中断源使用向量处理模式还是非向量处理模式，参见第 6.2.10 节了解其详细介绍。

6.2.5.5 寄存器 clicintctl[i]

clicintctl[i]寄存器是中断源的控制寄存器，软件可以通过改写该寄存器配置中断源的级别（Level）和优先级（Priority），Level 和 Priority 域根据 cliccfg.nlbits 的值动态进行分配，参见第 6.2.9 节了解其详细介绍。

6.2.6. ECLIC 中断源的使能位（IE）

如图 6-2 所示，ECLIC 为每个中断源分配了一个中断使能位（IE），反映在寄存器 clicintie[i].IE 中，其功能如下：

- 每个中断源的 clicintie[i]寄存器是存储器地址映射的可读可写寄存器，从而使得软件可以对其编程。
- 如果 clicintie[i]寄存器被编程配置成为 0，则意味着此中断源被屏蔽。
- 如果 clicintie[i]寄存器被编程配置成为 1，则意味着此中断源被打开。

6.2.7. ECLIC 中断源的等待标志位（IP）

如图 6-2 所示，ECLIC 为每个中断源分配了一个中断等待标志位（IP），反映在寄存器 clicintip[i].IP 中，其功能如下：

- 如果某个中断源的 IP 位为高，则表示该中断源被触发。中断源的触发条件取决于它是电平触发还是边沿触发的属性，请参见第 6.2.8 节的详细介绍。
- 中断源的 IP 位软件可读可写，软件写 IP 位的行为取决于它是电平触发还是边沿触发的属性，请参见第 6.2.8 节的详细介绍。
- 对于边沿触发的中断源，其 IP 还可能存在硬件自清的行为，请参见第 6.2.8 节的详细介绍。

6.2.8. ECLIC 中断源的电平或边沿属性（Level or Edge-Triggered）

如图 6-2 所示，ECLIC 的每个中断源均可以设置电平触发或者边沿触发的属性（通过寄存器 clicintattr[i]的 trig 域），其要点如下：

- 当 clicintattr[i].trig[0] == 0 时，设置该中断属性为电平触发的中断：
  - 如果该中断源被配置为电平触发，中断源的 IP 位会实时反映该中断源的电平值。
  - 如果该中断源被配置为电平触发，由于中断源的 IP 位实时反映该中断源的电平值，所以软件对该中断 IP 位的写操作会被忽略，即，软件无法通过写操作设置或者清除 IP 位的值。如果软件需要清除中断，只能通过清除中断的最终源头的方式进行。
- 当 clicintattr[i].trig[0] == 1 和 clicintattr[i].trig[1] == 0 时，设置该中断属性为上升沿触发的中断：
  - 如果该中断源被配置为上升沿触发，则 ECLIC 检测到该中断源的上升沿时，该中断源在 ECLIC 中被触发，该中断源的 IP 位被置高。
  - 如果该中断源被配置为上升沿触发，软件对该中断 IP 位的写操作会生效，即，软件可以通过写操作设置或者清除 IP 位的值。
  - 注意：对于上升沿触发的中断而言，为了能够提高中断处理的效率，当该中断被响应，处理器内核跳入中断服务程序（Interrupt Service Routines，ISR）之时，ECLIC 的硬件会自动清除该中断的 IP 位，从而无需软件在 ISR 内部对该中断的 IP 位进行清除。
- 当 clicintattr[i].trig[0] == 1 和 clicintattr[i].trig[1] == 1 时，设置该中断属性为下降沿触发的中断：
  - 如果该中断源被配置为下降沿触发，则 ECLIC 检测到该中断源的下降沿时，该中断源在 ECLIC 中被触发，该中断源的 IP 位被置高。
  - 如果该中断源被配置为下降沿触发，软件对该中断 IP 位的写操作会生效，即，软件可以通过写操作设置或者清除 IP 位的值。
  - 注意：对于下降沿触发的中断而言，为了能够提高中断处理的效率，当该中断被响应，处理器内核跳入中断服务程序（Interrupt Service Routines，ISR）之时，ECLIC 的硬件会自动清除该中断的 IP 位，从而无需软件在 ISR 内部对该中断的 IP 位进行清除。

6.2.9. ECLIC 中断源的级别和优先级（Level and Priority）

如图 6-2 所示，ECLIC 的每个中断源均可以设置特定的级别和优先级（通过寄存器 clicintctl[i]），

其要点如下：

- 每个中断源的 clicintctl[i]寄存器理论上有 8 位宽，其中硬件真正实现的位有效位数由 clicinfo 寄存器的 CLICINTCTLBITS 域来指定。譬如，假设 clicinfo.CLICINTCTLBITS 域的值为 6，则表示 clicintctl[i]寄存器只有高 6 位是真正的有效位，最低 2 位的值为常数 1，如图 6-3 中示例所示。
  - 注意：CLICINTCTLBITS 域的值是只读的固定常数，软件无法对其进行编程改写。其理论上的合理范围是 2 <= CLICINTCTLBITS <= 8，具体的实际值由处理器内核的硬件实现决定。
- 在 clicintctl[i]寄存器的有效位中，包含两个动态的域，分别用于指定该中断源的级别（Level）和优先级（Priority）。Level 域的宽度由 cliccfg 寄存器的 nlbits 域来指定。譬如，假设 cliccfg.nlbits 域的值为 4，则表示 clicintctl[i]寄存器有效位的高 4 位是 Level 域，其他的低位有效位为 Priority 域，如图 6-3 中示例所示。
  - 注意：cliccfg.nlbits 域的值是可读可写域，软件可以对其进行编程改写。

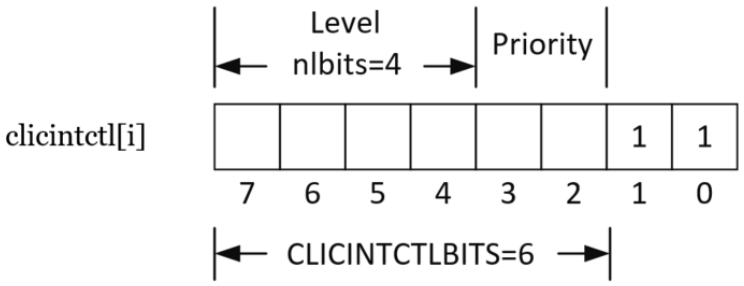

图 6-3 寄存器 clicintctl[i]的格式示例

- 中断源的级别（Level）相关的要点如下：
  - Level 的数字值采取左对齐的方式进行解读，有效位宽（由 cliccfg.nlbits 指定）之外的低位全部采用补常数 1 的方式填充，如图 6-4 中示例所示。
    - 注意：如果 cliccfg.nlbits > clicinfo.CLICINTCTLBITS，则意味着 nlbits 指示的位数超出了 clicintctl[i]寄存器的有效位，则超出的位全部采用补常数 1 的方式填充。
    - 注意：如果 cliccfg.nlbits = 0，Level 的数字值会被认为是固定的 255。如图 6-5 中示例所示。
  - Level 的数字值越大，则表示其级别越高，注意：

- 高级别的中断可以打断低级别的中断处理，从而形成中断嵌套，请参见第 5.11 节的详细介绍。
- 多个中断同时等待（IP 位为高），ECLIC 需要仲裁决定哪个中断被发送给内核进行处理，仲裁时需要参考每个中断源的 Level 数字值。请参见第 5.5 节的详细介绍。

#nlbits	编码	Level的数字值
1	L . . . . . (= L1111111)	127, 255
2	LL . . . . . (= LL111111)	63, 127, 191, 255
3	LLL . . . . . (= LLL11111)	31, 63, 95, 127, 159, 191, 223, 255
4	LLLL . . . . (= LLLL1111)	15, 31, 47, 63, 79, 95, 111, 127, 143, 159, 175, 191, 207, 223, 239, 255

“L” 比特表示Level的域
“.” 表示Level域之外的其他比特，采用全部补1的方式

图 6-4 Level 的数字值解读方式

cliccfg设置的若干示例:

CLICINTCTLBITS	nlbits	clicintctl[i]	Level的数字值
0	2	. . . . .	255
1	2	L	127, 255
2	2	LL	63, 127, 191, 255
3	3	LLL	31, 63, 95, 127, 159, 191, 223, 255
4	1	LPPP. . . .	127, 255

图 6-5 cliccfg 设置的若干示例

- 中断源的优先级（Priority）相关的要点如下：
  - Priority 的数字值也采取左对齐的方式进行解读，有效位宽（clicinfo.CLICINTCTLBITS -cliccfg.nlbits）之外的低位全部采用补常数 1 的方式填充。
  - Priority 的数字值越大，则表示其优先级越高，注意：

- 中断优先级（Priority）不参与中断嵌套的判断，即中断能否嵌套与中断优先级（Priority）的数值大小没有关系，而是与中断级别（Level）的数值大小有关。
- 多个中断同时 Pending 时，ECLIC 需要仲裁决定哪个中断被发送给内核进行处理，仲裁时需要参考每个中断源的 Priority 数字值。请参见第 6.2.12 节的详细介绍。

6.2.10. ECLIC 中断源的向量或非向量处理（Vector or Non-Vector Mode）

ECLIC 的每个中断源均可以设置成向量或者非向量处理（通过寄存器 clicintattr[i]的 shv 域），其要点如下：

- 如果被配置成为向量处理模式，则该中断被处理器内核响应后，处理器直接跳入该中断的向量入口（Vector Table Entry）存储的目标地址。有关中断向量处理模式的详细介绍，请参见第 5.13 节。
- 如果被配置成为非向量处理模式，则该中断被处理器内核响应后，处理器直接跳入所有中断共享的入口地址。有关中断非向量处理模式的详细介绍，请参见第 5.13 节。

6.2.11. ECLIC 中断目标的阈值级别

如图 6-1 中所示，ECLIC 可以设置特定的中断目标的阈值级别（mth），其要点如下：

- mth 寄存器是完整的 8 位寄存器，所有位均可读可写，软件可以通过写此寄存器配置目标阈值。注意：该阈值表征的是一种级别（Level）的数字值。
- ECLIC 最终仲裁出的中断的“级别（Level）数字值”只有高于“mth 寄存器中的值”，该中断才能够被发送给处理器内核。

6.2.12. ECLIC 中断的仲裁机制

如图 6-2 所示，ECLIC 对其所有中断源进行仲裁选择的原则如下：

- 只有满足下列所有条件的中断源才能参与仲裁：
  - 中断源的使能位（clicintie[i]寄存器）必须为 1。
  - 中断源的等待标志位（clicintip[i]寄存器）必须为 1。
- 从所有参与仲裁的中断源中进行仲裁的规则为：

- 首先判断级别（Level），Level 数字值越大的中断源，其仲裁优先级越高。
- 如果 Level 相等，则其次判断优先级（Priority），Priority 数字值越大的中断源，其仲裁优先级越高。
- 如果 Level 和 Priority 都相等，则再次判断判断中断 ID，中断 ID 越大的中断源，其仲裁优先级越高。

- 如果最后仲裁出的中断源的 Level 数字值高于中断目标的阈值级别（mth），则产生最终的中断请求，将通往处理器内核的中断请求信号拉高。

6.2.13. ECLIC 中断的响应、嵌套、咬尾机制

ECLIC 中断请求发送给处理器内核之后，处理器内核将对其进行响应。通过 ECLIC 和内核协同，可以支持中断嵌套、快速咬尾等等机制。请参见第 5.6 节、第 5.11 节、第 5.12 节的详细介绍。

7. Bumblebee 内核 CSR 寄存器介绍

7.1. Bumblebee 内核 CSR 寄存器概述

RISC-V 的架构中定义了一些控制和状态寄存器（Control and Status Register， CSR）， 用于配置或者记录一些运行的状态。CSR 寄存器是处理器核内部的寄存器，使用其专有的 12 位地址编码空间。

7.2. Bumblebee 内核的 CSR 寄存器列表

Bumblebee 内核支持的 CSR 寄存器列表如表 7-1 所示，其中包括 RISC-V 标准的 CSR 寄存器（RV32IMAC 架构支持 Machine Mode 和 User Mode 相关）和 Bumblebee 内核自定义扩展的 CSR 寄存器。

表 7-1 Bumblebee 内核支持的 CSR 寄存器列表

类型	CSR 地址	读写属性	名称	全称
RISC-V 标准 CSR（Machine Mode）	0xF11	MRO	mvendorid	商业供应商编号寄存器（Machine Vendor ID Register）
	0xF12	MRO	marchid	架构编号寄存器（Machine Architecture ID Register）
	0xF13	MRO	mimpid	硬件实现编号寄存器（Machine Implementation ID Register）
	0xF14	MRO	mhartid	Hart 编号寄存器（Hart ID Register）
	0x300	MRW	mstatus	异常处理状态寄存器
	0x301	MRO	misa	指令集架构寄存器（Machine ISA Register）
	0x304	MRW	mie	局部中断屏蔽控制寄存器（Machine Interrupt Enable Register）
	0x305	MRW	mtvec	异常入口基地址寄存器
	0x307	MRW	mtvt	ECLIC 中断向量表的基地址
	0x340	MRW	mscratch	暂存寄存器（Machine Scratch Register）
	0x341	MRW	mepc	异常 PC 寄存器（Machine Exception Program Counter）
	0x342	MRW	mcause	异常原因寄存器（Machine Cause Register）
	0x343	MRW	mtval	异常值寄存器（Machine Trap Value Register）

	0x344	MRW	mip	中断等待寄存器（Machine Interrupt Pending Register）
	0x345	MRW	mnxti	标准寄存器用于使能中断，处理下一个中断并返回下一个中断的 handler 入口地址
	0x346	MRO	mintstatus	标准寄存器用于保存当前中断 Level
	0x348	MRW	mscratchcsw	标准寄存器用于在特权模式变化时交换 mscratch 与目的寄存器的值
	0x349	MRW	mscratchcswl	标准寄存器用于在中断 Level 变化时交换 mscratch 与目的寄存器的值
	0xB00	MRW	mcycle	周期计数器的低 32 位（Lower 32 bits of Cycle counter）
	0xB80	MRW	mcycleh	周期计数器的高 32 位（Upper 32 bits of Cycle counter）
	0xB02	MRW	minstret	完成指令计数器的低 32 位（Lower 32 bits of Instructions-retired counter）
	0xB82	MRW	minstreth	完成指令计数器的高 32 位（Upper 32 bits of Instructions-retired counter）
RISC-V 标准 CSR（User Mode）	0xC00	URO	cycle	mcycle 寄存器的只读副本 注意：该寄存器在 User Mode 下是否可读由 CSR 寄存器 mcounteren 的 CY 比特域来控制，请参见第 7.4.29 节了解其详情。
	0xC01	URO	time	mtime 寄存器的只读副本 注意：该寄存器在 User Mode 下是否可读由 CSR 寄存器 mcounteren 的 TM 比特域来控制，请参见第 7.4.29 节了解其详情。
	0xC02	URO	instret	minstret 寄存器的只读副本 注意：该寄存器在 User Mode 下是否可读由 CSR 寄存器 mcounteren 的 IR 比特域来控制，请参见第 7.4.29 节了解其详情。
	0xC80	URO	cycleh	mcycleh 寄存器的只读副本 注意：该寄存器在 User Mode 下是否可读由 CSR 寄存器 mcounteren 的 CY 比特域来控制，请参见第 7.4.29 节了解其详情。
	0xC81	URO	timeh	mtimeh 寄存器的只读副本 注意：该寄存器在 User Mode 下是否可读由 CSR 寄存器 mcounteren 的 TM 比特域来控制，请参见第 7.4.29 节了解其详情。
	0xC82	URO	instreth	minstreth 寄存器的只读副本 注意：该寄存器在 User Mode 下是否可读由 CSR 寄存器 mcounteren 的 IR 比特域来控制，请参见第 7.4.29 节了解其详情。
Bumblebee 内核自定义 CSR	0x320	MRW	mcountinhibit	自定义寄存器用于控制计数器的开启和关闭
	0x7c3	MRO	mnvec	NMI 处理入口基地址寄存器
	0x7c4	MRW	msubm	自定义寄存器用于保存 Core 当前的 Trap 类型，以及进入 Trap 前的 Trap 类型。

	0x7d0	MRW	mmisc_ctl	自定义寄存器用于控制 NMI 的处理程序入口地址
	0x7d6	MRW	msavestatus	自定义寄存器用于保存 mstatus 值
	0x7d7	MRW	msaveepc1	自定义寄存器用于保存第一级嵌套 NMI 或异常的 mepc
	0x7d8	MRW	msavecause1	自定义寄存器用于保存第一级嵌套 NMI 或异常的 mcause
	0x7d9	MRW	msaveepc2	自定义寄存器用于保存第二级嵌套 NMI 或异常的 mepc
	0x7da	MRW	msavecause2	自定义寄存器用于保存第二级嵌套 NMI 或异常的 mcause
	0x7eb	MRW	pushmsubm	自定义寄存器用于将 msubm 的值存入堆栈地址空间
	0x7ec	MRW	mtvt2	自定义寄存器用于设定非向量中断处理模式的中断入口地址
	0x7ed	MRW	jalmnxti	自定义寄存器用于使能 ECLIC 中断，该寄存器的读操作能处理下一个中断同时返回下一个中断 handler 的入口地址，并跳转至此地址。
	0x7ee	MRW	pushmcause	自定义寄存器用于将 mcause 的值存入堆栈地址空间
	0x7ef	MRW	pushmepc	自定义寄存器用于将 mepc 的值存入堆栈地址空间
	0x811	MRW	sleepvalue	WFI 的休眠模式寄存器
	0x812	MRW	txevt	发送 Event 寄存器
	0x810	MRW	wfe	Wait for Event 控制寄存器
注意： ■ MRW 表示 Machine Mode Readable/Writeable ■ MRO 表示 Machine Mode Read-Only ■ URW 表示 User Mode Readable/Writeable ■ URO 表示 User Mode Read-Only				

7.3. Bumblebee 内核的 CSR 寄存器的访问权限

Bumblebee 内核对于 CSR 寄存器的访问权限规定如下：

- 无论是在 Machine Mode 还是 User Mode 下：
  - 如果向不存在的 CSR 寄存器地址区间进行读写操作，则会产生 Illegal Instruction Exception。
- 在 Machine Mode 下：
  - 对 MRW 或者 URW 属性的 CSR 寄存器进行读写操作则一切正常。

- 对 MRO 或者 URO 属性的 CSR 寄存器进行读操作则一切正常。
- 如果向 MRO 或者 URO 属性的 CSR 寄存器进行写操作，则会产生 Illegal Instruction Exception。

- 在 User Mode 下：
  - 对 URW 属性的 CSR 寄存器进行读写操作均一切正常。
  - 对 URO 属性的 CSR 寄存器进行读操作则一切正常。
    - 注意：对于 URO 属性的 cycle、cycleh、time、timeh、instret、instreth 寄存器，其能否可读还受 mcounteren 的相关比特域来控制，请参见 7.4.29 了解其详情。
  - 如果向 URO 属性的 CSR 寄存器进行写操作，则会产生 Illegal Instruction Exception。
  - 如果向 MRW 或者 MRO 属性的 CSR 寄存器进行读写操作，则会产生 Illegal Instruction Exception。

7.4. Bumblebee 内核支持的 RISC-V 标准 CSR

本节介绍 Bumblebee 内核自定义的 CSR 寄存器（RV32IMAC 架构支持 Machine Mode 和 User Mode 相关）。

7.4.1. misa

misa 寄存器用于指示当前处理器所支持的架构特性。

misa 寄存器的最高两位用于指示当前处理器所支持的架构位数：

- 如果最高两位值为 1，则表示当前为 32 位架构（RV32）。
- 如果最高两位值为 2，则表示当前为 64 位架构（RV64）。
- 如果最高两位值为 3，则表示当前为 128 位架构（RV128）。

misa 寄存器的低 26 位用于指示当前处理器所支持的 RISC-V ISA 中不同模块化指令子集，每

一位表示的模块化指令子集如图 7-1 中所示。该寄存器其他未使用到的比特域为常数 0。

Bit	Character	Description
0	A	Atomic extension
1	B	*Tentatively reserved for Bit operations extension*
2	C	Compressed extension
3	D	Double-precision floating-point extension
4	E	RV32E base ISA
5	F	Single-precision floating-point extension
6	G	Additional standard extensions present
7	H	*Reserved*
8	I	RV32I/64I/128I base ISA
9	J	*Tentatively reserved for Dynamically Translated Languages extension*
10	K	*Reserved*
11	L	*Tentatively reserved for Decimal Floating-Point extension*
12	M	Integer Multiply/Divide extension
13	N	User-level interrupts supported
14	O	*Reserved*
15	P	*Tentatively reserved for Packed-SIMD extension*
16	Q	Quad-precision floating-point extension
17	R	*Reserved*
18	S	Supervisor mode implemented
19	T	*Tentatively reserved for Transactional Memory extension*
20	U	User mode implemented
21	V	*Tentatively reserved for Vector extension*
22	W	*Reserved*
23	X	Non-standard extensions present
24	Y	*Reserved*
25	Z	*Reserved*

图 7-1 misa 寄存器低 26 位各域表示的模块化指令子集

注意：misa 寄存器在 RISC-V 架构文档中被定义为可读可写的寄存器，从而允许某些处理器的设计能够动态地配置某些特性。但是在 Bumblebee 内核的实现中，misa 寄存器为只读寄存器，恒定地反映不同型号处理器核所支持的 ISA 模块化子集。

7.4.2. mie

ECLIC 中断模式下 mie 寄存器的控制位不起作用，读 mie 返回全 0。

7.4.3. mvendorid

此寄存器是只读寄存器，用于反映该处理器核的商业供应商编号（Vendor ID）。

如果此寄存器的值为 0，则表示此寄存器未实现。

7.4.4. marchid

此寄存器是只读寄存器，用于反映该处理器核的硬件实现微架构编号（Microarchitecture ID）。

如果此寄存器的值为 0，则表示此寄存器未实现。

7.4.5. mimpid

此寄存器是只读寄存器，用于反映该处理器核的硬件实现编号（Implementation ID）。

如果此寄存器的值为 0，则表示此寄存器未实现。

7.4.6. mhartid

此寄存器是只读寄存器，用于反映当前 Hart 的编号（Hart ID）。

Hart（取“Hardware Thread”之意）表示一个硬件线程，单个处理器核中可能实现多份硬件线程，譬如硬件超线程（Hyper-threading）技术，每套线程有自己独立的寄存器组等上下文资源，但大多数的运算资源均被所有硬件线程复用，因此面积效率很高。在这样的硬件超线程处理器中，一个核内便存在着多个硬件线程（Hart）。

Bumblebee 内核中 Hart 编号值受输入信号 core_mhartid 控制。注意：RISC-V 架构规定，如果在单 Hart 或者多 Hart 的系统中，起码要有一个 Hart 的编号必须是 0。

7.4.7. mstatus

mstatus 寄存器是机器模式（Machine Mode）下的状态寄存器。mstatus 寄存器中各控制位域如表 7-2 所示。

表 7-2 mstatus 寄存器各控制位

域	位	复位值	描述
Reserved	2:0	N/A	未使用的域为常数 0
MIE	3	0	参见第 7.4.8 节了解其详情
Reserved	6:4	N/A	未使用的域为常数 0
MPIE	7	0	参见第 7.4.9 节了解其详情
Reserved	10:8	N/A	未使用的域为常数 0
MPP	12:11	0	参见第 7.4.9 节了解其详情
FS	14:13	0	参见第 7.4.10 节了解其详情
XS	16:15	0	参见第 7.4.11 节了解其详情
Reserved	17	N/A	未使用的域，值是无关的
Reserved	30:18	N/A	未使用的域为常数 0
SD	31	0	参见第 7.4.12 节了解其详情

7.4.8. mstatus 的 MIE 域

mstatus 寄存器中的 MIE 域表示全局中断使能：

当 MIE 域的值为 1 时，表示中断的全局开关打开，中断能够被正常响应；

当 MIE 域的值为 0 时，表示全局关闭中断，中断被屏蔽，无法被响应。

注意：Bumblebee 内核在进入异常、中断或者 NMI 处理模式时，MIE 的值会被更新成为 0（意味着进入异常、中断或者 NMI 处理模式后中断被屏蔽）。

7.4.9. mstatus 的 MPIE 和 MPP 域

mstatus 寄存器中的 MPIE 和 MPP 域分别用于自动保存进入异常，NMI 和中断之前 mstatus.MIE、特权模式（Privilege Mode）时进行自动恢复。

Bumblebee 内核进入异常时更新 mstatus 寄存器 MPIE 和 MPP 域的硬件行为，请参见 3.4.5 节了解其详情。

Bumblebee 内核退出异常时（在异常处理模式下执行 mret 指令）更新 mstatus 寄存器 MPIE

和 MPP 域的硬件行为，请参见 3.5.2 节了解其详情。

Bumblebee 内核进入 NMI 时更新 mstatus 寄存器 MPIE 和 MPP 域的硬件行为，请参见 4.3.4 节了解其详情。

Bumblebee 内核退出 NMI 时（在异常处理模式下执行 mret 指令）更新 mstatus 寄存器 MPIE 和 MPP 域的硬件行为，请参见 4.4.2 节了解其详情。

Bumblebee 内核进入中断时更新 mstatus 寄存器 MPIE 和 MPP 域的硬件行为，请参见 5.6.5 节了解其详情。

Bumblebee 内核退出中断时（在异常处理模式下执行 mret 指令）更新 mstatus 寄存器 MPIE 和 MPP 域的硬件行为，请参见 5.7.2 节了解其详情。

注意： mstatus.MPIE 域和 mstatus.MPP 域的值与 mcause.MPIE 域和 mcause.MPP 域的值是镜像关系，即，在正常情况下，mstatus.MPIE 域的值与 mcause.MPIE 域的值总是完全一样，mstatus.MPP 域的值与 mcause.MPP 域的值总是完全一样。

7.4.10. mstatus 的 FS 域

mstatus 寄存器中的 FS 域用于维护或反映浮点单元的状态。

注意：此域只有在配置了浮点指令（“F”或者“D”指令子集）时才会存在。

FS 域由两位组成，其编码如下图所示。

Status	FS Meaning	XS Meaning
0	Off	All off
1	Initial	None dirty or clean, some on
2	Clean	None dirty, some clean
3	Dirty	Some dirty

图 7-2 FS 域表示的状态编码

FS 域的更新准则如下：

- FS 上电后的默认值为 0，意味着浮点单元的状态为 Off。因此为了能够正常使用浮点单元，软件需要使用 CSR 写指令将 FS 的值改写为非 0 值以打开浮点单元（FPU）的功能。
- 如果 FS 的值为 1 或者 2，当执行了任何的浮点指令之后，FS 的值会自动切换为 3，表示浮点单元的状态为 Dirty（状态发生了改变）。
- 如果处理器不想使用浮点运算单元（譬如将浮点单元关电以节省功耗），可以使用 CSR 写指令将 mstatus 寄存器的 FS 域设置成 0，将浮点单元的功能予以关闭。当浮点单元的功能关闭之后，任何访问浮点 CSR 寄存器的操作或者任何执行浮点指令的行为都将会产生非法指令（Illegal Instruction）异常。

除了用于上述功能，FS 域的值还用于操作系统在进行上下文切换时的指引信息，感兴趣的用户请参见 RISC-V“特权架构文档版本 1.10”原文。

7.4.11. mstatus 的 XS 域

mstatus 寄存器中的 XS 域与 FS 域的作用类似，但是其用于维护或反映用户自定义的扩展指令单元状态。

在标准的 RISC-V“特权架构文档版本 1.10”中定义 XS 域为只读域，其用于反映所有自定义扩展指令单元的状态总和。但请注意：在 Bumblebee 内核的硬件实现中，将 XS 域设计成可写可读域，其作用完全与 FS 域类似，软件可以通过改写 XS 域的值达到打开或者关闭协处理器扩展指令单元的目的。

与 FS 域类似，XS 除了用于上述功能之外还用于操作系统在进行上下文切换时的指引信息，感兴趣的用户请参见 RISC-V“特权架构文档版本 1.10”原文。

7.4.12. mstatus 的 SD 域

mstatus 寄存器中的 SD 域是一个只读域，其反映了 XS 域或者 FS 域处于脏（Dirty）状态。其逻辑关系表达式为：SD = （（FS==11） OR （XS==11））。

之所以设置此只读的 SD 域是为了方便软件快速的查询 XS 域或者 FS 域是否处于脏（Dirty）状态，从而在上下文切换时可以快速判断是否需要对浮点单元或者扩展指令单元进行上下文的保存。

感兴趣的用户请参见 RISC-V“特权架构文档版本 1.10”原文。

7.4.13. mtvec

mtvec 寄存器用于配置中断和异常处理程序的入口地址。

- 当 mtvec 配置中断的异常处理程序入口地址时要点如下：
  - 异常处理程序采用 4byte 对齐的 mtvec 地址（将 mtvec 的低 2bit 用 0 代替）作为入口地址。
- 当 mtvec 配置中断程序的入口地址时要点如下：
  - 当 mtvec.MODE != 6'b000011 时，处理器使用“默认中断模式”。
  - 当 mtvec.MODE = 6'b000011 时，处理器使用“ECLIC 中断模式”，推荐使用此模式。
    - 中断为非向量处理模式时的入口地址和要点如第 5.13.2 节中所述。
    - 中断为向量处理模式时的入口地址和要点如第 5.13.1 节中所述。

mtvec 寄存器各地址位域如表 7-3 所示。

表 7-3 mtvec 寄存器各控制位

域	位	描述
ADDR	31:6	mtvec 地址
MODE	5：0	■ MODE 域为中断处理模式控制域： ● 000011: ECLIC 中断模式（推荐模式） ● Others: 默认中断模式

7.4.14. mtvt

mtvt 寄存器用于保存 ECLIC 中断向量表的基地址，此基地址至少为 64byte 对齐。

为了提升性能减少硬件门数，硬件根据实际实现的中断的个数来决定 mtvt 的对齐方式，具体如表 7-4 所示。

表 7-4 mtvt 对齐方式

最大中断个数	mtvt 对齐方式
0 to 16	64-byte
17 to 32	128-byte
33 to 64	256-byte
65 to 128	512-byte
129 to 256	1KB
257 to 512	2KB
513 to 1024	4KB
1025 to 2048	8KB
2045 to 4096	16KB

7.4.15. mscratch

mscratch 寄存器用于 Machine Mode 下的程序临时保存某些数据。mscratch 寄存器可以提供一种保存和恢复机制，譬如，在进入中断或者异常处理模式后，将应用程序的用户堆栈指针（SP）寄存器临时存入 mscratch 寄存器中，然后在退出异常处理程序之前，将 mscratch 寄存器中的值读出恢复至用户堆栈指针（SP）寄存器。

7.4.16. mepc

mepc 寄存器用于保存进入异常之前处理器正在执行指令的 PC 值，作为异常的返回地址。

为了理解此寄存器，请先参见第 0 章系统地了解异常的相关信息。

注意：

- 处理器进入异常时，mepc 寄存器被同时更新以反映当前遇到异常的指令的 PC 值。
- 值得注意的是，虽然 mepc 寄存器会在异常发生时自动被硬件更新，但是 mepc 寄存器本身也是一个（在 Machine Mode 下）可读可写的寄存器，因此软件也可以直接写该寄存器以修改它的值。

mepc 寄存器各地址位域如表 7-5 所示。

表 7-5 mepc 寄存器各控制位

域	位	描述
EPC	31：1	保存异常发生前处理器正在执行的指令的 PC 值
Reserved	0	未使用的域为常数 0

7.4.17. mcause

mcause 寄存器，用于保存进入 NMI、异常和中断之前的出错原因，以便于对 Trap 原因进行诊断和调试。

mcause 寄存器各地址位域如表 7-6 所示。

表 7-6 mcause 寄存器各控制位

域	位	描述
INTERRUPT	31	表示当前 Trap 种类： ■ 0：异常或者 NMI ■ 1：中断
MINHV	30	表示处理器正在读取中断向量表
MPP	29:28	进入中断之前的特权模式，与 mstatus.mpp 相同
MPIE	27	进入中断之前的中断使能，与 mstatus.mpie 相同
Reserved	26:24	未使用的域为常数 0
MPIL	23:16	前一个中断级别
Reserved	15:12	未使用的域为常数 0
EXCCODE	11:0	异常/中断编码

注意：

- mstatus 寄存器的 MPIE 和 MPP 域与 mcause 的 MPIE 和 MPP 域为镜像关系。
- NMI 的 mcause.EXCCODE 可能为 0x1 或者 0xfff，实际值由 mmisc_ctl 控制，详情请参考 7.5.4 节。

7.4.18. mtval（mbadaddr）

mtval 寄存器（又名 mbadaddr，有些版本的工具链只识别此名称），用于保存进入异常之前的出错指令的编码值或者存储器访问的地址值，以便于对异常原因进行诊断和调试。

为了理解此寄存器，请先参见第 0 章系统地了解异常的相关信息。

Bumblebee 内核进入异常时，mtval 寄存器被同时更新以反映当前遇到异常的信息。

7.4.19. mip

ECLIC 中断模式下 mip 寄存器的控制位不起作用，读 mip 返回全 0。

7.4.20. mnxti

mnxti （Next Interrupt Handler Address and Interrupt-Enable CSR）可以被软件访问用来处理处于相同 Privilege Mode 下的下一个中断，同时不会造成冲刷流水线以及上下文保存恢复。

mnxti 寄存器可通过 CSRRSI/CSRRCI 指令来访问，读操作返回值是下一个中断的 handler 地址，而 mnxti 的写回操作会更新中断使能的状态。

注意：

1. 对于不同 Privilege Mode 的中断，硬件会以中断嵌套的方式处理，因此 mnxti 只会处理相同 Privilege Mode 下的下一个中断。
2. mnxti 寄存器与常规的 CSR 指令不一样，其返回值与常规寄存器的 RMW（read-modify-write）操作的值不同：
   - mnxti 的 CSR 读操作的返回值有以下两种情况：
     - 当出现以下情况时，返回值为 0。
       - 没有可以响应的中断
       - 当下最高优先级的中断是向量中断

- 当中断为非向量中断时，返回此中断的中断处理程序入口地址。

- mnxti 的 CSR 写操作会更新以下寄存器及寄存器域：
  - mstatus 是当前 RMW（read-modify-write）操作的目的寄存器
  - mcause.EXCCODE 域和会被分别更新为当前响应中断的中断 id
  - mintstatus.MIL 域被更新为当前响应中断的中断级别（Level）。

7.4.21. mintstatus

mintstatus 寄存器保存每个 Privilege Mode 下的有效中断的中断 level。

表 7-7 minstatus 寄存器的控制位

域	位	描述
MIL	31:24	Machine Mode 的有效中断 level
Reserved	23：8	未使用的域为常数 0
UIL	7:0	User Mode 的有效中断 level

7.4.22. mscratchcsw

mscratchcsw 寄存器用于在多个特权模式间切换时，交换目的寄存器与 mscratch 的值来加速中断处理。

使用带读操作的 CSR 指令访问 mscratchcsw，在出现中断前后特权模式不一致时，有以下伪指令所示的寄存器操作：

```
csrrw rd, mscratchcsw, rs1

// Pseudocode operation.
if (mcause.mpp!=M-mode) then {
    t = rs1; rd = mscratch; mscratch = t;
} else {
    rd = rs1; // mscratch unchanged.
}
```

```
    // Usual use: csrrw sp, mscratchcsw, sp
```

处理器在低特权模式（Privilege Mode）时发生中断，处理器进入高特权模式处理中断，在处理中断时，需要使用堆栈来保存进入中断前的处理器状态。此时如果继续使用低特权模式下的堆栈指针（SP）， 则高特权模式下堆栈的数据会存储在低特权模式可以访问的区间，导致出现高特权模式的数据泄露给低特权模式这一安全漏洞。为避免此安全漏洞，RISC-V 架构规定当处理器处于低特权模式时，需要将高特权模式的堆栈指针（SP）保存至 mscratch 寄存器，这样在进入高特权模式后，处理器可以用 mscratch 寄存器的值来恢复高特权模式的堆栈指针（SP）。

使用常规指令来执行以上的程序需要耗费较多的 cycle， 为此 RISC-V 架构定义 mscratchcsw 寄存器，在进入中断后立刻执行 mscratchcsw 寄存器指令，交换 mscratch 与 SP 的值，用来恢复高特权模式的堆栈指针（SP），同时备份低特权模式的堆栈指针（SP）至 mscratch 寄存器。在执行 mret 指令退出中断前，也加上一条 mscratchcsw 指令，交换 mscratch 寄存器和堆栈指针（SP）的值，将高特权模式的堆栈指针（SP）再次备份到 mscratch，同时恢复低特权模式的堆栈指针（SP）。这样，只需要两条指令便可以解决不同特权模式的堆栈指针（SP）切换问题，加速了中断处理。

注意：为了避免虚拟化的漏洞，软件不能直接读取处理器当前的特权模式（Privilege Mode）。如果软件试图在更低的特权模式下访问给定特权模式下的 mscratchcsw 做寄存器 swap 操作会导致处理器进入 Trap，因此 mscratchcsw 不会导致虚拟化漏洞。

7.4.23. mscratchcswl

mscratchcswl 寄存器用于在多个中断 level 间切换时，交换目的寄存器与 mscratch 的值来加速中断处理。

使用带读操作的 CSR 指令访问 mscratchcsw，当特权模式不变，在出现中断程序和应用程序的切换时，有以下伪指令所示的寄存器操作：

```
csrrw rd, mscratchcswl, rs1

  // Pseudocode operation.
  if ( (mcause.mpil==0) != (mintstatus.mil == 0) ) then {
     t = rs1; rd = mscratch; mscratch = t;
  } else {
     rd = rs1; // mscratch unchanged.
  }
```

```
    // Usual use: csrrw sp, mscratchcswl, sp
```

在单一特权模式下，将中断处理程序任务与应用程序任务的堆栈空间分离可以增强健壮性、减少空间使用并有助于系统调试。中断处理程序任务具有非零中断级别，而应用程序任务具有零中断级别，根据这一特性 RISC-V 架构定义了 mscratchcswl 寄存器。与 mscratchcsw 类似，在中断程序入口和出口分别添加一条 mscratchcswl 可以实现中断处理程序与应用程序之间的快速的堆栈指针切换，保证中断处理程序和应用程序的堆栈空间分离。

7.4.24. mcycle 和 mcycleh

RISC-V 架构定义了一个 64 位宽的时钟周期计数器，用于反映处理器执行了多少个时钟周期。只要处理器处于执行状态时，此计数器便会不断自增计数。

mcycle 寄存器反映了该计数器低 32 位的值，mcycleh 寄存器反映了该计数器高 32 位的值。

mcycle 和 mcycleh 寄存器可以用于衡量处理器的性能，且具备可读可写属性，因此软件可以通过 CSR 指令改写 mcycle 和 mcycleh 寄存器中的值。

由于考虑到此计数器计数会消耗某些动态功耗，因此在 Bumblebee 内核的实现中，在自定义 CSR 寄存器 mcountinhibit 中额外增加了一位控制域，软件可以配置此控制域将 mcycle 和 mcycleh 对应的计数器停止计数，从而在不需要衡量性能之时停止计数器以达到省电的作用。请参见 7.5.1 节了解更多 mcountinhibit 寄存器信息。

注意：如果在调试模式下时，此计数器并不会计数，只有在正常功能模式下，计数器才会进行计数。

7.4.25. minstret 和 minstreth

RISC-V 架构定义了一个 64 位宽的指令完成计数器，用于反映处理器成功执行了多少条指令。只要处理器每成功执行完成一条指令，此计数器便会自增计数。

minstret 寄存器反映了该计数器低 32 位的值，minstreth 寄存器反映了该计数器高 32 位的值。

minstret 和 minstreth 寄存器可以用于衡量处理器的性能，且具备可读可写属性，因此软件可以通过 CSR 指令改写 minstret 和 minstreth 寄存器中的值。

由于考虑到此计数器计数会消耗某些动态功耗，因此在 Bumblebee 内核的实现中，在自定义的 CSR 寄存器 mcountinhibit 中额外增加了一位控制域，软件可以配置此控制域将 minstret 和 minstreth 对应的计数器停止计数，从而在不需要衡量性能之时停止计数器以达到省电的作用。请参见 7.5.1 节了解更多 mcountinhibit 寄存器信息。

注意：如果在调试模式下时，此计数器并不会计数，只有在正常功能模式下，计数器才会进行计数。

7.4.26. cycle 和 cycleh

cycle 和 cycleh 分别是 mcycle 和 mcycleh 的只读副本。该寄存器在 User Mode 下是否可读由 CSR 寄存器 mcounteren 的 CY 比特域来控制，请参见第 7.4.29 节了解其详情。

7.4.27. instret 和 instreth

instret 和 instreth 分别是 minstret 和 minstreth 的只读副本。该寄存器在 User Mode 下是否可读由 CSR 寄存器 mcounteren 的 IR 比特域来控制，请参见第 7.4.29 节了解其详情。

7.4.28. time 和 timeh

time 和 timeh 分别是 mtime 和 mtimeh 的只读副本。该寄存器在 User Mode 下是否可读由 CSR 寄存器 mcounteren 的 TM 比特域来控制，请参见第 7.4.29 节了解其详情。

7.4.29. mcounteren

该寄存器只有在支持 User Mode 的配置下才会存在。mcounteren 寄存器中各控制位域如表 7-8 所示。

表 7-8 mcounteren 寄存器各控制位

域	位	描述
CY	0	此位控制在 User Mode 下是否能够访问 cycle 和 cycleh 寄存器： ■ 如果此位为 1，则在 User Mode 下能够正常访问 cycle 和 cycleh。 ■ 如果此为为 0，则在 User Mode 下访问 cycle 和 cycleh 会触发 illegal instruction exception。 此位复位默认值为 0
TM	1	此位控制在 User Mode 下是否能够访问 time 和 timeh 寄存器： ■ 如果此位为 1，则在 User Mode 下能够正常访问 time 和 timeh。 ■ 如果此为为 0，则在 User Mode 下访问 time 和 timeh 会触发 illegal instruction exception。 此位复位默认值为 0
IR	2	此位控制在 User Mode 下是否能够访问 instret 和 instreth 寄存器： ■ 如果此位为 1，则在 User Mode 下能够正常访问 instret 和 instreth。 ■ 如果此为为 0，则在 User Mode 下访问 instret 和 instreth 会触发 illegal instruction exception。 此位复位默认值为 0
Reserved	3~31	其他未使用的域为常数 0

7.5. Bumblebee 内核自定义的 CSR

本节介绍 Bumblebee 内核自定义的 CSR 寄存器。

7.5.1. mcountinhibit

mcountinhibit 寄存器用于控制 mcycle 和 minstret 的计数，各控制位域如表 7-9 所示。

表 7-9 mcountinhibit 寄存器各控制位

域	位	描述
Reserved	31:3	未使用的域为常数 0
IR	2	IR 为 1 时 minstret 的计数被关闭
Reserved	1	未使用的域为常数 0
CY	0	CY 为 1 时 mcycle 的计数被关闭

7.5.2. mnvec

mnvec 寄存器用于配置 NMI 的入口地址。

为了理解此寄存器，请先参见第 4 章系统地了解 NMI 的相关信息。

在处理器的程序执行过程中，一旦遇到 NMI 发生，则终止当前的程序流，处理器被强行跳转到一个新的 PC 地址，Bumblebee 内核进入 NMI 后跳入的 PC 地址即由 mnvec 寄存器指定。

注意：mnvec 的值由 mmisc_ctl 控制，更多细节请参考 7.5.4 节。

7.5.3. msubm

Bumblebee 内核自定义 msubm 寄存器用于保存进入 Trap 前后的 Trap 类型。

msubm 寄存器中各控制位域如表 7-10 所示。

表 7-10 msubm 寄存器各控制位

域	位	描述
Reserved	31:10	未使用的域为常数 0
PTYP	9:8	保存进入 Trap 之前的 Trap 类型： ■ 0：非 Trap 状态 ■ 1：中断 ■ 2：异常 ■ 3：NMI
TYP	7:6	指示 Core 当前的 Trap 类型： ■ 0：非 Trap 状态 ■ 1：中断 ■ 2：异常 ■ 3：NMI
Reserved	5:0	未使用的域为常数 0

7.5.4. mmisc_ctl

Bumblebee 内核自定义 mmisc_ctl 寄存器用于控制 mnvec 和 NMI 的 mcause 值。

mmisc_ctl 寄存器中各控制位域如表 7-11 所示。

表 7-11 mmisc_ctl 寄存器各控制位

域	位	描述
Reserved	31:10	未使用的域为常数 0
NMI_CAUSE_FFF	9	控制 mnvec 及 NMI 的 mcause.EXCCODE： ■ 0：mnvec 的值等于处理器 reset 后的 PC， NMI 的 mcause.EXCCODE 为 0x1 ■ 1：mnvec 的值与 mtvec 一致，NMI 的 mcause.EXCCODE 为 0xfff
Reserved	8:0	未使用的域为常数 0

7.5.5. msavestatus

msavestatus 用于存储 mstatus 和 msubm 的值，以保证 mstatus 和 msubm 的各个域的状态不会被 NMI 或者异常冲刷掉。msavestatus 有两级堆栈，最多可支持 3 级异常/NMI 状态保存。更多两级 NMI/异常状态堆栈，请参见 4.6 节。

msavestatus 寄存器各控制位如表 7-12 所示。

表 7-12 msavestatus 寄存器各控制位

域	位	描述
Reserved	31:16	未使用的域为常数 0
PTYP2	15:14	第二级嵌套 NMI/异常发生前的 Trap 类型
Reserved	13:11	未使用的域为常数 0
MPP2	10:9	第二级嵌套 NMI/异常发生前的 Privilege mode
MPIE2	8	第二级嵌套 NMI/异常发生前的中断使能状态
PTYP1	7:6	第一级嵌套 NMI/异常发生前的 Trap 类型
Reserved	5:3	未使用的域为常数 0
MPP1	2:1	第一级嵌套 NMI/异常发生前的 Privilege mode
MPIE1	0	第一级嵌套 NMI/异常发生前的中断使能状态

7.5.6. msaveepc1 和 msaveepc2

msaveepc1 和 msaveepc2 分别作为一级 NMI/异常状态堆栈和二级 NMI/异常状态堆栈，用来存储第一级嵌套 NMI/异常发生前的 PC，以及第二级嵌套 NMI/异常发生前的 PC。

- msaveepc2 <= msaveepc1 <= mepc <= interrupted PC <= NMI/exception PC

当执行 mret 指令，同时 mcause.INTERRUPT 为 0（例如 NMI，或者异常），msaveepc1 和 msaveepc2 分别通过一级和两级 NMI/异常状态堆栈来恢复处理器的 PC。

- msaveepc2 => msaveepc1 => mepc => PC

7.5.7. msavecause1 和 msavecause2

msavecause1 和 msavecause2 分别作为一级 NMI/异常状态堆栈和二级 NMI/异常状态堆栈，用来存储第一级嵌套 NMI/异常发生前的 mcause，以及第二级嵌套 NMI/异常发生前的 mcause。

- msavecause2 <= msavecause1 <= mcause <= NMI/exception cause

当执行 mret 指令，同时 mcause.INTERRUPT 为 0（例如 NMI，或者异常），msavecause1 和 msavecause2 分别通过一级和两级 NMI/异常状态堆栈来恢复 mcause 状态。

- msavecause2 => msavecause1 => mcause

7.5.8. pushmsubm

Bumblebee 内核定义了通过 pushmsubm 寄存器 csrrwi 操作实现的 CSR 指令，存储 msubm 的值到堆栈指针作为基地址的 memory 空间.

以如下指令为例介绍此 CSR 指令：

```
csrrwi x0, PUSHMSUBM, 1
```

该指令的操作是将 msubm 寄存器的值存到 SP（堆栈指针）+1*4 的地址。

7.5.9. mtvt2

mtvt2 用于指定 ECLIC 非向量模式的中断 common-code 入口地址。

mtvt2 寄存器中各控制位域如表 7-13 所示。

表 7-13 mtvt2 寄存器各控制位

域	位	描述
CMMON-CODE-ENTRY	31:2	在 mtvt2.MTVT2EN=1 时，此域决定 ECLIC 非向量模式中断 common-code 入口地址。
Reserved	1	未使用的域为常数 0
MTVT2EN	0	mtvt2 使能位： ■ 0：ECLIC 非向量模式中断 common-code 入口地址由 mtvec 决定 ■ 1: ECLIC 非向量模式中断 common-code 入口地址由 mtvt2.COMMON-CODE-ENTRY 决定

7.5.10. jalmnxti

Bumblebee 内核定义了 jalmnxti 寄存器用于减少中断延迟，加速中断咬尾。

jalmnxti 除了包含 mnxti 的开启中断使能，处理下一个中断，返回下一个中断的入口地址等功能之外，还有跳转至中断 handler 的功能，因此可以缩短中断处理的指令个数，达到减少中断延迟，加速中断咬尾的目的。有关 jalmnxti 的更多细节请参见 5.13.1.3 节。

7.5.11. pushmcause

Bumblebee 内核定义了通过 pushmcause 寄存器 csrrwi 操作实现的 CSR 指令，存储 mcause 的值到堆栈指针作为基地址的 memory 空间.

以如下指令为例介绍此 CSR 指令：

```
csrrwi x0, PUSHMCAUSE, 1
```

该指令的操作是将 mcause 寄存器的值存到 SP（堆栈指针）+1*4 的地址。

7.5.12. pushmepc

Bumblebee 内核定义了通过 pushmepc 寄存器 csrrwi 操作实现的 CSR 指令，存储 mepc 的值到堆栈指针作为基地址的 memory 空间.

以如下指令为例介绍此 CSR 指令：

```
csrrwi x0, PUSHMPEC, 1
```

该指令的操作是将 mepc 寄存器的值存到 SP（堆栈指针）+1*4 的地址。

7.5.13. sleepvalue

Bumblebee 内核自定义了一个 CSR 寄存器 sleepvalue 用于控制不同的休眠模式，请参见第 8.1 节了解更多详情。sleepvalue 寄存器中各控制位域如表 7-14 所示。

表 7-14 sleepvalue 寄存器各控制位

域	位	描述
SLEEPVALUE	0	控制 WFI 的休眠模式 ■ 0：浅度休眠模式（执行 WFI 后，处理器内核主工作时钟 core_clk 被关闭） ■ 1：深度休眠模式（执行 WFI 后，处理器内核主工作时钟 core_clk 和处理器内核的常开时钟 core_aon_clk 都被关闭） 此位复位默认值为 0
Reserved	31:1	未使用的域为常数 0

7.5.14. txevt

Bumblebee 内核自定义了一个 CSR 寄存器 txevt，用于对外发送 Event。

txevt 寄存器中各控制位域如表 7-15 所示。

表 7-15 txevt 寄存器各控制位

域	位	描述
TXEVT	0	控制发送 Event： ■ 如果向此位写 1，则会触发 Bumblebee 内核的输出信号 tx_evt 产生一个单周期脉冲信号，作为对外的 Event 信号。 ■ 该比特位为自清比特位，即，向此位写入 1 之后，下一个周期其被自清为 0。 ■ 向此位写入 0 则无任何反应和操作。 此位复位默认值为 0
Reserved	31:1	未使用的域为常数 0

7.5.15. wfe

Bumblebee 内核自定义了一个 CSR 寄存器 wfe，用于控制 WFI 指令的唤醒条件是使用中断还是使用 Event。请参见第 8.2.3 节了解更多详情。

wfe 寄存器中各控制位域如表 7-16 所示。

表 7-16 wfe 寄存器各控制位

域	位	描述
WFE	0	控制 WFI 指令的唤醒条件是使用中断还是使用 Event。 ■ 0：处理器内核进入休眠模式时，可以被中断和 NMI 唤醒。 ■ 1：处理器内核进入休眠模式时，可以被 Event 和 NMI 唤醒。 此位复位默认值为 0。
Reserved	31:1	未使用的域为常数 0

8. Bumblebee 内核低功耗机制介绍

Bumblebee 内核可以支持休眠模式实现较低的静态功耗。

8.1. 进入休眠状态

Bumblebee 内核可以通过 WFI 指令进入休眠状态。当处理器执行到 WFI 指令之后，将会：

- 立即停止执行当前的指令流；
- 等待处理器内核完成任何尚未完成的滞外操作（Outstanding Transactions），譬如取指令和数据读写操作，以保证发到总线上的操作都完成；
  - 注意：如果在等待总线上的操作完成的过程中发生了存储器访问错误异常，则会进入到异常处理模式，而不会休眠。
- 当所有的滞外操作（Outstanding Transactions）都完成后，处理器会安全地进入一种空闲状态，这种空闲状态可以被称之为“休眠”状态。
- 当进入休眠模式后：
  - Bumblebee 内核内部的各个主要单元的时钟将会被门控关闭以节省静态功耗；
  - Bumblebee 内核的输出信号 core_wfi_mode 会拉高，指示此处理器核处于执行 WFI 指令之后的休眠状态；
  - Bumblebee 内核的输出信号 core_sleep_value 会输出 CSR 寄存器 sleepvalue 的值（注意：该信号只有在 core_wfi_mode 信号为高电平时生效；core_wfi_mode 信号为低电平时 core_sleep_value 的值一定是 0）。软件可以通过事先配置 CSR 寄存器 sleepvalue 来指示不同的休眠模式（0 或者 1）。注意：
    - 对于不同的休眠模式而言，Bumblebee 内核的行为完全一样。此休眠模式只是仅供 SoC 系统层面的 PMU（Power Management Unit）进行相应不同的控制。

8.2. 退出休眠状态

Bumblebee 内核处理器退出休眠模式的要点如下：

- Bumblebee 内核的输出信号 core_wfi_mode 会相应拉低。
- Bumblebee 内核处理器可以通过以下四种方式被唤醒：
  - NMI
  - 中断
  - Event
  - Debug 请求

下文将予以详细介绍 。

8.2.1. NMI 唤醒

NMI 总能够唤醒处理器内核。当处理器内核检测到输入信号 nmi 的上升沿，处理器内核被唤醒，进入到 NMI 服务程序开始执行。

8.2.2. 中断唤醒

中断也可以唤醒处理器内核：

- 如果 CSR 寄存器 wfe.WFE 域被配置为 0，则：
  - 如果 mstatus.MIE 域被配置为 1（表示全局中断被打开），则：
    - 当 ECLIC（通过将外部请求的中断进行仲裁）向处理器内核发送了中断，处理器内核被唤醒，进入到中断服务程序开始执行。
  - 如果 mstatus.MIE 域被配置为 0（表示全局中断被关闭），则：
    - 当 ECLIC（通过将外部请求的中断进行仲裁）向处理器内核发送了中断，处理器内核被唤醒，继续顺序执行之前停止的指令流（而不是进入到中断服务程序）。

- 如果 CSR 寄存器 wfe.WFE 域被配置为 1，则等待 Event 唤醒，请参见下节描述。

8.2.3. Event 唤醒

当满足如下条件时，Event 可以唤醒处理器内核：

- 如果 CSR 寄存器 wfe.WFE 域被配置为 1，则：
  - 当处理器内核检测到输入信号 rx_evt（称之为 Event 信号）为高电平时，处理器内核被唤醒，继续执行之前停止的指令流（而不是进入到中断服务程序）。

8.2.4. Debug 唤醒

Debug 请求总能够唤醒处理器内核，如果调试器（Debugger）接入，也会将处理器内核唤醒而进入调试模式。

8.3. Wait for Interrupt 机制

Wait for Interrupt 机制，是指将处理器内核进入休眠模式，然后等待中断唤醒处理器内核，醒来后进入相应中断的处理函数中去。

如第 8.1 节和第 8.2 节所述，Wait for Interrupt 机制可以直接通过 WFI 指令（配合 mstatus.MIE 域被配置为 1）完成。

8.4. Wait for Event 机制

Wait for Event 机制，是指将处理器内核进入休眠模式，然后等待 Event 唤醒处理器内核，醒来后继续先前停止的程序（而不是进入中断的处理函数中去）。

如第 8.1 节和第 8.2 节所述，Wait for Event 机制可以直接通过 WFI 指令，配合如下指令序列完成：

第 1 步：配置 wfe.WFE 域为 1

第 2 步：调用 WFI 指令。调用此指令后处理器会进入休眠模式，当 Event 或者 NMI 将其唤醒后将会继续向下执行。

第 3 步：恢复 wfe.WFE 域为 0