北京大学
PEKING UNIVERSITY

10010010

嵌入式系统编程与实践

4-中断

燕博南

2024秋

Interrupts

How peripherals notify the CPU that their state just changed.

Example: A button just pressed

Interrupts

- Definition
  - An event external to the currently executing process that causes a change in the normal flow of instruction execution; usually generated by hardware devices external to the CPU.
  - Key point is that interrupts are asynchronous w.r.t. current process
  - Typically indicate that some device needs service

Why interrupts?

- MCUs have many external peripherals
  - Keyboard, mouse, screen, disk drives, scanner, printer, sound card, camera, etc.
  - These devices occasionally need CPU service
    - But we can't predict when
  - We want to keep the CPU busy (or asleep) between events
  - Need a way for CPU to find out devices need attention

Possible Solution: Polling

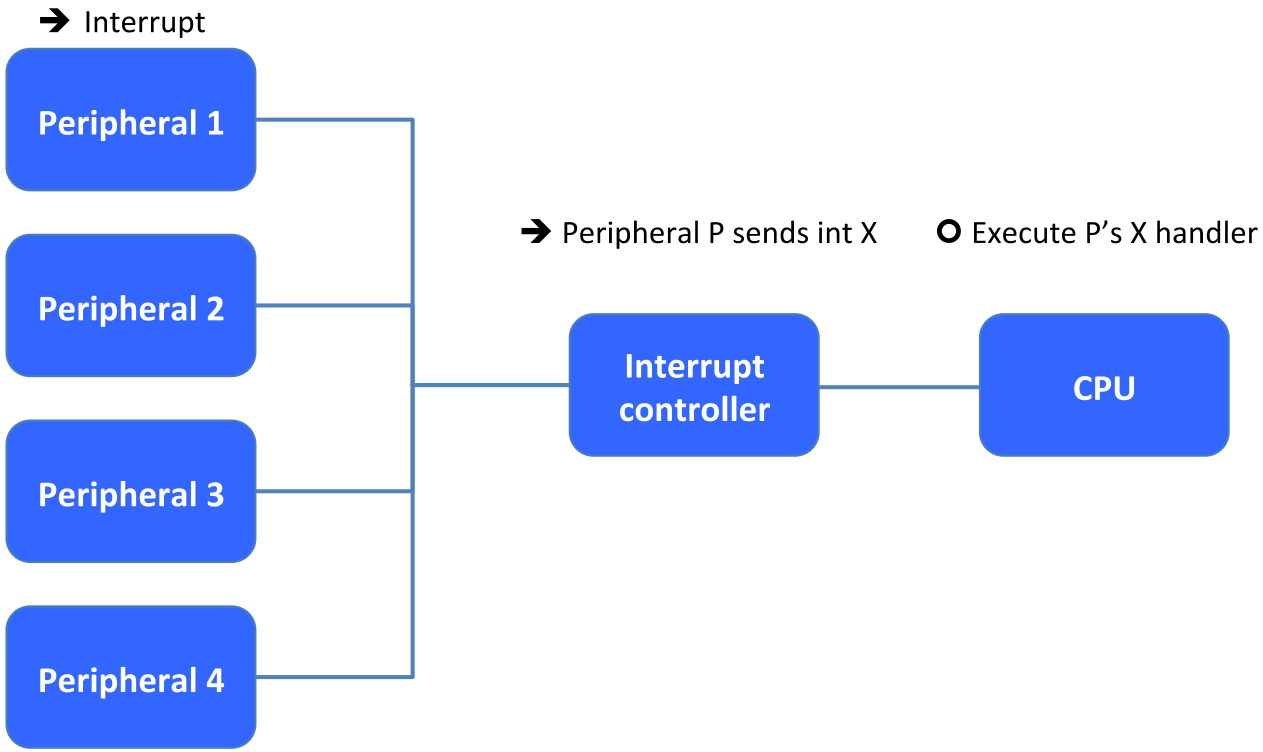
- CPU periodically checks each device to see if it needs service
  - "Polling is like picking up your phone every few seconds to see if you have a call. …"

Possible Solution: Polling

- CPU periodically checks each device to see if it needs service
  - "Polling is like picking up your phone every few seconds to see if you have a call. …"
  - Cons: takes CPU time even when no requests pending
  - Pros: can be efficient if events arrive rapidly

Alternative: Interrupts

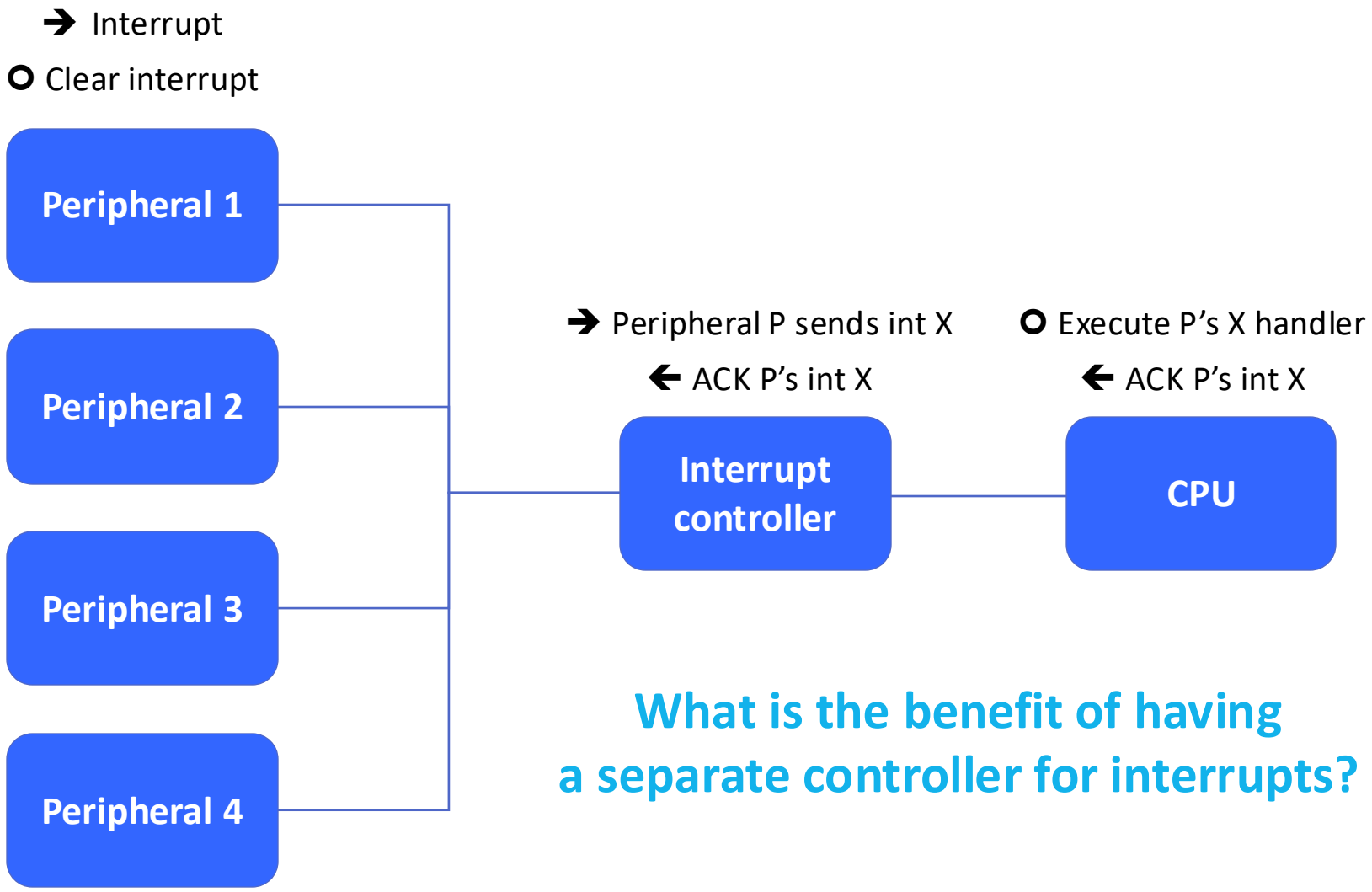
- Give each device a wire (interrupt line) that it can use to signal the processor



Alternative: Interrupts

- Give each device a wire (interrupt line) that it can use to signal the processor
  - When interrupt signaled, processor executes a routine called an interrupt handler to deal with the interrupt
  - No overhead when no requests pending

How do interrupts work?

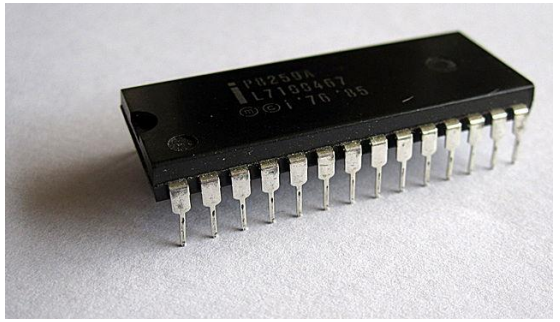


What is the benefit of having a separate controller for interrupts?

The Interrupt controller

- **Handles simultaneous interrupts**
  - Receives interrupts while the CPU handles interrupts
- **Maintains interrupt flags**
  - CPU can poll interrupt flags instead of jumping to a interrupt handler
- **Multiplexes many wires to few wires**
  - CPU doesn't need a interrupt wire to each peripheral

Fun fact: Interrupt controllers used to be separate chips!



Intel 8259A IRQ chip

Image by Nixdorf - Own work

How to use interrupts

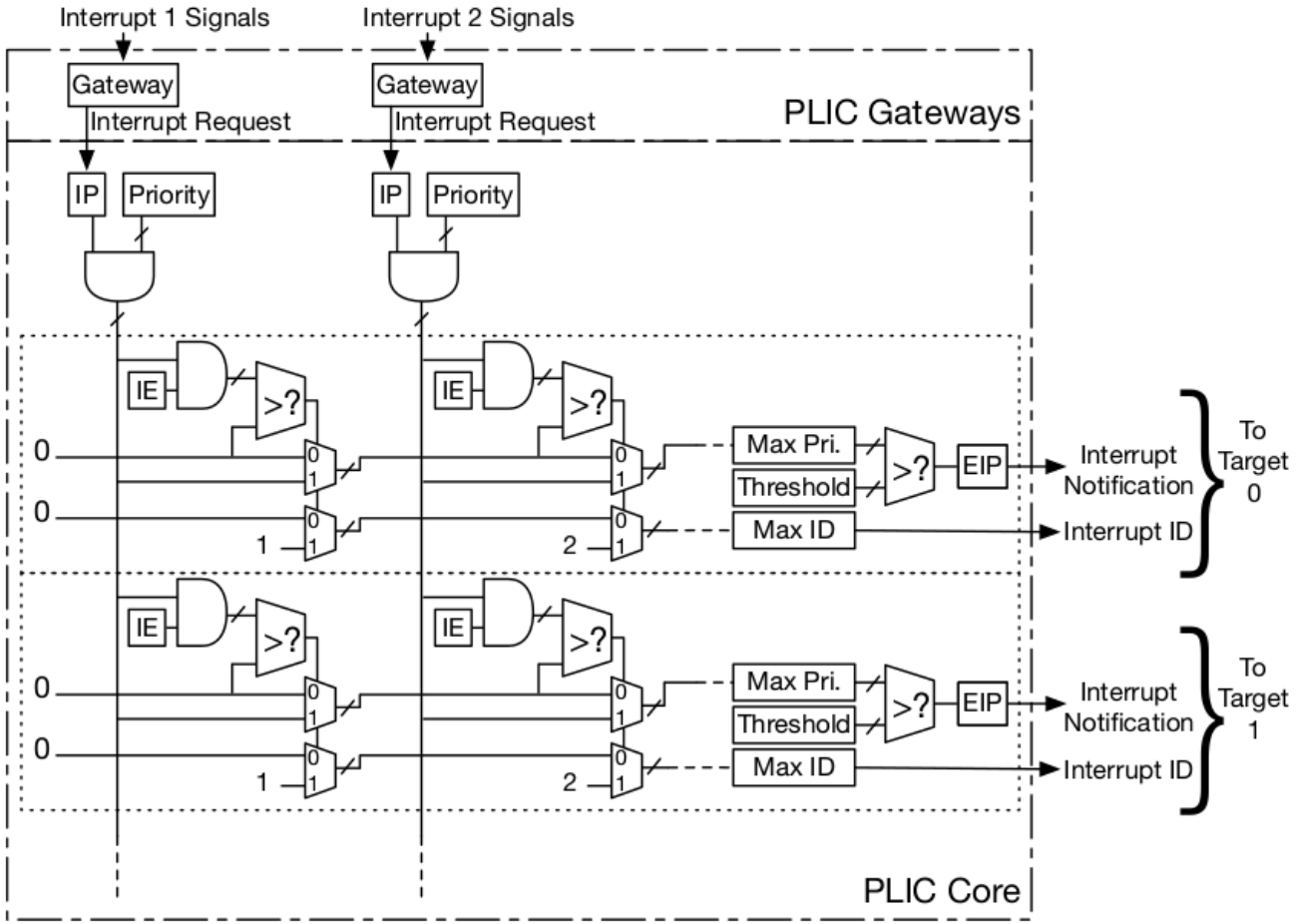
1. Tell the peripheral which interrupts you want it to output.
2. Tell the interrupt controller what your priority is for this interrupt.
3. Tell the processor where the interrupt handler is for that interrupt.
4. When the interrupt handler fires, do your business then clear the int.

CPU execution of interrupt handlers

INTERRUPT

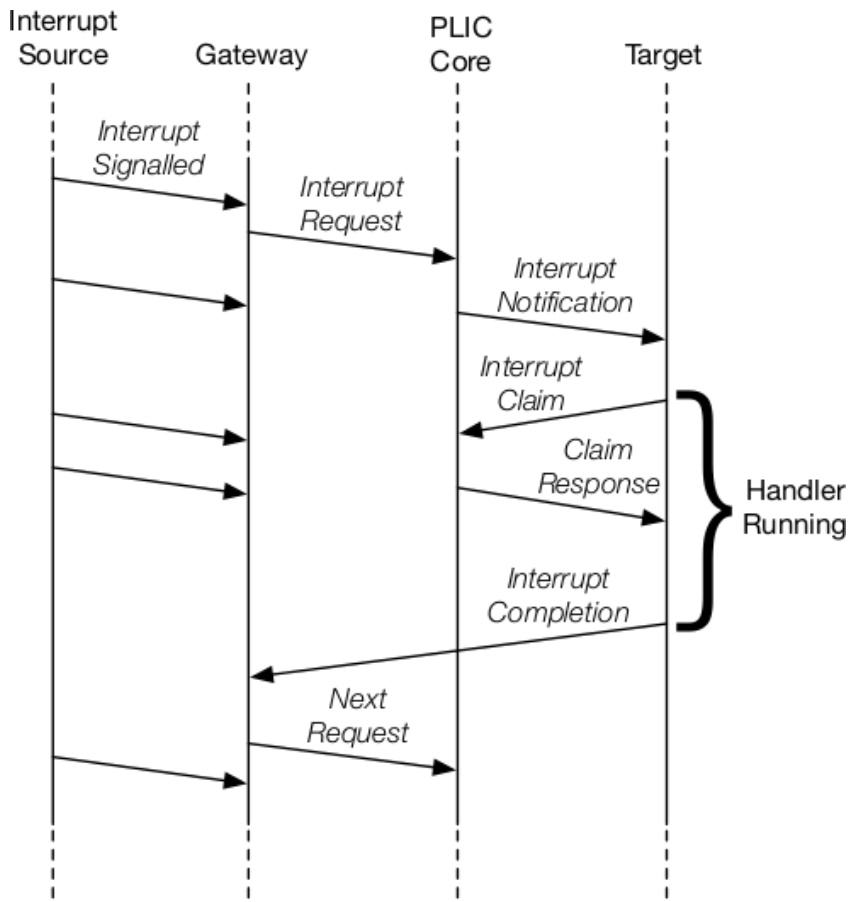
1. Wait for instruction to end
2. Push the program counter to the stack
3. Push all active registers to the stack
4. Jump to the interrupt handler in the interrupt vector
5. Pop the program counter off of the stack

RISC-V Platform-Level Interrupt Controller (PLIC)



- **Interrupt Gateways**
  - convert global interrupt signals into a common interrupt request format
  - control the flow of interrupt requests to the PLIC core
- **Interrupt Identifiers (IDs)**
  - Identify interrupt
- **Interrupt Enables (IE)**
  - stored in a register, 1 bit for each source
- **Interrupt Notifications**
  - Notify CPU

Interrupt Flow



- Interrupt Claim:
  - After receiving an interrupt notification, CPU decide to service the interrupt.
  - CPU sends an *interrupt claim* message to the PLIC core.
  - On receiving a claim message, the PLIC core will atomically determine the ID of the highest-priority pending interrupt for the target and then clear down the corresponding source's IP bit.
  - The PLIC core will then return the ID to CPU.

The End