Chapter Two: Addressing Modes

Addressing modes:

- The way of specifying data to be operated by an instruction is known as addressing modes. **In other words, addressing modes refer to the different methods of addressing the operands**.
- The 8086 memory addressing modes provide flexible access to memory, allowing you to easily access variables, arrays, records, pointers, and other complex data types.

Instructions are operations performed by the CPU. An instruction is a statement that is executed at runtime. In 8086 instruction statement can consist of four parts: ***label***, ***operation code*** (***opcode***), ***operand***, and ***comments*** as shown in this figure

Label (Optional)	Operation Code (Required)	Operand (Required in some instructions)	Comment (Optional)

Assembly instruction format

- ❖ **Labels** are used to provide symbolic names for memory addresses. A label is an identifier that can be used on a program line in order to *branch to* the labeled line. It can also be used to access data using symbolic names.
- ❖ **Operation code (opcode)** field contains the symbolic abbreviation of a given operation.
- ❖ **Operand** field consists of additional information or data that the opcode requires.
- ❖ **Comments** field provides a space for documentation to explain what has been done for the purpose of debugging and maintenance.

Operands are entities operated upon by the instruction. An 8086 instruction can have **zero to two operands.** For instructions with two operands, the first (left hand) operand is the source operand, and the second (right hand) operand is the destination operand. Also, operands are separated by commas (,) :

Example : $\text{Mov } \underbrace{\text{Ax}}_{\text{Destination}},\ \underbrace{\text{Bx}}_{\text{Source}}$

Machine code: maybe one, two, three, or four bytes in length. The first byte is an **actual operation called an opcode** (**is short for 'Operation Code'**) that tells the processor what should be done, and any other bytes that present are operand referenced an immediate value, a register, or a memory location

- There are 8 different addressing modes in 8086 programs. They are:

1. **Immediate addressing mode**
2. **Register addressing mode**
3. **Direct addressing mode**
4. **Register indirect addressing mode**
5. **Based addressing mode**
6. **Indexed addressing mode.**
7. **Based indexed addressing mode**
8. **Based, Indexed with displacement**

1. **Immediate addressing mode:** In this type of addressing, immediate data is a part of instruction and appears in the form of successive byte or bytes.

Examples:

MOV AX, 0005_H

ADD AX, 2387_H

MOV AL, FF_H

In this example, 0005H is the immediate data. The immediate data may be 8-bit or 16-bit in size.

2. **Register addressing mode:** In register addressing mode, the data is stored in a register and is referred using the particular register. All the registers, except IP, may be used in this mode.

Examples:

MOV BX, AX

ADD AL, BL

In this example, copies the contents of the 16-bit AX register into the 16-bit BX register.

3. **Direct addressing mode:** Here, the **effective address** (EA) of the memory location at which the data operand is stored is given in the instruction. The effective **address** is just a 16-bit number written directly in the instruction..

Example:

MOV AX, [5000_H]

MOV BX, [1354_H]

MOV BL, [0400_H]

The square brackets around the 1354_H denote the contents of the memory location.
When executed, this instruction will copy the contents of the memory location into BX register.

It loads or stores the data from memory to register and vice versa. The instruction consists of a register and an offset address. To compute physical address,

- shift left the DS register and
- add the offset address into it.

Example: Assume **$DS=2162_H$** , show the execution sequence of the following instruction:

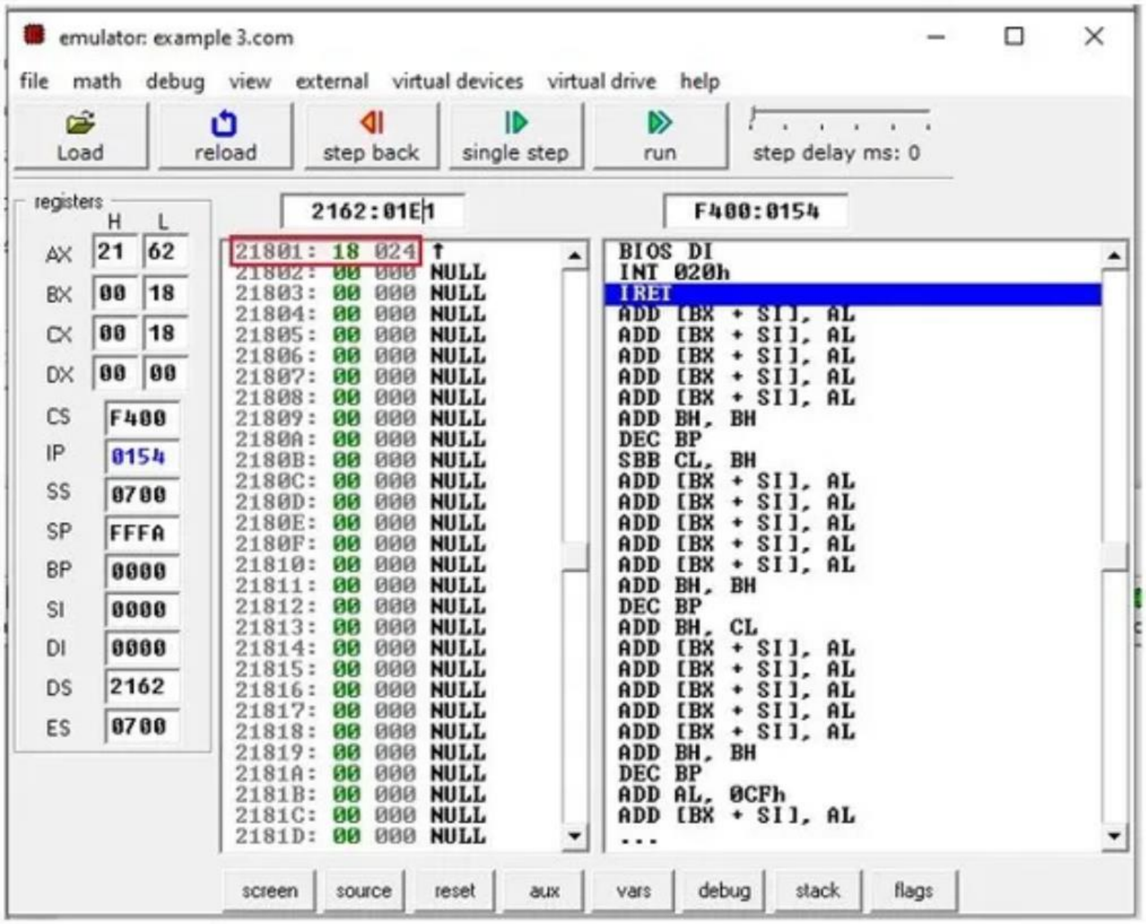
MOV CX, [481]

Sol.

- The logical address will be: **2162:01E1**
- To compute physical address, shift left the DS register and add it to offset address: The physical address will be: **$26120_H + 01E1_H=26301_H$**.
- After execution of the MOV instruction the contents of the memory location **26301_H** will be loaded into the register **CX**.

Example Code:

```
MOV AX, 2162H      ;copies hexadecimal value 2162h to AX
MOV DS, AX         ;copies value of AX into DS
MOV CX, 24         ;copies decimal value 24 into CX
MOV [481], CX      ;stores the data of CX to memory
                    address 2162:01E1
MOV BX, [481]      ;load data from memory address
                   2162:01E1 into BX
RET                ;stops the program
```


emulator: example 3.com
file math debug view external virtual devices virtual drive help
Load
reload
step back
single step
run
step delay ms: 0
registers
H L
AX 21 62
BX 00 18
CX 00 18
DX 00 00
CS F400
IP 0154
SS 0700
SP FFFA
BP 0000
SI 0000
DI 0000
DS 2162
ES 0700
2162:01E1
F400:0154
21801: 18 024 ↑
21802: 00 000 NULL
21803: 00 000 NULL
21804: 00 000 NULL
21805: 00 000 NULL
21806: 00 000 NULL
21807: 00 000 NULL
21808: 00 000 NULL
21809: 00 000 NULL
2180A: 00 000 NULL
2180B: 00 000 NULL
2180C: 00 000 NULL
2180D: 00 000 NULL
2180E: 00 000 NULL
2180F: 00 000 NULL
21810: 00 000 NULL
21811: 00 000 NULL
21812: 00 000 NULL
21813: 00 000 NULL
21814: 00 000 NULL
21815: 00 000 NULL
21816: 00 000 NULL
21817: 00 000 NULL
21818: 00 000 NULL
21819: 00 000 NULL
2181A: 00 000 NULL
2181B: 00 000 NULL
2181C: 00 000 NULL
2181D: 00 000 NULL
BIOS DI
INT 020h
IRET
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD BH, BH
DEC BP
SBB CL, BH
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD BH, BH
DEC BP
ADD BH, CL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD BH, BH
DEC BP
ADD AL, 0CFh
ADD [BX + SI], AL
...
screen source reset aux vars debug stack flags