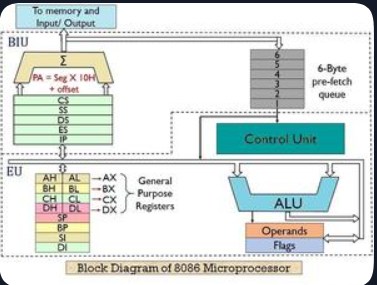
**Question Solve 19-20 season**

**1. Write down the steps to execute a machine instruction. Illustrate the intel 8086 Microprocessor organization.**

Ans: **Steps to Execute a Machine Instruction (8086):**

1. **Fetch:**  
   The **Bus Interface Unit (BIU)** fetches the instruction from memory using the **Instruction Pointer (IP)** and stores it in the **Instruction Queue**.
2. **Decode:**  
   The **Execution Unit (EU)** takes the instruction from the queue and decodes it to understand the operation.
3. **Fetch Operands:**  
   The EU or BIU fetches the required operands from registers or memory.
4. **Execute:**  
   The **Arithmetic and Logic Unit (ALU)** performs the operation specified by the instruction.
5. **Store Result:**  
   The result is stored back in a register or memory.
6. **Update IP:**  
   The **Instruction Pointer (IP)** is updated to point to the next instruction.

**Intel 8086 Microprocessor Organization:**



**Summary:**

* **BIU** handles **fetching and addressing**.
* **EU** handles **decoding and execution**.  
  Both units work **in parallel** to improve speed.

**2. Define memory segment. Write down the feature of 80286 microprocessor.**

**Memory Segment:**  
A **memory segment** is a **block of memory** used by the processor to store a specific type of data or code.  
In the **Intel 8086/80286**, memory is divided into segments like:

* **Code Segment (CS)** – holds program instructions.
* **Data Segment (DS)** – holds data.
* **Stack Segment (SS)** – holds stack data.
* **Extra Segment (ES)** – used for additional data storage.

Each segment can be up to **64 KB** in 8086 and **up to 1 MB (protected mode)** in 80286.

**Features of Intel 80286 Microprocessor:**

1. **16-bit microprocessor** with **24-bit address bus**.
2. Can **address up to 16 MB of physical memory**.
3. Supports **Protected Mode** and **Real Mode** operation.
4. **Clock speed:** 6–25 MHz.
5. Has **segmentation with memory protection** (prevents illegal access).
6. **20-bit address bus in real mode**, **24-bit in protected mode**.
7. Improved **instruction execution speed** compared to 8086.
8. Supports **multitasking** and **virtual memory**.

**3. Write down the difference between physical and logical memory. A memory location has physical address 80FD2h. In what segment does it have offset BFD2h?**

Ans: **Difference between Logical and Physical Memory:**

| **Logical Memory** | **Physical Memory** |
| --- | --- |
| CPU-generated (segment + offset) address | Actual address in RAM |
| Used by programs | Used by hardware |
| Example: 2000:0010h | Example: 20010h |

**Given:**  
Physical = 80FD2h, Offset = BFD2h

**Formula:**  
Physical = (Segment × 10h) + Offset

**So,**  
Segment = (80FD2h − BFD2h) / 10h = **8000h**

✅ **Answer:** Segment = **8000h**, Offset = **BFD2h** (Logical address = 8000:BFD2h)

**4. Which inter microprocessor addresses 1T of memory? What is the purpose of the microprocessor in a microprocessor-based computer.**

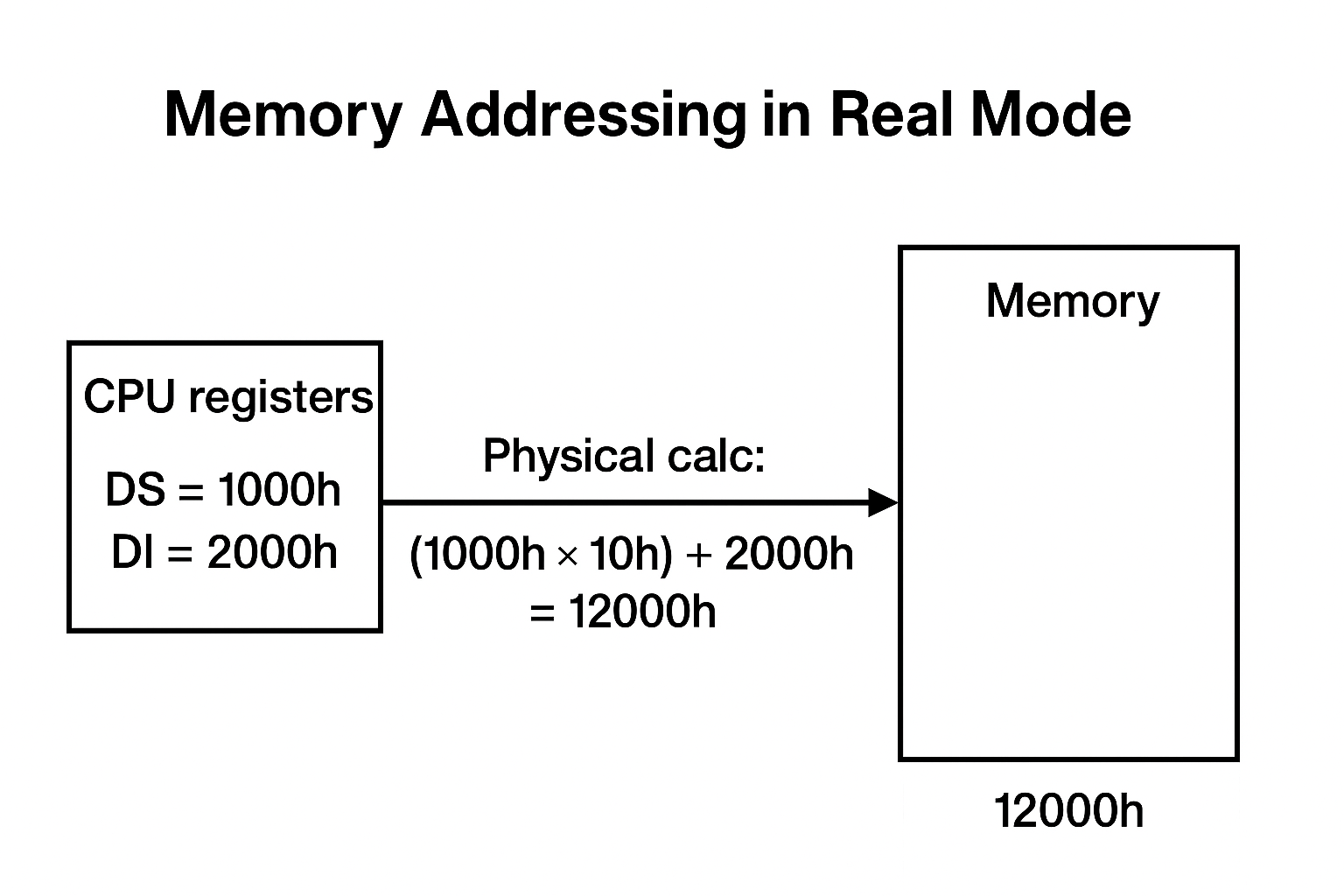
Ans: The **Intel Core 2 (x86-64)** microprocessor can address **1 TB of memory**.  
The **microprocessor** acts as the **CPU** — it **fetches, decodes, and executes** instructions to control all operations of the computer.

**5. Determine the memory location addressed by the following real mode 80286 register combinations: DS = 1000H and DI = 2000H also the diagram of the memory access.**

Ans: **Physical Address = 1000H × 10H + 2000H = 12000H**

In real mode, 80286 uses:

So, DS = 1000H and DI = 2000H gives **12000H**.



**Real Question Answer of 2019 -20 Start from here**

**The Question number 01**

**(a) How large is the windows application programming area? Distinguish between microcontroller and microprocessor. Give me features of 8085.**

**Ans:** Windows Application Programming Area

* The Windows application programming area refers to the user-mode address space available to applications in a Windows system.
* In 32-bit Windows, each process typically has 4 GB of virtual address space, split as:
  + 2 GB for the user mode (applications)
  + 2 GB for the kernel mode (OS)
* In 64-bit Windows, the user-mode space can be much larger (theoretically up to 8 TB in modern systems), though practical limits are smaller.

**Microcontroller vs Microprocessor**

| **Feature** | **Microcontroller** | **Microprocessor** |
| --- | --- | --- |
| Components | CPU + memory + I/O on one chip | CPU only, memory external |
| Use | Embedded devices, IoT | PCs, laptops |
| Power & Cost | Low | High |
| Real-time | Good | Usually no |
| Examples | 8051, PIC, AVR | 8085, 8086 |

**8085 Microprocessor Features**

* **8-bit CPU**, **16-bit address bus** (64 KB memory)
* Single +5 V supply, TTL logic
* **Registers:** A, B, C, D, E, H, L, PC, SP
* **Instruction set:** 74 instructions (arithmetic, logic, control)
* **Interrupts:** 5 hardware interrupts
* Supports **serial I/O**, real-time interfacing

**(b) What is program-visible registers? What is the purpose of the IP/EIP register? Determine the memory location addressed by the following mode 80286/Core2 register combinations:**

**i. DS = 1000H and DI = 2000H.**

**ii. DS = 2000H and EAX = 00003000H.**

**iii. SS = 8000H and ESP = 00009000H**

Ans: **Program-Visible Registers**

* Registers that can be directly accessed and modified by a programmer through instructions.
* Examples: **AX, BX, CX, DX, SI, DI, BP, SP, IP/EIP, segment registers (CS, DS, ES, SS), flags**.

**Purpose of IP/EIP Register**

* **IP (Instruction Pointer)** in 8086/80286 or **EIP** in 32-bit CPUs.
* Holds the **offset address of the next instruction** to execute in the **code segment**.
* Automatically updated after each instruction fetch.

**Memory Address Calculation**

* **i. DS=1000H, DI=2000H → 12000H**
* **ii. DS=2000H, EAX=3000H → 23000H**
* **iii. SS=8000H, ESP=9000H → 89000H**

**© What is the flat mode memory system? Protected mode memory addressing allows the access to the area of the memory in the 80286 micro-processor?**

Ans: **1. Flat Mode Memory System**

* **Flat memory model** treats the **entire memory as a single linear address space**.
* No segmentation; all addresses are **direct 32-bit linear addresses**.
* Used in **modern 32-bit protected mode operating systems** for simplicity.

**2. Protected Mode Memory Addressing (80286)**

* 80286 in **protected mode** can access **up to 16 MB of memory**.
* Uses **segment selectors and descriptors** (from **GDT/LDT**) to map logical addresses to **physical addresses**.
* Provides **memory protection** and **multitasking support**.

**(d) What are the differences between a register and memory location? List one special function for each data registers AX, BX, CX and DX.**

Ans: **Register vs Memory**

| **Feature** | **Register** | **Memory** |
| --- | --- | --- |
| Speed | Fast | Slow |
| Size | Small | Large |
| Access | Direct | Via address |
| Purpose | Temp storage | Long-term |

**2. Data Registers Special Functions**

* **AX** → Accumulator (ALU ops)
* **BX** → Base (memory addressing)
* **CX** → Count (loops/strings)
* **DX** → Data (I/O, multiply/divide)

**The Question No: 02**

**(a) What is wrong with MOV BL, CX instruction? List the 16-bit segment registers used with register addressing by MOV, PUSH and POP.**

Ans: **Problem with MOV BL, CX**

* **BL** is 8-bit, **CX** is 16-bit → size mismatch.
* Cannot move 16-bit register into 8-bit register directly.

**16-bit Segment Registers with MOV, PUSH, POP**

| Instruction | Allowed Segment Registers |
| --- | --- |
| MOV | CS, DS, ES, SS |
| PUSH | CS, DS, ES, SS |
| POP | DS, ES, SS |

Note: **CS cannot be directly loaded using POP**.

**(b) What is a displacement? How does it determine the memory address in a MOV DS:[2000H], AL instruction?**

Ans: **1. Displacement**

* A **displacement** is a **constant value added to a base or index** to calculate a memory address.
* It allows **flexible memory addressing** within a segment.

**2. Example: MOV DS:[2000H], AL**

* **DS** = Data Segment register (base of segment)
* **2000H** = Displacement (offset within the segment)
* **Physical Address** = DS × 16 + 2000H
* AL is stored at this **calculated physical memory location**.

**© What do the symbols [] indicate? Suppose that DS = 0200H, BX = 0300H and DI = 0400H. Determine the memory address accessed by each of the following instructions, assuming real mode operations:**

**i. MOV AL, [1234H].**

**ii. MOV EAX, [BX].**

**iii. MOV [DI], AL.**

Ans: **[] meaning:** memory indirect — the value inside [] is an address (offset) whose contents are accessed.

Given DS = 0200H (so DS×16 = 0200H × 10H = 2000H), BX = 0300H, DI = 0400H:

i. MOV AL, [1234H]

* offset = 1234H
* physical = 2000H + 1234H = 3234H.

ii. MOV EAX, [BX]

* **Invalid in real (16-bit) mode** — EAX is a 32-bit register (real mode uses 16-bit registers).
* If intent was MOV AX, [BX]: offset = BX = 0300H → physical = 2000H + 0300H = 2300H.

iii. MOV [DI], AL

* offset = DI = 0400H
* physical = 2000H + 0400H = 2400H.

**(d) What base register addresses data in the stack segment? Suppose that DS = 1300H, SS = 1400H, BP = 1500H and SI = 0100H. Determine the address accessed by each of the following instructions, assuming real mode operation:**

**i. MOV EAX, [BP+0200H].**

**ii. MOV AL, [BP+SI-0200H].**

Ans: **1. Base Register for Stack Segment**

* **BP (Base Pointer)** addresses data in the **stack segment (SS)**.
* **SP** can also be used to access the stack.

**2. Memory Address Calculation (Real Mode)**

**Physical Address Formula:**

* **SS = 1400H → SS × 16 = 14000H**

**i. MOV EAX, [BP+0200H]**

* Offset = BP + 0200H = 1500H + 0200H = 1700H
* Physical = 14000H + 1700H = **15700H**

**ii. MOV AL, [BP+SI-0200H]**

* Offset = BP + SI − 0200H = 1500H + 0100H − 0200H = 1400H
* Physical = 14000H + 1400H = **15400H**

**(e) What is the difference between an intersegment and intra-segment jump? Show which JMP instruction assembles (short, near or far) if the JMP THERE instruction is stored at memory address 10000H and the address of THERE is:**

**i. 10020H.**

**ii. 0FFFEH.**

Ans: **1. Intersegment vs Intra-segment Jump**

| **Feature** | **Intra-segment Jump** | **Intersegment Jump** |
| --- | --- | --- |
| **Definition** | Jump within the same code segment | Jump to a different code segment |
| **Addressing** | Uses **offset only** | Uses **segment:offset** |
| **Instruction Type** | Short or near JMP | Far JMP (specifies new CS:IP) |

**2. JMP THERE Example**

* **Instruction stored at 10000H**

**i. THERE = 10020H**

* Offset difference = 10020H − 10000H = 20H (within same segment)
* **Type:** Short or Near JMP (intra-segment)

**ii. THERE = 0FFFEH**

* Offset difference = 0FFFEH − 10000H = −2 (wraps back in segment)
* **Type:** Near JMP (still intra-segment, same CS)

**Far JMP** is used **only when the target is in a different code segment**.

**The Question No: 03**

**(a) Describe the purpose of the D-bits and W-bits found in some machine language instructions. If the register field (REG) of an instruction contains 010 and W = 0, what register is selected, assuming that the instruction is a 16-bit mode instruction.**

**Ans: 1. Purpose of D and W Bits**

* **D-bit (Direction bit):**
  + **Determines data transfer direction in register/memory instructions.**
  + **D = 0 → REG field is source, r/m is destination**
  + **D = 1 → REG field is destination, r/m is source**
* **W-bit (Word/Byte bit):**
  + **Determines operand size.**
  + **W = 0 → byte operation (8-bit)**
  + **W = 1 → word operation (16-bit)**

**2. Register Selection**

* **Given: REG = 010, W = 0, 16-bit mode instruction**
* **W = 0 → 8-bit registers**
* **REG 010 → DL (the 8-bit lower half of DX)**

**(b) Identify the default segment registers assigned to the following: i. SP.**

**ii. EBX.**

**iii. DI.**

**iv. SI.**

**Ans:** **Default Segment Registers and Uses**

| **Register** | **Default Segment** | **Use** |
| --- | --- | --- |
| **SP** | SS (Stack Segment) | Points to the **top of the stack** for PUSH, POP, CALL, RET. |
| **EBX** | DS (Data Segment) | Used as a **base register** for memory addressing in data access. |
| **DI** | ES (Extra Segment) | Often used as **destination index** in string operations (MOVS, STOS). |
| **SI** | DS (Data Segment) | Often used as **source index** in string operations (MOVS, LODS). |

**Extra:** Default segments allow instructions to access memory without specifying the segment explicitly, making code shorter and faster.

**© If the start of a segment is identified with .DATA, what type of memory organization is in effect? Convert an 8B07H from machine language to assembly language.**

Ans: **1. Segment Type**

* .DATA identifies the **data segment**.
* **Memory organization:** **Segmented memory** (real-mode style, each segment has a start address, e.g., DS for data).

**Extra:** Segmentation allows separate storage for **code, data, and stack**.

**2. Convert 8B07H (Machine → Assembly)**

* 8B → **MOV r16, r/m16** opcode (move 16-bit from memory/register to register)
* 07 → **ModR/M byte**
  + Mod = 00 → memory, no displacement
  + REG = 000 → **AX**
  + R/M = 111 → **[BX+SI]**

✅ **Assembly:**

MOV AX, [BX+SI]

**Extra:** ModR/M byte specifies which registers/memory locations are involved in the operation.

**(d) What directives indicate the start and end of a procedure? Explain what happens when the PUSH BX instruction executes. Make sure where BH and BL are stored. (Assume that SP = 0100H and SS = 0200H).**

**Ans**: **1. Procedure Directives**

* **Start of procedure:** PROC
* **End of procedure:** ENDP

**Extra:** These directives define a named block of code for modular programming.

**2. PUSH BX Instruction**

* **BX** is a 16-bit register (BH = high byte, BL = low byte)
* **Execution steps:**
  1. **SP is decremented by 2** → SP = 0100H − 2 = 00FEH
  2. **BX contents are stored at memory address SS × 16 + SP** → 0200H × 16 + 00FEH = 2000H + 00FEH = 20FEH
  3. **BH (high byte) is stored at 20FEH**, **BL (low byte) is stored at 20FFH**

**Extra:** Stack grows **downward**, so higher addresses hold lower-order bytes in 8086 real mode.

**(e) Write a program to (i) display a "?". (ii) read two decimal digits whose sum is less than 10. (iii) sample execution: ?27 THE SUM OF 2 AND 7 IS 9.**

**Ans:** Here’s the **main logic in x86 assembly (8086 real mode)** using DOS interrupts:

MOV AH, 02H ; Function to display character

MOV DL, '?' ; Character to display

INT 21H ; Display '?'

MOV AH, 01H ; Function to read character

INT 21H ; Read first digit

SUB AL, '0' ; Convert ASCII to number

MOV BL, AL ; Store first digit in BL

INT 21H ; Read second digit

SUB AL, '0'

MOV BH, AL ; Store second digit in BH

ADD BL, BH ; Sum the digits

CMP BL, 10 ; Check if sum < 10

JAE INVALID ; Jump if sum ≥ 10

; Display result

MOV AH, 09H

LEA DX, MSG ; "THE SUM OF X AND Y IS Z$"

INT 21H

JMP EXIT

INVALID:

MOV AH, 09H

LEA DX, ERR ; "INVALID INPUT$"

INT 21H

EXIT:

MOV AH, 4CH

INT 21H

MSG DB " THE SUM OF 2 AND 7 IS 9$"

ERR DB " INVALID INPUT$"

**The Question No: 04**

**Question (a)**

**What is wrong with ADD RCX, AX? Develop a short sequence of instructions that adds AL, BL, CL, DL, and AH and saves the sum in DH.**

**Solution:**

* Problem: RCX is a **64-bit register**, AX is **16-bit** → operand size mismatch.
* Correct approach: Use registers of **same size**.

**Sequence to add AL, BL, CL, DL, AH → DH:**

MOV AL, AL ; first value

ADD AL, BL ; AL = AL + BL

ADD AL, CL ; AL = AL + CL

ADD AL, DL ; AL = AL + DL

ADD AL, AH ; AL = AL + AH

MOV DH, AL ; save result in DH

**Question (b)**

**Explain the difference between SUB and CMP. Multiply AL by 8; divide 32142 by 4 and put quotient in AX.**

**Solution:**

* SUB dest, src → **subtracts src from dest and stores result in dest**
* CMP dest, src → **subtracts src from dest but only updates flags; does NOT change dest**

**Instructions:**  
i. Multiply AL by 8:

MOV BL, 8

MUL BL ; AL \* 8 → AX (unsigned multiplication)

ii. Divide 32142 by 4, quotient in AX:

MOV AX, 32142

MOV BL, 4

DIV BL ; AX / BL → quotient in AL, remainder in AH (if 8-bit divisor)

; For 16-bit divisor:

MOV DX, 0

MOV AX, 32142

MOV BX, 4

DIV BX ; AX / BX → quotient in AX, remainder in DX

**Question (c)**

**Explain JMP AX. Identify near/far jump. List five flags tested by conditional jumps.**

**Solution:**

* JMP AX → **jump to offset stored in AX within current code segment** → **near jump**
* **Five flags tested by conditional jumps:**
  + Zero Flag (ZF)
  + Sign Flag (SF)
  + Carry Flag (CF)
  + Overflow Flag (OF)
  + Parity Flag (PF)

**Question (d)**

**How many interrupt types? Explain near/far CALL function.**

**Solution:**

* **Intel 8086** supports **256 interrupt types (0–255)**.
* **Near CALL:** jumps to a procedure **within the same code segment** (pushes only IP on stack)
* **Far CALL:** jumps to a procedure in **different code segment** (pushes CS and IP on stack)

**Question (e)**

**If legal, give values of DX, AX, CF/OF after MUL BX. AX=0008H, BX=0003H. Write IF AX<0 THEN PUT -1 IN BX END IF.**

**Solution:**

* Multiply: AX \* BX → 0008H \* 0003H = 0018H → AX = 0018H, DX = 0, CF = 0, OF = 0

**Assembly code for decision structure:**

MOV AX, 0008H

MOV BX, 0003H

MUL BX          ; AX = AX \* BX, DX = 0

CMP AX, 0

JGE SKIP        ; jump if AX >= 0

MOV BX, -1      ; AX < 0 → put -1 in BX

SKIP:

**The Question No: 05**

(a) **i. “CPU actually works on binary digits.” Justify this statement.**  
**ii. Enlist the major evolution in computational era with its key technology.**

**Ans: i.** CPU works on binary digits (0 and 1) because all logic and arithmetic operations are performed using electronic circuits like logic gates that respond to two voltage levels — High (1) and Low (0).

ii. Major evolution in computational era:

1. 1st Generation: Vacuum tubes → Machine language
2. 2nd Generation: Transistors → Assembly language
3. 3rd Generation: Integrated Circuits → High-level languages
4. 4th Generation: Microprocessors → Personal computers
5. 5th Generation: AI and Parallel processing → Knowledge-based systems

**(b) Explain the learning outcome from this course.**

**Answer:**

Students gain:

* Understanding of **microprocessor architecture and operations**
* Ability to **write and analyze assembly programs**
* Knowledge of **memory organization, interrupts, and interfacing**
* Skills to apply **microprocessor concepts in embedded system design**

**© Give advantages and disadvantages of flags in CPU. Give the flag status after performing the following operation: ABCD × A = B6602.**

**Answer:**

**Advantages:**

* Indicate **status of arithmetic or logical operations** (e.g., carry, zero).
* Help control **program flow** using conditional jumps.

**Disadvantages:**

* Flags can be **modified unintentionally** by other instructions.
* Not all flags are relevant for every operation.

**Flag status after operation:**

* **Carry (CF):** 0
* **Zero (ZF):** 0
* **Sign (SF):** 0
* **Auxiliary Carry (AF):** 0
* **Parity (PF):** 1
* **Overflow (OF):** 0

(d) **i. Distinguish between coprocessor and peripheral.**  
**ii. Mention the features of 80287.**

**Answer:**

**i. Difference:**

| **Feature** | **Coprocessor** | **Peripheral** |
| --- | --- | --- |
| Function | Enhances CPU performance | Performs I/O or support tasks |
| Connection | Directly linked to CPU | Connected via I/O ports or buses |
| Example | 80287 math co-processor | Keyboard, printer, etc. |

**ii. Features of 80287:**

* Works with **80286 CPU** for floating-point arithmetic.
* Supports **real, protected, and virtual modes**.
* 80-bit internal registers for precision.
* Executes **floating-point, logarithmic, and trigonometric** instructions.

**The Question No: 06**

**Question 6(a)**

**Explain responsibilities of segment register in protected mode memory addressing. If DS=0105H in protected mode, which entry, table, and requested privilege level are selected?**

**Answer:**

**Responsibilities:**

* **Segment registers hold selectors that index into descriptor tables (GDT/LDT).**
* **They define the base address, limit, and access rights of memory segments.**
* **Protects memory using privilege levels (0–3) and ensures isolation.**

**For DS = 0105H:**

* **Entry selected: 0000000000010 (index = 2)**
* **Table: GDT (Global Descriptor Table)**
* **Requested Privilege Level (RPL): 1**

**Question 6(b)**

**i. Why is accumulator so called?  
ii. Enlist the differences between 8086 and 8088 microprocessors.**

**Answer:**

**i. It’s called accumulator because it accumulates the results of arithmetic and logic operations.**

**ii. Differences:**

| **Feature** | **8086** | **8088** |
| --- | --- | --- |
| **Data Bus** | **16-bit** | **8-bit** |
| **Instruction Queue** | **6 bytes** | **4 bytes** |
| **Speed** | **Faster** | **Slower** |
| **Performance** | **Higher** | **Slightly lower** |

**Question 6(c)**

**i. Explain handshaking with respect to CPU.  
ii. Describe addressing modes of DSP56300. Explain trigger of DMA.**

**Answer:**

i. Handshaking is a signal-based synchronization method between CPU and I/O devices to ensure data transfer occurs at the correct time using READY, ACK, and STROBE signals.

ii. **DSP56300 Addressing Modes:**

* Immediate
* Direct
* Indirect
* Register Direct
* Indexed

**DMA Trigger:**

* DMA (Direct Memory Access) is triggered by a hardware request to transfer data directly between I/O and memory without CPU involvement.

**Question 6(d)**

**i. How does 82C55 can be programmed?**  
**ii. Distinguish among various modes of operation of 82C55.**

**Answer:**

**i.** 82C55 is programmed by writing a **control word** into its **control register**, which sets mode (0,1,2) and port directions (input/output).

**ii. Modes of Operation:**

| **Mode** | **Description** |
| --- | --- |
| Mode 0 | Simple I/O without handshaking |
| Mode 1 | I/O with handshaking for synchronization |
| Mode 2 | Bidirectional data transfer using strobes and handshakes |