National Yang Ming Chiao Tung University
Computer Architecture & System Lab

Interrupt & Exceptions

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

- Interrupt
- Software Interrupt -- Exception
- Hardware Interrupt
- Interrupt Vector
- Interrupt Workflow

What is an interrupt?

- **What is an interrupt?**
  - An interrupt is a hardware signal from a device to the CPU
  - Tells the CPU that the device needs attention
  - CPU should stop performing what it is doing and respond to the device
- **What kinds of interrupts do we have?**
  - Hardware interrupt
  - Software interrupt

What is an interrupt?

- **What is interrupt handler?**
  - Service the device and stop it from interrupting
- **What is the job of an interrupt handler?**
  - Save additional CPU context (written in assembly)
  - Process interrupt (communicate with I/O devices)
  - Invoke kernel scheduler
  - Restore CPU context and return (written in assembly)

What is an interrupt?

- **When an interrupt occurs …**
  - **Preempt current task**
    - The kernel must pause the execution of the current process
  - **Execute interrupt handler**
    - Search for the handler of the interrupt and transfer control
  - **After the interrupt handler completes execution**
    - The interrupted process can resume execution

Exceptions

- Exceptions
  - Events generated by the CPU in response to exceptional conditions when executing instructions
  - Exceptions usually trigger an exception handling
    - The exceptional condition may be dealt with before the CPU continue executing the program
  - Causes the CPU to redirect the execution flow to a system routine

Exceptions

- Exceptions
  - Causes the CPU to redirect the execution flow to a system routine
    - Save the current program context
    - Handle the exceptional condition
    - Restore the context of the saved program to continue the exception

Exceptions

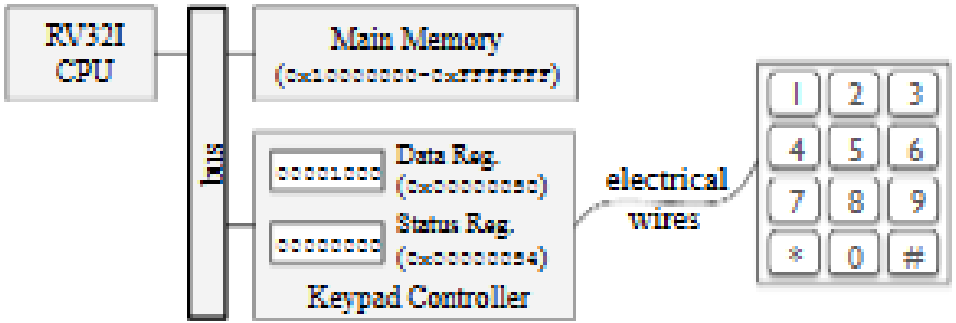
- **Exception**
  - Caused by an exceptional condition in the processor itself
  - An example of an exceptional condition is division by zero
  - Exiting a program with syscall instruction
- Categories
  - **Faults**: an exception reported before the execution of a “faulty” instruction
  - **Traps**: an exception reported by the trap instruction
  - **Aborts**: an exception doesn’t always report the exact instruction which caused the exception

Hardware Interrupt

- **Why do we need the hardware interrupt?**
  - Several devices connected to the CPU
    - E.g. keyboards, mouse, network card, etc.
  - These devices occasionally need to be serviced by the CPU
    - Tell the CPU that a key has been pressed
    - Interrupts can occur at any time
    - Need a way for the CPU to determine when a device needs attention

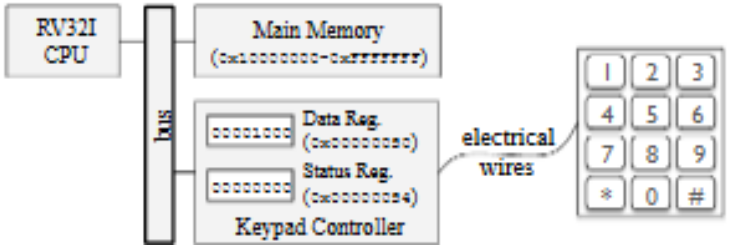
External Interrupt

- **When do we need interrupt?**
  - When the keypad is pressed
  - The keypad controller registers this information
  - This information requires the CPU attention and perform some actions
  - There must be a way to inform the CPU that peripheral needs its attention

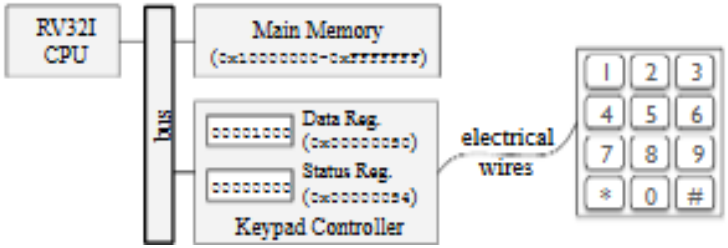


External Interrupt

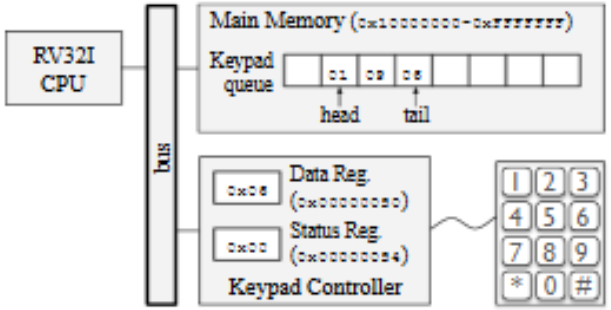
- The keypad controller contains two 8-bit register
  - The data register
    - Indicates the last key pressed on the keypad
  - The status register
    - Indicates the keypad's current status
    - Its LSB bit is the REDAY bit that indicates whether the keypad was pressed since the last time the CPU read a value from the data register



External Interrupt



- The keypad controller contains two 8-bit register
  - The status register
    - Its second LSB bit is called the OVRN bit that indicates whether the keypad was passed more than once
  - Data overrun (OVRN)
    - The keypad controller has only one data register
    - If keypad is pressed more than once before the CPU gets the chance to read the data register, one or more key values are lost
    - How to resolve this OVRN problem?



External Interrupt

- To prevent data overruns
  - Copy the data register value to a FIFO queue located at the main memory as soon as the keypad is pressed
  - FIFO queue is an 8-element circular buffer and two pointers
    - Pointers points to queue's head and tail
    - Whenever a key is pressed, its value is pushed into the queue's tail
    - CPU must execute this routine as soon as possible to prevent data overruns

External Interrupt

- There are two main methods to direct the CPU handle events caused by external hardware
  - Polling
  - Hardware interrupt

Polling

- **Polling**
  - The CPU periodically checks whether peripherals need attention
  - Assuming the peripheral program is designed
  - Whenever there is a peripheral that needs attention, the program invokes a routine to handle the peripheral

```
for p in Peripherals do
    if keypadPressed() then
        if needsAttention(p) then
            handlePeripheral(p);
        end
    end
    PerformSomeComputation();
end
```

Polling

- **Polling**
  - keypadPressed()
    - checks if the keypad READY bit is set
  - getKey():
    - Read the contents of the data register
  - pushKeyOnQueue()
    - Push it to the queue's tail
  - Compute ()
    - Represents the work that is done by the program

```
While True do
    if keypadPressed() then
        k <- getKey()
        pushKeyOnQueue(k);
    end
    Compute();
end
```

Polling

- **Polling**
  - The Compute() affects the frequency in which the keypad is checked
  - The longer Compute() -> increase the occurring data overrun
  - Could we directly remove Compute()?
  - Polling is usually not the best approach to check for and handle peripheral events, why?

```
While True do
    if keypadPressed() then
        k <- getKey()
        pushKeyOnQueue(k);
    end
    Compute();
end
```

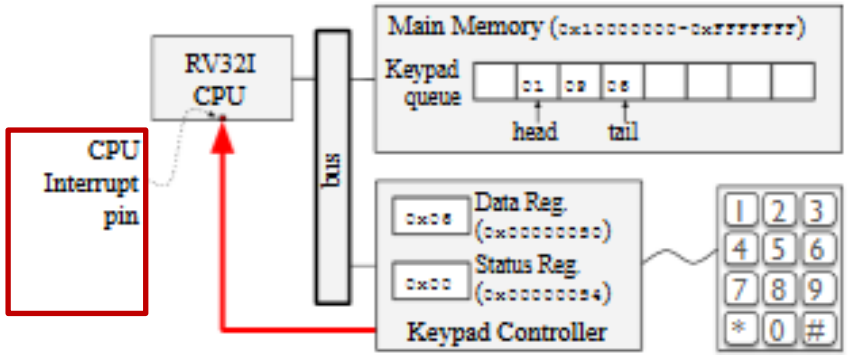
Hardware Interrupt

- **Hardware interrupt**
  - Allows hardware to inform the CPU they require attention
  - Caused by external (non-CPU) hardware such as peripherals to inform the CPU they require attention
  - The peripheral sends an interrupt to the CPU

Hardware Interrupt

- **Hardware interrupt**
  - Once the CPU receives this signal
    - The CPU saves the context of the current program
    - Invokes a routine to handle the hardware interrupt
    - Restores the context of the saved program and continues executing

Hardware Interrupt



- **Hardware interrupt**
  - The CPU contains an interrupt pin and the keypad controller is connected to the CPU interrupt pin
  - The interrupt pin as an input pin
    - Informs the CPU whether or not there is an external interrupt
    - The CPU constantly monitors the interrupt pin and interrupts the current execution flow to execute and ISR
    - The interrupt service routine (ISR) is a software routine that handle the interrupt

Hardware Interrupt

- **Detecting external interrupt**
  - Before fetching an instruction for execution
  - The CPU verifies if the interrupt.pin is set
  - If the CPU receives an interrupt signal & interrupts are enabled
    - The CPU saves the program counter (PC) into the SAVED_PC register
    - Set the PC register with (ISR_ADDRESS) and disable interrupts by clearing the interrupt.enabled register
    - Then, the next instruction (ISR) will be fetched

Hardware Interrupt

- **Invoking the proper ISR**
  - Each peripheral usually requires a specialized routine to handle its interrupts
  - SW-only design, the ISR is responsible for
    - Identifying which peripheral interrupted the CPU
    - Invoking the proper routine to handle the interrupt
    - Upon an interrupt
      - The CPU invokes a generic ISR doing above tasks
      - The ISR may have to interact with all peripherals to find out which one is requiring the CPU attention

Hardware Interrupt

- **Invoking the proper ISR**
  - SW-only design pros & cons
    - Simplifies the CPU hardware design
    - The ISR may take a long time trying to figure out which peripheral interrupted the CPU

RISC-V registers for interrupts

- **Control and Status Registers (CSR)**
  - Special registers that expose the CPU status to the software
  - Allow the software to configure the CPU behavior
  - In RV32I ISA
    - **`mstatus`** `CSR` is a 32-bit register that exposes the current status of the CPU
    - **`csrrw`** `rd, csr, rs1` instruction
      - Atomically swaps the contents of register a0 and CSR
    - **`csrr`** `a0, mstatus` instruction
      - Copies the contents of the mstatus CSR into a0

RISC-V registers for interrupts

- **Interrupt related CSR registers**
  - **`mip`** (Machine Interrupt Pending)
    - Interrupts are pending (i.e., have been signaled but not handled by the CPU yet)
  - **`mepc`** (Machine Exception Program Counter):
    - The CPU saves the contents of the PC register into mpec
  - **`mscratch`** (Machine Scratch)
    - Is visible in machine mode

RISC-V registers for interrupts

- **Interrupt related CSR registers**
  - **`sie`:** supervisor interrupt enable register
    - One bit per software interrupt, external interrupt, and timer interrupt
  - **`sstatus`**: supervisor status register
    - SIE bit enables or disables interrupts
    - Other bits record the mode when interrupt occurred
  - **`sip`**: supervisor interrupt pending register
    - Which type of interrupts are waiting to be handled

RISC-V registers for interrupts

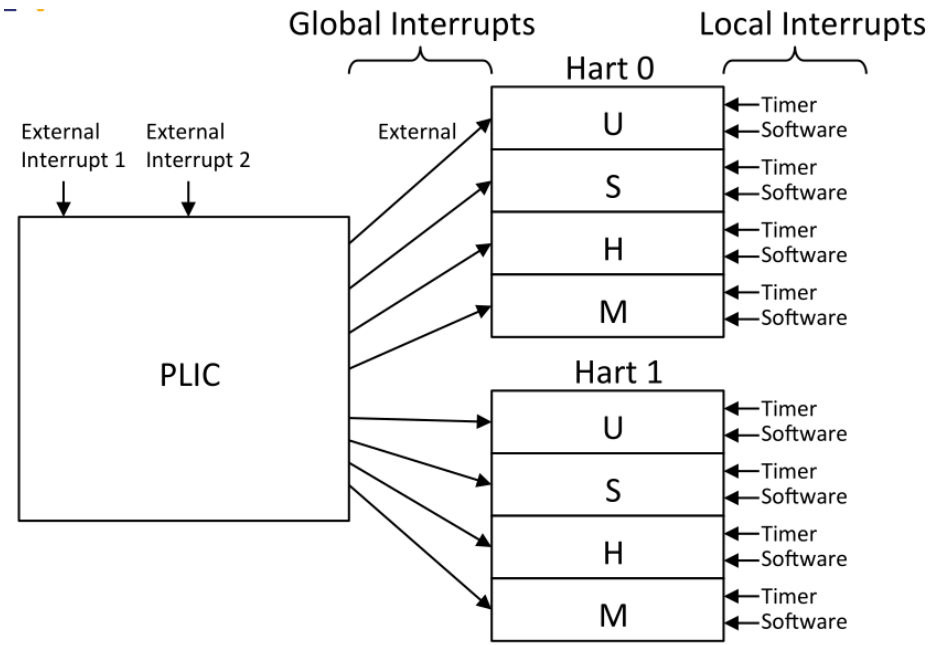
- **Interrupt related CSR registers**
  - **`scause`: the platform-specific cause of the interrupt**
    - Records the specific cause of a trap in Supervisor Mode
    - Holds a code indicating the reason for the trap, allowing the software trap handler to understand what happened.
    - Common scause Exception Codes
      - 0: Instruction address misaligned
      - 2: Illegal instruction
      - 8: Environment call from U-mode (syscall)
      - 12: Instruction page fault

RISC-V registers for interrupts

- **Interrupt related CSR registers**
  - **`stvec`: Supervisor Trap Vector Base Address Register**
    - A read/write CSR register
    - Hold the entry point address for supervisor-level trap
    - Stores information that allows the CPU to identify the proper ISR
    - When exceptions or interrupts occur in S-mode or U-mode
      - The CPU jumps to the address stored in `stvec`

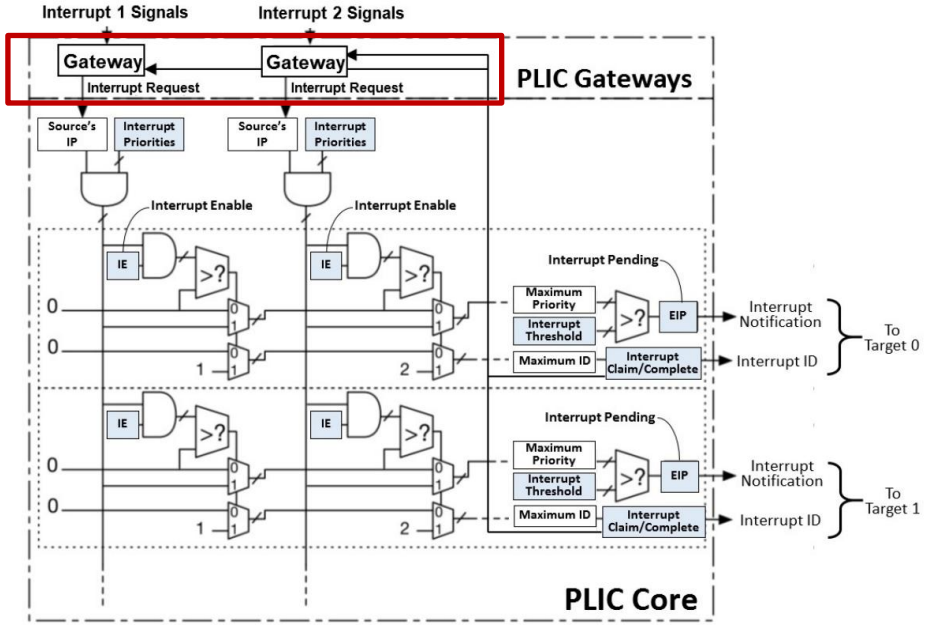
Where do interrupts come from

- **PLIC: Platform-level interrupt controller**
  - Tackle external interrupt
- **CLINT: Core-local interrupt controller**
  - Handle local interrupt



PLIC: Platform-level interrupt controller

- **Interrupt Gateway**
  - A hardware component in PLIC
  - Serves as the bridge between external peripherals and the PLIC core



PLIC: Platform-level interrupt controller

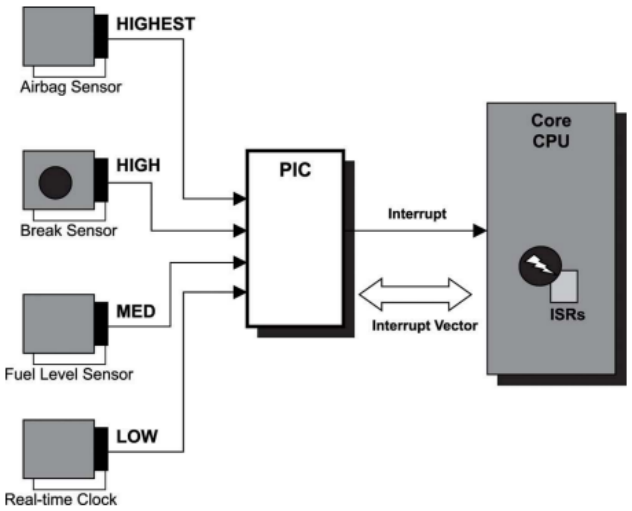
- **Interrupt Gateway**
  - **Signal conversion**
    - Convert diverse physical interrupt signals from external devices into a “interrupt request” format that the PLIC core can understand
  - **Flow Control**
    - Prevent the PLIC core from being overwhelmed
    - Ensure that only one request per interrupt source is pending at any given time

PLIC: Platform-level interrupt controller

- **Interrupt Gateway**
  - **Blocking Logic**
    - Once a gateway sends a request to the PLIC core, it will not forward another request from the same source
  - **Trigger Management**
    - Manage specific trigger behaviors
    - The gateway forwards the request when the level is first asserted
    - Only checks again once the current interrupt is fully serviced

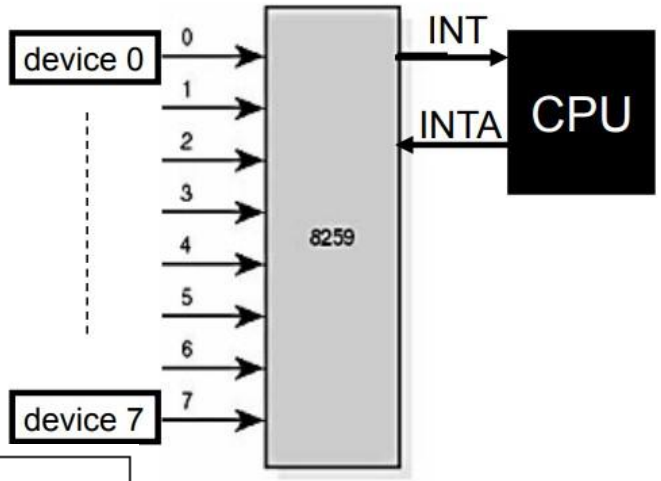
Hardware Interrupt

- **Programmable Interrupt Controller (PIC)**
  - Responsible for sequential multiple interrupt requests from devices
  - Advanced PIC (APIC)
    - Local APIC
      - Located on each CPU core
      - Handle interrupts from APIC-timer, thermal sensor
    - I/O APIC
      - Distributed external interrupts among the CPU cores



Hardware Interrupt

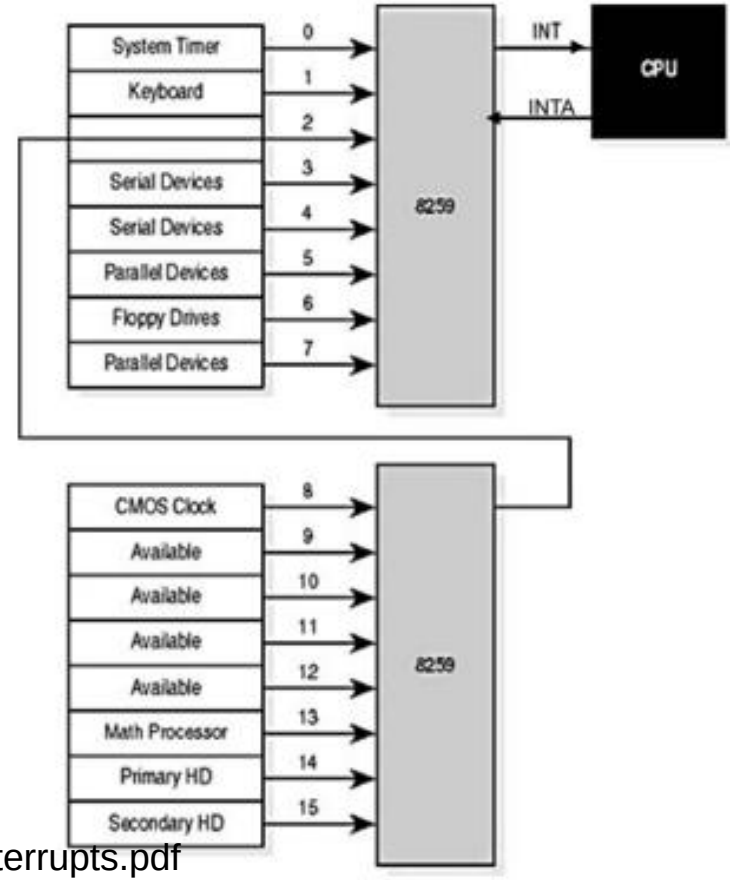
- 8259 PIC relays up to 8 interrupts to the CPU
  - Devices raise interrupts by an ‘interrupt request’ (IRQ)
  - CPU acknowledges and queries the 8259 to determine which device interrupted
  - Priorities can be assigned to each IRQ line
  - 8259s can be cascaded to support more interrupts
    - Two PICs and cascade buffer
    - IRQ2 -> IRQ9



INTA is a signal used to identify that a CPU has an interrupt made by the interrupt controller.

Hardware Interrupt

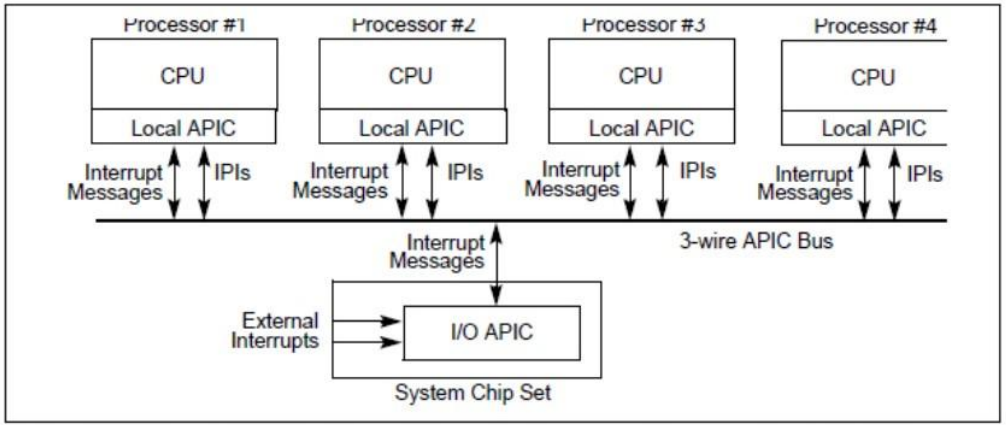
- IRQ 0 to IRQ 15, 15 possible devices
- Interrupt types
  - Edge
  - Level
- Limitations
  - Limited IRQs
  - Multi-processor support limited



http://www.cse.iitm.ac.in/~chester/courses/16o_os/slides/6_Interrupts.pdf

Hardware Interrupt

- **Advanced PIC (APIC)**
  - External interrupts are routed from peripherals to CPUs in multi-processor systems through APIC
  - APIC distributes and prioritizes interrupts to processors
  - APICs communicates through a special 3-wire APIC bus



Hardware Interrupt

- **LAPIC**
  - Receives interrupts from I/O APIC and routes it to the local CPU
  - Can also receive local interrupts such as thermal sensors, internal timers, etc.
  - Send and receive IPIs (Inter-processor interrupts)
    - IPIs are used to distribute interrupts between processors or execute system-wide functions like booting, load distribution, etc.
- **I/O APIC**
  - Present in the chipset (northbridge)
  - Used to route external interrupts to local APIC

RISC-V Interrupt Vector

- **Interrupt vector**
  - Directly jump to a specific interrupt service routine (ISR) address based on the interrupt source
  - The `mtvec` control register supports direct or vectored modes
  - Vector modes
    - **Direct Mode (MODE 0)** : all exceptions and interrupts jump to the same address defined in mtvec
    - **Vectored Mode (MODE 1)**
      - Exceptions jump to the `mtvec` base address
      - Interrupts jump to BASE + 4 * mcause

RISC-V Interrupt Vector

- **Vectored Mode Operation Flow**
  - **Interrupt arrives**
    - A hardware device raises an interrupt
  - **PLIC Arbitration**
    - The PLIC selects the highest-priority interrupt
  - **Trap Triggered**
    - The HART (CPU) traps, setting mcause with interrupt ID
  - **Vector Jump**
    - If mtvec is in vectored mode, the PC jumps to BASE + (Interrupt ID x 4)
  - **ISR execution**

RISC-V Interrupt workflow

- **Trap Triggering (Hardware)**
  - When an interrupt occurs (enabled via `mstatus.MIE` and `mie`)
  - **Save the PC**
    - The address of the current (or next) instruction is copied into the mepc
  - **Update Status**
    - The current interrupt enable bit (mstatus.MIE) is copied to mstatus.MPIE (Previous Interrupt Enable)
    - mstatus.MIE is set to 0, disable further interrupts
    - The current privilege mode is saved in mstatus.MPP

RISC-V Interrupt workflow

- **Trap Triggering (Hardware)**
  - **Set the Cause**
    - The reason for the interrupt (Timer, External) is written to the `mcause` register
  - **Jumps to Vector**
    - The CPU changes the PC to the address stored in `mtvec`

RISC-V Interrupt workflow

- **Context Saving (Software)**
  - Once the CPU jumps to the address in `mtvec`, the ISR begins
  - **Push registers**
    - Save the current value of x1 – x31 registers onto the stack (saving the context)
  - **Identify Source**
    - In Direct Mode, the software reads `mcause` to determine which specific interrupt occurred and branches to the correct handler

RISC-V Interrupt workflow

- **Execution (Software)**
  - The actual logic of the interrupt is performed
  - Clearing the interrupt pending bit
  - Performing necessary computations or OS scheduling tasks
- **Trap Return (Software & Hardware)**
  - To go back to the original program, the software execute `mret` instruction (M-Mode)

RISC-V Interrupt workflow

- **Trap Return (Software & Hardware)**
  - Set the PC to the value stored in `mepc`
  - Restores the interrupt state by copying `mstatus.MPIE` back to `mstatus.MIE`
  - Restore the privilege model from `mstatus.MPP`

Summary Table for RISC-V Interrupts

Phase	Responsibility	Key Register Involved
Trap Entry	Hardware	mepc, mcause, mstatus, mtvec
Context Save	Software (ISR)	Stack pointer (sp), x1 – x31
Processing	Software (ISR)	User-defined
Context Restore	Software (ISR)	Stack pointer (sp), x1 – x31
Trap Exit	Hardware (mret)	mepc, mstatus

Summary

- Interrupt changes the sequence of instruction execution
- Exception occurs since the illegal operation
- Hardware interrupt – programmable interrupt controller
- Interrupt vector records interrupt commands

Takeaway Questions

- Who can issue an interrupt?
  - (A) Network card
  - (B) GPU
  - (C) Keyboard
- How do peripherals tell the CPU to know their requests?
  - (A) External interrupt
  - (B) Polling
  - (C) Exception

Takeaway Questions

- Which situations will raise the exception?
  - (A) External interrupt
  - (B) Software interrupt
  - (C) Divided by zero
- What are purposes of interrupt vector?
  - (A) Recognizing the type of interruption or exception
  - (B) Improve the performance of the CPU
  - (C) Raise the privilege of executed instructions