Instruction Format of 8086 Microprocessor

Introduction

An **instruction** in the 8086 microprocessor is a binary-encoded command that specifies an operation for the CPU to perform. The 8086 instruction set is **variable-length** (1 to 6 bytes) and supports arithmetic, logical, data transfer, branching, and control operations.

Key Points About 8086 Instruction Format

- Instruction length varies from **1 to 6 bytes**.
- Comprised of **Opcode, Mod-Reg-R/M, displacement/offset, and immediate data** fields.
- Supports addressing modes: **register, immediate, direct, indirect, and relative**.
- Flexible encoding allows efficient instruction representation with minimal bytes.

General Structure of an 8086 Instruction

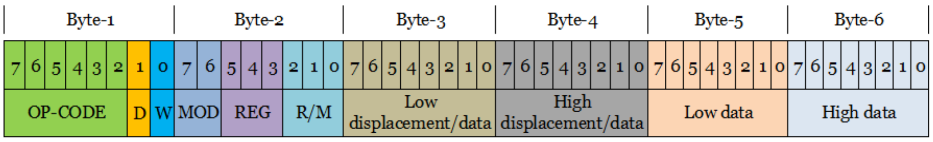

Figure 1: Byte-wise structure of an 8086 instruction

Instruction Components

An 8086 instruction may consist of the following components:

- **Opcode (Operation Code):** Specifies the operation to be performed (e.g., MOV, ADD, SUB, JMP).
- **Addressing Mode / Mod Field:** Indicates how the operand(s) should be accessed (register, memory, or immediate).
- **Register Field (REG):** Specifies the register operand if used.
- **R/M Field (Register/Memory):** Specifies the memory location or register operand.
- **Displacement / Offset:** Used for memory addressing; can be 8-bit or 16-bit.
- **Immediate Data:** Constant value included within the instruction (optional).

Byte-wise Breakdown of Instructions

Byte-wise Structure

Byte	Field / Bits	Description
1	Bits 7-2 (6 bits)	**OP-CODE:** Specifies the operation to be performed (e.g., MOV, ADD, JMP).
	Bit 1 (1 bit)	**D (Direction):** Determines data transfer direction (Reg ↔ Mem).
	Bit 0 (1 bit)	**W (Word):** 0 = Byte operation, 1 = Word / 16-bit operation.
2	Bits 7-6 (2 bits)	**MOD:** Specifies addressing mode: 00 = memory no displacement, 01 = 8-bit displacement, 10 = 16-bit displacement, 11 = register addressing.
	Bits 5-3 (3 bits)	**REG:** Register operand (source or destination depending on D bit).
	Bits 2-0 (3 bits)	**R/M:** Specifies register or memory operand, used in combination with MOD field.
3-4	Optional	**Displacement / Data for memory addressing:**
	Byte 3 (8 bits)	Low displacement or low immediate data.
	Byte 4 (8 bits)	High displacement or high immediate data (combined with Byte 3 for 16-bit values).
5-6	Optional	**Additional Immediate Data (for 16-bit instructions):**
	Byte 5 (8 bits)	Low data byte.
	Byte 6 (8 bits)	High data byte.

Instruction Fields in Detail

1. Opcode Field

Opcode Field

- The **Opcode Field** specifies the operation that the CPU must perform.
- Typically occupies **6 bits** in the first byte of the instruction.
- Determines the type of operation: data transfer, arithmetic, logic, control, or branching.
- The first byte also includes two important modifier bits:
  - **D (Direction) Bit:** Specifies the direction of data transfer between the register and memory.
    - D = 1: The **destination operand** is a register, and the source is memory.
    - D = 0: The **source operand** is a register, and the destination is memory.
  - **W (Word) Bit:** Indicates the size of the data being operated on.
    - W = 0: 8-bit (byte) operation.
    - W = 1: 16-bit (word) operation.
- **Summary of First Byte (Opcode Byte):**

Bits	Description
7–2	6-bit **Opcode** (operation code)
1	**D (Direction) Bit**
0	**W (Word) Bit**

- **Example Opcodes:**
  - `MOV` = 100010b
  - `ADD` = 000000b
  - `SUB` = 001010b
- **Working Example:**
  - Instruction: `MOV AX, [BX]`
  - Opcode = 100010b, D = 1 (destination = AX), W = 1 (16-bit operation)
  - First byte encoding: 10001011b
  - The first byte tells the CPU what operation to perform, which operand is the destination, and the data size.
- Notes:
  - The D and W bits help the CPU distinguish between different variants of the same operation, reducing the need for multiple opcodes.
  - Understanding the first byte is crucial for assembly-level programming and instruction encoding.

2. Mod-Reg-R/M Field (Second Byte)

Mod-Reg-R/M Field

- The **second byte** of an 8086 instruction is called the **Mod-Reg-R/M field**.
- It is an 8-bit field divided into three parts:
  - **MOD (2 bits):** Specifies the addressing mode.
  - **REG (3 bits):** Specifies one register operand. Source or destination depends on the D bit of the first byte.
  - **R/M (3 bits):** Specifies the other operand, either a register or a memory location, depending on MOD.
- **MOD Field (Addressing Mode):**

MOD Bits	Addressing Mode
00	Memory, no displacement (except BP)
01	Memory, 8-bit displacement
10	Memory, 16-bit displacement
11	Register addressing (both operands in registers)

- **REG Field (3 bits):** Specifies the register used as one operand. Which operand depends on the D bit in byte-1:
  - D = 1: REG = destination operand
  - D = 0: REG = source operand
- REG field encoding for 8086:

REG Bits	W = 0 (8-bit)	W = 1 (16-bit)
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

- **R/M Field (3 bits):** Determines the register or memory operand based on the MOD bits.

R/M	MOD=00	MOD=01	MOD=10	MOD=11	Segment
000	BX+SI	BX+SI+D8	BX+SI+D16	AL/AX	DS
001	BX+DI	BX+DI+D8	BX+DI+D16	CL/CX	DS
010	BP+SI	BP+SI+D8	BP+SI+D16	DL/DX	SS
011	BP+DI	BP+DI+D8	BP+DI+D16	BL/BX	SS
100	SI	SI+D8	SI+D16	AH/SP	DS
101	DI	DI+D8	DI+D16	CH/BP	DS
110	Direct addr*	BP+D8	BP+D16	DH/SI	DS/SS
111	BX	BX+D8	BX+D16	BH/DI	DS

* Direct addressing: 16-bit memory address is used instead of register combination.

- Notes:
  - If MOD = 11, both operands are in registers.
  - If MOD $\neq$ 11, one operand is in memory and the other is in a register.
  - **Important:** Direct memory-to-memory operations are **not allowed** in 8086. At least one operand must be a register.
  - The D and W bits of the first byte determine which operand is the source and which is the destination, and whether the operation is 8-bit or 16-bit.

Memory/Register Combinations

- **Register-to-register:** MOD = 11
- **Register-to-memory or Memory-to-register:** MOD = 00, 01, or 10
- **Memory-to-memory:** Not allowed. Use register as intermediary.

Example: MOV Instruction

Instruction: `MOV AX, [BX+SI]`

- Opcode: 100010 (MOV, 16-bit register/memory)
- MOD: 00 (memory, no displacement)
- REG: 000 (AX register)
- R/M: 000 ([BX+SI])
- Encoded bytes (simplified): 10001000 00000000

- Note: One operand is memory, the other is register → valid. Memory-to-memory not allowed.

3. Displacement / Offset Field

Displacement / Offset Field

- The **Displacement (Offset) Field** is an optional part of 8086 instructions.
- It provides an **8-bit or 16-bit value** added to the effective address during memory access.
- The displacement is used in combination with base and/or index registers to calculate the **final memory address**.
- Examples:
  - `MOV AX, [BX+SI+05H]` → 8-bit displacement = 05H
  - `MOV AX, [BP+DI+1234H]` → 16-bit displacement = 1234H
- Notes:
  - Displacement size is determined by the **MOD field** in the Mod-Reg-R/M byte:
    - MOD = 01 → 8-bit displacement
    - MOD = 10 → 16-bit displacement
    - MOD = 00 → no displacement (except direct addressing or BP)
  - Displacement allows flexible addressing of memory locations without changing instruction encoding for registers.
- Memory access formula:

$$\text{Effective Address} = \text{Base Register} + \text{Index Register} + \text{Displacement}$$

- Example encoding:
  - Instruction: `MOV AX, [BX+SI+05H]`
  - MOD = 01 (8-bit displacement), REG = AX, R/M = BX+SI
  - Bytes: `100010 00 000 000` 05H (simplified)

4. Immediate Data Field

Immediate Data Field

- The **Immediate Data Field** contains a **constant value** that is directly used by the instruction.
- The size of the immediate data is determined by the **W bit** in the opcode:

- W = 0 → 8-bit immediate
- W = 1 → 16-bit immediate

- Examples:
  - `MOV AL, 5H` → 8-bit immediate data = 05H
  - `MOV AX, 1234H` → 16-bit immediate data = 1234H

- Notes:
  - Immediate data is encoded directly after the opcode and Mod-Reg-R/M bytes (if present).
  - No memory or register reference is needed; the value is embedded in the instruction.
  - For 16-bit operations, low-order byte comes first (little-endian format), followed by the high-order byte.

- Instruction encoding example:
  - Instruction: `MOV AX, 1234H`
  - Opcode: 1011 0000 + W bit = 1
  - Immediate bytes: 34H 12H (low byte first, little-endian)
  - Complete instruction bytes: `B8 34 12` (simplified)

Instruction Size Examples

Examples of Instruction Lengths

- **1-byte instruction:** `NOP` – no operation, only opcode.
- **2-byte instruction:** `MOV AX, BX` – opcode + Mod-Reg-R/M byte.
- **3-byte instruction:** `MOV AX, 1234H` – opcode + immediate 16-bit value.
- **4-byte instruction:** `MOV AX, [BX+SI+05H]` – opcode + Mod-Reg-R/M + 8-bit displacement.

Sample Instruction Formats

MOV Instruction Examples

- `MOV AX, BX` → 2 bytes (register-to-register)
- `MOV AX, 1234H` → 3 bytes (register-to-immediate)
- `MOV AX, [BX]` → 2 bytes (register-to-memory with no displacement)

- `MOV AX, [1234H]` → 4 bytes (register-to-memory with direct addressing)

8086 Instruction Exceptions

- **Exception 1: Direct memory-to-memory transfer is not allowed.**
  - At least one operand must be a register. AX can be used as an intermediate register.
  - **Not allowed:** `MOV [DI], [SI]`
  - **Allowed workaround:**

```
MOV AH, [SI]
MOV [DI], AH
```

- **Exception 2: DS register cannot be loaded directly with a data segment address.**
  - Use a general-purpose register (e.g., AX) to load the address first.
  - Example:

```
MOV AX, DS_ADDR ; load address into AX
MOV DS, AX      ; load AX into DS
```

Conclusion

The 8086 instruction format is **flexible and variable-length**, allowing efficient execution of different operations with multiple addressing modes.

A solid understanding of the **Opcode, Mod-Reg-R/M, Displacement, and Immediate fields** is essential for:

- Assembly-level programming
- Machine code analysis
- Interfacing directly with CPU and memory

Proper knowledge of instruction encoding helps in optimizing code size and execution efficiency.