Programmable Interrupt Controller (8259 PIC)

Course Teacher:

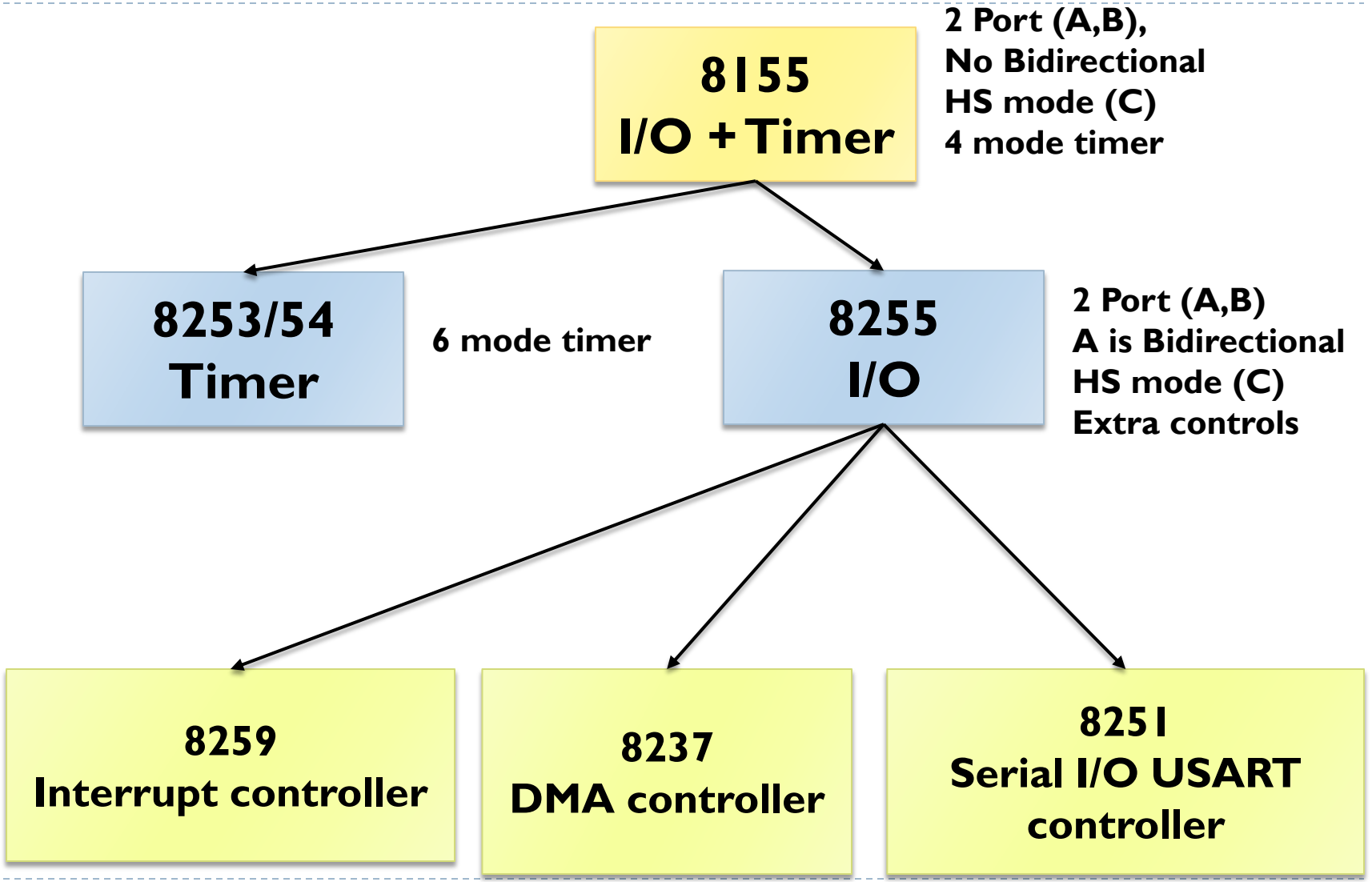
Md. Obaidur Rahman, Ph.D.
Professor
Department of Computer Science and Engineering (CSE),
Dhaka University of Engineering & Technology (DUET), Gazipur.

Course ID: CSE - 4619
Course Title: Peripherals, Interfacing and Embedded Systems
Department of Computer Science and Engineering (CSE),
Islamic University of Technology (IUT), Gazipur.

Lecture References:

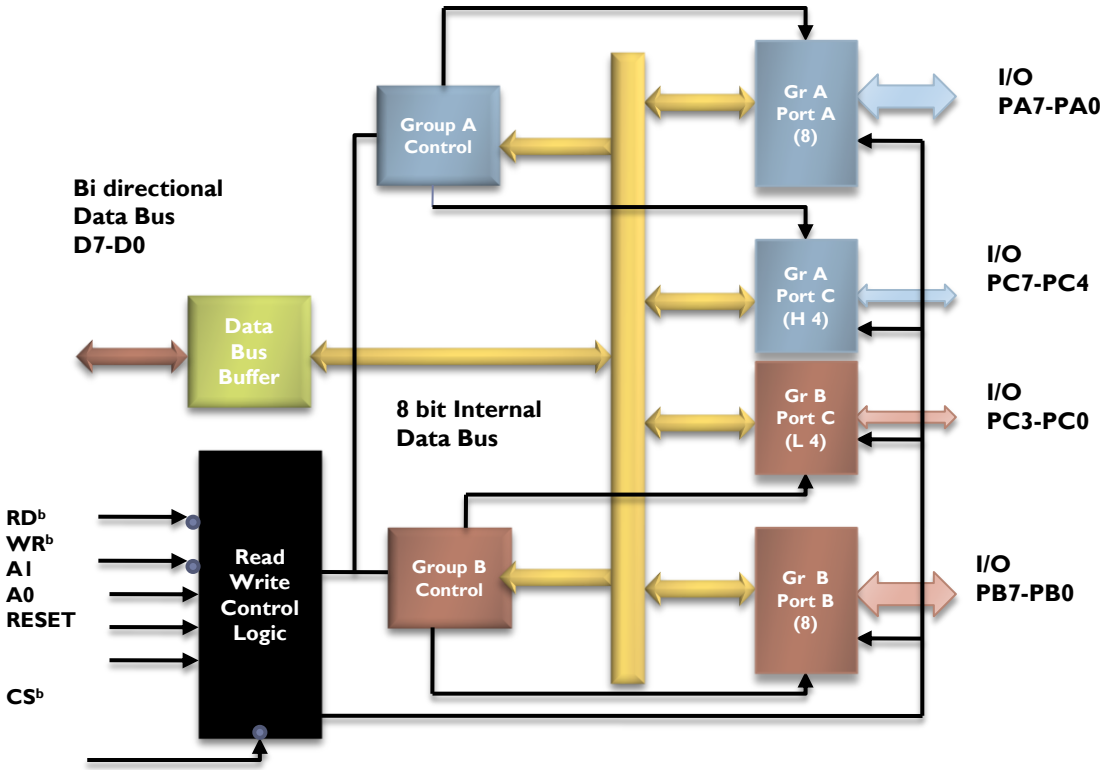
- Book:
  - *Microprocessor Architecture, Programming and Applications with 8085 (Chapter-15)*, **Author:** Ramesh Gaonkor
- Lecture Materials:
  - *I/O System Design*, Dr. Esam Al_Qaralleh, CE Department, Princess Sumaya University for Technology.

Hierarchy of I/O Control Devices



Number of I/O Devices connected through 8255 PPI

- Two Port A & Port B as Input or Output Port
- Port C as Handshaking/Interrupt port
- Both can work Simultaneously

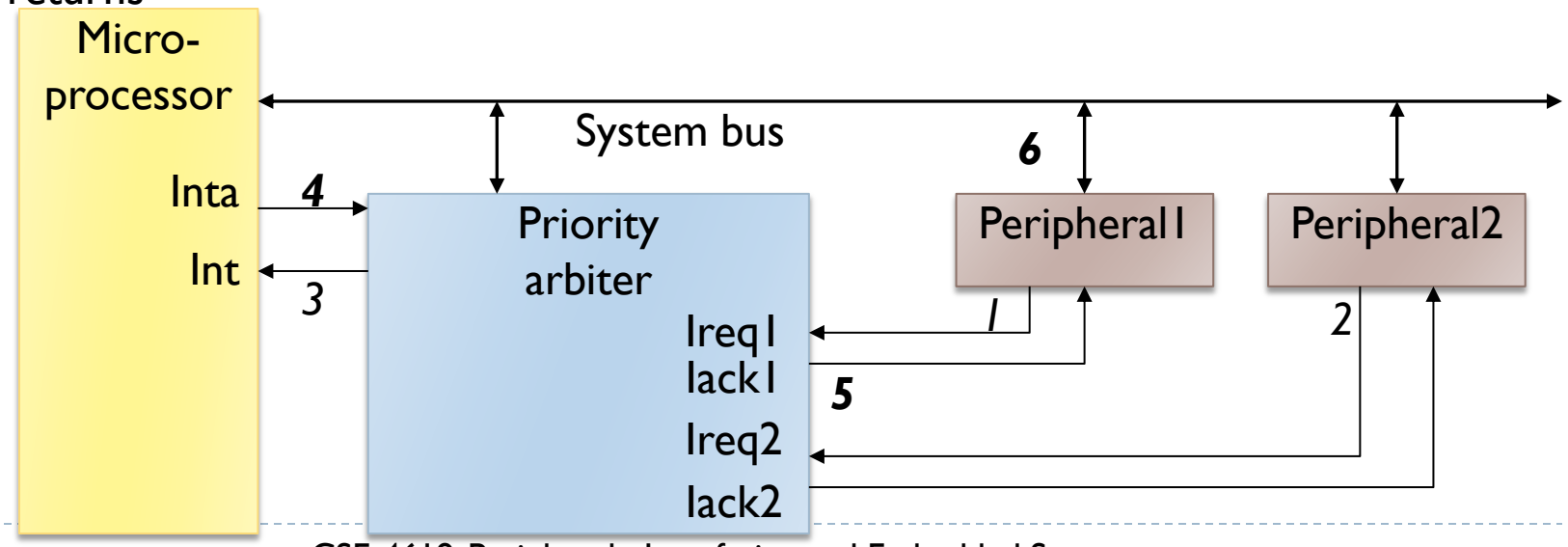


How to Connect Multiple I/O Devices?

- There are two possible ways:
  - Use priority resolver to connect them
  - Use **Interrupt** as generalized mechanism to connect

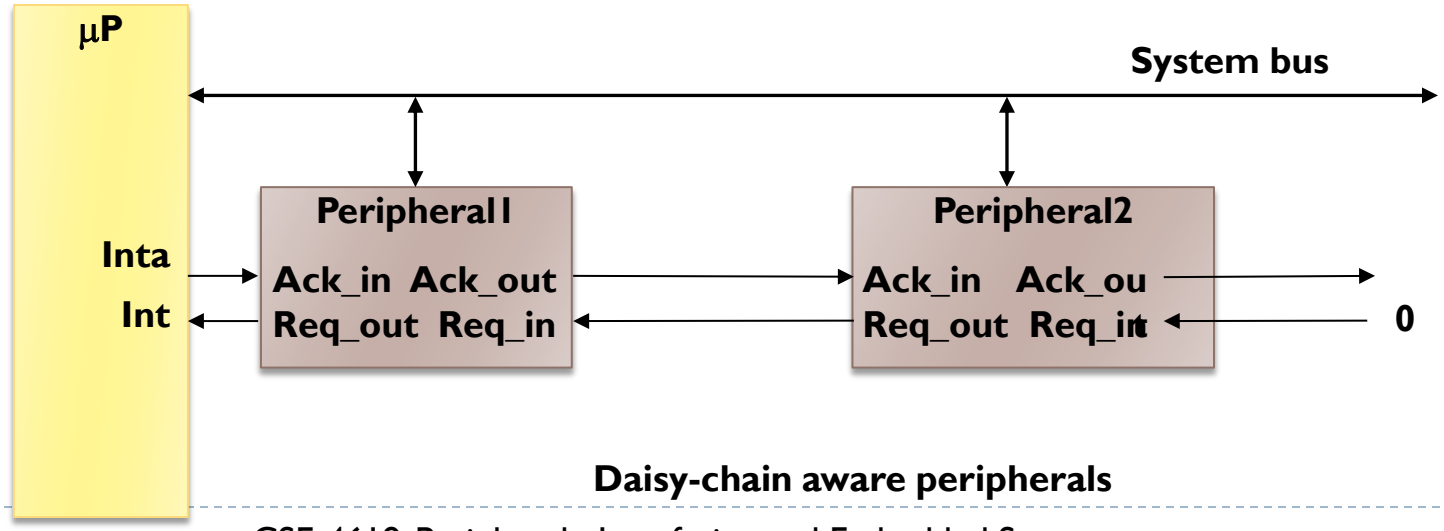
Arbitration using a Priority Arbiter

- Peripheral1 needs servicing so asserts *Ireq1*. Peripheral2 also needs servicing so asserts *Ireq2*.
- Priority arbiter sees at least one *Ireq* input asserted, so asserts *Int*.
- Microprocessor stops executing its program and stores its state.
- Microprocessor asserts *Inta*. Priority arbiter asserts *lack1* to acknowledge Peripheral1.
- Peripheral1 puts its interrupt address vector on the system bus
- Microprocessor jumps to the address of ISR read from data bus, ISR executes and returns



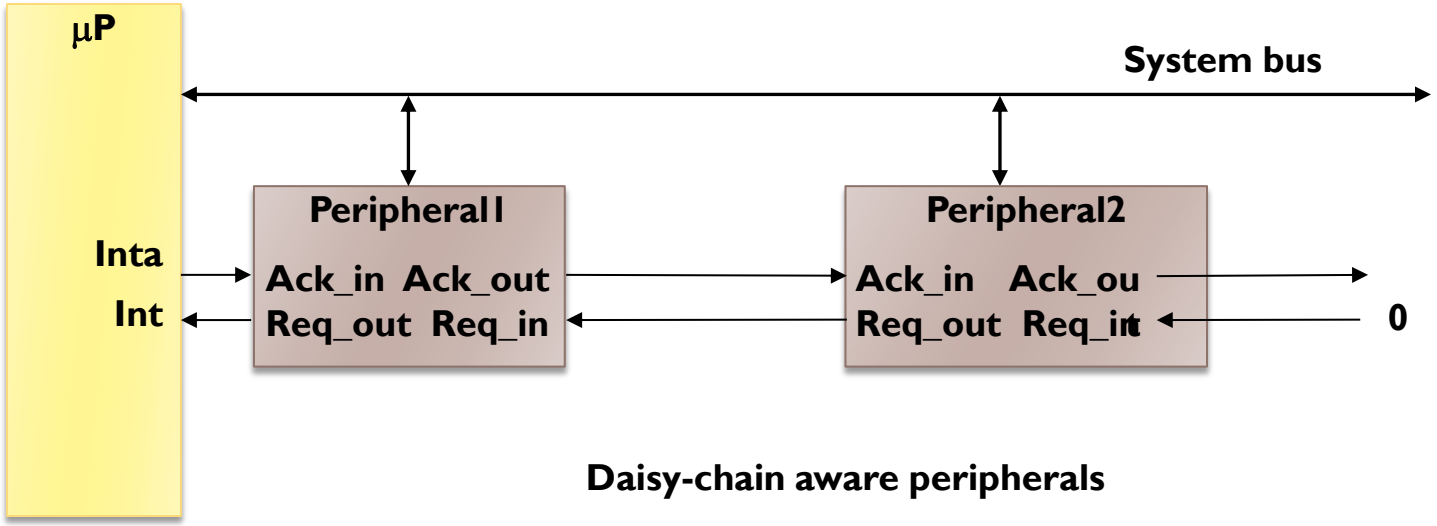
Daisy-Chain Arbitration

- Arbitration done by peripherals
  - Built into peripheral or external logic added
    - *req* input and *ack* output added to each peripheral
- Peripherals connected to each other in daisy-chain manner
  - One peripheral connected to resource, all others connected "upstream"
  - Peripheral's *req* flows "downstream" to resource, resource's *ack* flows "upstream" to requesting peripheral
  - Closest peripheral has highest priority



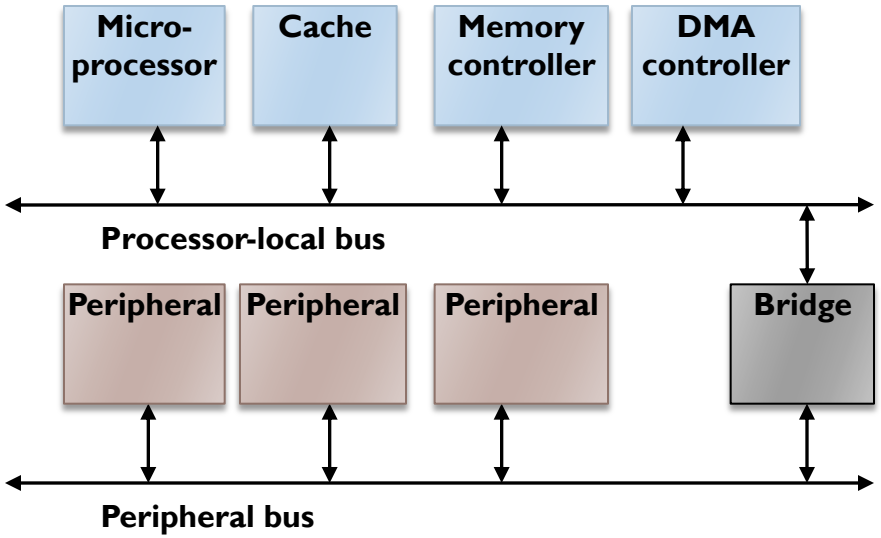
Daisy-Chain Arbitration

- **Pros and Cons**
  - Easy to add/remove peripheral - no system redesign needed
  - Does not support rotating priority
  - One broken peripheral can cause loss of access to other peripherals



Multilevel Bus Architectures

- Don't want one bus for all communication
  - Peripherals would need high-speed, processor-specific bus interface
  - Too many peripherals slows down bus
- **Processor-local bus**
  - High speed, wide, most frequent communication
  - Connects microprocessor, cache, memory controllers, etc.
- **Peripheral bus**
  - Lower speed, narrower, less frequent communication
  - Typically industry standard bus (ISA, PCI) for portability



- **Bridge**
  - Single-purpose processor converts communication between buses

Revision: Interrupt Concept

- Interrupts alter a program's flow of control
  - Behavior is similar to a procedure call
    - Some significant differences between the two
- Interrupt causes transfer of control to an *interrupt service routine* (ISR)
  - ISR is also called a *handler*
- When the ISR is completed, the original program resumes execution
- **Interrupts are used to interface I/Os**
- Interrupts provide an efficient way to handle unanticipated events

Revision: Basic Procedure for Interrupts

- When an interrupt is executed, the *μp*:
  - finishes executing its current instruction (if any).
  - saves (PUSH) the flag register, IP and CS register in the stack.
  - goes to a fixed memory location.
  - reads the address of the associated ISR.
  - Jumps to that address and executes the ISR.
  - gets (PULL) the flag register, CS:IP register from the stack.
  - continues executing the previous job (if any).

Revision: 8088/86 Hardware Interrupts pins

- INTR: Interrupt Request.
  - Input signal into the CPU
  - If it is activated, the CPU will finish the current instruction and respond with the interrupt acknowledge operation
  - Can be masked (ignored) through CLI or STI
- NMI: Non-Maskable interrupt.
  - Input signal
  - Cannot be masked or unmasked through CLI or STI
  - Examples of use: power frailer. Memory error
- INTA: Interrupt Acknowledge.
  - Output signal

Revision: The Interrupt flag

- IF (Interrupt Enable Flag) D9: used to mask any hardware interrupt that may come in from the INTR pin.
- When IF=0, all hardware interrupt requests through INTR are masked.
- This has no effect on interrupts coming from the NMI pin or "INT nn" instructions.
- CLI sets IF to 0, STI sets IF to 1.

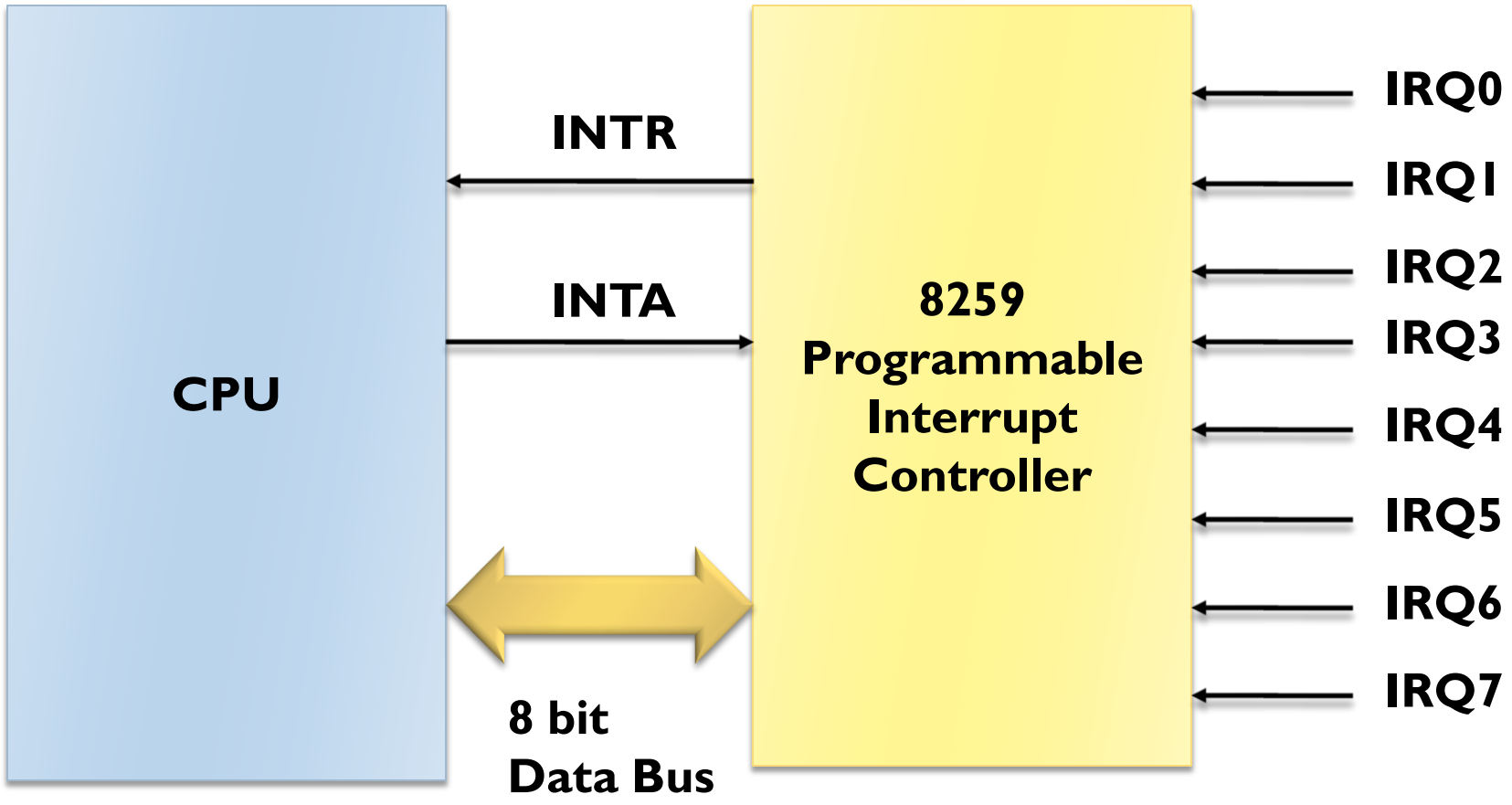
Revision: INT *n* and ISR

- *n* is multiplied by 4
- In the address "*4n*" the offset address the ISR is found.
- Example: Intel has set aside INT 2 for the NMI interrupt.
- Whenever the NMI pin is activated, the CPU jumps to physical memory location 00008 to fetch the CS:IP of the interrupt service routine associated with the NMI.

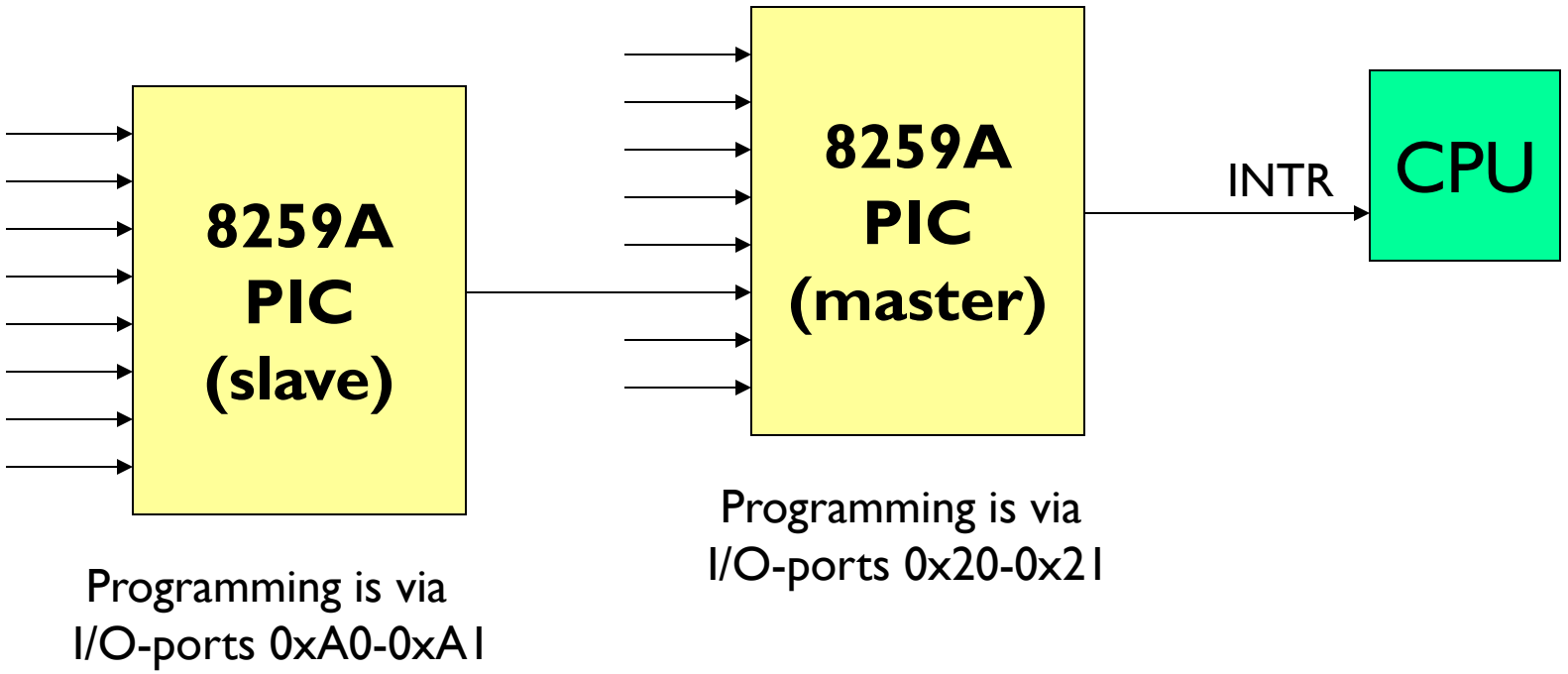
8259 Programmable Interrupt Controller (PIC)

- It is a tool for managing the interrupt requests.
- 8259 is a very flexible peripheral controller chip:
  - PIC manages 8 (eight) levels of requests and has built-in features for expandability to other PIC (up to 64 levels)
  - Interrupts can be masked
  - Various priority schemes can also programmed.
- Originally in PIC it is available as a separate IC
- Later the functionality of *(two PICs)* is in the motherboards chipset.
- In some of the modern processors, the functionality of the *PIC* is built in the MP.

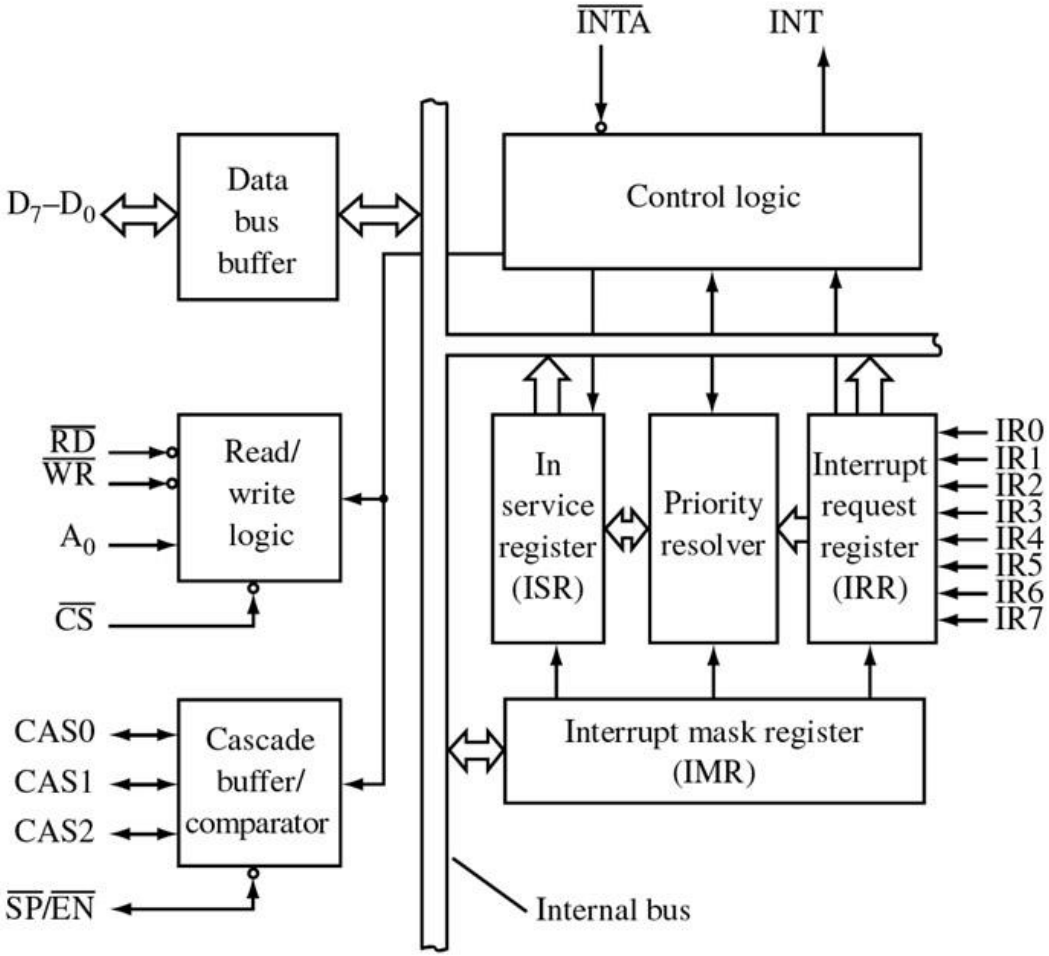
8259 Programmable Interrupt Controller (PIC)

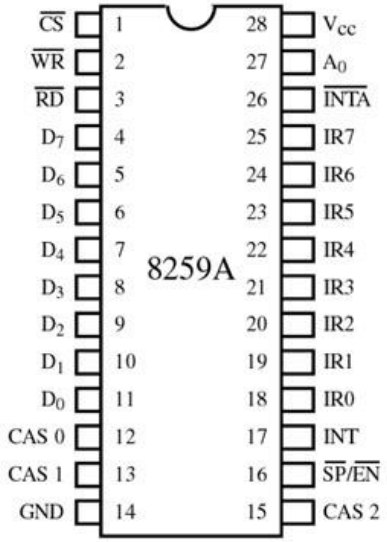


8259 Programmable Interrupt Controller (PIC)

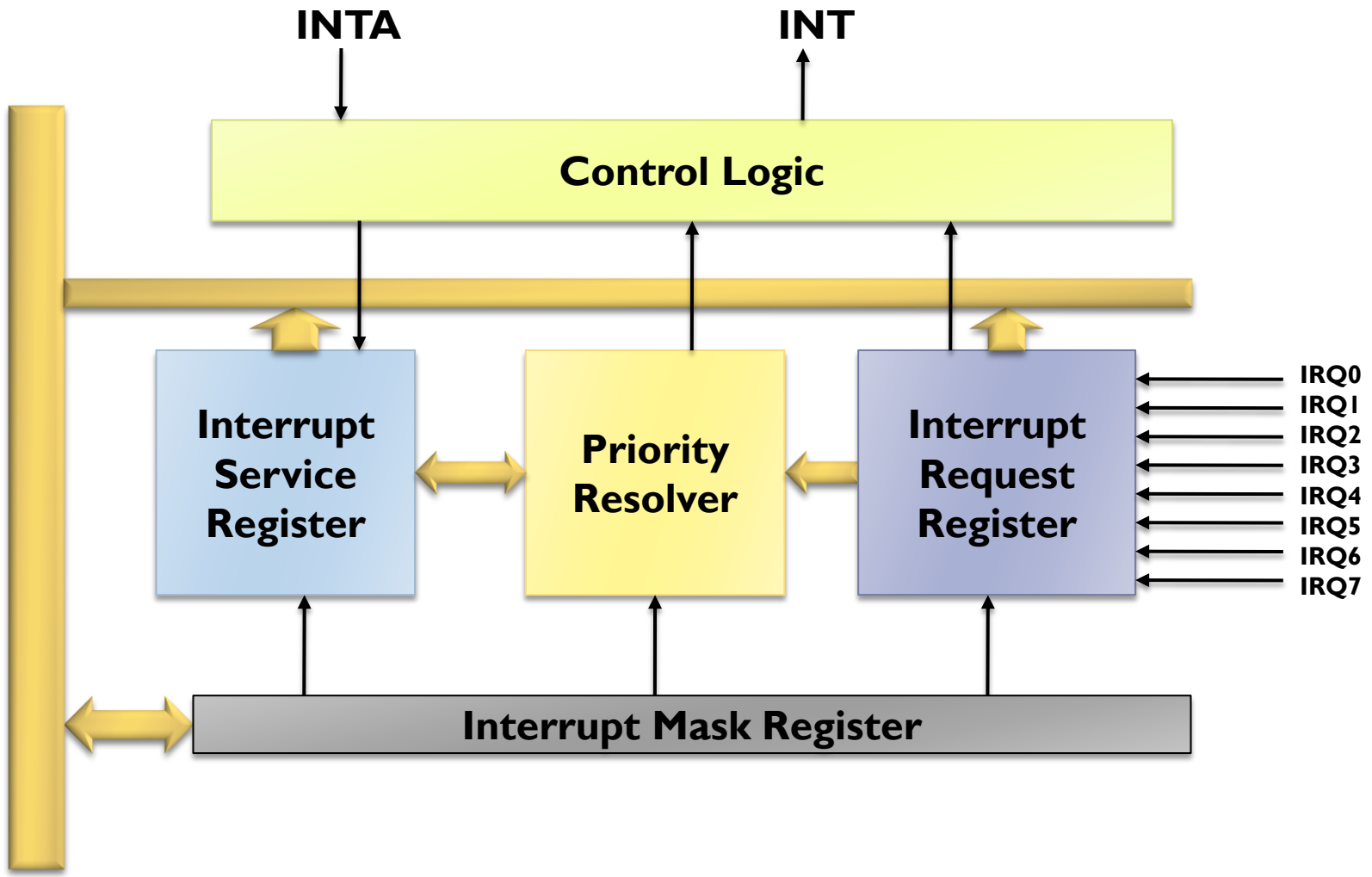


Block Diagram Architecture of 8259





Block Diagram Architecture of 8259



8259 Programmable Interrupt Controller (PIC)

- 8259 can service up to eight hardware devices
  - Interrupts are received on IRQ0 through IRQ7
- 8259 can be programmed to assign priorities in several ways
  - Fixed priority scheme is used in the PIC
    - IRQ0 has the highest priority and IRQ7 lowest
- 8259 has following registers
  - Interrupt Request Register
  - Interrupt Service Register
  - Interrupt Command Register (ICR)
    - Used to program 8259
  - Interrupt Mask Register (IMR)

How to program the 8259

- The 8259A has two modes:
  - Initialization Mode
  - Operational Mode
- Operational Mode Programming:
  - Write a (9-bit) command to the PIC
  - Maybe read a return-byte from the PIC
- Initialization Mode Programming:
  - Write a complete initialization sequence

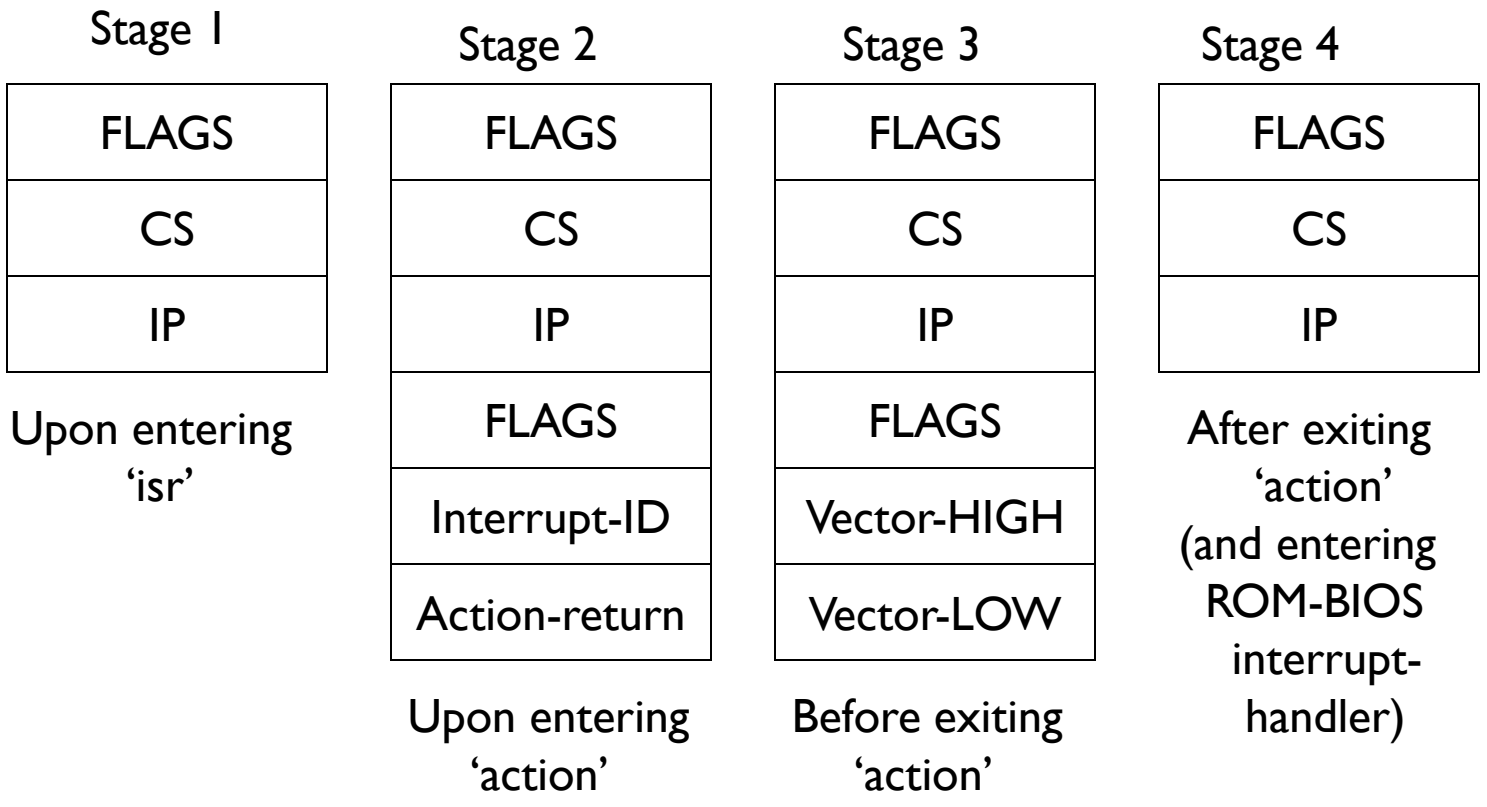
Initializing the 8259

- A series of 9-bit values is sent to the PIC
- Once it's begun, it must be completed
- Each 9-bit values is called an Initialization Command Word (abbreviated ICW)
- The least significant 8 bits are sent on the PC's data-bus, while the 9th bit is sent as bit 0 on the PC's address-bus

Operating the 8259

- After the Initialization Command Words (ICWs) are programmed into the 8259A
- The chip is ready to accept interrupt requests at its input lines.
- The 8259A to operate in various modes through the Operation Command Words (OCWs).
  - Fully nested mode
  - Rotating priority mode
  - Special mask mode
  - Polled mode

Stack Operation States during Interrupt



Thank You !!