ISCAS NIST

循序渐进，学习开发一个 RISC-V 上的操作系统

第 13 章 抢占式多任务

汪辰

2021/4

本章内容

- 抢占式多任务
- 抢占式多任务的设计
- 兼容协作式多任务

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

本章内容

- 抢占式多任务
- 抢占式多任务的设计
- 兼容协作式多任务

多任务系统的分类

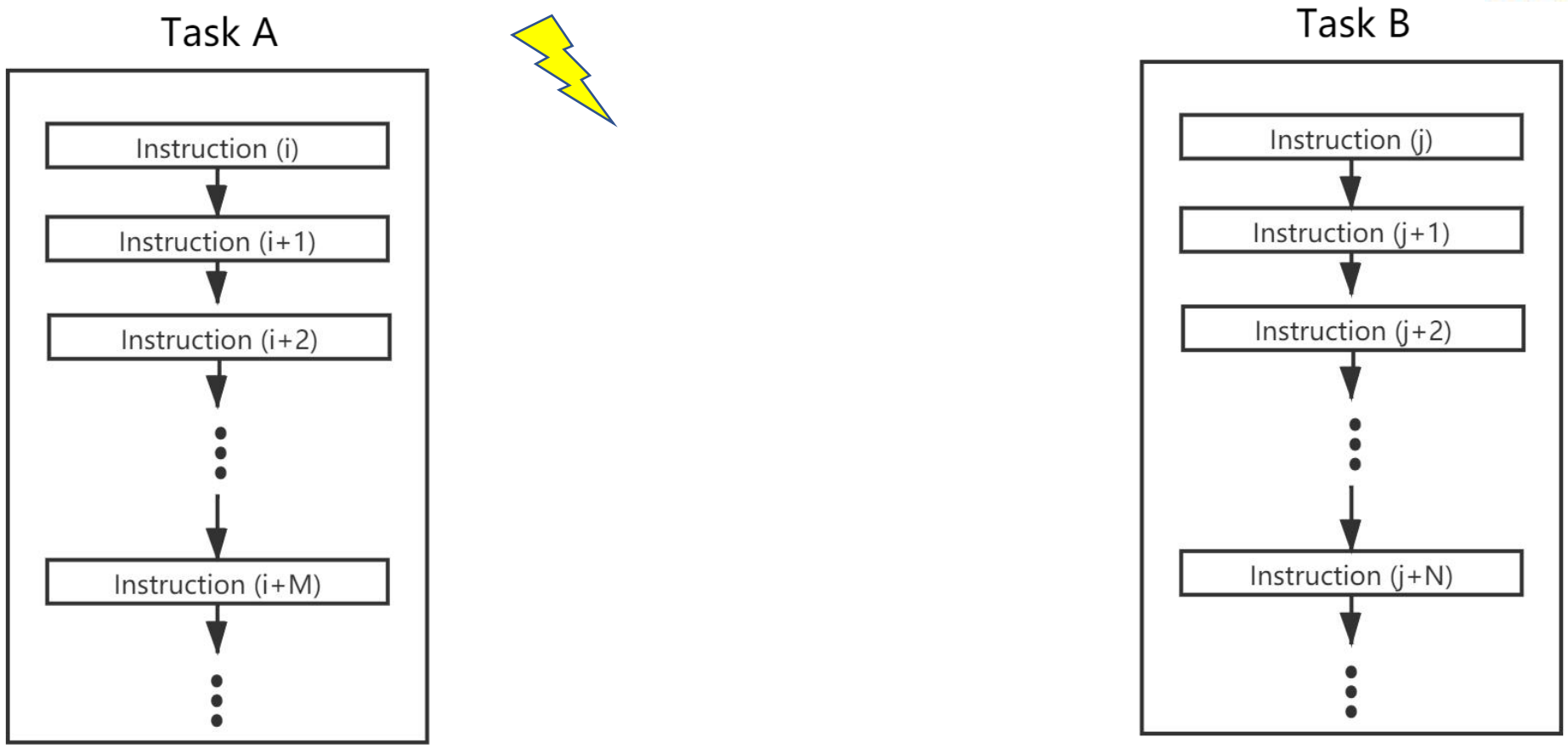
- **协作式多任务 (Cooperative Multitasking)**：协作式环境下，下一个任务被调度的前提是当前任务主动放弃处理器。

- **抢占式多任务 (Preemptive Multitasking)**：抢占式环境下，操作系统完全决定任务调度方案，操作系统可以剥夺当前任务对处理器的使用，将处理器提供给其它任务。

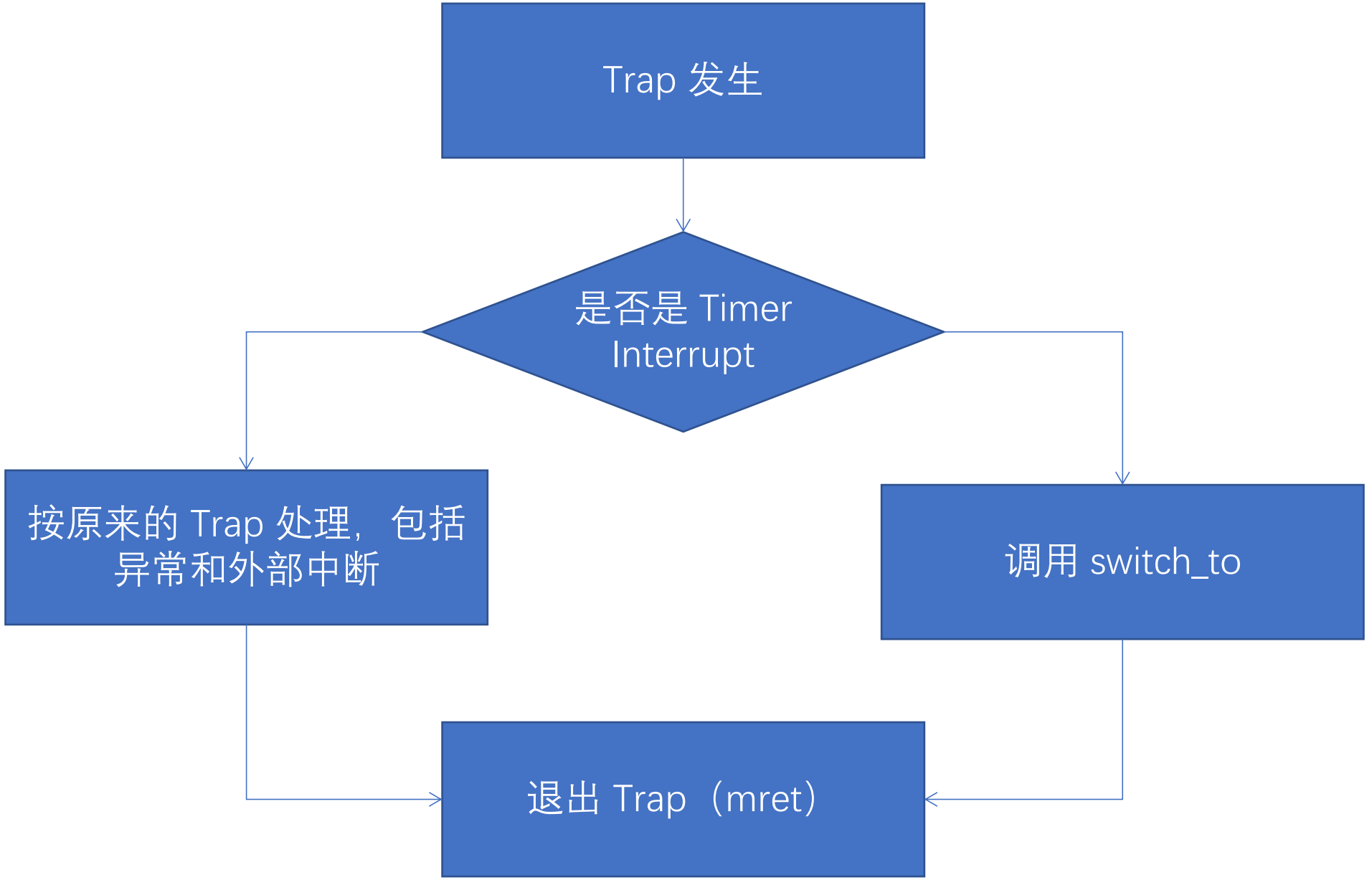
本章内容

- 抢占式多任务
- 抢占式多任务的设计
- 兼容协作式多任务

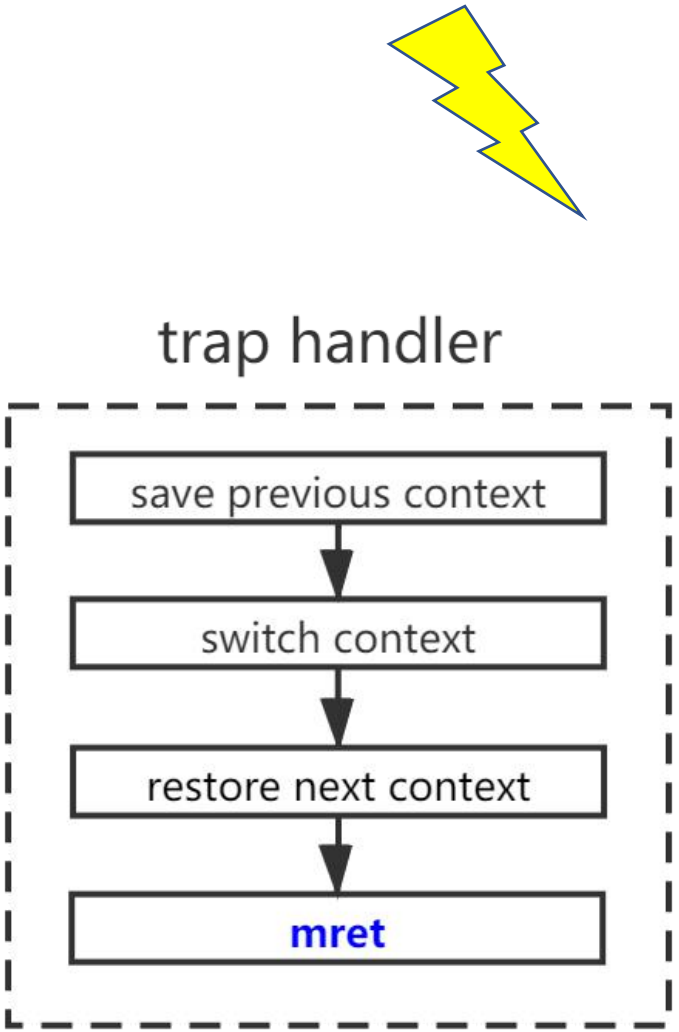
抢占式多任务的设计


Task A
Instruction (i)
Instruction (i+1)
Instruction (i+2)
Instruction (i+M)
Task B
Instruction (j)
Instruction (j+1)
Instruction (j+2)
Instruction (j+N)

抢占式多任务的设计


Trap 发生
是否是 Timer Interrupt
按原来的 Trap 处理，包括异常和外部中断
调用 switch_to
退出 Trap（mret）

抢占式多任务的设计



code/os/08-preemptive/entry.S

```
trap_vector:
        ......
        call    trap_handler
        ......
        mret

switch_to:
        ......
        mret
```

code/os/08-preemptive/trap.c

```
reg_t trap_handler(reg_t epc, reg_t cause)
{
        ......
        if (cause & 0x80000000) {
                switch (cause_code) {
                ......
                case 7:
                        timer_handler();
                        break;
                ......
}
```

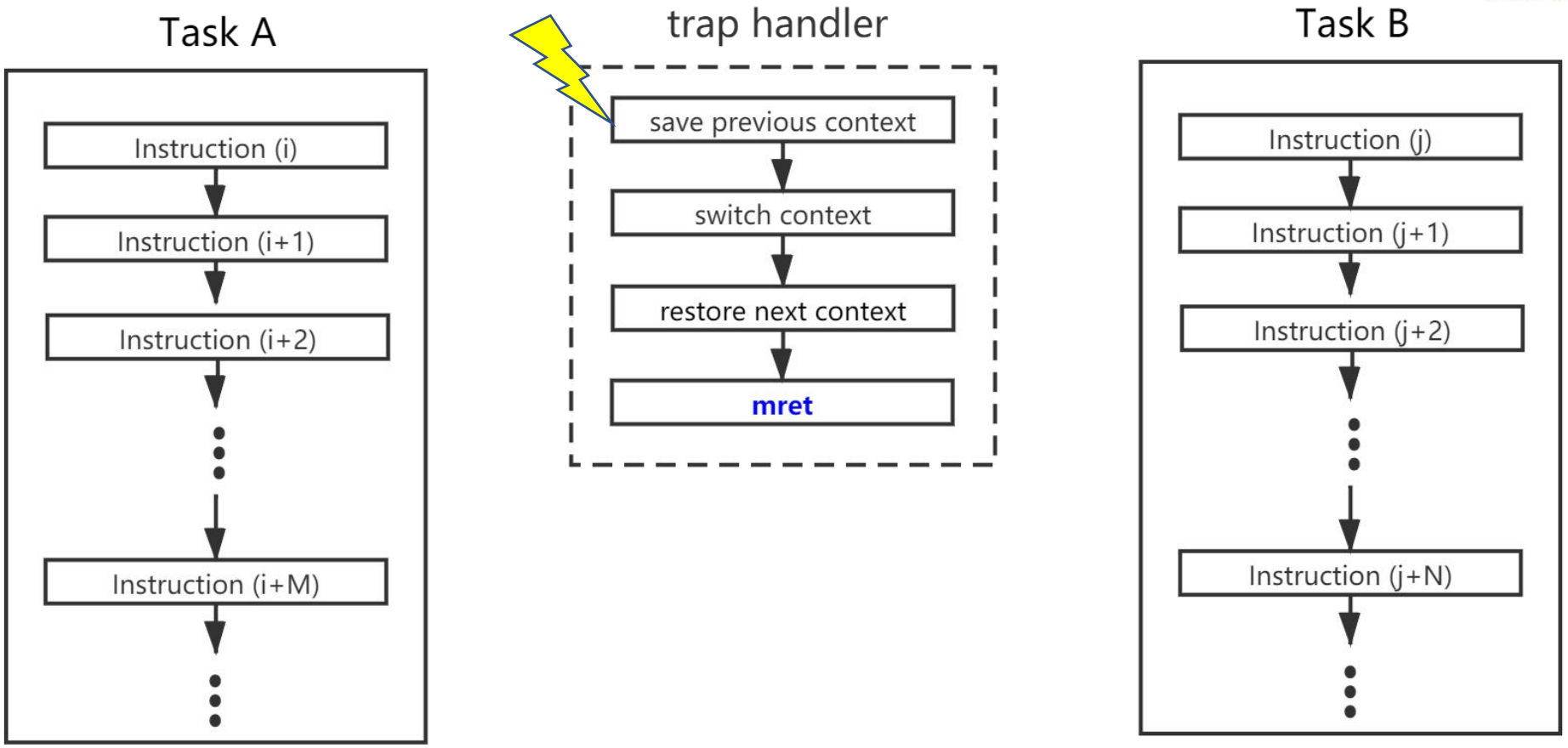
code/os/08-preemptive/sched.c

```
void schedule()
{
        ......
        switch_to(next);
}
```

code/os/08-preemptive/timer.c

```
void timer_handler()
{
        schedule();
}
```

抢占式多任务的设计


Task A
Instruction (i)
Instruction (i+1)
Instruction (i+2)
Instruction (i+M)
trap handler
save previous context
switch context
restore next context
mret
Task B
Instruction (j)
Instruction (j+1)
Instruction (j+2)
Instruction (j+N)

抢占式多任务的设计

code/os/08-preemptive/os.h

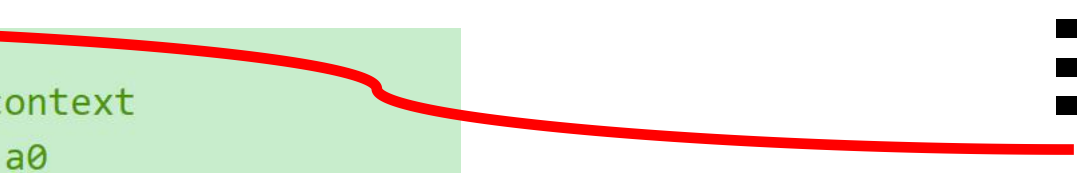
```
struct context {
        /* ignore x0 */
        reg_t ra;
        reg_t sp;
        ......
        reg_t t6;
        // upon is trap frame

        // save the pc to run in next schedule cycle
        reg_t pc; // offset: 31 *4 = 124
};
```

抢占式多任务的设计

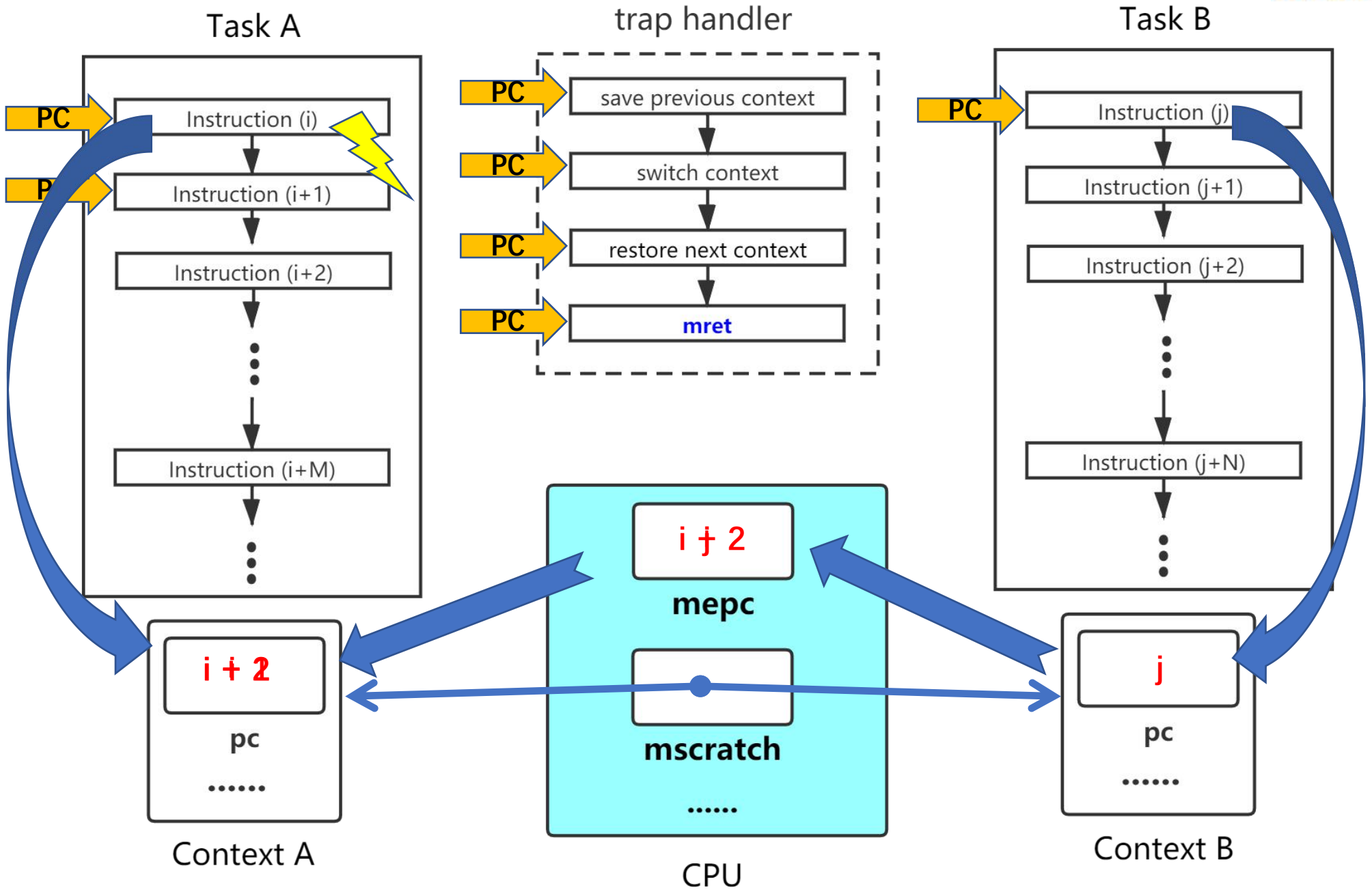
code/os/08-preemptive/entry.S

```
trap_vector:
        # Save all GP registers
        ......
        # save mepc to context of current task
        csrr    a0, mepc
        sw      a0, 124(t6)
        ......
        call    trap_handler
        ......
        mret
```

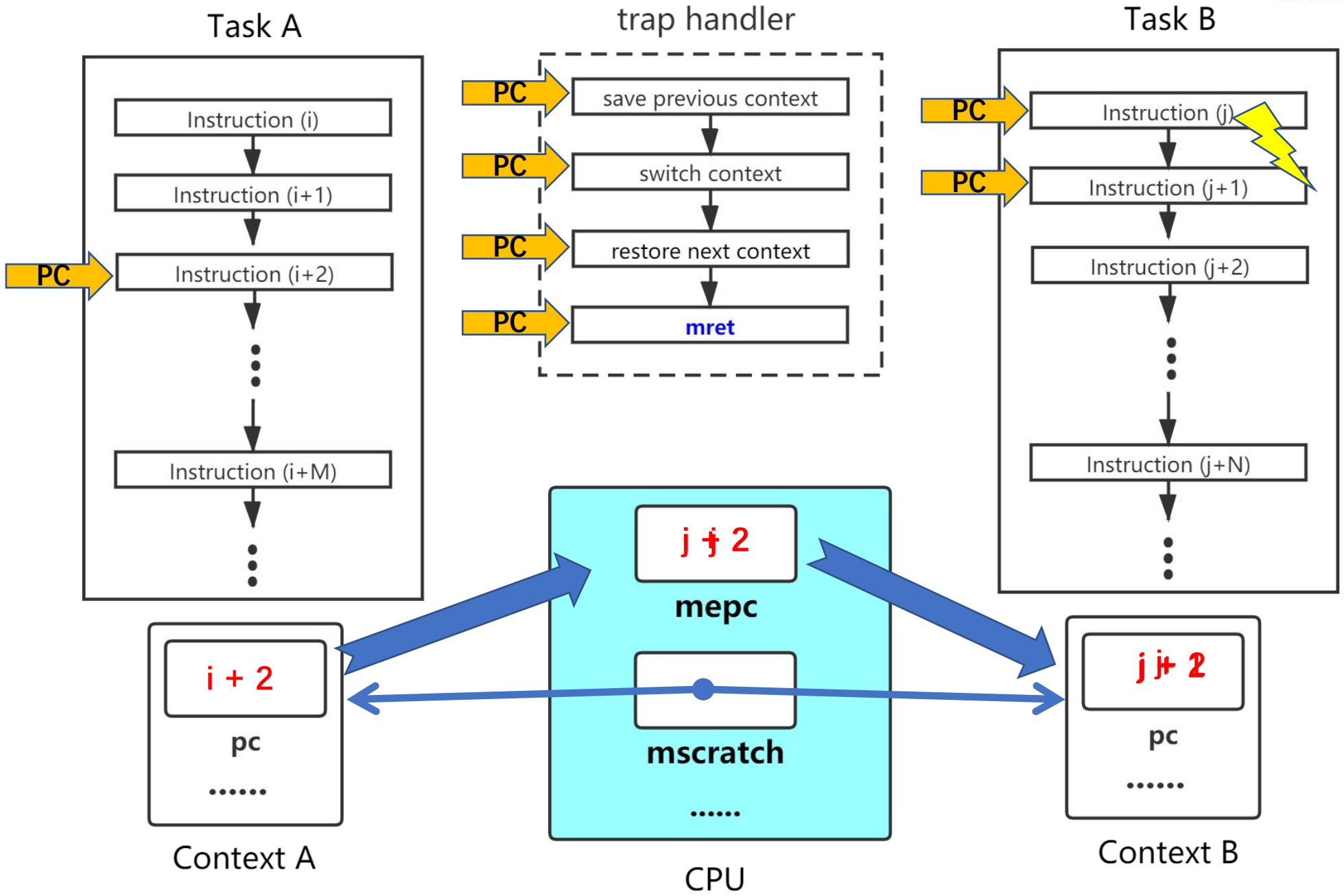


```
switch_to:
        # switch current context
        csrw    mscratch, a0
        # set mepc to the pc of the next task
        lw      a1, 124(a0)
        csrw    mepc, a1
        # Restore all GP registers
        ......
        mret
```

抢占式多任务的设计


Task A
trap handler
Task B
PC
Instruction (i)
Instruction (i+1)
Instruction (i+2)
Instruction (i+M)
save previous context
switch context
restore next context
mret
Instruction (j)
Instruction (j+1)
Instruction (j+2)
Instruction (j+N)
i + 2
mepc
mscratch
i + 2
pc
j
pc
Context A
CPU
Context B

抢占式多任务的设计


Task A
Instruction (i)
Instruction (i+1)
PC
Instruction (i+2)
Instruction (i+M)
trap handler
PC
save previous context
PC
switch context
PC
restore next context
PC
mret
Task B
PC
Instruction (j)
PC
Instruction (j+1)
Instruction (j+2)
Instruction (j+N)
j + 2
mepc
mscratch
.....
CPU
i + 2
pc
.....
Context A
j + 2
pc
.....
Context B

本章内容

- 抢占式多任务
- 抢占式多任务的设计
- 兼容协作式多任务

兼容 task_yield()

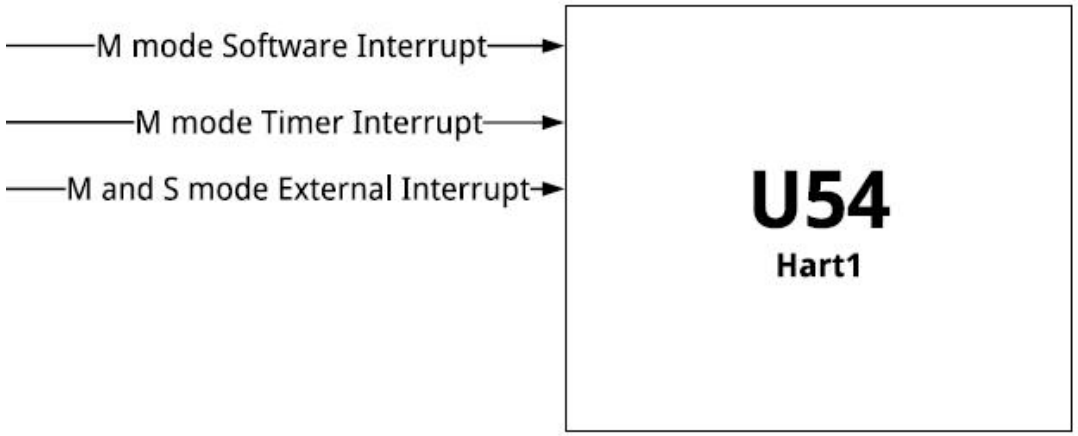
```
/*
 * DESCRIPTION
 *      task_yield()  causes the calling task to relinquish the CPU and a new
 *      task gets to run.
 */
void task_yield()
```

RISC-V 中断（Interrupt）的分类

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

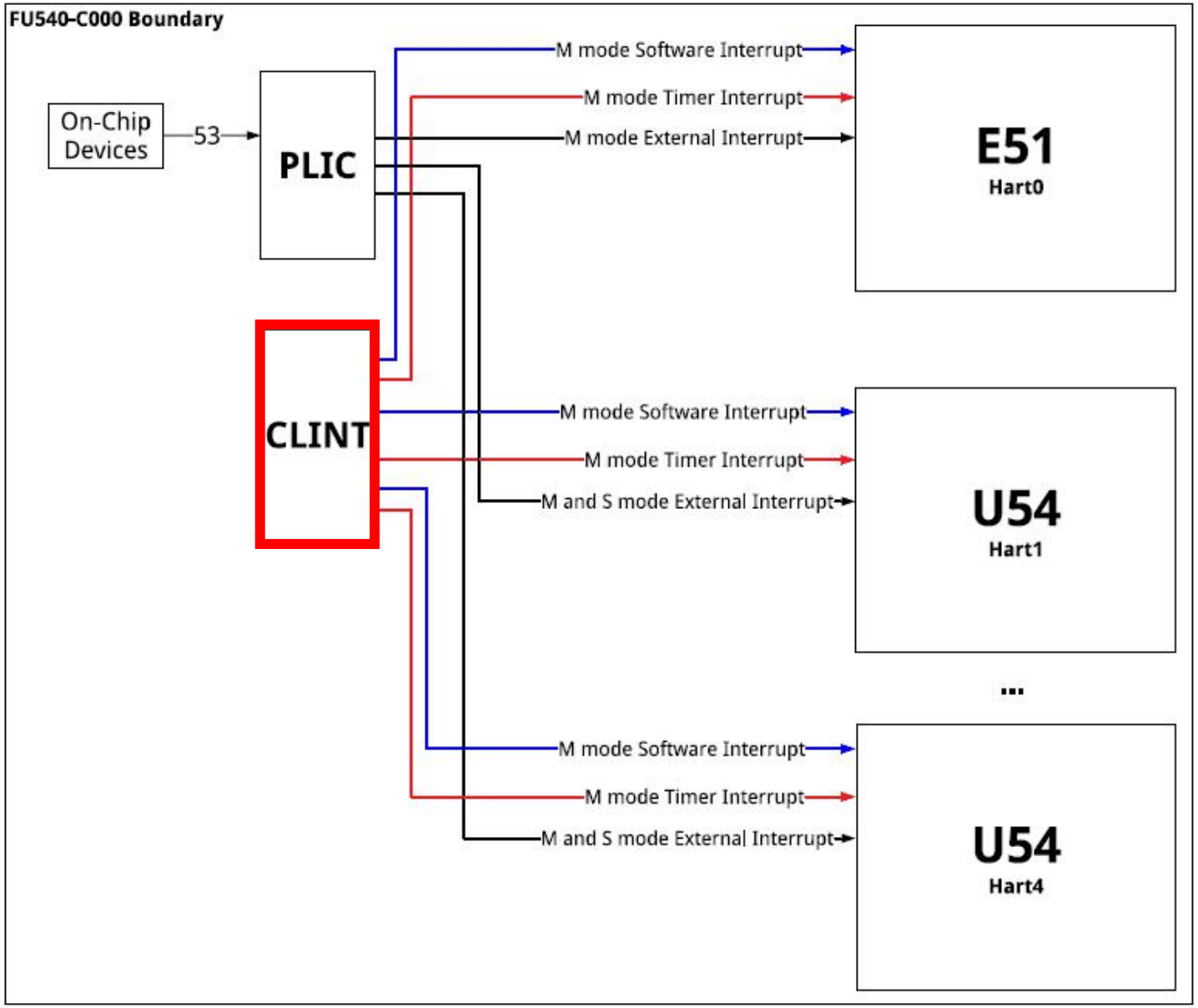
Interrupt	Exception Code	Description
1	0	User software interrupt
1	1	Supervisor software interrupt
1	2	*Reserved for future standard use*
1	3	Machine software interrupt
1	4	User timer interrupt
1	5	Supervisor timer interrupt
1	6	*Reserved for future standard use*
1	7	Machine timer interrupt
1	8	User 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.

Core Local INTerruptor



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

CLINT 编程接口 - 寄存器（software interrupt 部分）

可编程寄存器	功能描述	内存映射地址
MSIP	， 最低位和 CSR mip.MSIP 对应。	BASE + 4 * (hart)

- 每个 Hart 拥有一个 MSIP 寄存器
- RISCV 规范规定，Machine 模式下的 mip.MSIP 对应到一个 memory-mapped 的 控制寄存器。为此 QEMU-virt 提供 MSIP，该 MSIP 寄存器为 32-bit，高 31 位不可用，最低位映射到 mip.MSIP。
- 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 CLINT 的设计参考了 SFIVE，base 为 0x2000000。

```
#define CLINT_MSIP(hartid) (CLINT_BASE + 4 * (hartid))
```

CLINT 编程接口 - 寄存器（software interrupt 部分）

可编程寄存器	功能描述	内存映射地址
MSIP	， 最低位和 CSR mip.MSIP 对应。	BASE + 4 * (hart)

- **对 MSIP 写入 1 时触发 software interrupt，写入 0 表示对该中断进行应答。**

code/os/08-preemptive/sched.c

```
void task_yield()
{
        int id = r_mhartid();
        *(uint32_t*)CLINT_MSIP(id) = 1;
}
```

code/os/08-preemptive/entry.S

```
trap_vector:
        ......
        call    trap_handler
        ......
        mret

switch_to:
        ......
        mret
```

code/os/08-preemptive/trap.c

```
reg_t trap_handler(reg_t epc, reg_t cause)
{
        ......
        if (cause & 0x80000000) {
                switch (cause_code) {
                case 3:
                        int id = r_mhartid();
                        *(uint32_t*)CLINT_MSIP(id) = 0;
                        schedule();
                        break;
                ......
```

code/os/08-preemptive/sched.c

```
void schedule()
{
        ......
        switch_to(next);
}
```

练习 13-1

谢 谢

欢迎交流合作