芯动力——硬件加速设计方法

第七章 基于平头哥E902处理器的SoC设计

——(3)无剑100 SoC 异常与中断

邸志雄@西南交通大学

zxdi@home.swjtu.edu.cn

slides与源代码网址 http://www.dizhixiong.cn/class5/

异常与中断机制

异常（Exception）

中断（Interrupt）

顺序执行程序指令流

- 突然被中断请求
- 异常事件打断而中止当前的程序

区别

- 起因内外有别

中断 → 外围硬件设备

异常 → 内部事件：非法指令和非对齐访问错误、执行特殊的系统服务指令
如ecall、ebreak等

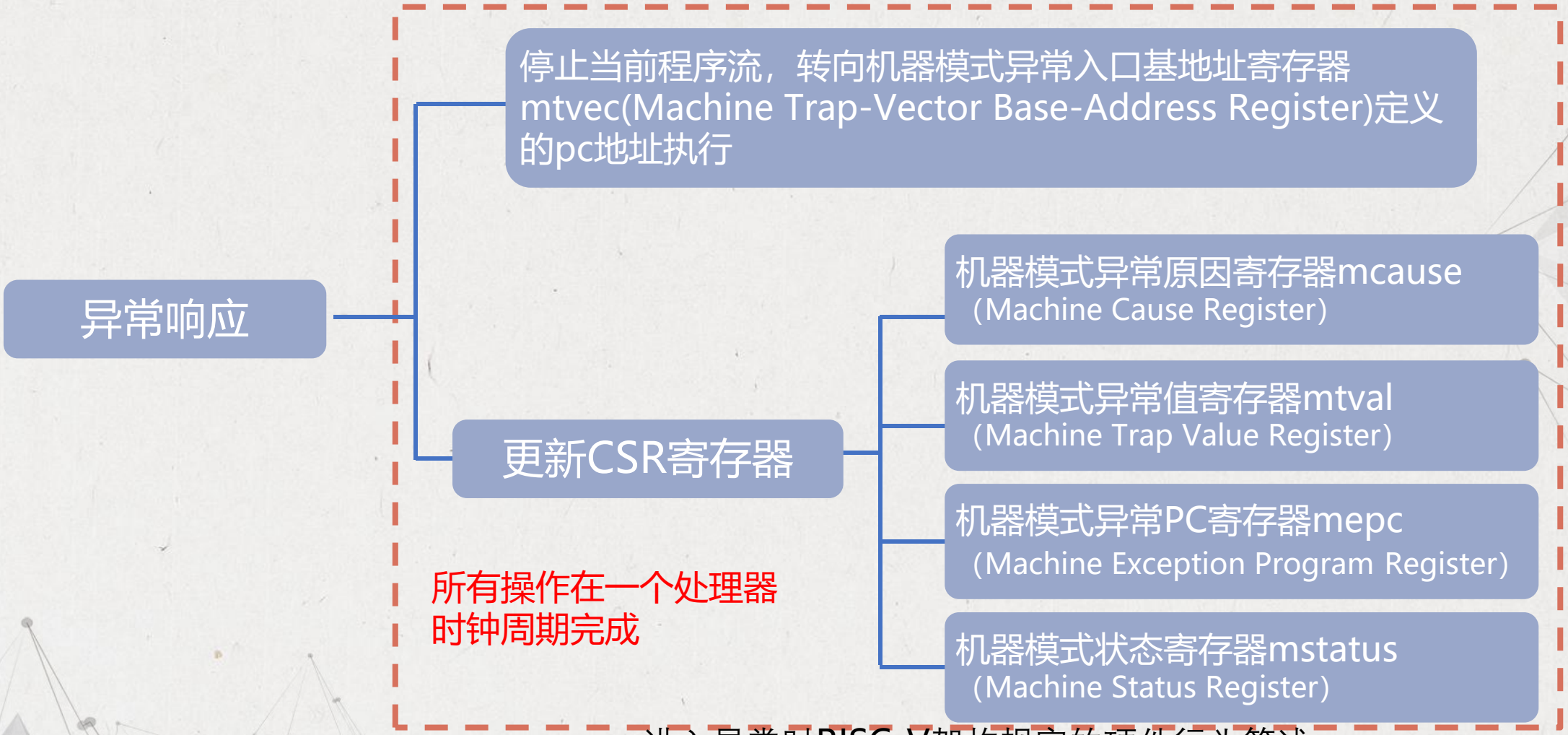
广义上的异常

异常机制包含

- 异常响应
- 异常处理
- 异常返回

异常响应

RISC-V 编程模型中没有异常使能寄存器，因此一旦触发异常立刻进行响应。



进入异常时RISC-V架构规定的硬件行为简述

异常响应

mepc　mcause　mtval　mstatus

mepc（机器模式可读写）

硬件自动将遇到异常时的PC值存入mepc，在异常结束后通过它保存的PC值返回之前的程序断点。
值得注意的是，如果异常因ecall/ebreak指令（主动触发异常）产生，则异常结束后仍会执行该指令造成死循环，可以在异常处理程序中修改mepc为下一条指令地址解决。

mcause

mcause记录了中断或异常的向量号

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

mtval

mtval记录了异常的详细信息，异常时硬件自动更新这两个寄存器。

异常响应

mepc　mcause　mtval　mstatus

mstatus（机器模式可读写）

MIE域为1，中断全局打开，为0则全局关闭，异常响应时MIE域被自动设置为0（所有中断被屏蔽）
异常响应时MPIE被更新为异常发生前MIE的值，用于异常结束后使MIE恢复异常前的值
MPP域用于记录异常发生前的工作模式，用于处理器在异常退出时恢复之前模式

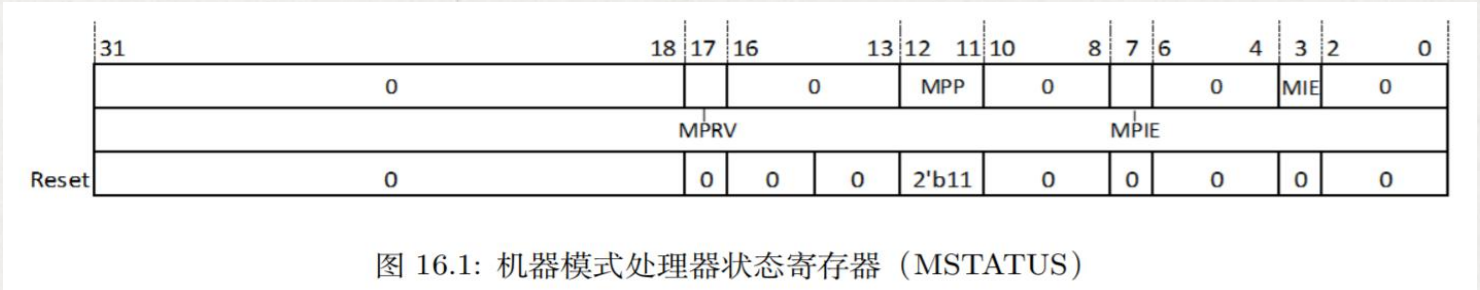


图 16.1: 机器模式处理器状态寄存器（MSTATUS）

异常处理

所有异常响应时的跳转执行入口均由 MTVEC 寄存器指定，因此在跳转到该入口之后，软件可以依据MCAUSE 寄存器中的异常向量号来决定是否实现跳转到各自对应的服务程序进行处理，即针对每个不同的异常的处理程序可以不同。

特别注意

与ARM架构不同，RISC-V规定异常和中断机制中没有硬件自动保存和恢复上下文的操作，软件需要明确地使用指令进行上下文的保存和恢复，即在异常和中断服务程序入口需要软件对 **GPR** （General Purpose Register） 等需要用到的寄存器进行压栈处理，结束时进行出栈恢复。

异常处理

- 首先判Wujian100的SDK中将所有异常按同一标准处理，即都执行异常服务函数Default_Handler，跳转执行trap程序并完成以下操作：

① 判断是否发生中断，若发生则先跳入中断服务程序执行（优先处理中断事件）

② 保护上下文，将GPR等寄存器压入堆栈中

③ 进入trap_c程序实现通过串口输出异常的原因和GPR通用寄存器的值，随后进入死循环。

```
la      a0, Default_Handler
ori     a0, a0, 3
csrw    mtvec, a0
```

在startup.s启动文件中
将Default_Handler函数地址
存入mtvec寄存器

```
    .align  6
    .weak   Default_Handler
    .global Default_Handler
    .type   Default_Handler, %function
Default_Handler:
    j       trap
    .size   Default_Handler, . - Default_Handler
```

跳入并执行trap函数

异常服务程序分析

```
trap:
  /* Check for interrupt */
  addi    sp, sp, -4
  sw      t0, 0x0(sp)
  csrr    t0, mcause

  blt     t0, x0, .Lirq

  addi    sp, sp, 4

  la      t0, g_trap_sp
  addi    t0, t0, -68
  sw      x1, 0(t0)
  sw      x2, 4(t0)
  sw      x3, 8(t0)
  sw      x4, 12(t0)
  sw      x6, 20(t0)
  sw      x7, 24(t0)
  sw      x8, 28(t0)
  sw      x9, 32(t0)
  sw      x10, 36(t0)
  sw      x11, 40(t0)
  sw      x12, 44(t0)
  sw      x13, 48(t0)
  sw      x14, 52(t0)
  sw      x15, 56(t0)
  csrr    a0, mepc
  sw      a0, 60(t0)
  csrr    a0, mstatus
  sw      a0, 64(t0)

  mv      a0, t0
  lw      t0, -4(sp)
  mv      sp, a0
  sw      t0, 16(sp)

  jal     trap_c
```

判断是否发生中断

GPR等寄存器压栈保存处理

执行trap_c函数并通过a0寄存器传递堆栈值给函数

- 在异常服务程序中会首先判断是否有中断发生，若有则优先进行中断响应。
- 随后，需要软件将CSR和GPR等寄存器压入堆栈进行保存
- 调用trap_c函数，并将存有寄存器值的堆栈空间首地址通过a0寄存器传递给函数的形参

异常服务程序分析

```c
void (*trap_c_callback)(void);

void trap_c(uint32_t *regs)
{
    int i;
    uint32_t vec = 0;

    vec = __get_MCAUSE() & 0x3FF;

    printf("CPU Exception: NO.%d", vec);
    printf("\n");

    for (i = 0; i < 15; i++) {
        printf("x%d: %08x\t", i + 1, regs[i]);

        if ((i % 4) == 3) {
            printf("\n");
        }
    }

    printf("\n");
    printf("mepc    : %08x\n", regs[15]);
    printf("mstatus: %08x\n", regs[16]);

    if (trap_c_callback) {
        trap_c_callback();
    }

    while (1);
}
```

通过串口输出异常向量号、通用寄存器和mepc、mstatus状态寄存器

只有声明，可自定义实现的函数

trap_c 函数会通过串口打印输出异常向量号、CSR和GPR等寄存器的值

异常服务程序分析

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。

- 将 PC 恢复成 MEPC 寄存器的值，保证 CPU 从异常服务程序返回后可以从触发异常的地方重新执行指令。这就要求异常处理过程中将触发该异常的事件进行修复，避免再次触发异常。

注意

对于ECALL EBREAK指令造成的异常，异常返回后继续执行该指令造成异常，可在异常返回前将mepc修改加2/4从而执行下一条指令

```c
void (*trap_c_callback)(void);

void trap_c(uint32_t *regs)
{
    int i;
    uint32_t vec = 0;

    vec = __get_MCAUSE() & 0x3FF;

    printf("CPU Exception: NO.%d", vec);
    printf("\n");

    for (i = 0; i < 15; i++) {
        printf("x%d: %08x\t", i + 1, regs[i]);

        if ((i % 4) == 3) {
            printf("\n");
        }
    }

    printf("\n");
    printf("mepc    : %08x\n", regs[15]);
    printf("mstatus: %08x\n", regs[16]);

    if (trap_c_callback) {
        trap_c_callback();
    }

    while (1);
}
```

异常服务程序分析

- MSTATUS寄存器MIE域被恢复成 MPIE 的值即恢复异常发生前的值，MPIE 被设置为 1。

wujian100 sdk中设计的异常服务程序会最终进入死循环，

所以在异常服务程序结束没有手动恢复GPR的值、

没有调用**MRET**指令恢复现场

```c
void (*trap_c_callback)(void);

void trap_c(uint32_t *regs)
{
    int i;
    uint32_t vec = 0;

    vec = __get_MCAUSE() & 0x3FF;

    printf("CPU Exception: NO.%d", vec);
    printf("\n");

    for (i = 0; i < 15; i++) {
        printf("x%d: %08x\t", i + 1, regs[i]);

        if ((i % 4) == 3) {
            printf("\n");
        }
    }

    printf("\n");
    printf("mepc    : %08x\n", regs[15]);
    printf("mstatus: %08x\n", regs[16]);

    if (trap_c_callback) {
        trap_c_callback();
    }

    while (1);
}
```

RISC-V中断类型

- 定时器中断（Timer Interrupt）
- 软件中断（Software Interrupt）
- 调试中断（Debug Interrupt）Interrupt）
- 外部中断（External Interrupt）Interrupt）

定时器中断（Timer Interrupt）

RISC-V架构定义了系统平台中必须有一个定时器，并给该定时器定义了两个64位宽的寄存器用于计数和比较。系统必须以一种恒定的频率作为定时器的时钟，该时钟频率必须为低速的电源常开（Always-on）时钟，低速是为了省电，常开是为了提供准确的计时。

软件中断（Software Interrupt）

通过软件配置msip寄存器触发软件中断。

RISC-V中断类型

定时器中断（Timer Interrupt）

软件中断（Software Interrupt）

调试中断（Debug Interrupt）Interrupt）

外部中断（External Interrupt）Interrupt）

调试中断（Debug Interrupt）Interrupt）

当处理器核收到此中断之后，将进入调试模式。

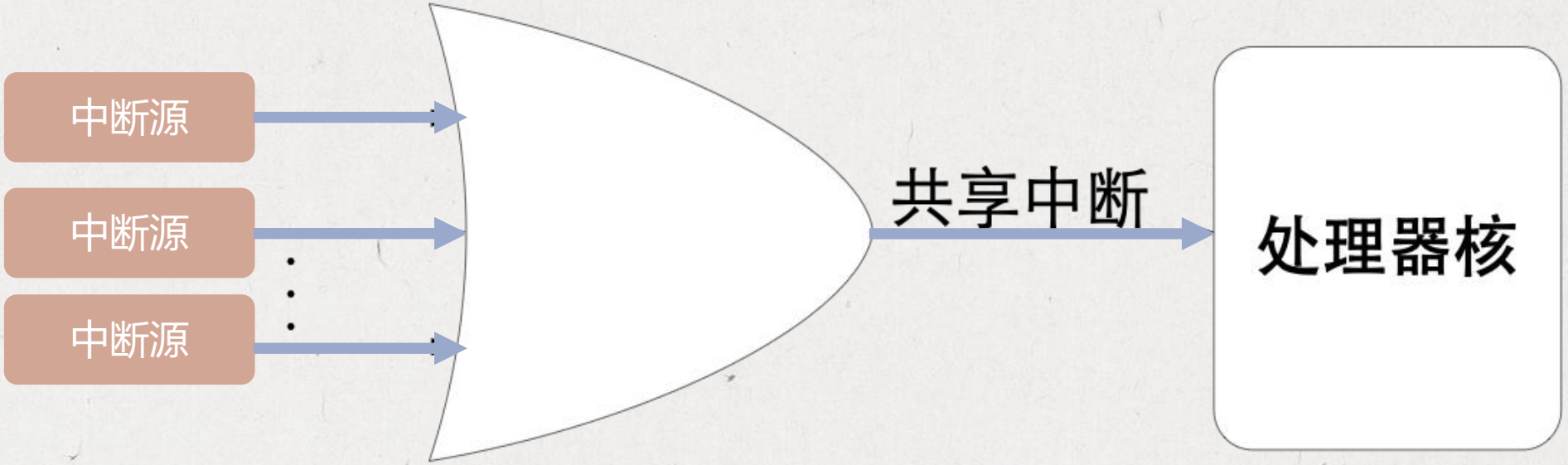
外部中断（External Interrupt）Interrupt）

外部中断来自于处理器核外部，如外部设备UART、GPIO等产生的中断。外部中断可直接作为处理器核的一个单比特输入信号，假如处理器需要支持很多个外部中断源，也可使用硬件的中断控制器来将多个外部中断源进行优先级仲裁和分发为一个信号。

RISC-V中断硬件控制器

共享中断

在中小规模硬件控制器的嵌入式系统中，对于多个中断源的情况，可以简单地将它们线或（Wired-OR）在一起，作为共享中断。



优点 硬件结构简单

缺点 需要CPU确定中断源，延迟较大，中断优先级、中断嵌套等问题均需交给软件处理。

RISC-V中断硬件控制器

CLINT + PLIC管理方式

- 局部中断
- 外部中断

处理器核局部中断控制器CLINT（Core Local Interrupts Controller）

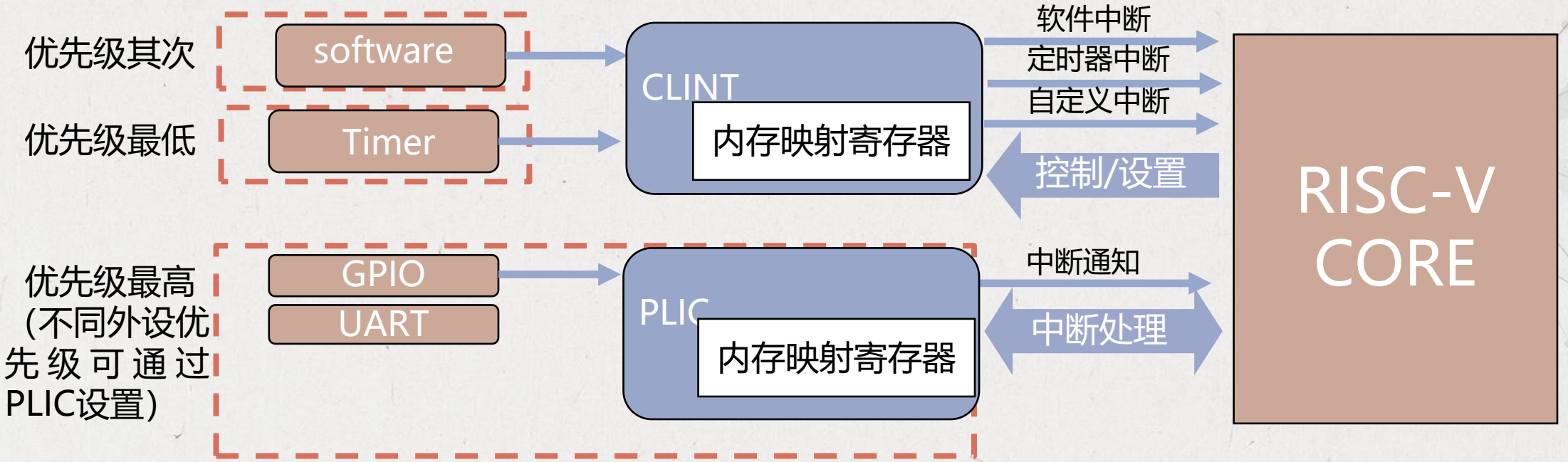
用于管理局部中断即定时器中断和软件中断和其它预留供扩展的自定义中断。

平台级别中断控制器PLIC（Platform Level Interrupt Controller）

用于管理外部中断。连接SoC系统的GPIO、UART、PWM等外设的外部中断源，将多个外部中断源仲裁为一个单比特的外部中断信号送入处理器核。

RISC-V中断硬件控制器

CLINT + PLIC管理方式



优点　扩展性好，可以利用PLIC管理更多的中断源

缺点　中断延迟仍较大，仍需软件读取内存映射寄存器确定中断源

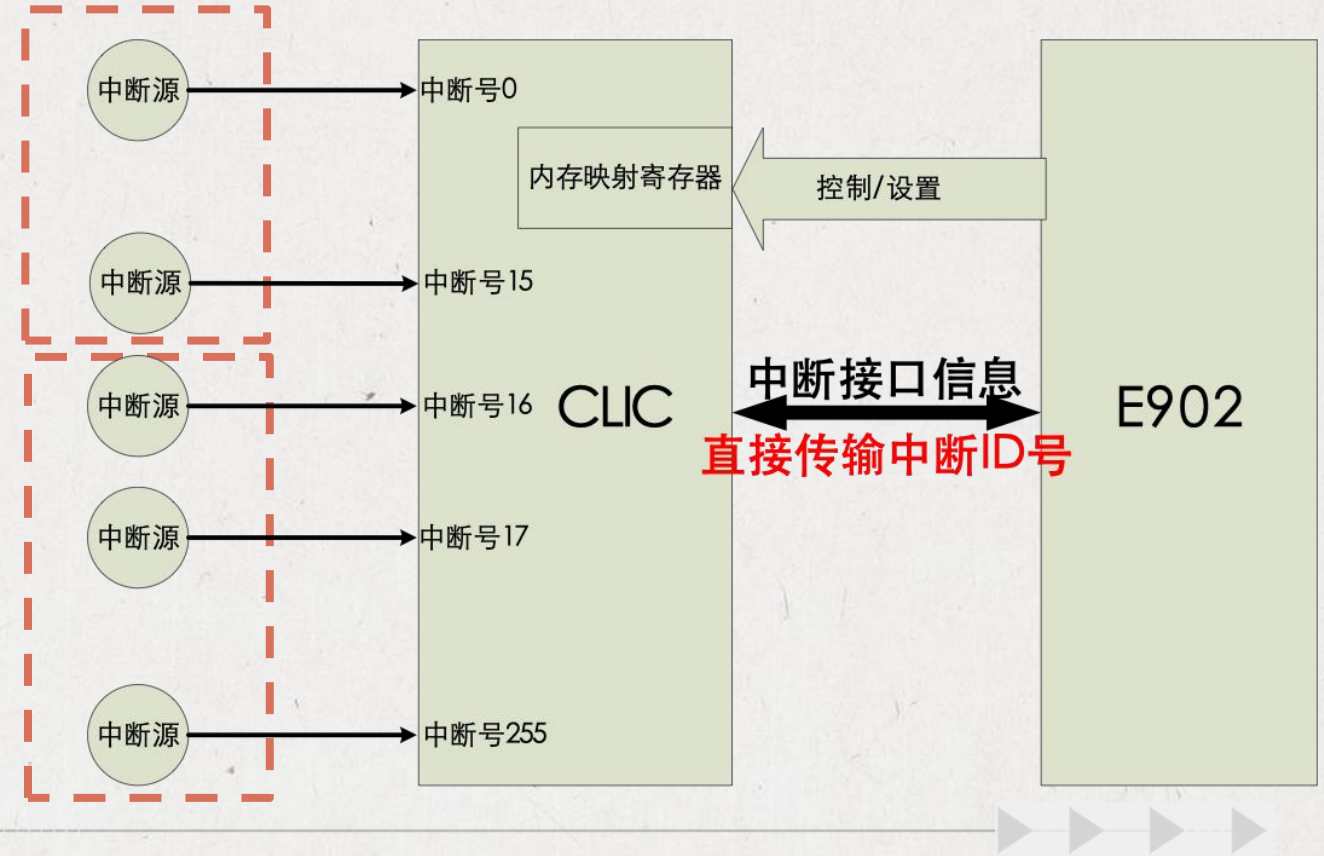
RISC-V中断硬件控制器

CLIC管理方式

CLIC（Core-Local Interrupt Controller）可以看作是PLIC与CLINT的合并与简化，用于对所有中断源进行采样，优先级仲裁和分发。E902的CLIC控制器支持最多240个外部中断源，并且兼容CLINT的至多16 个中断。

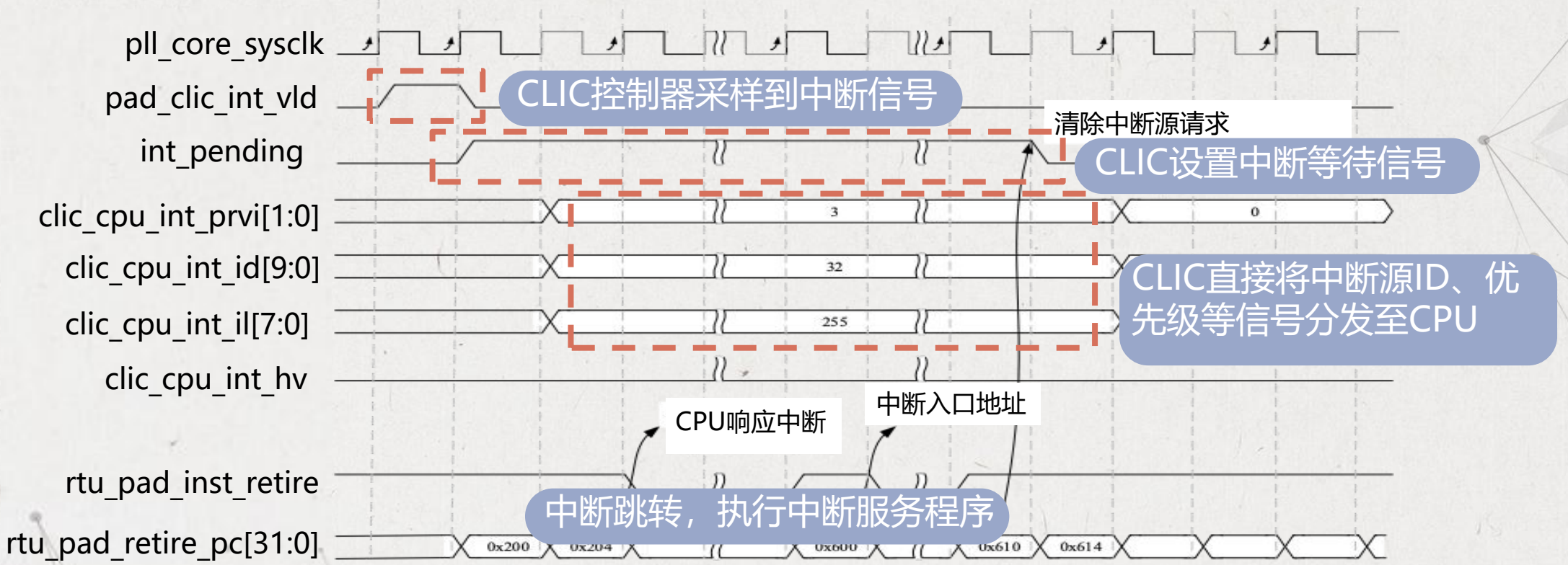
兼容CLINT的16个中断源。

除CLINT中断外，E902最多可扩展240个中断源。wujian100 SoC只扩展使用了48个



RISC-V中断硬件控制器

CLIC管理方式



CPU无需再通过总线读取状态寄存器来获取中断源

RISC-V中断硬件控制器

E902内核的CLIC中断控制器为每个中断目标提供了4个memory-mapped 的控制寄存器，用于配置中断源的各个属性如优先级。

其中CLICCFG寄存器的nvbits位恒为1，代表CLIC控制器支持硬件矢量模式中断。

表 10.1: CLIC 地址映射

地址	名称	类型	初始值	描述
0xE0800000	CLICCFG	RW	0x1	CLIC 配置寄存器
0xE0800004	CLICINFO	RO	详见计时器中断	CLIC 信息寄存器
0xE0800008	MINTTHRESH	RW	0x0	中断阈值寄存器
0xE0801000+4*i	CLICINTIP[i]	R or RW	0x0	中断源 i 等待寄存器
0xE0801001+4*i	CLICINTIE[i]	RW	0x0	中断源 i 使能寄存器
0xE0801002+4*i	CLICINTATTR[i]	RW	0x0	中断源 i 属性寄存器
0xE0801003+4*i	CLICINTCTRL[i]	RW	0x0	中断源 i 控制寄存器

区别

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。

CLICNTATTR寄存器的shv域可以设置该中断源为矢量中断或非矢量中断。

wujian100 中的中断默认设置为矢量中断，后续中断处理过程的介绍都基于矢量中断。

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域为 1，以及该中断对应的 CLICINTIE 寄存器中的使能位为 1。

中断响应时硬件自动完成处理，所有步骤在一个处理器时钟周期完成：

在进入异常或中断之后，全局中断使能位会被硬件自动更新成0，即RISC-V架构定义的硬件机制默认无法支持硬件中断嵌套行为，但是可以通过软件的方式将其写1达到中断嵌套的目的，但在开启中断前注意保存CSR和上下文等。

中断处理

硬件矢量中断的中断服务程序入口都是统一由 MTVT加中断号偏移量指定的模式，因此需要在开启中断前应先搭建中断向量表并将中断向量表的地址写入MTVT寄存器，wujian100 SDK中在Startup.s启动文件完成了以上操作。

搭建__Vectors异常向量表

```
.section .vectors
    .align  6
    .globl  __Vectors
    .type   __Vectors, @object
__Vectors:
    .long   Default_Handler
    .long   Default_Handler
    .long   Default_Handler
    .long   tspend_handler
    .long   Default_Handler
    .long   Default_Handler
    .long   Default_Handler
    .long   Default_IRQHandler
    .long   Default_Handler
    .long   Default_Handler
    .long   Default_Handler
    .long   Default_Handler
    .long   Default_Handler
    .long   Default_Handler
    .long   Default_Handler
    .long   Default_Handler
    /* External interrupts */
    .long   Default_IRQHandler
    .long   Default_IRQHandler
    .long   Default_IRQHandler
    .long   Default_IRQHandler
    .long   Default_IRQHandler
    .long   Default_IRQHandler
    .long   Default_IRQHandler
```

兼容CLINT的16个中断源及其中断服务函数

对于定时器和其它外设中断，处理器会跳转到Default_IRQHandler函数执行

```
la      a0, __Vectors
csrw    mtvt, a0
```

将异常向量表首地址写入mtvt

Table 1-4 Interrupt source

Number	Interrupt Source
6:0	REV
1	CoreTim
15:8	REV
16	GPIO0
18:17	TIM0[1:0]
20:19	TIM1[1:0]
22:21	TIM2[1:0]
24:23	TIM3[1:0]
25	PWM

部分中断源及其中断号
详细可参考wujian100开源用户手册

中断服务程序分析

根据是否配置使用Alios的Rhino RTOS内核，中断服务程序会有所不同

```
Default_IRQHandler:
    addi    sp, sp, -48
    sw      t0, 4(sp)
    sw      t1, 8(sp)
    csrr    t0, mepc
    csrr    t1, mcause
    sw      t1, 40(sp)
    sw      t0, 44(sp)
    csrs    mstatus, 8

    sw      ra, 0(sp)
    sw      t2, 12(sp)
    sw      a0, 16(sp)
    sw      a1, 20(sp)
    sw      a2, 24(sp)
    sw      a3, 28(sp)
    sw      a4, 32(sp)
    sw      a5, 36(sp)

    andi    t1, t1, 0x3FF
    slli    t1, t1, 2

    la      t0, g_irqvector
    add     t0, t0, t1
    lw      t2, (t0)
    jalr    t2
```

开辟堆栈空间并将mepc、mcause进行保存

开启全局中断

保存上下文，将用到的GPR压入堆栈

获取中断号并将偏移地址写进t1

再次跳转到以g_irqvector为基址的程序

```
void (*g_irqvector[48])(void);

void irq_vectors_init(void)
{
    int i;

    for (i = 0; i < 48; i++) {
        g_irqvector[i] = Default_Handler;
    }

    g_irqvector[CORET_IRQn] = CORET_IRQHandler;
}
```

①由于RISC-V硬件机制不支持中断嵌套，因此可以在软件中手动开启中断以支持中断嵌套，但在开启中断前应先将mepc、mcause寄存器压栈保存

②保护上下文，将函数调用相关的GPR压栈保护

③获取中断号并计算出偏移地址，随后跳转到g_irqvector加偏移地址对应的程序执行。在使用外设中断前，应先将该外设的中断服务程序地址写入g_irqvector数组的指定位置

中断服务程序分析

```
csrc    mstatus, 8
```

关闭全局中断

```
lw      a1, 40(sp)
andi    a0, a1, 0x3FF

/* clear pending */
li      a2, 0xE000E100
add     a2, a2, a0
lb      a3, 0(a2)
li      a4, 1
not     a4, a4
and     a5, a4, a3
sb      a5, 0(a2)
```

清除CLIC对应中断请求位，防止重复响应中断（主要针对电平中断）

```
li      t0, MSTATUS_PRV1
csrs    mstatus, t0
csrw    mcause, a1
lw      t0, 44(sp)
csrw    mepc, t0
lw      ra, 0(sp)
lw      t0, 4(sp)
lw      t1, 8(sp)
lw      t2, 12(sp)
lw      a0, 16(sp)
lw      a1, 20(sp)
lw      a2, 24(sp)
lw      a3, 28(sp)
lw      a4, 32(sp)
lw      a5, 36(sp)
```

上下文恢复，将CSR和函数调用相关的GPR出栈恢复

```
addi    sp, sp, 48
mret
```

```
void SystemInit(void)
{
    int i;

    CLIC->CLICCFG = 0x4UL;

    for (i = 0; i < 12; i++) {
        CLIC->INTIP[i] = 0;
    }

    drv_irq_enable(Machine_Software_IRQn);

#ifdef CONFIG_KERNEL_NONE
    _system_init_for_baremetal();
#else
    _system_init_for_kernel();
#endif
}
```

预先清除中断请求位

执行完以g_irqvector为基址的程序后会执行的操作：

①软件将中断关闭以保证后续操作不被打断

②清除CLIC中中断请求位，防止重复响应中断。另外在进入main主函数之前，程序还会执行SystemInit函数预先清除CLIC中断请求位。

③退出中断前，进行上下文恢复，将CSR和函数调用相关的GPR出栈恢复。

中断返回

```
csrc    mstatus, 8

lw      a1, 40(sp)
andi    a0, a1, 0x3FF

/* clear pending */
li      a2, 0xE000E100
add     a2, a2, a0
lb      a3, 0(a2)
li      a4, 1
not     a4, a4
and     a5, a4, a3
sb      a5, 0(a2)

li      t0, MSTATUS_PRV1
csrs    mstatus, t0
csrw    mcause, a1
lw      t0, 44(sp)
csrw    mepc, t0
lw      ra, 0(sp)
lw      t0, 4(sp)
lw      t1, 8(sp)
lw      t2, 12(sp)
lw      a0, 16(sp)
lw      a1, 20(sp)
lw      a2, 24(sp)
lw      a3, 28(sp)
lw      a4, 32(sp)
lw      a5, 36(sp)

addi    sp, sp, 48
mret
```

上下文恢复，将CSR和函数调用相关的GPR出栈恢复

执行MRET指令以退出中断

中断服务程序的退出必须使用 MRET 指令完成，通过执行 MRET 指令将响应中断之前的 CPU 现场进行恢复。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。

wujian100中断操作总结

在启动文件中，进入main函数前完成，无需实现

搭建中断向量表

将__Vectors地址写入mtvt寄存器

清除中断请求位，开启全局中断

需要用户在Main函数中实现完成

将外设中断程序地址写入g_irqvector数组的指定位置

配置CLIC特定中断的属性（如优先级等）并开启指定外设中断

中断前完成的配置操作

处理器跳转到MTVT加中断号偏移量为地址的中断服务程序

开辟堆栈空间并将mepc、mcause压栈保护，以支持中断嵌套

开启全局中断软件实现支持中断嵌套

保护上下文将函数调用相关的GPR压入堆栈

根据中断号跳转到g_irqvector数组中指定的程序入口

关闭中断以保证后续操作不被打断

恢复上下文将CSR和函数调用相关的GPR压出堆栈恢复

中断过程中完成的操作，此部分无需修动

异常与中断机制比较

异常

- 异常是由”内因”造成的
- 异常不可屏蔽，出现异常立刻响应
- 异常不支持嵌套，CPU在处理异常服务程序时若再次遇到异常则会锁死
- 异常服务程序地址由 MTVEC 寄存器指定
- wujian100 SDK中将所有异常按同一标准处理，通过串口打印异常向量号和寄存器的值

中断

- 中断是由外围设备造成的
- 中断可以通过全局中断、外设中断决定是否屏蔽
- 中断支持嵌套，但需要通过软件实现（RISC-V硬件架构不支持）
- 矢量中断的中断服务程序地址由MTVT加中断号偏移量指定，非矢量中断则统一由 MTVEC 寄存器指定
- wujian100 SDK中将根据中断号执行不同的中断服务程序

注意 异常或中断响应时处理器自动进入机器模式，返回时恢复先前模式

异常与中断机制比较

注意

异常或中断响应时处理器自动进入机器模式，返回时恢复先前模式

总结

异常与中断机制是处理器实现的重要的一环，对处理器指令的正常执行和实时响应至关重要。