



易 灵 思

RISC-V 培训

进阶篇

CSR寄存器

- Control and Status Registers

- 特权级配置

- M-Mode

- 最高权限，没有任何限制

- S-Mode

- 特权级，通常和用户级同时使用，用于实现现代操作系统权限管理

- U-Mode

- 用户级，大部分应用程序运行在次特权，CSR寄存器访问受限

- PMP

- 内存存取权限配置

- Mtvec/Mtval

- 异常/错误抓取配置

mcause寄存器 - 中断

- RISC-V的中断类型：
 - 外部中断
 - 最常用的外部输入中断
 - 计时中断
 - 通常用于分时操作
 - 软中断
 - 通常用于调度操作
- RISC-V中断处理流程
 - 配置mtvec寄存器，指向trap_entry函数
 - 配置mie寄存器启用位MIE_MEIE
 - 用户逻辑->Plic->mcause寄存器

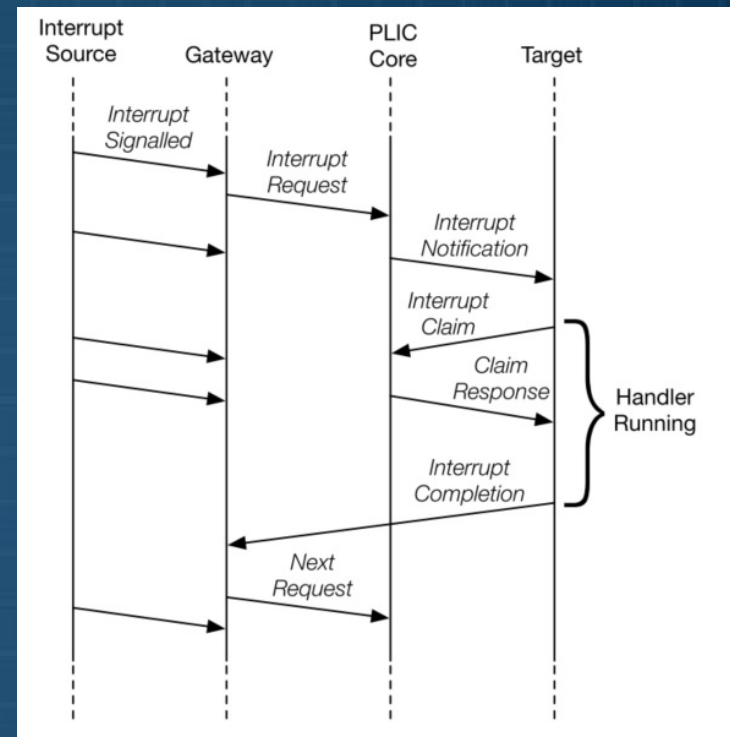
Interrupt	Exception Code	Description
1	0	<i>Reserved</i>
1	1	Supervisor software interrupt
1	2	<i>Reserved</i>
1	3	Machine software interrupt
1	4	<i>Reserved</i>
1	5	Supervisor timer interrupt
1	6	<i>Reserved</i>
1	7	Machine timer interrupt
1	8	<i>Reserved</i>
1	9	Supervisor external interrupt
1	10	<i>Reserved</i>
1	11	Machine external interrupt
1	12-15	<i>Reserved</i>
1	≥16	<i>Designated for platform use</i>

PLIC

- PLIC配置

- 配置mtval指向异常处理入口 (trap)
- 配置mie寄存器, 启用异常抓取
- 配置mstatus掩码, 抓取mie寄存器状态

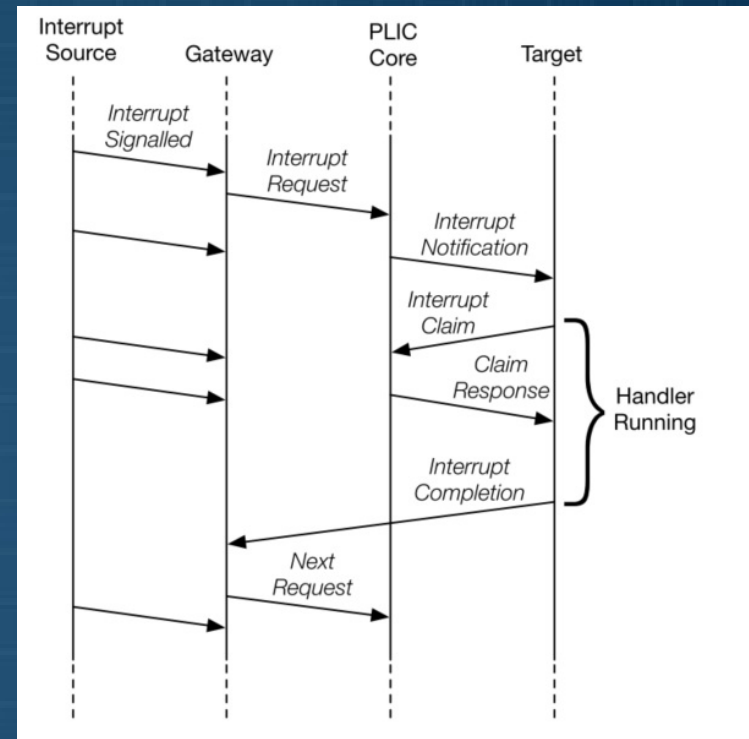
```
csr_write(mtvec, trap_entry);  
csr_set(mie, MIE_MEIE);  
csr_write(mstatus, MSTATUS_MPP | MSTATUS_MIE);
```



PLIC

- Handler = trap函数
 - 获取claim
 - 判断claim
 - 释放claim

```
void UartInterrupt()  
{  
    uint32_t claim;  
    //While there is pending interrupts  
    while(claim = plic_claim(BSP_PLIC, BSP_PLIC_CPU_0)){  
        switch(claim){  
            case SYSTEM_PLIC_SYSTEM_UART_0_IO_INTERRUPT: UartInterrupt_Sub(); break;  
            default: crash(); break;  
        }  
        //unmask the claimed interrupt  
        plic_release(BSP_PLIC, BSP_PLIC_CPU_0, claim);  
    }  
}
```



mstatus寄存器 - 异常

- 异常分类
 - 指令相关
 - 指令未对齐
 - 指令异常
 - 指令权限
 - 内存相关
 - 地址未对齐
 - 越界/权限不足
 - 内存页面相关 (MMU+虚拟内存)

0	0	Instruction address misaligned
0	1	Instruction access fault
0	2	Illegal instruction
0	3	Breakpoint
0	4	Load address misaligned
0	5	Load access fault
0	6	Store/AMO address misaligned
0	7	Store/AMO access fault
0	8	Environment call from U-mode
0	9	Environment call from S-mode
0	10	<i>Reserved</i>
0	11	Environment call from M-mode
0	12	Instruction page fault
0	13	Load page fault
0	14	<i>Reserved</i>
0	15	Store/AMO page fault
0	16–23	<i>Reserved</i>
0	24–31	<i>Designated for custom use</i>
0	32–47	<i>Reserved</i>
0	48–63	<i>Designated for custom use</i>
0	≥64	<i>Reserved</i>

示例

- mcause 值为2, 显示异常
- mepc 异常指令地址
- mtval 错误地址/指令

The screenshot shows a debugger window with assembly code on the left and CSR registers on the right. A red arrow points from the assembly instruction at address 0x1a94 to the mepc register in the CSR window.

Assembly Code:

```
crash("interrupt");
1a80: 00003537      lui a0,0x3
1a84: c2850513      addi a0,a0,-984 #
1a88: fclff0ef      jal ra,1a48 <crash>

00001a8c <wrong_instruction>:
}

void masm();
void wrong_instruction() {
1a8c: ff010113      addi sp,sp,-16
1a90: 00112623      sw ra,12(sp)
1a94: 501f 0000 00ef ← 0xef0000501f
asm(".word(0x501F)");
masm();
1a9a: 0700          addi s0,sp,896
}

1a9c: 00c12083      lw ra,12(sp)
1aa0: 01010113      addi sp,sp,16
1aa4: 00008067      ret

00001aa8 <main>:

int main(int argc, char **argv) {
1aa8: ff010113      addi sp,sp,-16
1aac: 00112623      sw ra,12(sp)
icache_clear();
1ab0: eblff0ef      jal ra,1960 <icache_c
csr_write(mtvec, trap_entry);
1ab4: 000027b7      lui a5,0x2
1ab8: b1478793      addi a5,a5,-1260 #
```

CSR Registers:

Name	Value
> General Purpose Registers	
> Floating Point Registers	
▼ CSR	
1010 0101 mhartid	0
> 1010 0101 mstatus	2147514496
> 1010 0101 mie	2048
> 1010 0101 mtvec	6932
1010 0101 mscratch	0
1010 0101 mepc	0x1a94 <wrong_in
> 1010 0101 mcause	2
1010 0101 mtval	20511
> 1010 0101 mip	128
> 1010 0101 sstatus	2147508224
> 1010 0101 fflags	128
> 1010 0101 frm	0
> 1010 0101 fcsr	0

mepc Details:

Name : mepc
Hex:0x1a94
Decimal:6804
Octal:015224
Binary:1101010010100
Default:0x1a94 <wrong_instruction+8>

内存查看器

- Debug Mode -> Window -> Show View -> Memory -> Add Monitors
 - 查看、监控内存数据
 - 修改内存数据

The screenshot shows a debugger's memory window. The left pane displays a list of memory addresses and their corresponding data. A red box highlights a specific row of data. The right pane shows a table of memory addresses and their corresponding data, with a red box highlighting a specific row of data.

Address	0 - 3	4 - 7	8 - B	C - F
0000FE0	13070600	13070600	13070600	13070600
0000FF0	13070600	13070600	13070600	13070600
00001000	97210000	938101A6	17210000	13018128
00001010	17150000	130505C4	97150000	938585C3
00001020	13868182	63FCC500	83220500	23A05500
00001030	13054500	93854500	E3E8C5FE	13858182
00001040	93858182	6378B500	23200500	13054500
00001050	E36CB5FE	EF000001	EF000060	6F000000
00001060	67800000	130101FF	23248100	23202101
00001070	17140000	130404BE	17190000	130989BD
00001080	33098940	23261100	23229100	13592940
00001090	630E0900	93040000	83270400	93841400
000010A0	13044400	E7800700	E31899FE	17140000
000010B0	130444BA	17190000	1309C9B9	33098940
000010C0	13592940	630E0900	93040000	83270400
000010D0	93841400	13044400	E7800700	E31899FE
000010E0	8320C100	03248100	83244100	03290100
000010F0	13010101	67800000	93870181	83A70700
00001100	23A4A70A	23A6070A	67800000	93870181
00001110	03A80700	B785954C	9385D5F2	8326880A
00001120	0327C80A	B387B602	13861700	B337F600
00001130	2324C80A	37F65158	1306D642	3386C602
00001140	3307B702	B386B602	3307C700	3307D700

缓存清理

- 用于清理指令和数据缓存
 - 数据缓存清理
 - 保证数据从cache写回内存
 - 腾出dcache空间，方便程序加速
 - 实现：asm(".word(0x500f)")
 - 指令缓存清理
 - 保证清空缓存指令，排除分支预测干扰
 - 实现：asm("fence.i");

```
void icache_clear() {
    asm("fence.i");
}

void dcache_clear() {
    asm(".word(0x500F)");
}

void cache_test() {
    typedef unsigned long ul;
    icache_clear();
    dcache_clear();
    ul st = clint_getTime(BSP_CLINT);
    for(int i = 0; i < 128; i++) {
        for(int j = 0; j < 64; j++)
            for(int k = 0; k < 32; k++)
                for(int m = 0; m < 4; m++)
                    a[i*64*32+m] = b[i*64*32+m] * 3.0;
    }
    ul ed = clint_getTime(BSP_CLINT) - st;
    bsp_printf("%s use: %ldms\n\r", __func__, ed/(SYSTEM_CLINT_HZ/1000));
}
```



谢谢