National Yang Ming Chiao Tung University
Computer Architecture & System Lab

Trap

IOC5226 Operating System Capstone

Tsung Tai Yeh
Department of Computer Science
National Yang Ming Chiao Tung University

Acknowledgements and Disclaimer

- Slides were developed in the reference with
  - MIT 6.828 Operating system engineering class, 2018
  - MIT 6.004 Operating system, 2018
  - Remzi H. Arpaci-Dusseau etl. , Operating systems: Three easy pieces. WISC

Outline

- Trap
- RISC-V Control Registers
- Setup Trap Handler

What is an trap?

- **What is a trap?**
  - Causes the CPU to stop normal instruction flow
  - Jump to a trap handler running in a more privileged mode
- **Two types of traps**
  - **Exceptions (synchronous)**
    - Occur during the execution of an instruction
    - They are caused by the instruction itself
      - System calls: using `ecall` to request OS services
      - Errors: division by zero, illegal instructions
      - Access memory without correct permissions (page fault)

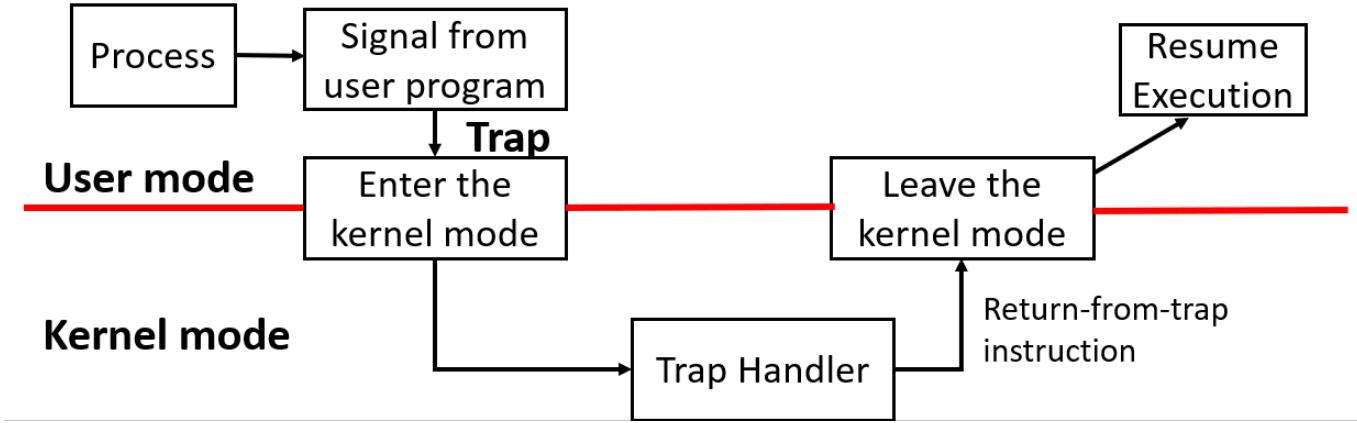
What is an trap?

- **Two types of traps**
  - **Interrupts (Asynchronous)**
    - Occur outside the normal flow, can happen at any time
    - Caused by external hardware events
      - **Timer interrupt:**
        - Used to switch between processes
      - **Software interrupt**
        - One processor core signaling another
      - **External interrupt**
        - Input/output devices such as keyboard, disk

Trap in the OS

1. During a trap, the execution of a process is set as high priority compared to user code
2. When the OS detects a trap, it pauses the user process
3. The OS resumes the execution when the system call is completed

- Using a **trap** that is a synchronous interrupt triggered by an exception in a user process to execute functionality.



Examples of Triggering Traps

- **System call**
  - When a user program executes the `ecall` instruction to ask the kernel to do something for it
- **Exception**
  - An instruction (user or kernel) does something illegal
- **Device interrupt**
  - When a device signals that it needs attention
  - E.g. when the disk finishes a read or write request

RISC-V trap machinery

- Control registers that tell the CPU how to handle traps
  - `stvec`: the kernel writes the address of its trap handler here
  - `sepc`: RISC-V saves the PC here.
    - The `sret` (return from trap) instruction copies `sepc` to pc
    - The kernel can write `sepc` to control where `sret` goes
  - `scause`: Puts a number that describes the reason for the trap
  - `sscratch`: The trap handler uses it avoid overwriting user registers before saving them
  - `sstatus`: SIE bit controls whether device interrupts are enabled; SPP bit indicates a trap comes from U/S-mode

Key Components to Trap in RISC-V

- **Trap Vector Table (mtvec/stvec)**
  - The entry point for the CPU when a trap occurs
  - Tell the hardware exactly where to jump to find the code that handles an interrupt or exception
  - The last two bits determine mode
    - Direct
      - all trap jump to the one address
    - Vectored
      - Different address per cause

Key Components to Trap in RISC-V

- **Trap handler code (trap vector)**
  - The entry point that saves registers
    - ra, a0 – a7, t0 – t6 to memory
- **Interrupt/Exception Registers**
  - `mcause/scause:`
    - identifies why the trap occurred (e.g. ecall, divide by zero)
  - `mepc/sepc`
    - Store the address of the interrupted instruction
  - `mstatus/sstatus`
    - Hold privilege mode, interrupt enable bits …

Key Components to Trap in RISC-V

- **Return instructions**
  - Use `mret/sret` instructions to restore the PC from `mepc/sepc`
  - Return to the previous privilege mode

When a Trap occurs

- When a trap occurs
  - RISC-V CPU performs “atomic” steps before software takes
  - 1. Disable interrupts by clearing the SIE bit in `sstatus`
  - 2. Save the PC into `sepc`
  - 3. Save the current mode in the SPP bit in `sstatus`
  - 4. Set `scause` to reflect the trap’s cause
  - 5. Set the mode to supervisor
  - 6. Copy `stvec` to the pc
  - 7. Start executing at the new cp

https://reurl.cc/L25zDe

Setup Trap

- **Step 1: Trap Vector Setup**
  - The trap vector tells the CPU where to go when a trap happens

```
la      t0,   trap_handler     # Load trap handler address
csrw    mtvec,  t0             # Set machine trap vector
```

  - Direct mode -> all traps go to the same address
  - Vectored mode -> traps jumps to a base + (cause x 4)

https://reurl.cc/L25zDe

Setup Trap

- **Step 2: Enabling interrupt**
  - To receive interrupt, RISC-V needs
    - Global enable in mstatus
    - Specific enable in mie
    - Set MIE in mstatus -> globally allows interrupts

```
li      t0,   (1 << 3)          # MIE bit in mstatus
csrs    mstatus,  t0

li      t0,   (1 << 7)          # Enable machine timer interrupt
csrs    mie,  t0
```

https://reurl.cc/L25zDe

Setup Trap

- **Step 3: PLIC setup for external interrupts**
  - The platform-level interrupt controller routes device interrupts
  - Enable a specific interrupt ID (e.g. UART, GPIO)
  - Set priority threshold to 0 so all priorities are accepted

```
# Set priority for UART interrupt (ID 10)
li    t0,   1
la    t1,   PLIC_PRIORITY_BASE
sw    t0,   40(t1)
# Enable source for target hart
la    t1,   PLIC_PRIORITY_BASE
li    t0,   (1 << 10)
sw    t0,   0(t1)
```

https://reurl.cc/L25zDe

Setup Trap

Bare-Metal
External Interrupt

```
    .section .init
    .globl _start
_start:
    # Set trap vector base address (direct mode)
    la   t0, trap_vector
    csrw mtvec, t0

    # Enable machine external interrupt in mie
    li   t0, (1 << 11)       # MEIE = bit 11 in mie
    csrs mie, t0

    # Enable global machine interrupts
    li   t0, (1 << 3)        # MIE = bit 3 in mstatus
    csrs mstatus, t0

    # (Platform-specific) Enable external interrupt source in PLIC
    # Example: Enable interrupt ID=1 (UART)
    li   t0, 1
    la   t1, PLIC_ENABLE
    sw   t0, 0(t1)

    # Set priority threshold = 0
    la   t1, PLIC_THRESHOLD
    sw   x0, 0(t1)

main_loop:
    wfi                      # Wait for interrupt
    j   main_loop
```

https://reurl.cc/L25zDe

Setup Trap

Trap Handler Implementation

```
# ------------------------------------------------------------
# Trap handler for all interrupts and exceptions
# ------------------------------------------------------------
    .align 4
trap_vector:
    csrr t0, mcause           # Read cause
    bgez t0, handle_exception # If bit 31 == 0, it's exception
    # Interrupt: high bit set
    li   t1, 0x8000000b       # Machine external interrupt code
    beq  t0, t1, handle_mei
    j    trap_done

handle_mei:
    # Read interrupt claim from PLIC
    la   t2, PLIC_CLAIM
    lw   t3, 0(t2)

    # Here: handle the device (UART, GPIO, etc.)
    # For demo: just toggle an LED register
    la   t4, LED_REG
    lw   t5, 0(t4)
    xori t5, t5, 1
    sw   t5, 0(t4)

    # Complete interrupt in PLIC
    sw   t3, 0(t2)
    j    trap_done

handle_exception:
    # Exception handling code (not covered here)
    j    trap_done

trap_done:
    mret                      # Return from trap
```

https://reurl.cc/L25zDe

Interrupt Flow

- Step 1
  - Wait for interrupt halts until an interrupt occurs
- Step 2
  - When a device triggers, PLIC signals M-mode -> CPU jumps to trap_vector
- Step 3
  - mcause high bit = interrupt (1), low bits = cause code (11 = MEI)
- Step 4
  - Claim the interrupt ID from PLIC, handle the device, then complete the claim

https://reurl.cc/L25zDe

System call with ecall

- OS uses traps to switch from user mode to kernel mode
  - The OS trap handler reads a7 to determine the syscall and services it before sret

```
    li a7, 64           # syscall number: write
    li a0, 1            # fd: stdout
    la a1, message      # buffer address
    li a2, 13           # length
    ecall               # trap to OS

message:
    .asciz "Hello, RISC-V!\n"
```

How S-mode Interrupt Handling Works

- M-mode Firmware delegates interrupts to S-mode
  - M-mode firmware (as OpenSBI) is still required
    - Hardware boots in M-mode
  - The firmware's jobs
    - mideleg -> delegates interrupts (e.g. timer) to S-mode
    - medeleg -> delegates exceptions to S-mode
    - stvec -> points to the S-mode trap handler

https://reurl.cc/L25zDe

How S-mode Interrupt Handling Works

- OS runs entirely in S-mode
  - All delegated interrupts go directly to the S-mode trap handler
  - The S-mode trap handler
    - Read scause (cause of trap/interrupt)
    - Read sepc (return address)
    - Services the interrupt (e.g. from a PLIC)
      - PLIC is memory-mapped and can be accessed from S-mode (if M-mode firmware allows)
    - Return with sret

https://reurl.cc/L25zDe

S-mode Trap Handler

```
    .section .text
    .globl _start
_start:
    # Assume we are already in S-mode (M-mode firmware handled the switch)

    # Set S-mode trap vector
    la   t0, supervisor_trap_vector
    csrw stvec, t0

    # Enable S-mode external interrupt (SEIE)
    li   t0, (1 << 9)      # SEIE = bit 9 in sie
    csrs sie, t0

    # Enable global S-mode interrupts
    li   t0, (1 << 1)      # SIE = bit 1 in sstatus
    csrs sstatus, t0

    # Enable PLIC source for a device (example: interrupt ID 1)
    la   t1, PLIC_ENABLE
    li   t2, 0x1
    sw   t2, 0(t1)

    # Set PLIC priority threshold to 0
    la   t1, PLIC_THRESHOLD
    sw   x0, 0(t1)

main_loop:
    wfi                    # Wait for interrupt
    j main_loop
```

li t0, (1<<9) # delegate supervisor external interrupt

https://reurl.cc/L25zDe

S-mode Trap Handler

```
# ------------------------------
# S-mode Trap Handler
# ------------------------------
supervisor_trap_vector:
    csrr t0, scause
    bgez t0, handle_s_exception  # If bit 31 = 0 → exception
    # Interrupt: bit 31 = 1
    li   t1, 0x80000009          # Code 9 = Supervisor External Interrupt
    beq  t0, t1, handle_sei
    j    trap_done

handle_sei:
    # Read interrupt ID from PLIC claim/complete register
    la   t2, PLIC_CLAIM
    lw   t3, 0(t2)

    # Handle device interrupt (example: toggle LED)
    la   t4, LED_REG
    lw   t5, 0(t4)
    xori t5, t5, 1
    sw   t5, 0(t4)

    # Complete interrupt at PLIC
    sw   t3, 0(t2)
    j    trap_done

handle_s_exception:
    # Handle exception (optional)
    j    trap_done

trap_done:
    sret                         # Return to interrupted code
```

Bit 31 -> interrupt, else exception

PLIC claim/complete done -> sret

https://reurl.cc/L25zDe

Trap Flow Recap

- 1. Normal Execution
- 2. Event occurs (exception or interrupt)
- 3. CPU saves state into CSRs
  - mepc/sepc, mcause/scause, mtval/stval
- 4. PC -> trap vector (mtvec/stvec)
- 5. Handler runs
- 6. mret/sret restores state and return to main thread

https://reurl.cc/L25zDe

RISC-V Traps Summary

- Trap means a transfer of control to trap handler
- System calls, exception, and interrupt cause CPU to force a trap
- In bare-metal embedded system
  - Trap are usually executed in M-mode
- In OS-based systems
  - Traps could be delegated to OS kernel in S-mode

https://reurl.cc/L25zDe

Takeaway Questions

- When will trigger the trap?
  - (A) System call
  - (B) Interrupts
  - (C) Divide by zero
- Which register is used to store the reason of the trap?
  - (A) stev
  - (B) scause
  - (C) sepc

Takeaway Questions

- What are purpose of the trap vector?
  - (A) Entry point for the CPU when a trap occurs
  - (B) Tell the CPU to jump to the code that handle an interrupt
  - (C) Triggers the trap handler