ISCAS NIST

循序渐进，学习开发一个 RISC-V 上的操作系统

第 11 章 外部设备中断

汪辰

2021/4

本章内容

- RISC-V 中断（Interrupt）的分类
- RISC-V 中断编程中涉及的寄存器
- RISC-V 中断处理流程
- PLIC 介绍
- 采用中断方式从 UART 实现输入

参考

- 【参考 1】： The RISC-V Instruction Set Manual，Volume I: Unprivileged ISA，Document Version 20191213
- 【参考 2】: The RISC-V Instruction Set Manual，Volume II: Privileged Architecture，Document Version 20190608-Priv-MSU-Ratified
- 【参考 3】: SiFive FU540-C000 Manual，v1p0
- 【参考 4】: RISC-V Platform-Level Interrupt Controller Specification：https://github.com/riscv/riscv-plic-spec

本章内容

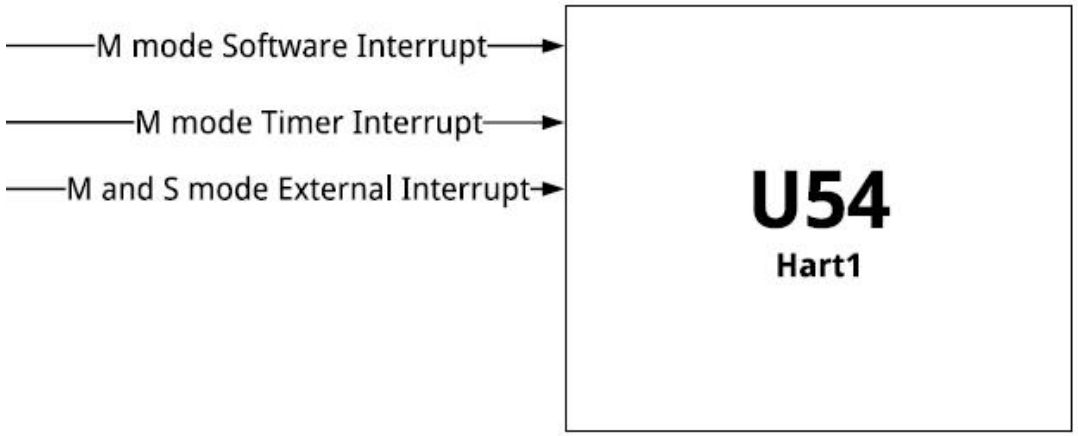
- RISC-V 中断（Interrupt）的分类
- RISC-V 中断编程中涉及的寄存器
- RISC-V 中断处理流程
- PLIC 介绍
- 采用中断方式从 UART 实现输入

RISC-V 中断（Interrupt）的分类

- 本地（Local）中断
  - software interrupt
  - timer interrupt
- 全局（Global）中断
  - externel interrupt

Interrupt	Exception Code	Description
1	0	User software interrupt
1	1	Supervisor software interrupt
1	2	*Reserved for future standard use*
1	3	Machine software interrupt
1	4	User timer interrupt
1	5	Supervisor timer interrupt
1	6	*Reserved for future standard use*
1	7	Machine timer interrupt
1	8	User external interrupt
1	9	Supervisor external interrupt
1	10	*Reserved for future standard use*
1	11	Machine external interrupt
1	12–15	*Reserved for future standard use*
1	≥16	*Reserved for platform use*

【参考 2】Table 3.6: Machine cause register (`mcause`) values after trap.



【参考 3】Figure 3: FU540-C000 Interrupt Architecture Block Diagram.

本章内容

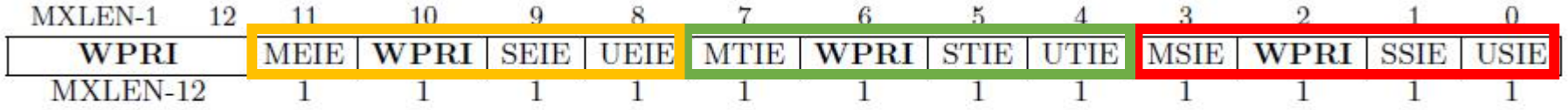
- RISC-V 中断（Interrupt）的分类
- **RISC-V 中断编程中涉及的寄存器**
- RISC-V 中断处理流程
- PLIC 介绍
- 采用中断方式从 UART 实现输入

RISC-V Trap （中断）处理中涉及的寄存器

寄存器	用途说明
mtvec（Machine Trap-Vector Base-Address）	它保存发生异常时处理器需要跳转到的地址。
mepc（Machine Exception Program Counter）	当 trap 发生时，hart 会将发生 trap 所对应的指令的地址值（pc）保存在 mepc 中。
mcause（Machine Cause）	当 trap 发生时，hart 会设置该寄存器通知我们 trap 发生的原因。
mtval（Machine Trap Value）	它保存了 exception 发生时的附加信息：譬如访问地址出错时的地址信息、或者执行非法指令时的指令本身，对于其他异常，它的值为 0。
mstatus（Machine Status）	用于跟踪和控制 hart 的当前操作状态（特别地，包括关闭和打开全局中断）。
mscratch（Machine Scratch）	Machine 模式下专用寄存器，我们可以自己定义其用法，譬如用该寄存器保存当前在 hart 上运行的 task 的上下文（context）的地址。
mie（Machine Interrupt Enable）	用于进一步控制（打开和关闭）software interrupt/timer interrupt/external interrupt
mip（Machine Interrupt Pending）	它列出目前已发生等待处理的中断。

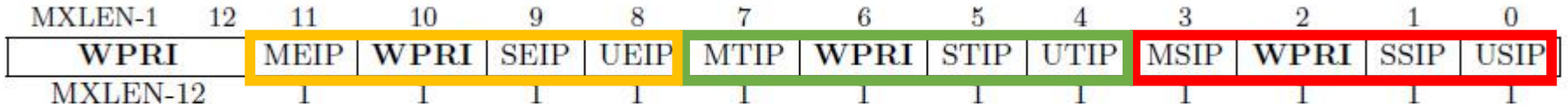
寄存器 mie、mip

- mie(Machine Interrupt Enable)：打开（1）或者关闭（0）M/S/U 模式下对应的 External/Timer/Software 中断



【参考 2】Figure 3.12: Machine interrupt-enable register (mie).

- mip(Machine Interrupt Pending)：获取当前 M/S/U 模式下对应的 External/Timer/Software 中断是否发生



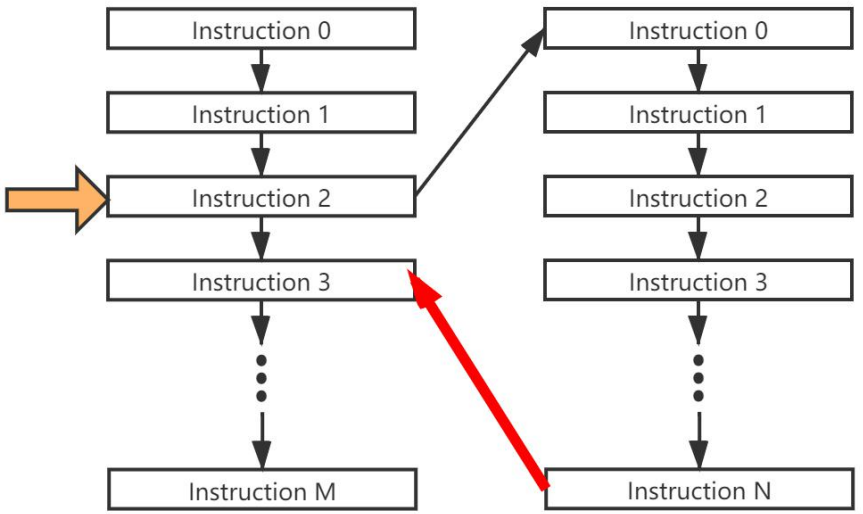
【参考 2】Figure 3.11: Machine interrupt-pending register (mip).

本章内容

- RISC-V 中断（Interrupt）的分类
- RISC-V 中断编程中涉及的寄存器
- **RISC-V 中断处理流程**
- PLIC 介绍
- 采用中断方式从 UART 实现输入

中断发生时 Hart 自动执行如下状态转换

- 把 mstatus 的 MIE 值复制到 MPIE 中，清除 mstatus 中的 MIE 标志位，效果是中断被禁止。
- 当前的 PC 的下一条指令地址被复制到 mepc 中，同时 PC 被设置为 mtvec。注意如果我们设置 mtvec.MODE = vetcored，PC = mtvec.BASE + 4 × exception-code。
- 根据 interrupt 的种类设置 mcause，并根据需要为 mtval 设置附加信息。
- 将 trap 发生之前的权限模式保存在 mstatus 的 MPP 域中，再把 hart 权限模式更改为 M。



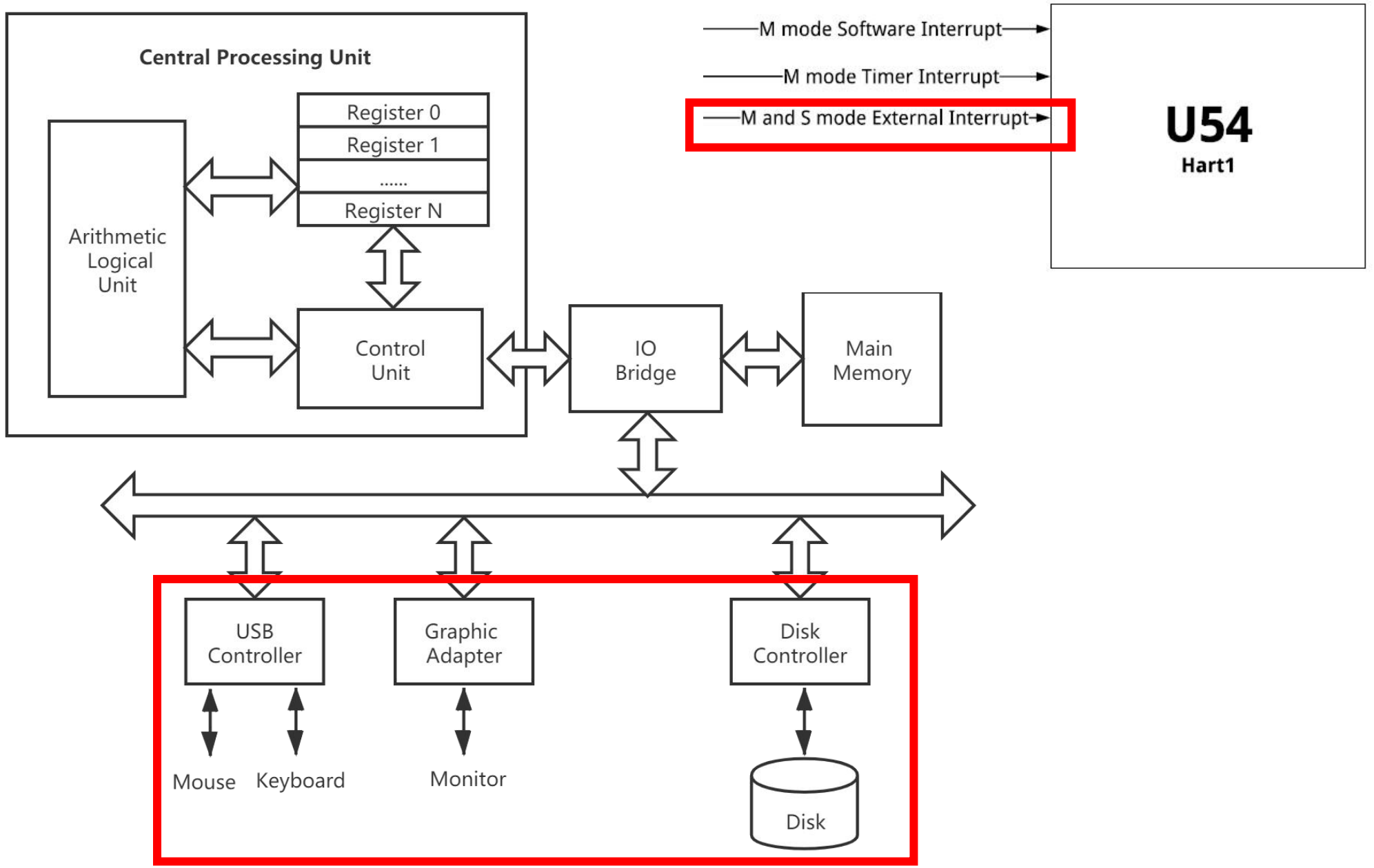
退出中断： 编程调用 MRET 指令

- 以在 M 模式下执行 mret 指令为例，会执行如下操作:
  - 当前 Hart 的权限级别 = mstatus.MPP; mstatus.MPP = U（如果 hart 不支持 U 则为 M）
  - mstatus.MIE = mstatus.MPIE; mstatus.MPIE = 1
  - pc = mepc

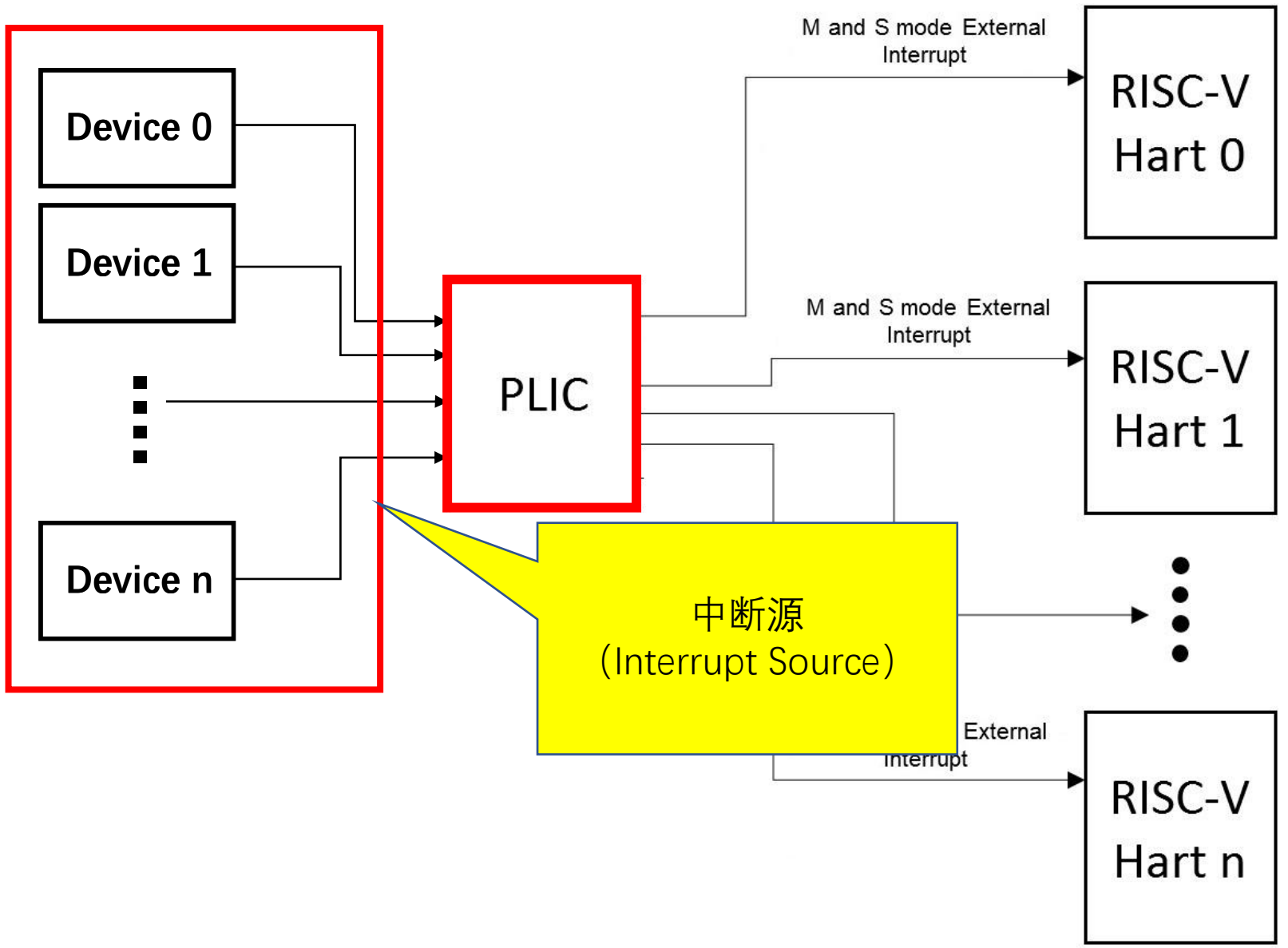
本章内容

- RISC-V 中断（Interrupt）的分类
- RISC-V 中断编程中涉及的寄存器
- RISC-V 中断处理流程
- **PLIC 介绍**
- 采用中断方式从 UART 实现输入

外部中断（external interrupt）



Platform-Level Interrupt Controller



PLIC Interrupt Source

https://github.com/qemu/qemu/blob/master/include/hw/riscv/virt.h

```
enum {
    UART0_IRQ = 10,
    RTC_IRQ = 11,
    VIRTIO_IRQ = 1, /* 1 to 8 */
    VIRTIO_COUNT = 8,
    PCIE_IRQ = 0x20, /* 32 to 35 */
    VIRTIO_NDEV = 0x35 /* Arbitrary maximum number of interrupts */
};
```

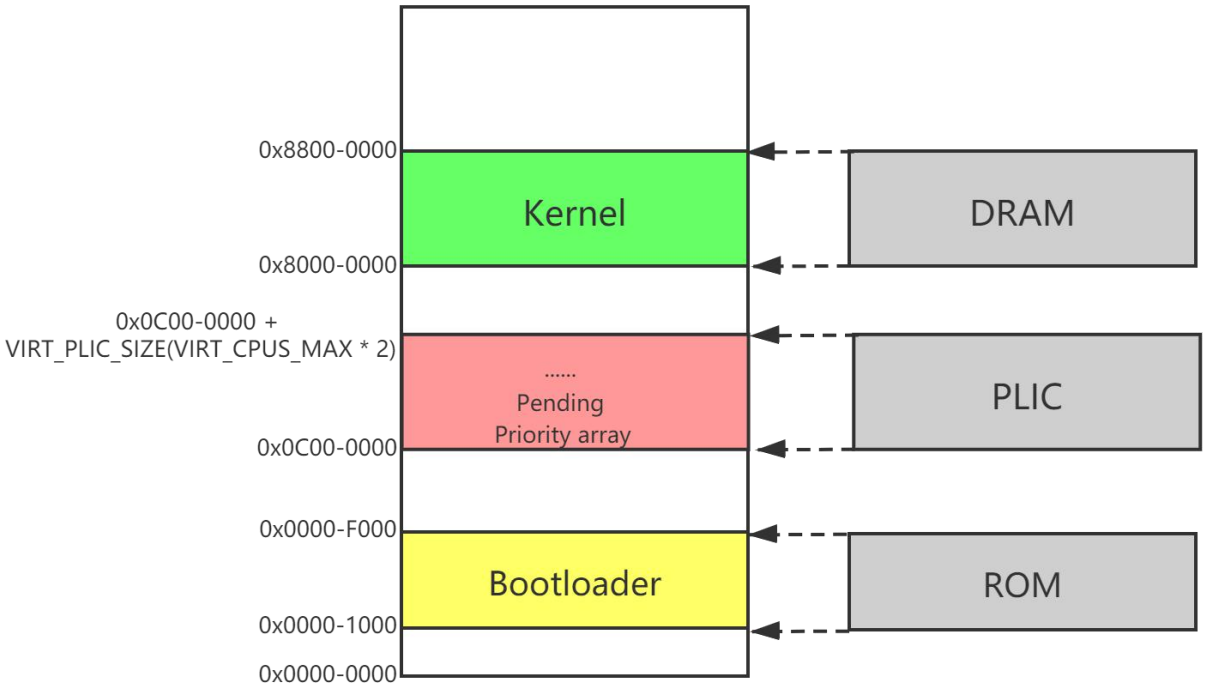
- **Interrupt Source ID 范围：1 ~ 53（0x35）**
- **0 预留不用**

PLIC 编程接口 - 寄存器

- RISC-V 规范规定，PLIC 的寄存器编址采用内存映射（memory map）方式。每个寄存器的宽度为 32-bit。
- 具体寄存器编址采用 base + offset 的格式，且 base 由各个特定 platform 自己定义。针对 QEMU-virt，其 PLIC 的设计参考了 FU540-C000，base 为 0x0c000000。 `#define PLIC_BASE 0x0c000000L`

https://github.com/qemu/qemu/blob/master/hw/riscv/virt.c

```
static const MemMapEntry virt_memmap[] = {
    [VIRT_DEBUG] =       {        0x0,         0x100 },
    [VIRT_MROM] =        {     0x1000,        0xf000 },
    [VIRT_TEST] =        {   0x100000,        0x1000 },
    [VIRT_RTC] =         {   0x101000,        0x1000 },
    [VIRT_CLINT] =       {  0x2000000,       0x10000 },
    [VIRT_PCIE_PIO] =    {  0x3000000,       0x10000 },
    [VIRT_PLIC] =        {  0xc000000, VIRT_PLIC_SIZE(VIRT_CPUS_MAX * 2) },
    [VIRT_UART0] =       { 0x10000000,         0x100 },
    [VIRT_VIRTIO] =      { 0x10001000,        0x1000 },
    [VIRT_FLASH] =       { 0x20000000,     0x4000000 },
    [VIRT_PCIE_ECAM] =   { 0x30000000,    0x10000000 },
    [VIRT_PCIE_MMIO] =   { 0x40000000,    0x40000000 },
    [VIRT_DRAM] =        { 0x80000000,           0x0 },
};
```



PLIC 编程接口 - 寄存器

可编程寄存器	功能描述	内存映射地址
Priority	设置某一路中断源的优先级。	BASE + (interrupt-id) * 4

- 每个 PLIC 中断源对应一个寄存器，用于配置该中断源的优先级。
- QEMU-virt 支持 7 个优先级。 0 表示对该中断源禁用中断。其余优先级，1 最低，7 最高。
- 如果两个中断源优先级相同，则根据中断源的 ID 值进一步区分优先级，ID 值越小的优先级越高。

```
#define PLIC_PRIORITY(id) (PLIC_BASE + (id) * 4)

*(uint32_t*)PLIC_PRIORITY(UART0_IRQ) = 1;
```

PLIC 编程接口 - 寄存器

可编程寄存器	功能描述	内存映射地址
Pending	用于指示某一路中断源是否发生。	BASE + 0x1000 + ((interrupt-id) / 32) * 4

- 每个 PLIC 包含 2 个 32 位的 Pending 寄存器，每一个 bit 对应一个中断源，如果为 1 表示该中断源上发生了中断（进入 Pending 状态），有待 hart 处理，否则表示该中断源上当前无中断发生。
- Pending 寄存器中断的 Pending 状态可以通过 claim 方式清除。
- 第一个 Pending 寄存器的第 0 位对应不存在的 0 号中断源，其值永远为 0。

PLIC 编程接口 - 寄存器

可编程寄存器	功能描述	内存映射地址
Enable	针对某个 hart 开启或者关闭某一路中断源。	BASE + 0x2000 + (hart) * 0x80

- 每个 Hart 有 2 个 Enable 寄存器 （Enable1 和 Enable2）用于针对该 Hart 启动或者关闭某路中断源。
- 每个中断源对应 Enable 寄存器的一个 bit，其中 Enable1 负责控制 1 ~ 31 号中断源；Enable2 负责控制 32 ~ 53 号中断源。 将对应的 bit 位设置为 1 表示使能该中断源，否则表示关闭该中断源。

```
#define PLIC_MENABLE(hart) (PLIC_BASE + 0x2000 + (hart) * 0x80)

*(uint32_t*)PLIC_MENABLE(hart)= (1 << UART0_IRQ);
```

PLIC 编程接口 - 寄存器

可编程寄存器	功能描述	内存映射地址
Threshold	针对某个 hart 设置中断源优先级的阈值。	BASE + 0x200000 + (hart) * 0x1000

- 每个 Hart 有 1 个 Threshold 寄存器用于设置中断优先级的阈值。
- 所有小于或者等于（<=）该阈值的中断源即使发生了也会被 PLIC 丢弃。特别地，当阈值为 0 时允许所有中断源上发生的中断；当阈值为 7 时丢弃所有中断源上发生的中断。

```
#define PLIC_MTHRESHOLD(hart) (PLIC_BASE + 0x200000 + (hart) * 0x1000)

*(uint32_t*)PLIC_MTHRESHOLD(hart) = 0;
```

PLIC 编程接口 - 寄存器

可编程寄存器	功能描述	内存映射地址
Claim/Complete	详见下描述。	BASE + 0x200004 + (hart) * 0x1000

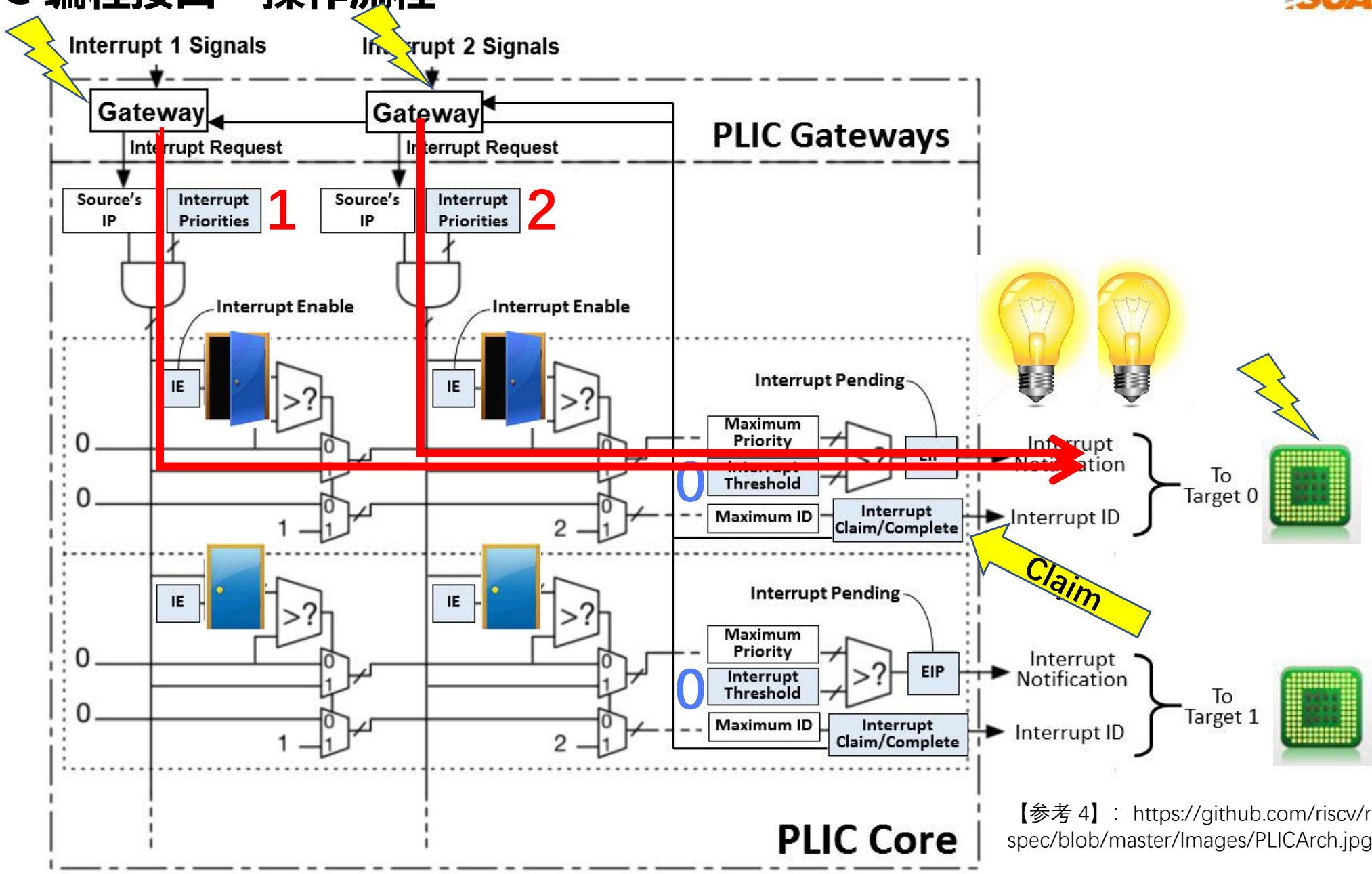
- Claim 和 Complete 是同一个寄存器，每个 Hart 一个。
- 对该寄存器执行读操作称之为 Claim，即获取当前发生的最高优先级的中断源 ID。Claim 成功后会清除对应的 Pending 位。
- 对该寄存器执行写操作称之为 Complete。所谓 Complete 指的是通知 PLIC 对该路中断的处理已经结束。

```
#define PLIC_MCLAIM(hart) (PLIC_BASE + 0x200004 + (hart) * 0x1000)
#define PLIC_MCOMPLETE(hart) (PLIC_BASE + 0x200004 + (hart) * 0x1000)
```

```
int plic_claim(void)
{
        int hart = r_tp();
        int irq = *(uint32_t*)PLIC_MCLAIM(hart);
        return irq;
}
```

```
void plic_complete(int irq)
{
        int hart = r_tp();
        *(uint32_t*)PLIC_MCOMPLETE(hart) = irq;
}
```

PLIC 编程接口 - 操作流程



【参考 4】：https://github.com/riscv/riscv-plic-spec/blob/master/Images/PLICArch.jpg

本章内容

- RISC-V 中断（Interrupt）的分类
- RISC-V 中断编程中涉及的寄存器
- RISC-V 中断处理流程
- PLIC 介绍
- **采用中断方式从 UART 实现输入**

UART 中断处理

code/os/06-interrupts/plic.c

```
void plic_init(void)
{
        int hart = r_tp();

        *(uint32_t*)PLIC_PRIORITY(UART0_IRQ) = 1;

        *(uint32_t*)PLIC_MENABLE(hart)= (1 << UART0_IRQ);

        *(uint32_t*)PLIC_MTHRESHOLD(hart) = 0;

        /* enable machine-mode external interrupts. */
        w_mie(r_mie() | MIE_MEIE);

        /* enable machine-mode global interrupts. */
        w_mstatus(r_mstatus() | MSTATUS_MIE);
}
```

```
int plic_claim(void)
{
        int hart = r_tp();
        int irq = *(uint32_t*)PLIC_MCLAIM(hart);
        return irq;
}
```

```
void plic_complete(int irq)
{
        int hart = r_tp();
        *(uint32_t*)PLIC_MCOMPLETE(hart) = irq;
}
```

UART 中断处理

code/os/06-interrupts/trap.c

```
reg_t trap_handler(reg_t epc, reg_t cause)
{
        ......
        if (cause & 0x80000000) {
                /* Asynchronous trap - interrupt */
                switch (cause_code) {
                ......
                case 11:
                        uart_puts("external interruption!\n");
                        external_interrupt_handler();
                        break;
        ......

        return return_pc;
}
```

```
void external_interrupt_handler()
{
        int irq = plic_claim();
        if (irq == UART0_IRQ){
                uart_isr();
        ......
        }
        if (irq) {
                plic_complete(irq);
        }
}
```

UART 中断处理

code/os/06-interrupts/uart.c

```c
void uart_init()
{
        /* disable interrupts. */
        uart_write_reg(IER, 0x00);
        ......
        /* enable receive interrupts. */
        uint8_t ier = uart_read_reg(IER);
        uart_write_reg(IER, ier | (1 << 0));
}
```

```c
int uart_getc(void)
{
        if (uart_read_reg(LSR) & LSR_RX_READY){
                return uart_read_reg(RHR);
        } else {
                return -1;
        }
}

void uart_isr(void)
{
        while (1) {
                int c = uart_getc();
                if (c == -1) {
                        break;
                } else {
                        uart_putc((char)c);
                        uart_putc('\n');
                }
        }
}
```

```c
trap_handler(reg_t epc, reg_t cause)
```

```c
external_interrupt_handler()
```

练习 11-1

谢 谢

欢迎交流合作