In assembly language programming, the terms **opcode** and **operand** are fundamental concepts:

**Mnemonic**

* A mnemonic is a **human-readable abbreviation** that represents an instruction in assembly language.
* It is designed to make the code more understandable to programmers by using easily remembered names.
* Example: MOV, ADD, SUB, JMP, etc.
* Mnemonics are used in assembly language programming to symbolize machine language instructions in a more readable way.

**Opcode (Operation Code)**

* The **opcode** is the first part of an instruction that specifies the operation to be performed by the CPU. It tells the processor what action to take, such as adding numbers, moving data from one place to another, or comparing values.
* An opcode (short for **operation code**) is the **binary or hexadecimal value** that directly represents a specific mnemonic.
* The opcode is a **numeric value** that is interpreted by the CPU, while the mnemonic is a symbolic representation.
* Example: The mnemonic MOV might correspond to the opcode 0x89 in binary.
* For example, in the instruction MOV AX, BX, MOV is the opcode, indicating that the operation is to move data. More examples can be ADD, SUB, CMP, JNE etc.

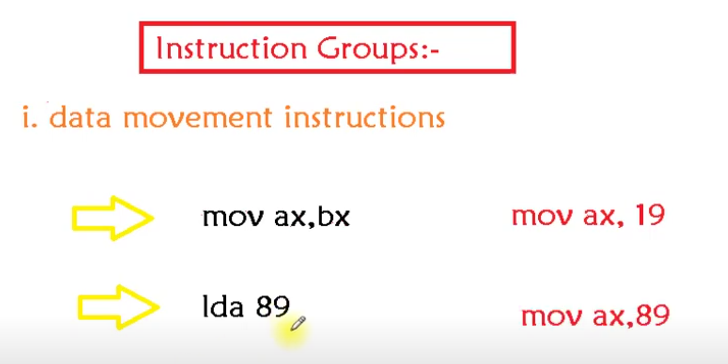
**Operand**

* The **operand** is the second part of the instruction that specifies the data to be operated on or the location where data is stored. Operands can be:
  + **Registers** (e.g., AX, BX)
  + **Memory addresses** (e.g., [1000h])
  + **Immediate values** (e.g., 5, 0xFF)
* For example, in the instruction MOV AX, BX, AX and BX are operands, indicating that the data in the BX register should be moved into the AX register. More examples can be, CX, DX, EAX, EBX, [0x0500], 8, 0x02AF.

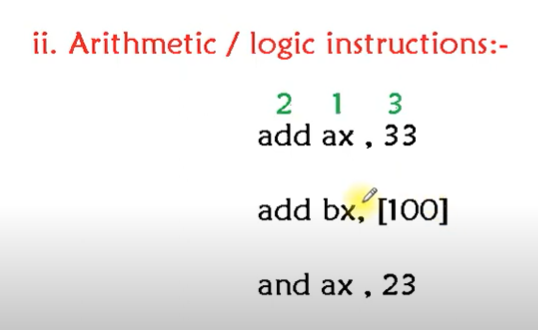
In summary, the **opcode** tells the CPU what to do, and the **operands** tell it what data to use in performing that operation.

**In the example**, MOV ax, 5; MOV is the opcode, 5 is the source operand and ax is the destination operand

**Instruction Groups**

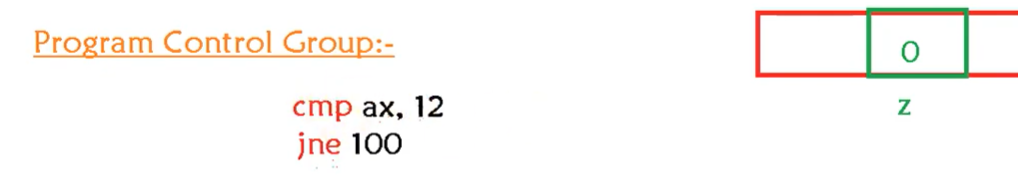


lda = Load Data in Accumulator ; also can be written as mov ax, 89 = lda 89



[100] represents a memory address, it should be written as [0100h] ; h = hexadecimal value and the square brackets shows it is an address

and = represents multiplication operation



cmp = compare ; this will subtract the immediate value 12 from ax, and if the answer is 0, it will set the zero flag to 1.

jne = Jump Not Equal ; if the Zero flag is not equal to 1 then jump to address 100



cli = Clear interrupt flag = 0

sti = Set interrupt flag = 1

If the interrupt flag is 1, then the CPU can perform other random tasks as well, but when it is set to 0, the CPU is focused on our current task and no other program can interrupt our work.