Unit 5: Basic I/O, Memory R/W and Interrupt Operations

Syllabus

- **Memory mapped I/O, I/O Mapped I/O and Hybrid I/O**
- **Direct Memory Access (DMA)**
  - Introduction, Advantage and Application
  - 8237 DMA Controller and Interfacing
- **Interrupt**
  - 8085 Interrupt Pins and Priority
  - Maskable and Non-maskable Interrupts
  - RST Instructions
  - Vector and Polled Interrupt
- **8259 Interrupt Controller**
  - Block Diagram and Explanation
  - Priority Modes and Additional Features

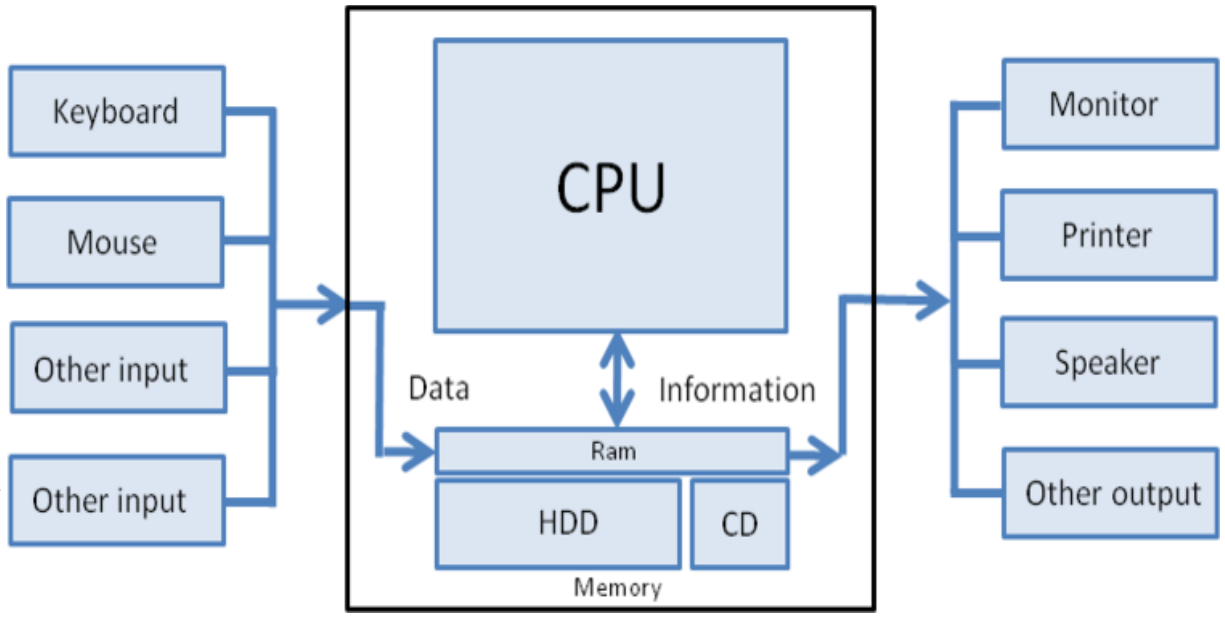
Introduction

- The microprocessor cannot do anything by itself therefore, It needs to be linked with memory, extra peripherals, or IO devices. This linking is called **Interfacing**.
- The interfacing of the I/O devices in 8085 can be done in two ways :

1. **Memory-Mapped I/O Interfacing:** In this kind of interfacing, we assign a memory address that can be used in the same manner as we use a normal memory location.
2. **I/O Mapped I/O Interfacing:** A kind of interfacing in which we assign an 8-bit address value to the input/output devices which can be accessed using IN and OUT instruction is called I/O Mapped I/O Interfacing.

I/O Operations

- CPU uses two methods to perform input/output operations between the CPU and peripheral devices in the computer. These two methods are called **memory mapped IO** and **IO mapped IO**.
- **Memory-mapped IO** uses the same address space to address both memory and I/O devices.
- On the other hand, **IO mapped IO** uses separate address spaces to address memory and IO devices.

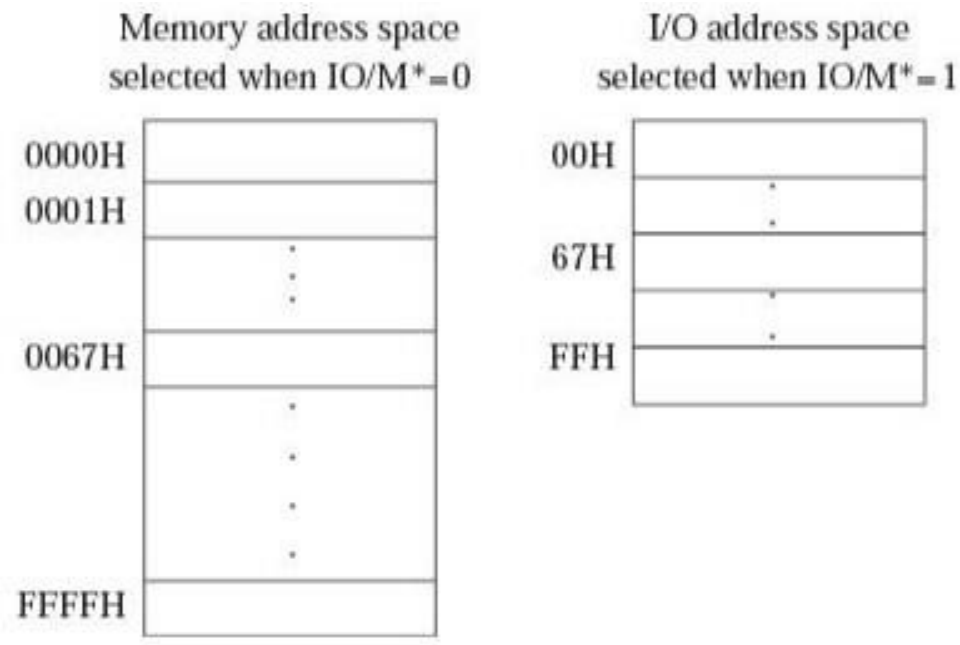


Memory Mapped I/O

- **Memory mapped IO** uses one address space for memory and input and output devices. In other words, some addresses are assigned to memory while others are assigned to store the addresses of IO devices.
- There is one set of read and write instruction lines. The same set of instructions work for both memory and IO operations. Therefore, the instructions used to manipulate memory can be used for IO devices too. Hence, it can lessen the addressing capability of memory because some are occupied by the IO.

Isolated (I/O) Mapped I/O

- **IO mapped IO** uses two separate address spaces for memory locations and for IO devices. There are two separate control lines for both memory and IO transfer. In other words, there are different read-write instruction for both IO and memory.
- IO read and IO write are for IO transfer whereas memory read and memory write are for memory transfer. IO mapped IO is also called port-mapped IO or isolated IO.



Hybrid I/O

- Combination of both previous I/O.
- Uses same address in some cases and two address space in some cases.

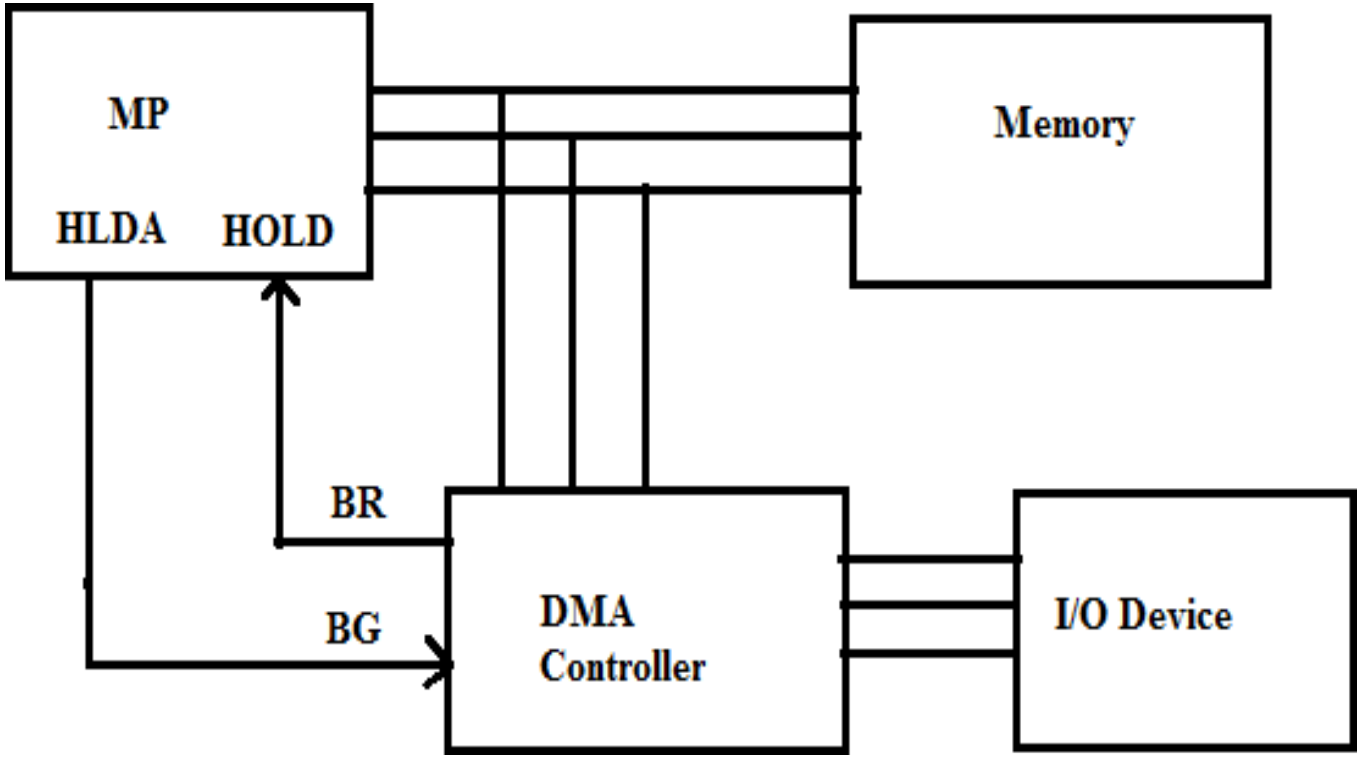
I/O Mapped I/O VS. Memory Mapped I/O

	Memory Mapped I/O	I/O Mapped I/O
1	**I/O devices are mapped into memory space.**	**I/O devices are mapped into I/O space.**
2	I/O devices are allotted **memory addresses.**	I/O devices are allotted **I/O addresses.**
3	Processor **does not differentiate** between memory and I/O. Treats I/O devices also like memory devices.	Processor **differentiates between I/O devices and memory.** It isolates I/O devices.
4	I/O addresses are as big as memory addresses. E.g.:: in 8085, **I/O addresses will be 16 bit** as memory addresses are also 16-bit.	I/O addresses are smaller than memory addresses. E.g.:: in 8085, **I/O addresses will be 8 bit** though memory addresses are 16-bit.
5	This allows us to increase the number of I/O devices. E.g.:: in 8085, we can access up to 2^{16} = **65536 I/O devices.**	This allows us to access limited number of I/O devices. E.g.:: in 8085, we can access only up to 2^{8} = **256 I/O devices.**
6	We can transfer data from I/O devices using **any instruction like MOV** etc.	We can transfer data from I/O device using **dedicated I/O instructions like IN and OUT ONLY.**
7	Data can be transferred using **any register** of the processor.	Data can be transferred only using a fixed register. E.g.:: in 8085 only **"A" register.**
8	We need **only two control** signals in the system: Read and Write.	We need **four control signals:** Memory Read, Memory Write and I/O Read and I/O Write
9	Memory addresses are big so **address decoding will be slower.**	I/O addresses are smaller so **address decoding will be faster.**
10	Address decoding will be **more complex and costly.**	Address decoding will be **simpler and cheaper.**

I/O Mapped I/O VS. Memory Mapped I/O

I/O Mapped I/O	Memory Mapped I/O
I/O device is treated as an I/O device and hence given an I/O address.	I/O device is treated like a memory device and hence given a memory address.
I/O device has an 8 bit address.	I/O device has a 16 bit Memory address.
I/O device is given $\overline{IOR}$ and $\overline{IOW}$ control signals.	I/O device is given $\overline{MEMR}$ and $\overline{MEMW}$ control signals.
Decoding is easier due to lesser address lines.	Decoding is more complex due to more address lines.
Decoding is cheaper	Decoding is more expensive
Works faster due to less delays	More gates add more delays hence slower
Allows max. $2^8 = 256$ I/O devices	Allows many more I/O devices as I/O addresses are 16 bits($2^{16} = 64K$).
I/O devices can only be accessed by IN and OUT instructions.	I/O devices can be accessed using any memory instruction.
Popular technique in Microprocessors.	Popular technique in Microcontrollers.

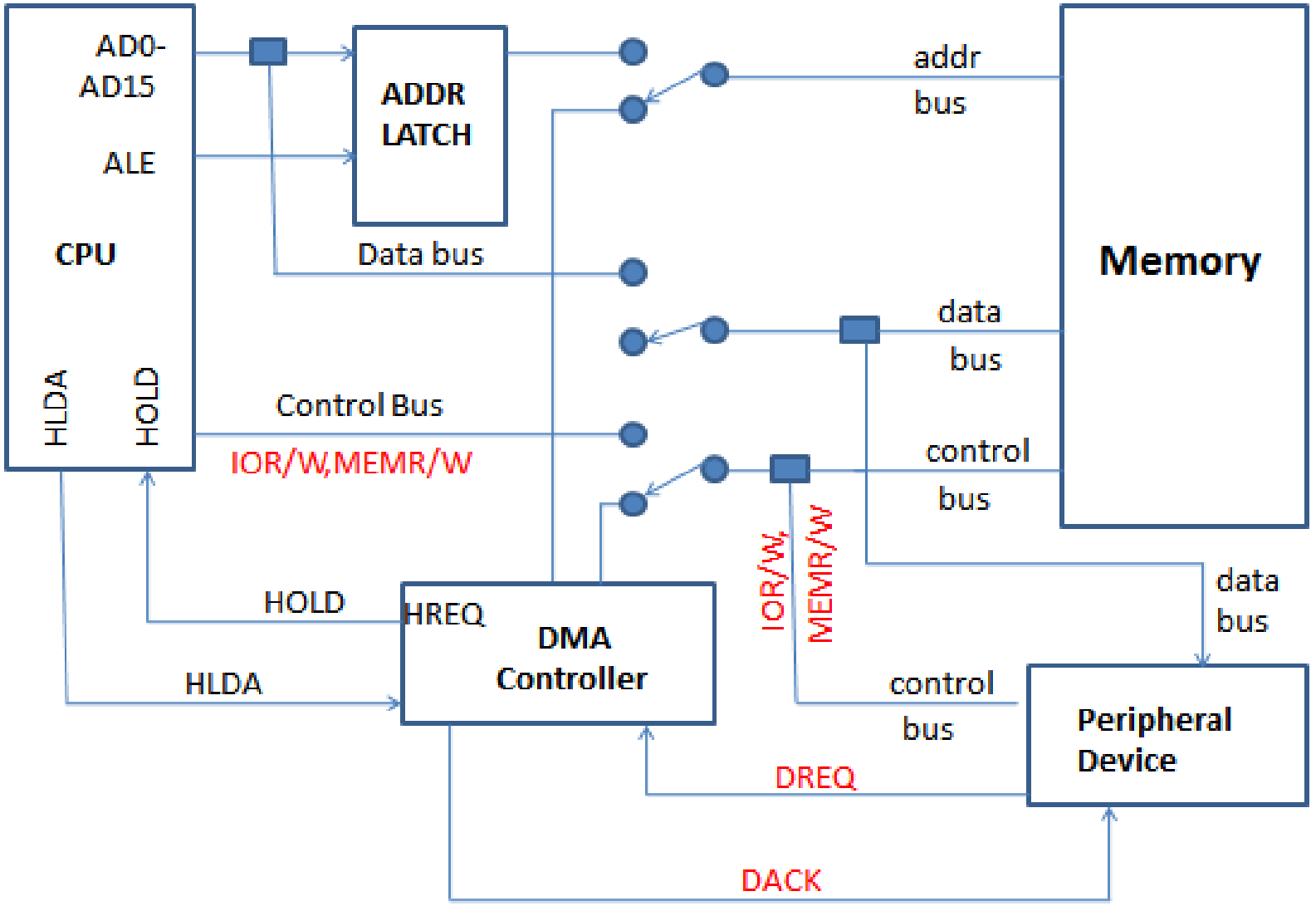
Direct Memory Access (DMA)



Direct Memory Access (DMA)

- DMA is a process of communication for data transfer between memory and input/output, controlled by an external circuit called **DMA controller**, without involvement of CPU.
- 8085 MP has two pins **HOLD** and **HLDA** which are used for DMA operation.
- First, DMA controller sends a request by making Bus Request (BR) control line high. When MP receives high signal to HOLD pin, it first completes the execution of current machine cycle, it takes few clocks and sends HLDA signal to the DMA controller.
- After receiving HLDA through Bus Grant (BG) pin of DMA controller, the DMA controller takes control over system bus and transfers data directly between memory and I/O without involvement of CPU. During DMA operation, the processor is free to perform next job which does not need system bus.
- At the end of data transfer, the DMA controller terminates the request by sending low signal to HOLD pin and MP regains control of system bus by making HLDA low.

Concept of DMA



Concept of DMA

- DMA Transfer is a **hardware controlled I/O** Transfer technique.
- It is mainly used for **high-speed data transfer between I/O and Memory** where the speed of the peripheral is generally faster than the μP.
- In Program Controlled I/O, Status or interrupt driven I/O the speed of transfer is **slow** mainly **because instructions** need to be **decoded** and **then executed** for the transfer.
- **DMA transfer** is **software independent** and **hence much faster**.
- A device known as the DMA Controller (DMAC) is responsible for the DMA transfer.

Sequence of DMA Transfer

1) µP initializes the DMAC by giving the starting address and the number of bytes to be transferred.

2) An **I/O device requests** the **DMAC**, to perform DMA transfer, **through the DREQ line**.

3) The **DMAC** in turn sends a **request signal to** the **µP**, through the **HOLD line**.

4) The **µP finishes the current machine cycle** and **releases** the **system bus** (gets disconnected from it).

It also **acknowledges** receiving the HOLD signal through the **HLDA line**.

5) The **DMAC acquires control of the system bus.**

The **DMAC sends** the **DACK signal** to the I/O peripheral and the **DMA transfer begins**.

6) After every byte is transferred, the Address Reg is incremented (or decremented) and the Count Reg is decremented.

Sequence of DMA Transfer

7) This continues till the Count reaches zero (Terminal Count). Now the DMA transfer is completed.

8) **At the end** of the transfer, **the system bus is released by the DMAC** by making HOLD = 0.

Thus **µP takes control of the system bus** and continues its operation.

The DMA Controller (DMAC) does DMA transfer through its channels.

The minimum requirements of each channel are:

i. **Address Register** (to store the starting address for the transfer).
ii. **Count Register** (to store the number of bytes to be transferred).
iii. **A DREQ signal** from the IO device**.**
iv. **A DACK signal** to the IO device.

Advantages & Disadvantages of DMA

- **Advantages**
  - DMA allows a peripheral device to read from/write to memory without going through the CPU. Thus it decreases the clock cycle needed to write or read a block of data.
  - DMA allows faster processing since the processor can be working on something else while the peripheral can be populating memory.
  - DMA enables more efficient use of interrupts.
  - High transfer rates, It transfers the data without involving the processor, so the read-write task gets speed up.
  - DMA capable device can communicate directly with memory.
  - It reduces the workload of the processor.
- **Disadvantages**
  - Cost of DMA hardware.
  - Data has to be stored in continuous memory locations.
  - DMA controller is slow in comparison to CPU.

Application of DMA

- DMA has been a built-in feature of PC architecture since the introduction of the original IBM PC.
- PC-based DMA was used for floppy disk I/O in the original PC and for hard disk I/O in later versions.
- PC-based DMA technology, along with high speed bus technology, is driven by data storage, communications, and graphics needs-all of which require the highest rates of data transfer between system memory and I/O devices.
- Applications areas are**: cinemas, theatres, hotels, railway stations, shopping centres, trade shows, museums & many more**.

8257 DMA Controller

❖Here is a list of some of the prominent features of 8257:

- ❑It has four channels which can be used over four I/O devices.
- ❑Each channel has 16-bit address and 14-bit counter.
- ❑Each channel can transfer data up to 64kb.
- ❑Each channel can be programmed independently.
- ❑Each channel can perform read transfer, write transfer and verify transfer operations.
- ❑It generates MARK signal to the peripheral device that 128 bytes have been transferred.
- ❑It requires a single phase clock.
- ❑Its frequency ranges from 250Hz to 3MHz.
- ❑It operates in 2 modes, i.e., **Master mode** and **Slave mode**.

8257 Direct Memory Access

❖ Basics of 8257 Direct Memory Access

- ❑ 8257 is used to for high speed data transfer in between in IO devices and memory.
- ❑ Using Microprocessor data transfer is slow, as microprocessor have to execute instructions and it have to check interrupt as well.
- ❑ Using 8257, MPU releases the control of the buses to the DMA.
- ❑ Here, Data transfer between memory and IO is been done by bypassing MPU.
- ❑ To take control of Address and Data bus from MPU, it has HOLD and HLDA control terminals which are used by DMA.

8257 Direct Memory Access

❖ HOLD and HLDA

❑ HOLD :

- This is active high signal input MPU.
- It gives request to MPU for address and data bus control.
- After receiving HOLD signal, MPU relinquishes the buses in following machine cycle.
- All buses are tri stated and HLDA sent out by MPU.
- MPU regains the control of buses after HOLD goes low.

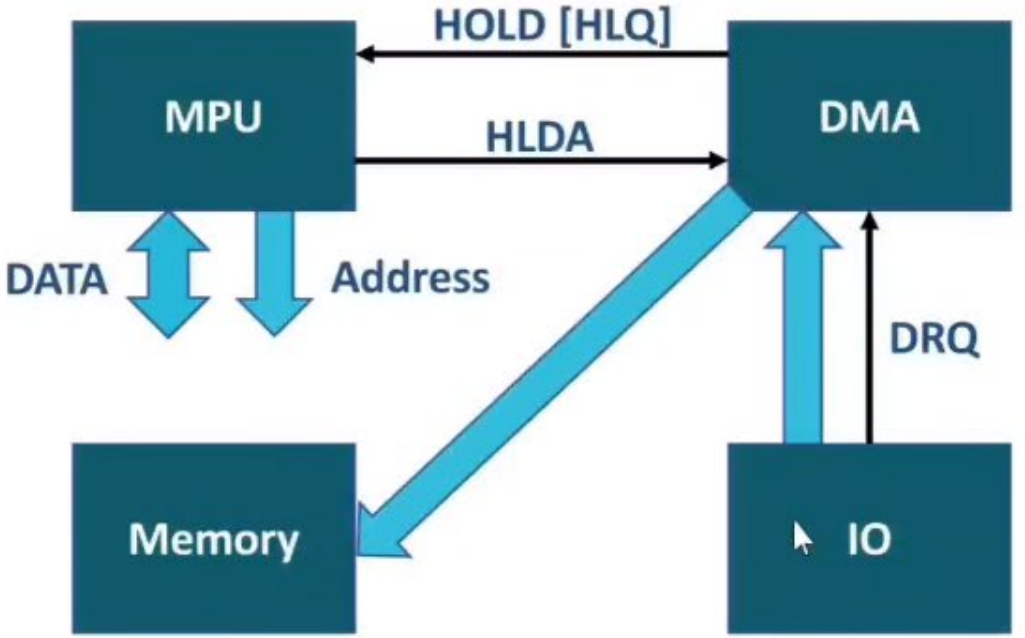
❑ HLDA :

- This is active high signal indicating that the MPU is relinquishing control of buses.

8257 Direct Memory Access

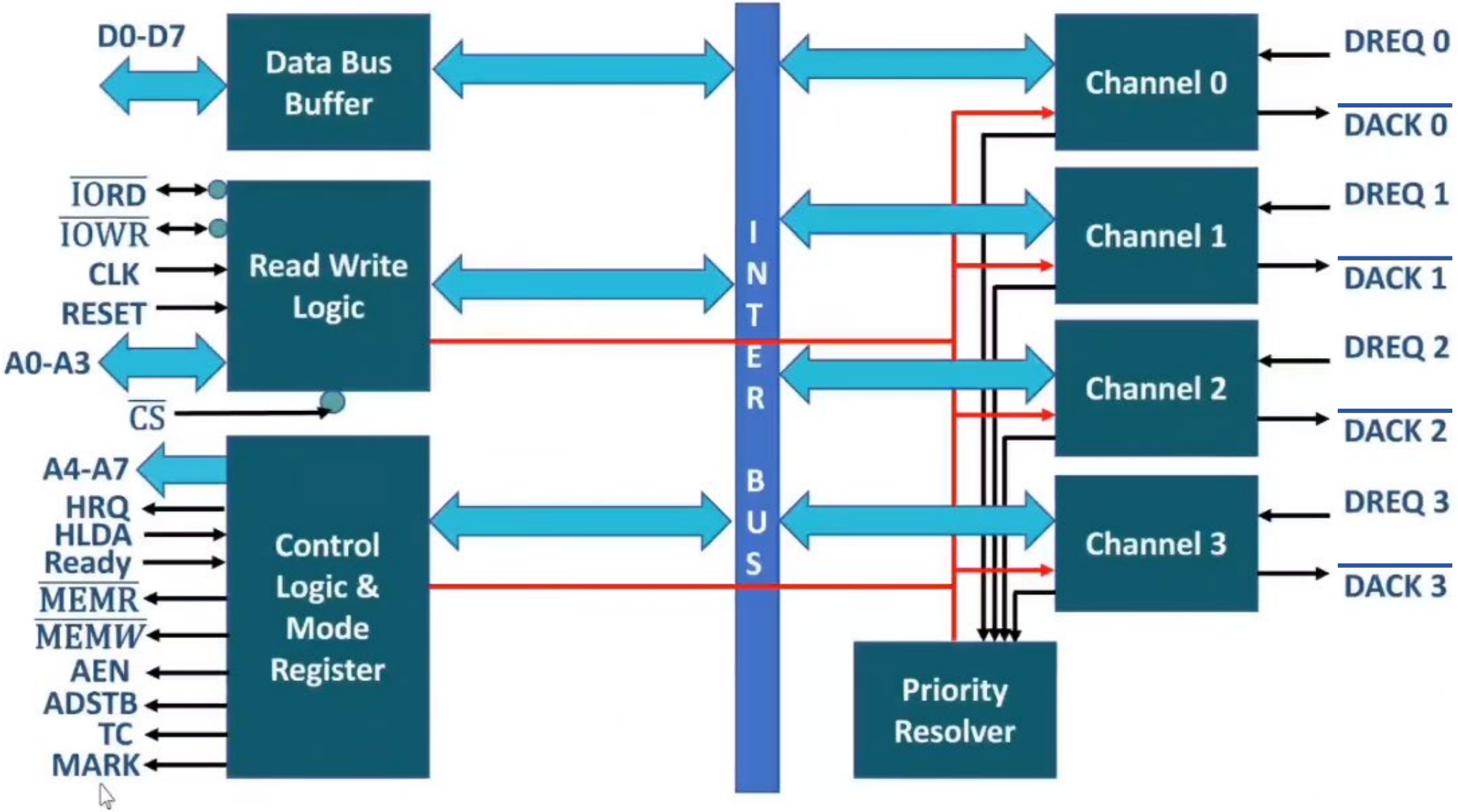
❖ Working of DMA

- If IO wants to send data to memory, then it will send request to DMA. [DRQ]
- To take control of system buses, DMA will send HOLD signal to MPU.
- To give control of address and data, MPU will give HLDA [HOLD Acknowledge]. Which indicates that now DMA is master of Buses. So now buses will be managed by DMA for memory and IO.



- Now data transfer can happen without involvement of MPU. Here now MPU don't need to execute instructions, so data exchange will be faster.
- Once HOLD signals goes low, MPU will take control of system buses and then MPU becomes master.

Block Diagram of 8257



Pin Configuration

The **PIN CONFIGURATION** of 8237 are explained as follows:

1) DREQ3 – DREQ0 (Data Request)

- These **input pins** are used by the **I/O device to request the DMAC** for DMA Transfer.
- These pins can be made **active high / active low through program**.
- By **default**, they are **active high**.
- There are **4 DREQ** pins **as** there are **4 channels** (one per channel).
- **At a time only one channel** can perform DMA transfer.
- Thus, for **multiple requests**, **priorities** are fixed.
- **DREQ0** has the **highest priority** while **DREQ3** has the **lowest priority**.

Pin Configuration

2) $\overline{\text{DACK3}}$ – $\overline{\text{DACK0}}$ (Data Request)

- These **input pins** are used by the **DMAC to inform the I/O device** that it has acquired the system bus, and hence the **DMA transfer can begin**.
- These pins can be made **active high / active low through program**.
- By **default**, they are **active low**.
- There are **4 DACK pins** as there are **4 channels** (one per channel).

3) HRQ (Hold Request)

- This **output** pin is used by the DMAC to **request the µP** to release the system bus.
- It is **connected to** the **HOLD** pin of the µP.

4) HLDA (Hold Acknowledge)

- This **input pin** is used by the µP **to inform the DMAC** that it as released to system bus.
- It is **connected** to the **HLDA** pin of the µP.

Pin Configuration

5) AEN (Address Enable)

- This is an **output** pin from the DMAC.
- When this pin is high, **it disconnects the µP from the system bus**.
- It is **also** used to **enable** the **external latch**.

6) ADSTB (Address Strobe)

- It is an **output** signal.
- It is used to **strobe** the higher order **address** byte **into the latch**.

7) DB7 – DB0 (Data Bus)

- These are **8 bi-directional data lines** used to connect the internal data bus of 8237 with the external (system) data bus.
- In **idle cycle**, **µP writes or reads** from 8237 **using this bus**.
- In **active cycle**, this bus **carries** the **8 higher order bits** of the 16 bit **address** (the other 8 bits being in the **A7–A0**).
- During **memory-to-memory transfer**, this bus **carries** the **data** byte to be transferred.

Pin Configuration

8) A7 – A4 (Address bits)

- These are **4 output address lines**.
- In **active cycle**, these lines **carry the A7–A4 bits of the address** at which the transfer is to be done.
- As this address is generated by the 8237, these are output lines.

9) A3 – A0 (Address bits)

- These are **4 bi-directional address lines**.
- In **idle cycle**, **µP sends** the **address A3–A0**, to select one of its registers.
- Since µP sends the address to 8237, these are input lines.
- In **active cycle**, these lines **carry the A3–A0 bits of the address** at which the transfer is to be done.
- As this address is generated by the 8237, these are output lines.

Pin Configuration

10) $\overline{\text{IOR}}$, $\overline{\text{IOW}}$

- These are **bi-directional control lines**.
- During **idle state**, the **µP issues** these signals **to read from or write into the 8237** (as the DMAC itself is an I/O device w.r.t. the µP).
- During **active state**, the **DMAC issues** these signals **to read from or write into an I/O Device**.

11)$\overline{\text{MEMR}}$, $\overline{\text{MEMW}}$

- These are **output control lines**.
- During **active state**, the **DMAC issues** these signals **to read from or write into the Memory**.

12) CLK

- This is a **clock-input** signal for the DMAC.
- It is usually connected to the system clock.

Pin Configuration

13) RESET

- This is a **reset-input** signal for the DMAC.
- This signal **clears the internal registers** of the DMAC and causes it to enter **Idle State**.

14) READY

- This is an **input** signal to the DMAC.
- It is used to **synchronize** the DMAC with "**Slower**" peripherals.
- **If** a peripheral is slow (**not ready**), it put a **low** on this line causing the **DMAC** to **wait**.
- Only after the DMAC finds a high on this pin, it executes the DMA operation.

Pin Configuration

15) TC

- This is an output signal from the DMAC.
- It becomes 1 to indicate that terminal count is reached, that means the count has become 0 and
- hence the transfer has been completed.

16) Mark

- This is an output signal from the DMAC.
- It is a modulo 128 mark output. This means it becomes 1 on every 128th cycle of the data transfer
- as a marker to indicate that a batch of 128 cycles has been completed.

Pin Configuration

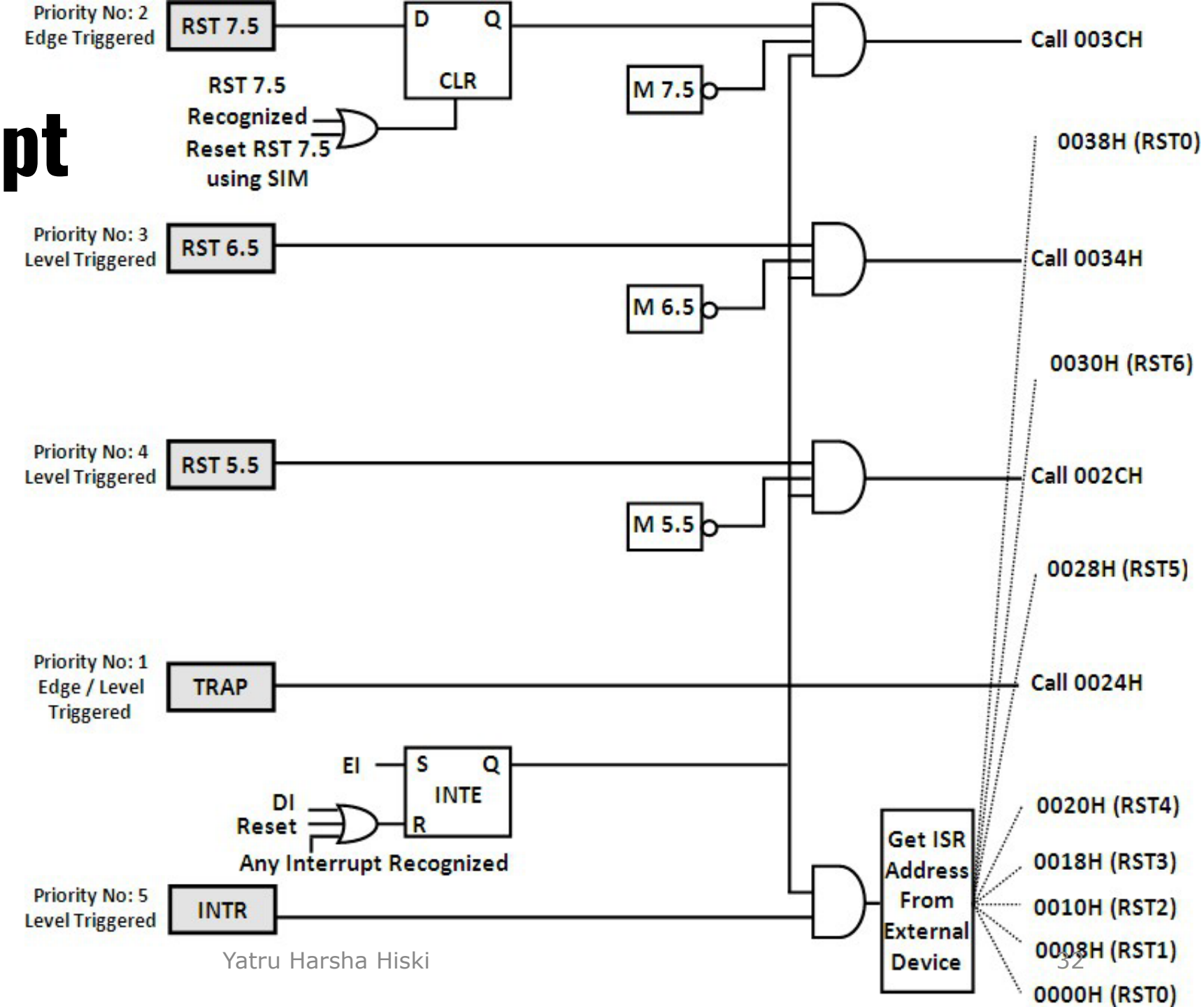
❖ **EOP**

- This is a **bi-directional signal** indicating the **end of DMA process**.
- During **active cycle**, **after each byte** is transferred through DMA, the **Count Register decrements** by 1.
- When the **Terminal Count** is reached, it means that **all** the required **bytes** are **transferred**.
- At this point, the **DMAC issues** this signal and the **DMA operation is terminated**. Hence, it is an output signal.
- During the course of the transfer, the **DMA operation can be explicitly terminated by giving this signal** externally to the DMAC. Hence, it is also an input signal.

8085 Interrupt

- An interrupt is a **special condition** that arises during the working of a µP.
- In response, the **µP services** the interrupt **by executing** a subroutine called as the **Interrupt Service Routine**.
- The **µP checks** for interrupts **during every instruction**.
- When an **interrupt occurs**, the **µP first finishes the current instruction**.
- It then **Pushes** the address of the next instruction **(contents of PC) on the STACK**.
- It **resets** the **INTE flip-flop** so that **no more interrupts are recognized**.
- Thereafter the program control transfers to the **address of** the Interrupt Service Routine (**ISR**) and the **µP** thus **executes the ISR**.
- Interrupts are of two types:
  - Software Interrupts
  - Hardware Interrupts

8085 Interrupt



Software Interrupts

- Interrupts that are **initiated through program** (software) are called as software interrupts.
- 8085 supports 8 software interrupts:
- **RSTn** where n = 0,1,2, … 7 i.e. RST0, RST1 … upto RST7.
- This instruction causes a **service routine** to be Called from the **address (n*8)**.
- Hence, if RST1 occurs then the program control moves to location 0008 (1*8 = 0008).
- The respective addresses for software interrupts are given below.

S/W Interrupt	ISR Address
RST0	00**0**0H
RST1	00**08**H
RST2	00**1**0H
RST3	00**18**H
RST4	00**2**0H
RST5	00**28**H
RST6	00**3**0H
RST7	00**38**H

Hardware Interrupts

- Interrupts that are initiated through a hardware pin are called as hardware interrupts.
- 8085 supports the following hardware interrupts:
  - TRAP
  - RST 7.5
  - RST 6.5
  - RST 5.5
  - INTR

8085 Interrupt

❖ **Vectored Interrupts:**

- Interrupts that **have** a **FIXED Address** for their ISR (Interrupt Service Routine) are called as Vectored Interrupts.
- **Eg: TRAP** is a vectored interrupt. Its vector address is 0024H.

❖ **Non-Vectored Interrupts:**

- Interrupts that **have** a **Variable Address** for their ISR are called as Non-Vectored Interrupts.
- **Eg: INTR** is a Non-Vectored Interrupt.

❖ **Methods of preventing an interrupt from occurring.**

- **MASK Individual Bits** through **SIM** Instruction
- **Disable all** Interrupts through **DI** Instruction

8085 Interrupt

❖ **MASKING:**

- ❑ We can prevent an interrupt from occurring by MASKING its individual bit through **SIM Instruction**.
- ❑ If an interrupt is masked it will not be serviced.
- ❑ One of the **main advantages** of masking as opposed to disabling interrupts is that by masking we can **selectively disable a particular interrupt** while keeping other interrupts active**,** whereas through DI instruction all interrupts are disabled.
- ❑ ONLY **RST 7.5, RST 6.5 and RST 5.5** can be masked by this method.

❖ **DISABLING INTERRUPTS:**

- ❑ Interrupts can be disabled through the **DI** Instruction.
- ❑ This instruction resets the INTE Flip Flop and hence none of the interrupts can occur (**Except TRAP**). I.e. INTE F/F ← 0.
- ❑ Once disabled, these interrupts can be **re-enabled** through **EI** instruction, which sets the INTE Flip Flop. I.e. INTE F/F ← 1.

Hardware Interrupts

❖ **TRAP:**

- TRAP has the **highest priority.**
- It is **Edge as well as Level triggered** hence the signal must go High and also Remain high for some time for it to be recognized. This prevents any noise signal from being accepted.
- It is a Non-Maskable Interrupt i.e. it can **neither be masked nor be disabled.**
- It is a vectored interrupt and has a **vector address of 0024H.**

❖ **RST 7.5:**

- RST 7.5 has the **priority lower than TRAP.**
- It is **Edge triggered.**
- It is a **Maskable Interrupt** i.e. it can be masked through the **SIM Instruction**.
- It can also be disabled though the **DI Instruction**.
- It is a vectored interrupt and has a **vector address of 003CH.**
- RST 7.5 can also be **reset** through the **R 7.5** bit in the **SIM** Instruction irrespective of whether it is Masked or not.

Hardware Interrupts

❖ **RST 6.5:**

- RST 6.5 has the **priority lower than RST 7.5.**
- It is **Level triggered.**
- It is a **Maskable Interrupt** i.e. it can be masked through the **SIM Instruction**.
- It can also be disabled though the **DI Instruction**.
- It is a vectored interrupt and has a **vector address of 0034H.**

❖ **RST 5.5:**

- RST 5.5 has the **priority lower than RST 6.5.**
- It is **Level triggered.**
- It is a **Maskable Interrupt** i.e. it can be masked through the **SIM Instruction**.
- It can also be disabled though the **DI Instruction**.
- It is a vectored interrupt and has a **vector address of 002CH.**

Hardware Interrupts

❖ **INTR:**

- ❑ INTR has the **priority lower than RST 5.5.**
- ❑ It is **Level triggered.**
- ❑ It can only be disabled though the **DI Instruction**.
- ❑ **It cannot be masked through the SIM Instruction**.
- ❑ It is a **Non-Vectored** interrupt.

Hardware Interrupts

❖ **Response to INTR:**

- ❑ When INTR occurs the μP, in response, issues the **first INTA** cycle.
- ❑ The **External Hardware sends** an **opcode**, which can be of **RSTn** Instruction **or** of **CALL** instruction.

a) If opcode of **RSTn** is sent by the external hardware
- ✓ The μP calculates the address of the ISR as **n*8.**

b) If opcode of **CALL** is sent:
- ✓ As Call is a **3-Byte Instruction** the μP send **2 more INTA** signals
- ✓ In response to the **2nd and the 3rd INTA** cycle the external hardware returns the **lower and the higher byte of the address** of the ISR respectively.

- ❑ This is how the address is determined when INTR occurs.

Hardware Interrupts

❖ **INTA :** (Interrupt Acknowledge)

- ❑ This is an **active low acknowledge** signal going out of 8085.
- ❑ This signal is **given in response to** an interrupt on **INTR ONLY**.
- ❑ After the **first INTA** is given, the interrupting **peripheral sends an opcode**.
- ❑ **If** the **Opcode** is **of RSTn**, then the **ISR address is calculated as n × 8.**
- ❑ **If** the **Opcode** is **of CALL**, then **two more INTA signals** are given

EI and DI Instructions

❖ **INTE F/F :** (Interrupt Enable Flip Flop)

- ❑ This flip-flop decides if interrupts are enabled in the μP i.e. **if it is set, all interrupts are enabled**.
- ❑ It is **set by** the **EI Instruction**.
- ❑ It is **reset** in the following 3 ways:

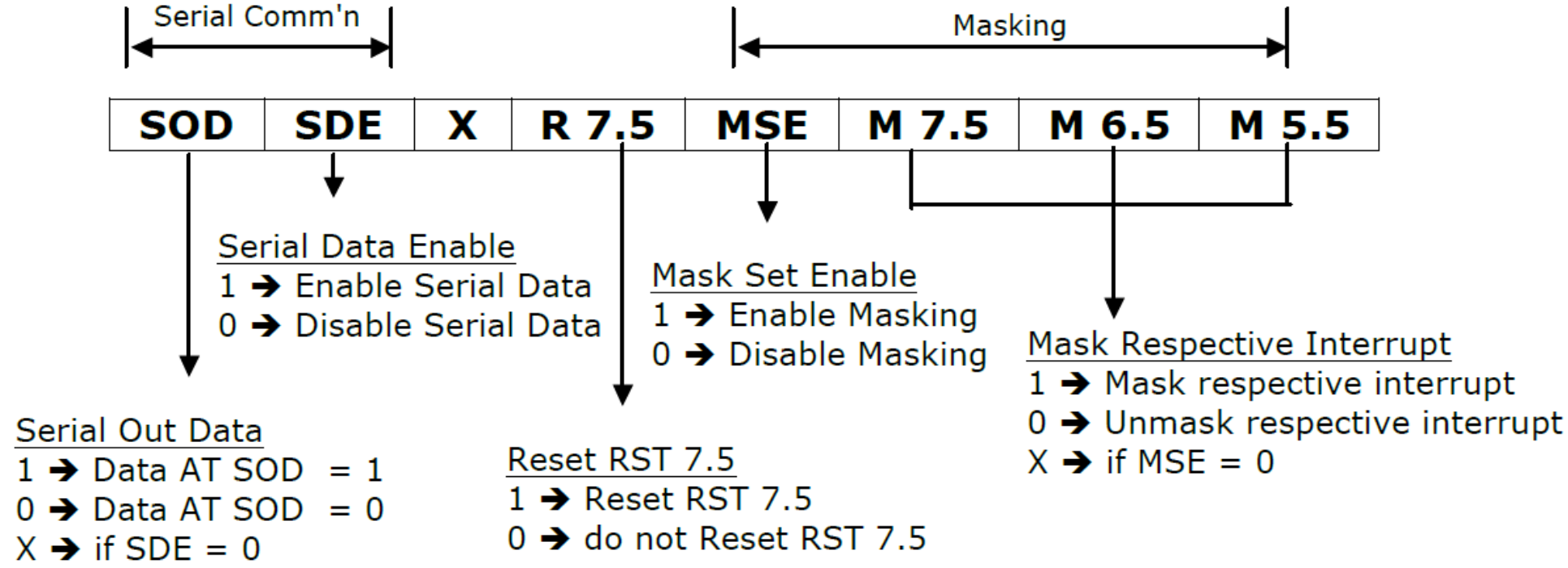
i. **If μP is reset**.
ii. If **DI instruction** is executed.
iii. If **any other interrupt is recognized** by the μP. In this case the INTE F/F is later set in the ISR by EI.

- ❑ The INTE F/F affects all interrupts **EXCEPT TRAP**, as it cannot be disabled.

SIM Instruction: (Set Interrupt Mask)

❖ This instruction is used for the following purposes:

i. To **Mask** or Un-Mask the **RST7.5, RST6.5 and RST5.5** interrupts.
ii. To send the data out serially (bit - by - bit) through the **SOD** line of the μP.
iii. To **reset RST7.5** interrupt irrespective of whether it is masked or not.



SIM Instruction: (Set Interrupt Mask)

❖ **Method of execution:**

- ❑ The appropriate byte is formed and **loaded into** the **Accumulator**.
- ❑ Then the **SIM** Instruction is **executed**.
- ❑ The µP reads the contents of the accumulator in the above order.

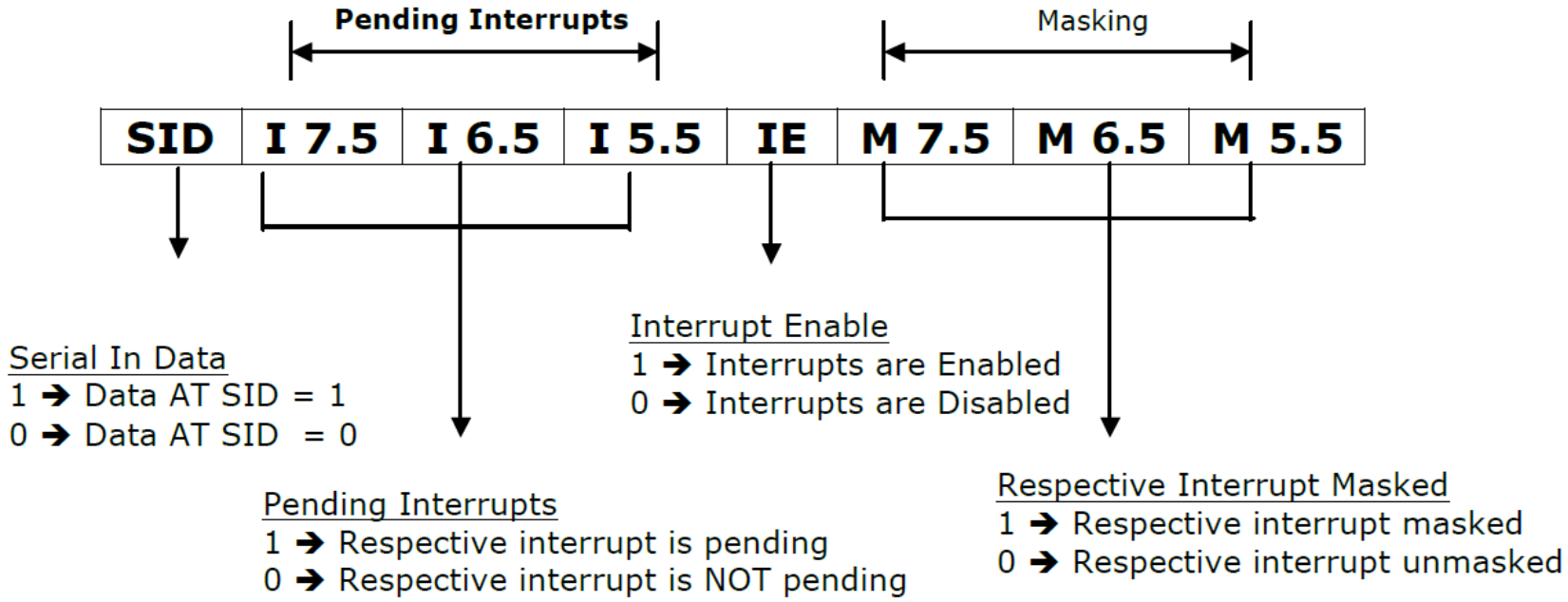
RIM Instruction: (Read Interrupt Mask)

❖This instruction is used for the following purposes:

i. To **read the Interrupt Mask** of the μP.

ii. To accept data serially through the **SID** pin.

iii. To see the **"Pending Interrupts"** of the μP.



RIM Instruction: (Read Interrupt Mask)

❖ **Pending Interrupts:**

❑ Pending interrupts are those interrupts, which are waiting to be serviced.

❑ An interrupt becomes pending as a higher priority interrupt is currently being serviced.

❑ RIM Instruction indicates the Pending Status of RST7.5, RST6.5 and RST5.5.

❖ **Method of execution:**

❑ Then the **RIM** Instruction is **executed**.

❑ The **µP loads** the appropriate byte into the **Accumulator.**

❑ The programmer reads the contents of the Accumulator.

Process of INTR interrupt

1. With the use of the EI instruction, the interrupt process should be enabled.
2. Whenever an instruction is executed, the 8085 checks for an interrupt signal.
3. If INTR is high, the microprocessor completes current instruction, disables the interrupt and sends $\overline{\text{INTA}}$ signal to the peripheral device.
4. $\overline{\text{INTA}}$ allows the peripheral device to send an RST instruction through data bus.
5. Upon receiving the $\overline{\text{INTA}}$ signal, the microprocessor saves the memory location of the next instruction on the stack and the program is transferred to 'call' location (ISR Call) specified by the RST instruction.
6. Microprocessor executes the ISR.
7. Within the program, In order to enable the further interrupt, ISR must include the 'EI' instruction.
8. The RET instruction at the end of the ISR retrieves the return address from the stack and the program is transferred back to main program which was interrupted.

Interrupt Properties

Interrupt	Priority	Triggering	Maskable by SIM	Disabled by DI	Vectored	Vector Address
TRAP	**1**	**Edge / Level**	**No**	**No**	Yes	0024 H
RST 7.5	2	**Edge**	Yes	Yes	Yes	003C H
RST 6.5	3	Level	Yes	Yes	Yes	0034 H
RST 5.5	4	Level	Yes	Yes	Yes	002C H
INTR	5	Level	**No**	Yes	**No**	**Get ISR Address from External Hardware**

Uses of Interrupts in 8085 microprocessor

Interrupts in the 8085 microprocessor are used for various purposes, including:

1. **Real-time processing:** Interrupts allow the microprocessor to respond quickly to external events in real-time, such as user input or hardware signals. This is particularly useful in applications such as control systems and data acquisition systems where time-critical operations are required.
2. **Multi-tasking:** Interrupts enable the microprocessor to perform multiple tasks simultaneously by temporarily suspending the current task and executing the ISR for the interrupting event. This allows the microprocessor to switch between different tasks and maximize the utilization of system resources.
3. **Input/output operations:** Interrupts can be used for handling input/output operations, such as data transfer between the microprocessor and external devices. This allows the microprocessor to perform other tasks while waiting for input/output operations to complete.
4. **Error handling:** Interrupts can be used for error handling and exception handling, such as detecting and recovering from hardware or software errors.
5. **Power management:** Interrupts can be used for power management, such as putting the microprocessor into a low-power mode when it is not needed and waking it up when an interrupt occurs.

Interrupt structures

- A processor is usually provided with one or more interrupt pins on the chip.
- Therefore a special mechanism is necessary to handle interrupts from several devices that share one of these interrupt lines.
- There are mainly two ways of servicing multiple interrupts which are **polled interrupts** and **daisy chain (vectored) interrupts**.

Polled Interrupt

- In a computer, a polled interrupt is a specific type of I/O interrupt that notifies the part of the computer containing the I/O interface that a device is ready to be read or otherwise handled but does not indicate which device. The interrupt controller must poll (send a signal out to) each device to determine which one made the request.
- Polled interrupts are handled using mostly software and are therefore slower compared to vectored (hardware) interrupts. The processor responds to an interrupt by executing one general service routine for all devices. The priority of these devices is determined by the order in which the routine polls each device. Once the processor determines the source of interrupt, it branches to the service routine for that device. The typical configuration of the polled interrupt is shown in figure below.
- As shown in figure, several external devices (Device1, Device2,……., Device N) are connected to a single interrupt line (INTR) of the processor. When one or more devices activate the INTR line high, the processor saves the content of the PC and other registers and then branches to an address defined by the manufacturer of the processor. The user can write a program at this address in order to poll each device starting with highest priority device in order to find the source of the interrupt.
- Polled interrupts are very simple. But for a large number of devices, the time required to poll each device may exceed the time to service the device.

Polled Interrupt

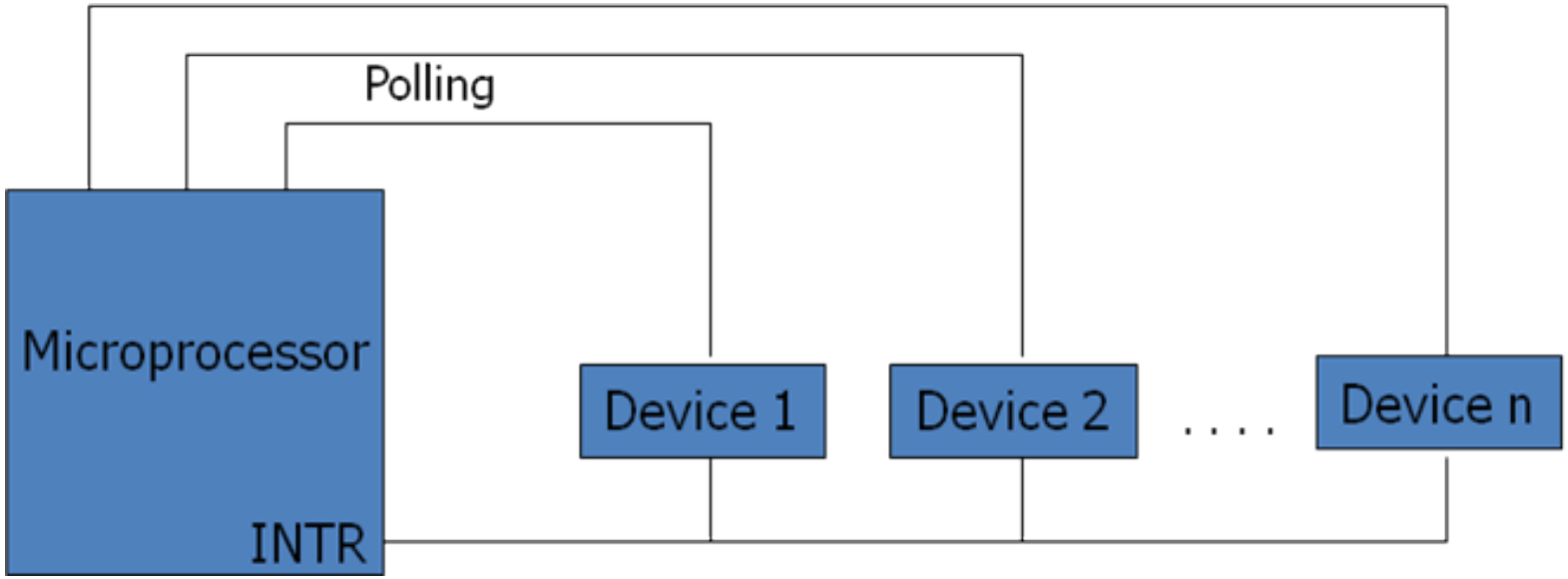


Fig: Polled Interrupt

Vectored(Daisy Chain) Interrupt

- In polled interrupt, the time required to poll each device may exceed the time to service the device through software. To improve this, the faster mechanism called vectored or daisy chain interrupt is used.
- Here the devices are connected in chain fashion. When INTR pin goes up, the processor saves its current status and then generates $\overline{\text{INTA}}$ signal to the highest priority device. If this device has generated the interrupt, it will accept the INTA; otherwise it will push $\overline{\text{INTA}}$ to the next priority device until the $\overline{\text{INTA}}$ is accepted by the interrupting device.
- When $\overline{\text{INTA}}$ is accepted, the device provides a means to the processor for findings the interrupt address vector using external hardware. The accepted device responds by placing a word on the data lines which becomes the vector address with the help of any hardware through which the processor points to appropriate device service routine. Here no general interrupt service routine need first that means appropriate ISR of the device will be called.

Vectored(Daisy Chain) Interrupt

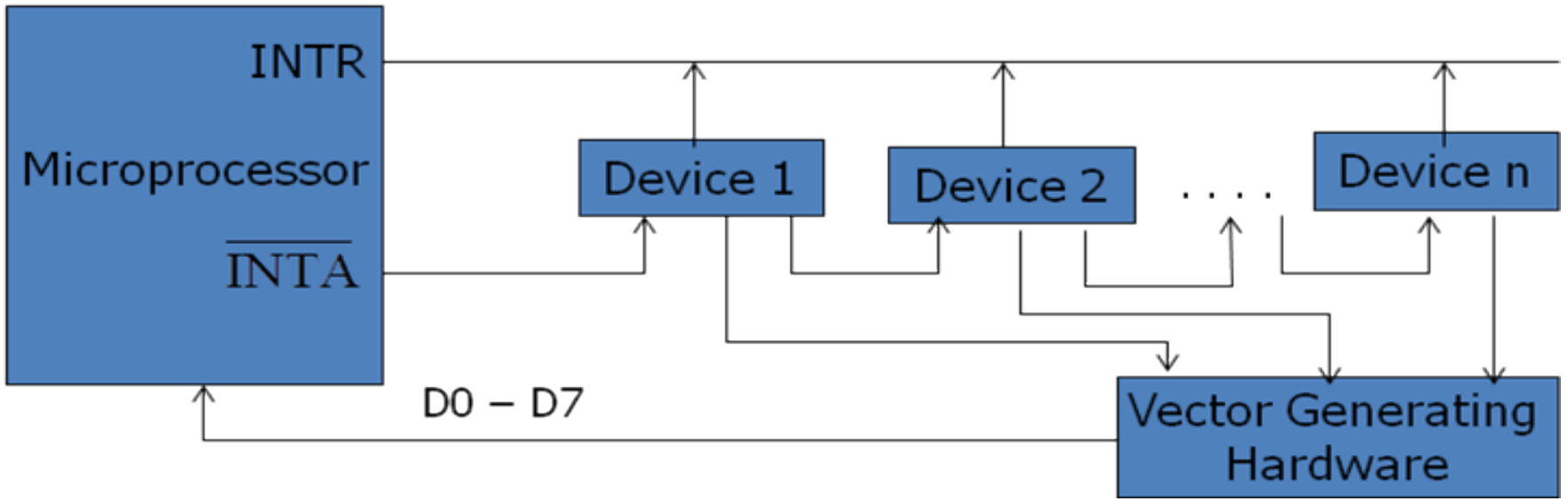


Fig: Vectored (Daisy Chain) Interrupt

Q. If INTR is non vectored interrupt, then why it is used as vectored in Daisey-chaining interrupt handling mechanism?

Answer:

While the INTR itself is technically a non-vectored interrupt because it doesn't directly specify an interrupt vector (memory address) like some other interrupt systems, it can be used in a vectored manner when combined with external hardware, such as a priority encoder or daisy-chaining logic or The 8259 Interrupt Controller.

Summary of 8085 Interrupts

Interrupt	Vector address	Priority	Type
TRAP	0024H	1 (highest priority)	Hardware interrupt Vectored interrupt Non-maskable interrupt
RST 7.5	003CH	2	Hardware interrupt Vectored interrupt Maskable interrupt
RST 6.5	0034H	3	Hardware interrupt Vectored interrupt Maskable interrupt
RST 5.5	002CH	4	Hardware interrupt Vectored interrupt Maskable interrupt
INTR	-	5(lowest priority)	Hardware interrupt Non - Vectored interrupt Maskable interrupt
RST instruction	RST 0 – 0000H RST 1 – 0008H RST 2 – 0010H RST 3 – 0018H RST 4 – 0020H RST 5 - 0028H RST 6 – 0030H RST 7 – 0038H	-	Software interrupt Vectored interrupt Maskable interrupt

The 8259 Interrupt Controller/ 8259 Programmable Interrupt Controller

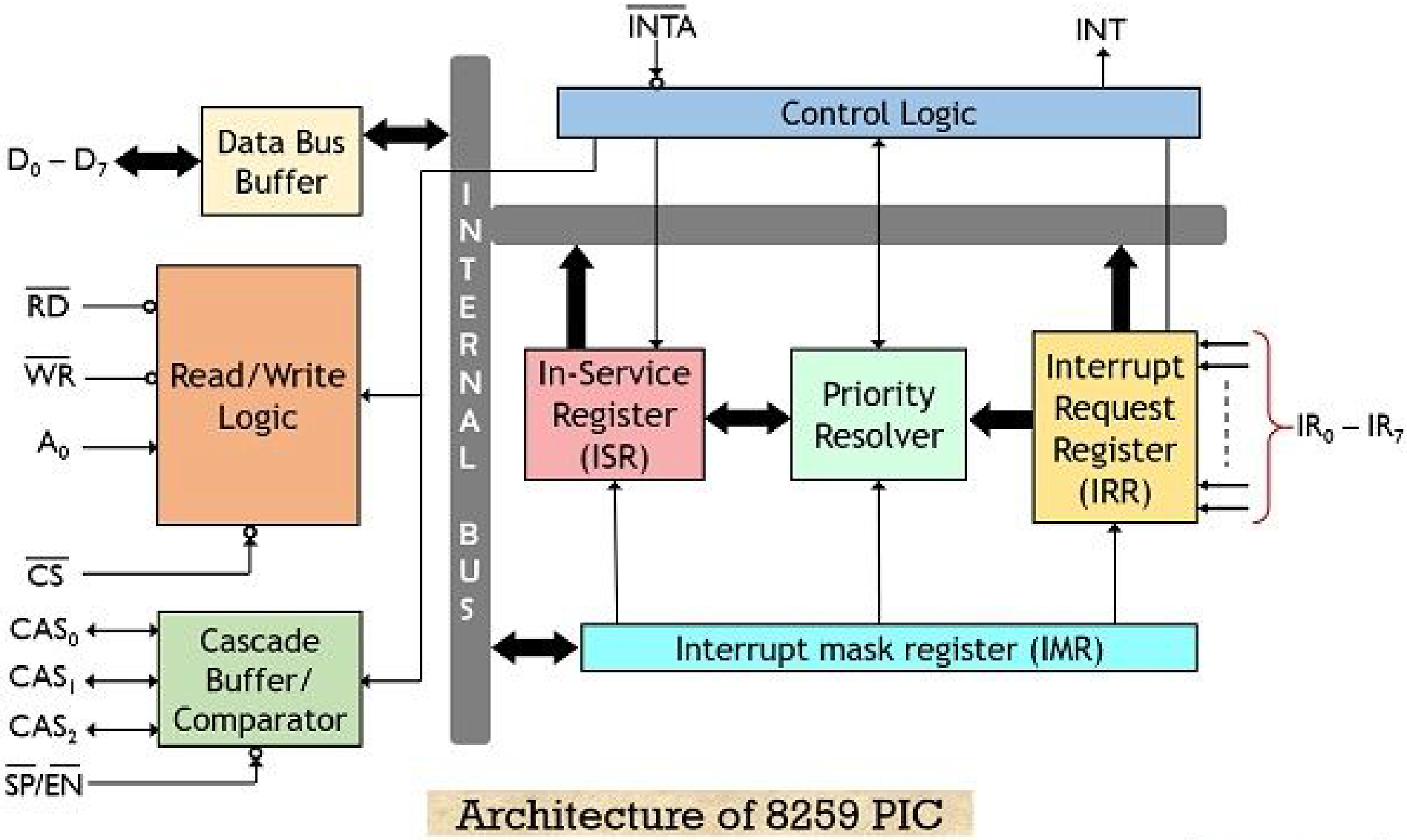
- It is also known as a **priority interrupt controller** and was designed by Intel to increase the interrupt handling ability of the microprocessor.
- The 8259A is a programmable interrupt controller designed to work with Intel microprocessors 8085, 8086, and 8088.
- An **8259 PIC** never services an interrupt; it simply forwards the interrupt to the processor for the execution of interrupt service routine.
- The advanced version of **8259 PIC is 8259A PIC** which can be used with Intel's 8086/88 16-bit Microprocessor.

Features of 8259

- The 8259A is a programmable interrupt controller designed to work with Intel micro-processors 8085, 8086, and 8088The 8259A interrupt controller can
  - manage eight interrupts according to the instructions written into its control registers. This is equivalent to providing eight interrupt pins on the processor in place of one INTR (8085) pin.
  - vector an interrupt request anywhere in the memory map. However, all eight interrupts are spaced at the interval of either four or eight locations. This eliminates the major drawback of the 8085 interrupts in which all interrupts are vectored to memory locations on page 00H.
  - resolve eight levels of interrupt priorities in a variety of modes, such as fully nested mode, automatic rotation mode, and specific rotation mode (to be explained later).
  - mask each interrupt request individually.
  - read the status of pending interrupts, in-service interrupts, and masked interrupts.
  - be set up to accept either the level-triggered or the edge-triggered interrupt request.
  - be expanded to 64 priority levels by cascading additional 8259As.
  - be set up to work with either the 8085 microprocessor mode or the 8086/8088 microprocessor mode.

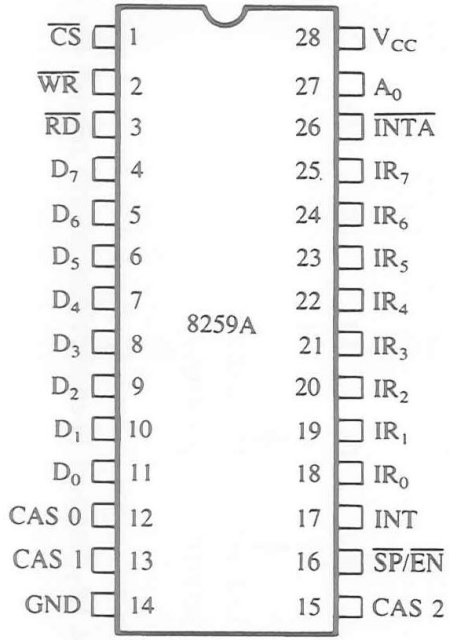
Block Diagram of 8259/8259A

- The figure shows the architectural representation of 8259 programmable interrupt controller:



Architecture of 8259 PIC

Pin Configuration



Pin Names

D_7–D_0	Data Bus (Bidirectional)
$\overline{RD}$	Read Input
$\overline{WR}$	Write Input
A_0	Command Select Address
$\overline{CS}$	Chip Select
CAS_2–CAS_0	Cascade Lines
$\overline{SP}/\overline{EN}$	Slave Program/Enable Buffer
INT	Interrupt Output
$\overline{INTA}$	Interrupt Acknowledge Input
IR_0–IR_7	Interrupt Request Inputs

Block Diagram

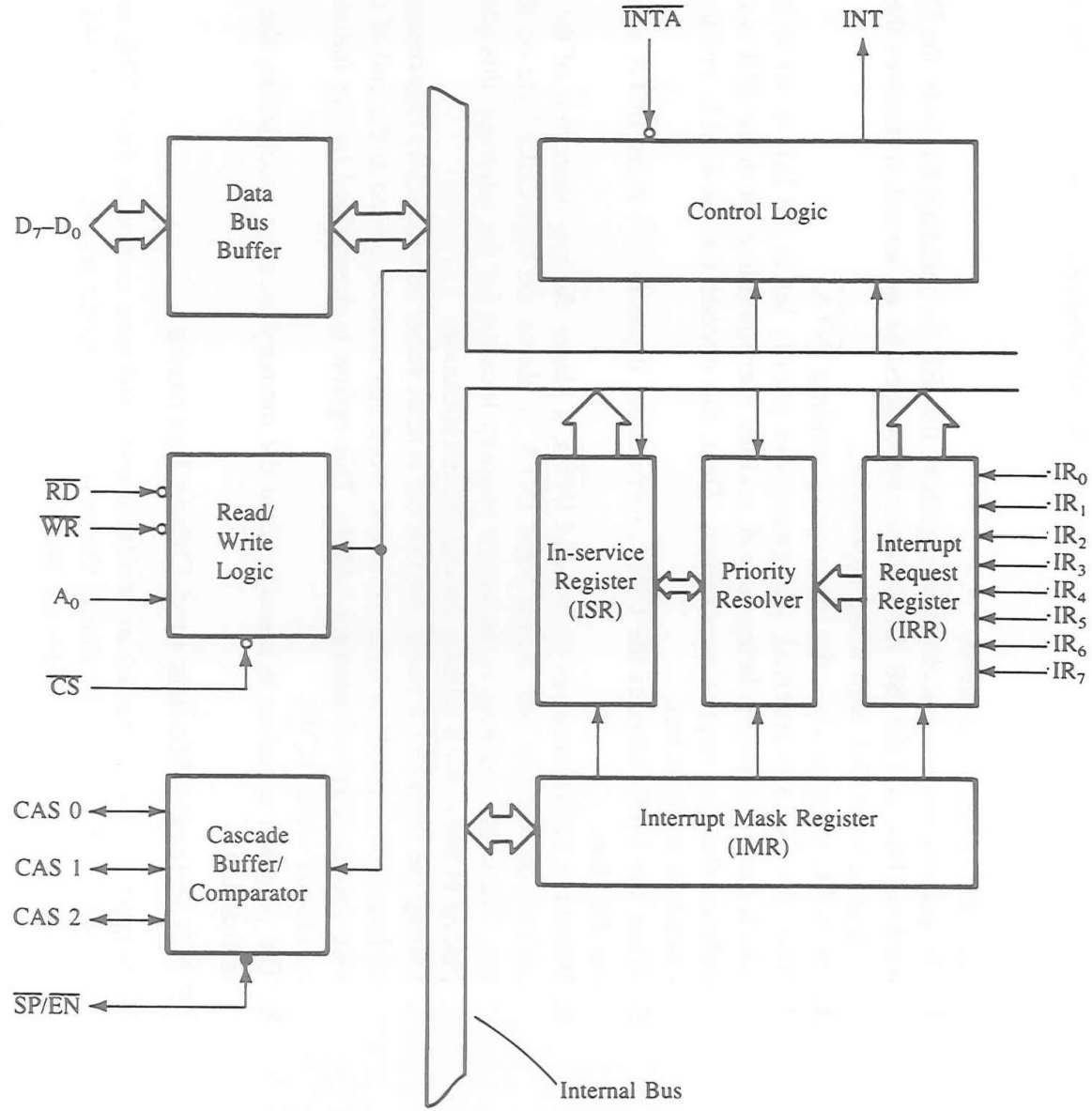


FIGURE 15.29
The 8259A Block Diagram

Block Diagram of the 8259A

- Figure in previous slide shows the internal block diagram of the 8259A. It includes eight blocks: **control logic, Read/Write logic, data bus buffer, three registers (IRR, ISR, and IMR), priority resolver, and cascade buffer**.
- This diagram shows all the elements of a programmable device, plus additional blocks.
- The functions of some of these blocks is given below.

1. **READ/WRITE LOGIC:** This is a typical Read/Write control logic. When the address line A_0 is at logic 0, the controller is selected to write a command or read a status. The Chip Select logic and A_0 determine the port address of the controller.

2. **CONTROL LOGIC:** This block has two pins: INT (Interrupt) as an output, and $\overline{\text{INTA}}$ (Interrupt Acknowledge) as an input. The INT is connected to the interrupt pin of the MPU. Whenever a valid interrupt is asserted, this signal goes high. The $\overline{\text{INTA}}$ is the Interrupt Acknowledge signal from the MPU.

Block Diagram of the 8259A

3. **INTERRUPT REGISTERS AND PRIORITY RESOLVER:** The Interrupt Request Register (IRR) has eight input lines (IR0-IR7) for interrupts. When these lines go high, the requests are stored in the register. The In-Service Register (ISR) stores all the levels that are currently being serviced, and the Interrupt Mask Register (IMR) stores the masking bits of the interrupt lines to be masked. The Priority Resolver (PR) examines these three registers and determines whether INT should be sent to the MPU.
4. **CASCADE BUFFER/COMPARATOR:** This block is used to expand the number of interrupt levels by cascading two or more 8259As.

Block Diagram of the 8259A

- It has an 8-bit of data bus. As we have already discussed that 8259 never services the interrupt, it simply forwards the interrupts to the microprocessor.
- Thus, the above architecture has different units that combinedly functions to increase the interrupts handled by the processor.
- Let us understand the operation performed by each unit in detail:

1) **Data Bus Buffer:** 8259 has tri-stated bidirectional 8-bit data bus buffer (i.e., D0 to D7) that interfaces with the internal bus of the processor. The 8085 microprocessor sends/ receives, control or status words to/ from the 8259 using data bus buffer.
2) **Read/ Write Logic:**
   - This unit is responsible for controlling the internal read-write operations of the system. It holds initialization command word register and operation command word register inside which various control formats exist that are needed for the device operation.
   - $\overline{RD}$, $\overline{WR}$, A0 and $\overline{CS}$ are the pins that are associated with this unit. Basically, these pins are used by the processor for read and write operations.
   - A low signal at $\overline{CS}$ i.e., chip select shows that now the communication has been set up between the processor and 8259.

Block Diagram of the 8259A

3) Control Logic:

- This unit is the heart of the architecture of 8259. It controls the overall operation of the system by sending the INTR signal to the processor whenever an interrupt request is generated.
- Also, it receives $\overline{\text{INTA}}$ signal by the processor when microprocessor demands for the address of the interrupt service routine. The control logic is responsible for sending the address of the desired interrupt service routine through the data bus.

4) Interrupt request register (IRR): This unit stores the interrupt requests generated by the peripheral devices. We know that 8259 has 8 interrupt request pins (i.e., IR0 to IR7). So, the unit can store 8 interrupt requests that are requesting the service from the processor.

5) Priority Resolver:

- This logic unit decides that among all the interrupt request present in the IRR which holds the highest priority and needs to be executed first.
- Suppose at the time of servicing an interrupt, another incoming interrupt request gets generated then that request will be ignored as the one in-service is holding the highest priority.
- But in case the incoming request has greater priority than the one which is being in current execution then that respective bit will be set in ISR and INTR signal is sent to the microprocessor. This simply means that only the interrupt holding the highest priority will be forwarded by the 8259 to the processor.

Block Diagram of the 8259A

6) **In-service register:**
   - Here the name of the unit is itself indicating the operation performed by it. This register unit stores the interrupts which are currently being executed by the processor.
   - The priority resolver sets each bit of ISR and after getting interrupt word command by the processor, the bits get reset. As the processor holds the ability to directly read the status of in-service register.
7) **Interrupt mask register:** This register unit holds the masking bit of those interrupts which are to be masked. Through operation command word (OCW) the processor sends the required information and programs the interrupt mask register.
8) **Cascade buffer/comparator:** As we have already discussed that by cascading multiple 8259, the number of interrupts handled by 8259 can be expanded up to 64. The unit allows the comparison of IDs of different 8259s cascaded together.

8259A Interrupt Operation

- To implement interrupts, the Interrupt Enable flip-flop in the microprocessor should be enabled by writing the EI instruction, and the 8259A should be initialized by writing control words in the control register.
- The 8259A requires two types of control words: **Initialization Command Words (ICWs)** and **Operational Command Words (OCWs)**.
- The ICWs are used to set up the proper conditions and specify RST vector addresses.
- The OCWs are used to perform functions such as masking interrupts, setting up status-read operations, etc.
- After the 8259A is initialized, the following sequence of events occurs when one or more interrupt request lines go high:

8259A Interrupt Operation

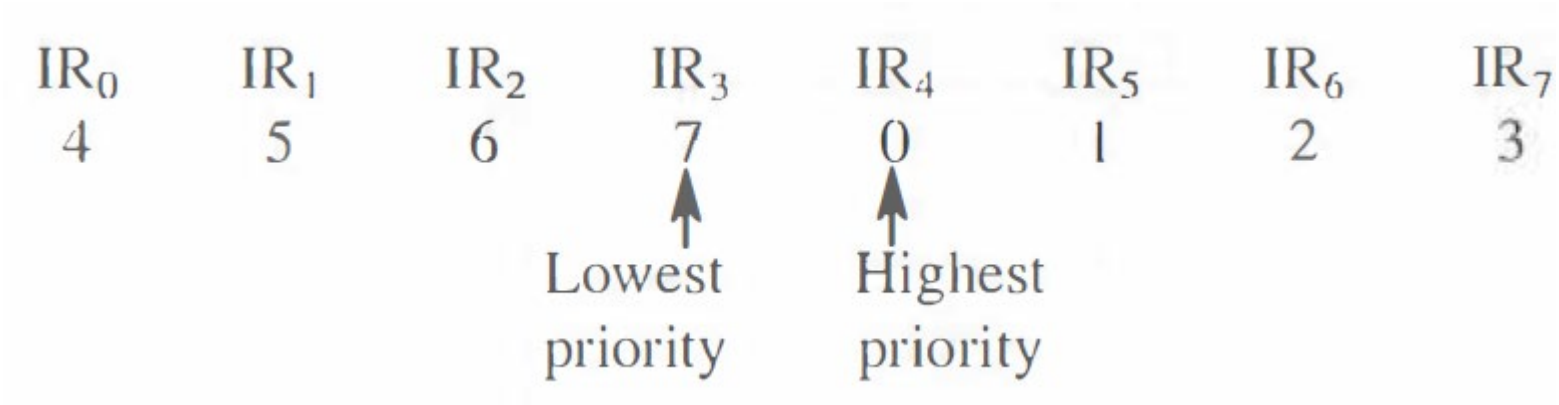
1. The IRR stores the requests.
2. The priority resolver checks three registers: the **IRR** for interrupt requests, the **IMR** for masking bits, and the **ISR** for the interrupt request being served. It resolves the priority and sets the INT high when appropriate.
3. The MPU acknowledges the interrupt by sending $\overline{\text{INTA}}$.
4. After the $\overline{\text{INTA}}$ is received, the appropriate priority bit in the ISR is set to indicate which interrupt level is being served, and the corresponding bit in the IRR is reset to indicate that the request is accepted. Then, the opcode for the CALL instruction is placed on the data bus.
5. When the MPU decodes the CALL instruction, it places two more $\overline{\text{INTA}}$ signals on the data bus.
6. When the 8259A receives the second $\overline{\text{INTA}}$, it places the low-order byte of the CALL address on the data bus. At the third $\overline{\text{INTA}}$, it places the high-order byte on the data bus. The CALL address is the vector memory location for the interrupt; this address is placed in the **control register** during the initialization.
7. During the third $\overline{\text{INTA}}$ pulse, the ISR bit is reset either automatically (Automatic-End of-Interrupt — AEOI) or by a command word that must be issued at the end of the service routine (End-of-Interrupt-EOI). This option is determined by the **initialization command word (ICW)**.
8. The program sequence is transferred to the memory location specified by the CALL instruction.

Priority Modes

- Many types of priority modes are available under software control in the 8259A, and they can be changed dynamically during the program by writing appropriate command words.

1. Fully Nested Mode:

- This is a general-purpose mode in which all IRs (Interrupt Requests) are arranged from highest to lowest, with IR0 as the highest and IR7 as the lowest.
- In addition, any IR can be assigned the highest priority in this mode; the priority sequence will then begin at that IR.
- In the example below, IR4 has the highest priority, and IR3 has the lowest priority:



Priority Modes

2. **Automatic Rotation Mode:** In this mode, a device, after being serviced, receives the lowest priority. Assuming that the **IR_2** has just been serviced, it will receive the seventh priority, as shown below:

IR_0	IR_1	IR_2	IR_3	IR_4	IR_5	IR_6	IR_7
1	6	7	0	1	2	3	4

3. **Specific Rotation Mode:** This mode is similar to the automatic rotation mode, except that the user can select any IR for the lowest priority, thus fixing all other priorities.

End of Interrupt

- After the completion of an interrupt service, the corresponding ISR bit needs to be reset to update the information in the ISR. This is called the **End-of-Interrupt (EOI)** command.
- It can be issued in three formats:
  1. **Nonspecific EOI Command** When this command is sent to the 8259A, it resets the highest priority ISR bit.
  2. **Specific EOI Command** This command specifies which ISR bit to reset.
  3. **Automatic EOI** In this mode, no command is necessary. During the third $\overline{\text{INTA}}$, the ISR bit is reset. The major drawback with this mode is that the ISR does not have information on which IR is being serviced. Thus, any IR can interrupt the service routine, irrespective of its priority, if the Interrupt Enable flip-flop is set.

Additional Features

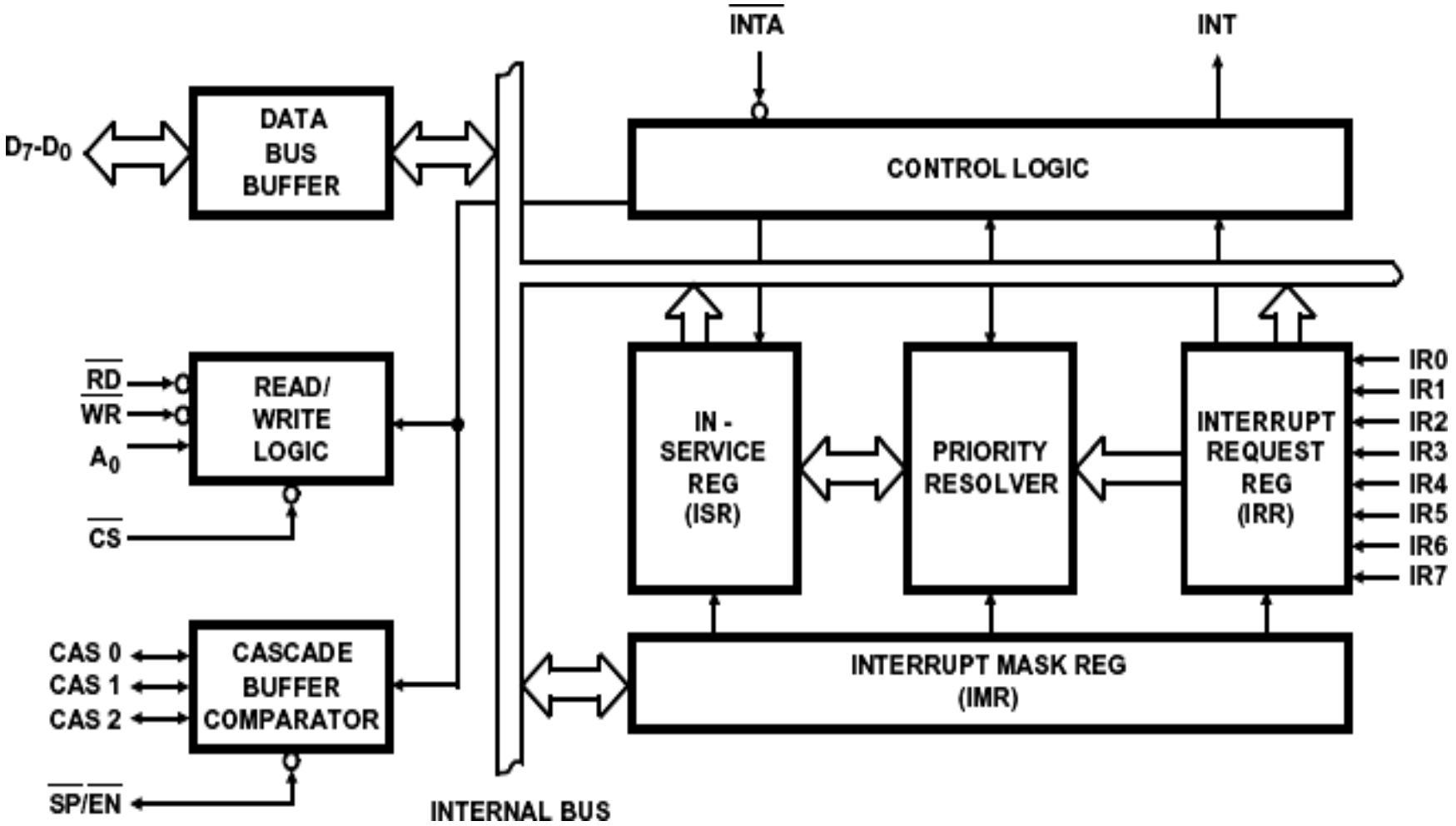
- The 8259A is a complex device with various modes of operation. These modes are listed below for reference;
  - **Interrupt Triggering:** The 8259A can accept an interrupt request with either the edge triggered mode or the level-triggered mode. The mode is determined by the initialization instructions.
  - **Interrupt Status:** The status of the three interrupt registers (IRR, ISR, and IMR) can be read, and this status information can be used to make the interrupt process versatile.
  - **Poll Method:** The 8259A can be set up to function in a polled environment. The MPU polls the 8259A rather than each peripheral.

8259 PROGRAMMABLE INTERRUPT CONTROLLER

Salient Features of 8259 PIC

- 8259 is a **Programmable Interrupt Controller** (PIC) designed to work with **8085**, **8086** etc.
- A **single 8259** can handle **8 interrupts,** while a **cascaded** configuration of 8 slave 8259's and 1 master 8259 can handle **64 interrupts**.
- 8259 can **handle edge** as well as **level triggered** interrupts.
- 8259 has a **flexible priority** structure.
- In 8259 interrupts can be **masked** individually.
- The **Vector address** of the interrupts is **programmable**.
- Status of interrupts (pending, In-service, masked) can be easily read by the µP.

Architecture of 8259



Architecture of 8259

The architecture of 8259 can be divided into the following parts:

1) Interrupt Request Register (IRR)

– 8259 has **8 interrupt** input lines **IR7 … IR0**.

– The IRR is an **8-bit register** having **one bit** for **each** of the **interrupt** lines.

– When an **interrupt request** occurs on any of these lines, the **corresponding bit** is **set** in the Interrupt Request Register (**IRR**).

2) In-Service Register (InSR)

– It is an **8-bit** register, which **stores** the **level** of the Interrupt Request, which is **currently** being **serviced**.

3) Interrupt Mask Register (IMR)

– It is an **8-bit** register, which stores the **masking pattern** for the interrupts of 8259.

It stores **one bit per interrupt level**.

Architecture of 8259

4) Priority Resolver

– It **examines** the **IRR**, **InSR**, and **IMR** and determines which interrupt is of **highest priority** and should be sent to the µP.

5) Control Logic

– It has **INT output** signal **connected to** the **INTR** of the µP, to **send** the **Interrupt** to the µP.

– It also has the $\overline{\textbf{INTA}}$ **input** signal **connected to** the $\overline{\textbf{INTA}}$ of the µP, to **receive** the interrupt **acknowledge**.

– It is also used to control the remaining blocks.

6) Data Bus Buffer

– It is a bi-directional buffer used to **interface** the internal **data bus** of 8259 with the external(system) data bus.

Architecture of 8259

7) Read/Write Logic

– It is used to accept the $\overline{RD}$,$\overline{WR}$, A_0 and $\overline{CS}$ signal.

– It also holds the Initialization Command Words (ICW's) and the Operational Command Words (OCW's).

8) Cascade Buffer / Comparator

– It is used in **cascaded mode** of operation.

– It has two components:

i. CAS2, CAS1, CAS0 lines:

- ❑ These lines are **output for the master**, **input for the slave**.
- ❑ The **Master sends** the **address of** the **slave** on these lines (hence output).
- ❑ The **Salves read** the **address** on these lines (hence input).
- ❑ As there are 8 interrupt levels for the Master, there are **3 CAS lines (∵ $2^3 = 8$).**

Architecture of 8259

8) Cascade Buffer / Comparator

ii. $\overline{\text{SP}}/\overline{\text{EN}}$ (Slave Program/Master Enable):

- In **Buffered Mode**, it **functions** as the **$\overline{\text{EN}}$ line** and is used to **enable** the **buffer**.
- In **Non buffered mode**, it **functions** as the **$\overline{\text{SP}}$ output line**.
- **For Master** 8259 **$\overline{\text{SP}}$** should be **high**, and **for** the **Slave $\overline{\text{SP}}$** should be **low**.

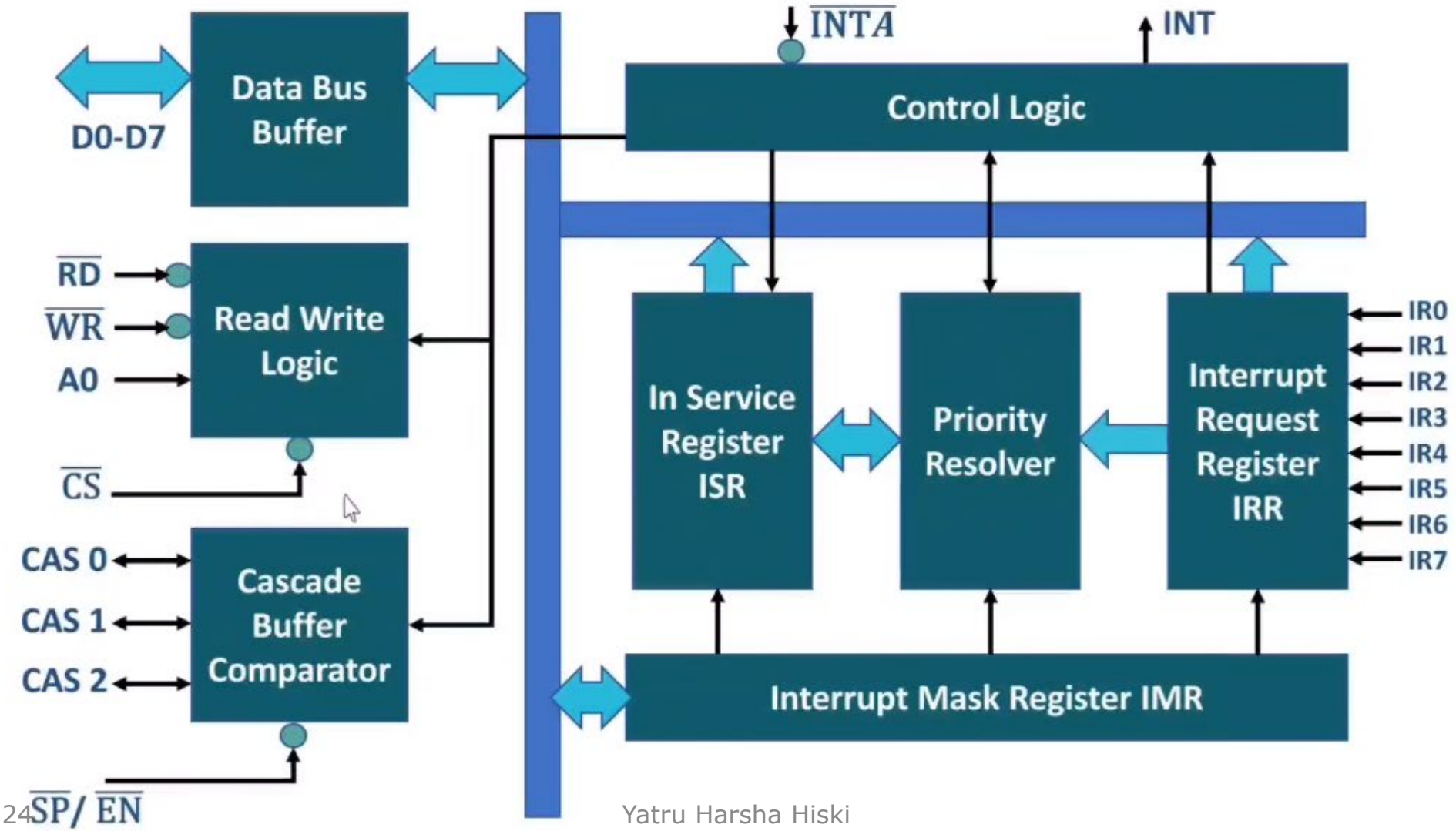
8259 Programmable Interrupt Controller

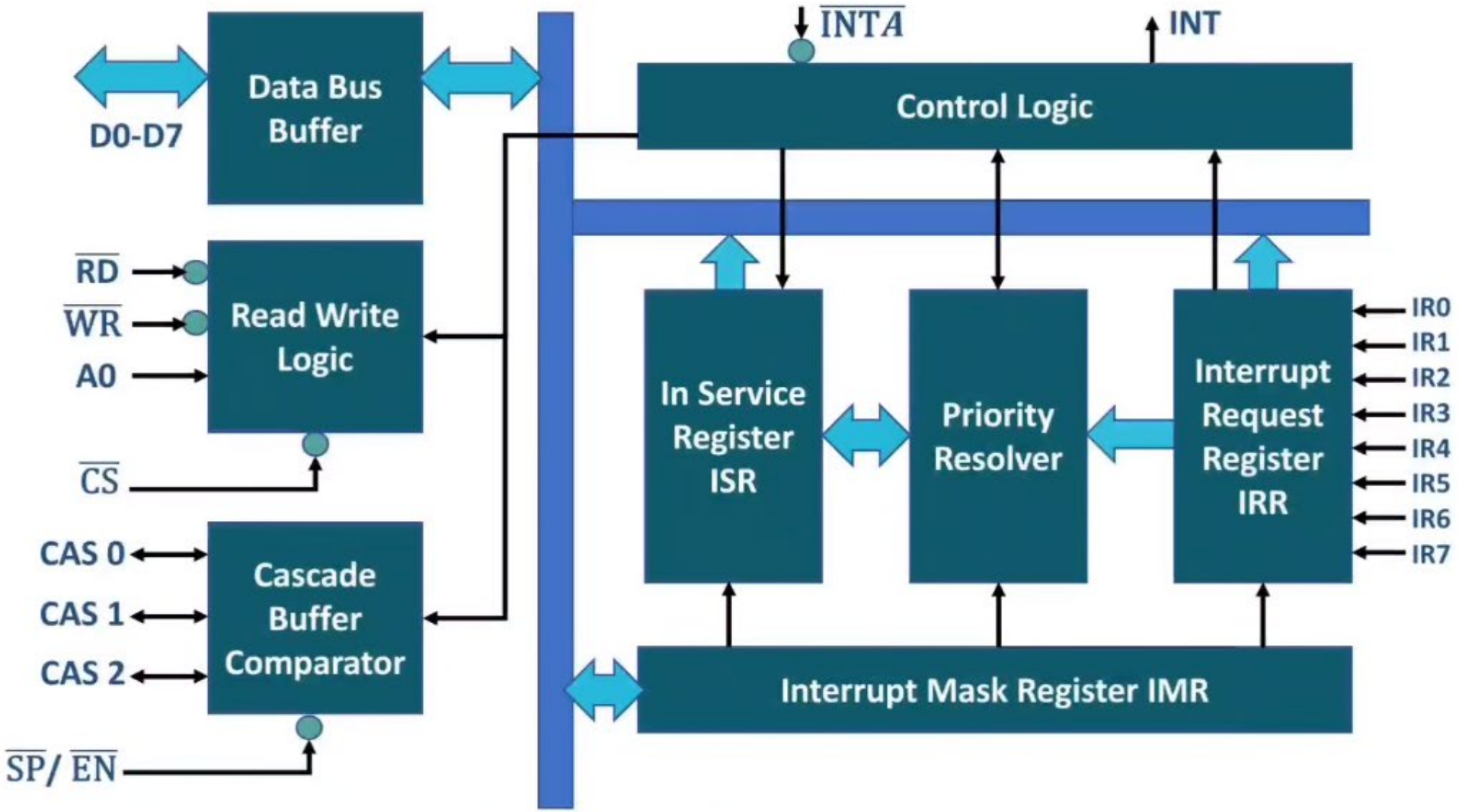
❖ Features of 8259 Programmable Interrupt Controller

- ❑ 8259 is designed to work with various microprocessors like 8085, 8086 etc.
- ❑ 8259 is designed to increase capacity of interrupts.
- ❑ 8259 can handle 8 interrupts with single IC.
- ❑ A cascade connection of 8259 can handle 64 interrupts.
  - One Master 8259 can works with eight Slave 8259, so total capacity will be 64.
- ❑ 8259 has flexible interrupt priority structure.
- ❑ Using 8259, we can mask interrupt as well.
- ❑ The vector address of 8259 is programmable.
- ❑ Status of interrupt can be observed by microprocessor :
  - Pending status
  - In service status
  - Masked status

8259 Programmable Interrupt Controller

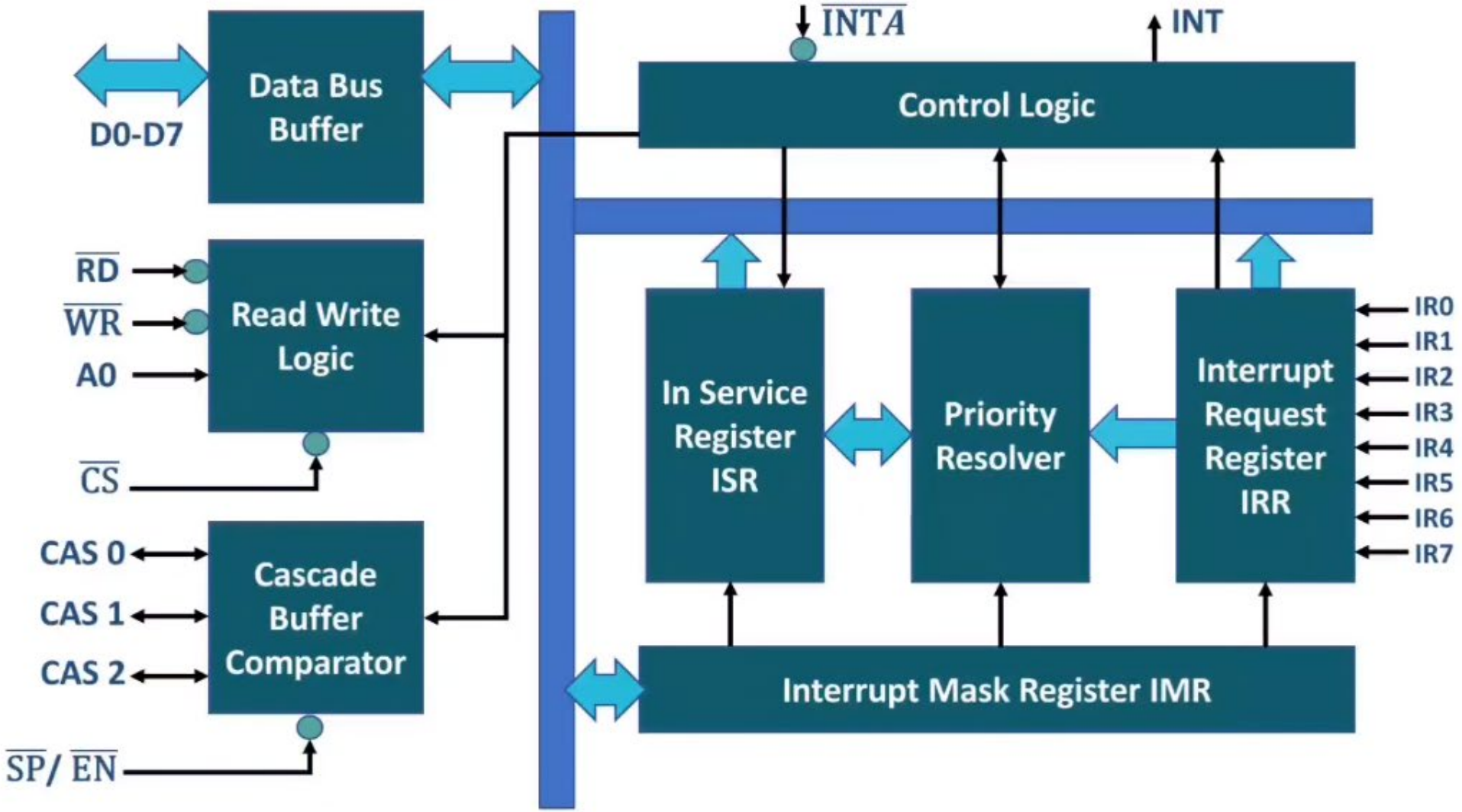
❖ Block Diagram / Architecture of 8259





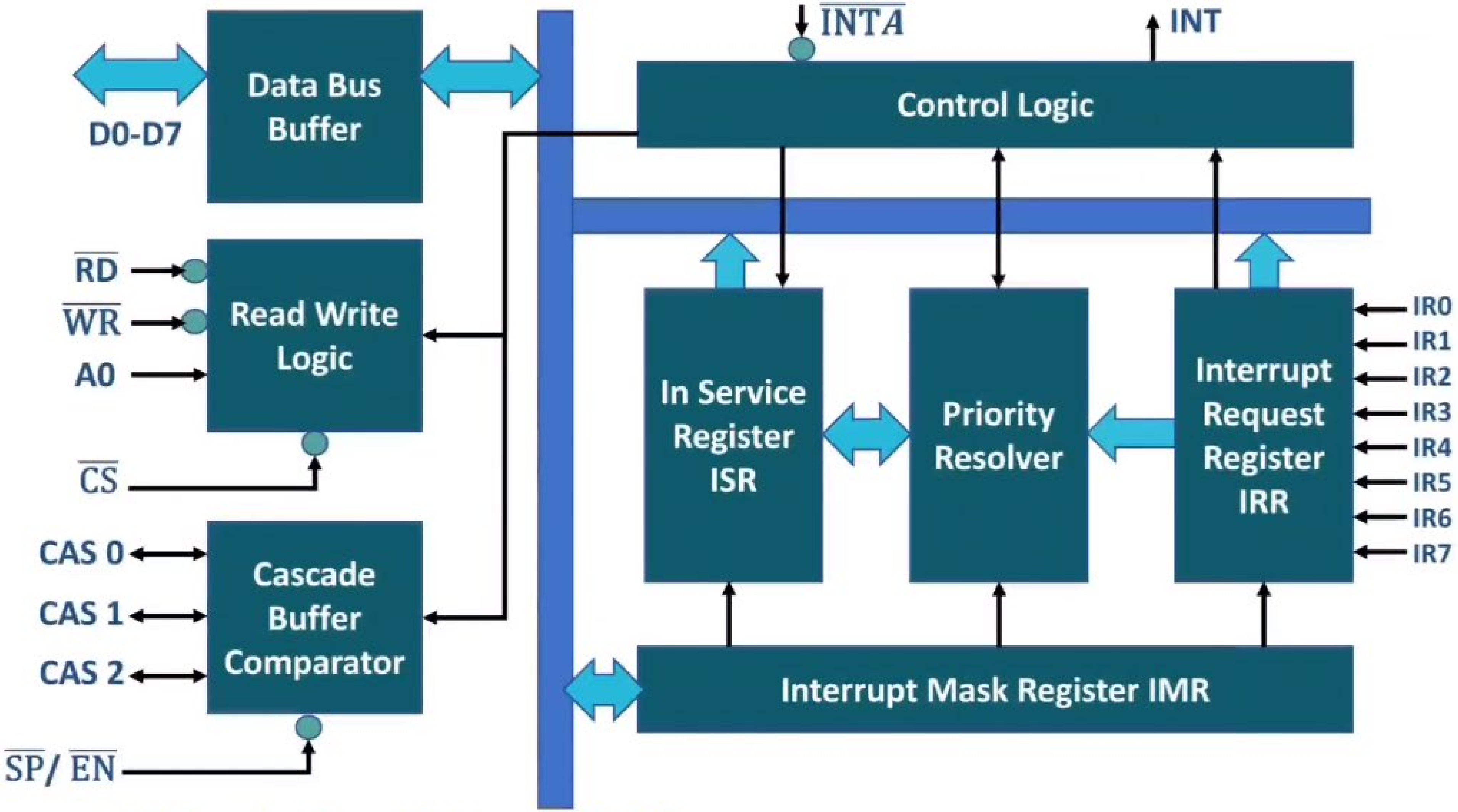
❖ Interrupt Request Register - IRR

- ❑ Here, we have 8 interrupt lines IR7 – IR0.
- ❑ IRR is Eight bits register, each bit stores individual interrupt.
- ❑ When interrupt occurs on any lines, corresponding bit will get set to One.



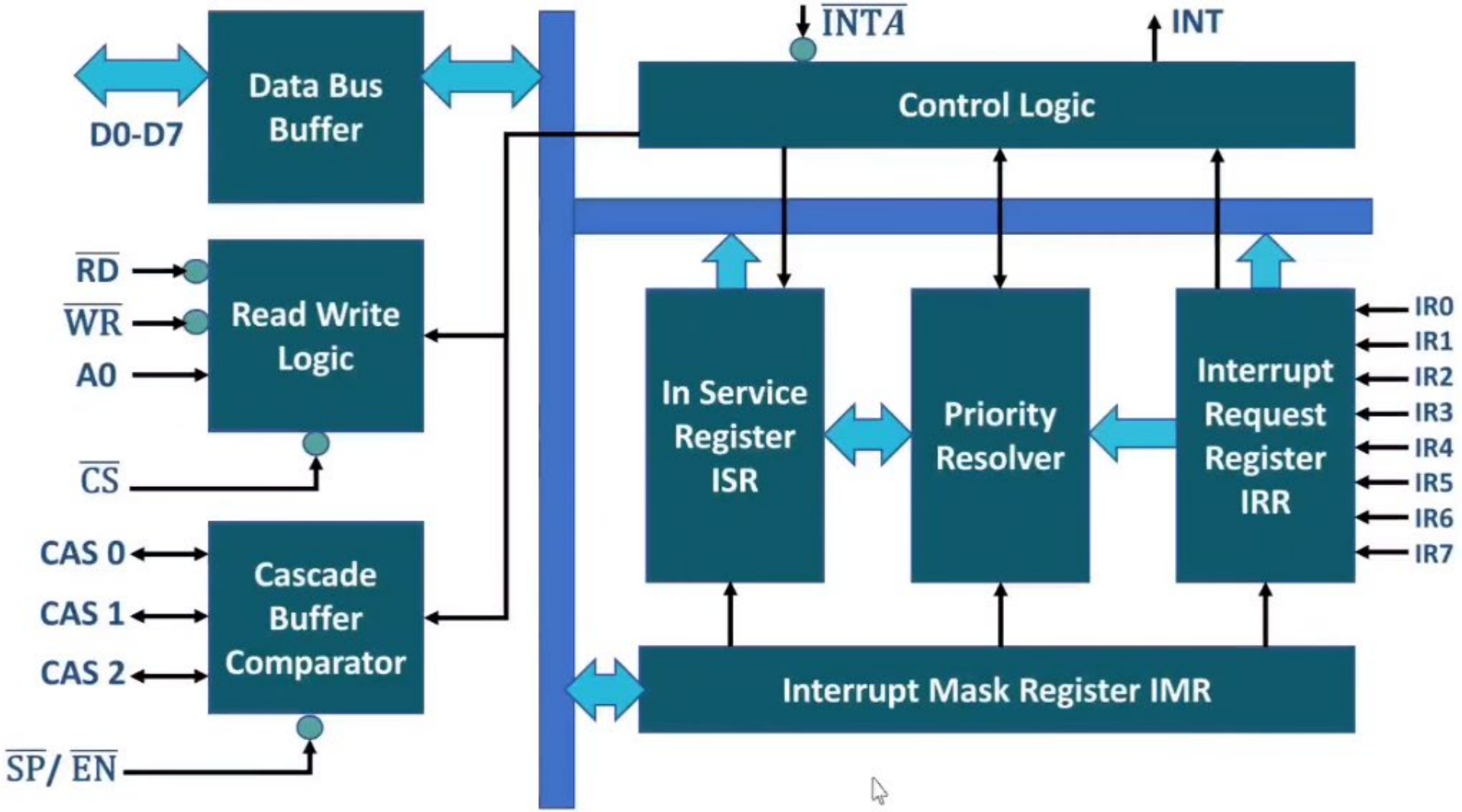
❖ In Service Register - ISR

- ❑ It is 8 bits register.
- ❑ It stores the data of currently served interrupt.



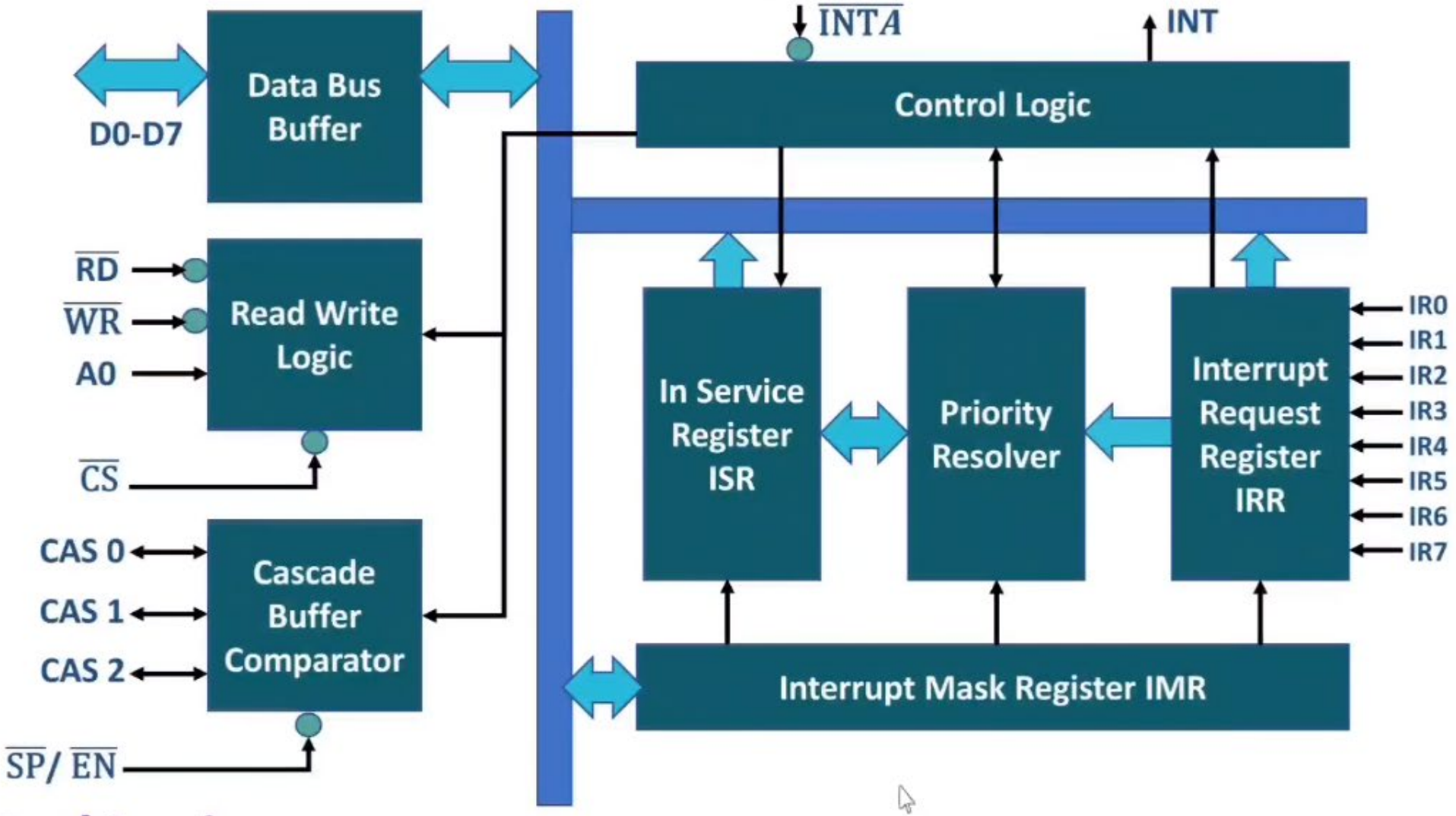
❖ Interrupt Mask Register - IMR

- ❑ It is 8 bits register.
- ❑ It stores the masking pattern of 8259.
- ❑ Each bits holds masking of individual interrupt.



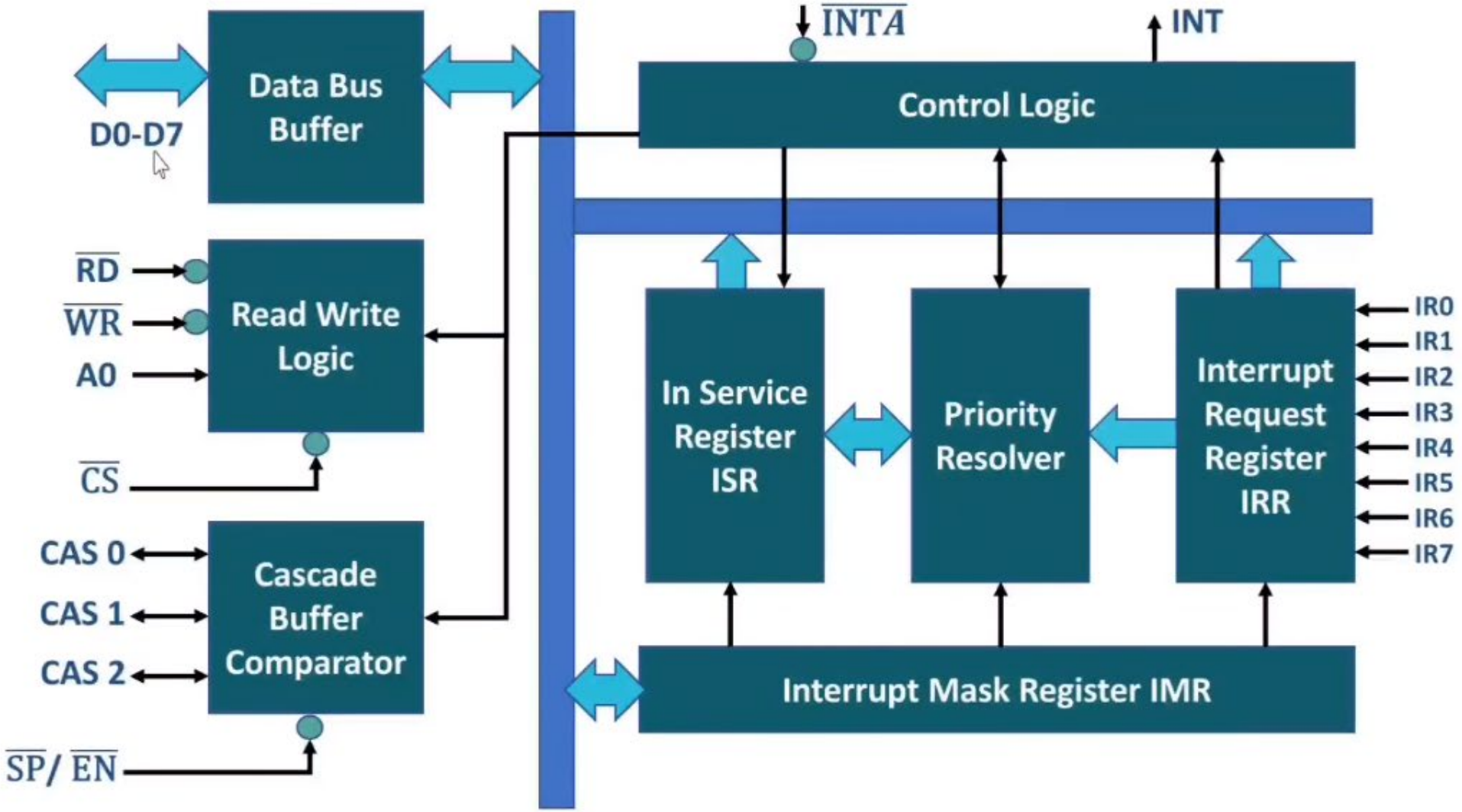
❖ Priority Resolver

- ❑ It examines IRR, ISR and IMR. Based on that determines which interrupt has maximum priority and should be sent to microprocessor.



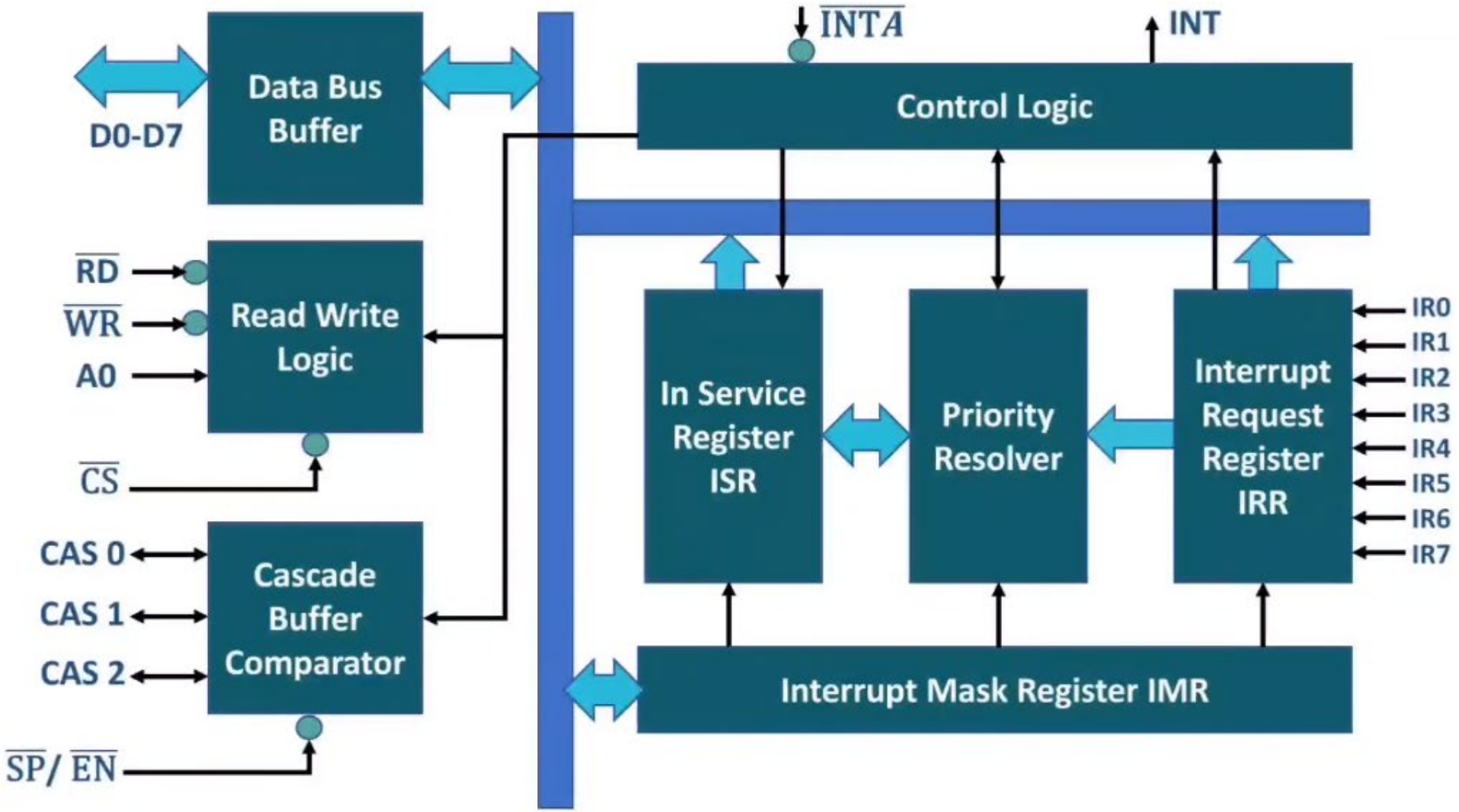
❖ Control Logic

- ❑ It has INT pin connected with INTR of microprocessor to send interrupt request.
- ❑ It has $\overline{INTA}$ pin connected with $\overline{INTA}$ of microprocessor to receive acknowledgment.
- ❑ It is also used to control remaining blocks.



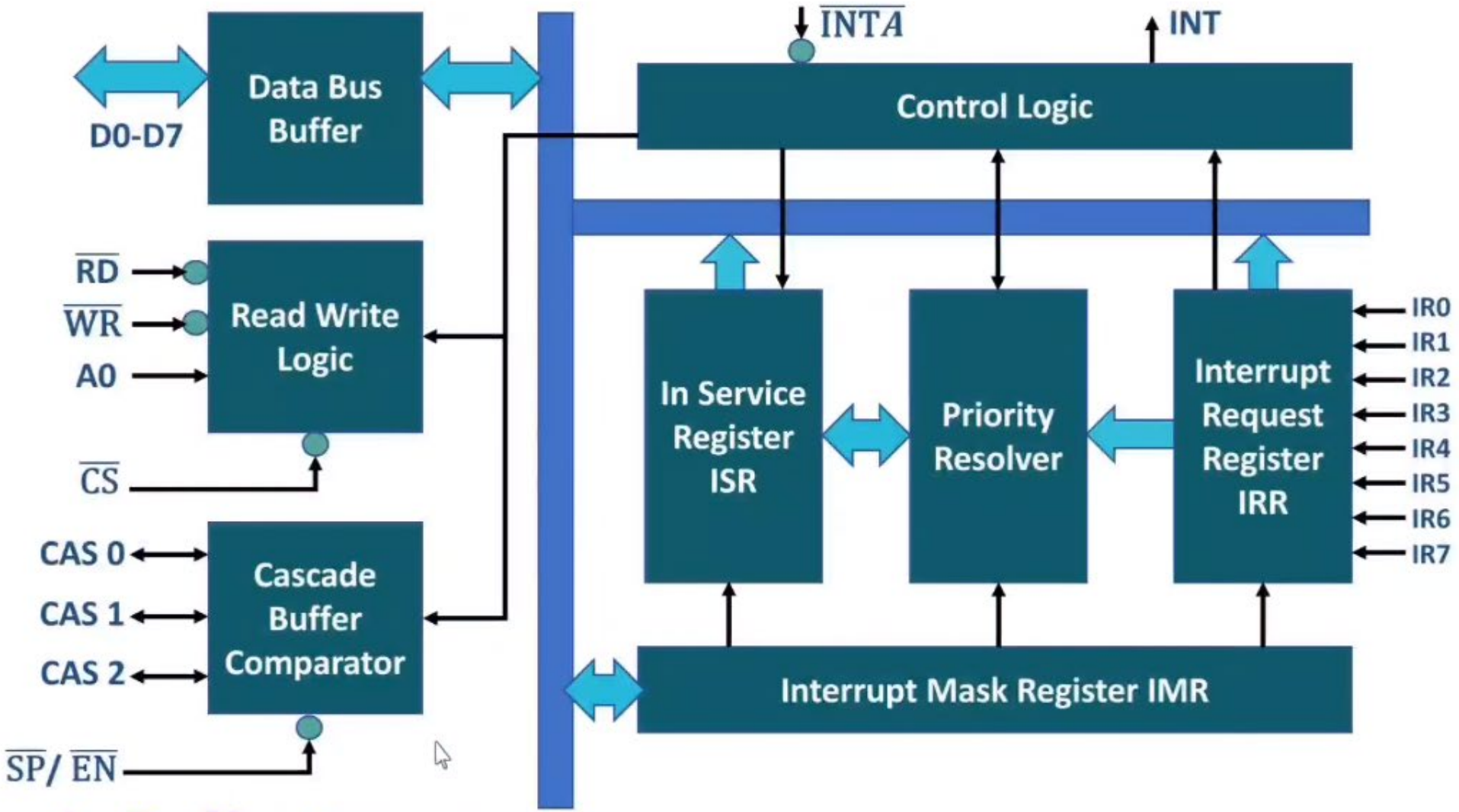
❖ Data Bus Buffer

- ❑ It has bidirectional data bus D0 – D7.
- ❑ D0 – D7 is interfaced with system data bus of Microprocessor.



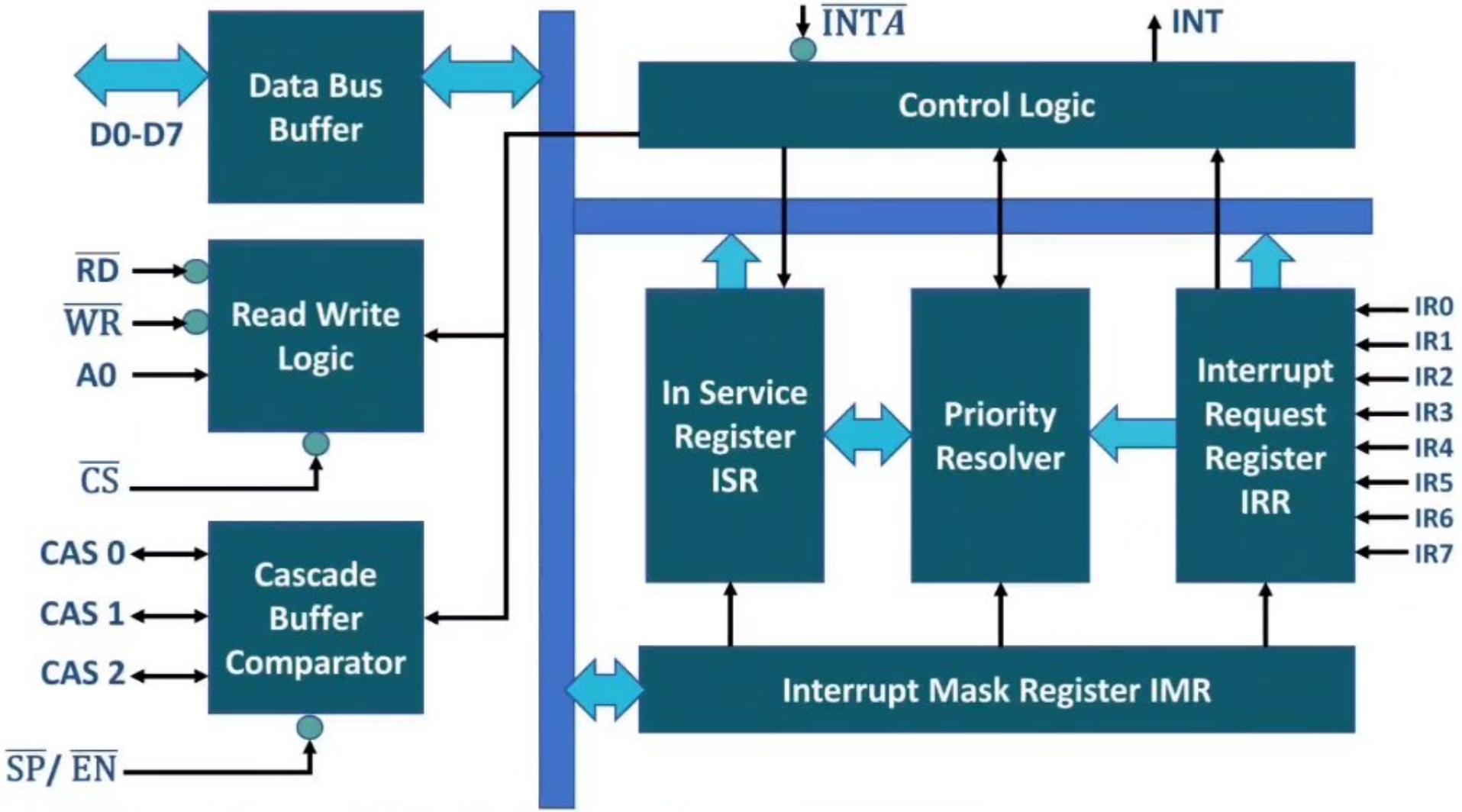
❖ Read Write Logic

- ❑ It is used to take read, write, A0 and Chip select.
- ❑ It also holds Initialization Command Words (ICW) and Operational Command Words (OCW).

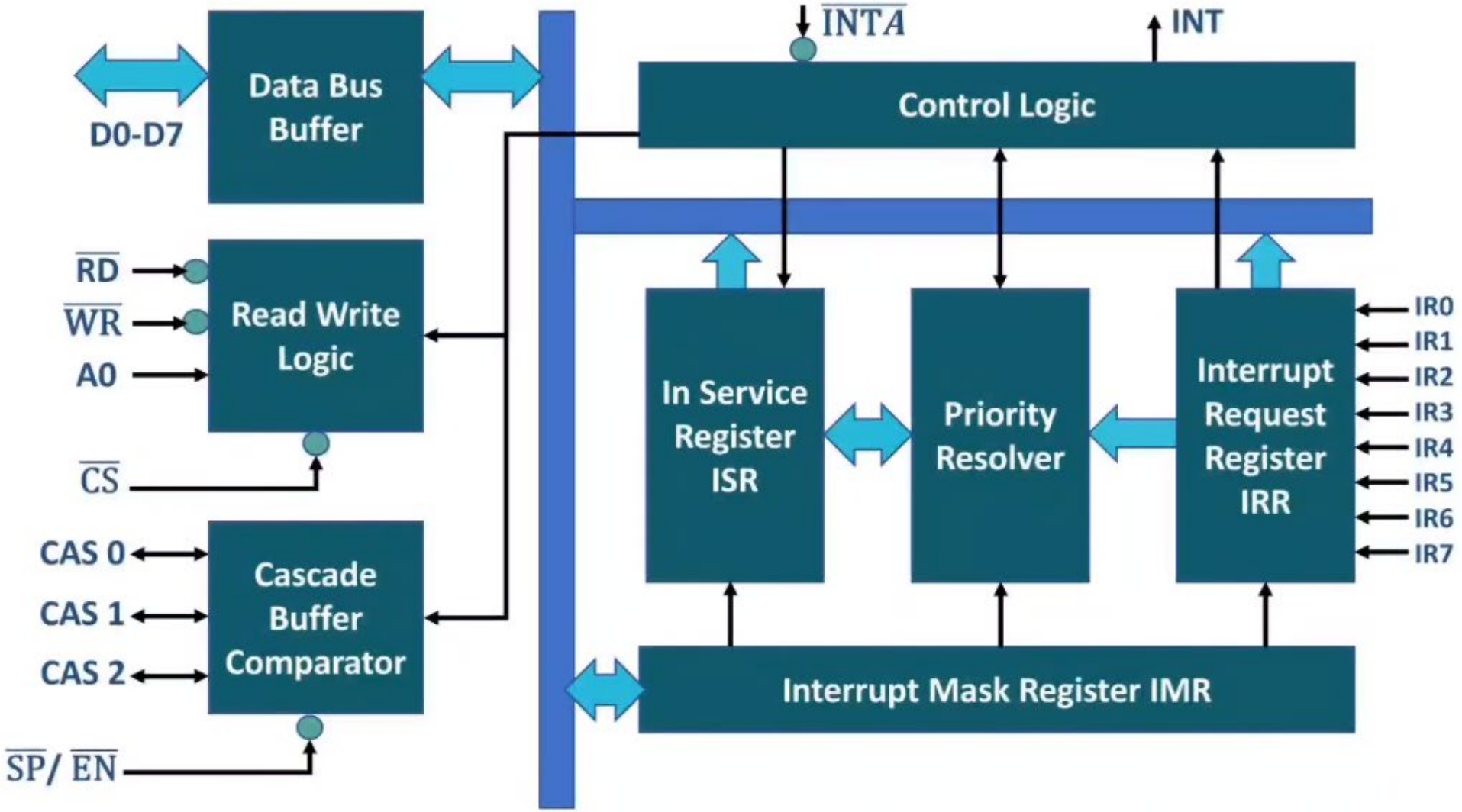


❖ Cascade Buffer Comparator

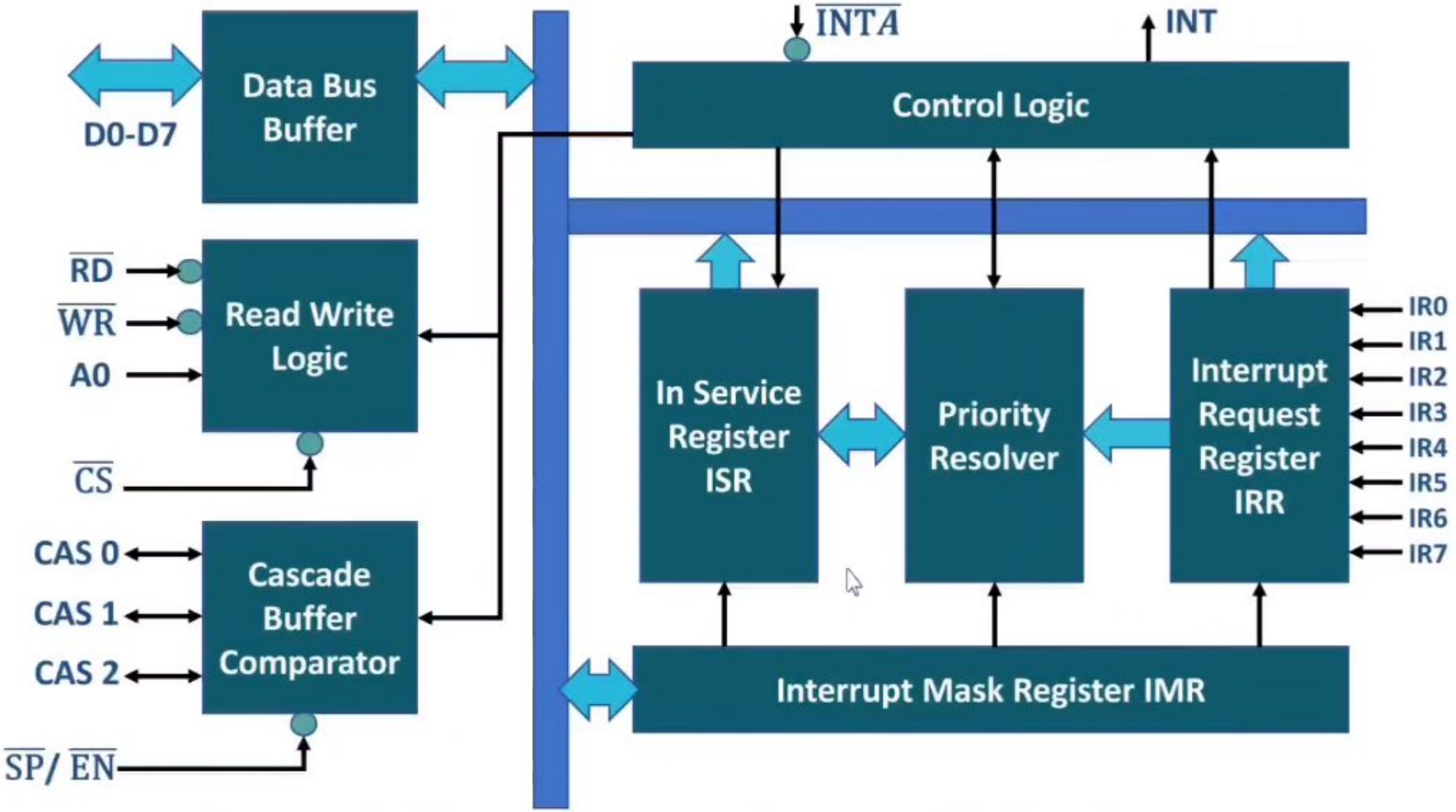
- ❑ It is used in cascade mode operation.
- ❑ It is used for Master Slave Interrupt control from multiple 8259.
- ❑ $\overline{SP}/\overline{EN}$ – It is Slave Program/ Master Enable line. In buffer Mode, it function as enable line and In non buffer mode it functions as SP enable line.



1. INTR of 8085 must be enabled with EI Instruction.
2. 8259 is initialized by necessary commands. [ICW]
3. Once 8259 is initialized, if multiple interrupts comes then corresponding bit of IRR is set to one.
4. Priority resolver checks IRR, IMR & ISR. Based on that, it will decide highest priority and then it gives signal to Microprocessor 8085 at INTR.



5. The Microprocessor completes current instruction, after that it give acknowledgement to 8259 on $\overline{INTA}$.
6. On receiving $\overline{INTA}$ from Microprocessor, ISR will set corresponding bit to one in ISR to indicate service to this interrupt is started and the bit in IRR is reset to indicate request is accepted. Now 8259 can give opcode of CALL instruction to Microprocessor.



7. The microprocessor decodes the CALL instruction and sends two more $\overline{INTA}$ to 8259.
8. In response to $\overline{INTA}$ signals, 8059 sends address of interrupt service routine. So it completes three bytes CALL instruction.
9. Now Microprocessor perform Interrupt Service Routine by pushing content of PC on stack.
10. At the end of interrupt, Microprocessor will send EOI command to 8259, that makes corresponding bit 0 in ISR of 8259.

End of Unit 5
Thank You