Different Groups of Instructions in Instruction Set of 8088/8086: -

Instruction Set:-

→ The instruction set of a microprocessor defines the basic operations that a programmer can make the device perform.
→ The instruction set of 8088/8086 microprocessor has 117 basic instructions.
→ The wide range of operands and addressing modes permitted for use with these instructions executable at the machine code level.
→ The instruction set can be divided into a number of groups of functionally related instructions.

1. Data Transfer Instructions.
2. Isolated I/O Instructions.
3. Arithmetic Instructions.
4. Logic Instructions.
5. Compare Instructions.
6. Shift Instructions.
7. Rotate Instructions.
8. Flag control Instructions.
9. Jump Instructions.
10. Subroutine handling Instructions.
11. Stack operations Instructions.
12. Loop handling Instructions.
13. String handling Instructions.

Data Transfer Instructions: -

→ Instructions in this group are provided to move data either between its internal registers or between an internal register and a storage location in memory.
→ This group consists of:

- → Move byte or word (MOV) Instruction.
- → Exchange byte or word (XCHG) Instruction.
- → Translate byte (XLAT) Instruction.
- → Load effective address (LEA) Instruction.
- → Load data segment (LDS) Instruction.
- → Load extra segment (LES) Instruction.

MOV Instruction:
→ MOV instruction is used to transfer a byte or a word of data from a source operand to a destination operand.

Format: MOV D , S
Operation: (S) → (D)
No flags affected.

	Destination	Source
1	Mem	Acc
2	Acc	Mem
3	Reg	Reg
4	Reg	Mem
5	Mem	Reg
6	Reg	Imm
7	Mem	Imm
8	Seg	Reg
9	Seg	Mem
10	Reg	Seg
11	Mem	Seg

XCHG Instruction:

→ XCHG instruction exchanges the contents of a register with the contents of any other register or memory location.

Format: XCHG OPERAND1 , OPERAND2
Operation: operand1 ↔ operand2.

	OPERAND1	OPERAND2
1	Reg8	Reg8
2	Reg16	Reg16
3	Reg	Mem
4	Mem	Reg

XLAT Instruction:

→ XLAT instruction converts the contents of the AL register into a number stored in a memory table.

Format: XLAT
Operation: ((DS)*10H+(BX)+(AL)) → (AL)

LEA Instruction:

→ LEA instruction loads a 16-bit register with the offset address of the data specified by the operand.

Format: LEA reg16 , mem ; (reg16) ← Offset address of mem.

LDS and LES Instructions:

→ LDS and LES instructions load any 16-bit register with an offset address and load the DS or ES register with a segment address.

Format: LDS reg16 , mem ; (reg16) ← DS*10H + mem, (DS) ← DS*10H + mem+2.
Format: LES reg16 , mem ; (reg16) ← DS*10H + mem, (ES) ← DS*10H + mem+2.

Isolated I/O Instructions: -
→ IN and OUT instructions transfer data between the AL, AX registers of microprocessor and I/O devices.

Format: IN acc , port ; (acc) ← data present at port.

Format: OUT port , acc ; data appear at port ← (acc).

	Port	Accumulator
1	P8	AL
2	P8	AX
3	DX	AL
4	DX	AX

Arithmetic Group of Instructions: -
→ Instructions in this group consist of addition, subtraction, multiplication and division.

ADD/SUB Instruction:
→ Add/subtract byte or word.

Format: ADD/SUB OPERAND1 , OPERAND2
Operation: (OPERAND1) ← (OPERAND1) ± (OPERAND2).

ADC/SBB Instruction:
→ Add/subtract byte or word with carry.

Format: ADC/SBB OPERAND1 , OPERAND2
Operation: (OPERAND1) ← (OPERAND1) ± (OPERAND2) ± CARRY.

INC/DEC Instruction:
→ Increment/decrement byte or word.

Format: INC/DEC OPERAND
Operation: (OPERAND) ← (OPERAND) ± 1.

NEG Instruction:
→ 2's compliment byte or word.

Format: NEG OPERAND
Operation: (OPERAND) ← 0 - (OPERAND).

Flag affected: All status flags.

	OPERAND1	OPERAND2
1	Reg	Reg
2	Reg	Mem
3	Mem	Reg
4	Reg	Imm
5	Mem	Imm
6	Acc	Imm

OPERAND
Reg16
Reg8
Mem

MUL Instruction:
→ Multiply unsigned numbers

Format: MUL S
Operation: (AX) ← (AL) * (S8)
Operation: (DX-AX) ← (AX) * (S16)

→ After MUL
→ CF/OF = 0, If the upper half of the result is zero.
→ CF/OF = 1, otherwise.

IMUL Instruction:
→ Multiply signed numbers

Format: IMUL S
Operation: (AX) ← (AL) * (S8)
Operation: (DX-AX) ← (AX) * (S16)

→ After IMUL
→ CF/OF = 0, If the upper half of the result is the sign extension of the lower half (this means that the bits of the upper half are the same as the sign bit of the lower half)
→ CF/OF = 1, otherwise.

Flags affected: C, O.
Undefined: S, Z, A, P.

DIV Instruction:
→ Divide unsigned numbers

Format: DIV S
Operation: (AL) ← Q((AX)/(S8))
(AH) ← R((AX)/(S8))
Operation: (AX) ← Q((DX-AX)/(S16))
(DX) ← R((DX-AX)/(S16))

Format: IDIV S
Operation: (AL) ← Q((AX)/(S8))
(AH) ← R((AX)/(S8))
Operation: (AX) ← Q((DX-AX)/(S16))
(DX) ← R((DX-AX)/(S16))

Flags affected: None.
Undefined: O, S, Z, A, P, C.

SOURCE
Reg8
Reg16
Mem8
Mem16

Examples:
→ (AX) = 0001h, (BX) = FFFFh.
After MUL BX
(DX) = 0000, (AX) = FFFFh, CF/OF = 0.
After IMUL BX
(DX) = FFFFh, (AX) = FFFFh, CF/OF = 0.

→ (AX) = FFFFh, (BX) = FFFFh.
After MUL BX
(DX) = FFFE, (AX) = 0001h, CF/OF = 1.
After IMUL BX
(DX) = 0000, (AX) = 0001h, CF/OF = 0.

→ (AX) = 0FFFh, (BX) = 0FFFh.
After MUL BX
(DX) = 00FFh, (AX) = E001h, CF/OF = 1.
After IMUL BX
(DX) = 00FFh, (AX) = E001h, CF/OF = 1.

→ (AX) = 0100h, (BX) = FFFFh.
After MUL BX
(DX) = 00FFh, (AX) = FF00h, CF/OF = 1.
After IMUL BX
(DX) = FFFFh, (AX) = FF00h, CF/OF = 0.

→ (AL) = 80h, (BL) = FFh.
After MUL BL
(AH) = 7Fh, (AL) = 80h, CF/OF = 1.
After IMUL BL
(AH) = 00h, (AL) = 80h, CF/OF = 1.

BCD and ASCII Arithmetic:

Packed BCD:
→ Following instructions are needed to do arithmetic on packed BCD numbers.

DAA Instruction:
Purpose:
→ This instruction corrects the result in AL of adding two packed BCD operands.
Format: DAA
Operation:
→ If the low nibble of AL is greater than 9h or if AF is set, then a 6 is added to AL, and AF is set to 1.
→ If AL is greater than 9Fh or if the CF is set, then 60h is added to AL and CF is set to 1.

DAS Instruction:
Purpose:
→ This instruction corrects the result in AL of subtracting two packed BCD operands.
Format: DAS
Operation:
→ If the low nibble of AL is greater than 9 or if AF is set, then 66h is subtracted from AL, and CF is set to 1.
Flags affected: A,C, P, S, Z.

Unpacked BCD:
→ Following instructions are needed to do arithmetic on unpacked BCD numbers.

AAA Instruction:
Purpose:
→ This instruction corrects the result in AL of adding two unpacked BCD digits or two ASCII digits.
Format: AAA
Operation:
→ If the low nibble of AL is greater than 9 or the AF is set, then a 6 is added to AL, the high nibble of AL is cleared, and a 1 is added to AH.
→ Both AF and CF are set if the adjustment is made. Other flags are undefined.

AAS Instruction:
Purpose:
→ This instruction corrects the result in AL of subtracting two unpacked BCD digits or two ASCII digits.
Format: AAS
Operation:
→ If the low nibble of AL is greater than 9 or the AF is set, then a 6 is subtracted from AL, the high nibble of AL is cleared, and a 1 is subtracted from AH.
→ Both AF and CF are set if the adjustment is made. Other flags are undefined.

AAM Instruction:

Purpose:

→ This instruction converts the result of multiplying two BCD digits into unpacked BCD format. It can be used in converting numbers lower than 100 into unpacked BCD format.

Format: AAM

Operation:

→ The contents of AL are converted into two unpacked BCD digits and placed in AX.

→ It divides the contents of AL by 10.

→ The quotient, corresponding to the ten's digit is placed in AH.

→ The remainder, corresponding to the unit's is placed in AL.

AAD Instruction:

Purpose:

→ Adjust the unpacked BCD dividend in AX in preparation for division.

Format: AAD

Operation:

→ The unpacked BCD operand in AX is converted into binary for division.

→ It multiplies AH by 10.

→ Adds the product to AL, and then clears AH.

Comparison Instructions: -

→ Compares the operands by subtracting them. Result is not stored anywhere only the flag bits change.

Format: CMP OPERAND1 , OPERAND2

Operation: (OPERAND1) - (OPERAND2) affects the status flags.

	OPERAND1	**OPERAND2**
1	Reg	Reg
2	Reg	Mem
3	Mem	Reg
4	Reg	Imm
5	Mem	Imm
6	Acc	Imm

Logic Group of Instructions: -
→ Instructions in this group performs AND, OR, XOR and NOT logic operations.

AND Instruction:
→ AND byte or word.

Format: AND OPERAND1 , OPERAND2
Operation: (OPERAND1) ← (OPERAND1) . (OPERAND2).

OR Instruction:
→ OR byte or word.

Format: OR OPERAND1 , OPERAND2
Operation: (OPERAND1) ← (OPERAND1) + (OPERAND2).

XOR Instruction:
→ XOR byte or word.

Format: XOR OPERAND1 , OPERAND2
Operation: (OPERAND1) ← (OPERAND1) $\oplus$ (OPERAND2).

NOT Instruction:
→ Invert byte or word.

Format: NOT OPERAND
Operation: (OPERAND) ← $\overline{(OPERAND)}$.

Flags affected: O, S, Z, P, C.
Undefined: A.

	OPERAND1	**OPERAND2**
1	Reg	Reg
2	Reg	Mem
3	Mem	Reg
4	Reg	Imm
5	Mem	Imm
6	Acc	Imm

OPERAND
Reg16
Reg8
Mem

Shift and Rotate Instructions: -

→ Shift and Rotate instructions manipulate binary numbers at the binary bit level.

→ Shift instructions are:

- → Shift Logical Left (SHL).
- → Shift Arithmetic Left (SAL).
- → Shift Logical Right (SHR).
- → Shift Arithmetic Right (SAR).

Format: SAR/SHR/SAL/SHL D , COUNT.

→ Rotate instructions are:

- → Rotate Left (ROL).
- → Rotate Right (ROR).
- → Rotate Left through carry (RCL).
- → Rotate Right through carry (RCR).

Format: RCR/RCL/ROR/ROL D , COUNT.

	DESTINATION	**COUNT**
1	Reg	1
2	Reg	1
3	Mem	CL
4	Mem	CL

Flag Control Instructions: -

→ Affects carry, direction, and interrupt flags.

Format: CLC; Clear Carry Flag
Operation: (CF) ← 0.

Format: CLD; Clear Direction Flag
Operation: (DF) ← 0.

Format: CLI; Clear Interrupt Flag.
Operation: (IF) ← 0.

Format: CMC; Complement Carry Flag.
Operation: (CF) ← (CF).

Format: STC; Set Carry Flag
Operation: (CF) ← 1.

Format: STD; Set Direction Flag
Operation: (DF) ← 1.

Format: STI; Set Interrupt Flag.
Operation: (IF) ← 1.

Jump Instructions: -

→ These instructions allow the programmer to skip sections of a program and branch to any part of the memory for the next instruction.

→ There are two types of jump instructions:

→ Unconditional Jumps (JMP).

→ Conditional Jumps (Jcond).

Unconditional Jumps (JMP): -

Format: JMP OPERAND;

→ Three types of unconditional jump instructions are available to the microprocessor.

→ Short jump

→ Near jump

→ Far jump

OPERAND
Short label
Near label
Far label
Mem ptr
Reg16

e.g.

JMP [06h];	(IP) ← (IP) + 06h.
JMP [0340h];	(IP) ← (IP) + 0340h.
JMP 2040:3050;	(IP) ← 3050h, (CS) ← 2040h.
JMP [BX];	(IP) ← [DS: (BX)]
JMP CX;	(IP) ← (CX)
JMP NEAR PTR[DI];	(IP) ← [DS: (DI)]
JMP FAR PTR[BP];	(IP) ← [SS: (BP)], (CS) ← [SS: (BP)+2]

Conditional Jumps (Jcond): -

Format: Jcond OPERAND

OPERAND: short label.

→ Conditional jumps are always short jumps in the 8086/8088 microprocessors. So, the limits of the range of jumps are within +7Fh bytes and -7Fh bytes from the location following the conditional jump.

→ S, Z, C, P, and O flags are used to test the condition and execute the jump.

→ Operation of most conditional jump instructions is straight forward because they often test just one flag bit, although some test more than one.

→ The term greater and lesser than refers to signed numbers jump instructions. Z, S and O flags are checked before executing jump.

→ The term above and below refers to unsigned numbers jump instructions. Z and C flags are checked before executing jump.

Subroutine Handling (CALL and RET) Instructions: -

→ CALL and RET instructions together provide the mechanism for calling a subroutine into operation and returning control back to the main program at its completion.

→ CALL instruction allows implementation of two types of operations:

→ Intrasegment call.

→ Intersegment call.

Format: CALL OPERAND

OPERAND
Near proc
Far proc
Mem ptr
Reg16

e.g.

CALL [1040h];	PUSH IP, (IP) ← (IP)+1040h
CALL 4060:0280;	PUSH CS, PUSH IP, (IP) ← 0280h, (CS) ← 4060h
CALL BX	PUSH IP, (IP) ← (BX)
CALL NEAR PTR[BX]	PUSH IP, (IP) ← [DS: (BX)]
CALL FAR PTR[SI+08h]	PUSH CS, PUSH IP, (IP)←[DS: (SI)+08], (CS)←[DS: (SI)+08]

Format: RET OPERAND

OPERAND
None
Disp16

e.g.

RET (near);	POP IP
RET (far);	POP IP, POP CS
RET 10h (near);	POP IP, (SP) ← (SP)+10h
RET 10h (far);	POP IP, POP CS, (SP) ← (SP)+10h

LOOP Instructions: -

→ Loop instructions are used to repeatedly execute a piece of code/instructions.

Format: LOOP LABEL; Loop until CX = 0.

Format: LOOPE/LOOPZ LABEL; Loop while ZF = 1.

Format: LOOPNE/LOOPNZ LABEL; Loop while ZF = 0.

LABEL : short label.

Stack Operations Instructions: -

→ PUSH and POP instructions store data onto and retrieve data from the LIFO stack memory.

Format: PUSH OPERAND

Format: POP OPERAND

OPERAND
Reg
Mem
Seg
Imm(not for POP & not available in 8088/8086)
F

e.g.

PUSH AX; ((SP)-1)←(AH), ((SP)-2)←(AL), (SP)←(SP)-2.
PUSH [BX]; ((SP)-1)←[DS:(BX)+1], ((SP)-2)←[DS:(BX)], (SP)←(SP)-2.
PUSH DS; ((SP)-1)←(DS_H), ((SP)-2)←(DS_L), (SP)←(SP)-2.
PUSH 80C0h; ((SP)-1)←80h, ((SP)-2)←C0h, (SP)←(SP)-2.
PUSHF; ((SP)-1)←(FL_H), ((SP)-2)←(FL_L), (SP)←(SP)-2.
POP DX; (DL)←((SP)), (DH)←((SP)+1), (SP)←(SP)+2.
POP [BP]; [SS:(BP)]←((SP)), [SS:(BP)+1]←((SP)+1), (SP)←(SP)+2.
POPF; (FL_L)←((SP)), (FL_H)←((SP)+1), (SP)←(SP)+2.
POP ES; (ES_L)←((SP)), (ES_H)←((SP)+1), (SP)←(SP)+2.

String Handling Instructions: -

→ These instructions are used to deal with string data.

Format: LODSB
Operation: (AL) ← [DS: (SI)], (SI) ← (SI)±1.

Format: LODSW
Operation: (AX) ← [DS: (SI)], (SI) ← (SI)±2

Format: STOSB
Operation: [ES: (DI)] ← (AL), (DI) ← (DI)±1.

Format: STOSW
Operation: [ES: (DI)] ← (AX), (DI) ← (DI)±2.

Format: MOVSB
Operation: [ES: (DI)] ← [DS: (SI)], (SI) ← (SI)±1, (DI) ← (DI)±1.

Format: MOVSW
Operation: [ES: (DI)] ← [DS: (SI)], (SI) ← (SI)±2, (DI) ← (DI)±2.

Format: INSB
Operation: [ES: (DI)] ← [DX], (DI) ← (DI)±1.

Format: INSW
Operation: [ES: (DI)] ← [DX], (DI) ← (DI)±2.
.
Format: OUTSB
Operation: [DX] ← [DS: (SI)] , (SI) ← (SI)±1.

Format: OUTSW
Operation: [DX] ← [DS: (SI)] , (SI) ← (SI)±2.

---------- * ----------

Subroutine:

→ In some program structures, one part of the program is called the main program.
→ In addition to this, we find a segment attached to the main program, known as a subroutine.
→ A subroutine is a special segment of program that can be called for execution from any point in a program.
→ The subroutine is written to provide a function that must be performed at various points in the main program.
→ Instead of including this piece of code in the main program each time the function is needed, it is put into the program just once as a subroutine.
→ Whenever the function is needed in the main program, a single instruction is inserted into the code.

MACRO:

→ A MACRO is a group of instructions that perform one task, just as a procedure performs one task. The difference is that a procedure is accessed via a CALL instruction, while a MACRO is inserted in the program at the point of usage as a new sequence of instructions.
→ Creating a MACRO is very similar to creating a new opcode that can be used in the program.
→ MACRO sequences execute faster than procedures because there are no call and ret instructions to execute.
→ The MACRO's instructions are placed in your program by the assembler at the point they are invoked/referenced.
→ The MACRO and ENDM directives are used to delineate a MACRO sequence.
→ The first statement of a MACRO is the MACRO instruction, which contains the name of the MACRO and any parameters associated with it.
e.g.

```
MOVE    MACRO    A, B
        PUSH  AX
        MOV  AX, B
        MOV  A, AX
        POP  AX
        ENDM
```

Here,
MACRO name: MOVE (Opcode)
Parameters: A and B.
MACRO termination: ENDM.