Programmable Interrupt Controller

Table of Contents

[Maintaining Priority without Interrupts 2](#_Toc106753094)

[Priority Arbiters 2](#_Toc106753095)

[Daisy Chain Arbitration 3](#_Toc106753096)

[Multi-Level Bus Architecture 3](#_Toc106753097)

[Interrupts 4](#_Toc106753098)

[8259 PIC 4](#_Toc106753099)

[Block Diagram 6](#_Toc106753100)

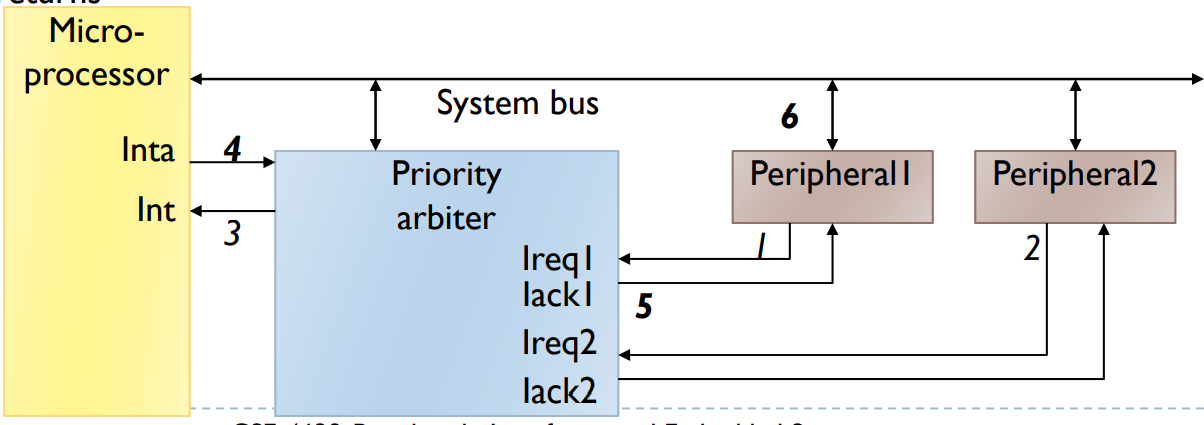
[Programming 7](#_Toc106753101)

We know that peripheral devices are connected to microprocessors using interfaces. The **programmable interrupt controller** (PIC) is one such interface that deals with **interrupts**.

## Maintaining Priority without Interrupts

Before the invention of interrupts, requests from multiple devices were handled in several different ways.

### Priority Arbiters

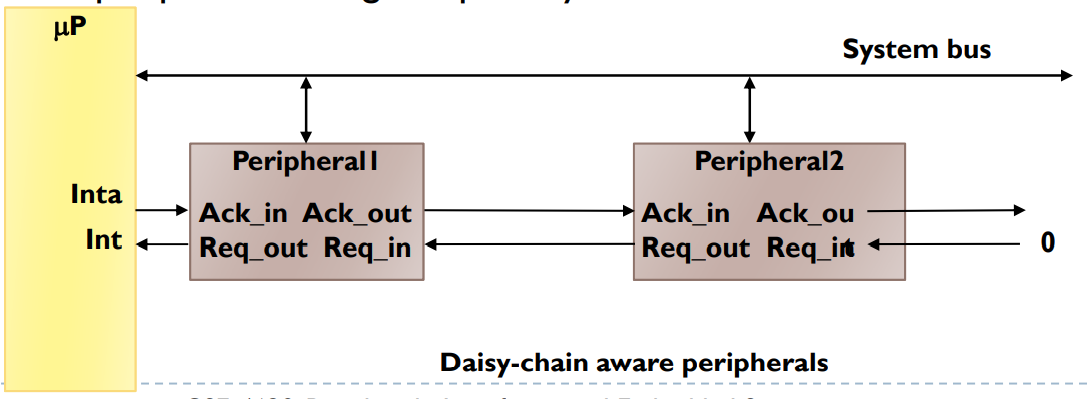


The **priority arbiter** has multiple devices connected to it. If several devices requested servicing, the priority arbiter gave priority to the device with the lowest ID and informed the microprocessor about just that device.

This is a bad solution since the actual **request** should define the priority, not the device. We can have a situation where a high priority device is giving a less important request than a low priority device. This can also lead to **starvation**.

### Daisy Chain Arbitration

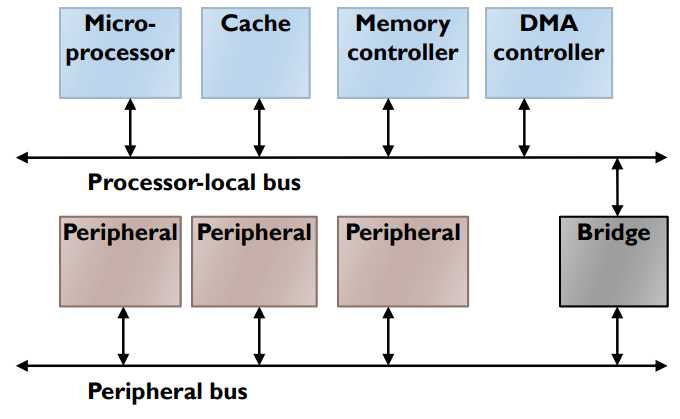
In **daisy chain arbitration**, several devices are connected one after the other. Requests from further devices must go through the nearer devices to reach the microprocessor.



In this case, the closest device always wins. This is still not allowing us to prioritize requests, but rather devices. Additionally, there is a **single point of failure**, since if one connection goes down, we lose all the devices from that point forwards.

### Multi-Level Bus Architecture

Peripheral devices are very slow. Having a bunch of peripheral devices on the bus slows down all the important devices. The **multi-level bus architecture** places peripheral devices on a separate line and places a **bridge** in between to control which device communicates with the main bus.



In this case as well, we have a **single point of failure**.

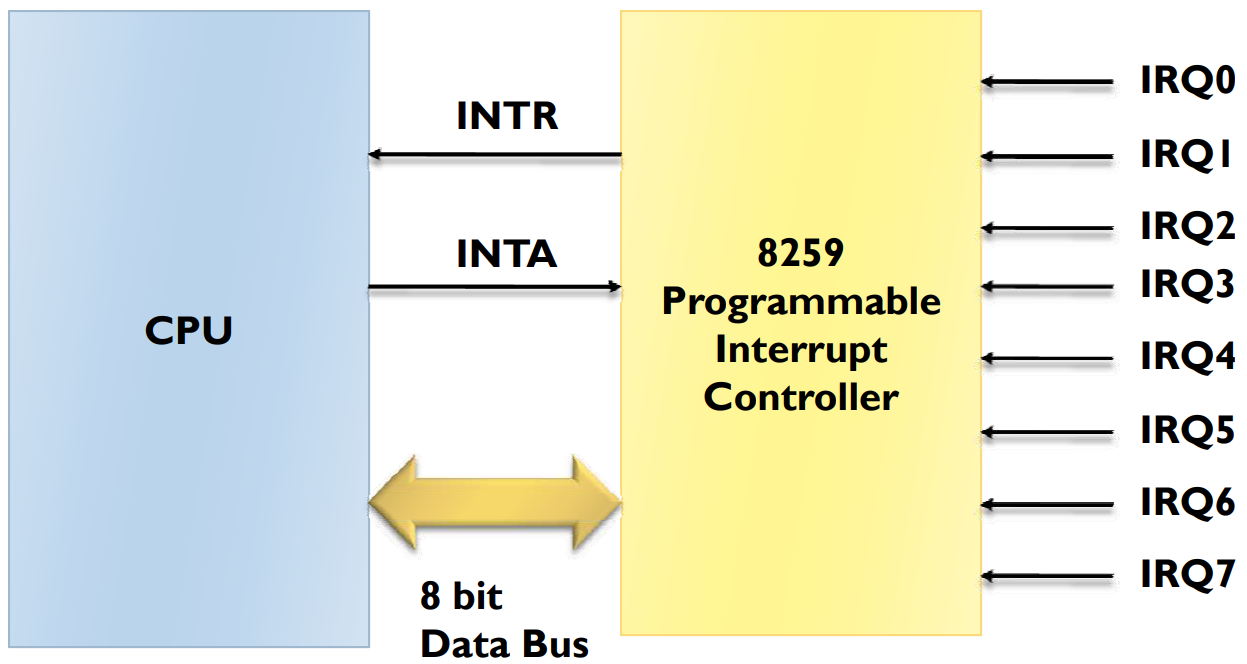
## Interrupts

As a reminder, interrupts halt the current program flow and execute an **Interrupt Service Routine** (ISR). When the ISR completes, the previous program flow resumes. This process allows us to efficiently handle unanticipated events.

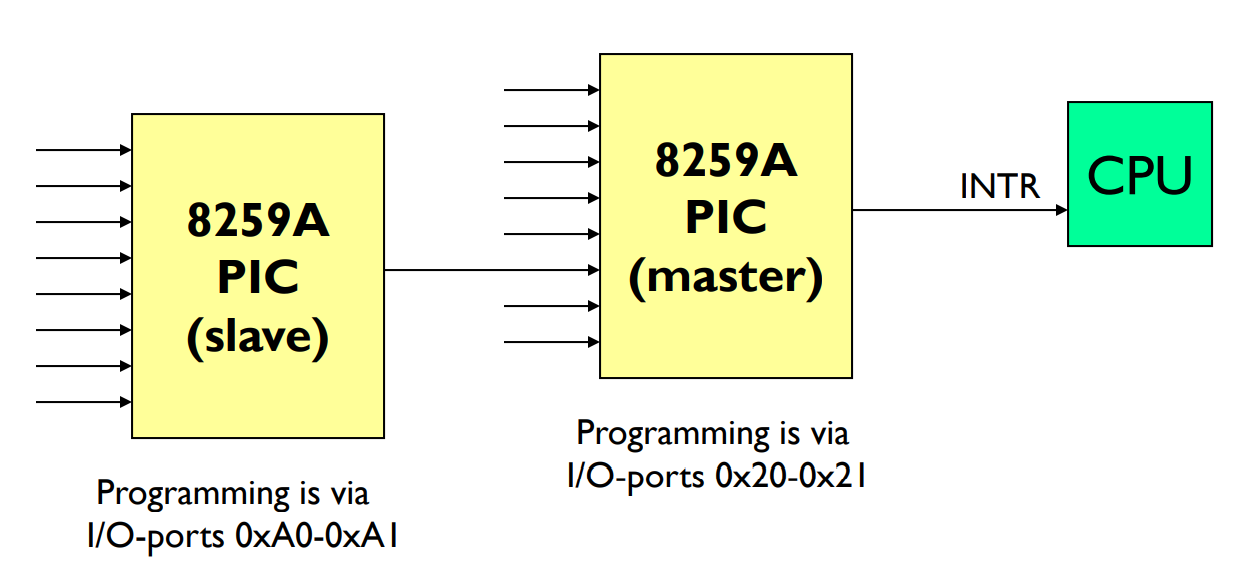
When an interrupt occurs, the current instruction is finished, the flag register, IP and CS register values are pushed onto the stack, the address of the required ISR is read and the ISR is executed. Upon completion of the ISR, the flag register, IP and CS register values are popped and the previous program resumes. The ISR address is found at , where is the **interrupt number**.

## 8259 PIC

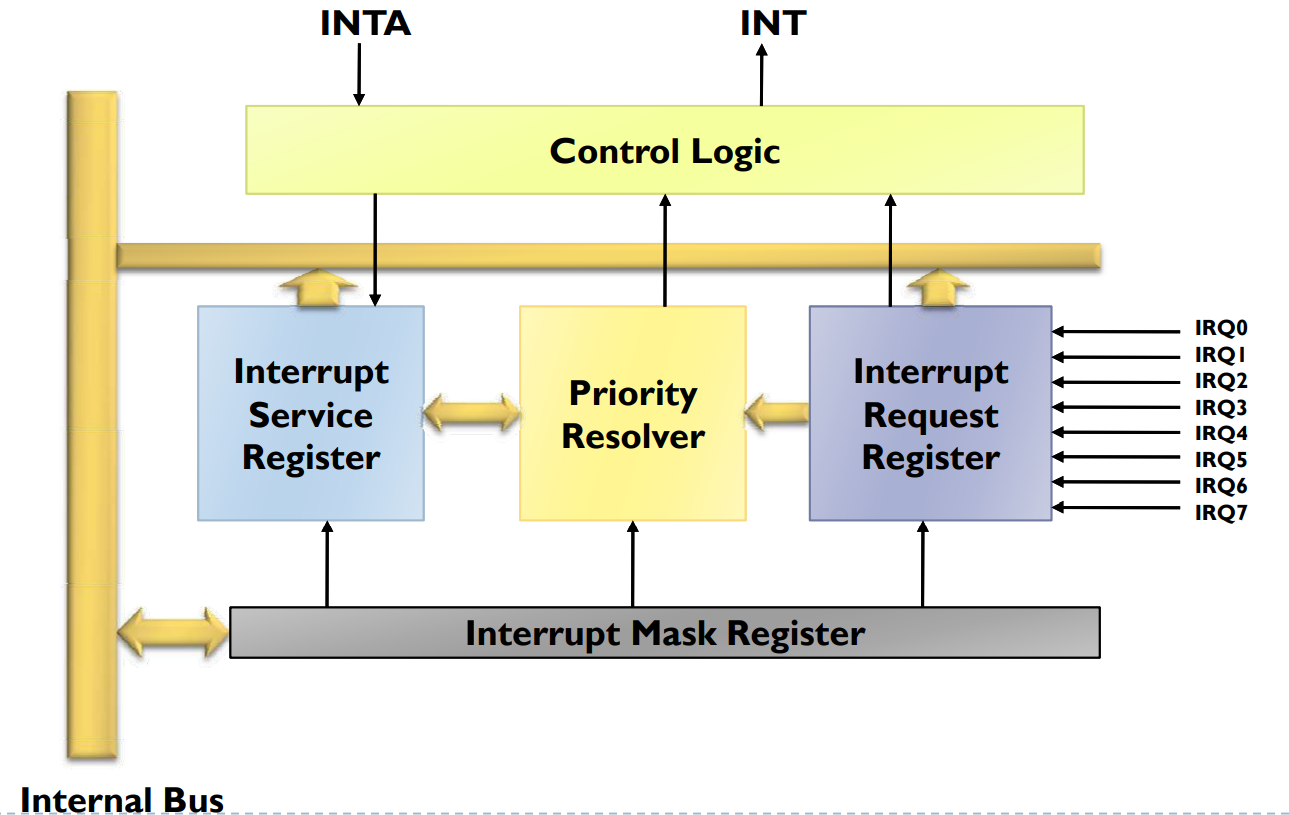
The **8259 PIC** is a **separate IC** which has **8 pins** to connect devices to. The IC in turn connects to the INTR, INTA and data bus of the MPU. The PIC can be programmed in various ways to ensure different priority configurations.



Instead of connecting 8 devices to the 8 pins, we can connected **8 8259 PICs**. This allows us to expand the total number of devices to 64.



## Block Diagram

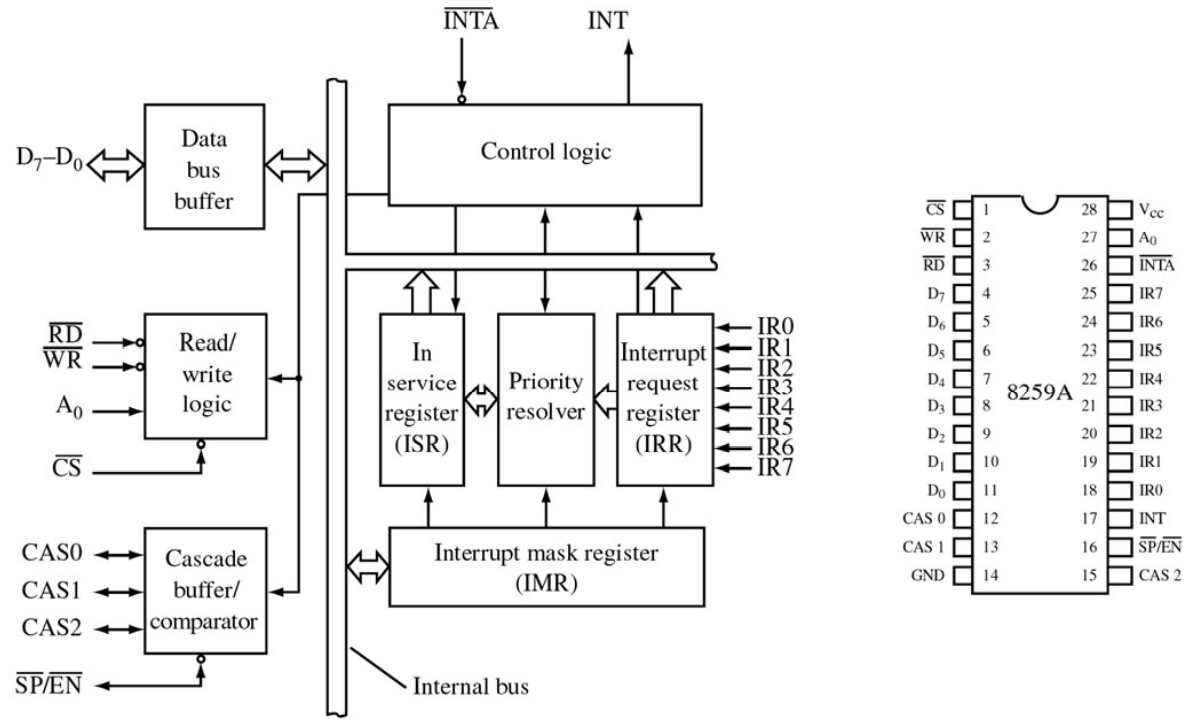


The **Control Logic** block is where the logic for the priority levels is stored.

The **Interrupt Request Register** accepts all interrupt requests. Based on the control logic, the **Priority Resolver** figured out which interrupt has the highest priority. Finally, the **Interrupt Service Register** passes on the interrupt which was chosen.

Interrupt types that just got serviced are masked by the **Interrupt Mask Register**. This ensures that they have the lowest priority.

The above diagram is a **simplified version**. The actual diagram is given below:



## Programming

The 8259 PIC has two modes, the initialization mode and the operation mode.

In the **initialization mode**, a series of **9 bit values** is sent to the PIC. These values are called **Initialization Command Words** (ICWs). The MSB is sent via bit 0 of the address bus, while the other 8 bits are sent via the data bus.

After the ICWs are programmed into the 8259 PIC, the chip is ready to accept interrupt requests at its input lines. The 8259 can be operated in various modes by providing 9 bit commands. We can even read a **return byte** if required.

The possible operation modes are:

* **Fully Nested Mode** – This is for when there is another interrupt nested inside an ISR.
* **Rotating Priority Mode** – All interrupt types have the same priority. The interrupts are given priority in a round robin fashion. This is not always the best policy.
* **Polled Mode** – In the rotating priority mode, the interrupt accesses the MPU. In the polled mode, the MPU checks on the interrupt instead.
* **Special Mask Mode** – This mode is used to deal with NMIs, which are high-priority interrupts which cannot be rejected.