Solutions to Selected Questions from the Microprocessor and Microcontroller Exam

UNIVERSITY OF BARISHAL

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

3rd Year 1st Semester Final Examination-2023

Session: 2019-20

Course Title: Microprocessor and Microcontroller, Course Code: CSE-3101

Time: 3 hours **Marks:** 60

Answer any five Questions from the followings.

1.	a)	Draw the internal architecture of 8086. Describe the functionalities of Execution Unit and Bus Interface Unit.	[5]
	b)	Write down the basic functionalities of general purpose registers of 8086.	[4]
	c)	Write down the major features of various processor generations.	[3]
2.	a)	Why need flag registers? Explicate the main objectives of all flag registers using the 16 bit representation.	[4]
	b)	Why do we need various addressing modes of 8086? Describe prominent addressing modes with suitable diagram and assembly code examples.	[2+6]
3.	a)	What is physical memory? A memory location has physical address (80FD2h). In what segment it have offset BFD2h?	[4]
	b)	Discuss how an array can be declared in 8086 processor with DUP.	[4]
	c)	Explain the concept of cache memory in microprocessors. What are its advantages, and how is it implemented in the system?	[4]
4.	a)	What do you mean by odd address bank and even address bank? How to address them using available prominent approach.	[5]
	b)	Explain how LOOP instruction works in assembly language? Write an assembly language program for For Loop to show the "CSE Department" as reverse string.	[5]
	c)	How instruction affect the Flags: SUB AX, BX [AX=90h, BX=30h] ADD AL, BL [AL=30h, BL=70h]	[2]
5.	a)	Draw the architecture of 8255. Describe the operating modes of 8255.	[6]
	b)	What do you mean by programmable peripheral interface (PPI)? Mention the role of 8255 PPI.	[3]
	c)	Write assembly code statement for each of the high level language assignments statements. i) A=B*C-5/D ii) B= (A-C)*D/7	[3]
6.	a)	What is recursion? Write an assembly language program for calculating the factorial of 5.	[4]
	b)	Write short notes on: i) Logical Shift ii) Arithmetic Shift ii)ROL iv)ROR	[8]
7.	a)	What is Microcontroller? What are the advantages and disadvantages of Microcontroller?	[4]
	b)	What is the difference between timer and counter of microcontroller? List the factors to be considered for selection of microcontroller for particular application.	[4]
	c)	Differentiate between Memory mapped IO and IO mapped IO with reference to 8085 microprocessors.	[4]
8.	a)	Short notes on various types of jumping instruction : i)Signed Conditional Jumps ii) Unsigned Conditional Jumps iii) Single Flag Jumps	[6]
	b)	Write an assembly language program for comparing two string and concatenating two string	[6]

Question 1

a) Internal Architecture of 8086 and Functionalities of EU and BIU

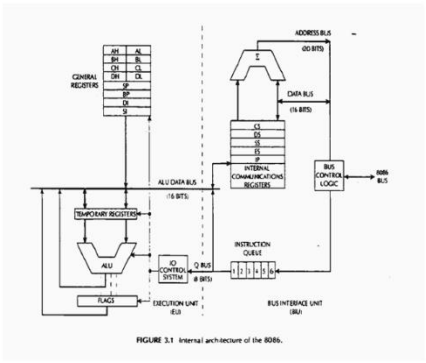


FIGURE 3.1 Internal architecture of the 8086.

The 8086 microprocessor has two main functional units:

1. Bus Interface Unit (BIU):

   - Handles all data and address transfers on the buses
   - Contains the instruction queue (6-byte FIFO)
   - Contains segment registers (CS, DS, SS, ES)
   - Generates the 20-bit physical address
   - Fetches instructions and stores them in the instruction queue

2. Execution Unit (EU):

   - Contains the ALU for arithmetic and logic operations
   - Contains general purpose registers (AX, BX, CX, DX)
   - Contains pointer and index registers (SP, BP, SI, DI)
   - Contains the flag register
   - Decodes and executes instructions from the instruction queue

Or

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

o Function**:** Handles communication with memory and I/O through the system bus.

o Components**:**

▪ Instruction Queue**:** Stores prefetched instructions (up to 6 bytes) for pipelining.

▪ Segment Registers**:** Includes CS (Code Segment), DS (Data Segment), SS (Stack Segment), and ES (Extra Segment) to calculate physical addresses.

▪ **Instruction Pointer (IP):** Points to the address of the next instruction.

▪ **Address Calculation:** Combines segment and offset values to compute a 20-bit physical memory address.

2. **Execution Unit (EU):**

o **Function:** Executes instructions fetched by the BIU.

o **Components:**

▪ **Arithmetic Logic Unit (ALU):** Performs arithmetic and logical operations.

▪ **General Purpose Registers:** AX, BX, CX, DX (used for temporary data storage and calculations).

▪ **Control Flags:** Direct the operations of the EU (e.g., zero flag, carry flag).

▪ **Instruction Decoder:** Decodes instructions for execution.

b) Basic Functionalities of General Purpose Registers

The **8086 microprocessor** has four general-purpose registers, each 16 bits wide. These registers can also
be accessed as 8-bit registers (higher and lower parts).

1.AX (Accumulator):

Functionality:

o Used for arithmetic operations, logical operations, and data transfer.

o Acts as the default operand for instructions like MUL, DIV, ADD, and SUB.

o **AL** (lower 8 bits) and **AH** (higher 8 bits) can be accessed independently.

2.BX (Base Register):

• **Functionality:**

o Primarily used as a base pointer for addressing memory locations.

o Holds the base address during indirect addressing modes.

o Used to store data for calculations or transfers.

3. CX (Count Register):

• **Functionality:**

o Acts as a counter in loop operations (e.g., LOOP instruction).

o Used as the default counter for shift and rotate instructions.

o **CL** is used in bitwise shift/rotate instructions for specifying the number of shifts.

4. DX (Data Register):

• **Functionality:**

o Holds the data for multiplication and division operations.

o Often used in I/O operations to store the address of the port.

o Functions as an extension to AX in 32-bit operations (e.g., the high-order word in MUL and DIV).

c) Major Features of Processor Generations

1. 1st Generation (1971-1973):

- 4-bit and 8-bit processors

- PMOS technology

- Basic instruction set (e.g., Intel 4004)

2. 2nd Generation (1974-1978):

- 8-bit processors

- NMOS technology

- Enhanced instruction sets (e.g., Intel 8080)

3. 3rd Generation (1979-1980):

- 16-bit processors

- Introduced segmentation (e.g., Intel 8086)

- Pipelining concepts

4. 4th Generation (1981-1995):

- 32-bit processors

- On-chip cache memory

- Superscalar architecture (e.g., Intel 80386, 80486)

Question 2

a)

The flag register in the 8086 microprocessor is a special-purpose register that stores information about
the status of the processor and the results of arithmetic or logical operations. It is crucial for decisionmaking during program execution.

Main Objectives of Flag Registers:

1. **Indicate Status of Operations:**
   o The flag register reflects the outcome of operations, such as whether a result is zero, whether there was an overflow, or whether a carry occurred.
2. **Control Program Flow:**
   o Flags are used in conditional jump, loop, and call instructions to decide the next course
   of action based on the results of comparisons or calculations.
3. **Facilitate Debugging:**
   o Flags help in understanding and debugging program logic by revealing the processor's state.
4. **Support Interrupt Handling:**
   o Some flags, like the interrupt flag (IF), control the enabling and disabling of interrupts.

Structure of the 16-bit Flag Register in 8086:
The flag register contains 16 bits, divided into two categories:

1. Status Flags:

- CF (Carry Flag): Set when carry/borrow occurs
- PF (Parity Flag): Set when result has even parity
- AF (Auxiliary Flag): Set when carry between nibbles
- ZF (Zero Flag): Set when result is zero
- SF (Sign Flag): Set when result is negative
- OF (Overflow Flag): Set when signed overflow occurs

2. Control Flags:

- TF (Trap Flag): Used for single-step debugging
- IF (Interrupt Flag): Enables/disables interrupts
- DF (Direction Flag): Controls string operation direction

b) Addressing Modes in 8086

Addressing modes in the **8086 microprocessor** specify how the operand (data) for an instruction is
accessed. These modes enhance flexibility in accessing data stored in registers, memory, or provided as
immediate values.

Prominent Addressing Modes in 8086:

1. Immediate Addressing Mode:

- The operand is directly specified in the instruction.
- Example: MOV AX, 1234H
  - **Explanation:** The value 1234H is directly moved into register AX.

2. Register Addressing Mode:

- The operand is stored in a register, and the instruction specifies the register.
- Example: MOV BX, AX
  - **Explanation:** The content of register AX is moved to register BX.

3. Direct Addressing Mode:

- The effective address (memory location) is directly specified in the instruction.
- Example: MOV AX, [5000H]
  - **Explanation:** The data at memory location 5000H is moved to register AX.

4. Indirect Addressing Mode:

- The effective address is held in a register (e.g., BX, SI, DI, BP).
- Example: MOV AX, [BX]
  - **Explanation:** The content of the memory location pointed to by BX is moved to AX.

5. Based Addressing Mode:

- Combines a base register (BX or BP) with a displacement value to calculate the effective address.
- Example: MOV AX, [BX + 10H]
  - **Explanation:** Adds 10H to the value in BX to calculate the effective address and then moves the data from that address to AX.

6. Indexed Addressing Mode:

- Combines an index register (SI or DI) with a displacement value to calculate the effective address.
- Example: MOV AX, [SI + 20H]
  - **Explanation:** Adds 20H to the value in SI and moves the data at the resulting address to AX.

7. Based-Indexed Addressing Mode:

- Combines a base register and an index register to calculate the effective address.
- Example: MOV AX, [BX + SI]
  - **Explanation:** Adds the values of BX and SI to calculate the effective address, then moves the data to AX.

8. Relative Addressing Mode:

- Adds an offset to the current instruction pointer (IP) to calculate the address.
- Example: JMP LABEL
  - **Explanation:** Adds the offset of LABEL to IP to jump to the desired instruction.

Diagram for Addressing Modes:

Here's an example of how the memory and registers interact for different addressing modes:

- Registers**:** BX, SI, DI
- Memory**:** Segments + Offset address combination
- Effective Address = Segment Base + Displacement/Offset

Question 3

a) Physical Memory and Segment Calculation

Physical memory refers to the actual memory hardware that stores data and instructions. The physical address in 8086 is calculated as:

Physical Address = (Segment Register × 16) + Offset

Given physical address = 80FD2h and offset = BFD2h:

Segment × 16 + BFD2h = 80FD2h

Segment × 16 = 80FD2h - BFD2h = 75000h

Segment = 75000h / 10h = 7500h

b) Array Declaration with DUP

In 8086 assembly, arrays can be declared using the DUP directive:

```assembly
array1 DB 10 DUP(0)      ; Array of 10 bytes initialized to 0
array2 DW 20 DUP(?)      ; Array of 20 words uninitialized
array3 DB 5 DUP('A')     ; Array of 5 bytes with 'A'
```

c) Cache Memory

Cache memory is a small, fast memory located between the CPU and main memory that stores frequently accessed data and instructions. Its main purpose is to **store frequently used data and instructions** so the CPU can access them quickly instead of fetching from slower main memory (RAM). Because accessing data from cache is much faster than from RAM, cache improves the overall speed and performance of the processor.

Advantages:

- Reduces average memory access time
- Improves system performance
- Reduces bus traffic

Implementation of Cache Memory in the System

- Cache is implemented as a small block of high-speed SRAM (Static RAM).
- It is placed **between the CPU and the main memory**.
- Cache is organized in levels:
    - **L1 Cache:** Smallest and fastest, built directly into the CPU core.
    - **L2 Cache:** Larger than L1 but slower; can be inside the CPU or on a separate chip.

    - **L3 Cache:** Even larger and slower, shared among multiple CPU cores in modern processors.
- The system uses **cache controllers** to manage which data to keep in the cache and when to update it.
- Cache uses different mapping techniques (direct, associative, set-associative) to store and locate data efficiently.

Question 4

a) Odd and Even Address Banks

The 8086 has a 16-bit data bus divided into two 8-bit banks:

- Even address bank: Connected to D0-D7, stores even-addressed bytes

- Odd address bank: Connected to D8-D15, stores odd-addressed bytes

Addressing approach:

- The BHE (Bus High Enable) signal and A0 address line are used:

  - BHE=0, A0=0: Access both banks (16-bit word)

  - BHE=0, A0=1: Access odd bank only (upper byte)

  - BHE=1, A0=0: Access even bank only (lower byte)

  - BHE=1, A0=1: No access

b) LOOP Instruction and Reverse String Program

The LOOP instruction decrements CX and jumps to the label if CX ≠ 0.

```assembly

.MODEL SMALL
```

```
.STACK 100H
.DATA
    msg DB 'CSE Department$'
    len EQU $-msg-1
.CODE
MAIN PROC
    MOV AX, @DATA
    MOV DS, AX

    ; Print original string
    MOV AH, 09H
    LEA DX, msg
    INT 21H

    ; Print new line
    MOV AH, 02H
    MOV DL, 0DH
    INT 21H
    MOV DL, 0AH
    INT 21H

    ; Reverse string
    MOV SI, OFFSET msg
    MOV DI, SI
    ADD DI, len-1
    MOV CX, len/2
```

```
REVERSE_LOOP:
    MOV AL, [SI]
    MOV BL, [DI]
    MOV [DI], AL
    MOV [SI], BL
    INC SI
    DEC DI
    LOOP REVERSE_LOOP

    ; Print reversed string
    MOV AH, 09H
    LEA DX, msg
    INT 21H

    MOV AH, 4CH
    INT 21H
MAIN ENDP
END MAIN
```

c) Flag Effects

1. SUB AX, BX [AX=90h, BX=30h]
   - Result = 60h (positive)
   - CF = 0 (no borrow)

- ZF = 0 (non-zero)

- SF = 0 (positive)

- OF = 0 (no overflow)

2. ADD AL, BL [AL=30h, BL=70h]

- Result = A0h (negative in signed interpretation)

- CF = 0 (no carry)

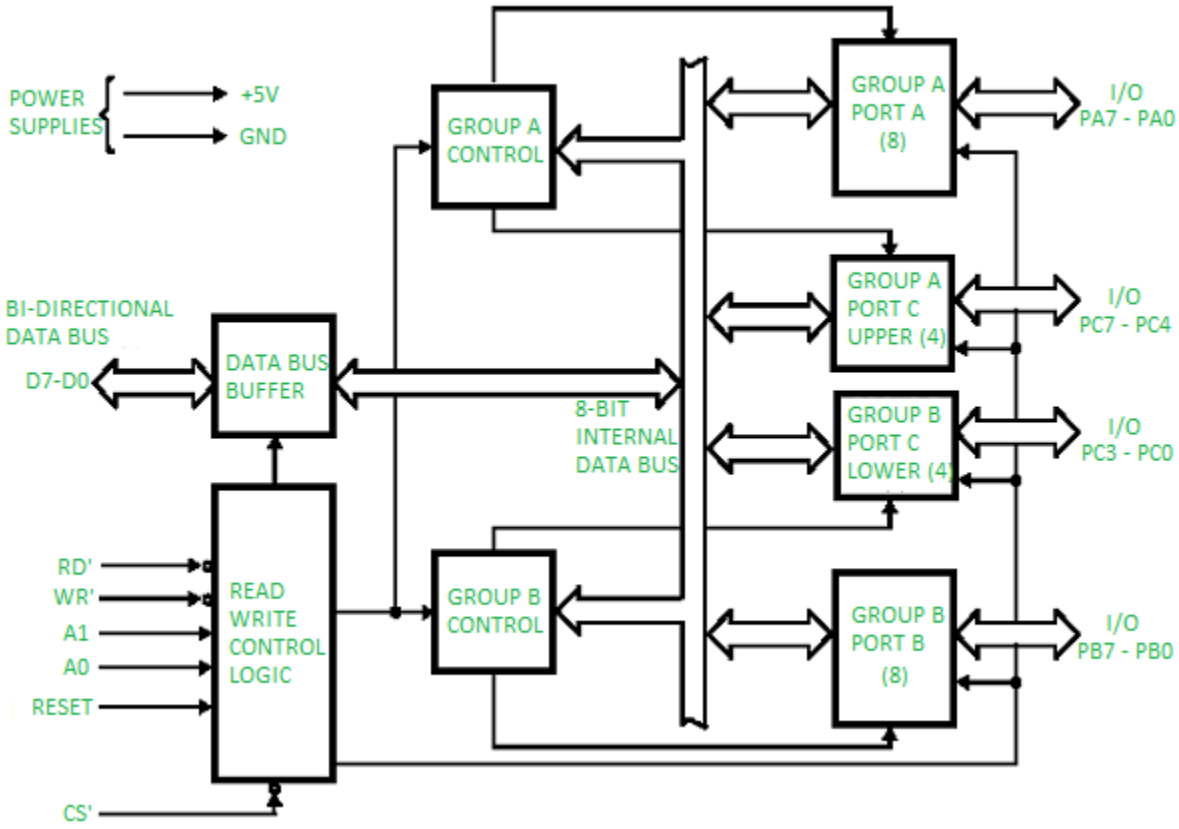
- ZF = 0 (non-zero)

- SF = 1 (negative)

- OF = 1 (signed overflow: 30h + 70h = A0h which is -96 in signed 8-bit)

Question 5

a) 8255 Architecture and Operating Modes



The 8255 Programmable Peripheral Interface has:

- Three 8-bit ports (PA, PB, PC)

- Control logic for mode selection

- Two groups (Group A: PA and upper PC, Group B: PB and lower PC)

Operating Modes:

1. Mode 0 (Basic I/O): Ports act as simple input or output

2. Mode 1 (Strobed I/O): Handshake signals for data transfer

3. Mode 2 (Bidirectional Bus): Only PA can be used for bidirectional data transfer

b) Programmable Peripheral Interface (PPI)

Programmable Peripheral Interface (PPI):

A **PPI** is a device used to connect **microprocessors with input/output devices** (like keyboard, printer, LED, switches, etc.).
It helps in **data transfer** between the microprocessor and these devices.

The most common PPI is the **Intel 8255**.

Role of 8255 PPI:

1. **Interface Between CPU and Peripheral Devices:**
   8255 connects the microprocessor to devices like LCD, keyboard, printer, etc.
2. **Three 8-bit Ports (Port A, Port B, Port C):**
   These ports can be programmed as **input or output** ports.
3. **Modes of Operation:**
   8255 can work in different modes (Mode 0, Mode 1, Mode 2) depending on the application (simple I/O, handshake I/O, etc.).
4. **Handles Multiple Devices:**
   Allows communication with multiple devices at the same time.
5. **Reduces CPU Load:**
   By handling I/O tasks, 8255 reduces the burden on the CPU.

c) Assembly Code for Assignment Statements

i) A = B * C - 5 / D

```assembly
MOV AX, B     ; Load B
IMUL C        ; B*C (result in DX:AX)
MOV BX, AX    ; Save lower part of result
MOV AX, 5     ; Load 5
CWD           ; Convert to double word
IDIV D        ; 5/D (quotient in AX)
SUB BX, AX    ; (B*C) - (5/D)
MOV A, BX     ; Store result in A
```

ii) B = (A - C) * D / 7

```assembly
MOV AX, A     ; Load A
SUB AX, C     ; A - C
IMUL D        ; (A-C)*D (result in DX:AX)
MOV BX, 7     ; Load 7
IDIV BX       ; (A-C)*D / 7 (quotient in AX)
MOV B, AX     ; Store result in B
```

Question 6

a) Recursion and Factorial Program

Recursion is a programming technique where a function calls itself to solve a problem by breaking it down into smaller subproblems.

```assembly
.MODEL SMALL
.STACK 100H
.DATA
    result DW ?
.CODE
MAIN PROC
    MOV AX, 5       ; Calculate factorial of 5
    CALL FACTORIAL
    MOV result, AX   ; Store result
    MOV AH, 4CH
    INT 21H
MAIN ENDP

FACTORIAL PROC
    CMP AX, 1       ; Base case: if n <= 1
    JLE BASE_CASE
    PUSH AX         ; Save current n
    DEC AX          ; n-1
    CALL FACTORIAL   ; Recursive call
    POP CX          ; Retrieve saved n
    MUL CX          ; AX = AX * CX (n * factorial(n-1))
    RET

BASE_CASE:
```

```
    MOV AX, 1       ; Return 1 for base case
    RET
FACTORIAL ENDP
END MAIN
```

b) Short Notes on Shift Operations

i) Logical Shift

- In Logical Shift, the bits in a binary number are moved left or right.
- **For left shift:** All bits move left, and 0 is added at the rightmost position.
- **For right shift:** All bits move right, and 0 is added at the leftmost position.
- Example:
  If number = 1011 0001
  Logical Right Shift by 1 = 0101 1000
- **Use:** Usually used for unsigned numbers.

ii) Arithmetic Shift

- Like logical shift, but special for signed numbers.
- **For left shift:** Same as logical shift (shift left, add 0 at right).
- **For right shift:** Keeps the **sign bit (MSB)** the same to keep the number's sign correct.
- Example:
  If number = 1101 0001 (Negative in signed)
  Arithmetic Right Shift by 1 = 1110 1000
- **Use:** To divide or multiply signed numbers by powers of 2.

iii) ROL (Rotate Left)

- All bits rotate left.
- The leftmost bit goes to the **rightmost place**.

- Example:
  If number = 1001 0110
  After ROL by 1 = 0010 1101
- **No bits are lost.**

iv) ROR (Rotate Right)

- All bits rotate right.
- The rightmost bit goes to the **leftmost place**.
- Example:
  If number = 1001 0110
  After ROR by 1 = 0100 1011
- **No bits are lost.**

Question 7

a) Microcontroller: Advantages and Disadvantages

A microcontroller (MCU) is a small computer on a single integrated circuit that is designed to control specific tasks within electronic systems. It combines the functions of a central processing unit (CPU), memory, and input/output interfaces, all on a single chip.

Microcontrollers are widely used in embedded systems, such as home appliances, automotive systems, medical devices, and industrial control systems. They are also used in consumer electronics products, such as gaming systems, digital cameras, and audio players.

Advantages:

- All components in single chip (CPU, RAM, ROM, I/O)

- Low power consumption

- Cost effective for embedded systems

- Small size

- Easy to interface with peripherals

Disadvantages:

- Limited processing power compared to microprocessors

- Fixed amount of on-chip memory

- Less flexible for upgrades

- Limited number of I/O ports

Or

Advantages of the Microcontroller

- **Time required:** Low time required for performing operation.
- **Easy usage** : It is easy to use, troubleshooting and system maintenance is straightforward.
- **Multitasking:** At an equivalent time, many tasks are often performed therefore the human effect are often saved.
- **Processor chip size:** Processor chip is extremely small and adaptability occurs.
- **Overall cost:** Cost and size of the system is less.
- **Interface interaction:** Microcontroller is straightforward to interface additional RAM, ROM, and I/O port.
- **Functioning:** Once microcontroller is programmed then they can't be reprogrammed.
- **Small size:** Microcontrollers are small and compact, which makes them well-suited for use in small electronic devices and systems.
- **Low power consumption:** Microcontrollers are designed to be energy-efficient, which can extend the battery life of electronic devices and systems.
- **Cost-effective:** Microcontrollers are generally less expensive than other types of computer chips, which can make them a cost-effective choice for manufacturers.

- **Real-time processing:** Microcontrollers are designed to perform real-time processing, which is important for devices that require rapid response times, such as in automotive and aerospace applications.

Disadvantages of the Microcontroller

- It is generally utilized in micro equipment.
- **Complex Structure:** It has a complex structure.
- Microcontroller cannot interface a better power device directly.
- **Execution time:** Number of executions is limited.
- As every Microcontrollers does not have analog I/O so there are issues related to microcontroller.
- Microcontrollers are composed of complementary metal-oxide-semiconductor (CMOS) and can be damaged by a static charge.
- **Limited processing power:** Microcontrollers are generally less powerful than other types of computer chips, which can limit their ability to handle more complex tasks.
- **Limited memory:** Microcontrollers typically have limited memory capacity, which can limit the size and complexity of programs that can be run on them.
- **Limited connectivity: Mi**crocontrollers may not have the ability to connect to external networks or devices, which can limit their functionality.
- **Limited software support:** Microcontrollers may have limited software support compared to other types of computer chips, which can make programming and development more challenging.

b) Timer vs Counter and Microcontroller Selection Factors

Point	**Timer**	**Counter**
Definition	A device that counts based on internal clock pulses.	A device that counts external events or signals.
Source of Input	Internal system clock.	External signals (like pulses from a switch, sensor, or encoder).

Point	Timer	Counter
Usage / Application	Used for time delay generation, real-time clocks, baud rate generation, PWM generation.	Used for event counting, frequency calculation, number of revolutions, etc.
Example Instruction	Timer interrupt (to measure time gaps).	Counting external pulses (for example, from a wheel sensor).
Mode Control	Usually controlled by setting timer mode in control registers.	Controlled by setting counter mode in control registers.
Increment Trigger	Automatically increments on every internal clock cycle.	Increments when an external edge (rising/falling) is detected.
Effect of CPU Clock	Depends on CPU/system clock.	Independent of CPU clock, depends on external events.
Example Usage in 8051	**Timer Mode:** THx, TLx used to produce precise delays.	**Counter Mode:** THx, TLx used to count external events on T0/T1 pins.

Selection Factors:

1. Processing power requirements
2. Memory requirements (RAM, ROM)
3. Number and type of I/O ports needed
4. Power consumption constraints
5. Required peripherals (ADC, PWM, etc.)
6. Cost constraints
7. Development tools availability
8. Operating voltage range

c) Memory Mapped I/O vs I/O Mapped I/O

Differentiate between I/O mapped I/O and Memory mapped I/O of 8086.

Sr. No.	I/O Mapped I/O	Memory Mapped I/O
1.	I/O device is treated as an I/O device and hence given an I/O address.	I/O device is treated like a memory device and hence given a memory address.
2.	I/O device has an 8 or 16 bit I/O address.	I/O device has a 20 bit Memory address.
3.	I/O device is given IOR# and IOW# control signals	I/O device is given MEMR# and MEMW# control signals
4.	Decoding is easier due to lesser address lines	Decoding is more complex due to more address lines
5.	Decoding is cheaper	Decoding is more expensive
6.	Works faster due to less delays	More gates add more delays hence slower
7.	Allows max 2^16 = 65536 I/O devices	Allows many more I/O devices as I/O addresses are now 20 bits.
8.	I/O devices can only be accessed by IN and OUT instructions.	I/O devices can now be accessed using any memory instruction.
9.	ONLY AL/ AH/ AX registers can be used to transfer data with the I/O device	Any register can be used to transfer data with the I/O device

Question 8

a) Types of Jumping Instructions

1. Signed Conditional Jumps:

   - JG/JNLE: Jump if greater/not less or equal (SF=OF and ZF=0)

   - JGE/JNL: Jump if greater or equal/not less (SF=OF)

   - JL/JNGE: Jump if less/not greater or equal (SF≠OF)

   - JLE/JNG: Jump if less or equal/not greater (SF≠OF or ZF=1)

2. Unsigned Conditional Jumps:

   - JA/JNBE: Jump if above/not below or equal (CF=0 and ZF=0)

   - JAE/JNB: Jump if above or equal/not below (CF=0)

   - JB/JNAE: Jump if below/not above or equal (CF=1)

   - JBE/JNA: Jump if below or equal/not above (CF=1 or ZF=1)

3. Single Flag Jumps:

   - JC: Jump if carry (CF=1)

   - JNC: Jump if no carry (CF=0)

   - JZ/JE: Jump if zero/equal (ZF=1)

   - JNZ/JNE: Jump if not zero/not equal (ZF=0)

   - JS: Jump if sign (SF=1)

   - JNS: Jump if no sign (SF=0)

   - JO: Jump if overflow (OF=1)

   - JNO: Jump if no overflow (OF=0)

b) String Comparison and Concatenation Program

```assembly
.MODEL SMALL

.STACK 100H

.DATA

    str1 DB 'Hello$'
```

```
    str2 DB 'World$'
    result DB 20 DUP('$')
.CODE
MAIN PROC
    MOV AX, @DATA
    MOV DS, AX
    MOV ES, AX      ; For string operations

    ; Compare strings
    LEA SI, str1
    LEA DI, str2
    MOV CX, 5       ; Length to compare
    REPE CMPSB      ; Compare while equal
    JNE NOT_EQUAL

    ; Strings are equal
    ; (Code for equal case)
    JMP CONCATENATE

NOT_EQUAL:
    ; Strings are not equal
    ; (Code for not equal case)

CONCATENATE:
    ; Copy str1 to result
    LEA SI, str1
```

```
    LEA DI, result
    MOV CX, 5       ; Length of str1
    REP MOVSB

    ; Copy str2 to result after str1
    LEA SI, str2
    MOV CX, 5       ; Length of str2
    REP MOVSB

    ; Print concatenated string
    MOV AH, 09H
    LEA DX, result
    INT 21H

    MOV AH, 4CH
    INT 21H
MAIN ENDP
END MAIN
```