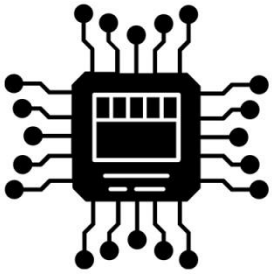


قـســـــــــم علوم الامن السيبراني
DEPARTMENT OF CYBER SECURITY

SUBJECT:

# Microprocessor

CLASS:

SECOND

LECTURER:

Dr. Abdulkadhem A. Abdulkadhem

# LECTURE: (6)

## Data Movement Instructions

# 1. Introduction

Data movement instructions are among the most frequently used instructions in 8086 assembly language programming. They are responsible for transferring data between registers, memory locations, stack, and I/O-related buffers.

A solid understanding of these instructions is essential for mastering program execution flow, parameter passing, array processing, and string handling.

**Exam Importance:**

Data movement instructions appear extensively in **MCQ, tracing questions, and short theoretical questions**.

# 2. Data Transfer Instructions

Data transfer instructions copy data from a source operand to a destination operand **without altering the source**. These instructions do not perform arithmetic or logical operations. They are classified into four groups as explain in table below

| General-purpose transfer instructions | Special address transfer instructions | Simple input/output port transfer instructions | Flag transfer instructions |
|---|---|---|---|
| MOV<br>XCHG<br>PUSH<br>POP | LEA<br>LDS<br>LES | IN<br>OUT | LAHF<br>SAHF<br>PUSHF<br>POPF |

## 2.1 MOV Instruction

The `MOV` instruction is the most fundamental data transfer instruction in 8086.

**General Syntax:**

```
MOV destination, source
```

**Key Rules:**

- Source and destination **cannot both be memory locations**
- Operand sizes must match (8-bit ↔ 8-bit, 16-bit ↔ 16-bit)

- CS cannot be used as a destination register
- Segment registers can only be loaded from registers or memory (not immediate)

**Examples:**

```
MOV AX, BX
MOV AL, 25H
MOV BX, [SI]
MOV [2000H], AX
MOV DS, AX
```

 **Exam Note:**
`MOV [1000H], [2000H]` is **invalid** (memory-to-memory transfer not allowed).

## 2.2 XCHG Instruction

The `XCHG` instruction exchanges the contents of two operands.

**Syntax:**

```
XCHG operand1, operand2
```

**Rules:**

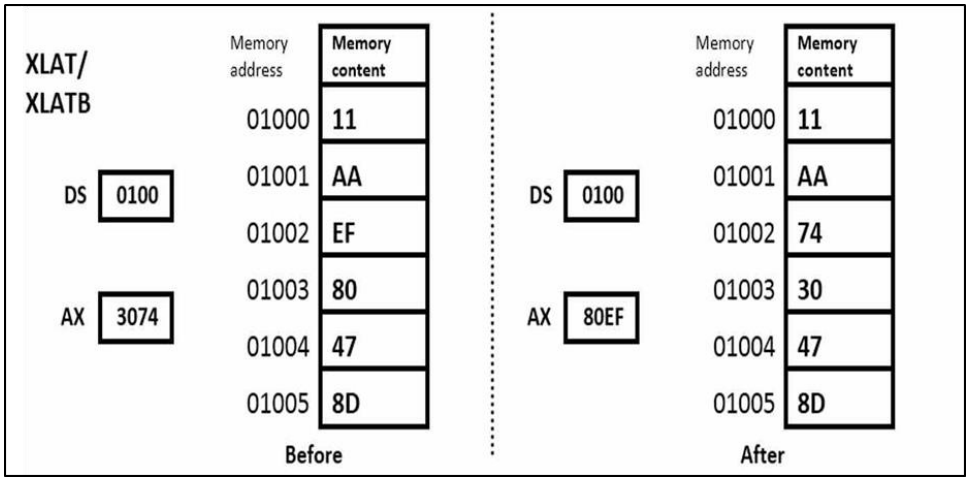
- One operand must be a register
- Both operands must be of the same size

**Examples:**

```
XCHG AX, BX
XCHG AL, BL
XCHG AX, [SI]
```

 **Exam Tip:**
`XCHG` is often used in **sorting algorithms**.

**Q/ what is the result of executing the following instruction?**
**XCHG AX, [0002]**




## 2.3 LEA Instruction (Load Effective Address)

`LEA` loads the **offset address** of a memory operand into a register.

**Syntax:**

```
LEA register, memory_operand
```

**Example:**

```
LEA SI, ARRAY
```

 Loads the **address of ARRAY**, not its content.

 **Exam Trap:**
`LEA` ≠ `MOV`

- `MOV` transfers data
- `LEA` transfers addresses

---

## 2.4 LDS and LES Instructions

These instructions load a **far pointer** (segment:offset).

**Syntax:**

```
LDS register, memory
LES register, memory
```

**Example:**

```
LDS SI, PTR
LES DI, PTR
```

 Used mainly in **advanced memory addressing**.

Example : Assuming that (BX)=100H, DI=200H, DS=1200H, SI=F002H, AX=0105H and the following memory content. What is the result of executing the following instructions?

**a. LEA SI , [ DI + BX +2H**
**SI = (DI) + (BX) + 2H= 0200H+0100H+0002H= 0302H**

**b. MOV SI , [DI+BX+2H]**
**EA=(DI+BX+2H)= 0302H**
**PA=DS*10+EA=1200*10+0302=12302**
**SI = 80EFH**

**c. LDS CX , [300]**
**PA = DS*10+EA= 1200H*10+300H = 12300H**
**CX= AA11H and DS=80EFH**

**d. LES BX , [DI+AX]**
**EA = (DI+AX)= 0200H+0105H =0305H**
**PA= DS*10+EA = 1200H*10+0305H = 12305H**
**BX = 5A8DH and ES = C592H**

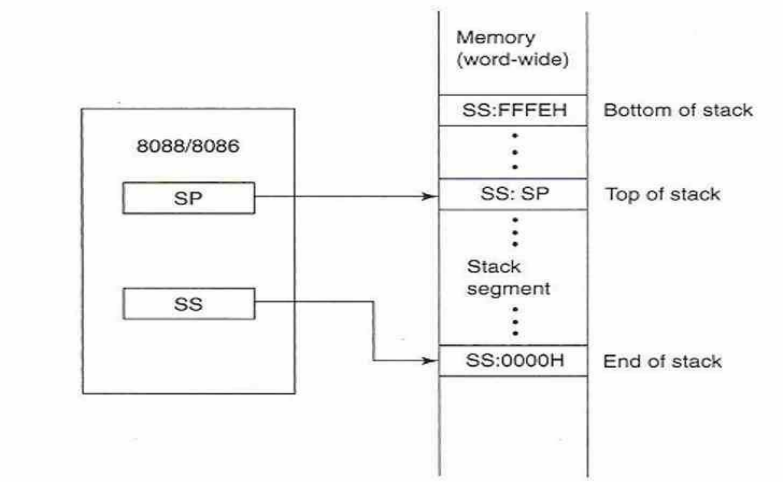
| Memory address | Memory content |
|---|---|
| 12300 | 11 |
| 12301 | AA |
| 12302 | EF |
| 12303 | 80 |
| 12304 | 47 |
| 12305 | 8D |
| 12306 | 5A |
| 12307 | 92 |
| 12308 | C5 |

# 3. Stack Operations

The stack is a special memory area that follows the **Last-In First-Out (LIFO)** principle. The stack is implemented in the memory and it is used for temporary storage of information such as data and addresses. The stack is 64k bytes long and is organized from a software point of view as 32k words

- SS register points to the lowest address word in the stack.
- SP and BP point to the address within the stack.
- Data transferred to and from the stack are word-wide, not byte-wide.
- The first address in the stack segment (SS:0000) is called End of Stack.
- The last address in the stack segment (SS: FFFE) is called Bottom of Stack.

- The address (SS:SP) is called Top of Stack. Figure below shows the stack representation




## 3.1 PUSH Instruction

Stores a word on the stack.

**Syntax:**

```
PUSH source
```

**Operation:**

- SP = SP – 2
- Data stored at SS:SP

**Examples:**

```
PUSH AX
PUSH BX
PUSH DS
```

## 3.2 POP Instruction

Retrieves a word from the stack.

**Syntax:**

```
POP destination
```

**Operation:**

- Data retrieved from SS:SP
- SP = SP + 2

**Examples:**

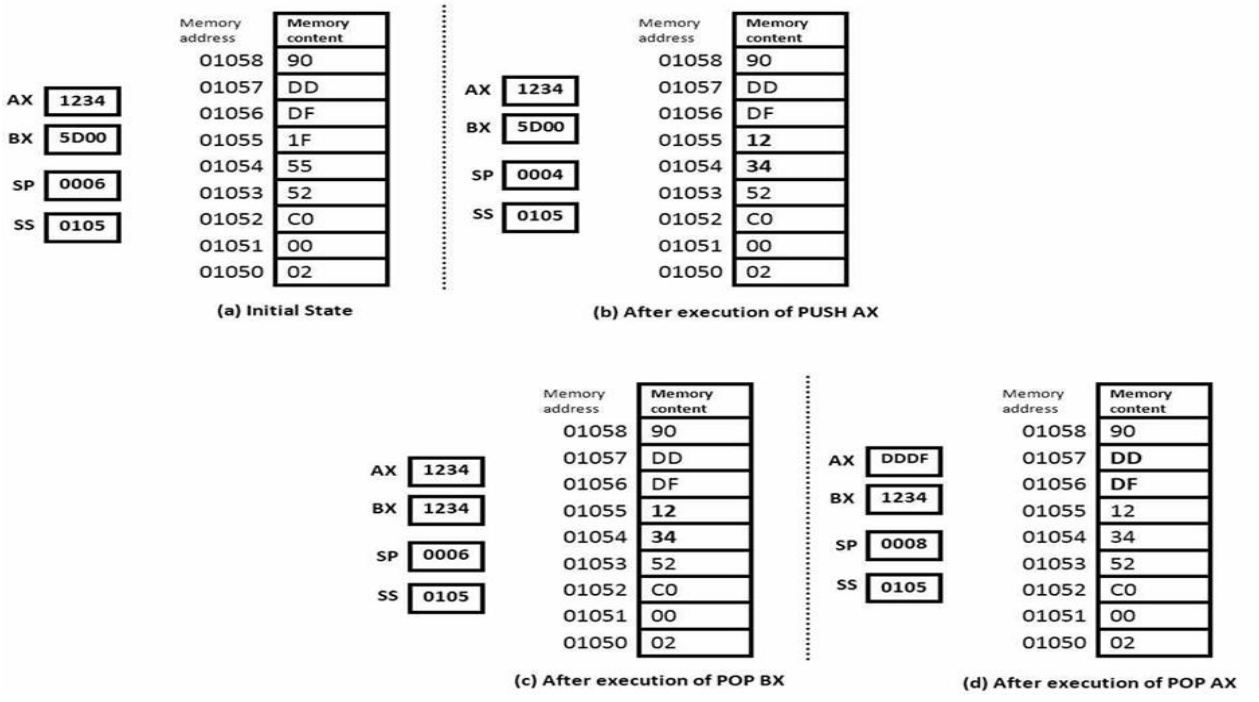
```
POP AX
POP BX
POP DS
```

 **Exam Rule:**
`POP CS` is **not allowed**

## 3.3 Stack Use Cases

- Temporary data storage
- Saving registers before procedure calls
- Parameter passing
- Handling return addresses

Example: let AX= 1234H, SS = 0105H and SP= 0006H. Below is the state of stack prior and after the execution of next program instruction
PUSH AX,
POP BX,
POP AX
Solution:



# 4. INPUT/ OUTPUT Instructions

There are two different forms of Input and Output instructions: the direct I/O instructions and variable I/O instructions. These two types of instructions can be used to transfer a byte or word of data. The data transfer takes place between the I/O device and the microprocessor unit's (MPU) accumulator register. Table 13 shows these instructions.

| Mnemonic | Meaning | Format | Operation | Flags effected |
|---|---|---|---|---|
| IN | Input direct Input variable | IN Acc, Port<br>IN Acc, DX | (Acc) ←(Port)<br>(Acc) ←(DX) | none |
| OUT | Output direct Output variable | OUT Port, Acc<br>Out DX, Acc | (Port) ←(Acc)<br>(DX) ← (Acc) | None |

**Example1:** IN AL, 0C8H ; Input a byte from port 0C8H to AL
**Example2:** IN AX, 34H ; Input a byte from port 34H to AX
**Example3:** OUT 3BH, AL ; Copy the content of AL to port 3BH
**Example4:** OUT 2CH, AX ; Copy the content of AX to port 2CH

## 5. LAHF Instruction- Load Register AH from Flags

LAHF instruction copies the values of SF, ZF, AF, PF and CF into 7,6,4,2 and 0 respectively of AH register the LAHF was provided to make conversion of assembly language programs written for 8080 and 8085 to 8986 easier.

## 6. SAHF instruction AH Register into FLAGS

SAHF instruction transfers the bits 0-7 of AH of SF, ZF, AF, PF and CF into the flag register

## 7. PUSHF Instruction- Push flag register on the stack

This instruction decrements the SP by 2 and copies the word in flag register to the memory location pointed by SP.

## 8. POPF −Pop word from top of stack to flag- register

This instruction copies a word from the two-memory location at the top of the stack to flag register and increments the stack pointer by 2.