1. 8251 USART (Universal Synchronous Asynchronous Receiver Transmitter)

The 8251 USART (Universal Synchronous Asynchronous Receiver Transmitter) is a critical interface for serial communication between microprocessors and peripheral devices. Below is a structured and concise analysis of its architecture, functionality, and operational modes:

Purpose of 8251 USART

Role: Acts as a mediator to convert parallel data from the microprocessor into serial data for transmission to peripherals, and vice versa for reception.

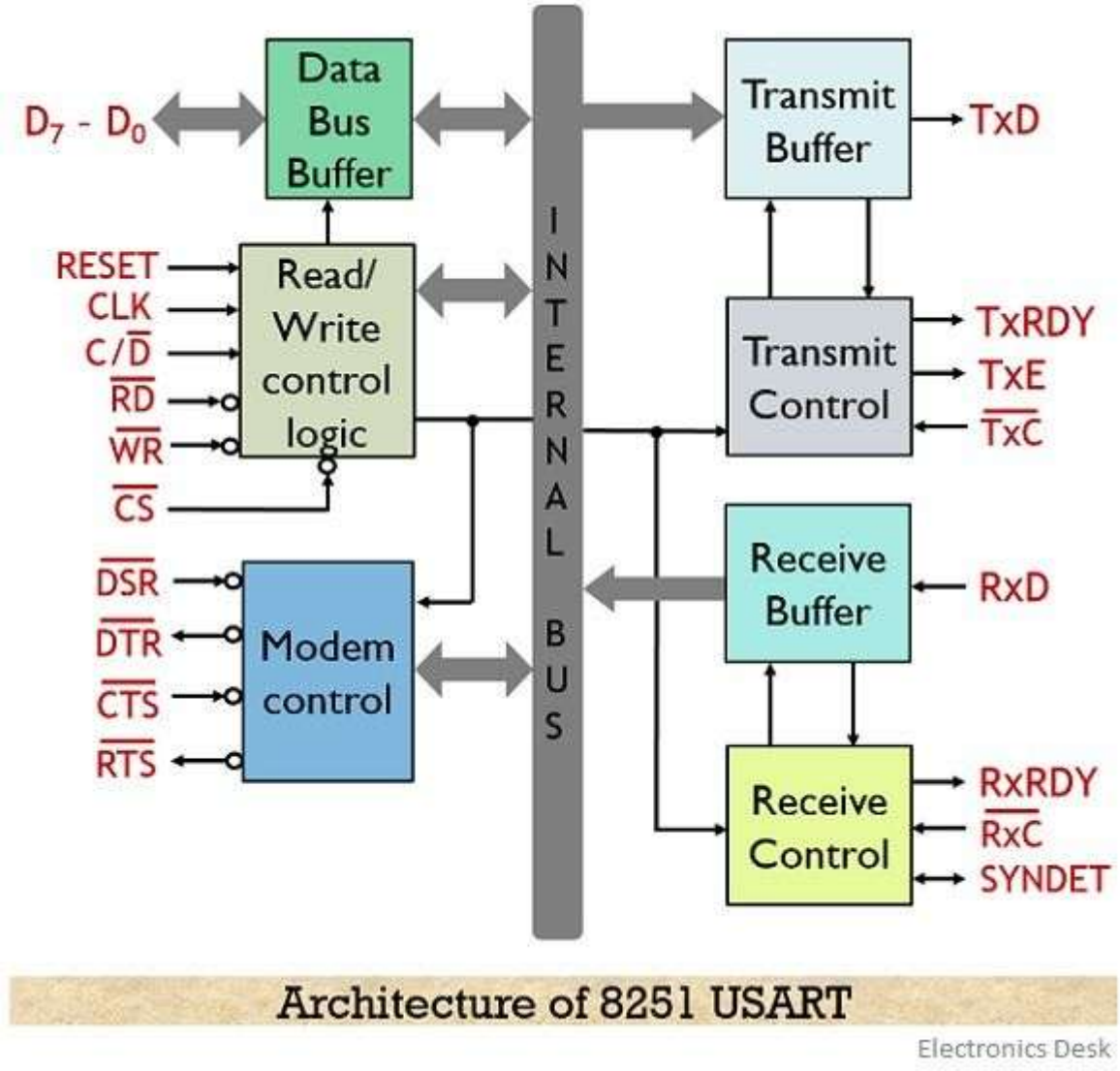
Need for Serial Communication: Reduces cost and complexity by minimizing wiring (vs. parallel communication) over long distances.

Modes Supported:

Synchronous: Continuous data transfer with clock synchronization (no start/stop bits; uses sync characters).

Asynchronous: Data framed with start/stop bits for synchronization.

Architecture of 8251 USART



Architecture of 8251 USART

Electronics Desk

The 8251 comprises 7 key functional units:

A. Data Bus Buffer

Interfaces with the CPU's data bus (D0-D7) for bidirectional data transfer (data, control words, status).

Key Operations:

- Transfers parallel data between CPU and USART.
- Holds control/status registers for configuration and monitoring.

B. Read/Write Control Logic

Decodes control signals from the CPU to manage USART operations.

Signals:

- *CS (Chip Select):* Enables communication with the USART.
- *C/D (Control/Data):* Selects control/status register (high) or data register (low).
- *RD/WR:* Read/write operations.
- *CLK/RESET:* Clock for timing; reset forces idle mode.

C. Transmit Buffer

Converts parallel data to serial format for transmission.

Components:

- *Buffer Register:* Stores parallel data from the CPU.
- *Output Register:* Converts parallel data to serial bits (sent via TxD pin).

Framing:

Asynchronous: Adds start/stop bits.

Synchronous: Uses clock and optional sync characters.

D. Transmit Control

Manages data transmission timing and readiness.

Signals:

- TxRDY (Transmit Ready): Indicates buffer is empty (ready for new data).
- TxE (Transmitter Empty): Output register is empty.
- TxC (Transmit Clock): Controls transmission rate (programmable).

E. Receive Buffer

Converts serial data (from RxD) to parallel format for the CPU.

Components:

- *Receiver Input Register:* Captures serial data and detects start bits (asynchronous) or sync characters (synchronous).
- *Buffer Register:* Stores converted parallel data for CPU retrieval.

F. Receiver Control

Manages data reception and error detection.

Signals:

- *RxRDY (Receiver Ready):* Indicates data is ready for the CPU.

- *RxC (Receiver Clock):* Controls reception rate (matches baud rate in synchronous mode).
- *Error Detection:* Parity errors, framing errors, false start bits.

G. Modem Control

Handles handshaking signals for modem/peripheral communication.

Signals (Active Low):

- *DSR (Data Set Ready):* Peripheral is ready.
- *DTR (Data Terminal Ready):* USART is ready to accept data.
- *RTS (Request to Send):* Requests permission to transmit.
- *CTS (Clear to Send):* Grants permission to transmit.

3. Working Modes

A. Asynchronous Mode

- Data Format: Start bit + data (5-8 bits) + parity (optional) + stop bit(s).
- Clock Rate: Programmable (1×, 16×, or 64× baud rate).
- Use Case: Simple, low-speed communication (e.g., keyboards).

B. **Synchronous Mode**

- Data Format: Continuous stream with sync characters (no start/stop bits).
- Clock Synchronization: Uses TxC/RxC pins for timing.
- Use Case: High-speed, continuous data transfer (e.g., networking).

Advantages

- Dual Mode Support: Handles both synchronous (continuous data with clock sync) and asynchronous (start/stop bits) communication.
- Programmable Features: Adjustable baud rate, data bits (5–8), parity (even/odd/none), and stop bits (1, 1.5, 2).
- Error Detection: Built-in checks for parity, framing (missing stop bits), and overrun (buffer overflow) errors.
- Full-Duplex Operation: Simultaneous data transmission and reception.
- Modem Control: Simplifies peripheral interfacing via DTR, DSR, RTS, CTS signals.
- Cost-Effective: Reduces wiring complexity for long-distance serial vs. parallel communication.
- Easy Integration: Direct interface with 8-bit CPUs (e.g., Intel 8085/8086).

Limitations

- Outdated Technology: Lacks modern features like FIFO buffers or DMA support.
- CPU Dependency: Requires frequent processor intervention for byte-by-byte transfers.
- Complex Configuration: Prone to errors due to manual setup of Mode and Command Word registers.
- Clock Synchronization: synchronous mode demands precise transmitter-receiver clock alignment.
- Speed Constraints: Lower maximum baud rate vs. modern standards (USB, SPI).

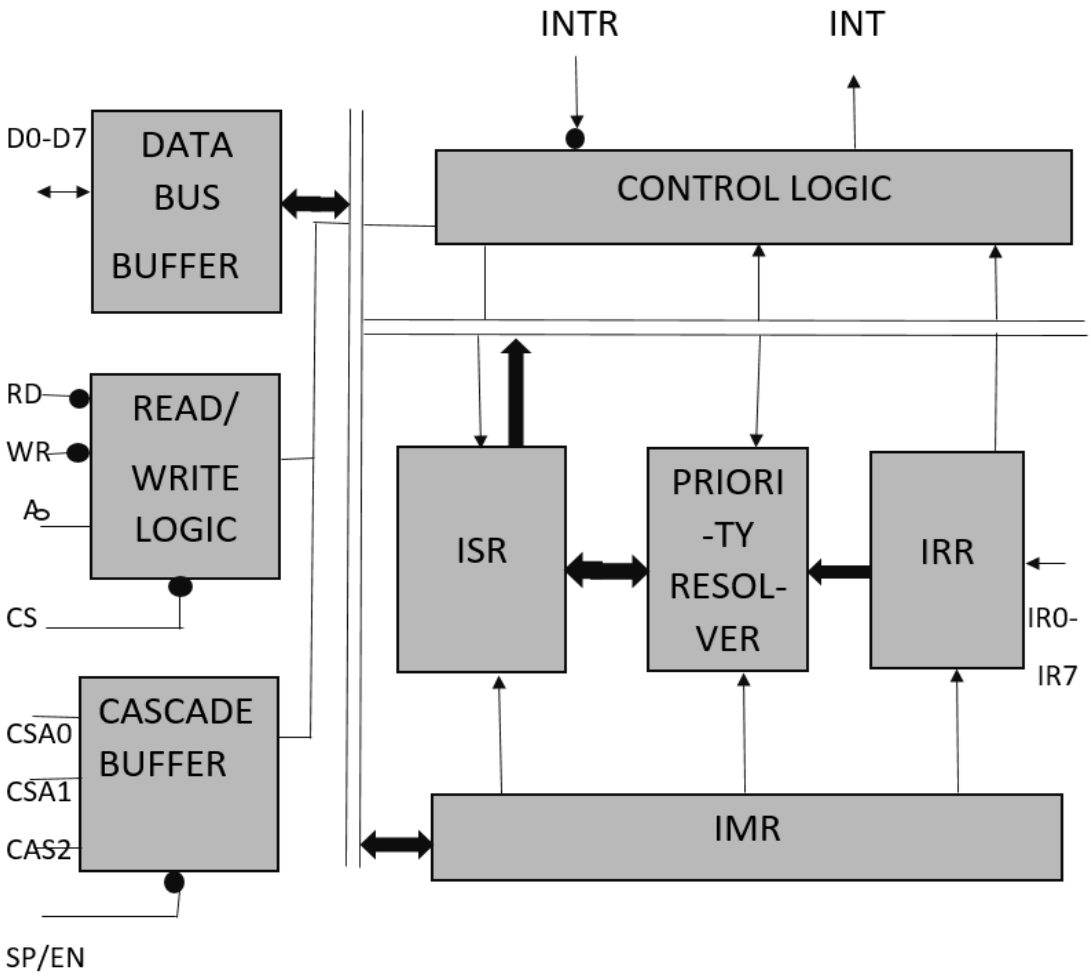
Applications

- Legacy Systems: RS-232 communication in early PCs and industrial controllers.
- Industrial Automation: PLC interfacing with sensors/actuators.
- Modem Communication: Dial-up networking and data transmission.

- Instrumentation: Data acquisition systems (e.g., oscilloscopes).
- Education: Teaching serial communication in microprocessor labs.
- Embedded Systems: Simple serial tasks (e.g., GPS modules, RFID readers).

2. 8259 PIC

The Intel 8259 is a Programmable Interrupt Controller (PIC) that manages and prioritizes multiple interrupt requests for the CPU. It supports 8 interrupt lines, expandable to 64 using a Master-Slave configuration. It includes key registers like IRR, IMR, and ISR to track and control interrupts. Used in x86 systems, embedded devices, and legacy computers, it efficiently handles interrupt-driven processes, improving CPU performance.



The Block Diagram consists of 8 blocks which are – Data Bus Buffer, Read/Write Logic, Cascade Buffer Comparator, Control Logic, Priority Resolver and 3 registers- ISR, IRR, IMR.

Data bus buffer :

This Block is used as a mediator between 8259 and 8085/8086 microprocessor by acting as a buffer. It takes the control word from the 8085 (let say) microprocessor and transfer it to the control logic of 8259 microprocessor. After selection of Interrupt by 8259 microprocessor (based on priority of the interrupt), it transfer the opcode of the selected Interrupt and address of the Interrupt service sub routine to the other connected microprocessor. The data bus buffer consists of 8 bits represented as D0-D7 in the block diagram. Thus, shows that a maximum of 8 bits data can be transferred at a time.

Read/Write logic :

This block works only when the value of pin CS is low (as this pin is active low). This block is responsible for the flow of data depending upon the inputs of RD and WR. These two pins are active low pins used for read and write operations.

Control logic:

It is the center of the PIC and controls the functioning of every block. It has pin INTR which is connected with other microprocessor for taking interrupt request and pin INT for giving the output. If 8259 is enabled, and the other microprocessor Interrupt flag is high then this causes the value of the output INT pin high and in this way 8259 responds to the request made by other microprocessor.

Interrupt request register (IRR):

It stores all the interrupt level which are requesting for Interrupt services.

Interrupt service register (ISR):

It stores the interrupt level which are currently being executed.

Interrupt mask register (IMR) :

It stores the interrupt level which have to be masked by storing the masking bits of the interrupt level.

Priority resolver:

It examines all the three registers and set the priority of interrupts and according to the priority of the interrupts, interrupt with highest priority is set in ISR register. Also, it reset the interrupt level which is already been serviced in IRR.

Cascade buffer:

To increase the Interrupt handling capability, we can further cascade more number of pins by using cascade buffer. So, during increment of interrupt capability, CSA lines are used to control multiple interrupt structure.

Advantages

- Efficient Interrupt Handling: Manages up to 8 interrupts (expandable to 64).
- Flexible: Works with 8085/8086 and other microprocessors.
- Easy to Use: Simple setup with programmable modes.
- Cost-Effective: Reduces CPU workload for interrupt management.

Disadvantages

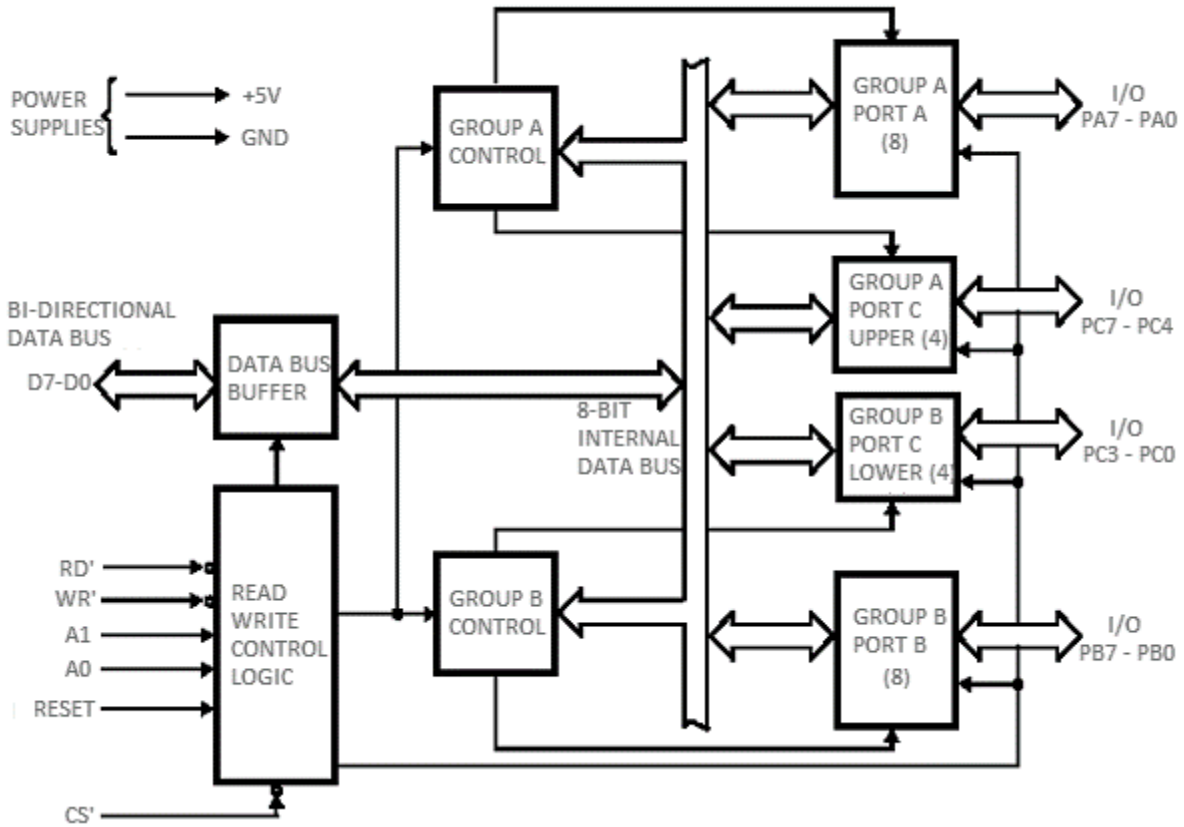
- Limited Per-Chip Interrupts: Only 8 interrupts per 8259 (requires cascading for more).
- Complex Programming: Requires careful setup of modes and priorities.
- No Advanced Features: Lacks modern capabilities like DMA (Direct Memory Access).

Applications

- Used in early PCs and industrial systems to manage interrupts from keyboards, timers, disks, etc.
- Ideal for systems needing organized handling of multiple devices (e.g., printers, sensors).

3. 8255 PPI

The Intel 8255 is a Programmable Peripheral Interface (PPI) used to interface microprocessors with external input/output (I/O) devices. It has three 8-bit ports (Port A, Port B, and Port C), which can be programmed as input or output. The 8255 operates in three modes: Mode 0 (Simple I/O), Mode 1 (Strobed I/O with Handshaking), and Mode 2 (Bidirectional Bus I/O). It is widely used in microprocessor-based systems, industrial automation, and embedded applications for controlling peripherals like keypads, displays, and sensors.



1. **Data Bus Buffer:**

   The data bus buffer is mainly used for connecting the inside bus of the microprocessor with the system bus so that proper interfacing can be established between these two. This buffer simply permits the read or write operation to be executed from or to the CPU. This buffer permits the data supplied from the control register or ports to the CPU in case of write operation & from the CPU to the status register or ports in case of the read operation.

2. **Read/Write Control Logic:**

   Read or write control logic unit controls the inside system operations. This unit holds the capability to manage both the data transfer & status or control words internally & externally. Once there needs data to fetch then it allows the provided address by the 8255 by the bus & generates a command immediately to the two control groups for the specific operation.

3. **Group A & Group B Control:**

   Both these groups are managed by the CPU and work based on the generated command by the CPU. This CPU transmits control words toward these two groups and they consecutively transmit the suitable command to their particular port. Group A controls port A with higher order port C bits whereas group B controls port B with lower order port C bits.

4. **Port A & Port B**

   Port A & Port B includes an 8-bit input latch and 8-bit buffered or latched output. The main function of these ports is also independent of the mode of operation. Port A can be programmed in 3 modes like modes 0, 1, and 2 whereas Port B can be programmed in modes 0 & mode 1.

5. **Port C**

Port C includes an 8-bit data input buffer and 8-bit bidirectional data o/p latch or buffer. This port is divided mainly into two sections – port C upper PCU & port C lower PC. So these two sections are mainly programmed & separately used as a 4-bit I/O port. This port is used for handshake signals, Simple I/O & status signal inputs. This port is used in combination with port A & Port B for both the status and handshaking signals. This port provides only direct but sets or resets capacity.

Advantages of 8255 PPI

✔ Flexible I/O operations – Ports can be programmed as input or output.

✔ Multiple modes – Supports different operational modes for various applications.

✔ Reduces CPU load – Helps in handling I/O operations efficiently.

✔ Easy to interface – Can be directly connected with microprocessors like 8085 and 8086.

Disadvantages of 8255 PPI

✘ Limited number of ports – Only three 8-bit ports, restricting the number of devices that can be connected.

✘ No direct memory access (DMA) support – Data transfer is slower compared to direct memory-mapped devices.

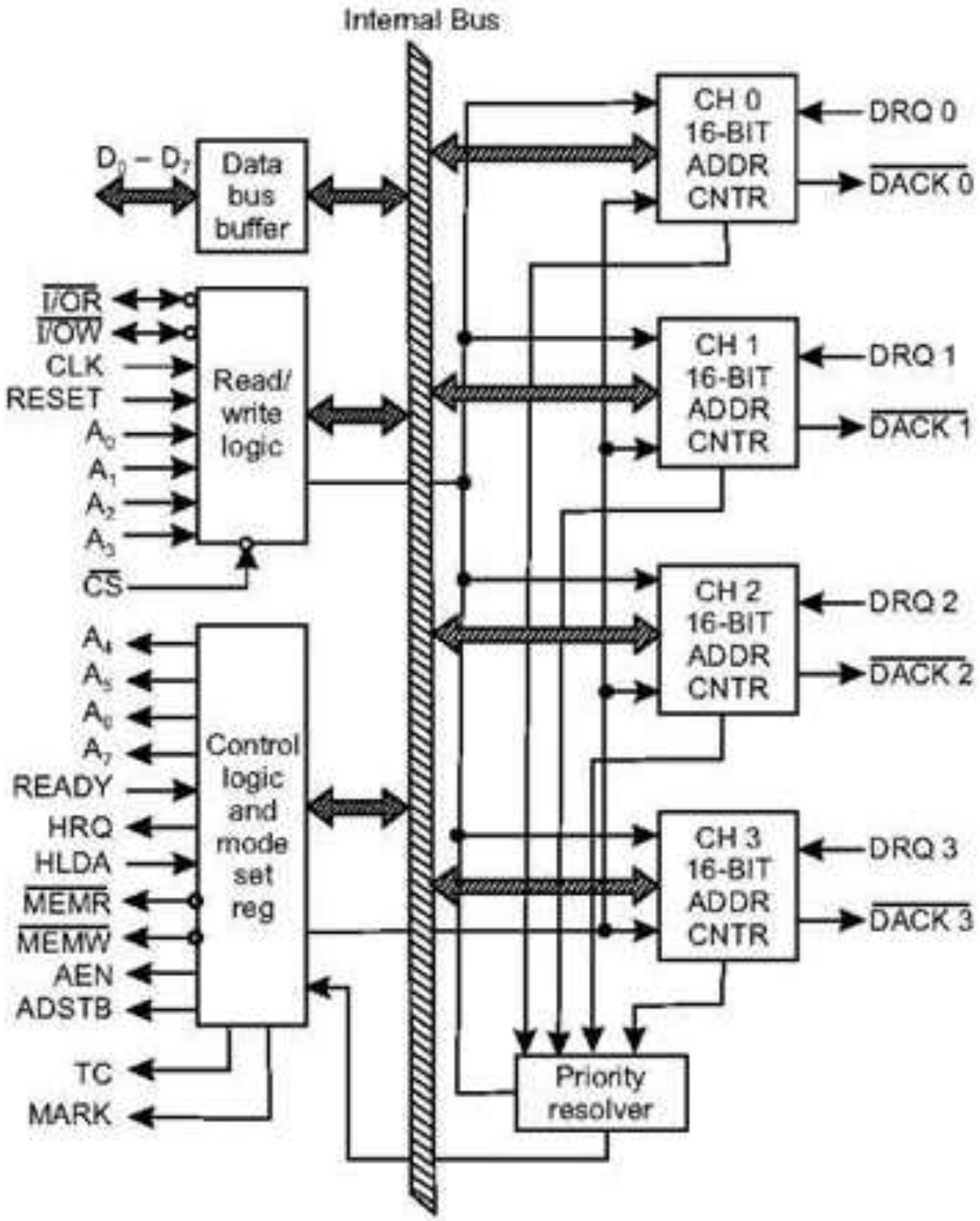
✘ Fixed architecture – Cannot be reconfigured dynamically for different I/O types.

Applications of 8255 PPI

- Interfacing keyboards and displays in microprocessor-based systems.
- Industrial automation for controlling sensors and actuators.
- Traffic light control systems for managing signals.
- Data acquisition systems for monitoring physical parameters.
- Robotics for controlling movement and sensors.

4. DMA Controller

The Intel 8257 is a Direct Memory Access (DMA) Controller that allows high-speed data transfer between memory and I/O devices without CPU intervention. This improves system performance by freeing the CPU from data transfer tasks. It has four independent DMA channels, each capable of transferring data at high speed.



Working of 8257 DMA Controller

- The CPU initializes the 8257 by sending control commands, setting up source and destination addresses, and defining the transfer count.
- When an I/O device requests data transfer, it sends a DMA request (DREQ) signal to the 8257.
- The 8257 sends a HOLD signal to the CPU, requesting control over the system bus.
- If the CPU grants permission, it responds with a HLDA (Hold Acknowledge) signal.
- The 8257 takes control of the bus and transfers data directly between memory and the I/O device.

- Once the transfer is complete, the 8257 releases the bus, and the CPU regains control.

Features of 8257 DMA Controller

✔ 4 DMA channels for multiple device connections.
✔ Memory-to-memory, I/O-to-memory, and I/O-to-I/O transfers supported.
✔ Burst mode and cycle stealing mode for efficient data transfer.
✔ Prioritization of DMA requests using a priority scheme.
✔ Autoloading of address and count registers for faster operations.

Advantages

✔ Faster data transfer compared to CPU-controlled transfers.
✔ Reduces CPU workload, allowing it to perform other tasks.
✔ Efficient handling of large data transfers without CPU intervention.
✔ Supports multiple data transfer modes for flexible operation.

Disadvantages:

✘ Consumes system bus bandwidth, which may slow down other processes.
✘ Increases system complexity due to additional control signals.
✘ Not suitable for small data transfers, as CPU intervention is minimal.

Applications:

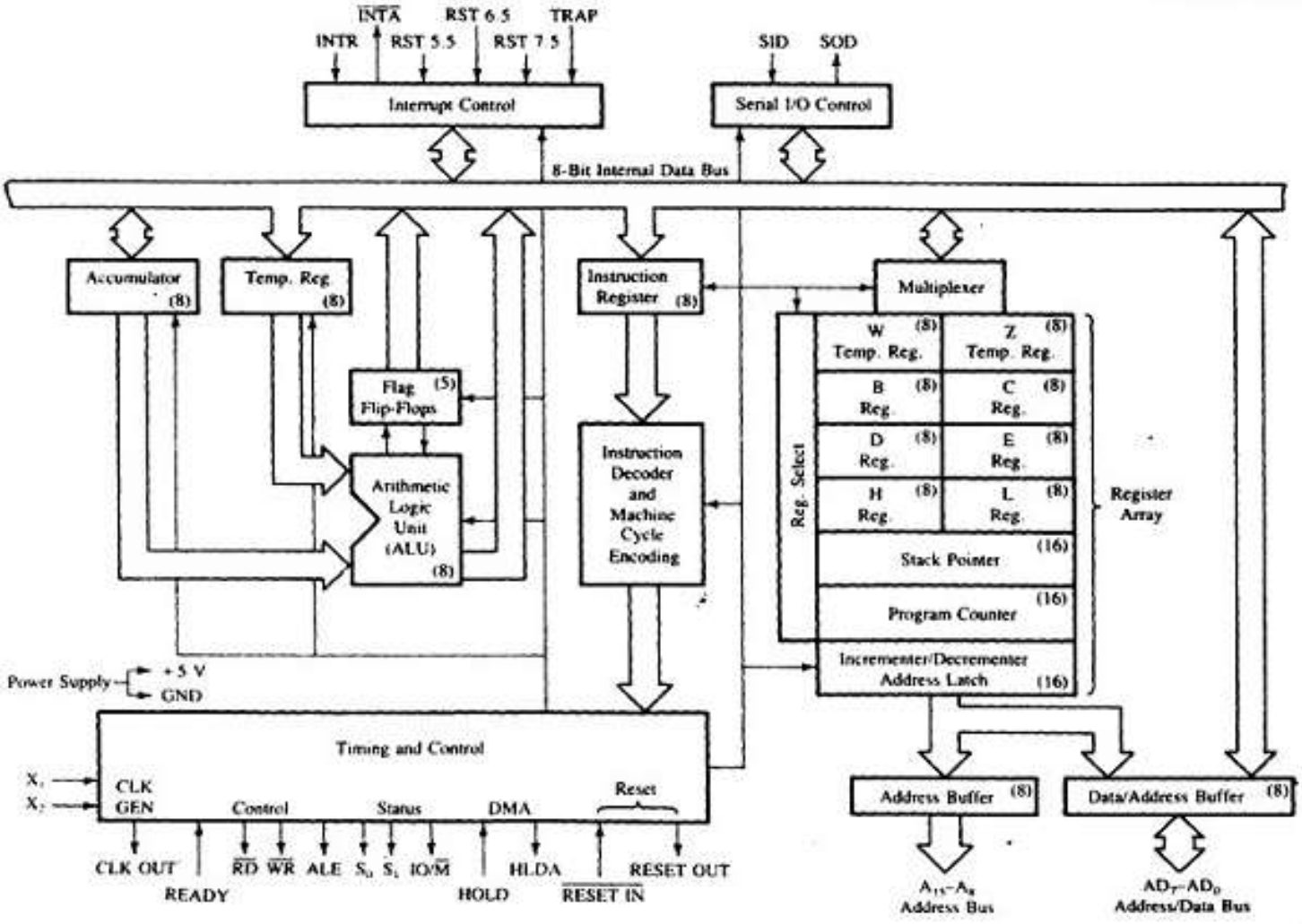
- High-speed data transfer in disk controllers.
- Real-time applications like audio/video streaming.
- Communication systems for fast data transmission.
- Embedded systems requiring low CPU usage for data transfer.

5. 8085 Microprocessor Architecture

8085 is pronounced as "eighty-eighty-five" microprocessor. It is an 8-bit microprocessor designed by Intel in 1977 using NMOS technology.

It has the following configuration −

- 8-bit data bus
- 16-bit address bus, which can address upto 64KB
- A 16-bit program counter
- A 16-bit stack pointer
- Six 8-bit registers arranged in pairs: BC, DE, HL
- Requires +5V supply to operate at 3.2 MHZ single phase clock
- It is used in washing machines, microwave ovens, mobile phones, etc.



The **8085 microprocessor** is an 8-bit processor developed by Intel in the mid-1970s. It was widely used in early personal computers and is still a popular choice in embedded systems and educational contexts due to its simplicity and ease of use. The 8085 architecture includes several key components, such as the

- Accumulator
- General-Purpose Registers
- Program Counter,
- Stack Pointer
- Instruction Register
- Flags Register
- buses

The **Accumulator** is an 8-bit register that holds the result of arithmetic and logical operations performed by the processor. It is connected to the Arithmetic and Logic Unit (ALU), and almost all the operations use the accumulator as the primary register for storing results. The 8085 also has six **general-purpose registers** (B, C, D, E, H, L), which are used to hold data during operations. These registers can also be paired together to form 16-bit registers like BC, DE, and HL, which are commonly used for memory addressing.

The **Program Counter (PC)** is a 16-bit register that holds the memory address of the next instruction to be fetched. The Program Counter is automatically incremented after each instruction fetch to point to the next instruction in sequence. The **Stack Pointer (SP)**, another 16-bit register, manages the stack, a special area of memory used for temporary data storage during subroutine calls, interrupts, or saving the state of the processor.

Instruction Register and Decoder

The **Instruction Register (IR)** is an 8-bit register that holds the current instruction being executed. After the instruction is fetched from memory, it is stored in the Instruction Register for decoding. The **Instruction Decoder** then interprets this instruction to understand what operation the microprocessor needs to perform (such as arithmetic, logic, memory read/write, etc.).

Flags Register

The **Flags Register** is an 8-bit register that contains various status flags, which are updated after arithmetic or logical operations. The flags indicate the result of these operations, such as whether there was a carry (Carry Flag), whether the result is zero (Zero Flag), whether the result is negative (Sign Flag), and whether the result has an even number of 1's (Parity Flag). These flags help the processor decide the next course of action, such as branching or jumping to another part of the program.

Address Bus, Data Bus, and Control Bus

The **Address Bus** is a 16-bit bus used to transfer the memory address to access data from memory or I/O devices. The **Data Bus** is an 8-bit bidirectional bus that carries data to and from memory and I/O devices. The **Control Bus** is a collection of signals that direct the operations of the microprocessor, including read/write operations, interrupts, and resets. It ensures that the processor interacts correctly with memory and I/O devices.

Execution Cycle

The **execution cycle** of the 8085 microprocessor begins with the **Program Counter** pointing to the address of the next instruction. The microprocessor fetches the instruction from memory and stores it in the **Instruction Register**. The instruction is then decoded by the **Instruction Decoder**, which determines the operation to perform. The **ALU** carries out the operation, and the results are typically stored in the **Accumulator**. Throughout this process, the **Timing and Control Unit** generates the necessary control signals, and the **Flags Register** updates its status to reflect the outcome of the operation.

Interrupts and Serial I/O

The **8085 microprocessor** supports several interrupt signals, including **TRAP**, **RST 7.5**, **RST 6.5**, **RST 5.5**, and **INTR**. These interrupts allow the processor to temporarily halt its current operation and process high-priority requests. **Serial I/O** capabilities are available through the **SID** (Serial Input Data) and **SOD** (Serial Output Data) pins. These pins allow the processor to communicate with external devices in a serial fashion, transmitting one bit of data at a time.

Timing and Control Unit:

The **Timing and Control Unit** (TCU) is a crucial part of the 8085 microprocessor. It generates the necessary timing signals and control signals required for the operation of the microprocessor, ensuring proper coordination between different components. It controls the sequence of operations by managing the **fetch** and **execute** cycles of the instructions and facilitating communication with memory and I/O devices.

The 8085 uses a clock signal to synchronize the execution of instructions. It operates with a clock frequency of 3-5 MHz.

ALE is used to latch the low-order address during the instruction fetch cycle. It enables the multiplexed address/data bus to act as an address bus during the first part of the cycle.

The **RD** signal indicates that the microprocessor is performing a read operation. When **RD** is active (low), the processor is reading data from memory or I/O devices.

The **WR** signal is active (low) during a write operation. It indicates that the processor is writing data to memory or I/O devices.

The **IO/M** signal determines whether the operation is related to memory or I/O devices. When **IO/M** is high, it signifies an I/O operation, and when low, it indicates a memory operation.

The **interrupt control signals** (like **INTR**, **TRAP**, **RST 7.5**, **RST 6.5**, and **RST 5.5**) allow external devices to interrupt the processor and gain control over the execution of the program.

HLDA is used to acknowledge the hold request from external devices. It allows the external device to take control of the system bus.

The **HLD** signal is sent to external devices to request them to relinquish control of the system bus.

The **Reset** signals, including **Reset In** and **Reset Out**, are used to reset the microprocessor. **Reset In** initiates the reset process, while **Reset Out** is used to reset external devices.

Advantages of the 8085 Microprocessor:

- Simple architecture.
- Low cost.
- Low power consumption.
- 16-bit address bus (64KB memory access).
- Versatile I/O capabilities.
- Supports multiple interrupts.
- Serial I/O support.
- Easy to program.
- Widely used in embedded systems.

Common Issues

1. **Overheating**: Can lead to malfunctions.
2. **Power Supply**: Voltage fluctuations affect performance.
3. **Timing Issues**: Problems with clock signals cause errors.
4. **Memory Interface**: Faulty connections can disrupt operations.
5. **Programming Errors**: Logic and syntax mistakes can result in incorrect outputs

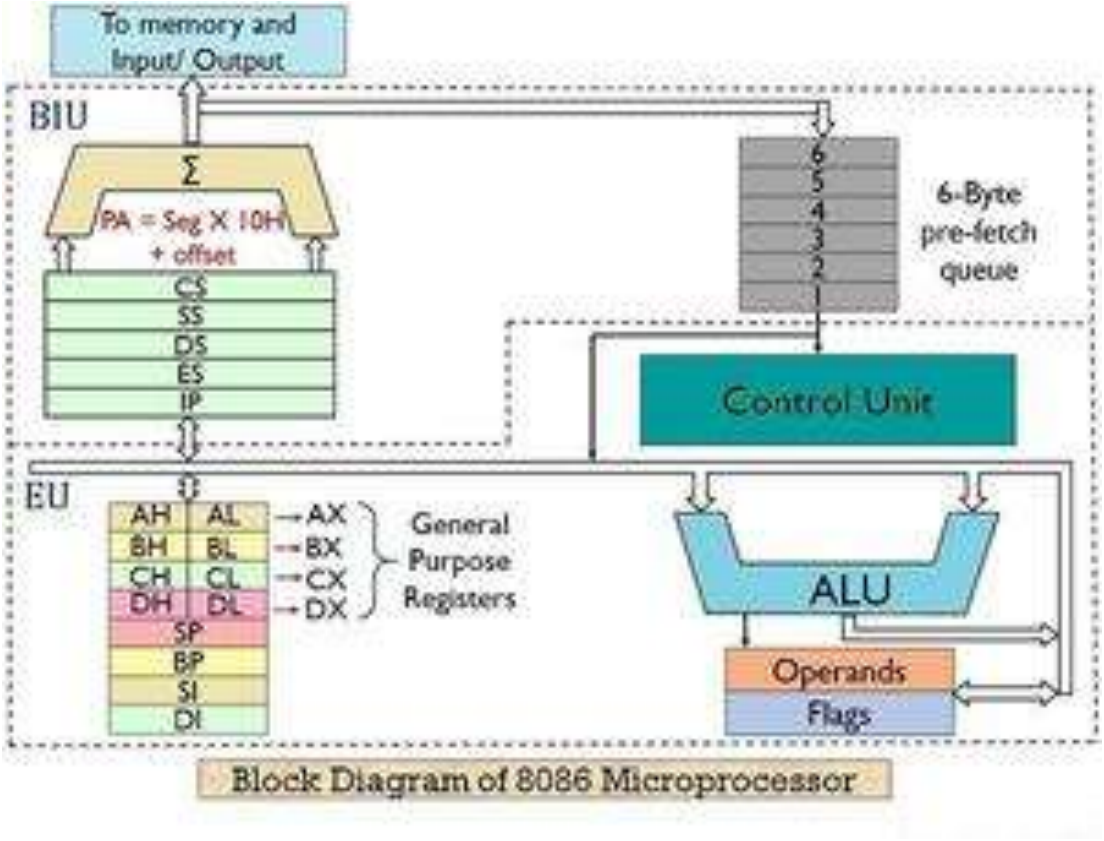
Uses of 8085

- **Embedded Systems**: Found in control systems, automotive electronics.
- **Peripherals**: Used in printers, scanners, disk drives.
- **Communication**: Used in modems and network interfaces.
- **Home Appliances**: Powers devices like washing machines, microwaves.
- **Education**: A simple, low-cost processor for learning about microprocessor concepts.

6. 8086 Microprocessor Architecture

The **8086 microprocessor** was developed by Intel in the late 1970s and was the first member of the x86 family of processors, which became the foundation for many subsequent microprocessors used in personal computers. It has a **20-bit address bus**, which allows it to access **up to 1MB of memory**, and a **16-bit data bus**, enabling efficient data transfer between the microprocessor and memory or I/O devices. The 8086 uses a **complex instruction set computer (CISC)** architecture, meaning it supports a wide variety of instructions, many capable of performing multiple operations in a single instruction.

The 8086 microprocessor architecture is based on **memory segmentation**, meaning the memory is divided into segments that are accessed using both a **segment register** and an **offset**. This architecture allows it to handle large memory spaces while using a 16-bit data bus. The microprocessor includes two primary execution units: the **Execution Unit (EU)** and the **Bus Interface Unit (BIU)**. The **BIU** is responsible for fetching instructions from memory and managing data transfer between the microprocessor and external memory or I/O devices, while the **EU** handles the execution of instructions.



Bus Interface Unit (BIU)

The **Bus Interface Unit (BIU)** is a critical part of the **8086 microprocessor** that interfaces the processor with external memory and I/O devices via the system bus. It handles multiple responsibilities to ensure smooth communication between the microprocessor and memory or I/O devices. Here's a breakdown of its key functions:

Functions of BIU:

1. **Memory Access & Address Calculation**:

The BIU calculates the **physical memory address** by combining the **segment address** (from the segment registers) and the **offset address** (from the instruction pointer or other pointers). The formula used is:

Physical Address = (Segment Address × 10H) + Offset Address

This address is then used to access the desired memory location, where the data or instruction resides.

2. **Instruction Fetching**:

   One of the primary responsibilities of the BIU is fetching instructions from memory. It fetches the instructions from the **Code Segment (CS)**, stores them temporarily in the **pre-fetch queue**, and then sends them to the **Execution Unit (EU)** for decoding and execution. This process is integral to **pipelining** in the 8086, where instruction fetching occurs while another instruction is being executed.

3. **Pre-fetch Queue**:

   The **pre-fetch queue** is a **6-byte FIFO (First-In-First-Out)** queue that temporarily holds instructions fetched from memory. This allows the processor to **fetch, decode, and execute instructions in parallel**, which speeds up execution. The **pre-fetch queue** is flushed whenever a branch or jump instruction is encountered because the instruction flow is altered.

4. **Data Transfer Between Microprocessor and I/O Devices**:

   The BIU also facilitates data transfer between the **microprocessor** and **I/O devices** or memory. When the **EU** needs to interact with external memory or I/O devices, the BIU manages these data transfers. It sends and receives data to/from memory using the **Data Bus** and also handles control signals related to read and write operations.

Components of BIU:

1. **Segment Registers (CS, DS, SS, ES)**:

   These registers hold the **base address** of various segments of memory:

   - **CS (Code Segment)**: Contains the starting address of the code (program) stored in memory.
   - **DS (Data Segment)**: Contains the base address for data storage.
   - **SS (Stack Segment)**: Contains the base address for the stack, where temporary data, such as return addresses for function calls, is stored.
   - **ES (Extra Segment)**: Used for additional data storage in certain operations, particularly for string manipulations.

   - These segment registers help in accessing different areas of memory using **segmented addressing**.

2. **Instruction Pointer (IP)**:

   The **Instruction Pointer (IP)** is a 16-bit register that points to the **offset address** of the next instruction in the **Code Segment (CS)**. The IP is incremented after each instruction is fetched, ensuring the correct sequence of instruction execution.

   When a branch instruction (like a jump or call) occurs, the IP is updated with the new offset address, allowing the program to continue from a different part of memory

3. **Address Generation Circuit**:

   The **Address Generation Circuit** in the BIU generates the **physical address** by using the segment address stored in the segment registers and the offset address. This circuit ensures that the right address is computed for fetching data or instructions, allowing the processor to access the appropriate memory location.

Execution Unit (EU)

The **Execution Unit (EU)** is the second major functional unit of the **8086 microprocessor**. It is responsible for executing the instructions that have been fetched and decoded by the **Bus Interface Unit (BIU)**. The EU works closely with the **ALU (Arithmetic Logic Unit)** and other internal registers to carry out the microprocessor's operations.

Functions of EU

1. **Instruction Decoding**:

   The **EU** receives instructions from the **pre-fetch queue** of the **BIU** and decodes them to determine which operations need to be executed. This decoding process involves translating the machine code into a set of micro-operations that the EU can execute.

2. **Execution of Instructions**:

   The EU is responsible for executing the instructions using the **ALU** (Arithmetic Logic Unit). The ALU performs **arithmetic** operations (like addition, subtraction, multiplication, division) and **logical operations** (like AND, OR, XOR, NOT).

   After decoding, the EU uses the appropriate registers and the ALU to perform the necessary operation, whether it's a simple arithmetic task or a complex logical comparison.

3. **Data Movement**:

   The EU also handles data movement operations, such as **moving data between registers**, **memory**, and **I/O devices**. It directs the flow of data between general-purpose registers, segment registers, the ALU, and memory.

4. **Control and Coordination**:

   The **EU** manages the flow of data within the microprocessor and sends control signals to the **BIU** to request data or instructions when needed. It coordinates the execution of operations by interacting with the **control unit** to ensure the correct sequence of events.

1. **General-Purpose Registers (AX, BX, CX, DX)**:

   The **general-purpose registers** are used to hold **operands** and **intermediate results** during the execution of arithmetic and logical operations.

   These registers can be accessed as 16-bit registers, or each can be divided into two 8-bit registers (e.g., AH and AL for AX).

- **AX** (Accumulator): The primary register used for arithmetic operations.
- **BX**: Often used for memory addressing or as a base register in certain operations.
- **CX**: Used for loop counters and string operations (count register).
- **DX**: Often used with **AX** for 32-bit operations, particularly for multiplication and division.

2. **Special-Purpose Registers**:

   The EU also contains several **special-purpose registers** that help manage the program's execution, particularly with respect to **stack operations** and **string operations**.

   - **Stack Pointer (SP)**: Points to the top of the stack in the **Stack Segment (SS)**.
   - **Base Pointer (BP)**: Used to access data on the stack during procedures or function calls.
   - **Source Index (SI)** and **Destination Index (DI)**: Used for string operations, particularly for moving blocks of data.

3. **Flag Register**:

   The **Flag Register** is a 16-bit register that contains **status flags** that reflect the results of arithmetic and logical operations. The flags indicate conditions such as:

   - **Carry Flag (CF)**: Set when an operation generates a carry out of the most significant bit.
   - **Zero Flag (ZF)**: Set if the result of an operation is zero.
   - **Sign Flag (SF)**: Set if the result of an operation is negative.
   - **Overflow Flag (OF)**: Set if the result of an operation causes an overflow.

   - The **Flag Register** also contains **control flags**, such as the **Trap Flag (TF)** and **Interrupt Flag (IF)**, which control the processor's behavior with respect to interrupts.

Advantages

1. **Wide Range of Instructions**: The 8086 supports a large set of instructions, making it versatile for complex tasks.
2. **Segmented Memory Architecture**: It can access up to **1MB** of memory using **16-bit registers** and a **20-bit address bus**.
3. **Powerful Instruction Set**: The instruction set includes commands that can perform multiple operations in one instruction.
4. **Pipelining**: The 8086 uses a **6-byte pre-fetch queue** that enables pipelining, allowing instructions to be fetched, decoded, and executed in parallel.
5. **Rich Set of Registers**: It has a variety of general-purpose and special-purpose registers, allowing for efficient data manipulation and control.

Disadvantages

1. **Complex Programming**: The architecture requires in-depth knowledge of assembly language and is harder to program compared to modern processors.
2. **Limited Memory Addressing**: The 8086 can address only **1MB** of memory, which is limiting for modern applications.
3. **Limited Instruction Set**: Compared to modern processors, the 8086's instruction set is relatively limited.
4. **Segmented Memory Management**: Managing memory using segment registers and offsets can be difficult, especially for complex programs.

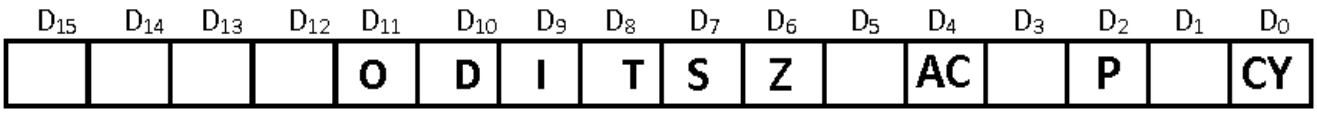
Applications:

- Personal Computers
- Embedded Systems

- Workstations/Terminals
- POS Systems
- Video Game Consoles
- Consumer Electronic

7. Flag Register in 8086 Microprocessor

- The **flag register** is a 16-bit register in the Intel 8086 microprocessor, reflecting the processor's state after instruction execution.
- Often called the **status register**, it holds various flags that indicate the outcome of arithmetic, logic, and other operations.
- Used primarily for **conditional branching** in assembly language programming, it helps control the flow based on the results of prior operations.



1. Sign Flag (S)

The Sign Flag is set if the result of an operation is negative. In binary arithmetic, this is determined by the most significant bit (MSB) of the result. If the MSB is 1, it indicates a negative number for signed numbers.

Example:

Operation: Subtract 00000001 (1) from 10000000 (-128 in signed 8-bit).

- **First Operand**: 10000000 (binary) = -128.
- **Second Operand**: 00000001 (binary) = 1.
- **Result**: 10000000 - 00000001 = 01111111 (binary) = 127 (positive result).
- Since the MSB is 0 (positive result), the **Sign Flag (S)** is **reset**.

2. Zero Flag (Z)

The Zero Flag is set if the result of an operation is zero. This means all bits of the result are 0.

Example:

Operation: 00001000 (8) - 00001000 (8).

- **First Operand**: 00001000 (binary) = 8.
- **Second Operand**: 00001000 (binary) = 8.
- **Result**: 00001000 - 00001000 = 00000000 (binary) = 0.
- Since the result is 00000000, the **Zero Flag (Z)** is **set**.

3. Auxiliary Carry Flag (AC)

The Auxiliary Carry Flag is set if there is a carry from bit 3 to bit 4 in an operation. This is primarily used in binary-coded decimal (BCD) operations.

Example:

Operation: Add 00001001 (9) and 00000111 (7).

- **First Operand**: 00001001 (binary) = 9.
- **Second Operand**: 00000111 (binary) = 7.
- **Result**: 00001001 + 00000111 = 00001100 (binary) = 12.
- There is a carry from bit 3 to bit 4, so the **Auxiliary Carry Flag (AC)** is **set**.

4. Parity Flag (P)

The Parity Flag is set if the result of an operation has an **even** number of 1s. If the result has an odd number of 1s, the Parity Flag is reset.

Example:

Operation: XOR 00001100 (12) with 00000011 (3).

- **First Operand**: 00001100 (binary) = 12.
- **Second Operand**: 00000011 (binary) = 3.
- **Result**: 00001100 XOR 00000011 = 00001111 (binary) = 15.
- The result 00001111 has **4 ones** (even number of 1s), so the **Parity Flag (P)** is **set**.

5. Carry Flag (CY)

The Carry Flag is set if there is a carry or borrow in an operation. For addition, it's set if the result exceeds the register size, and for subtraction, it's set if there's a borrow.

Example:

Operation: Add 11111111 (255) and 00000001 (1).

- **First Operand**: 11111111 (binary) = 255.
- **Second Operand**: 00000001 (binary) = 1.
- **Result**: 11111111 + 00000001 = 100000000 (binary).
- Since the result is a 9-bit number (overflow for an 8-bit result), the **Carry Flag (CY)** is **set**.

6. Overflow Flag (O)

The Overflow Flag is set if the result of a signed operation overflows beyond the range that can be represented. It indicates that the result cannot be correctly represented with the available number of bits.

Example:

- **Operation**: Add 01111111 (127) and 00000001 (1) in signed 8-bit.
  - **First Operand**: 01111111 (binary) = 127.
  - **Second Operand**: 00000001 (binary) = 1.
  - **Result**: 01111111 + 00000001 = 10000000 (binary) = -128.
  - Since adding two positive numbers results in a negative number, the **Overflow Flag (O)** is **set**.

Control Flags

7. Directional Flag (D)

The Directional Flag is used in string operations. If it's set (1), the processor processes strings from higher memory addresses to lower addresses (decrementing). If reset (0), it processes strings from lower memory addresses to higher addresses (incrementing).

Example:

Directional Flag set to 1 (decrementing) means the string will be processed from higher memory addresses to lower.

8. Interrupt Flag (I)

The Interrupt Flag enables or disables interrupts. If set (1), interrupts are enabled, and if cleared (0), interrupts are disabled.

Example:

If the **Interrupt Flag (I)** is set, the processor will recognize interrupt requests from peripherals.

9. Trap Flag (T)

The Trap Flag is used for debugging. If set (1), the processor will execute one instruction and then automatically generate a trap interrupt, allowing the debugger to inspect the state of the program after each instruction.

Example:

If the **Trap Flag (T)** is set, the processor will generate an interrupt after each instruction execution for debugging purposes.

8. Memory Segmentation in 8086 Microprocessor

Segmentation is the process in which the main memory of the computer is logically divided into different segments and each segment has its own base address. It is basically used to enhance the speed of execution of the computer system, so that the processor is able to fetch and execute the data from the memory easily and fast.

Need for Segmentation:

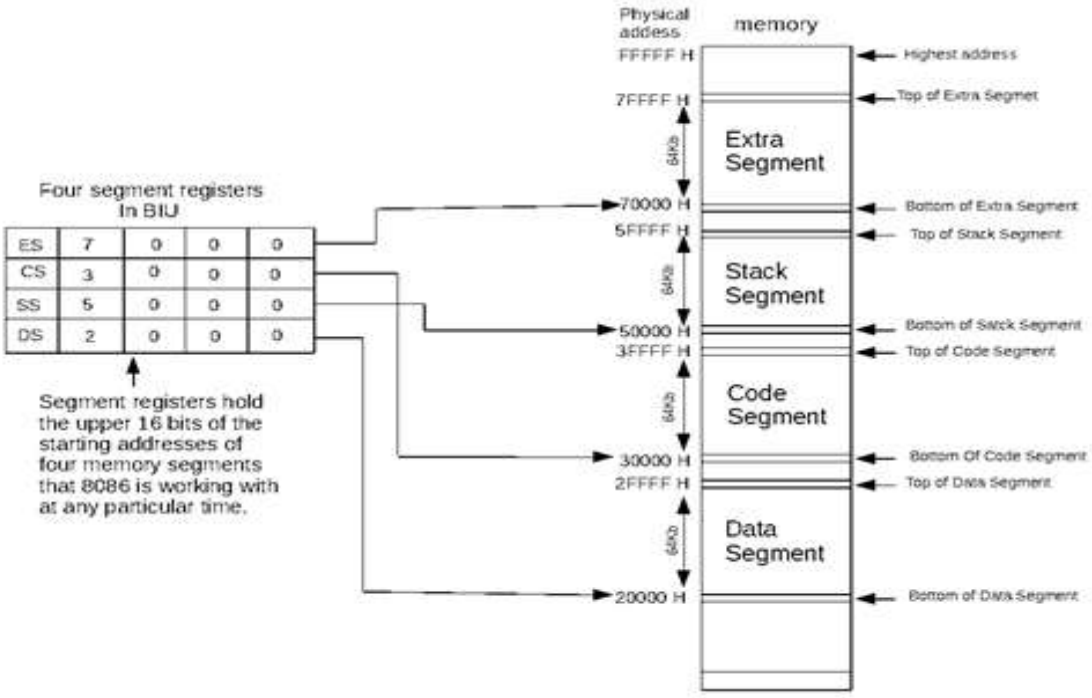
The Bus Interface Unit (BIU) contains four 16 bit special purpose registers (mentioned below) called as Segment Registers.

- **Code segment register (CS):** is used for addressing memory location in the code segment of the memory, where the executable program is stored.
- **Data segment register (DS):** points to the data segment of the memory where the data is stored.
- **Extra Segment Register (ES):** also refers to a segment in the memory which is another data segment in the memory.
- **Stack Segment Register (SS):** is used for addressing stack segment of the memory. The stack segment is that segment of memory which is used to store stack data.

The number of address lines in 8086 is 20, 8086 BIU will send 20bit address, so as to access one of the 1MB memory locations. The four segment registers actually contain the upper 16 bits of the starting addresses of the four memory segments of 64 KB each with which the 8086 is working at that instant of time. A segment is a logical unit of memory that may be up to 64 kilobytes long. Each segment is made up of contiguous memory locations. It is an independent, separately addressable unit. Starting address will always be changing. It will not be fixed.

Note: that the 8086 does not work the whole 1MB memory at any given time. However, it works only with four 64KB segments within the whole 1MB memory.

Below is the one way of positioning four 64 kilobyte segments within the 1M byte memory space of an 8086.



Rules of Segmentation :

Segmentation process follows some rules as follows:

- The starting address of a segment should be such that it can be evenly divided by 16.
- Minimum size of a segment can be 16 bytes and the maximum can be 64 kB.

Segment	Offset Registers	Function
CS	IP	Address of the next instruction
DS	BX, DI, SI	Address of data
SS	SP, BP	Address in the stack
ES	BX, DI, SI	Address of destination data (for string operations)

Advantages of the Segmentation:

The main advantages of segmentation are as follows:

- It provides a powerful memory management mechanism.
- Data related or stack related operations can be performed in different segments.
- Code related operation can be done in separate code segments.
- It allows to processes to easily share data.

- It allows to extend the address ability of the processor, i.e. segmentation allows the use of 16 bit registers to give an addressing capability of 1 Megabytes. Without segmentation, it would require 20 bit registers.
- It is possible to enhance the memory size of code data or stack segments beyond 64 KB by allotting more than one segment for each area.

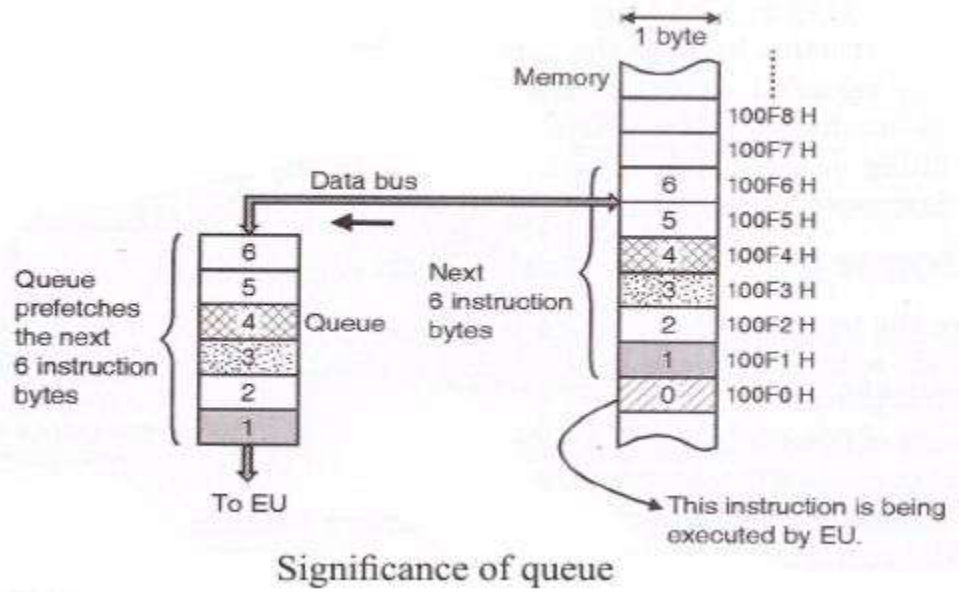
9. Instruction pipelining or queuing 8086 Microprocessor

- In 8086, to speed up the execution of program, the instruction fetching and execution of instructions are overlapped with each other.
- This process of fetching the next instruction when the present instruction is being executed is called as pipelining.
- In pipelining, when the nth instruction is executed, the (n+1) th instruction is fetched and thus the processing speed is increased.
- Pipelining has become possible due to the use of queue.
- BIU (Bus Interfacing Unit) fills in the queue until the entire queue is full.
- BIU restarts filling in the queue when at least two locations of queue are vacant.

The Instruction Queue:

1. The execution unit (EU) is supposed to decode or execute an instruction.
2. Decoding does not require the use of buses.
3. When EU is busy in decoding and executing an instruction, the BIU fetches up to six instruction bytes for the next instructions.
4. These bytes are called as the pre-fetched bytes and they are stored in a first in first out (FIFO) register set, which is called as a queue.

Significance of Queue:



Significance of queue

1. As shown in the above figure, while the EU is busy in decoding the instruction corresponding to memory location 100F0, the BIU fetches the next six instruction bytes from locations 100F1 to 100F6 numbered as 1 to 6.
2. These instruction bytes are stored in the 6 byte queue on the first in first out (FIFO) basis.

3. When EU completes the execution of the existing instruction and becomes ready for the next instruction, it simply reads the instruction bytes in the sequence 1, 2…. from the Queue.
4. Thus the Queue will always hold the instruction bytes of the next instructions to be executed by the EU.

Advantages of pipelining:

1. The EU always reads the next instruction byte from the queue in BIU. This is much faster than sending out an address to the memory and waiting for the next instruction byte to come.
2. In short pipelining eliminates the waiting time of EU and speeds up the processing.
3. The 8086 BIU will not initiate a fetch unless and until there are two empty bytes in its queue. 8086 BIU normally obtains two instruction bytes per fetch.

Disadvantages of pipelining:

1. While pipelining can severely cut the time taken to execute a program, there are problems that cause it to not work as well as it perhaps should.
2. The three stages of the instruction execution process do not necessarily take an equal amount of time, with the time taken for 'execute' being generally longer than 'fetch'. This makes it much harder to synchronise the various stages of the different instructions.
3. Also, some instructions may be dependent on the results of other earlier instructions. This can arise when data produced earlier needs to be used or when a conditional branch based on a previous outcome is used.