支持中断

在操作系统中，中断（interrupt）是非常重要的一个机制，

> An **interrupt** is a response by the processor to an event that needs attention from the software. An interrupt condition alerts the processor and serves as a request for the processor to interrupt the currently executing code when permitted, so that the event can be processed in a timely manner.
>
> from Wikipedia

在课上提到的DMA就需要中断机制来发挥作用。而CPU要运行操作系统就必须支持中断，RISC-V也对中断进行了详细的规定。

要支持中断需要实现两部分代码，1：CSR指令（Control and Status Registers），2：中断处理。

其中CSR指令是用来处理CSR寄存器相关的指令的，这些寄存器标记了现在CPU执行处于什么状态，机器态或用户态，硬件信息如支持指令集等，需要实现的CSR指令在RISC-V Instruction Set Manual Volume I: User-Level ISA 2.8，2.9节可以看到。

中断处理规定了CPU核心在接收到中断信号时如何进入中断处理代码以及如何退出中断处理代码恢复原来的指令流，这些处理需要利用CSR指令，关于CSR寄存器的相关规范和对应的中断处理的行为更详细的描述在RISC-V Instruction Set Manual Volume II: Privileged Architecture 1，2，3节可以看到。

稍微详细一些的说明：

中断机制的实现

进入中断

- 停止执行当前程序，从寄存器 `mtvec` 定义的 `PC` 地址开始执行 `mtvec[1:0]` 称为 `MODE` 值，表示跳转方式： `MODE` 值为 `0` —— 跳转到的 `PC` 地址为 `{mtvec[31:2],2'b00}` `MODE` 值为 `1` —— 跳转到的 `PC` 地址为 `{{mtvec[31:2] + cause},2'b00}` `cause` 值与中断类型有关
- 更新 `CSR` 寄存器
  - `mcause` 用以记录异常原因 `mcause[31]` 表示是中断还是异常（中断为 `1`，异常为 `0`） `mcause[30:0]` 存上述 `cause` 值，用以表示中断类型
  - `mepc` 用以记录返回地址 对于中断而言，记录的是下一条未被执行的指令，即当前 `ex_pc + 4`（非跳转、分支指令）或 `ex` 计算得的跳转地址（跳转、分支指令）。
  - `mtval` 记录引起异常的地址 由于要实现的是中断，这里偷懒没写
  - `mstatus` 用以表示机器模式状态 `mstatus[3]` 为 `MIE` 域，表示是否相应中断（1为响应，0为不响应） 进入中断时 `mstatus[7] <= mstatus[3], mstatus[3] <= 1'b0`

退出中断

`MRET` 指令用于指示退出中断，CPU执行 `MRET` 指令后的硬件行为有

- 跳转到 `mepc` 寄存器定义的 `PC` 地址继续执行
- 更新 `mcause` `mstatus[3] <= mstatus[7], mstatus[7] <= 1'b1`

一个简单的测试例子

MODE值为1的测试点例子：

测试点运行后会连续输出由1开始的整数，且包含两种中断，中断号分别为0和1，0的效果为输出一个字符I，1的效果为终止程序。其中中断的触发是手动按按键来模拟CPU外围设备的中断。

测试点中中断相关的指令包含如下几个：(手写的RISC-V指令)

```
csrrw zero,mtvec,a5 7390 5730 // 往mtvec写入a5，该指令在程序开始时会被执行，并将mtvec的base设置为24

24: jal     6f00 0005 // 中断0
28: jal     6f00 8006 // 中断1

64: li t2,73    9303 9004 // 中断0
68: lui t4,0x30 b70e 0300
72: sb t2,0(t4) 2380 7e00
76: mret    7300 2030
80: li t2,255   9303 f00f // 中断1
84: lui t4,0x30 b70e 0300
88: sb t2,4(t4) 2382 7e00
```

MODE值为0的测试点例子：

```
void (*interrupt_handler[MAX_INTERRUPTS])();
void (*exception_handler[MAX_INTERRUPTS])();

void handle_trap(void) __attribute((interrupt)); // 规定此函数是中断处理函数，省去手写进入退出中断的相关CSR指令
void handle_trap(){
    unsigned long mcause = read_csr(mcause);
    if (mcause & MCAUSE_INT) { // 判断trap原因为中断
        print("interrupt. cause=)");
        outl(mcause & MCAUSE_CAUSE); // 输出中断号
        interrupt_handler[mcause & MCAUSE_CAUSE](); // 调用对应处理函数
    } else { // 判断trap原因为异常
        print("exception=");
        outlln(mcause & MCAUSE_CAUSE); // 输出异常号
        exception_handler[mcause & MCAUSE_CAUSE](); // 调用应处理函数
    }
}
```