

---

# **UNIT-4: INTRODUCTION TO 8086 MICROPROCESSOR**

**PROF. SHANKAR MALI**

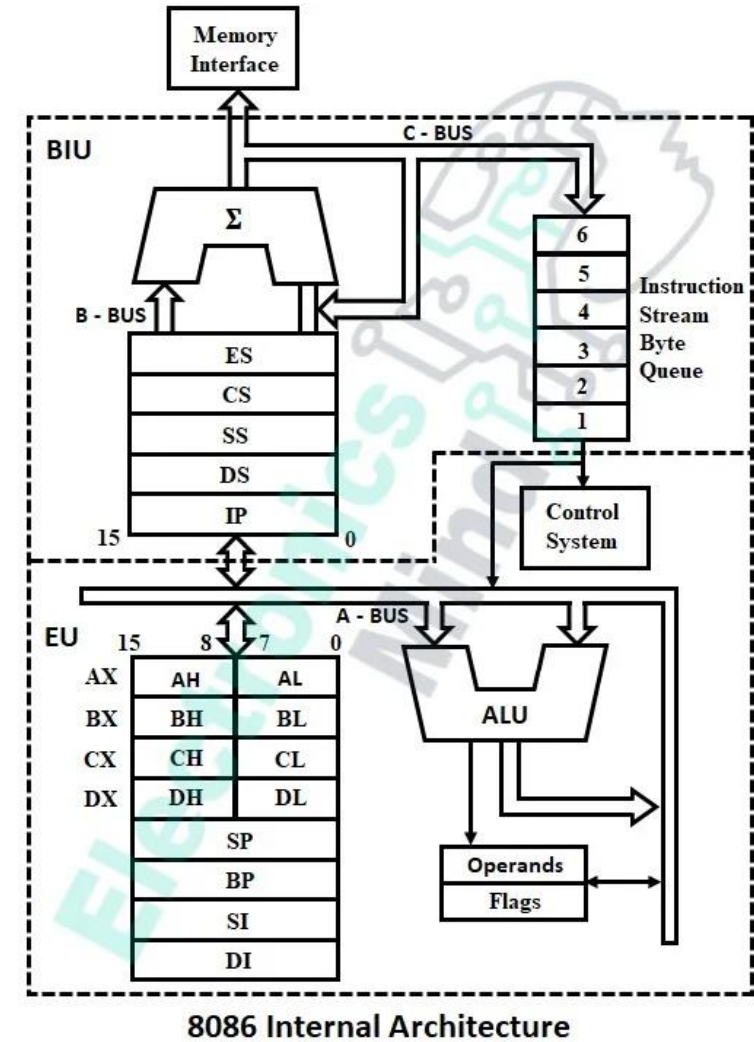
## 8086 MICROPROCESSOR ARCHITECTURE

- The **Intel 8086** is a **16-bit microprocessor** with a **20-bit address bus**, allowing it to access **1 MB of memory** ( $2^{20} = 1,048,576$  bytes). It has a **16-bit data bus**, meaning it processes **16-bit** data at a time. The **8086 architecture** follows the **Von Neumann architecture**, meaning **data and instructions share the same memory**.
- The architecture consists of two main units:
  1. **Bus Interface Unit (BIU)**
  2. **Execution Unit (EU)**
- These two units work together to improve processing efficiency using **pipelining**, where **BIU fetches instructions while EU executes previously fetched instructions**.

# 8086 MICROPROCESSOR ARCHITECTURE

1. Bus Interface Unit (BIU)

2. Execution Unit (EU)



## 1. BUS INTERFACE UNIT (BIU)

- The **BIU** is responsible for **interfacing with memory and I/O devices**. It generates the **20-bit physical address** and fetches instructions before passing them to the **Execution Unit (EU)**.
- **Components of BIU:**
- **1. Instruction Queue**
  - The **8086** uses a **6-byte instruction queue**.
  - It **prefetches** instructions from memory and stores them in the queue to speed up execution.
  - **Pipelining:** While the **Execution Unit (EU)** executes an instruction, the **BIU** fetches the next instruction, reducing wait time.

## WORKING OF INSTRUCTION QUEUE

- **Prefetching of Instructions:**

- The BIU fetches instructions from memory and stores them in a 6-byte queue (since 8086 has a 16-bit data bus, it can fetch 2 bytes at a time).
- This process happens in parallel while the EU executes previous instructions.

- **Execution of Instructions:**

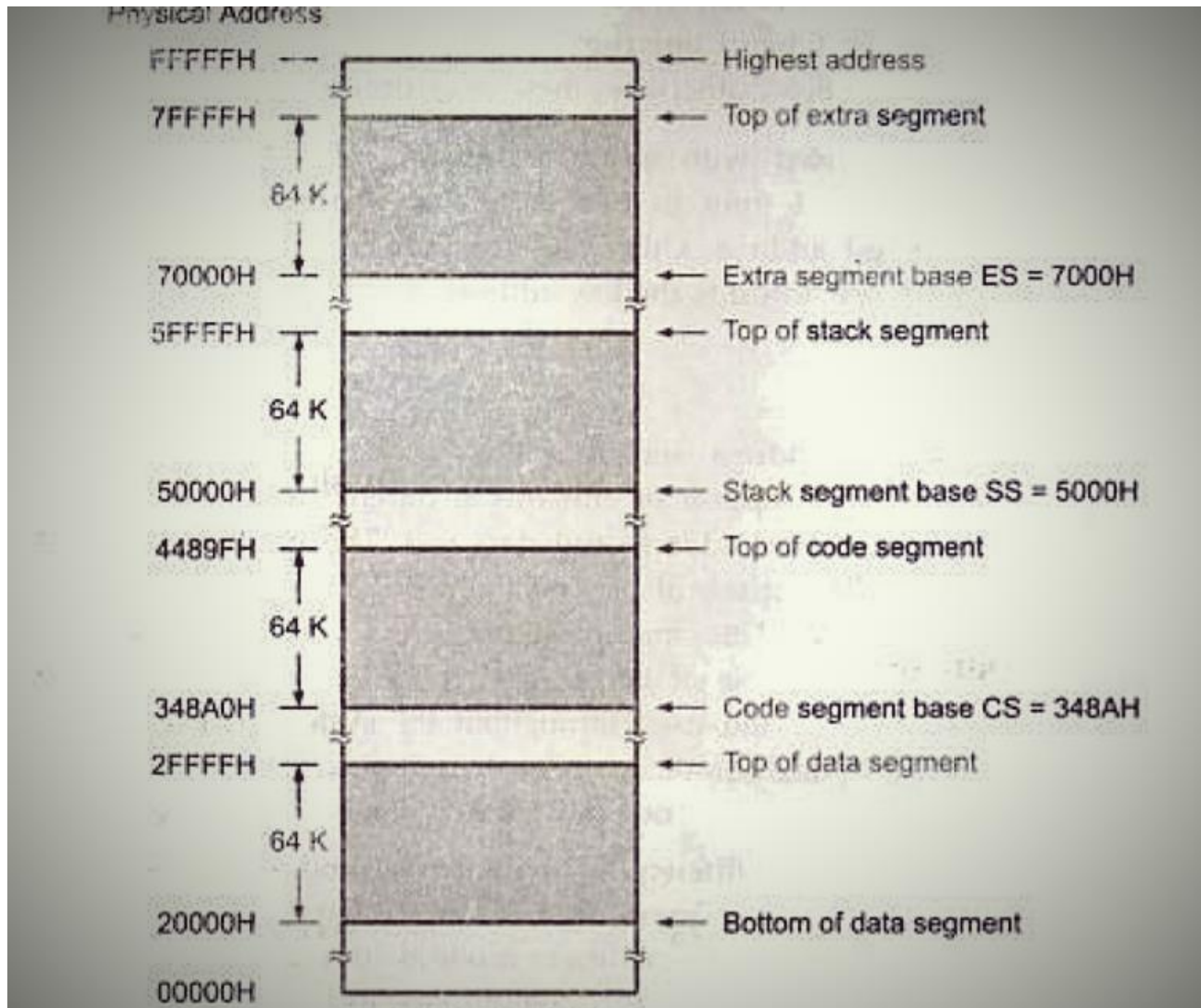
- The EU takes the next instruction from the queue and decodes/executes it.
- If the queue is empty (e.g., due to a jump or branch instruction), the EU waits for the BIU to fetch new instructions.

## WORKING OF INSTRUCTION QUEUE

- **Advantage of the Queue:**
- Reduces memory access time because instructions are preloaded.
- Improves overall execution speed since fetching and execution happen simultaneously.
- **Queue Flush Mechanism:**
- When a branch (such as JMP, CALL, or INT) occurs, the previously fetched instructions become invalid.
- The queue is flushed, and the BIU starts fetching instructions from the new target address.

## 2. SEGMENT REGISTERS (MEMORY SEGMENTATION)

- The **8086** uses **segmented memory**, dividing the 1 MB memory into **four segments of 64 KB each**.  
Each segment starts at a **base address** stored in **segment registers**, and the **offset address** provides the exact location.
- **Segment Registers:**
  - **Code Segment Register (CS):** Holds the base address of the **code segment** (program instructions).
  - **Data Segment Register (DS):** Holds the base address of the **data segment** (variables, constants).
  - **Stack Segment Register (SS):** Holds the base address of the **stack segment** (stack operations like PUSH, POP).
  - **Extra Segment Register (ES):** Used for additional data storage and **string operations**.



# SEGMENT REGISTERS (MEMORY SEGMENTATION)



### 3. INSTRUCTION POINTER (IP)

- The **IP** holds the **offset address** of the next instruction to be executed.
- Since the **8086 has a 20-bit address bus**, it can access up to **1 MB ( $2^{20}$  bytes)** of memory. However, registers in the **8086 are only 16-bit**, so to generate a **20-bit physical address**, the **segment address and offset address are combined** using the formula:

$$\text{Physical Address} = (\text{Segment Address} \times 16) + \text{Offset Address}$$

### 3. INSTRUCTION POINTER (IP) (CONT.)

#### Purpose of $\Sigma$ in the BIU

The  $\Sigma$  (Adder) plays a crucial role in effective address calculation. The 8086 microprocessor uses segmented memory architecture, where memory addresses are calculated by combining a segment address and an offset. The  $\Sigma$  symbol represents the hardware adder that performs this calculation.

#### Given:

- Code Segment (CS) = 2000H
- Instruction Pointer (IP) = 0500H

#### Step-by-Step Calculation:

1. Convert the segment address to a 20-bit base address:

$$2000H \times 16 = 20000H$$

(Multiplying by 16 means shifting left by 4 bits, so 2000H  $\rightarrow$  20000H)

2. Add the offset (IP) value to the base address:

$$20000H + 0500H = 20500H$$

#### Final Physical Address:

**20500H**

## 2. EXECUTION UNIT (EU)

- The **Execution Unit (EU)** is responsible for **executing instructions** fetched by the BIU. It performs **arithmetic, logical, and control operations**.
- **1. Arithmetic and Logic Unit (ALU)**
  - Performs **arithmetic operations** (ADD, SUB, MUL, DIV) and **logical operations** (AND, OR, XOR, NOT).
- **2. General Purpose Registers (GPRs)**
  - 8086 has **four 16-bit general-purpose registers**, which can also be used as **two 8-bit registers** (higher and lower bytes).

Register	16-bit	8-bit (Higher)	8-bit (Lower)	Purpose
AX	Accumulator	AH	AL	Used for arithmetic operations, I/O operations
BX	Base	BH	BL	Used for memory addressing
CX	Counter	CH	CL	Used for looping and shift operations
DX	Data	DH	DL	Used in multiplication and division

Register	Purpose
SP (Stack Pointer)	Points to the top of the stack
BP (Base Pointer)	Used for accessing stack data
SI (Source Index)	Used for string operations
DI (Destination Index)	Used for string operations

### 3. POINTER AND INDEX REGISTERS

THESE REGISTERS ARE USED FOR STACK OPERATIONS, MEMORY ADDRESSING, AND STRING OPERATIONS.

Flag	Meaning	Set When...
CF (Carry Flag)	Carry in arithmetic operations	Result is larger than 16 bits
PF (Parity Flag)	Parity of result	Even number of 1s in result
AF (Auxiliary Carry Flag)	Half-carry in BCD operations	Used for BCD arithmetic
ZF (Zero Flag)	Zero result	Result = 0
SF (Sign Flag)	Sign of result	1 = Negative, 0 = Positive
TF (Trap Flag)	Single-step debugging	If set, executes one instruction at a time
IF (Interrupt Flag)	Interrupt enable/disable	If set, enables interrupts
DF (Direction Flag)	String operation direction	1 = Auto-decrement, 0 = Auto-increment
OF (Overflow Flag)	Arithmetic overflow	Result too large

## FLAG REGISTER

THE FLAG REGISTER IS A 16-BIT REGISTER THAT STORES THE STATUS OF THE PROCESSOR AFTER EXECUTION. IT CONTAINS STATUS AND CONTROL FLAGS

# OPERAND

- An operand is the part of an instruction that specifies what data is to be manipulated or operated upon. In the 8086 microprocessor, operands can be of different types depending on the instruction format.

## Immediate Operand

- `MOV AX, 1234H` ; Immediate operand = 1234H

## Register Operand

- `ADD AX, BX` ; Operands are AX and BX (both registers)

## Memory Operand

- `MOV AL, [5000H]` ; Memory operand is located at address 5000H

## I/O Port Operand

- `IN AL, 60H` ; Operand is I/O port 60H

## ACCUMULATOR REGISTER (AX)

- The AX register is known as the "Accumulator" because it is heavily used in arithmetic and logical operations.
- It is also used in I/O operations and string processing.
- Divided into two 8-bit registers:
- AH (Accumulator High)
- AL (Accumulator Low)
- Used in Multiplication and Division
  - In multiplication, one operand must be in AX.
  - In division, AX stores the dividend.
- Used in I/O Operations
  - Used for input and output instructions (IN, OUT).
  - Used in String Operations
- Works with SI (Source Index) and DI (Destination Index) for string manipulation.

Example:

`MOV AX, 1234H ; Load 1234H into AX`

`MOV AL, 56H ; Load 56H into AL (lower byte of AX)`

## BASE REGISTER (BX)

- The BX register is mainly used as a base register for memory addressing.
- It helps in indirect addressing where an effective address (EA) is formed using BX.
- Divided into two 8-bit registers: BH (Base High) and BL (Base Low)
- ✓ Memory Addressing
  - Holds offset addresses in memory.
  - Used in register indirect addressing.
- ✓ Loop and Counter Operations
  - Can be used to store loop variables.
- ✓ Arithmetic Operations
  - Can be used for addition, subtraction, and other mathematical operations.

Example:

`MOV BX, 1000H ; Load 1000H into BX`

`MOV [BX],AX ; Store AX value at the memory location pointed by BX`



## COUNT REGISTER (CX)

- The CX register is primarily used as a counter.
- It is widely used in looping instructions (LOOP, LOOPZ, LOOPNZ).
- Divided into two 8-bit registers:
  - CH (Count High)
  - CL (Count Low)
- ✓ Loop Control
  - Works with the LOOP instruction, where CX is decremented automatically.
- ✓ Shift and Rotate Instructions
  - CL is used in bitwise shift and rotate operations.
- ✓ String Operations
  - Works with REP (Repeat) instructions for processing arrays and strings.

### Example:

MOV CX, 10 ; Load CX with 10 (Loop counter)

LOOP\_LABEL: DEC CX ; Decrement CX

JNZ LOOP\_LABEL ; Jump back to LOOP\_LABEL if CX is not zero

## DATA REGISTER (DX)

- The DX register is often used for multiplication, division, and I/O operations.
- In some cases, it works along with AX as a 32-bit pair (DX:AX).

- Divided into two 8-bit registers:

- DH (Data High)

- DL (Data Low)

- ✓ Multiplication and Division

- Used to store the high-order (upper) bits in multiplication.
- Used to store the remainder in division.

- ✓ I/O Port Addressing

- Used in port-based I/O operations (IN, OUT instructions).

- ✓ 16-bit & 32-bit Data Handling

- Works with AX to process large numbers.

### Example:

MOV AX, 1234H ; Load AX with 1234H

MOV DX, 5678H ; Load DX with 5678H

MUL DX ; Multiply AX with DX, result stored in DX:AX

# POINTER AND INDEX REGISTERS IN 8086

Register	Full Name	Purpose
SP	Stack Pointer	Points to the top of the stack
BP	Base Pointer	Used to access data in the stack segment
SI	Source Index	Used for string operations and memory addressing
DI	Destination Index	Used for string operations and memory addressing

## STACK POINTER (SP)

- The Stack Pointer (SP) is a 16-bit register that always points to the top of the stack.
- The stack is a LIFO (Last In, First Out) data structure.
- The SS (Stack Segment) register and SP together form the 20-bit physical address of the stack top.
- Example:
- `PUSH AX` ; Push AX onto stack (SP decreases by 2)
- `PUSH BX` ; Push BX onto stack (SP decreases by 2)
- `POP BX` ; Pop value from stack into BX (SP increases by 2)
- `POP AX` ; Pop value from stack into AX (SP increases by 2)

## BASE POINTER (BP)

- The Base Pointer (BP) is similar to SP but is mainly used to access data inside the stack segment.
- Unlike SP, BP does not change automatically during stack operations.
- Used for high-level language function calls (C, Pascal, etc.).
- Example:
- `MOV BP, SP` ; Copy SP to BP (to reference function parameters)
- `MOV AX, [BP+4]` ; Access function argument at BP+4
- `MOV BX, [BP-2]` ; Access local variable at BP-2

## SOURCE INDEX (SI)

- The Source Index (SI) is a 16-bit register mainly used for string operations and memory addressing.
- It holds the offset address of the source operand in string operations.
- Example:
- `MOV SI, 1000H` ; SI points to source data
- `MOV AL, [SI]` ; Load value from memory address (DS:SI) into AL
- `INC SI` ; Move SI to next memory location

## DESTINATION INDEX (DI)

- The Destination Index (DI) is a 16-bit register mainly used for string operations and memory addressing.
- It holds the offset address of the destination operand in string operations.
- Example:
- `MOV DI, 2000H` ; DI points to destination data
- `MOV [DI], AL` ; Store AL at memory address (ES:DI)
- `INC DI` ; Move DI to next memory location

# SUMMARY GPR AND POINTER/INDEX REGISTER

Feature	General Purpose Registers (GPRs)	Pointer/Index Registers
Registers Involved	AX, BX, CX, DX	SI, DI, BP, SP
Primary Purpose	Versatile for arithmetic, logic, data storage, and temporary results.	Used for memory addressing, pointer manipulation, and stack operations.
Data Size	Can be used as <b>16-bit</b> (full register) or split into <b>8-bit</b> parts (e.g., <b>AH/AL, BH/BL</b> ).	Typically used as <b>16-bit</b> registers for address calculation.
Arithmetic Operations	Commonly used for arithmetic ( <sup>ADD</sup> , <sup>SUB</sup> , etc.). Example: ADD AX, BX	Less common in arithmetic but can be indirectly involved in calculations.
String Operations	Used in data manipulation but not dedicated for string instructions.	<b>SI</b> and <b>DI</b> are key in string operations ( <sup>MOVSB</sup> , <sup>SCASB</sup> , etc.).
Loop Control	<b>CX</b> is the default counter for loop instructions ( <sup>LOOP</sup> ).	Not directly used for loop control.
Memory Addressing	<b>BX</b> can serve as a base register in addressing modes. Example: MOV AX, [BX + 4]	<b>SI</b> , <b>DI</b> , and <b>BP</b> are dedicated for effective memory addressing. Example: MOV AX, [SI + 4]
Stack Operations	Rarely used directly in stack operations.	<b>SP</b> (Stack Pointer) and <b>BP</b> (Base Pointer) are key for stack management.
Special Roles	Each GPR has specific roles in multiplication, division, and I/O.	Pointer/Index registers specialize in navigating arrays, tables, and stack frames.



# CONTROL SIGNAL GENERATION

- The control unit generates signals for:
  - Memory read/write
  - I/O operations
  - ALU operations (addition, subtraction, etc.)
  - Register operations
  - Interrupt handling

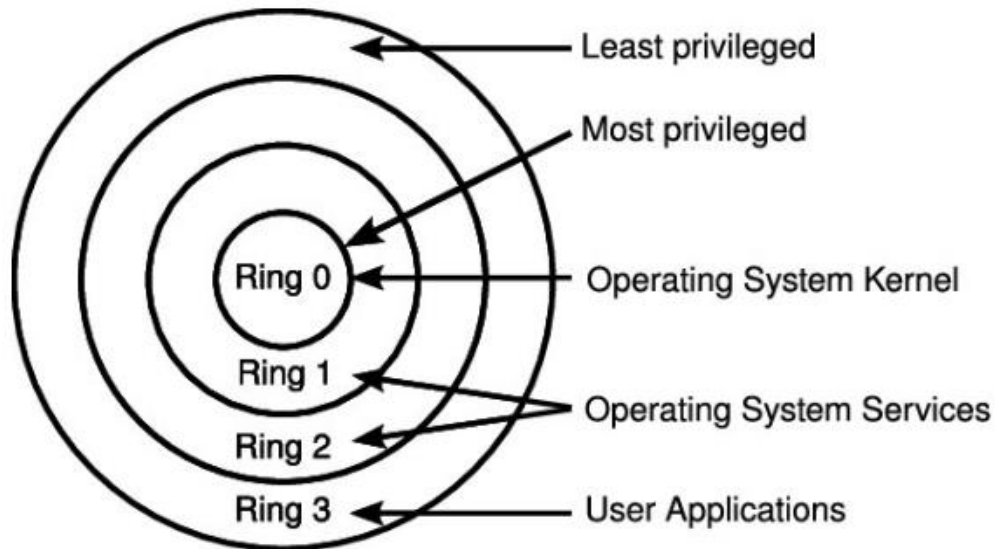
## WORKING OF 8086 MICROPROCESSOR

- **Instruction Fetching:**
  - The BIU fetches the instruction from memory and places it in the instruction queue.
- **Instruction Decoding:**
  - The EU decodes the fetched instruction to understand the operation.
- **Operand Fetching:**
  - If additional data is required (e.g., constants or memory values), the BIU fetches them.
- **Execution:**
  - The EU executes the decoded instruction using the ALU and updates the flag register accordingly.
- **Result Storage:**
  - The processed result is either stored in a register or memory.
- **Next Instruction:**
  - While the EU executes an instruction, the BIU continues fetching the next instruction, enabling pipelining.

## WHAT IS REAL MODE?


1. **Direct Hardware Access:** In **Real Mode**, programs can directly access **any part of memory**, including hardware devices like disk controllers and video memory, without restrictions.
  2. **Unrestricted Memory Access:** There is **no memory protection**, meaning any program can read or overwrite system memory, which can cause crashes.
  3. **"Real" 8086 Behavior:** This mode behaves just like the original **Intel 8086 processor**, which did not have advanced memory protection features.
- Since it provides **raw, unrestricted access to memory**, it is called **Real Mode**—implying that the program operates directly on the **real physical hardware** without interference from the system.

# WHAT IS PROTECTED MODE?



1. **Memory Protection:** In **Protected Mode**, the CPU prevents one program from modifying another program's memory, protecting system stability.
  2. **Privilege Levels:** It introduces **rings (Ring 0, Ring 3, etc.)**, where user applications run with limited access, preventing them from modifying system-critical data.
  3. **Multitasking & Security:** The system ensures that a faulty program **cannot crash the entire computer**, making it much safer than Real Mode.
  4. **Segmentation and Paging:** These features **protect memory** from unauthorized access by keeping processes in separate memory spaces.
- Because this mode enforces **memory protection and controlled access**, it is called **Protected Mode**—emphasizing the security and stability it provides.

# REAL VS PROTECTED MODE

Feature	Real Mode (8086 Mode)	Protected Mode
Addressing	20-bit (1 MB max)	32-bit or 64-bit (Up to 4 GB or more)
Memory Model	Segmented Memory	Paging and Segmentation
Memory Protection	No (Any program can overwrite memory)	Yes (Prevents memory corruption)
Multitasking	No	Yes
Privilege Levels	No (All programs run at the same level)	Yes (Ring 0 - Kernel, Ring 3 - User)
Virtual Memory	No	Yes
Operating Systems	MS-DOS, early BIOS	Windows, Linux, macOS
Use Cases	Legacy systems, booting 	Modern computing, OS operation

## WHY WE NEED REAL & PROTECTED MODE IN MODERN CPU

- Modern x86 CPUs (like Intel Core and AMD) still support **both Real Mode and Protected Mode**, even though modern operating systems primarily use **Protected Mode**.
- Modern CPUs need **both Real Mode and Protected Mode** because:
  1. **Real Mode ensures legacy compatibility** for boot processes and old software.
  2. **Protected Mode (or 64-bit mode) enables modern features** like multitasking, security, and memory protection.
  3. **Mode switching allows flexibility**, such as running old DOS programs or virtual machines.

## WHAT IS AN INTERRUPT? (8086)

- An interrupt is a signal that temporarily pauses the CPU's current execution to handle an urgent task. After executing the interrupt service routine (ISR), the CPU resumes its previous task.
- Hardware Interrupts
- Software Interrupts

## HARDWARE INTERRUPTS

- These interrupts are triggered by external devices through physical signal lines. The 8086 has two hardware interrupt pins:
- **INTR (Interrupt Request):**
- Maskable interrupt — can be enabled/disabled using the IF (Interrupt Flag).
- Requires an Interrupt Acknowledge (INTA) signal from the CPU.
- Example: Keyboard input, printer requests, etc.
- **NMI (Non-Maskable Interrupt):**
- Non-maskable — cannot be disabled by software.
- Always jumps to vector address 00008H.
- Used for critical alerts like hardware failure or power issues.



## SOFTWARE INTERRUPTS

- These are triggered by executing the INT instruction within the program itself.
- Format: INT n (where n is the interrupt number).
- Each software interrupt corresponds to a specific vector address in the Interrupt Vector Table (IVT).
- Example:
- INT 21H → DOS services like printing text, file handling, etc.
- INT 10H → Video display functions.

## INTERNAL (EXCEPTION) INTERRUPTS

- While not always classified as a separate category, these are technically software interrupts but are automatically triggered by the CPU in response to specific conditions:
- Divide Error (INT 0): Triggered when division by zero occurs.
- Single-Step Interrupt (INT 1): Triggered when the Trap Flag (TF) is set for debugging.
- Breakpoint Interrupt (INT 3): Used for setting breakpoints during debugging.
- Overflow Interrupt (INT 4): Triggered by the INTO instruction if the OF (Overflow Flag) is set.

# INTERRUPT VECTOR TABLE (IVT)

The CPU uses this table to determine the address of the corresponding ISR (Interrupt Service Routine).

## Common IVT Entries

Interrupt Number	Vector Address	Purpose
INT 0	00000H	Divide by Zero
INT 1	00004H	Single Step (Debugging)
INT 2	00008H	NMI (Non-Maskable Interrupt)
INT 3	0000CH	Breakpoint (Debugging)
INT 4	00010H	Overflow Error
INT 21H	DOS Service Routine	System Calls (I/O Operations)

VCC	1	40	GND
AD14	2	39	AD15
AD13	3	38	A19/S6
AD12	4	37	A18/S5
AD11	5	36	A17/S4
AD10	6	35	A16/S3
AD9	7	34	S2
AD8	8	33	S1
AD7	9	32	S0
AD6	10	31	READY
AD5	11	30	RESET
AD4	12	29	TEST
AD3	13	28	CLK
AD2	14	27	MN/MX
AD1	15	26	INTA
AD0	16	25	ALE
NMI	17	24	DEN
INTR	18	23	DT/R
GND	19	22	WR
VCC	20	21	RD

8086 PIN Diagram

In the context of the 8086 microprocessor, PIN refers to the physical connection points on the microprocessor package that allow it to communicate with external devices, such as memory and I/O peripherals.

The 8086 microprocessor has a 40-pin dual in-line package (DIP), and each pin has a specific function, including power supply, clock input, address and data transmission, control signals, and interrupt handling.

## 1. Power Supply & Ground Pins

Pin No.	Pin Name	Pin Type	Detailed Description
1	VCC	Power	Provides +5V power supply to the microprocessor.
20	VCC	Power	Additional +5V power supply pin for stable power distribution.
19	GND	Power	Ground connection, ensures circuit completion and reference voltage.
40	GND	Power	Additional ground pin for better stability and noise reduction.

## 2. Clock & Control Pins

Pin No.	Pin Name	Pin Type	Detailed Description
28	CLK	Input	Provides the main clock signal for synchronizing all internal operations of the processor. Typically 5 MHz, 8 MHz, or 10 MHz.
30	RESET	Input	A high signal resets the processor, clearing registers and flushing the instruction queue. Must be held high for at least 4 clock cycles.
31	READY	Input	Used to insert wait states for slower memory or I/O devices. The processor waits when this pin is low.
29	TEST	Input	Used for debugging and external synchronization. The execution stops when this pin is low in WAIT instruction execution.

### Purpose of the CLK Pin:

- Synchronization: Ensures that all internal processes (such as instruction execution and data transfer) happen in a coordinated manner.
- Timing Control: Defines the speed at which the processor executes instructions by setting the clock cycle duration.
- Bus Operations: Controls the timing of communication between the processor, memory, and peripheral devices.

### Purpose of the RESET Pin:

- Clears Registers and Flags: Resets all internal registers, including the instruction pointer (IP), ensuring a clean start.
- Forces Execution from a Known Address: After a reset, the processor starts execution from memory location FFFF0H (the reset vector in ROM).
- Stops Current Execution: Any ongoing instruction execution or bus activity is halted.
- Flushes Internal Queues: The instruction prefetch queue is cleared to prevent execution of corrupted instructions.
- Used for System Initialization: Ensures proper system startup, often triggered by power-on reset circuits or manually via external hardware.

### Purpose of the READY Pin:

Wait State Control: If a memory or I/O device is too slow to respond, the READY pin is used to introduce wait states, preventing data loss.

Synchronization: Ensures that the CPU does not proceed with the next operation until the external device is ready.

Dynamic Speed Adjustment: Allows the 8086 to work with peripherals that have varying response times.

How It Works:

If READY = 1 (High) → The processor continues normal operation.

If READY = 0 (Low) → The processor inserts wait states and pauses execution until READY goes high.

### Purpose of the TEST Pin:

Wait State Control: The CPU executes a WAIT instruction and checks the TEST pin.

Conditional Execution: If the TEST pin is low (0), the processor continues execution. If it is high (1), the processor remains in a wait state until it goes low.

Synchronization with External Devices: It allows external hardware to signal the processor when it is ready for the next operation.



### 3. Address/Data Bus Pins (Multiplexed AD Lines)

Pin No.	Pin Name	Pin Type	Detailed Description
2 - 16	AD0 - AD14	Bidirectional	Multiplexed address/data bus used for both memory addressing and data transfer. Lower 16 bits are used for address during T1 and data transfer during T2-T4.
39	AD15	Bidirectional	Most significant bit of the address/data multiplexed bus.

Pin No.	During T1 (Address Phase)	During T2, T3, T4 (Status Phase in Max Mode)
35	A16	S3 (Segment Register Status)
36	A17	S4 (Segment Register Status)
37	A18	S5 (Interrupt Enable Flag Status)
38	A19	S6 (Always 0, Reserved)

### Multiplexing in the First Clock Cycle (T1)

During the first clock cycle (T1) of a bus cycle, pins 35, 36, 37, and 38 carry the high-order address bits A16,A17, A18, and A19, respectively.

This is necessary because 8086 has a 20-bit address bus, but only 16 dedicated address lines (A0–A15). The higher address lines (A16–A19) are multiplexed with S3–S6.

### Status Signals in Later Clock Cycles (T2,T3,T4)

After T1, in the remaining cycles (T2,T3, and T4), the same pins are used as status signals (S3–S6) instead of address lines.

These status signals help the 8288 Bus Controller generate appropriate control signals.

Pin	Function	S4	S3	Segment Register Used
S3, S4	Indicate which segment register is being used for memory access.	0	0	Extra Segment (ES)
S5	Reflects the state of the Interrupt Enable (IF) flag.	0	1	Stack Segment (SS)
		1	0	Code Segment (CS)
S6	Always logic 0 (Reserved for future use).	1	1	Data Segment (DS)

## 5. Status & Mode Control Signals

Pin No.	Pin Name	Pin Type	Detailed Description
32	S0	Output	Status signal used to indicate processor operation in maximum mode.
33	S1	Output	Status signal used to indicate processor operation in maximum mode.
34	S2	Output	Status signal used to indicate processor operation in maximum mode.
27	MN/M $\bar{X}$	Input	Mode selection pin: High (1) for minimum mode, Low (0) for maximum mode.

## Bus Cycle Operation Table for S0, S1, S2

S2	S1	S0	Operation Type
0	0	0	Interrupt Acknowledge
0	0	1	I/O Read
0	1	0	I/O Write
0	1	1	Halt
1	0	0	Opcode Fetch
1	0	1	Memory Read
1	1	0	Memory Write
1	1	1	Passive (No Operation)

In minimum mode ( $MN/\overline{MX} = 1$ ) of the 8086 microprocessor, the S0, S1, and S2 signals are not used. These signals are only relevant in maximum mode ( $MN/\overline{MX} = 0$ ), where they help the 8288 Bus Controller generate control signals.

## Key Differences Between Minimum and Maximum Mode

Feature	Minimum Mode	Maximum Mode
MN/M $\bar{X}$ Pin (33)	HIGH (1)	LOW (0)
No. of Processors	Single Processor	Multiprocessor System
Control Signals	Generated by 8086	Generated by 8288 Bus Controller
System Complexity	Simple	Complex
External Bus Controller	Not Required	Required (8288)
Multiprocessing Support	No	Yes (Supports 8087, 8089)
Example Use Cases	Simple computers	High-end servers, coprocessor systems

## Summary: Processors and Support Chips in Maximum Mode

Processor/Chip	Purpose
8086 (Main CPU)	Executes instructions
8087 (Co-Processor)	Handles floating-point arithmetic
8288 Bus Controller	Generates control signals
8289 Bus Arbiter	Manages multiple processors on the bus
8259 PIC	Handles interrupts
8237 DMA Controller	Enables direct memory access

## 6. Control & Synchronization Pins

Pin No.	Pin Name	Pin Type	Detailed Description
21	$\overline{RD}$	Output	Active low Read control signal. Indicates that the processor is performing a memory or I/O read operation.
22	$\overline{WR}$	Output	Active low Write control signal. Indicates that the processor is performing a memory or I/O write operation.
23	DT/ $\overline{R}$	Output	Data Transmit/Receive signal. Determines whether the processor is sending or receiving data.
24	$\overline{DEN}$	Output	Data Enable signal. Enables the transceiver to separate data from the multiplexed bus.
25	ALE	Output	Address Latch Enable. Used to latch the address from the multiplexed address/data bus during T1 state.

$\overline{DEN}$  is used to enable or disable external data transceivers (bus drivers).

It ensures that only one device drives the data bus at a time, avoiding conflicts.

◆ The 8086 microprocessor has a multiplexed address/data bus (AD0–AD15), meaning these 16 lines are used for both address and data at different times.

◆ ALE is used to latch (store) the address so that the same lines can be used for data transfer afterward.



## 7. Interrupt Control Pins

Pin No.	Pin Name	Pin Type	Detailed Description
17	NMI	Input	Non-Maskable Interrupt. A high-priority interrupt that cannot be disabled by software. Used for critical errors like hardware failures.
18	INTR	Input	Interrupt Request. A general-purpose interrupt that can be enabled/disabled using software instructions.
26	INT $\bar{A}$	Output	Interrupt Acknowledge. Activated when the processor acknowledges an interrupt request.

# WHAT IS TWO STAGE PIPELINING IN 8086?

## 1. Fetch Cycle (Fetching the Instruction from Memory)

Purpose: Retrieve the instruction from memory into the CPU.

Components Involved:

Code Segment (CS) → Holds the base address of the program code.

Instruction Pointer (IP) → Holds the offset address of the next instruction.

Bus Interface Unit (BIU) → Handles memory accesses and instruction queue.

- Prefetch Queue → Stores fetched instructions.

## Step-by-Step Fetch Process in 8086:

### Calculate the Physical Address

8086 uses a segmented memory model:

Physical Address = ( CS × 16 ) + IP

Example: If CS = 2000H and IP = 0030H, then  $(2000H \times 10H) + 0030H = 20030H$

The instruction is fetched from memory location 20030H.

### Fetch Instruction from Memory

BIU requests the instruction from memory and loads it into the Instruction Queue (6-byte FIFO queue).

The fetched instruction is temporarily stored in the queue until required by the EU.

### Increment IP

IP is incremented to point to the next instruction (based on instruction length).

If instruction is 1-byte,  $IP = IP + 1$ , if 2-byte,  $IP = IP + 2$ , etc.

-  At the end of the fetch cycle, the instruction is stored in the queue, and IP points to the next instruction.

## Decode Cycle (Decoding the Instruction)

Purpose: Identify the type of instruction and determine how to execute it.

Components Involved:

Execution Unit (EU) → Decodes and executes instructions.

Control Unit (CU) → Identifies the operation and addressing mode.

General Purpose Registers (AX, BX, CX, DX, etc.) → May be involved as operands.

Addressing Mode Decoder → Determines addressing mode (Register, Immediate, Memory).

### Step-by-Step Decode Process in 8086:

#### Fetch Instruction from the Queue

EU takes the instruction from the Instruction Queue and sends it to the Control Unit (CU).

#### Opcode Extraction

The Control Unit extracts the opcode from the instruction.

Example: MOV AX, BX

- Opcode = MOV, Operands = AX, BX.

## Execute Cycle (Executing the Instruction)

Purpose: Perform the actual operation and store the result.

Components Involved:

Arithmetic Logic Unit (ALU) → Performs arithmetic/logic operations.

General Purpose Registers (AX, BX, etc.) → Store data and results.

Control Unit (CU) → Directs execution.

Flags Register → Stores condition flags (Zero, Carry, Sign, etc.).

## Step-by-Step Execute Process in 8086:

### Fetch Operands (if needed)

If an operand is in memory, BIU calculates the memory address using Segment Registers and retrieves the data.

Example: MOV AX, [5000H]

BIU fetches value from memory address 5000H.

- Value is loaded into AX.

### Perform ALU Operation (if required)

If the instruction involves computation, the ALU executes it.

Example: ADD AX, BX

ALU performs  $AX = AX + BX$ .

### Store Result

The result is stored in the destination register or memory.

Example: MOV [6000H], AX → Stores AX content in memory location 6000H.

### Update Flags Register (if needed)

If the instruction affects flags (Zero, Carry, Sign, etc.), they are updated accordingly.

At the end of the execute cycle, the instruction is fully processed, and the CPU moves to the next instruction.



**Thank You!**

