## Chapter 2 – Understand the Applications of Assembly Language

---

### Short Questions

**2.5 Discuss Fields of Assembly Language Statement**

**Q1. What are the main fields of an Assembly language statement?**

An Assembly instruction is divided into four main fields: **Label, Opcode, Operand, and Comment**. Each part has a specific role in defining the instruction.

**Q2. Explain the Label field.**

The Label field is used to mark a line in the program with a name. It is helpful for jumps and loops, so the processor can easily return to that line.

**Q3. Explain the Opcode and Operand fields.**

The **Opcode** specifies the operation (e.g., MOV, ADD). The **Operand** specifies the data or location where the operation is performed, such as registers or memory.

**Long Question**

**Q: Explain the different fields of Assembly Language Statement with examples.**

Assembly language instructions have the following fields:

1. **Label Field:**
    - Identifies a memory address for an instruction.
    - Example: LOOP: ADD AX, BX.
2. **Opcode Field:**
    - Defines the operation to perform.
    - Example: MOV means move data.
3. **Operand Field:**
    - Specifies the data source or destination.
    - Example: MOV AX, 1234H moves value 1234H into AX.
4. **Comment Field:**
    - Adds explanation for the programmer, ignored by the assembler.
    - Example: ; This adds AX and BX.

Thus, fields in Assembly ensure readability and proper instruction execution.

---

**2.6 Compare Assembly Language with Machine Language**

**Short Questions**

**Q1. What is machine language?**

Machine language is the lowest-level language consisting of binary codes (0s and 1s). It is directly executed by the processor but is difficult for humans to understand.

**Q2. What is assembly language?**

Assembly language is a symbolic representation of machine language using mnemonics like MOV, ADD, and JMP. It is easier to write than binary codes.

**Q3. How is Assembly better than Machine language?**

Assembly is more readable and less error-prone. For example, ADD AX, BX is much easier to understand than 0000000100001011.

**Long Question**

**Q: Compare Assembly language with Machine language.**

1. **Machine Language:**
    - Uses binary code (0s and 1s).
    - Directly understood by CPU.
    - Very fast but unreadable for humans.
2. **Assembly Language:**
    - Uses symbolic mnemonics like MOV, SUB, JMP.
    - Requires an assembler to convert into machine code.
    - Easier for programmers to understand.

**Comparison:**

- Machine language is difficult to write, debug, and maintain.

- Assembly is easier and closer to human understanding, though still hardware-dependent.
- Example: Machine code 10110000 01100001 = Assembly MOV AL, 61H.

Hence, Assembly is a user-friendly version of machine code.

---

**2.7 Describe Data Addressing Modes**

**Short Questions**

**Q1. What is an addressing mode?**

Addressing modes define how the operand of an instruction is selected. It tells the CPU whether the data is in a register, memory, or is given directly.

**Q2. Why are addressing modes needed?**

They provide flexibility in programming by allowing data to be accessed in different ways. This improves efficiency in program execution.

**Q3. List different addressing modes of 8086.**

The 8086 supports: Immediate, Register, Direct, Indirect, Indexed, and Base-Indexed modes.

**Long Question**

**Q: Explain different data addressing modes in 8086 with examples.**

1. **Immediate Addressing:** Operand is directly in instruction.
    - Example: MOV AX, 05H.
2. **Register Addressing:** Operand is in a register.
    - Example: MOV AX, BX.
3. **Direct Addressing:** Operand is at a specific memory location.
    - Example: MOV AX, [1234H].
4. **Indirect Addressing:** Memory address is stored in a register.
    - Example: MOV AX, [BX].
5. **Indexed Addressing:** Uses index registers for array access.
    - Example: MOV AX, [SI+05H].
6. **Base-Indexed Addressing:** Combines base and index registers.
    - Example: MOV AX, [BX+SI].

Addressing modes make the CPU more versatile in handling different data storage formats.

---

**2.8 Apply Data Movement Instructions**

**Short Questions**

**Q1. What are Data Movement instructions?**

These instructions are used to transfer data between registers, memory, and I/O devices. They don't change the actual data, only move it.

**Q2. Give examples of Data Movement instructions.**

Examples include MOV AX, BX, PUSH AX, POP BX, and XCHG AX, BX.

**Q3. What is the purpose of PUSH and POP?**

PUSH saves register data into the stack, while POP retrieves data from the stack back into a register.

**Long Question**

**Q: Explain Data Movement Instructions in 8086 with examples.**

1. **MOV Instruction:** Moves data between registers/memory.
    - Example: MOV AX, BX.
2. **PUSH & POP:** Used for stack operations.
    - Example: PUSH AX stores AX in stack, POP BX retrieves it.
3. **XCHG:** Exchanges contents of two registers.
    - Example: XCHG AX, BX.
4. **LEA (Load Effective Address):** Loads address into a register.
    - Example: LEA SI, [1234H].

These instructions form the backbone of assembly programming as they allow smooth data transfer within CPU and memory.

---

**2.9 Apply Arithmetic and Logic Instructions**

**Short Questions**

**Q1. What are Arithmetic Instructions?**

Arithmetic instructions perform mathematical operations like addition, subtraction, multiplication, and division.

**Q2. What are Logic Instructions?**

Logic instructions perform bitwise operations such as AND, OR, XOR, and NOT. They are used in condition checking and masking.

**Q3. Give two examples of Arithmetic and Logic instructions.**

Arithmetic: ADD AX, BX, SUB AX, 05H.

Logic: AND AX, 0F0H, OR BX, 01H.

**Long Question**

**Q: Explain Arithmetic and Logic Instructions of 8086 with examples.**

1. **Arithmetic Instructions:**
    - **ADD:** Adds two operands. ADD AX, BX.
    - **SUB:** Subtracts. SUB AX, 05H.
    - **INC/DEC:** Increment or decrement. INC CX.
    - **MUL/DIV:** Multiplication & division.
2. **Logic Instructions:**
    - **AND:** Bitwise AND. AND AX, 0F0H.
    - **OR:** Bitwise OR. OR BX, 01H.
    - **XOR:** Exclusive OR. XOR AX, AX clears AX.
    - **NOT:** Inverts bits.

Arithmetic instructions are used for calculations, while logic instructions are used for decision-making in programs.

---

**2.10 Apply Program Control Instructions**

**Short Questions**

**Q1. What are Program Control Instructions?**

These instructions change the normal flow of execution in a program. Examples include jumps, calls, returns, and interrupts.

**Q2. What is the difference between JMP and CALL?**

JMP transfers control unconditionally to another address, while CALL transfers control to a subroutine and returns after execution.

**Q3. Give two examples of program control instructions.**

Examples: JMP START, CALL SUB1, RET, and INT 21H.

**Long Question**

**Q: Explain Program Control Instructions of 8086 with examples.**

1. **JMP (Jump):** Transfers execution to another part of program.
    - Example: JMP LOOP1.
2. **Conditional Jumps:** Jump only if a condition is true.
    - Example: JE, JNE, JC.
3. **CALL & RET:** CALL transfers control to a subroutine, RET returns back.
    - Example: CALL PRINT.
4. **LOOP:** Repeats instructions until CX becomes zero.
    - Example: LOOP AGAIN.
5. **INT (Interrupt):** Transfers control to a predefined routine.
    - Example: INT 21H for DOS services.