从零开始的RISC-V模拟器开发

第11讲 QEMU篇之中断虚拟化1

中国科学院软件研究所
PLCT实验室

王俊强 wangjunqiang@iscas.ac.cn
李威威 liweiwei@iscas.ac.cn
吴伟 wuwei2016@iscas.ac.cn

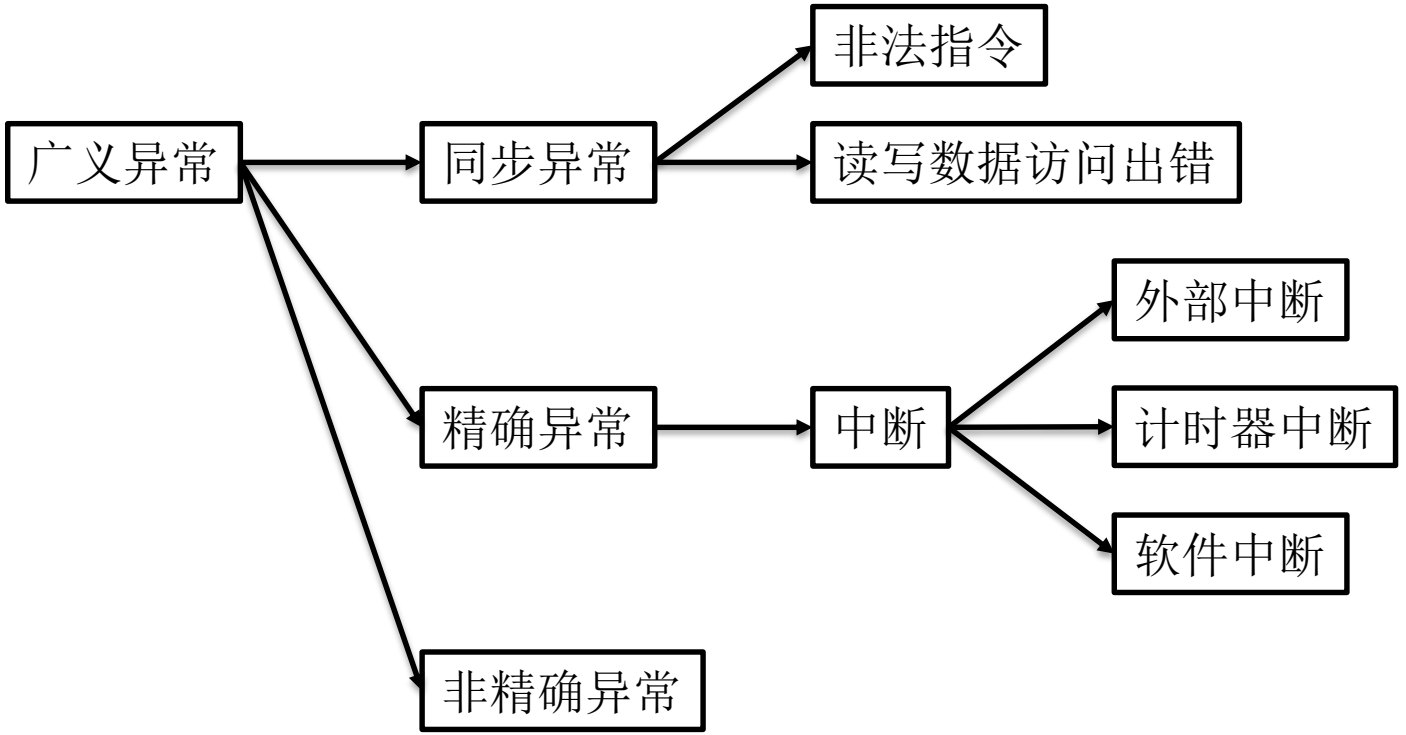
本课内容

中断虚拟化

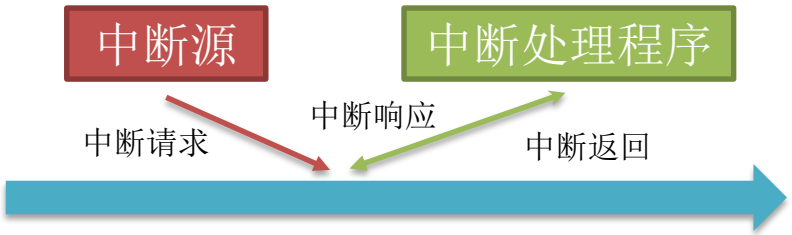
- QEMU中的IRQ
- ECLIC实现原理

RISC-V中断

- 外部中断External Interrupt
  - 来自核心外的中断，常见的GPIO、UART中断
- 定时器中断Timer Interrupt
  - 来自定时器的中断
- 软件中断Software Interrupt
  - 来自软件自己触发的中断
- 调试中断Debug Interrupt
  - 用于实现调试器的中断



来自:《RISC-V架构与嵌入式开发快速入门》



CPU运行过程中

QEMU中的IRQ之基础类型

irq类型定义:

```
typedef struct NucLeiUARTState
{
    ……
    /*< public >*/
    qemu_irq irq;
}
```

```
struct SIFIVEGPIOState {
    ……
    qemu_irq irq[SIFIVE_GPIO_PINS];
    qemu_irq output[SIFIVE_GPIO_PIN
S];
    ……
}
```

```
typedef struct NucLeiSYSTIMERState
{
    ……
    DeviceState *eclic;
    qemu_irq *timer_irq;
    qemu_irq *soft_irq;
    ……
}
```

include\qemu\typedefs.h

```
typedef struct IRQState *qemu_irq;
```

```
struct IRQState {
    Object parent_obj;

    qemu_irq_handler handler;
    void *opaque;
    int n;
};
```

n是irq num
opaque指向所属设备
qemu_irq_handler类型函数指针
handler回调函数

创建

```
qemu_irq qemu_allocate_irq(qemu_irq_ha
ndler handler, void *opaque, int n)
{
    struct IRQState *irq;

    irq = IRQ(object_new(TYPE_IRQ));
    irq->handler = handler;
    irq->opaque = opaque;
    irq->n = n;

    return irq;
}
```

基于QOM的TYPE_IRQ创建和索引

```
static const TypeInfo irq_type_info = {
    .name = TYPE_IRQ,
    .parent = TYPE_OBJECT,
    .instance_size = sizeof(struct IRQState),
};
```

```
typedef void (*qemu_irq_handler)(void *opaque,
                                 int n, int level);
```

触发

```
void qemu_set_irq(qemu_irq irq, int level)
{
    if (!irq)
        return;

    irq->handler(irq->opaque, irq->n, level);
}
```

接口:

- qemu_allocate_irq
- qemu_extend_irqs
- qemu_allocate_irqs
- qemu_free_irqs
- qemu_set_irq
- qemu_irq_raise
- qemu_irq_lower
- qemu_irq_pulse

QEMU中的IRQ之基础类型

Device与GPIO

```
struct DeviceState {
    ……
    QLIST_HEAD(, NamedGPIOList) gpios;
    ……
};
```

```
struct NamedGPIOList {
    char *name;
    qemu_irq *in;    all_in首地址
    int num_in;
    int num_out;
    QLIST_ENTRY(NamedGPIOList) node;
};
```

hw\core\qdev.c

创建与初始化

```
void qdev_init_gpio_in(DeviceState *dev, qemu_irq_handler handler, int n)
```

- qdev_init_gpio_in
  - qdev_init_gpio_in_named
    - qdev_init_gpio_in_named_with_opaque
      - qemu_extend_irqs
      - object_property_add_child

```
void qdev_init_gpio_out(DeviceState *dev, qemu_irq *pins, int n)
```

- qdev_init_gpio_out
  - qdev_init_gpio_out_named
    - object_property_add_link

获取：

- qdev_get_gpio_in
  - qdev_get_gpio_in_named

连接：

- qdev_connect_gpio_out_named
  - object_property_add_child
  - object_property_set_link

QEMU中的IRQ之处理

target\riscv\cpu.c

```
static struct TCGCPUOps riscv_tcg_ops = {
    ……
    .cpu_exec_interrupt = riscv_cpu_exec_interrupt,
    ……
    .do_interrupt = riscv_cpu_do_interrupt,
    ……
};
```

文件:accel/tcg/cpu-exec.c

```
int cpu_exec(CPUState *cpu)
{
 ……
    cc->tcg_ops->do_interrupt(cpu)
    while (!cpu_handle_exception(cpu, &ret)) {
        TranslationBlock *last_tb = NULL;
        int tb_e
        cc->tcg_ops->cpu_exec_interrupt()
        while (!cpu_handle_interrupt(cpu, &last_tb))
            uint32_t cflags = cpu->cflags_next_tb;
            TranslationBlock *tb;
            ……
        }
    }
    cc->cpu_exec_exit(cpu);
    return ret;
}
```

do_interrupt

exception处理：

```
static inline bool cpu_handle_exception(CPUState *cpu, int *ret)
{
    if (cpu->exception_index < 0) {
        ……
        return false;
    }
    if (cpu->exception_index >= EXCP_INTERRUPT) {
        ……
        if (*ret == EXCP_DEBUG) {
            cpu_handle_debug_exception(cpu);
        }
        ……
    } else {
        ……
        if (replay_exception()) {
            ……
            qemu_mutex_lock_iothread();
            cc->tcg_ops->do_interrupt(cpu);
            qemu_mutex_unlock_iothread();
            cpu->exception_index = -1;
            ……
        }
        ……
    }
    return false;
}
```

```
#define EXCP_INTERRUPT  0x10000
#define EXCP_HLT        0x10001
#define EXCP_DEBUG      0x10002
#define EXCP_HALTED     0x10003
#define EXCP_YIELD      0x10004
#define EXCP_ATOMIC     0x10005
```

QEMU中的IRQ之处理

target\riscv\cpu.c

```
static struct TCGCPUOps riscv_tcg_ops = {
    ……
    .cpu_exec_interrupt = riscv_cpu_exec_interrupt,
    ……
    .do_interrupt = riscv_cpu_do_interrupt,
    ……
};
```

文件:accel/tcg/cpu-exec.c

```
int cpu_exec(CPUState *cpu)
{
 ……
    cc->tcg_ops->do_interrupt(cpu)
    while (!cpu_handle_exception(cpu, &ret)) {
        TranslationBlock *last_tb = NULL;
        int tb_e
        cc->tcg_ops->cpu_exec_interrupt()
        while (!cpu_handle_interrupt(cpu, &last_tb))
            uint32_t cflags = cpu->cflags_next_tb;
            TranslationBlock *tb;
            ……
        }
    }
    cc->cpu_exec_exit(cpu);
    return ret;
}
```

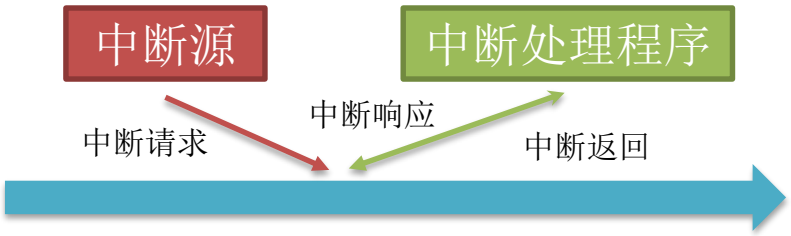
do_interrupt

```
static inline bool cpu_handle_interrupt(CPUState *cpu,
                                        TranslationBlock **last_tb)
{
    ……
    if (unlikely(qatomic_read(&cpu->interrupt_request))) {
        interrupt_request = cpu->interrupt_request;
        ……
        if (interrupt_request & CPU_INTERRUPT_DEBUG) {
            cpu->interrupt_request &= ~CPU_INTERRUPT_DEBUG;
            cpu->exception_index = EXCP_DEBUG;
            ……
        }
        if (interrupt_request & CPU_INTERRUPT_HALT) {
            cpu->interrupt_request &= ~CPU_INTERRUPT_HALT;
            cpu->halted = 1;
            cpu->exception_index = EXCP_HLT;
            ……
            return true;
        }
        else if (interrupt_request & CPU_INTERRUPT_RESET) {
            cpu_reset(cpu);
            ……
            return true;
        }
        else {
            if (cc->tcg_ops->cpu_exec_interrupt &&
            cc->tcg_ops->cpu_exec_interrupt(cpu, interrupt_request)) {
                ……
            }
        }
        if (interrupt_request & CPU_INTERRUPT_EXITTB) {
            ……
        }
    }
}
```

cpu->interrupt_request!

interrupt处理

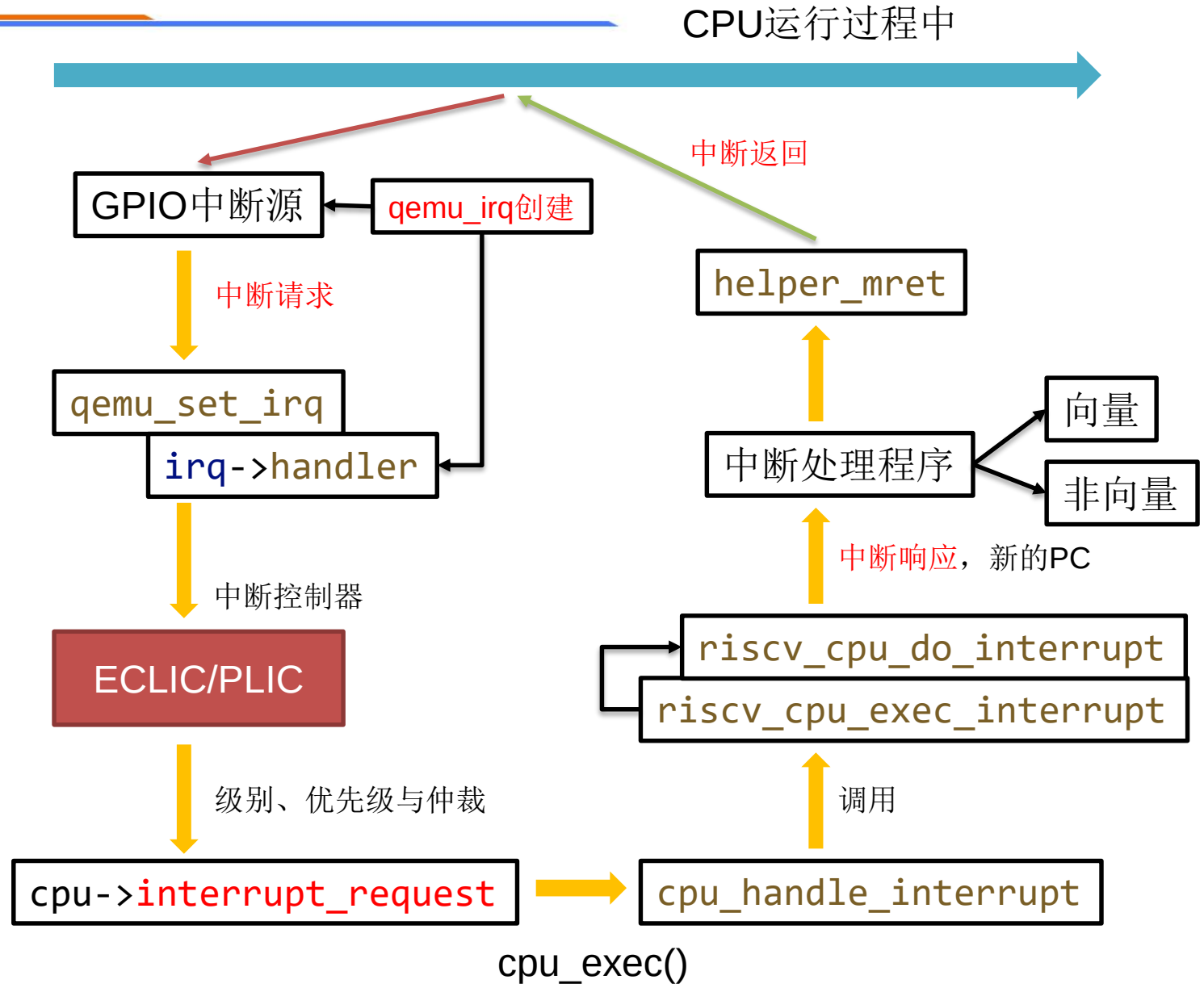
NUCLEI IRQ主要流程



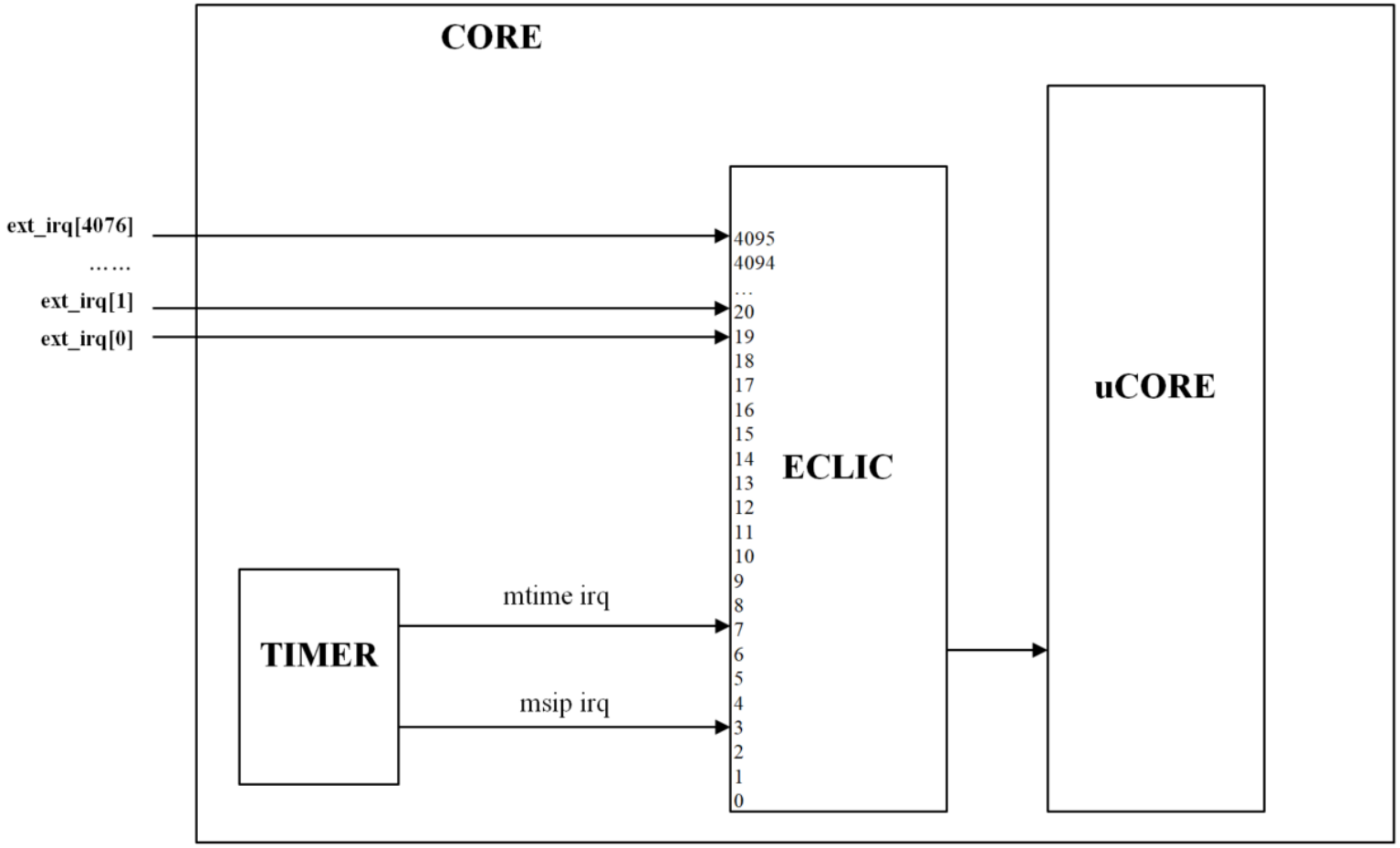
CPU运行过程中

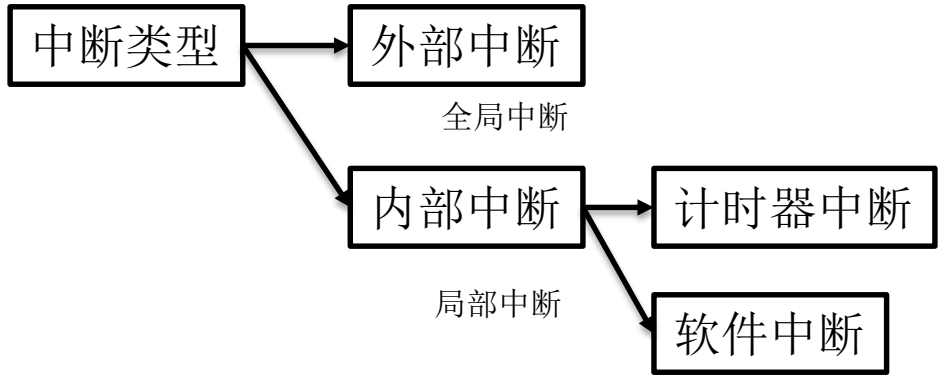
主要目标：

- qemu_irq创建
- ECLIC实现
- riscv_cpu_exec_interrupt更新
- riscv_cpu_do_interrupt更新
- 相应CSR的实现
- helper_mret更新



NUCLEI ECLIC(Enhanced Core Local Interrupt Controller)





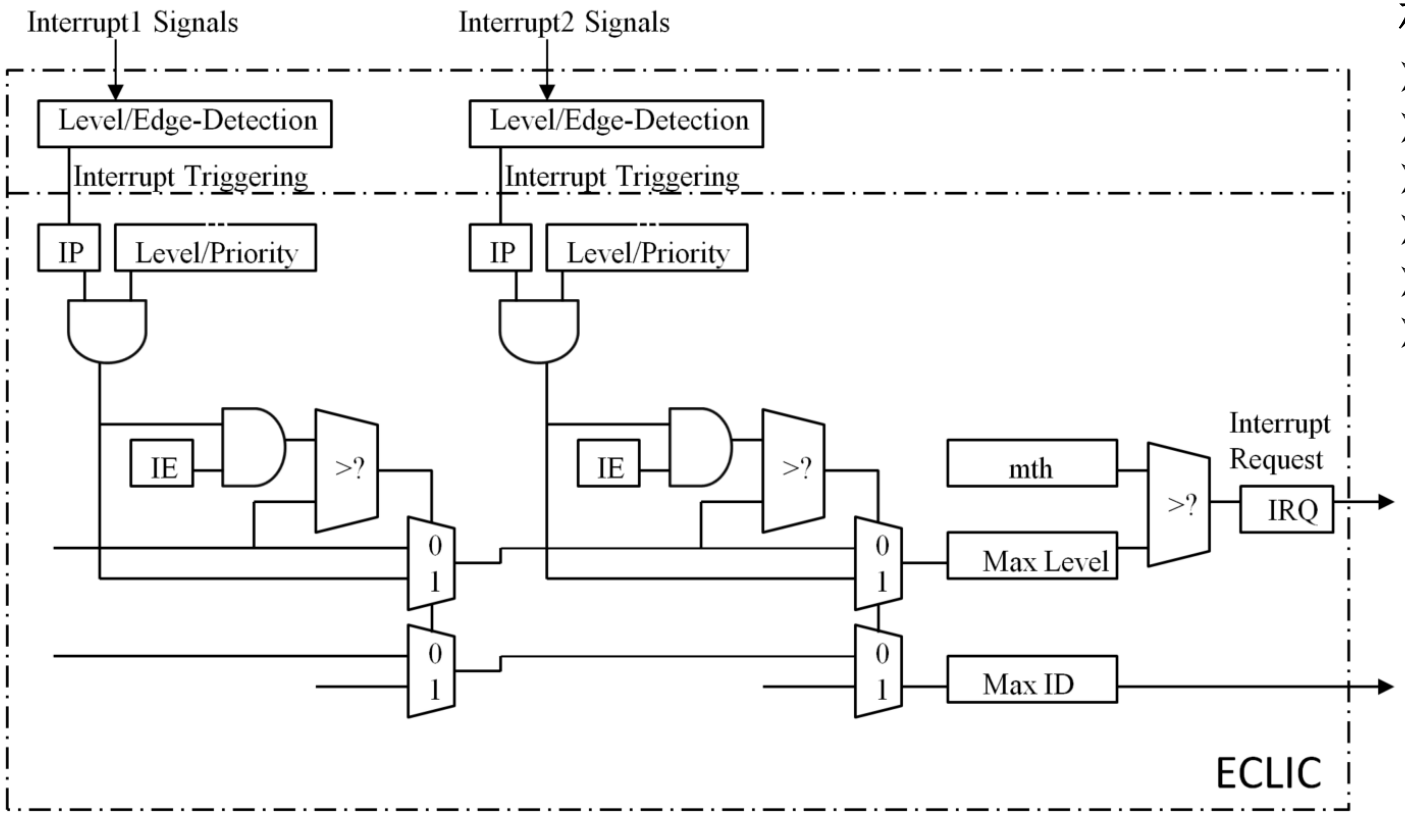
Machine mode

SYSTIMER+ ECLIC/CLIC for mcu
CLINT+PLIC for linux

- 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

CLIC: https://github.com/riscv/riscv-fast-interrupt

NUCLEI ECLIC(Enhanced Core Local Interrupt Controller)



相关概念：

- 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

阅读：Nuclei_N级别指令架构手册.pdf

QEMU中的IRQ之基础类型

ECLIC Registers:

Offset	Permission	Register	Width
0x0000	RW	cliccfg	8-bit
0x0004	R	clicinfo	32-bit
0x000b	RW	mth	8-bit
0x1000+4*i	RW	clicintip[i]	8-bit
0x1001+4*i	RW	clicintie[i]	8-bit
0x1002+4*i	RW	clicintattr[i]	8-bit
0x1003+4*i	RW	clicintctl[i]	8-bit

来自:Nuclei_N级别指令架构手册.pdf

include\hw\intc\nuclei_eclic.h

```
typedef struct NucLeiECLICState {
    /*< private >*/
    SysBusDevice parent_obj;
    /*< public >*/
    MemoryRegion mmio;
    uint32_t num_sources;
    /* config */
    uint32_t sources_id;
    uint8_t cliccfg;
    uint32_t clicinfo;
    uint8_t mth;
    uint8_t *clicintip;
    uint8_t *clicintie;
    uint8_t *clicintattr;
    uint8_t *clicintctl;
    ECLICPendingInterrupt *clicintlist;
    uint32_t aperture_size;

    QLIST_HEAD(, ECLICPendingInterrupt)
    pending_list;
    size_t active_count;

    /* ECLIC IRQ handlers */
    qemu_irq *irqs;
} NucLeiECLICState;
```

NUCLEI ECLIC之创建

eclic的静态注册:

```
static const TypeInfo nuclei_eclic_info = {
    .name = TYPE_NUCLEI_ECLIC,
    .parent = TYPE_SYS_BUS_DEVICE,
    .instance_size = sizeof(NucLeiECLICState),
    .class_init = nuclei_eclic_class_init,
};

static void nuclei_eclic_register_types(void)
{
    type_register_static(&nuclei_eclic_info);
}

type_init(nuclei_eclic_register_types);
```

mmio接口设置：

```
static const MemoryRegionOps nuclei_eclic_ops =
{
    .read = nuclei_eclic_read,
    .write = nuclei_eclic_write,
    .endianness = DEVICE_LITTLE_ENDIAN,
};
```

```
static void nuclei_eclic_realize(DeviceState *dev, Error **errp)
{
    NucLeiECLICState *eclic = NUCLEI_ECLIC(dev);
    int id;

    memory_region_init_io(&eclic->mmio, OBJECT(dev), &nuclei_eclic_ops, eclic,
                          TYPE_NUCLEI_ECLIC, eclic->aperture_size);
    sysbus_init_mmio(SYS_BUS_DEVICE(dev), &eclic->mmio);

    eclic->clicintip = g_new0(uint8_t, eclic->num_sources);
    ……
    QLIST_INIT(&eclic->pending_list);
    for (id = 0; id < eclic->num_sources; id++) {
        eclic->clicintlist[id].irq = id;
        update_eclic_int_info(eclic, id);
    }
    eclic->active_count = 0;

    /* Init ECLIC IRQ */
    eclic->irqs[Internal_SysTimerSW_IRQn] =
        qemu_allocate_irq(nuclei_eclic_irq_request,
                          eclic, Internal_SysTimerSW_IRQn);
    eclic->irqs[Internal_SysTimer_IRQn] =
        qemu_allocate_irq(nuclei_eclic_irq_request,
                          eclic, Internal_SysTimer_IRQn);

    for (id = Internal_Reserved_Max_IRQn; id < eclic->num_sources; id++) {
        eclic->irqs[id] = qemu_allocate_irq(nuclei_eclic_irq_request,
                                            eclic, id);
    }
    ……
}
```

NUCLEI ECLIC之读写

read and write

```
static uint64_t nuclei_eclic_read(void *opaque, hwaddr offset,
 unsigned size)
{
    NucLeiECLICState *eclic = NUCLEI_ECLIC(opaque);
    ……
    switch (offset) {
    case NUCLEI_ECLIC_REG_CLICCFG:
        value = eclic->cliccfg & 0xFF;
        break;
    case NUCLEI_ECLIC_REG_CLICINFO:
        value = (CLICINTCTLBITS << 21) & 0xFFFFFFFF;
        break;
    case NUCLEI_ECLIC_REG_MTH:
        value = eclic->mth & 0xFF;
        break;
    case NUCLEI_ECLIC_REG_CLICINTIP_BASE:
        value = eclic->clicintip[id] & 0xFF;
        break;
    case NUCLEI_ECLIC_REG_CLICINTIE_BASE:
        value = eclic->clicintie[id] & 0xFF;
        break;
      ……
    }
    return value;
}
```

```
static void nuclei_eclic_write(void *opaque, hwaddr offset, uint64_t value,
                               unsigned size)
{
    NucLeiECLICState *eclic = NUCLEI_ECLIC(opaque);
    uint32_t id = 0;
    ……
    switch (offset) {
    case NUCLEI_ECLIC_REG_CLICCFG:
        ……
    case NUCLEI_ECLIC_REG_MTH:
        nuclei_eclic_update_intmth(eclic, id, value & 0xFF);
        break;
    case NUCLEI_ECLIC_REG_CLICINTIP_BASE:
        if ((eclic->clicintlist[id].trigger & 0x1) != 0) {
            if ((eclic->clicintip[id] == 0) && (value & 0x1) == 1) {
                eclic->clicintip[id] = 1;
                eclic_insert_pending_list(eclic, id);
            } else if ((eclic->clicintip[id] == 1) && (value & 0x1) == 0) {
                eclic->clicintip[id] = 0;
                eclic_remove_pending_list(eclic, id);
            }
        }
        nuclei_eclic_next_interrupt(eclic);
        break;
    case NUCLEI_ECLIC_REG_CLICINTIE_BASE:
        nuclei_eclic_update_intie(eclic, id, value & 0xFF);
        break;
    ……
    default:
        break;
    }
```

NUCLEI ECLIC之核心设置函数

```
static void nuclei_eclic_next_interrupt(void *eclic_ptr)
{
    RISCVCPU *cpu = RISCV_CPU(qemu_get_cpu(0));
    CPURISCVState *env = &cpu->env;
    NucLeiECLICState *eclic = (NucLeiECLICState *)eclic_ptr;
    ECLICPendingInterrupt *active;
    target_ulong mil;
    int shv;

    QLIST_FOREACH(active, &eclic->pending_list, next)
    {
        if (active->enable) {
            mil = get_field(env->mintstatus, MINTSTATUS_MIL);
            if (active->level >= eclic->mth && active->level > mil) {
                shv = eclic->clicintattr[active->irq] & 0x1;
                eclic->active_count++;
                riscv_cpu_eclic_interrupt(cpu,
                    (active->irq & 0xFFF) | (shv << 12) | (active-
>level << 13));
                return;
            }
        }
    }
}
```

cpu->interrupt_request = CPU_INTERRUPT_ECLIC

```
static void riscv_cpu_eclic_interrupt(RISCVCPU *cpu,
                                      int exccode)
{
    CPURISCVState *env = &cpu->env;
    bool locked = false;

    env->exccode = exccode;

    if (!qemu_mutex_iothread_locked()) {
        locked = true;
        qemu_mutex_lock_iothread();
    }

    if (exccode != -1) {
        env->irq_pending = true;
        cpu_interrupt(CPU(cpu), CPU_INTERRUPT_ECLIC);
    } else {
        env->irq_pending = false;
        cpu_reset_interrupt(CPU(cpu), CPU_INTERRUPT_ECLIC);
    }

    if (locked) {
        qemu_mutex_unlock_iothread();
    }
}
```

ECLIC => CPU Request

NUCLEI ECLIC之riscv_cpu_exec_interrupt

```
bool riscv_cpu_exec_interrupt(CPUState *cs, int interrupt_request)
{
#if !defined(CONFIG_USER_ONLY)
    ……

    if (interrupt_request & CPU_INTERRUPT_ECLIC) {
        RISCVCPU *cpu = RISCV_CPU(cs);
        CPURISCVState *env = &cpu->env;
        int mode = PRV_M;
        int enabled = riscv_cpu_local_irq_mode_enabled(env, mode);
        if (enabled) {
            cs->exception_index = RISCV_EXCP_INT_ECLIC |  env->exccode;
            cs->interrupt_request &= ~CPU_INTERRUPT_ECLIC;
            riscv_cpu_do_interrupt(cs);
            return true;
        }
    }
#endif
    return false;
}
```

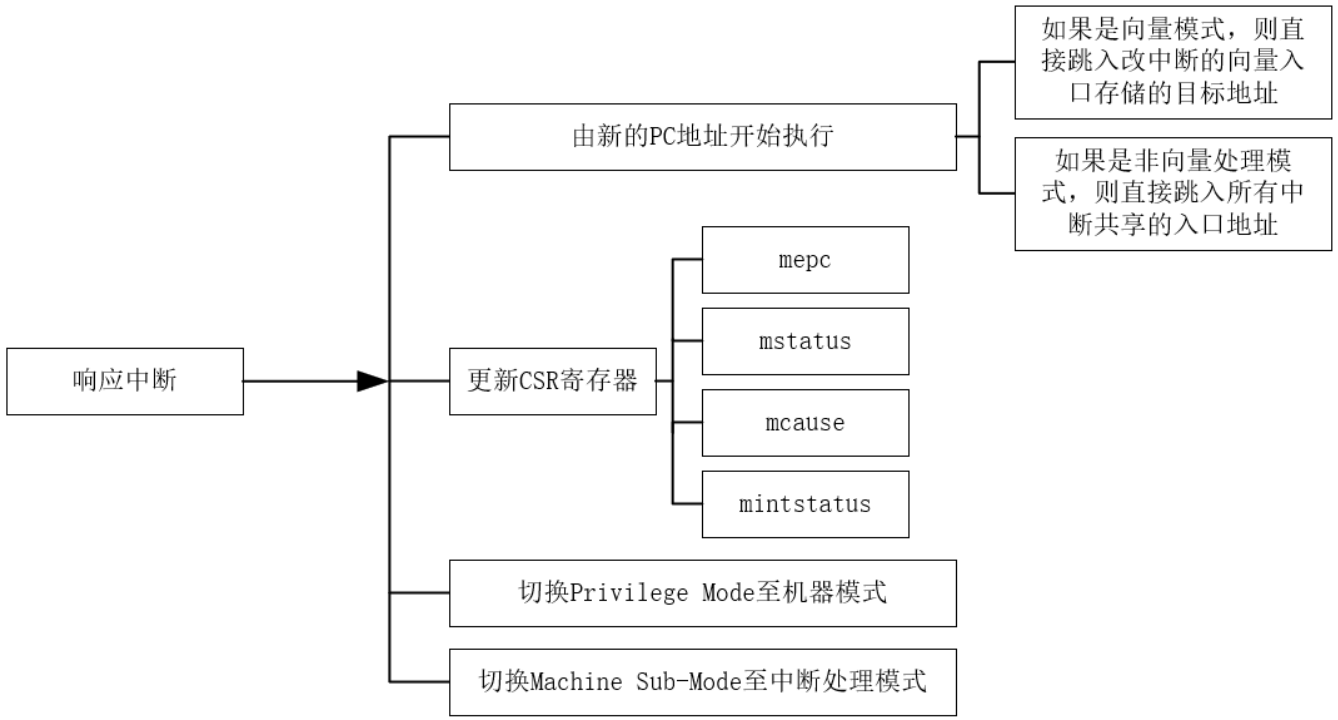
阅读:手册7.4.7

```
#define CPU_INTERRUPT_HARD        0x0002
#define CPU_INTERRUPT_EXITTB      0x0004
/* Halt the CPU.  */
#define CPU_INTERRUPT_HALT        0x0020
/* Debug event pending.  */
#define CPU_INTERRUPT_DEBUG       0x0080
/* Reset signal.  */
#define CPU_INTERRUPT_RESET       0x0400
/* Several target-
specific external hardware interrupts.  Each target/cpu.h
   should define proper names based on these defines.  */
#define CPU_INTERRUPT_TGT_EXT_0   0x0008
#define CPU_INTERRUPT_TGT_EXT_1   0x0010
#define CPU_INTERRUPT_TGT_EXT_2   0x0040
#define CPU_INTERRUPT_TGT_EXT_3   0x0200
#define CPU_INTERRUPT_TGT_EXT_4   0x1000
```

```
static int riscv_cpu_local_irq_mode_enabled(CPURISCVState *env,
 int mode)
{
    switch (mode) {
    case PRV_M:
            return env->priv < PRV_M ||
                        (env->priv == PRV_M &&
                        get_field(env->mstatus, MSTATUS_MIE));
    ……
    default:
        return false;
    }
}
```

MSTATUS—MIE

NUCLEI ECLIC之riscv_cpu_do_interrupt



阅读手册：5.6

mepc:7.4.20
mstatus:7.4.7
mcause: 7.4.21
mintstatus: 7.4.25

向量模式

5.13

非向量模式

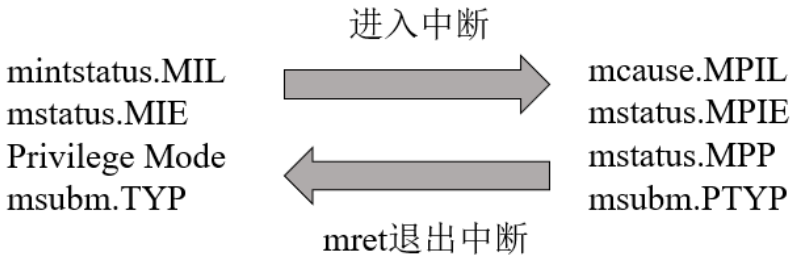
NUCLEI ECLIC之riscv_cpu_do_interrupt

更新寄存器：

```
if(eclic_flag)
    {
        mode = (cause >> 12) & 0x1;
        level = (cause >> 13) & 0xFF;
        cause &= 0x3ff;

        cause |= get_field(env->mstatus, MSTATUS_MPP) << 28;
        cause |= get_field(env->mintstatus, MINTSTATUS_MIL) << 16;
        cause |= get_field(env->mstatus, MSTATUS_MPIE) << 27;
        cause = set_field(cause, MCAUSE_MPP, PRV_M);
        cause = set_field(cause, MCAUSE_INTERRUPT, 1);

        env->mintstatus = set_field(env->mintstatus, MINTSTATUS_MIL, level);
    }
```



阅读手册：7.4.17
7.5.14

注意更新MTVEC实现

更新PC:

```
env->pc = riscv_intr_pc(env, async, eclic_flag, cause, mode);
```

```
if(eclic_flag ) {
            if(mode)
            {                                   向量模式下
                uint64_t vec_addr = (cause & 0x3FF) *4 + env->mtvt;
                cpu_physical_memory_rw(vec_addr, &newpc,  4, 0);
            }else{
                if ((env->mtvt2 & 0x1) == 0) {
                    newpc = env->mtvec & 0xfffffffc;
                } else if ((env->mtvt2 & 0x1) == 1) {
                    newpc = env->mtvt2 & 0xfffffffc;
                }
            }                                   非向量模式下
        }
```

NUCLEI ECLIC之中断处理程序

SoC\demosoc\Common\Source\GCC\startup_demosoc.S

```
/*
    * Intialize ECLIC vector interrupt
    * base address mtvt to vector_base
    */
   la t0, vector_base
   csrw CSR_MTVT, t0
   /*
    * Set ECLIC non-vector entry to be controlled
    * by mtvt2 CSR register.
    * Intialize ECLIC non-vector interrupt
    * base address mtvt2 to irq_entry.
    */
   la t0, irq_entry
   csrw CSR_MTVT2, t0
   csrs CSR_MTVT2, 0x1
   /*
    * Set Exception Entry MTVEC to exc_entry
    * Due to settings above, Exception and NMI
    * will share common entry.
    */
   la t0, exc_entry
   csrw CSR_MTVEC, t0
   /* Set the interrupt processing mode to ECLIC mode */
   li t0, 0x3f
   csrc CSR_MTVEC, t0
   csrs CSR_MTVEC, 0x3
```

中断向量表

中断共享入口表

向量模式

```
vector_base:
#if defined(DOWNLOAD_MODE) && (DOWNLOAD_MODE != DOWNLOAD_MODE_FLASH)
    j _start
    .align LOG_REGBYTES
#else
    DECLARE_INT_HANDLER     default_intexc_handler
#endif
    DECLARE_INT_HANDLER     default_intexc_handler
    DECLARE_INT_HANDLER     default_intexc_handler
    DECLARE_INT_HANDLER     eclic_msip_handler

    DECLARE_INT_HANDLER     default_intexc_handler
    DECLARE_INT_HANDLER     default_intexc_handler
    DECLARE_INT_HANDLER     default_intexc_handler
    DECLARE_INT_HANDLER     default_intexc_handler
    DECLARE_INT_HANDLER     eclic_mtip_handler
    ……
```

非向量模式

```
.global irq_entry
.weak irq_entry
irq_entry:
    SAVE_CONTEXT
    SAVE_CSR_CONTEXT
    /* This special CSR read/write operation, which is actually
     * claim the CLIC to find its pending highest ID, if the ID
     * is not 0, then automatically enable the mstatus.MIE, and
     * jump to its vector-entry-label, and update the link register
     */
    csrrw ra, CSR_JALMNXTI, ra
    DISABLE_MIE
    RESTORE_CSR_CONTEXT
    RESTORE_CONTEXT
    mret
```

NUCLEI ECLIC之中断处理程序

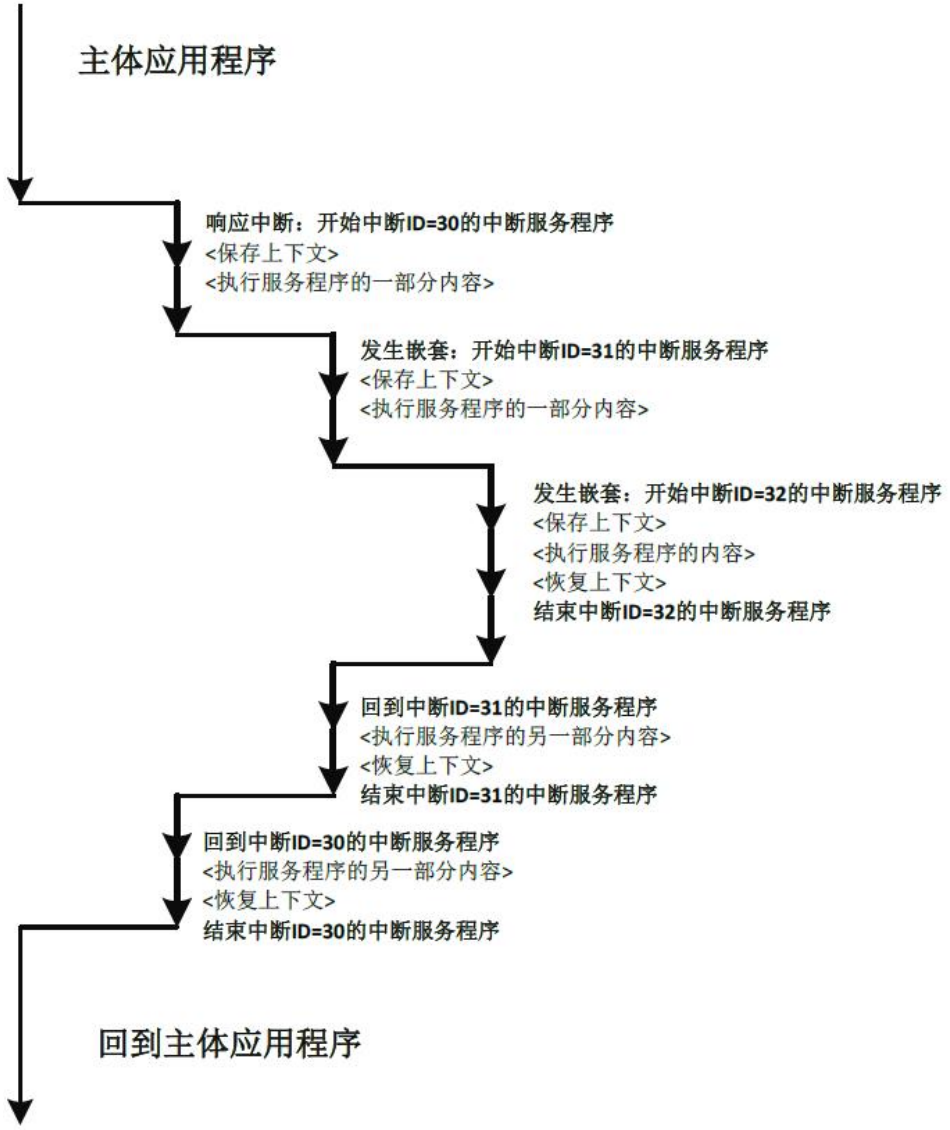
application\baremetal\demo_eclic\demo_eclic.c

```
__INTERRUPT void eclic_msip_handler(void)
{
    static uint32_t int_sw_cnt = 0;   /* software interrupt counter *
/
    // save CSR context
    SAVE_IRQ_CSR_CONTEXT();
    SysTimer_ClearSWIRQ();
    printf("[IN SOFTWARE INTERRUPT]software interrupt hit %d times\r\
n", int_sw_cnt++);
    printf("[IN SOFTWARE INTERRUPT]software interrupt end\r\n");
    // restore CSR context
    RESTORE_IRQ_CSR_CONTEXT();
}
```

```
#define SAVE_IRQ_CSR_CONTEXT()                                  \
        rv_csr_t __mcause = __RV_CSR_READ(CSR_MCAUSE);          \
        rv_csr_t __mepc = __RV_CSR_READ(CSR_MEPC);              \
        rv_csr_t __msubm = __RV_CSR_READ(CSR_MSUBM);            \
        __enable_irq();
```

向量模式下的嵌套

阅读手册:5.13.2



NUCLEI ECLIC之中断处理程序

SoC\demosoc\Common\Source\GCC\intexc_demosoc.S

```
/**
 * \brief  Macro for save necessary CSRs to stack
 * \details
 * This macro store MCAUSE, MEPC, MSUBM to stack.
 */
.macro SAVE_CSR_CONTEXT
    /* Store CSR mcause to stack using pushmcause */
    csrrwi  x0, CSR_PUSHMCAUSE, 11
    /* Store CSR mepc to stack using pushmepc */
    csrrwi  x0, CSR_PUSHMEPC, 12
    /* Store CSR msub to stack using pushmsub */
    csrrwi  x0, CSR_PUSHMSUBM, 13
.endm
```

CSR_JALMNXTI　　阅读手册：7.5.13~

非向量的嵌套和咬尾：5.13

application\baremetal\demo_eclic\demo_eclic.c

```
// If define SWIRQ_INTLEVEL_HIGHER equals 1 the software interrupt
will have a higher interrupt level.
// the software interrupt will run during timer interrupt.
// If define SWIRQ_INTLEVEL_HIGHER equals 0 the software interrupt
will have a lower interrupt level.
// the software interrupt will run after timer interrupt.
#define SWIRQ_INTLEVEL_HIGHER   0

// timer interrupt handler
// non-vector mode interrupt
void eclic_mtip_handler(void)
{
    static uint32_t int_t_cnt = 0;
    printf("--------------------\r\n");
    printf("[IN TIMER INTERRUPT]timer interrupt hit %d times\r\n", in
t_t_cnt++);

    printf("[IN TIMER INTERRUPT]trigger software interrupt\r\n");
#if SWIRQ_INTLEVEL_HIGHER == 1
    printf("[IN TIMER INTERRUPT]software interrupt will run during ti
mer interrupt\r\n");
#else
    printf("[IN TIMER INTERRUPT]software interrupt will run when time
r interrupt finished\r\n");
#endif
    // trigger software interrupt
    SysTimer_SetSWIRQ();
    // Reload Timer Interrupt
    SysTick_Reload(TIMER_TICKS);
    printf("[IN TIMER INTERRUPT]timer interrupt end\r\n");
}
```

NUCLEI ECLIC之中断处理程序

SoC\demosoc\Common\Source\GCC\intexc_demosoc.S

```
/**
 * \brief  Macro for save necessary CSRs to stack
 * \details
 * This macro store MCAUSE, MEPC, MSUBM to stack.
 */
.macro SAVE_CSR_CONTEXT
    /* Store CSR mcause to stack using pushmcause */
    csrrwi  x0, CSR_PUSHMCAUSE, 11
    /* Store CSR mepc to stack using pushmepc */
    csrrwi  x0, CSR_PUSHMEPC, 12
    /* Store CSR msub to stack using pushmsub */
    csrrwi  x0, CSR_PUSHMSUBM, 13
.endm
```

CSR_JALMNXTI

target\riscv\csr.c

```
static int rmw_pushmcause(CPURISCVState *env, int csrno, target_ulong *ret_value,
                target_ulong new_value, target_ulong write_mask)
{
    uint64_t notify_addr = new_value * 4 + env->gpr[2];
    cpu_physical_memory_rw(notify_addr, &env->mcause,  4, 1);
    return 0;
}

static int rmw_pushmepc(CPURISCVState *env, int csrno, target_ulong *ret_value,
                target_ulong new_value, target_ulong write_mask)
{
    uint64_t notify_addr = new_value * 4 + env->gpr[2];
    cpu_physical_memory_rw(notify_addr, &env->mepc, 4, 1);
    return 0;
}
```

```
static int rmw_jalmnxti(CPURISCVState *env, int csrno, target_ulong *ret_value,
                target_ulong new_value, target_ulong write_mask)
{
    target_ulong addr;

    if (env->irq_pending) {
        uint64_t vec_addr = (env->mcause & 0x3FF) * 4 + env->mtvt;
        cpu_physical_memory_rw(vec_addr, &addr, 4, 0);
        env->gpr[1] = env->pc + 4;
        env->gpr[5] = env->pc + 4;
        *ret_value = addr;
        riscv_cpu_eclic_clean_pending(env->eclic, env->mcause & 0x3ff);
        env->mstatus = set_field(env->mstatus, MSTATUS_MIE, 1);
    } else
        *ret_value = env->pc + 4;
    return 0;
}
```

NUCLEI ECLIC之中断处理程序

SoC\demosoc\Common\Source\GCC\intexc_demosoc.S

```
/**
 * \brief  Macro for save necessary CSRs to stack
 * \details
 * This macro store MCAUSE, MEPC, MSUBM to stack.
 */
.macro SAVE_CSR_CONTEXT
    /* Store CSR mcause to stack using pushmcause */
    csrrwi  x0, CSR_PUSHMCAUSE, 11
    /* Store CSR mepc to stack using pushmepc */
    csrrwi  x0, CSR_PUSHMEPC, 12
    /* Store CSR msub to stack using pushmsub */
    csrrwi  x0, CSR_PUSHMSUBM, 13
.endm
```

```
static bool trans_csrrw(DisasContext *ctx, arg_csrrw *a)
{
    TCGv source1, csr_store, dest, rs1_pass;
    RISCV_OP_CSR_PRE;
    gen_helper_csrrw(dest, cpu_env, source1, csr_store);
    if(a->csr == CSR_JALMNXTI)
        RISCV_OP_CSR_JAL_POST;
    else
        RISCV_OP_CSR_POST;
    return true;
}
```

target\riscv\csr.c

```
static int rmw_jalmnxti(CPURISCVState *env, int csrno, target_ulong *ret_value,
                target_ulong new_value, target_ulong write_mask)
{
    target_ulong addr;

    if (env->irq_pending) {
        uint64_t vec_addr = (env->mcause & 0x3FF) * 4 + env->mtvt;
        cpu_physical_memory_rw(vec_addr, &addr, 4, 0);
        env->gpr[1] = env->pc + 4;
        env->gpr[5] = env->pc + 4;
        *ret_value = addr;
        riscv_cpu_eclic_clean_pending(env->eclic, env->mcause & 0x3ff);
        env->mstatus = set_field(env->mstatus, MSTATUS_MIE, 1);
    } else
        *ret_value = env->pc + 4;
    return 0;
}
```

```
#define RISCV_OP_CSR_POST do {\
    gen_set_gpr(a->rd, dest); \
    tcg_gen_movi_tl(cpu_pc, ctx->pc_succ_insn); \
    exit_tb(ctx); \
    ctx->base.is_jmp = DISAS_NORETURN; \
    tcg_temp_free(source1); \
    tcg_temp_free(csr_store); \
    tcg_temp_free(dest); \
    tcg_temp_free(rs1_pass); \
} while (0)
```

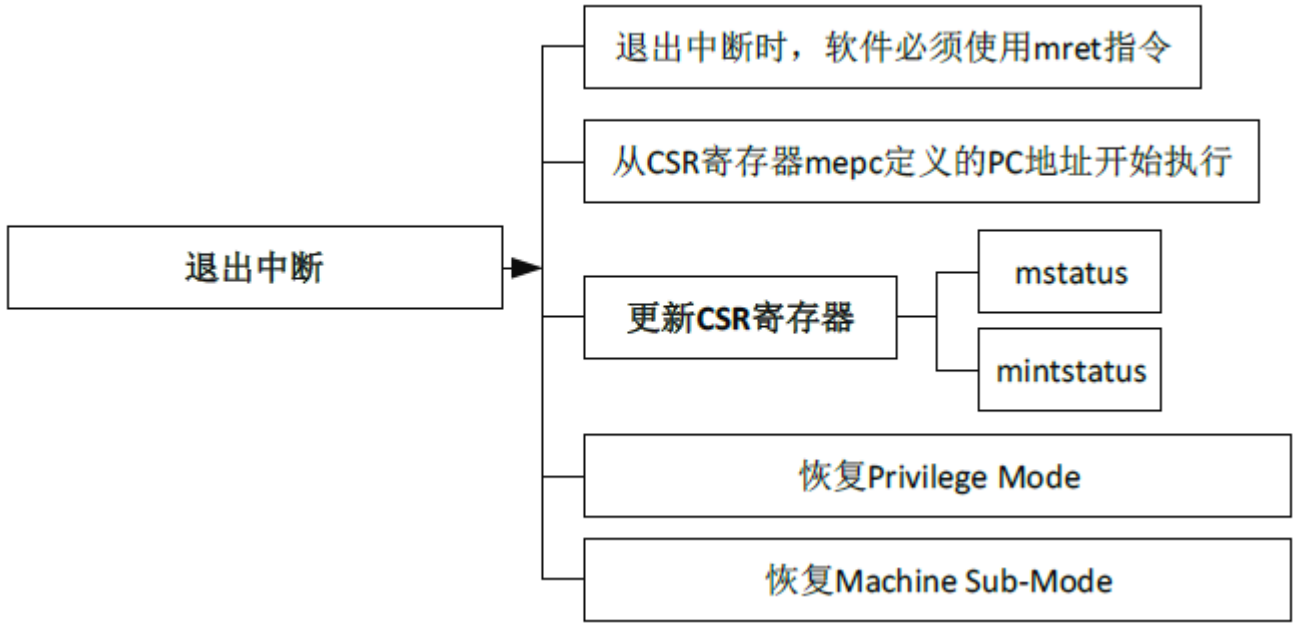
target\riscv\insn_trans\trans_rvi.c.inc

咬尾自跳转

CSR_JALMNXTI

```
#define RISCV_OP_CSR_JAL_POST do {\
    tcg_gen_mov_tl(cpu_pc, dest); \
    exit_tb(ctx); \
    ctx->base.is_jmp = DISAS_NORETURN; \
    tcg_temp_free(source1); \
    tcg_temp_free(csr_store); \
    tcg_temp_free(dest); \
    tcg_temp_free(rs1_pass); \
} while (0)
```

NUCLEI ECLIC之mret



参考手册：5.7

target\riscv\op_helper.c

```
target_ulong helper_mret(CPURISCVState *env,
                         target_ulong cpu_pc_deb)
{
    ……
    target_ulong retpc = env->mepc;
    ……
    target_ulong prev_priv = get_field(mstatus, MSTATUS_MPP);
    mstatus = set_field(mstatus, MSTATUS_MIE,
                        get_field(mstatus, MSTATUS_MPIE));
    mstatus = set_field(mstatus, MSTATUS_MPIE, 1);
    mstatus = set_field(mstatus, MSTATUS_MPP, PRV_U);
    mstatus = set_field(mstatus, MSTATUS_MPV, 0);
    env->mstatus = mstatus;
    riscv_cpu_set_mode(env, prev_priv);
    ……
    if ((env->mtvec & 0b111111) == 0b000011) {
        target_ulong mpil = get_field(env->mcause, MCAUSE_MPIL);
        env->mintstatus = set_field(env-
>mintstatus, MINTSTATUS_MIL, mpil);

        qemu_mutex_lock_iothread();
        riscv_cpu_eclic_get_next_interrupt(env->eclic);
        qemu_mutex_unlock_iothread();

        if (get_field(env->mcause, MCAUSE_INTERRUPT) == 1) {
            env->mstatus = set_field(env->mstatus, MSTATUS_MPP,
                        get_field(env->mcause, MCAUSE_MPP));
        }
    }
    return retpc;
}
```

NUCLEI ECLIC之使用

```
static void nuclei_mtimecmp_cb(void *opaque) {
    RISCVCPU *cpu = RISCV_CPU(qemu_get_cpu(0));
    CPURISCVState *env = &cpu->env;
    nuclei_eclic_systimer_cb(((RISCVCPU *)cpu)->env.eclic);
    timer_del(env->timer);
}
```

hw\intc\nuclei_systimer.c

nuclei_n_soc_realize函数中

hw\riscv\nuclei_n.c

```
 s->eclic = nuclei_eclic_create(memmap[NUCLEI_N_ECLIC].base,
                                memmap[NUCLEI_N_ECLIC].size, 51);
    if( s->eclic != NULL)
    {
        s->timer.eclic = s->eclic;
        s->timer.soft_irq =&(NUCLEI_ECLIC(s->eclic)->irqs[Internal_SysTimerSW_IRQn]);
        s->timer.timer_irq = &(NUCLEI_ECLIC(s->eclic)->irqs[Internal_SysTimer_IRQn]);

        s->uart0.irq = nuclei_eclic_get_irq(DEVICE(s->eclic),22);
    }
```

SYSTIMER中

```
env->timer = timer_new_ns(QEMU_CLOCK_VIRTUAL,
                          &nuclei_mtimecmp_cb, cpu);
```

timecmp对比：将来触发

```
uint64_t next_ns = qemu_clock_get_ns(QEMU_CLOCK_
VIRTUAL)+muldiv64(diff, NANOSECONDS_PER_SECOND,
s->timebase_freq);
timer_mod(env->timer, next_ns);
```

hw\intc\nuclei_eclic.c

```
DeviceState *nuclei_eclic_create(hwaddr addr,
                                 uint32_t aperture_size, uint32_t num_sources)
{
    DeviceState *dev = qdev_new(TYPE_NUCLEI_ECLIC);

    qdev_prop_set_uint32(dev, "aperture-size", aperture_size);
    qdev_prop_set_uint32(dev, "num-sources", num_sources);

    sysbus_realize_and_unref(SYS_BUS_DEVICE(dev), &error_fatal);
    sysbus_mmio_map(SYS_BUS_DEVICE(dev), 0, addr);
    return dev;
}
```

软件中断：

```
case NUCLEI_SYSTIMER_REG_MSIP:
    s->msip = value;
    if ((s->msip & 0x1) == 1) {
        qemu_set_irq(*(s->soft_irq), 1);
    }else{
        qemu_set_irq(*(s->soft_irq), 0);
    }
```

NUCLEI ECLIC之测试

```
$qemu-system-riscv32  \
-nographic -machine mcu_200t \
-kernel ./hbird/demo_eclic.elf \
-nodefaults -serial stdio
```

```
$qemu-system-riscv32  \
-nographic -machine mcu_200t \
-kernel ./hbird/demo_eclic2.elf \
-nodefaults -serial stdio
```

```
Nuclei SDK Build Time: Apr  9 2021, 09:56:38
Download Mode: ILM
CPU Frequency 1182112 Hz
Initialize timer and start timer interrupt periodly

[IN TIMER INTERRUPT]timer interrupt hit 0 times
[IN TIMER INTERRUPT]trigger software interrupt
[IN TIMER INTERRUPT]software interrupt will run when timer interrupt finished
[IN TIMER INTERRUPT]timer interrupt end
[IN SOFTWARE INTERRUPT]software interrupt hit 0 times
[IN SOFTWARE INTERRUPT]software interrupt end
------------------
[IN TIMER INTERRUPT]timer interrupt hit 1 times
[IN TIMER INTERRUPT]trigger software interrupt
[IN TIMER INTERRUPT]software interrupt will run when timer interrupt finished
[IN TIMER INTERRUPT]timer interrupt end
[IN SOFTWARE INTERRUPT]software interrupt hit 1 times
[IN SOFTWARE INTERRUPT]software interrupt end
------------------
[IN TIMER INTERRUPT]timer interrupt hit 2 times
[IN TIMER INTERRUPT]trigger software i
[IN TIMER INTERRUPT]software interrupt
[IN TIMER INTERRUPT]timer interrupt en
[IN SOFTWARE INTERRUPT]software interr
[IN SOFTWARE INTERRUPT]software interr
------------------
```

```
------------------
[IN TIMER INTERRUPT]timer interrupt hit 14 times
[IN TIMER INTERRUPT]trigger software interrupt
[IN TIMER INTERRUPT]software interrupt will run during timer interrupt
[IN SOFTWARE INTERRUPT]software interrupt hit 14 times
[IN SOFTWARE INTERRUPT]software interrupt end
[IN TIMER INTERRUPT]timer interrupt end
------------------
[IN TIMER INTERRUPT]timer interrupt hit 15 times
[IN TIMER INTERRUPT]trigger software interrupt
[IN TIMER INTERRUPT]software interrupt will run during timer interrupt
[IN SOFTWARE INTERRUPT]software interrupt hit 15 times
[IN SOFTWARE INTERRUPT]software interrupt end
[IN TIMER INTERRUPT]timer interrupt end
------------------
[IN TIMER INTERRUPT]timer interrupt hit 16 times
[IN TIMER INTERRUPT]trigger software interrupt
[IN TIMER INTERRUPT]software interrupt will run during timer interrupt
[IN SOFTWARE INTERRUPT]software interrupt hit 16 times
[IN SOFTWARE INTERRUPT]software interrupt end
[IN TIMER INTERRUPT]timer interrupt end
------------------
```

下节课内容：

中断虚拟化

- ECLIC 与 CLIC
- SYSTIMER 与 CLINT
- PLIC介绍
- ECLIC在UART中的使用

谢谢

wangjunqiang@iscas.ac.cn