**Microprocessor (2024-25)**

PT-1 Question Bank

**Module 1**

Draw and explain 8086 CPU Architecture.

**1. Bus Interface Unit (BIU)**

The BIU is responsible for fetching instructions and data from memory. The BIU has the following tasks:

* **Instruction Prefetching**: It fetches instructions and stores them in the instruction queue (6 bytes).
* **Memory Access**: It provides the interface between the 8086 and the system memory (RAM) and I/O devices. It uses the **address bus** and **data bus** to communicate with memory and I/O devices.
* **Segment Registers**: It holds the four segment registers: CS (Code Segment), DS (Data Segment), SS (Stack Segment), and ES (Extra Segment), which are used for addressing memory locations.

**2. Execution Unit (EU)**

The EU is responsible for executing instructions that are fetched by the BIU. It contains the following:

* **ALU (Arithmetic Logic Unit)**: Performs arithmetic and logical operations.
* **General-Purpose Registers**: These are 16-bit registers used for operations. They include:
  + **AX** (Accumulator)
  + **BX** (Base Register)
  + **CX** (Count Register)
  + **DX** (Data Register)
* **Index and Pointer Registers**: Used for addressing and indexing:
  + **SI** (Source Index)
  + **DI** (Destination Index)
  + **SP** (Stack Pointer)
  + **BP** (Base Pointer)
* **Flags Register**: The flags register contains the status flags (e.g., Zero Flag, Carry Flag) and control flags (e.g., Interrupt Enable Flag) that reflect the outcome of ALU operations.

**3. Control Unit (CU)**

The **Control Unit** decodes the instructions and generates the necessary control signals for the execution of each instruction. It coordinates the operations between the BIU and the EU, ensuring that each instruction is executed in the correct sequence.

**4. Registers**

The **registers** are used for storing data and addresses. The 8086 has a combination of general-purpose, pointer, index, and segment registers.

* **General-Purpose Registers**:
  + **AX**: Used for arithmetic and data transfer operations.
  + **BX**: Used for base addressing.
  + **CX**: Used for loop counting or shifting operations.
  + **DX**: Used in I/O operations and for multiplication and division.
* **Pointer and Index Registers**:
  + **SP (Stack Pointer)**: Points to the top of the stack.
  + **BP (Base Pointer)**: Used for accessing stack data.
  + **SI (Source Index)** and **DI (Destination Index)**: Used for string and memory operations.
* **Segment Registers**:
  + **CS (Code Segment)**: Holds the address of the current code being executed.
  + **DS (Data Segment)**: Holds the address of data variables.
  + **SS (Stack Segment)**: Holds the address of the stack.
  + **ES (Extra Segment)**: Used for extra data storage (e.g., during string operations).
* **Instruction Pointer (IP)**: The IP holds the address of the next instruction to be executed.

**5. Instruction Queue**

The 8086 features a **6-byte instruction queue**. This queue allows the microprocessor to prefetch the next set of instructions while executing the current instruction, improving processing efficiency.

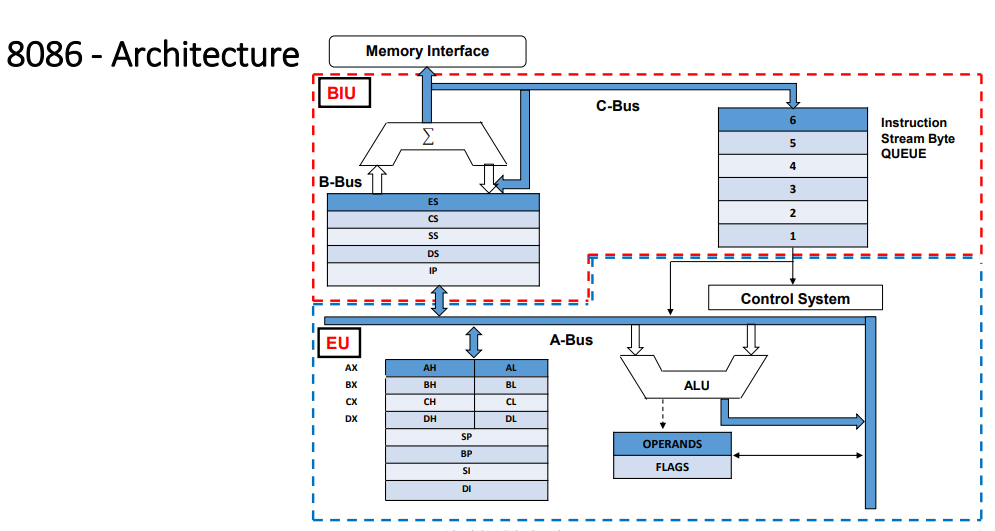
**6. Memory Segmentation**

The 8086 uses a **segmented memory model**:

* The **address bus** is 20 bits wide, which means it can address up to **1 MB** of memory (2^20 = 1,048,576 bytes).
* Memory is divided into **segments**, each with a maximum size of **64 KB**.
* The **segment registers** (CS, DS, SS, ES) define the starting point of each segment, and the offset within the segment is given by a 16-bit value.

**7. Address Bus and Data Bus**

* **Address Bus (20 bits)**: The 8086 has a 20-bit address bus, allowing it to access 1 MB of memory.
* **Data Bus (16 bits)**: The 8086 is a 16-bit processor, so the data bus is 16 bits wide, allowing the transfer of 16 bits of data at a time.



Explain Memory Segmentation related to 8086 microprocessor.

BIU contains 4 special purpose registers called Segment Registers

Segment Registers hold the upper 16 bits of the base/ starting address

of the 4 memory segments

i. Code Segement (CS) – Program memory

ii. Data Segement (DS) – Data memory

iii. Stack Segement (SS) – Stack memory

iv. Extra Segement (ES) – Extra Segment

**Rules for Memory Segmentation:**

1. The four segments can overlap for small programs. In minimum

system all four segments can start at the address 00000H.

2. The segment can begin/start at any memory address which is

divisible by 16.

**Advantages**

1. It allows the memory addressing capacity to be 1 MByte even though the

address associated with individual instruction is only 16-bit.

2. It allows instruction code, data, stack, and portion of program to be more

than 64KB long by using more than one code, data, stack segment , and

extra segment.

3. It facilitates use of separate memory areas for program, data, and stack.

4. It permits a program or its data to be put in different areas of memory,

each time the program is executed i.e., program can be relocated which is

very useful in multiprogramming.

Explain Banking in 8086.

• 8086 has a 16-bit data bus  it can access 16- bit data in one

operation.

• But memory chips available are normally such that one memory

location has 8 bit (1 byte).

• 1 Memory locations carries one byte- 8 bits.

• To access 16-bit data it needs to read 2 memory locations.

• If both memory locations are consecutive in the same memory chip then

the address bus has to contain 2 addresses at the same time (which is

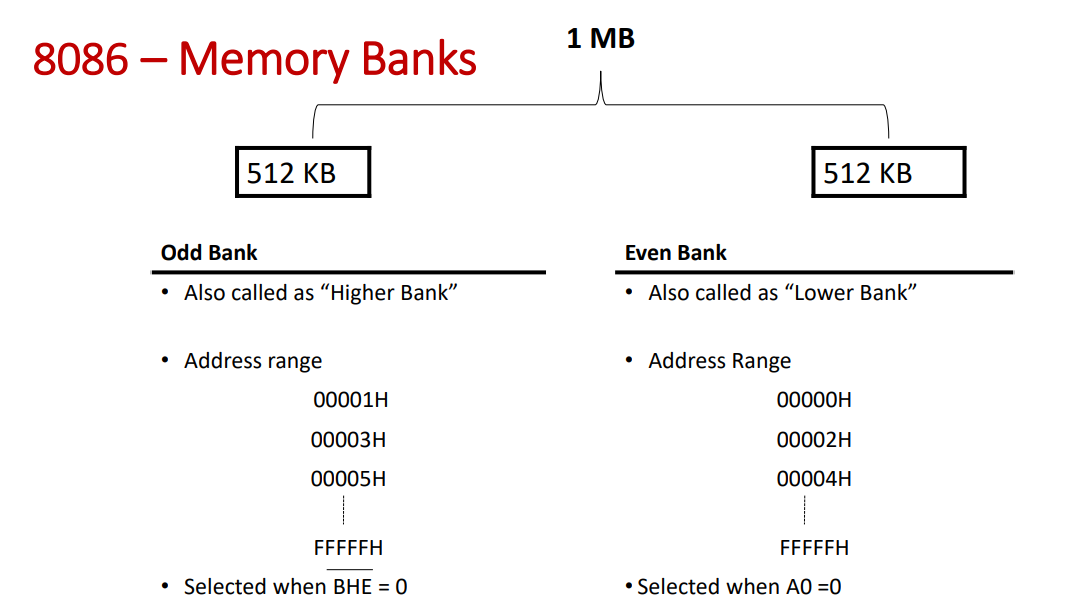
impossible) and hence require double time.

• Therefore to solve this problem, the memory of 8086 is divided into 2 banks.

• Each bank provides 1byte or 8bits.

• One bank contains all even addresses called “Even Bank”.

• The other bank contains all odd addresses called “Odd Bank”.



Explain Demultiplexing of Address/Data bus of 8086.

**Demultiplexing of Address/Data Bus in 8086 Microprocessor**

In the **8086 microprocessor**, **address and data lines** are multiplexed, meaning both **address** and **data** share the same physical bus lines. The 8086 has a **16-bit data bus** and a **20-bit address bus**. Due to pin constraints, the address and data are transmitted in two phases within the same bus cycle. The process of **demultiplexing** refers to separating the **address** from the **data** to ensure correct operation.

**Working of Demultiplexing:**

1. **Multiplexed Bus**:
   * The address and data share the same physical lines (A0-A19 and D0-D15). Initially, the address is placed on the bus, followed by the data.
2. **Control Signal - ALE (Address Latch Enable)**:
   * The **ALE (Address Latch Enable)** signal is used to latch the **address** from the multiplexed bus.
   * **ALE** is **high** during the **address phase** and **low** during the **data phase**.
   * When **ALE** is high, the address is latched by an address latch and stored separately for further use.
3. **Address Latch**:
   * The **address latch** holds the address value during the data phase, ensuring the address remains stable while the data is being transferred.
4. **Data Phase**:
   * After the address is latched, the **ALE signal goes low**, and the **data** is placed on the bus (D0-D15). This phase handles data reading or writing from/to memory or I/O devices.

**Steps in a Read Cycle:**

1. **Address Phase**:
   * The microprocessor places the **address** on the multiplexed bus (A0-A19), and the **ALE signal** is high.
   * The address is latched by the address latch.
2. **Data Phase**:
   * The **ALE signal** goes low.
   * The microprocessor places the **data** (D0-D15) on the bus for reading from or writing to memory or I/O.

**Importance of Demultiplexing:**

* **Pin Efficiency**: Reduces the number of pins needed on the microprocessor by sharing address and data lines.
* **Memory and I/O Access**: Allows both address and data to be transmitted over the same bus lines, making the design more efficient.
* **Synchronization**: Ensures that the address and data phases are properly separated using the **ALE** control signal.

Draw and explain Timing diagrams for Read and Write operations in minimum mode.

**1. Timing Diagram for Read Operation (8086 in Minimum Mode)**

In a **read operation**, the 8086 reads data from memory or I/O. Here's how the timing is represented in a diagram:

**Control Signals Involved:**

* **ALE (Address Latch Enable)**: Latches the address onto the address bus.
* **M/IO (Memory/I/O)**: Determines whether the operation is for memory or I/O.
* **IOR (I/O Read)**: Indicates a read operation from I/O.
* **RD (Read)**: Indicates a read operation from memory.

**Timing Diagram Explanation:**

1. **T1 Cycle (Address Phase)**:
   * The **address** is placed on the multiplexed address/data bus (A0-A19) by the 8086.
   * The **ALE signal** goes **high**, which **latches** the address into the external latches (address bus becomes stable).
   * **M/IO** is set low to indicate a memory or I/O operation.
2. **T2 Cycle**:
   * The **RD signal** goes **low**, indicating that the read operation is in progress.
   * The **data bus (D0-D15)** holds the data for the read operation. However, the **data** will not be stable until the next clock cycle.
   * The **IOR signal** stays **high**, indicating that the operation is not for I/O.
3. **T3 Cycle (Data Phase)**:
   * **Data** is placed on the **data bus** from the memory or I/O.
   * **RD** remains **low** for the entire data phase.
4. **T4 Cycle**:
   * **RD** goes **high**, ending the read cycle.
   * The **data bus** returns to a high-impedance state after the data transfer.

**Timing Diagram for Write Operation (8086 in Minimum Mode)**

In a **write operation**, the 8086 writes data to memory or I/O. The **timing diagram** for a write operation looks like this:

**Control Signals Involved:**

* **ALE (Address Latch Enable)**: Latches the address onto the address bus.
* **M/IO (Memory/I/O)**: Indicates whether the operation is for memory or I/O.
* **IOW (I/O Write)**: Indicates a write operation to I/O.
* **WR (Write)**: Indicates a write operation to memory.

**Timing Diagram Explanation:**

1. **T1 Cycle (Address Phase)**:
   * The **address** is placed on the multiplexed address/data bus (A0-A19) by the 8086.
   * The **ALE signal** goes **high**, latching the address into external latches, making the address stable on the bus.
   * **M/IO** goes low, indicating it is a memory operation.
2. **T2 Cycle**:
   * The **WR signal** goes **low**, indicating that data is being written to memory.
   * The **data bus** (D0-D15) carries the data that is to be written to the memory or I/O.
   * **M/IO** remains low, indicating that it is a memory write operation.
3. **T3 Cycle (Data Phase)**:
   * Data is transferred from the **data bus** to memory or I/O during this phase.
   * The **WR signal** remains low during the entire write cycle to ensure that the data is written properly.
4. **T4 Cycle**:
   * **WR** goes **high**, completing the write cycle.
   * The **data bus** returns to a high-impedance state.

Flag register of 8086 with suitable diagram.

The **Flag Register** is a 16-bit register, and it consists of the following flags:

1. **Sign Flag (S)**: (Bit 15)
   * Set if the result of an operation is negative (most significant bit of the result is 1).
   * Cleared if the result is positive (most significant bit of the result is 0).
2. **Zero Flag (Z)**: (Bit 14)
   * Set if the result of an operation is zero.
   * Cleared if the result is non-zero.
3. **Auxiliary Carry Flag (AC)**: (Bit 12)
   * Set if there is a carry out from the lower nibble (bit 3) during arithmetic operations.
   * Used primarily for **BCD (Binary Coded Decimal)** arithmetic.
4. **Parity Flag (P)**: (Bit 2)
   * Set if the number of 1s in the result is even.
   * Cleared if the number of 1s in the result is odd.
5. **Carry Flag (C)**: (Bit 0)
   * Set if there is a carry out of the most significant bit (in an addition operation) or a borrow in subtraction.
   * Cleared if no carry or borrow occurred.
6. **Overflow Flag (O)**: (Bit 11)
   * Set if there is a signed overflow, i.e., when the result of an arithmetic operation exceeds the range that can be represented in the result (for signed numbers).
   * Cleared if no overflow occurs.
7. **Trap Flag (T)**: (Bit 8)
   * Set to enable single-step debugging, causing the processor to generate an interrupt after each instruction is executed.
   * Used for debugging purposes.
8. **Interrupt Flag (I)**: (Bit 9)
   * Set if interrupts are enabled.
   * Cleared if interrupts are disabled, thus preventing the processor from recognizing interrupt requests.
9. **Direction Flag (D)**: (Bit 10)
   * Controls the direction of string operations (like **MOVSB**, **MOVSW**).
   * If set, string operations process from **high memory to low memory**.
   * If cleared, string operations process from **low memory to high memory**.

**Flag Register Format (16 bits)**

| **Bit Position** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Flag | **S** | **Z** | - | **AC** | **O** | **D** | **I** | **T** | - | - | - | - | **P** | - | **C** |  |

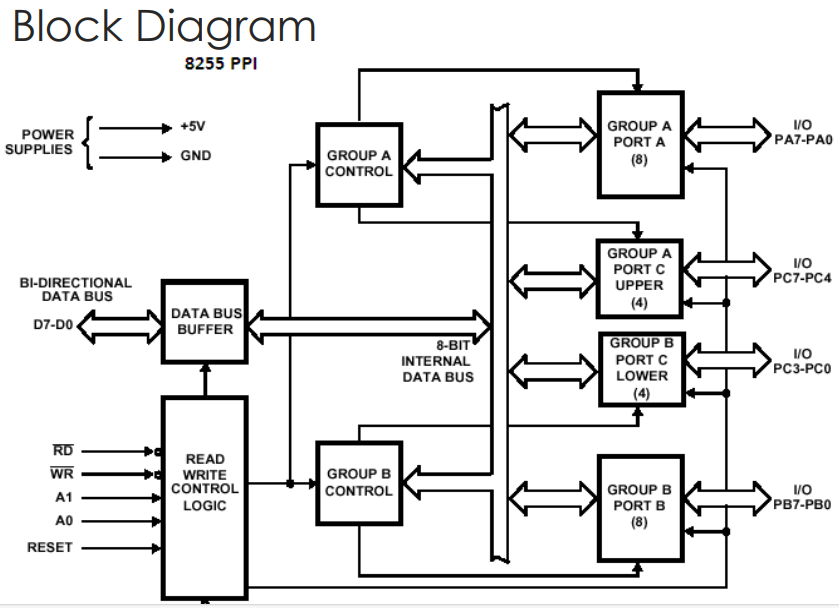
**Module 3**

Draw and explain architecture of 8255 PPI.

**Architecture of the 8255 PPI**

The 8255 PPI is typically structured as follows:

1. **Control Register (CR)**:
   * The control register is used to configure the operation mode of the 8255.
   * It selects the input/output operation and enables different modes like **Mode 0**, **Mode 1**, and **Mode 2**.
   * The control register is a write-only register and can only be written to by the microprocessor.
   * The control register is divided into several bits that specify the direction of data flow (input or output) and the mode of operation for each port.
2. **Data Bus**:
   * This is a bidirectional 8-bit bus that is used to transfer data between the microprocessor and peripheral devices.
   * It is connected to the I/O ports (Port A, Port B, and Port C) to facilitate data transfer.
3. **I/O Ports**:
   * The 8255 has three I/O ports: **Port A**, **Port B**, and **Port C**.
   * **Port A** and **Port B** are 8-bit ports that can be configured as input or output.
   * **Port C** is also an 8-bit port, but it can serve as either an input, output, or control port, depending on the configuration.
4. **Port A and Port B**:
   * These ports can be configured as **input** or **output** based on the mode selected.
   * **Mode 0** allows these ports to function in simple input or output modes.
   * **Mode 1** allows the use of these ports in handshaking or interrupt-driven operations.
   * **Mode 2** provides bi-directional data transfer between the microprocessor and peripherals.
5. **Port C**:
   * **Port C** is divided into two halves: **Port C Upper (PCU)** and **Port C Lower (PCL)**.
   * It functions as a control port or can be divided to work in different modes.
   * **Mode 0** uses Port C as a simple input/output port.
   * **Mode 1** and **Mode 2** use it as a control line to implement handshake operations.
6. **Read/Write Control Logic**:
   * This logic is responsible for controlling the data flow to and from the microprocessor.
   * It manages the read/write cycles for the data and control registers.
7. **Address Decoder**:
   * The address decoder ensures that the correct address is assigned to each of the registers (data and control) of the 8255.
   * It decodes the address from the microprocessor to access the correct port or control register.



Explain control word formats of 8255 PPI.

 **Bit 7 and Bit 6 (Set to 1)**:

* These two bits are always set to 1. This indicates that the control word is for programming the **8255** (not a different register).

 **Bits 5, 4, and 3 (Mode Selection: M2, M1, M0)**:

* These bits are used to select the operating **mode** of the 8255. The 3-bit combination of these bits defines whether the 8255 works in **Mode 0**, **Mode 1**, or **Mode 2**. These bits control how Port A, Port B, and Port C are configured.
* **M2, M1, M0** combinations:

| **M2** | **M1** | **M0** | **Mode Description** |
| --- | --- | --- | --- |
| 0 | 0 | 0 | Mode 0: Basic Input/Output (no handshaking) |
| 0 | 0 | 1 | Mode 1: Input/Output with Handshaking |
| 0 | 1 | 0 | Mode 2: Bidirectional Data Transfer (for Port A) |
| 0 | 1 | 1 | Mode 2: Not Used |
| 1 | 0 | 0 | Mode 3: Not Used |
| 1 | 0 | 1 | Mode 3: Not Used |
| 1 | 1 | 0 | Mode 3: Not Used |
| 1 | 1 | 1 | Mode 3: Not Used |

* **Mode 0 (Basic I/O Mode)**: Ports A, B, and C are configured as simple input/output ports.
* **Mode 1 (Input/Output with Handshaking)**: Provides handshaking signals for data transfer, typically for Port A and Port B.
* **Mode 2 (Bidirectional Data Transfer Mode)**: Port A supports bidirectional data transfer; Port C serves as a control port.

 **Bits 2, 1, and 0 (Port C Configuration: C2, C1, C0)**:

* These three bits determine the configuration of **Port C**. Port C is divided into two 4-bit halves: the **upper 4 bits** and the **lower 4 bits**. These bits define whether Port C should be used as an input port, output port, or control port.
* Depending on the mode (Mode 0, 1, or 2), these bits can control the direction of data flow on **Port C** and configure whether it is used for handshaking, input, or output.

Here's a description of the **C2, C1, C0** configuration options:

| **C2** | **C1** | **C0** | **Port C Function** |
| --- | --- | --- | --- |
| 0 | 0 | 0 | Port C Lower (PCL) and Port C Upper (PCU) as Input |
| 0 | 0 | 1 | Port C Lower (PCL) and Port C Upper (PCU) as Output |
| 0 | 1 | 0 | Port C Lower (PCL) as Input; Port C Upper (PCU) as Output |
| 0 | 1 | 1 | Port C Lower (PCL) as Output; Port C Upper (PCU) as Input |
| 1 | 0 | 0 | Port C Lower (PCL) and Port C Upper (PCU) as Control Lines |
| 1 | 0 | 1 | Port C Lower (PCL) as Control; Port C Upper (PCU) as Input |
| 1 | 1 | 0 | Port C Lower (PCL) as Input; Port C Upper (PCU) as Control |
| 1 | 1 | 1 | Port C Lower (PCL) as Output; Port C Upper (PCU) as Control |

**Example 1: Mode 0, Ports A and B as Output, Port C as Input**

* **Mode (M2, M1, M0) = 000**: Mode 0 (Basic Input/Output Mode)
* **Port C Configuration (C2, C1, C0) = 000**: Port C is configured as an input port for both upper and lower halves.

Control word:

CopyEdit

1 1 0 0 0 0 0 0

This means:

* M2 = 0, M1 = 0, M0 = 0 → **Mode 0** (Basic I/O)
* C2 = 0, C1 = 0, C0 = 0 → **Port C** is configured as input.

**Example 2: Mode 1, Ports A and B with Handshaking, Port C as Control**

* **Mode (M2, M1, M0) = 001**: Mode 1 (Input/Output with Handshaking)
* **Port C Configuration (C2, C1, C0) = 110**: Port C is configured as control lines.

Control word:

CopyEdit

1 1 0 0 1 1 0 0

This means:

* M2 = 0, M1 = 0, M0 = 1 → **Mode 1** (Handshaking)
* C2 = 1, C1 = 1, C0 = 0 → **Port C** as control lines.

