Microprocessor and Embedded System Ques. Bank Solution

- Aditya Dhiman

Unit 1

Q1. How many data lines and address lines are available in 8086?

- **Answer:**
  The 8086 microprocessor has **16 data lines** and **20 address lines**.

Q2. Define instruction in your own words.

- **Answer:**
  An **instruction** is a command given to a microprocessor to perform a specific operation, such as data transfer, arithmetic computation, or control actions.

Q3. Define loader and assembler.

- **Answer:**
  - **Loader:** A system program that loads executable code into memory for execution.
  - **Assembler:** A software that converts assembly language code into machine code.

Q4. Define linker and loader.

- **Answer:**
  - **Linker:** Combines object code from different modules into a single executable program.
  - **Loader:** Places the executable program into memory for execution.

Q5. Describe the function of the segment registers in the 8086 microprocessor.

- **Answer:**
  Segment registers in the 8086 microprocessor hold the base addresses of segments (code, data, stack, extra) to access memory efficiently within the 1MB address space.

Q6. What is data and address bus size in 8086?

- **Answer:**
  The 8086 has a **16-bit data bus** and a **20-bit address bus**.

Q7. Explain the function of assembler directives.

- **Answer:**
  Assembler directives are instructions for the assembler, not executed by the processor, that control the assembly process, like defining constants or reserving memory.

Q8. Give any four pin definitions for the minimum mode.

- **Answer:**
  1. **M/IO:** Distinguishes memory or I/O operation.
  2. **RD:** Read control signal.
  3. **WR:** Write control signal.
  4. **ALE:** Address Latch Enable for multiplexed address/data bus.

Q9. What are general data registers?

- **Answer:**
  General data registers in 8086 are **AX**, **BX**, **CX**, and **DX**, used for performing arithmetic, logical operations, and data storage.

Q10. What are the advantages of segmented memory?

- **Answer:**
  1. **Efficient memory management:** Divides memory into manageable segments.
  2. **Larger memory access:** Allows 1MB of memory with only 16-bit registers.
  3. **Modularity:** Separates code, data, and stack segments.

Q11. List out the three machine control flags and explain briefly.

- **Answer:**

    1. **Trap Flag (TF):** Enables single-step execution.
    2. **Interrupt Flag (IF):** Controls the enable/disable of interrupts.
    3. **Direction Flag (DF):** Controls string operations direction (increment/decrement).

Q12. What are the uses of AD15 – AD0 lines?

- **Answer:**
  AD15–AD0 lines are **multiplexed** and used for carrying the lower 16 bits of the address during the first clock cycle and data during subsequent cycles.

Q13. What are pointers and index registers?

- **Answer:**
    - **Pointers:** Hold memory addresses (e.g., **SP**, **BP**).
    - **Index Registers:** Used for indexed addressing (e.g., **SI**, **DI**).

Q14. What is the purpose of the BIU (Bus Interface Unit) in the 8086?

- **Answer:**
  The **Bus Interface Unit (BIU)** handles communication with memory and I/O by fetching instructions, reading/writing data, and calculating addresses.

Q15. List out various memory types in a computer.

- **Answer:**

    1. RAM (Random Access Memory)
    2. ROM (Read-Only Memory)
    3. Cache Memory
    4. Flash Memory
    5. Secondary Storage (e.g., HDD, SSD)

Q16. What are the functional parts of the 8086 CPU?

- **Answer:**
  1. **Bus Interface Unit (BIU)**
  2. **Execution Unit (EU)**

Q17. Explain the function of assembler directives.

- **Answer:**
  Assembler directives guide the assembler in handling data, organizing code, and controlling memory allocation without being part of the executable code.

Q18. The first microprocessor was _____.

- **Answer:**
  Intel 4004 (released in 1971).

Q19. The 8086 has (a) 16 bit data bus and 20 bit address bus (b) 8 bit data bus and 20 bit address bus (c) 16 bit data bus and 16 bit address bus (d) 8 bit data bus

- **Answer:**
  (a) 16 bit data bus and 20 bit address bus

Q20. Write the 2's complement of 1011011.

- **Answer:**
  The 2's complement of **1011011** is **0100101**.

Q21. Write the size of physical memory and virtual memory of 8086 microprocessor.

- **Physical Memory Size:**
  The 8086 microprocessor has a **20-bit address bus**, allowing it to address up to **2^20 = 1,048,576** memory locations. This results in a **physical memory size of 1 MB** (megabyte). The memory is accessed through a segmented memory model, which allows addressing different areas of memory.

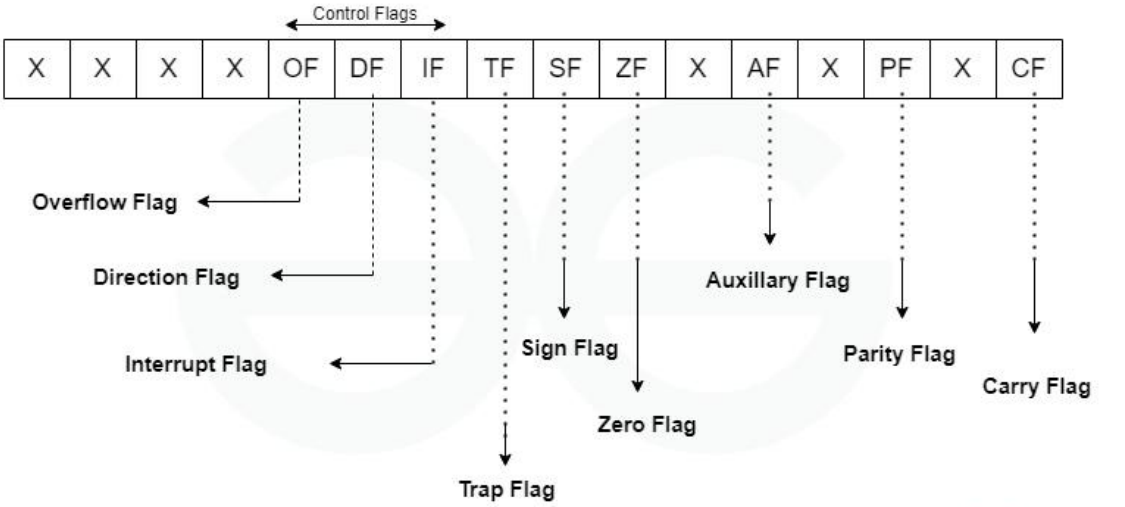
- **Virtual Memory Size:**
  The 8086 uses **segmented addressing**. Each segment register can address **64 KB**, and there are **16 segments** available at a time. Therefore, the total virtual memory size is theoretically **1 MB**, as physical and virtual memory coincide in 8086. However, segmentation can be used to access different parts of memory by changing segment registers.

Q22. Differentiate between INTEL 80386 and 8088.

Feature	Intel 80386	Intel 8088
Data Bus Width	32-bit data bus (can handle more data per clock cycle).	8-bit data bus (compatible with older systems, slower data transfer).
Address Bus Width	32-bit address bus, capable of addressing **4 GB** of memory.	20-bit address bus, capable of addressing **1 MB** of memory.
Memory Management	Supports **virtual memory** through paging and segmentation.	Does not support advanced memory management techniques.
Clock Speed	Operates at higher clock speeds (up to **33 MHz**).	Operates at lower clock speeds, typically around **5-10 MHz**.
Architecture	Fully 32-bit processor with multitasking capabilities and advanced protection modes.	16-bit internal architecture but with an 8-bit external data bus, mainly used in simpler applications.
Applications	Used in more advanced systems like servers, workstations, and personal computers.	Used in simpler systems such as early PCs and embedded applications.

Q23. Draw the flag register format of 8086 microprocessor.

The **8086 flag register** is a 16-bit register divided into **status flags** and **control flags**. Below is the format:



- **OF (Overflow Flag):** Indicates overflow in signed arithmetic operations.
- **DF (Direction Flag):** Controls string operations; if set, processes strings from high to low.
- **IF (Interrupt Flag):** If set, the processor responds to maskable interrupts.
- **TF (Trap Flag):** If set, the processor operates in single-step mode.
- **SF (Sign Flag):** Indicates if the result of the last operation was negative.
- **ZF (Zero Flag):** Set if the result of an arithmetic operation is zero.
- **AF (Auxiliary Carry Flag):** Used in binary-coded decimal (BCD) arithmetic.
- **PF (Parity Flag):** Indicates if the number of set bits in the result is even.
- **CF (Carry Flag):** Set if an arithmetic operation generates a carry or borrow.

Q24. Explain the functions of i. HLDA ii. RQ/GT0 iii. DEN iv. ALE

1. **HLDA (Hold Acknowledge):**
   This signal is asserted by the 8086 microprocessor to indicate that it has received a **HOLD** request from an external device and is acknowledging that it will relinquish control of the buses (address, data, and control) after the current bus cycle.

2. **RQ/GT0 (Request/Grant):**
   This is a **bus arbitration** signal used in **maximum mode**. It is used to request the bus from the processor and receive a grant for the bus. **RQ** is asserted by a device requesting control, and **GT** is asserted by the processor granting the request.

3. **DEN (Data Enable):**
   This signal is used to enable external data transceivers when the processor is sending or receiving data. **DEN** is active low and is asserted by the processor to indicate that data is available on the data bus.

4. **ALE (Address Latch Enable):**
   ALE is used to latch the low byte of the address onto external address latches during the first clock cycle of a bus operation. It is activated at the beginning of a bus cycle to distinguish the lower 16 bits of the address from data.

Q25. List the advantages of using segment registers in 8086.

1. **Efficient Memory Management:**
   Segmentation allows the 8086 to efficiently manage **1 MB** of memory using 16-bit segment registers. It can address a larger memory space while using only 16-bit addressing internally.

2. **Modularity:**
   The use of **segment registers** helps modularize memory, making it easier to manage code, data, and stacks in separate segments. This improves the overall structure of programs.

3. **Faster Access:**
   By using **segment**

addressing, memory accesses are faster as the processor only needs to change the offset within the segment to access nearby data.

4. **Ease of Relocation:**
   Segmented memory simplifies the relocation of code and data in memory, allowing programs to be loaded anywhere in memory without modification.

5. **Memory Protection:**
   Although the 8086 does not implement hardware protection, segmentation lays the groundwork for protecting certain memory areas in advanced processors, where specific segments can be marked as read-only or executable.

Q26. Explain pipelining. What happens when a high is applied to the RESET pin?

- **Pipelining:**
  Pipelining in the 8086 microprocessor is a technique that improves execution speed by overlapping the fetch, decode, and execute stages of instruction processing. While one instruction is being executed, another can be fetched, and yet another can be decoded. This creates a form of parallelism, increasing the throughput of instructions.

The 8086 uses a **6-byte prefetch queue** to store upcoming instructions while the current instruction is being executed. This allows the processor to fetch new instructions from memory before the current instruction is fully processed, thus speeding up execution.

- **Effect of a High Signal on the RESET Pin:**
  When a **high signal** is applied to the **RESET** pin:

  1. The **processor resets** its internal registers and starts from a predefined memory address (usually **FFFF:0000**).
  2. All **segment registers** are initialized to 0.
  3. The **Instruction Pointer (IP)** is reset to 0000.
  4. The **status flags** and **control flags** are cleared.
  5. The processor begins execution from the memory address pointed to by the **CS**

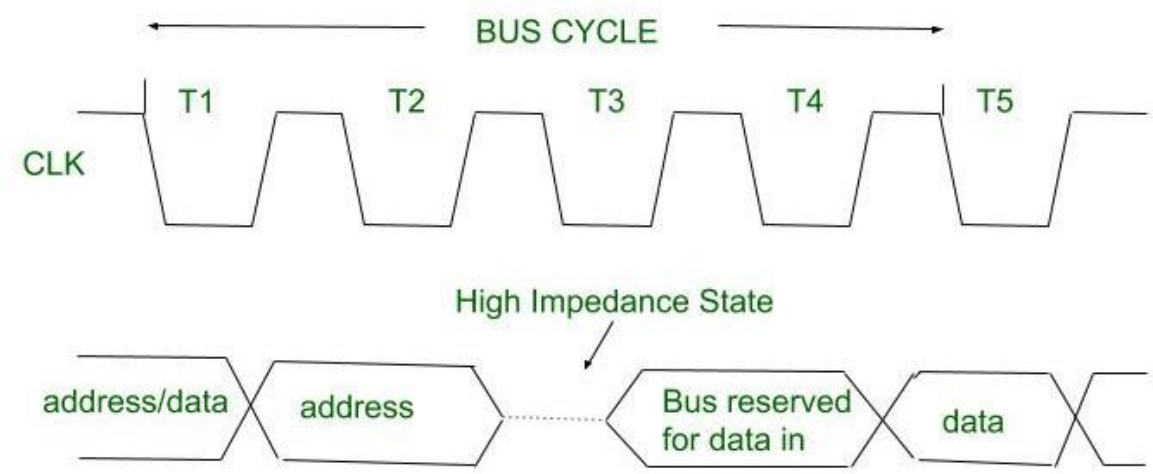
combination. This process ensures that the processor starts in a known state, ready for proper system initialization.

Q27. Define the bus cycle and minimum mode read and write bus cycles with proper timing diagram.

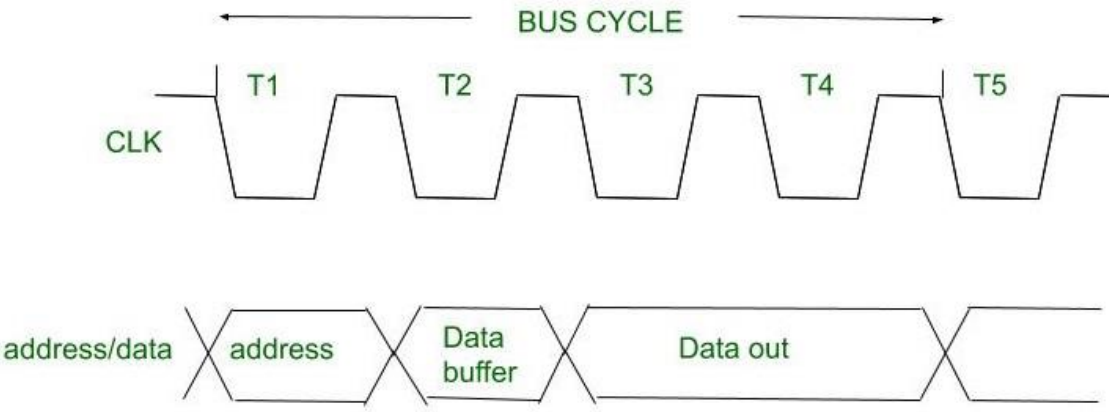
Answer:

Timing Diagrams:

1. **Read Cycle :**
   When a read cycle is to be performed, during T_1 microprocessor puts an address on address bus, and then bus is put in high impedance state during T_2 state. Data to be read must be out on bus during T_3 and T_4. During T_3 bus is made “reserved for data in” and finally data is read during T_4.



2. **Write Cycle :**
   In case of write memory cycle, during T_1 state microprocessor puts an address on address bus. Data is put on data bus by CPU during T_2 state and maintained during T_3 and T_4 states, that is written out to memory or I/O devices.



Minimum mode configuration

1. The 8086 microprocessor operates in minimum mode when MN/MX' = 1, generating all control signals for memory and I/O, simplifying circuit design but lacking multiprocessing support.
2. It features a 20-bit address bus, allowing access to 1MB of memory, with address lines A0 to A15 multiplexed with the data bus.
3. Address lines serve as address lines during the first T state and as data lines afterward, while A16 to A19 are multiplexed with S3 to S6 and BHE' with S7.

Q28. Describe the sequence of signals that occurs on the address bus, the control bus, and the data bus when a simple microcomputer fetches an instruction.

When a microcomputer fetches an instruction, the following sequence of signals occurs across the **address bus**, **control bus**, and **data bus**:

1. **Address Bus:**
    - The CPU places the address of the instruction to be fetched on the address bus.
    - This address corresponds to the memory location where the instruction resides.
2. **Control Bus:**
    - The CPU asserts control signals such as **Read** and **Memory Enable** on the control bus to indicate that it is fetching an instruction.
    - In 8086, the **ALE (Address Latch Enable)** signal is activated to latch the address onto external latches.
3. **Data Bus:**
    - The memory responds by placing the data (the instruction) onto the data bus.
    - The CPU reads this data (the instruction) from the data bus and transfers it to the instruction queue or internal registers

---

Level C

29. Maskable and Non-Maskable Interrupts

- **Maskable Interrupts**: These interrupts can be disabled or ignored by the processor when it is executing other instructions. They are used for standard events that do not require immediate attention. For example, keyboard interrupts can be masked. In the 8086 microprocessor, an interrupt can be masked by clearing the Interrupt Flag (IF) in the Flags Register, allowing the CPU to ignore these interrupts until it is ready to handle them.
- **Non-Maskable Interrupts (NMI)**: These interrupts cannot be disabled and are used for urgent conditions that require immediate attention, such as hardware malfunctions. An example is a power failure interrupt, which ensures that the system can take corrective action, like saving data.
- **Masking an Interrupt in 8086**: To mask an interrupt in 8086, you can set the Interrupt Flag (IF) to 0, effectively disabling all maskable interrupts. When the CPU completes its

critical operation, the IF can be set back to 1, allowing interrupts to be recognized once again.

30. Addressing Modes in 8086

- **Addressing Mode**: An addressing mode determines how the operand of an instruction is accessed or specified. Different modes provide flexibility in programming and efficient memory utilization.
- **Four Addressing Modes**:
    1. **Immediate Addressing**: The operand is specified directly within the instruction. For example, MOV AL, 5 moves the immediate value 5 into register AL. This mode is useful for constants and literals.
    2. **Register Addressing**: The operand is held in a register. For instance, MOV AX, BX copies the value from register BX to AX. This mode is efficient since it accesses data in the processor's fast registers.
    3. **Direct Addressing**: The address of the operand is given explicitly. For example, MOV AX, [1234h] moves the data from memory location 1234h into AX. This mode is straightforward but less flexible.
    4. **Indirect Addressing**: The operand's address is specified indirectly through a register. For instance, MOV AX, [BX] retrieves the data from the memory address contained in BX. This allows dynamic data access and is often used for arrays.

31. Memory Organization of 8086

- The 8086 microprocessor has a segmented memory architecture, which enables efficient memory management and organization. It can address up to 1MB of memory, divided into segments of 64KB each, allowing the processor to access different segments for code, data, and stack.
- **Timing Diagram of Memory Write Cycle**: In a typical memory write cycle, the 8086 places the address on the address bus and the data on the data bus. It then asserts the Memory Write signal (MEMW) to indicate that data is being written. The cycle consists of several clock cycles, with the first clock cycle (T1) focusing on address placement, followed by data placement in T2 and T3. This timing ensures proper synchronization between the processor and memory.

32. Timing Diagram of Memory Read Bus Cycle

- In a standard memory read cycle, the 8086 places the address on the address bus and activates the Memory Read signal (MEMR). The processor waits for the data to be available on the data bus.
- **Modified Timing Diagram with READY Signal**: If the READY signal goes low during the second T state, it indicates that the memory is not ready to provide data. The 8086 will extend the read cycle until the READY signal returns high, ensuring data integrity. This adjustment helps accommodate slower memory devices without impacting the overall operation of the processor.

33. General Purpose Registers of 8086

- The general-purpose registers in the 8086 include AX (Accumulator), BX (Base Register), CX (Count Register), and DX (Data Register). Each of these registers plays a crucial role in the operation of the microprocessor.
- **Functions**:
    - **AX (Accumulator)**: Used for arithmetic operations and serves as the primary register for operations like multiplication and division. It is also used for input/output operations with certain instructions.
    - **BX (Base Register)**: Primarily used for addressing data in memory, it can also serve as a pointer to data structures, enabling efficient data access.
    - **CX (Count Register)**: Used in loop operations and string processing. It acts as a counter for repeated operations, enhancing efficiency in tasks such as looping through arrays.
    - **DX (Data Register)**: Used in I/O operations and can hold intermediate results during arithmetic operations, particularly in multiplication and division.

34. Opcode Pre-Fetch Queue in 8086

The opcode pre-fetch queue is a significant feature of the 8086 microprocessor that enhances its performance by allowing the processor to fetch instructions ahead of time. This queue can hold up to six bytes of opcode, facilitating a more efficient execution process. Here's how it works and its importance:

- **Operation**: When the 8086 executes instructions, it does not wait for each instruction to complete before fetching the next one. Instead, it begins to pre-fetch subsequent instructions while the current instruction is being decoded and executed. This process reduces idle time for the processor and optimizes the overall throughput of instruction execution.

- **Efficiency**: The pre-fetch queue operates in parallel with the instruction execution, allowing the CPU to decode the next instruction while simultaneously executing the current one. This overlapping of operations minimizes the time the processor spends waiting for instruction fetching and significantly increases the number of instructions processed per unit of time.
- **Branching and Control Flow**: In cases of branching or jumps in the program, the pre-fetch queue can become misaligned with the instruction sequence. The 8086 uses a mechanism to discard the pre-fetched instructions that are no longer relevant, ensuring that the correct instructions are executed without unnecessary delays.
- **Performance Impact**: By utilizing the pre-fetch queue, the 8086 can maintain a high instruction throughput, making it suitable for applications requiring rapid processing speeds. This feature was a crucial factor in enhancing the overall efficiency of the 8086 architecture, contributing to its popularity in the early days of personal computing.

In summary, the opcode pre-fetch queue is integral to the 8086 microprocessor's architecture, facilitating faster instruction processing and improving overall system performance.

35. Differences Between 8086 and 8088 Microprocessors

Feature	8086 Microprocessor	8088 Microprocessor
Data Bus Width	16 bits	8 bits
Address Bus Width	20 bits	20 bits
Data Transfer Rate	Higher due to 16-bit data bus	Lower due to 8-bit data bus
Architecture	More complex with separate bus for data and address	Simplified architecture, single bus for data and address
Memory Segmentation	Supports efficient memory segmentation with 64KB segments	Also supports segmentation, but with reduced efficiency due to the data bus width
Cost and Complexity	More expensive and complex due to 16-bit design	Less expensive and simpler design
Usage	Suitable for high-performance applications	Ideal for low-cost, simpler systems
Instruction Set	Similar, but 8086 can handle more data at once	Also similar, but less efficient for large data operations

This table summarizes the key differences between the 8086 and 8088 microprocessors, highlighting their respective strengths and weaknesses in terms of architecture, data handling, and applications.

36. Instruction Cycle, Machine Cycle, and T-State

- **Instruction Cycle**: This refers to the complete cycle of fetching, decoding, and executing an instruction. It includes multiple machine cycles, encompassing all necessary actions for the CPU to perform the instruction.
- **Machine Cycle**: A machine cycle is the time required for the processor to complete a single basic operation, such as reading or writing data. Each machine cycle consists of multiple T-states, which represent the smallest time unit in the microprocessor's operation.
- **T-State**: The T-state is the basic clock cycle of the microprocessor, representing the time taken for one state in a machine cycle.
- **Instruction Format**: The instruction format of the 8086 includes an operation code (opcode) that specifies the operation to be performed, followed by operand(s) that indicate the data to be used.
- **Difference Between Assembler Directive and Instructions**: Assembler directives are commands that provide instructions to the assembler itself, guiding it on how to process the code (e.g., ORG to set the origin address). In contrast, instructions tell the CPU what specific operations to perform (e.g., MOV to move data).

Unit 2

2 Marks Questions

1. **What is the purpose of the AX register?**
   The AX register (Accumulator Register) is primarily used for arithmetic operations and I/O operations in the 8086 microprocessor. It can hold operands for instructions and store results.
2. **What is the purpose of the BX register?**
   The BX register (Base Register) serves as a pointer to data in the memory. It is often used for addressing data and can also store base addresses in base addressing modes.

3. **What is the purpose of the CX register?**
   The CX register (Count Register) is used primarily for loop control and counting operations. It is often involved in string and loop instructions, where it counts the number of iterations.

4. **What is the purpose of the DX register?**
   The DX register (Data Register) is used to store data for I/O operations and can also hold the high-order part of the result in multiplication and division operations.

5. **Explain about CY and AC in 8086 flag register.**
   The CY (Carry Flag) indicates an overflow in arithmetic operations, while the AC (Auxiliary Carry Flag) indicates a carry from the lower nibble to the upper nibble in binary-coded decimal (BCD) operations.

6. **What is the difference between loop and branch instruction?**
   Loop instructions repeat a block of code a specified number of times, usually using the CX register. Branch instructions redirect the flow of execution to a different part of the code based on a condition.

7. **What is interrupt? How is it classified?**
   An interrupt is a signal that temporarily halts the CPU's current operations to execute a specific routine. It is classified into two types: maskable (can be ignored) and non-maskable (cannot be ignored).

8. **What is masking? Why is it required?**
   Masking is the process of enabling or disabling interrupts. It is required to ensure that critical tasks are not interrupted, maintaining system stability during important operations.

9. **List down the advantages of modular programming.**
   Advantages include improved code organization, easier debugging and maintenance, enhanced reusability of code modules, and the ability to work on modules independently in a team environment.

10. **List down the disadvantages of modular programming.**
    Disadvantages include potential performance overhead due to function calls, increased complexity in managing modules, and possible difficulties in module integration if not designed properly.

11. **Briefly explain about string instructions.**
    String instructions in the 8086 processor perform operations on strings of data,

including copying, comparing, scanning, and filling. They typically use the SI and DI registers to point to the source and destination addresses.

12. **Which instruction can be used to call a procedure?**
    The CALL instruction is used to invoke a procedure, saving the return address on the stack so execution can resume after the procedure completes.

13. **What is the addressing mode of the instruction MOV AX, [BX+SI+06]?**
    The addressing mode is **Base Index Displacement** addressing, as it combines the base register (BX), index register (SI), and a displacement value (06).

14. **(a) Index addressing (b) Base addressing (c) Base index addressing (d) Base index displacement**
    The correct answer is **(d) Base index displacement**.

15. **What is the addressing mode of the instruction MOV AX, [BX+SI+06]?**
    The addressing mode is **Base Index Displacement** addressing.

16. **(a) Index addressing (b) Base addressing (c) Base index addressing (d) Base index displacement**
    The correct answer is **(d) Base index displacement**.

17. **A microcontroller has:**
    (d) all of these: ROM, RAM, and I/O ports are all integral components of a microcontroller.

18. **Physical address of 8086 is:**
    (c) 20-bit: The physical address space of the 8086 is 20 bits, allowing access to 1 MB of memory.

19. **8086 has the following units:**
    (a) EU and BIU: The 8086 consists of an Execution Unit (EU) and a Bus Interface Unit (BIU).

20. **The 8086 has memory segments:**
    (c) 4: The 8086 architecture divides memory into four segments: Code, Data, Stack, and Extra segments.

21. **Which of the following instruction is a four-byte instruction?**
    (d) ADD AX, [BP+0200]: This instruction requires four bytes for its operation, including the opcode and operand.

22. **Direction flag is used with which of the following instructions?**
    (c) String: The direction flag determines the direction (increment or decrement) for string operations.

Level B

25. Explain String Instructions in 8086

String instructions in the 8086 microprocessor are specialized commands designed to manipulate sequences of bytes or words. These instructions operate on data arrays and utilize the SI (Source Index) and DI (Destination Index) registers to point to the source and destination addresses in memory. Common string instructions include:

- **MOVS**: Moves a string from the source address to the destination address.
- **CMPS**: Compares two strings at the source and destination addresses.
- **SCAS**: Scans a string for a specified value.
- **LODS**: Loads a string into the accumulator.
- **STOS**: Stores a value from the accumulator into a string.

The execution of these instructions can be modified by the Direction Flag (DF), which determines whether the indices increment or decrement.

26. Briefly Explain the Use of REP Prefix with a Suitable Example

The REP prefix in the 8086 microprocessor is used to repeat string instructions for a specified number of times based on the count in the CX register. It enables efficient processing of strings without requiring multiple explicit instructions.

Example: If you want to move a block of data from one memory location to another, the following code snippet uses the REP MOVS instruction:

```
MOV CX, 5          ; Set the count to 5
LEA SI, Source     ; Load address of source string
LEA DI, Destination; Load address of destination string
REP MOVSB          ; Repeat moving bytes
```

This will move 5 bytes from the source to the destination.

27. Explain Any Two Assembler Directives Used in 8086 Microprocessor

1. **ORG (Origin Directive)**: This directive specifies the starting address for the program or data in memory. It helps in defining where the assembler should place the code in memory.

   Assembly:

```
ORG 100h ; The program starts at memory address 0100h
```

2. **DB (Define Byte)**: This directive is used to allocate storage for a byte or a series of bytes in memory. It initializes the memory with specified values.

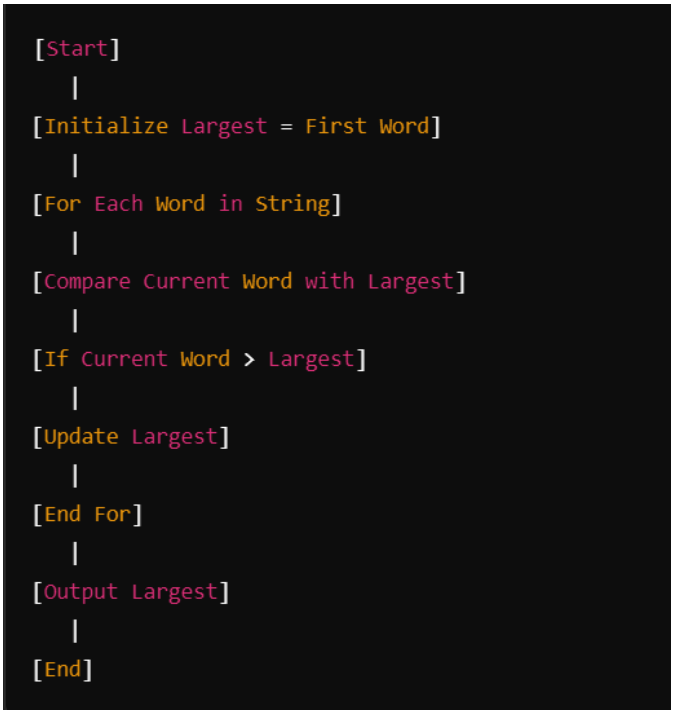
   Assembly:

```
DATA DB 10h, 20h, 30h ; Define three bytes in memory
```

28. Explain an ALP to Find Out the Largest Number from a String of Words with a Neat Flowchart

Algorithm:

1. Initialize a pointer to the first word.
2. Load the first word into a register as the largest number.
3. Compare each subsequent word with the current largest number.
4. If a larger number is found, update the largest number.
5. Repeat until all words are processed.

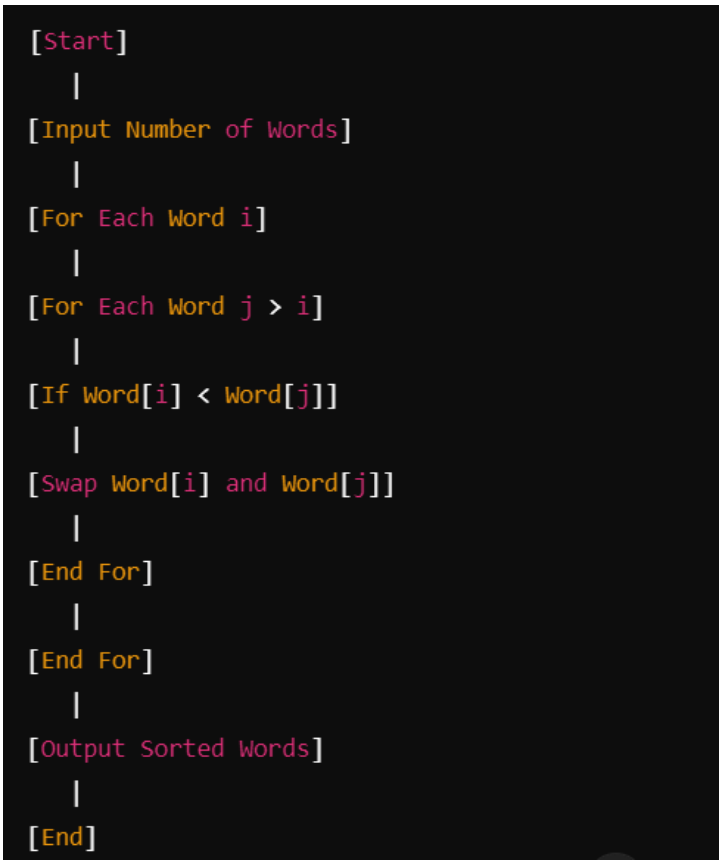


29. Explain an ALP to Arrange a String of Words in Descending Order with a Neat Flowchart

Algorithm:

1. Read the number of words.
2. For each word, compare it with every other word.
3. If a word is found that is larger than the current word, swap their positions.
4. Repeat until the entire list is sorted.

Flowchart:



30. Briefly Explain SHL/SAL Instruction

The SHL (Shift Left) and SAL (Arithmetic Shift Left) instructions are used to shift the bits of an operand to the left by a specified number of positions. Both instructions function identically in the context of the 8086 microprocessor.

- **Function**: Each left shift multiplies the operand by 2. The leftmost bit is discarded, and a zero is shifted into the rightmost position.

Example:

assembly

```
MOV AL, 3     ; AL = 00000011
SHL AL, 1     ; AL = 00000110 (3 * 2 = 6)
```

31. Explain the Functionality of TEST Instruction of 8086 Microprocessor with an Example

The TEST instruction performs a bitwise AND operation between two operands but does not store the result. Instead, it updates the flags based on the result of the operation, specifically the Zero Flag (ZF) and the Sign Flag (SF).

Example:

assembly

```
MOV AX, 5     ; AX = 0000000000000101
TEST AX, 3    ; AND operation with 0000000000000011
```

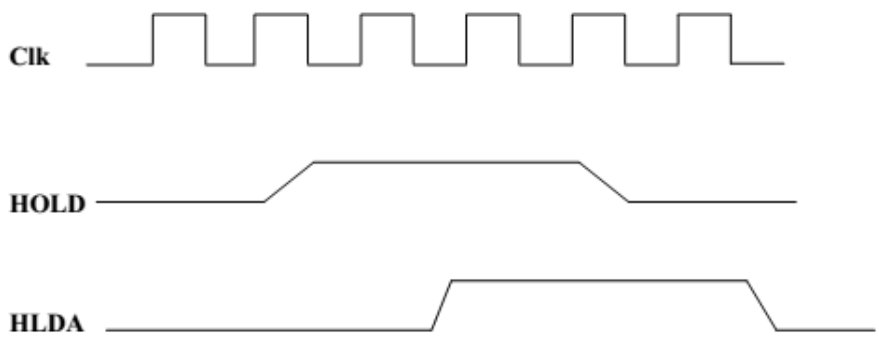
After execution, the Zero Flag will be cleared (since the result is not zero) and the Sign Flag will reflect the sign of the result.

32. Draw the Bus Request and Bus Grant Timings in Minimum Mode System

In minimum mode, the bus request and grant timing sequence involves the following steps:

1. **Bus Request**: The CPU signals the request to gain control of the system bus.
2. **Bus Grant**: The CPU receives the acknowledgment that it can control the bus.

Timing Diagram:



Bus Request and Bus Grant Timings in Minimum Mode System

Level C

33. Explain the Concept of Flag Registers in the 8086 Microprocessor

Introduction to Flag Registers

The 8086 microprocessor features a special register known as the FLAGS register (also called the status register), which is critical for controlling the operation of the CPU and the flow of programs. The FLAGS register contains various individual flags that reflect the outcomes of operations and determine the behavior of subsequent instructions.

Types of Flags

1. **Sign Flag (SF)**: Indicates the sign of the result of an operation. It is set if the most significant bit (MSB) of the result is 1 (negative number).
    - **Example**: After executing SUB AL, BL, if the result in AL is negative, SF is set.
2. **Zero Flag (ZF)**: Set if the result of an arithmetic or logical operation is zero. This flag is essential for making decisions based on comparison results.
    - **Example**: After executing CMP AX, BX, if AX equals BX, ZF is set.
3. **Parity Flag (PF)**: Set if the number of set bits in the result is even. It is used in error checking.
    - **Example**: In an operation such as AND AL, 0Fh, if the result has an even number of bits set, PF is set.
4. **Carry Flag (CF)**: Indicates a carry out or borrow in arithmetic operations. It is particularly important for unsigned arithmetic.
    - **Example**: After executing ADC AX, BX, if there is a carry from the addition, CF is set.

Instructions that Modify Flags

- **Arithmetic Instructions**: ADD, SUB, INC, DEC modify ZF, SF, and CF based on the results.
- **Logical Instructions**: AND, OR, XOR affect the ZF and SF based on the outcomes.
- **Comparison Instructions**: CMP sets the flags without changing the operands.

Significance

The FLAGS register enables conditional branching in programs, allowing decisions based on previous operations. For example, conditional jump instructions like JZ (Jump if Zero) rely on the ZF to control the program's flow, executing or skipping parts of code based on computations.

---

34. Discuss the Significance of the FLAGS Register in 8086 Assembly Language Programming

Role of the FLAGS Register

The FLAGS register is central to the execution of programs in 8086 Assembly language. It provides the processor with important status information after arithmetic, logical, and comparison operations.

Conditional Jump Instructions

Conditional jumps such as JZ, JNZ, JC, and JNC utilize the status of the flags to determine the next instruction to execute. For instance:

- **JZ (Jump if Zero)**: Jumps to a specified label if ZF = 1.
- **JNE (Jump if Not Equal)**: Jumps if ZF = 0.

Example of Conditional Jumps

assembly

```
MOV AX, 5
MOV BX, 5
CMP AX, BX  ; Compare AX and BX
JZ EqualLabel ; Jump if Zero (ZF = 1)
...
EqualLabel:
; Code to execute if AX equals BX
```

Significance

These instructions allow the program to make decisions, creating loops, if-else structures, and enabling complex logic flow based on real-time calculations.

---

35. Describe the Purpose of the CALL and RET Instructions in Subroutine Calls

Purpose of CALL and RET

The CALL and RET instructions are used for managing subroutines (functions) in 8086 assembly language programming.

- **CALL**: Saves the address of the next instruction onto the stack and transfers control to the specified subroutine. This allows for modular programming and code reusability.
- **RET**: Pops the saved address from the stack and returns control to that location, continuing program execution.

Example Program Demonstrating Subroutines

assembly

```
section .text
main:
    CALL MySubroutine ; Call the subroutine
    ; Continue execution
    ...

MySubroutine:
    ; Code for the subroutine
    RET ; Return to the caller
```

Significance

Using CALL and RET enhances code organization, simplifies debugging, and allows for more structured programs by enabling the reuse of code blocks.

---

36. Explain the Role of the Stack in 8086 Assembly Language Programming

Stack Overview

The stack is a special area in memory used for storing temporary data, such as local variables, return addresses, and parameters for subroutines. It operates on a Last In, First Out (LIFO) principle, which is crucial for managing nested function calls and local storage.

Stack Operations

1. **PUSH**: Decreases the stack pointer (SP) and stores the value at the top of the stack.
2. **POP**: Retrieves the value from the top of the stack and increases the stack pointer.

Example Program Demonstrating Stack Operations

assembly

```
section .text
main:
    MOV AX, 1234h
    PUSH AX         ; Push AX onto the stack
    MOV AX, 5678h
    POP BX          ; Pop the top of the stack into BX (BX = 1234h)
    ; Continue execution
```

Significance

The stack is essential for managing function calls, local variables, and control flow, ensuring that programs can handle multiple nested operations without losing data.

---

37. Discuss the Purpose of the INT (Interrupt) Instruction in 8086 Assembly Language Programming

Purpose of the INT Instruction

The INT instruction is used to generate software interrupts, allowing programs to request services from the operating system or hardware. It provides a way for programs to communicate with the system's interrupt handler, facilitating tasks like input/output operations, error handling, and more.

Examples of Software Interrupts

1. **INT 21h**: Used in DOS for various system services such as file handling, keyboard input, and screen output.

assembly

```
MOV AH, 09h
LEA DX, message
INT 21h ; Call DOS to display a string
```

2. **INT 10h**: Used for video services, such as changing the display mode or writing characters to the screen.

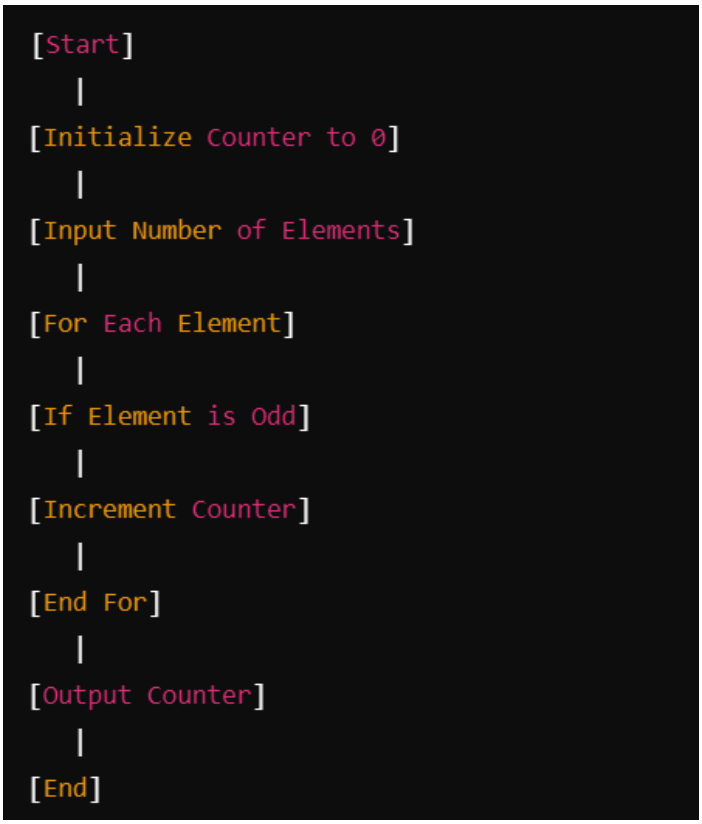
Significance

Interrupts enhance the efficiency and responsiveness of programs by allowing asynchronous handling of events and operations, making them fundamental to modern programming practices.

---

38. Draw the Flowchart and Write an Assembly Language Program to Count the Number of Odd Numbers

Flowchart



Example Assembly Language Program

```
section .data
    count db 0
    msg db "Odd numbers count: ", 0

section .text
global _start

_start:
    ; Assume numbers are already input in memory
    MOV CX, NumberOfElements ; Number of elements
    XOR BX, BX               ; Counter for odd numbers

next_number:
    ; Load next number into AX
    ; (Assuming numbers are in an array)
    MOV AL, [array + BX]
    TEST AL, 1               ; Check if odd
    JZ skip                  ; Jump if even
    INC count                ; Increment odd counter

skip:
    INC BX                   ; Move to the next number
    LOOP next_number         ; Repeat for all numbers

    ; Output the count (assume code to output msg is here)
    ; (Output logic not shown for brevity)
    ; ...
```

---

39. Explain the Following Instructions with an Example: (i) PUSH (ii) POP

(i) PUSH

The PUSH instruction saves a value onto the stack and decrements the stack pointer (SP). This operation is crucial for preserving data before a subroutine call or before performing operations that might overwrite registers.

Example:

assembly

```
MOV AX, 1234h
PUSH AX ; Push AX onto the stack
```

(ii) POP

The POP instruction retrieves the top value from the stack and increments the stack pointer. This operation restores previously saved data.

Example:

```
POP BX ; Pop the top of the stack into BX
```

---

40. What is Data Copy/Transfer Instructions? Give an Example. Explain Register Addressing Mode with an Example.

Data Copy/Transfer Instructions

Data copy/transfer instructions in assembly language are used to move data from one location to another. These instructions include MOV, PUSH, POP, and others, facilitating data manipulation and storage.

Example of MOV:

assembly

```
MOV AX, BX ; Copy the value in BX to AX
```

Register Addressing Mode

In register addressing mode, operands are specified directly in the registers. This mode is fast and efficient because it eliminates the need for memory access.

Example:

assembly

```
MOV CX, AX ; Move the value in AX to CX (register addressing mode)
```