8086 Instruction sets :-

Instruction set :- different types of instructions in 8086 are — i) data transfer instruction
ii) Arithmetic Instructions
iii) Logical instructions.
iv) shift and rotation instructions.
v) Branch instructions.
vi) flag manipulation and processor control Instructions.
vii) string instructions.

(i) data transfer instructions :- All instructions which performs data movements, comes here.

general instruction format

opcode	Address

some of them are :-

(a) MOV → moves data b/w register to register, register to memory, memory to register

(b) LDS → loads word → from given memory locations to register and also loads word from next 2 memory locations to DR or DS.

(c) LES → loads word → from specific memory locations into given register, also loads a word from next 2 memory locations into ES register.

(d) LEA → loads offset address into given register.

(g) LAHF → loads low order 8 bits of flag register to AH register.

(h) SAHF → stores content of AH register into low order bits of flag register.

(i) XCHG → exchanges contents of 16 bit (or) 8 bit given register with contents of AX register, specific register (or) memory location.

(j) PUSH → pushes content of register (or) memory to top of stack.

(k) POP → reads data from top of stack and stores in register (or) memory.

(l) IN → transfers data from port to register.

(m) OUT → register to output ports.

(II) Arithmetic instructions :- instructions containing arithmetic calculations comes here,

⊙ ADD → Adds data to AC, i.e. AL or AX reg. or memory.
⊙ ADC → adds operands with carry (previous stage).
⊙ SUB → subtract data from AC, memory or register.
⊙ SBB → subtract data with borrow from AC to register / memory.

⊙ MUL → unsigned 8 bit or 16 bit multiplication.
⊙ IMUL → signed 8 (or) 16 bit "

- DIV → unsigned 8/16 bit division.
- IDIV → signed 8/16 " "
- INC → increment Register/memory by 1.
- DEC → decrement " " " .

(iii) logical instructions :- instructions which has logical symbols.

- AND → performs AND of 2 operands.
- OR → " OR " " "
- XOR → " XOR " " "
- NOT → " complement.

shift / Rotate instructions :-

- SAL/SHL : are 2 mnemonics, these shifts each bit in specified destination to left and 0 is stored at LSB, number of shifts indicated by count.

eg :- Before 10101, after = 01010

ii) SHR : right shift, 0 at MSB, number of shifts indicated by count.

eg :- 10101 ⇒ 0 1010.

iii) SAR : right shift, arithmetic shift, new MSB = old MSB.

eg :- 10101 ⇒ 1 1010

- ROL :- rotates left, i.e. MSB to LSB and to carry flag.

eg :- 10101 ⇒ 01011 → CF = 1

- ROR → rotates to right, LSB to MSB and to carry flag.

eg :- 10101 ⇒ 11010.

- RCR ⇒ right rotation, LSB to CF and CF to MSB.
- RCL ⇒ left rotation, MSB to CF, CF to LSB.

Branch instructions :- loops also :-

- CALL → call a procedure and save return address to stack.
- RET → return from procedure to main program.
- JMP → jump to provided address to proceed to next instruction.
- JA/JNBE → jump if above/not below/equal instruction address.
- JAE/JNB :- jump if above instruction address.
- JC → jump if carry CF = 1.
- JE/JZ → jump if zero flag ZF = 1.
- JG/JNLE → jump if greater/not less than/equal instruction address.
- JGE/JNL → jump if greater than/but less than instruction true.

flag manipulation and process control instructions :-

- CLC → clear carry flag, reset CF = 0.
- CLD → clear Direction flag ⇒ DF = 0.
- CLI → clear interrupt flag ⇒ IF = 0.

- CMC ⇒ complement of carry flag CF.
- STC ⇒ set carry flag, CF = 1.
- STD ⇒ set direction flag.
- STI ⇒ set interrupt flag, IF = 1.
- HLT ⇒ Halt processing, stops program execution.
- NOP ⇒ performs no operation.
- ESC ⇒ Escape, makes bus free for external master like a coprocessor or peripheral devices.
- WAIT ⇒ processor enters an idle state in which processor does no processing.
- LOCK : It is prefix instruction, it makes lock pin low till execution of next instruction.

String Instructions :- 8086 provides some instructions which handle string operations such as string movement, comparison, scan, load and store.

- MOVS/MOVSB/MOVSW :- moves 8/16 bit data from memory location addressed by SI register to memory location addressed by DI register.
- CMPS/CMPSB/CMPSW ⇒ compare content of memory location addressed by DI register with content of memory location addressed by SI register,

① SCAS/ SCASB/ SCASW ⇒ compare content of accumulator with content of memory addressed by DI register in extra segment ES.

② LODS/ LODSB/ LODSW ⇒ loads 8/16 bit data from memory addres by SI register into AL or AX register.

③ STOS/ STOSB/ STOSW ⇒ stores 8/16 bit data from AL or AX register to addressed by DI register.

④ REP ⇒ Repeats given instruction until CX ≠ 0.

⑤ REPE/ REPZ ⇒ repeats given instruction till CX ≠ 0 and ZF = 1.

⑥ REPNE/ REPNZ ⇒ repeats instruction till CX ≠ 0 and ZF = 0.

Assembler directives (8086)

- special codes placed in the program to instruct assembler to perform a particular task of a function.
- they can be used to define symbol values, reserve & initialize storage space for variables & control the placement of program code.

(1) **Assume** : this is used to inform the assembler the name of logical segment it should use for a specific segment.

eg: ASSUME DS : DATA, tells the assembler, any program instruction which refers to the data segment, it should use the logical segment call DATA. (from now onwards DATA means DS, instead of DS we can write DATA).

(2) **DB** : Define byte. It reserves 1 byte in memory, which are declared as DB.

(3) **DW** : (Define word) → reserves word length memory space.

(4) **DD** define double word → ~~double~~ directive is used to declare a variable of type double word (or) restore memory locations which can be accessed as type double word.

(5) **DQ** (define quad word) → directive tells assembler to declare a variable 4 words in length (or) to reserve 4 words of storage in memory.

⑧ DT (Define ten bytes) :- It is used to inform assembler to define variable which is 10 bytes in length (or) reserve 10 bytes of storage in memory

⑦ EQU - Equate :- It is used to give a name to some value (or) symbol. From now onwards everytime when assembler finds that name, it replaces name with value or symbol we have equated with that name

⑥ ORG : Origin :- The ORG statement changes starting offset address of data. (starting address is represented).

⑨ PROC :- Procedure :- is used to identify the start of a procedure (or) subroutine.

⑩ END :- End program. Assembler ignores any statements after END directive.

⑪ ENDS :- End segment.

Eg :- CODE SEGMENT : start of logical segment containing code.

CODE ENDS : end of CODE segments.