National Yang Ming Chiao Tung University
Computer Architecture & System Lab

Interrupt

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
- Hardware Interrupt
- Interrupt Workflow
- Software Interrupt -- Exception
- Interrupt Vector
- Interrupt Descriptor Table
- Interrupt Stack Table

What is an interrupt? (1/6)

- **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
- **Interrupt handler?**
  - Service the device and stop it from interrupting
- **What kinds of interrupts do we have?**
  - Hardware interrupt
  - Software interrupt

What is an interrupt? (2/6)

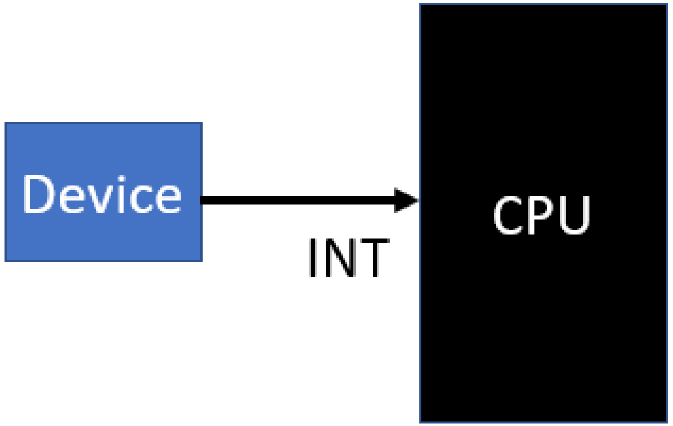
- **What is the job of an interrupt handler?**
  - Save additional CPU context (written in assembly)
  - Process interrupt (communicate with I/O devices)
  - Invoke kernel scheduler
  - Restore CPU context and return (written in assembly)

Hardware Interrupt (3/6)

- **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

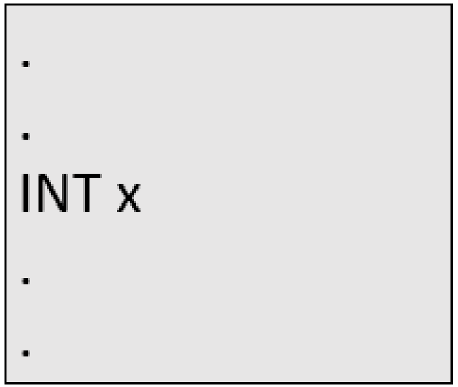
What is an interrupt? (4/6)

Hardware interrupt



A device (programmable interrupt controller (PIC)) asserts a pin in the CPU

Software interrupt



An executed instruction causes an interrupt

What is an interrupt? (5/6)

- **Synchronous interrupt**
  - Produced by the CPU control unit while executing instructions
  - The control unit issues interrupt only after terminating the execution of an instruction
- **Asynchronous interrupt**
  - Generated by other hardware devices at arbitrary times with respect to the CPU clock signals

What is an interrupt? (6/6)

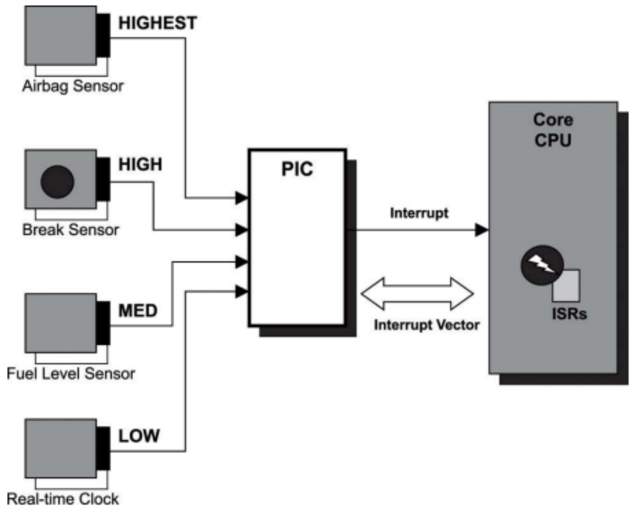
- **When an interrupt occurs …**
  - **Preempt current task**
    - The kernel must pause the execution of the current process
  - **Execute interrupt handler**
    - Search for the handler of the interrupt and transfer control
  - **After the interrupt handler completes execution**
    - The interrupted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

Outline

- Interrupt
- Hardware Interrupt
- Interrupt Workflow
- Software Interrupt -- Exception
- Interrupt Vector
- Interrupt Descriptor Table
- Interrupt Stack Table

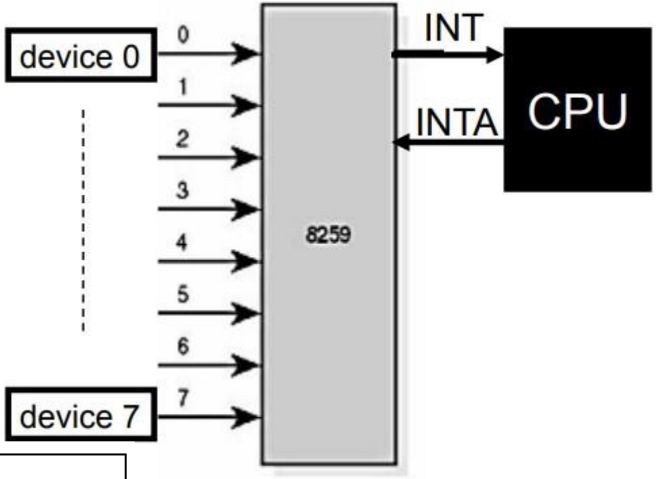
Hardware Interrupt (1/5)

- **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 (2/5)

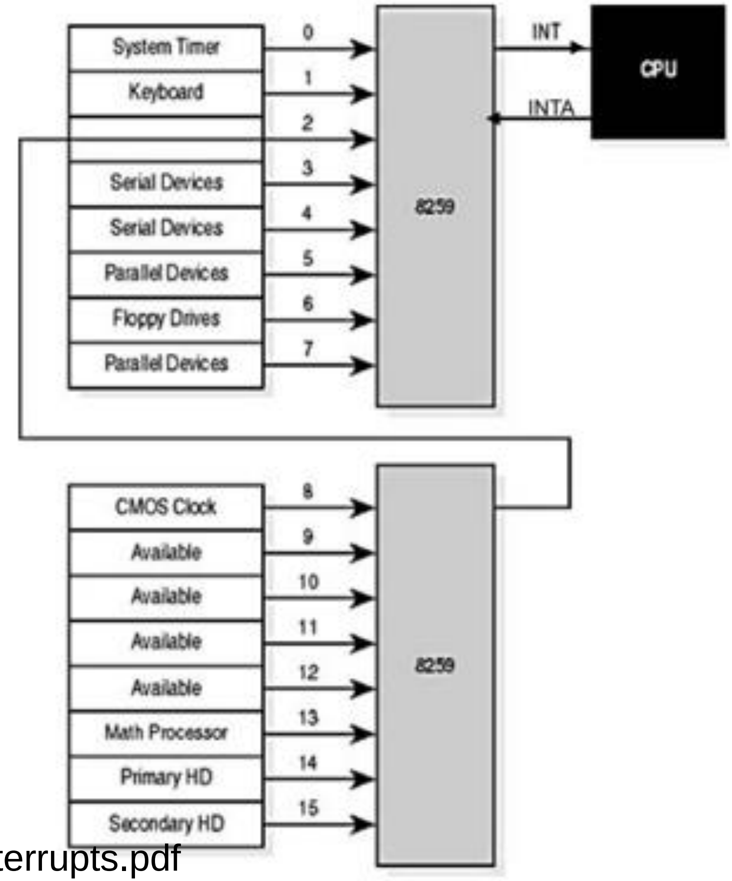
- 8259 PIC relays up to 8 interrupts to the CPU
  - Devices raise interrupts by an ‘interrupt request’ (IRQ)
  - CPU acknowledges and queries the 8259 to determine which device interrupted
  - Priorities can be assigned to each IRQ line
  - 8259s can be cascaded to support more interrupts
    - Two PICs 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 (3/5)

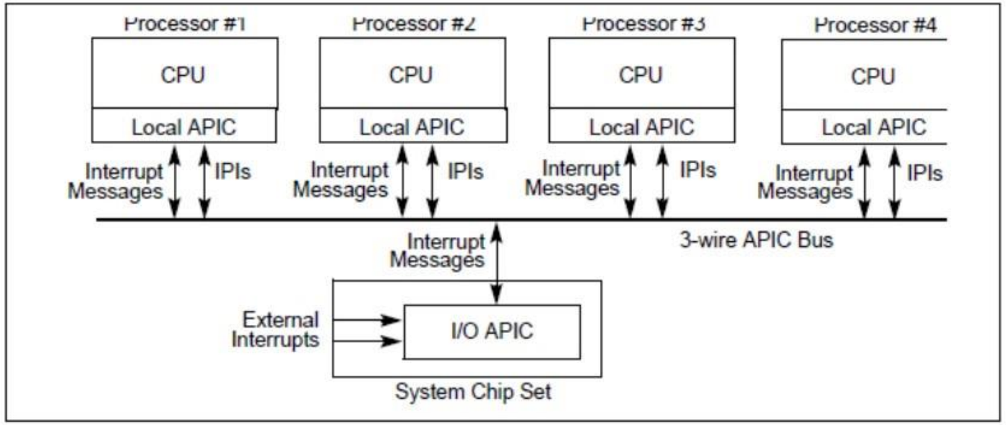
- IRQ 0 to IRQ 15, 15 possible devices
- Interrupt types
  - Edge
  - Level
- Limitations
  - Limited IRQs
  - Multi-processor support limited



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

Hardware Interrupt (4/5)

- **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



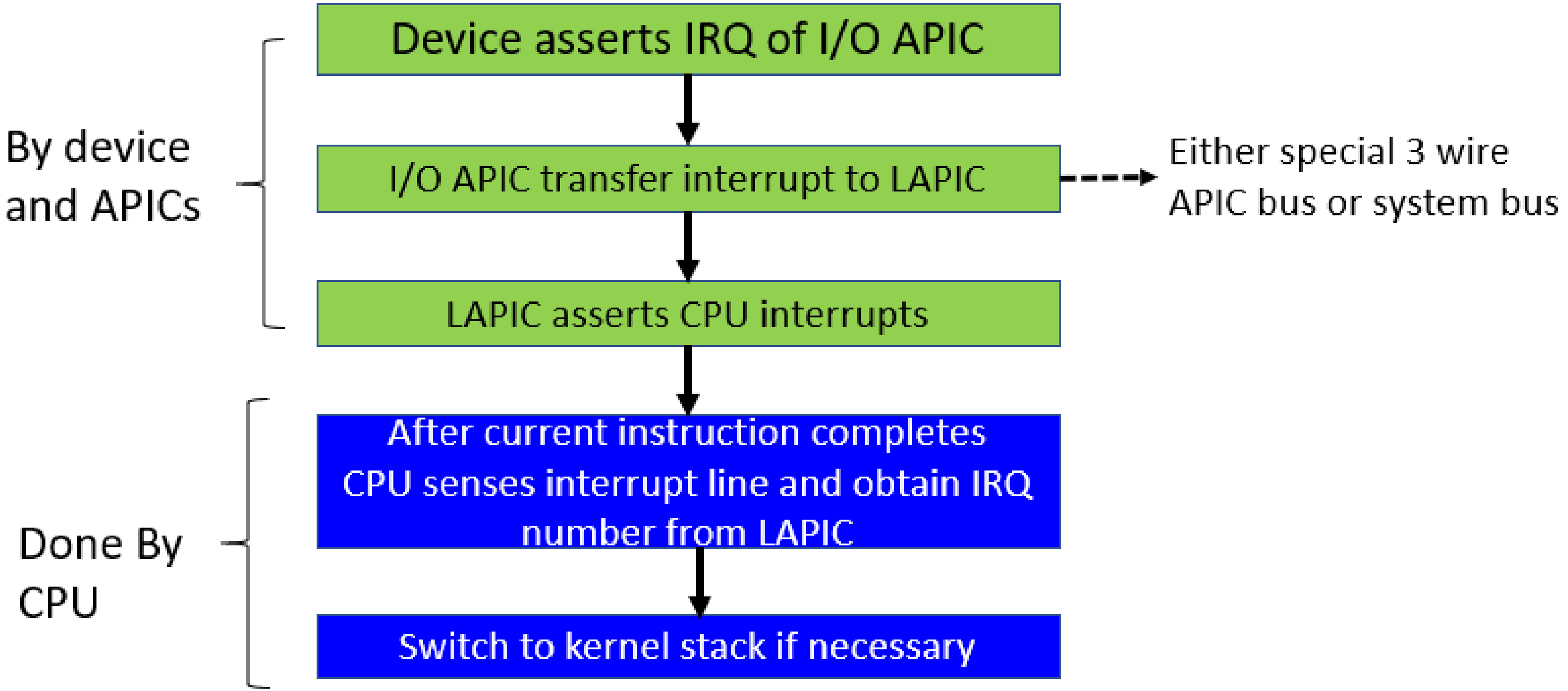
Hardware Interrupt (5/5)

- **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 (Interprocessor 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

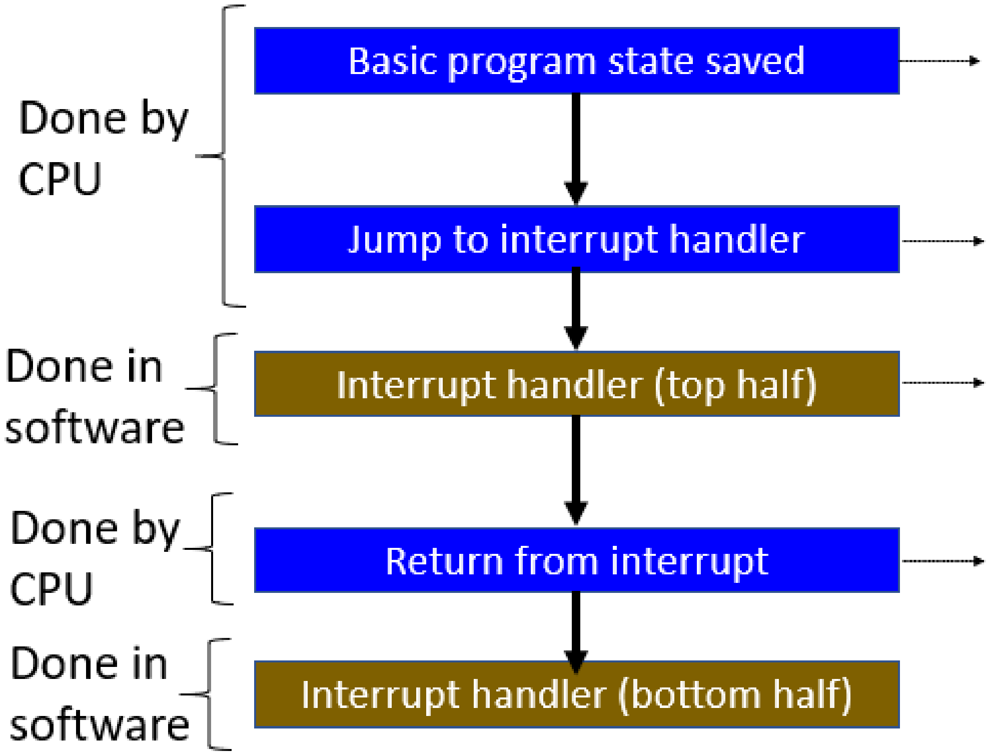
Outline

- Interrupt
- Hardware Interrupt
- Interrupt Workflow
- Software Interrupt -- Exception
- Interrupt Vector
- Interrupt Descriptor Table
- Interrupt Stack Table

Interrupt Workflow (1/3)


By device and APICs
Device asserts IRQ of I/O APIC
I/O APIC transfer interrupt to LAPIC
Either special 3 wire APIC bus or system bus
LAPIC asserts CPU interrupts
Done By CPU
After current instruction completes CPU senses interrupt line and obtain IRQ number from LAPIC
Switch to kernel stack if necessary

Interrupt Workflow (2/3)


Done by CPU
Basic program state saved
Jump to interrupt handler
Done in software
Interrupt handler (top half)
Done by CPU
Return from interrupt
Done in software
Interrupt handler (bottom half)

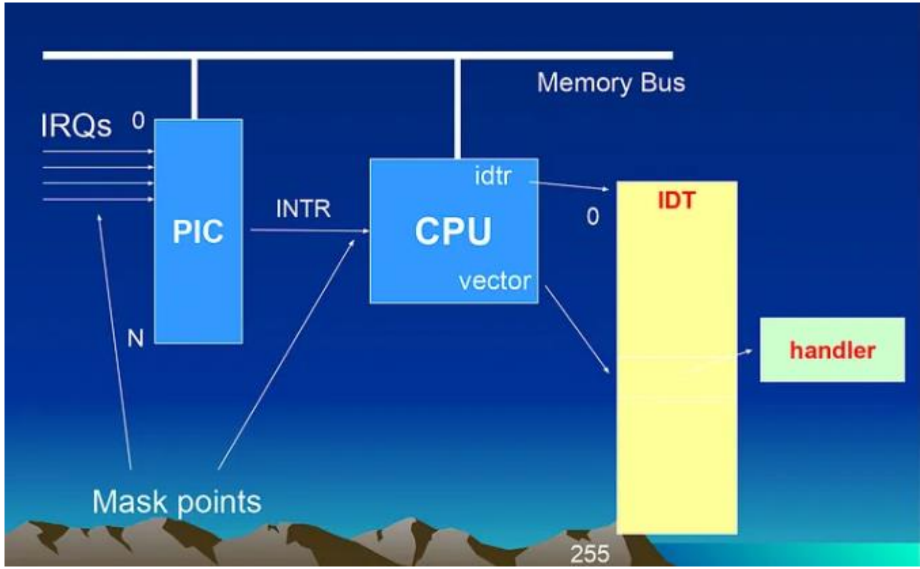
x86 saves the SS, ESP, EFLAGS, CS, EIP, error code on stack
(restored by **iret** instruction)
suspends current task

How does hardware find the OS interrupt handler ?

1. Respond to interrupt
2. More storing of program state
3. Schedule bottom half
4. IRET

Restore flags and registers saved earlier. Restore running task

Interrupt Workflow (3/3)



- Processing Interrupt
  - Device creates IRQ
  - PIC collects IRQs
  - PIC prioritizes IRQs
  - PIC issues interrupt to CPU
  - CPU saves interrupt states
  - CPU asks PIC interrupt number
  - CPU uses an interrupt vector number as an index to find IDT entry
  - ISR saves states in registers
  - Executing ISR
  - After completing, pass the EOI (End of Interrupt) command
  - Resume registers/iret command

Outline

- Interrupt
- Hardware Interrupt
- Interrupt Workflow
- Software Interrupt -- Exception
- Interrupt Vector
- Interrupt Descriptor Table
- Interrupt Stack Table

Software Interrupt

- **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

Outline

Interrupt Vector (1/3)

- **Interrupt vector**
  - The processor uses a unique number for recognizing the type of interruption or exception
  - Each interrupt/exception provided a number
  - Number used to index into an **interrupt descriptor table (IDT)**
  - IDT provides the entry point into an interrupt/exception handler
  - 0 to 255 vectors possible

Interrupt Vector (2/3)

- **Interrupt vector**
  - **0 to 31** correspond to **exception and nonmaskable interrupts (NMI,** handle non-recoverable error**)**
  - **32 – 47** are assigned to **maskable interrupts caused by IRQs**
  - **48 – 255** may be used to identify **software interrupts**
  - For example, Linux uses a 128 (0x80) vector that is used to make system calls to the kernel by other programs.
  - When a process in user mode executes int 0x80 assembly instruction, the CPU switches into kernel mode and starts executing the system_call() kernel function

Interrupt Vector (3/3)

- **Processor generates exception**

INT_NUM	Short description
0x00	Division by zero
0x01	Single-step interrupt
0x02	NMI
0x03	Breakpoint
0x04	Overflow
0x05	Bound range exceeded
0x06	Invalid opcode

Outline

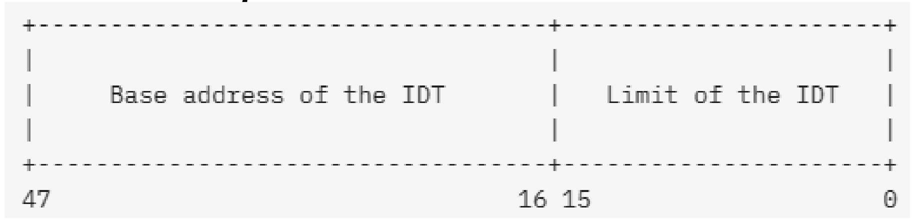
- Interrupt
- Hardware Interrupt
- Interrupt Workflow
- Software Interrupt -- Exception
- Interrupt Vector
- Interrupt Descriptor Table
- Interrupt Stack Table

Interrupt Descriptor Table (IDT) (1/3)

- **Interrupt descriptor table**
  - Stores entry points of the interrupts and exceptions handlers
  - The IDT entries are called gates
    - Interrupt gates
    - Task gates
    - Trap gates
  - The IDT is an array of 8-byte gates (256 entries) on x86 and 16-byte gates on x86_64
  - Loaded the IDT with the null gates while transitioning into protected mode

Interrupt Descriptor Table (IDT) (2/3)

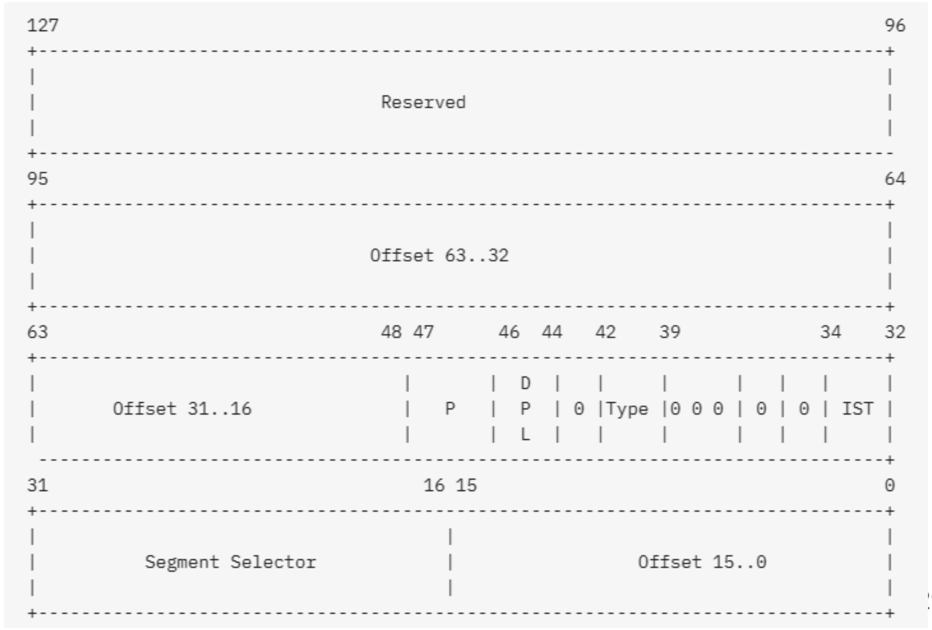
- **Interrupt descriptor table**
  - Can be located anywhere in the linear address space
  - The base address of it must be aligned on an 8-byte boundary on x86, a 16-byte boundary on x86_64
  - The base address of IDT is store in IDTR register
    - LIDT/SIDT instruction to read/write IDTR register
    - The IDTR register is 48-bits on the x86



```
+-----------------------------------+----------------------+
|                                   |                      |
|      Base address of the IDT      |   Limit of the IDT   |
|                                   |                      |
+-----------------------------------+----------------------+
47                                16 15                    0
```

Interrupt Descriptor Table (IDT) (3/3)

- **The IDT entries** (16 bytes on x86_64)
  - **0-15 bits** as the base address of entry point of the interrupt handler
  - **16-31 bits** as the base address of the segment selector
  - DPL (Descriptor Privilege Level)



```
127                                                                    96
+-----------------------------------------------------------------------+
|                                                                       |
|                               Reserved                                |
|                                                                       |
+-----------------------------------------------------------------------
95                                                                     64
+-----------------------------------------------------------------------+
|                                                                       |
|                              Offset 63..32                            |
|                                                                       |
+-----------------------------------------------------------------------+
63                             48 47      46  44   42    39          34    32
+-----------------------------------------------------------------------+
|                              |       | D |   |     |      |   |   |     |
|      Offset 31..16           |   P   | P | 0 |Type |0 0 0 | 0 | 0 | IST |
|                              |       | L |   |     |      |   |   |     |
 -----------------------------------------------------------------------+
31                                  16 15                               0
+-----------------------------------------------------------------------+
|                                  |                                    |
|          Segment Selector        |             Offset 15..0           |
|                                  |                                    |
+-----------------------------------------------------------------------+
```

Outline

- Interrupt
- Hardware Interrupt
- Interrupt Workflow
- Software Interrupt -- Exception
- Interrupt Vector
- Interrupt Descriptor Table
- Interrupt Stack Table

Interrupt Stack Table (IST) (1/3)

- **Interrupt stack table**
  - New mechanism in x86_64
  - An alternative to legacy stack-switch mechanism
  - Unconditionally switches stacks when it is enabled and can be enabled for any interrupt
  - Seven IST pointers in the task state segment (TSS)
    - TSS contains information about a process
    - TSS is used for stack switching during the execution of an interrupt or exception handler

Interrupt Stack Table (IST) (2/3)

- **Stack switching**
  - If the interrupt occurs when running in the user mode
    - The process switches from user stack to kernel stack
    - Then, switching to the interrupt stack
  - How to switch stack?
    - CPU should know the location of the new stack segment (SS) and ESP register
    - Done by task segment descriptor

Interrupt Stack Table (IST) (3/3)

- **Task state segment (TSS)**
  - TSS is used to find the new stack
  - TSS resides in the memory
    - Processor register states -> used for task switching
    - I/O port permission bitmap -> specifies individual ports to accessible program
    - Inner-level stack pointer -> specifies the new stack pointer when a privilege level change occurs
    - Previous TSS link

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

- How does the device identify itself to the process?
  - (A) By sending its device ID
  - (B) By sending the machine code for the ISR
  - (C) By sending the starting address of the service routine
- Which table stores the address of the interrupt handling sub-routines?
  - (A) interrupt-vector table
  - (B) Symbol link table
  - (C) Interrupt stack table