- For addressing modes DS may be overridden by CS, SS, or ES; and when BP is used, SS may be overridden by CS, DS, or ES. Specific cases that cannot involve overrides are as follows :

1) The CS register is always used as the segment register when computing the address of the next instruction to be executed.
2) For stack pointer SP, SS is the segment register.
3) For string operation ES is by default segment register for destination operand.

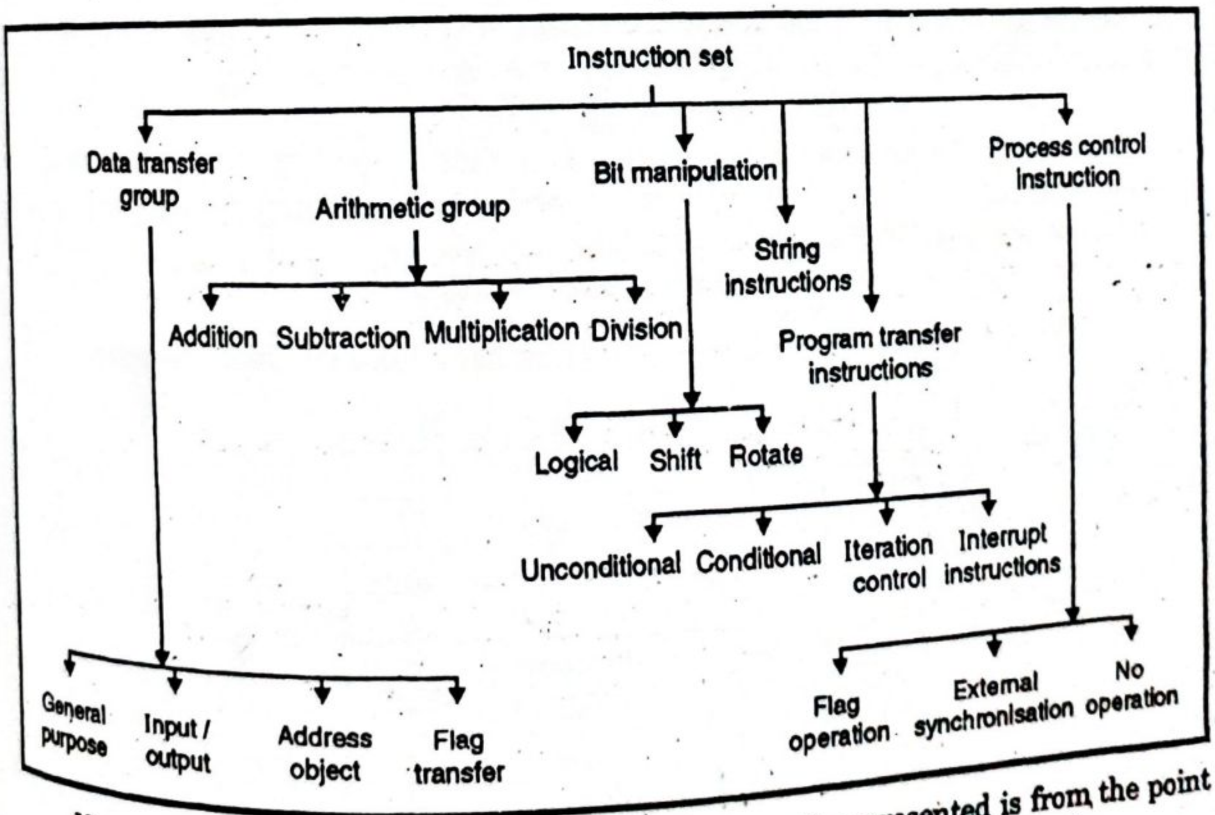
12.5 Instruction Set of 8086

The instruction set of 8086/8088 is divided into number of groups, of functionally related instructions.

Different groups are :

- Data transfer group.
- Bit manipulation group.
- Program transfer instruction group.
- Arithmetic group.
- String instruction group.
- Process control instruction group.

Graphical presentation of different groups is as shown.



Now we will start with instruction set. The information presented is from the point of view of utility to the assembly language programmer. The information given is :

1) *Mnemonic (Syntax of the instruction)*
2) *Algorithm*
3) *Operation of the instruction*
4) *Examples.*

- While giving you above information some typical symbols/labels are used. I feel that you should know the significance and meaning of those labels.

12.6 Data Transfer Group

59

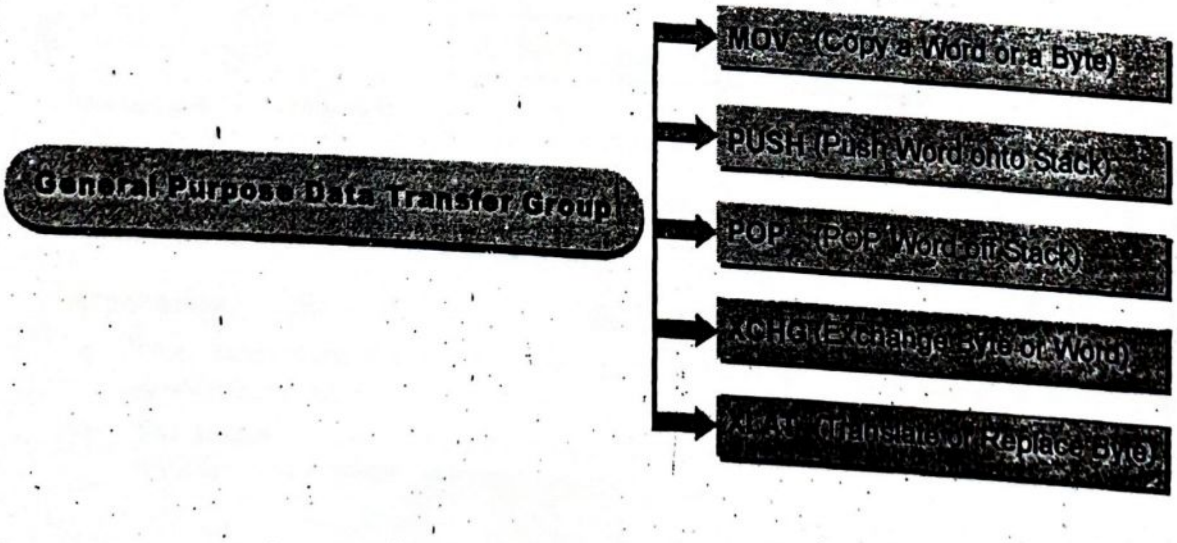
The 14 data transfer instructions are listed as follows :

Table 12.6.1 : Data transfer instructions

General Purpose		Address Object	
MOV	Move byte or word	LEA	Load effective address
PUSH	Push word onto stack	LDS	Load pointer using DS
POP	Pop word off stack	LES	Load pointer using ES
XCHG	Exchange byte or word		
XLAT	Translate byte		
Input/Output		**Flag Transfer**	
IN	Input byte or word	LAHF	Load AH register from flags
OUT	Output byte or word	SAHF	Store AH register in flags
		PUSHF	Push flags onto stack
		POPF	Pop flags off stack

These instructions move single bytes and words between memory and register as well as between register AL or AX and I/O ports. The stack manipulation instructions are included in the group as are instructions for transferring flag contents and for loading segment registers. Now let's start with one by one subgroup and study each instruction carefully.

12.6.1 General Purpose Data Transfer Group


General Purpose Data Transfer Group
MOV (Copy a Word or a Byte)
PUSH (Push Word onto Stack)
POP (POP Word off Stack)
XCHG (Exchange Byte or Word)
XLAT (Translate or Replace Byte)

1. MOV – Copy a Word or a Byte

Flags No flags are affected.

Mnemonic MOV destination, source

MOV operand1, operand2

Destination ⟵ Source

Operand 1 ⟵ Operand 2

Algorithm Destination = Source or Operand 1 = Operand 2

Addr. Mode Register addressing mode

Operation

- The MOV instruction copies a word or a byte of data from a fixed/specified source to a fixed/specified destination.

Examples

1. MOV [SI], AL
i.e. MOV [SI] , AL

This instruction copies the contents of the AL register to memory location whose offset is stored in SI register.

2. MOV AX, Temp_Result

The contents of memory location Temp_result will be transferred/copied to the AL register. Then the IP will increment by 1 and contents of location after Temp_Result will be copied to the AH register.

3. MOV AX, BX

This instruction copies the contents of BX register to AX register. The LSB of BX i.e. BL is copied to AL and MSB of BX i.e. BH is copied to AH.

4. MOV COUNT [DI],2DH
i.e. MOV COUNT [DI],2DH

This instruction copies immediate number 2DH to the required memory location. EA of the memory location is the sum of displacement COUNT and the contents of DI (EA = COUNT + DI)

Following table contains valid source and destination operands.

Sr. No.	Destination	Source
1.	Memory	Accumulator
2.	Accumulator	Memory
3.	Register	Register
4.	Register	Memory
5.	Memory	Register.
6.	Register	Immediate
7.	Memory	Immediate
8.	Seg – Reg	Reg – 16
9.	Seg – Reg	Mem – 16
10.	Reg – 16	Seg – Reg
11.	Memory	Seg – Reg

60

Following rules are observed while executing the instruction

The Source & Destination in an Instruction both **CANNOT** be Memory Locations	Incorrect ➡	MOV [1100], [1200] ↑ ↑ Memory Location 1 Memory Location 2
The Destination in an Instruction **CANNOT** be Immediate Number	Incorrect ➡	MOV 592F H, BX ↑ Immediate number
The Destination in an Instruction **CANNOT** be Code Segment Register CS incorrect	Incorrect ➡	MOV 592F H, CS ↑ Immediate number
The Source & Destination must both be of a type **BYTE**, or they must be of a type **WORD** Such a data transfer is not possible because BL is 8 Bit & AX is 16 Bit	Incorrect ➡	MOV AX, BL ↑ ↑ 16 bit (WORD) 8 bit (BYTE)
It **CANNOT** copy value of one segment Register to another segment Register (One should copy to general register first)	Incorrect ➡	MOV DS, CS ↑ ↑ Data Seg. Code Seg.
It **CANNOT** copy immediate value to segment Register.	Incorrect ➡	MOV CS, 5487 H ↑ ↑ Code Seg. Immediate number
It **CANNOT** set the value of CS & IP Registers		

Fig. 12.6.1

2. PUSH - Push Word onto Stack

Mnemonic PUSH Source **Flags** No flags are affected.

Algorithm SP = SP – 2

SS : [SP] (Top of the Stack) = Operand

Addr. Mode Register addressing mode

Operation SP → SP – 2 SS → data from specified source

- This instruction decrements the stack pointer by 2 and copies a word from a specified source to the location in the stack segment where stack pointer points.
- The source of operand (16 bit data to be stored on stack) can be a general purpose register, flag register, segment register or memory.

- The stack segment register and stack pointer must be initialised before using this instruction.
- PUSH can be used to save data on the stack so that it will not be destroyed by a execution of successive instructions.

Example PUSH AX

Now we will see detailed analysis of what exactly happens when instruction PUSH AX executes.

PUSH AX

SP → SP – 2

SS → Copy word from AX register to location in Stack segment (SS) where SP points.

3. POP – Pop Word off Stack

Mnemonic POP Destination OR POP Operand **Flags** No flags are affected

Algorithm Operand = SS : [SP] (top of stack) SP = SP + 2

Addr. Mode Register Addressing mode

Operation SS → Data copied to destination or operand SS → SS : [SP + 2]

- This instruction copies a word (two successive memory location contents) from stack segment in memory to a destination specified in the instruction.
- The destination can be a general purpose register, flag register, a segment register or a memory location.
- The data in the stack pointer is not changed.
- After the word is copied to specified destination the stack pointer is automatically incremented by 2 to point to next word on stack.

Example POP AX

This instruction copies a word (two successive memory location contents) from stack segment in memory to the AX register.

4. XCHG – Exchange byte or word

Mnemonic XCHG destination, source. **Flags** No flags are affected

Algorithm destination = source

Addr. Mode Implied addressing mode

Operation destination ↔ source

- This instruction exchanges the contents of a register with contents of another register or the contents of a register with contents of memory location.
- The register can be 8/16 bit.
- XCHG cannot directly exchange the contents of two memory locations.
  e.g. XCHG [1234],[5068] ; This transfer is not possible
- The source and destination must both be words or they both must be bytes.

Example XCHG AX, BX i.e. AX ↔ BX

- This instruction will exchange the word in AX register with the word in BX register.
- Contents of AL ↔ Contents of BL, Contents of AH ↔ Contents of BH

5. XLAT – Translate or Replace Byte

Mnemonic XLAT/XLATB [B indicates byte operation] (meaning of XLAT and XLATB is same, either XLAT or XLATB can be written)

Flags No flags are affected.

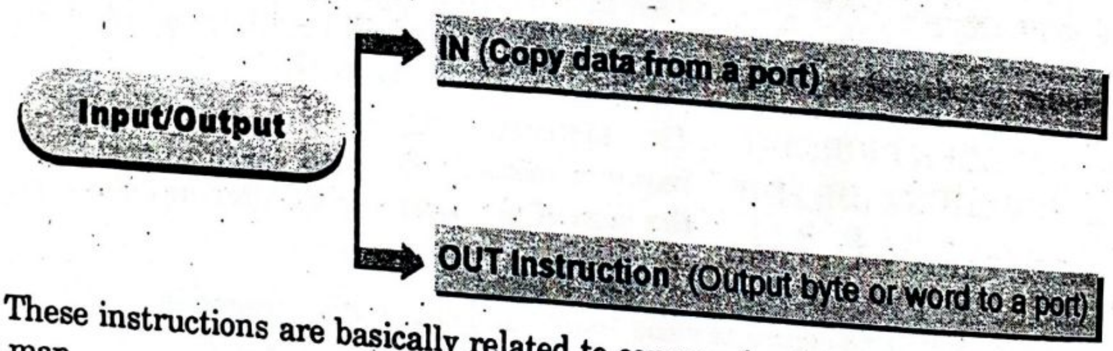
Algorithm AL = DS : [BX + unsigned AL]

Addr. Mode Implied Addressing mode

Operation AL ← DS : [BX + AL]

- This instruction replaces a byte in AL register with a byte from a look up table in the memory. i.e. it copies the value of memory byte at location DS : [BX + unsigned AL] to the AL register.
- Here contents of AL before execution acts as index to the desired location in lookup table.
- This instruction is used to translate a byte from one code to another code.
- Mostly this concept is used to convert BCD to seven segment code or ASCII to EDCBIC code conversion

12.6.2 Input/Output


Input/Output
IN (Copy data from a port)
OUT Instruction (Output byte or word to a port)

These instructions are basically related to communication with I/O devices, mapped in I/O map.

1. IN – Copy data from a port

Mnemonic IN accumulator, port address.

Algorithm —

Flags No flags are affected

Addr. Mode Direct port addressing mode

Operation AX ← Contents of port **or** AL ← contents of port.

This instruction will copy data from a port whose address is given in the instruction to AX register or AL register. The data can be either 8 bit or 16 bit.

Example IN AL, C8H.

This instruction will copy the contents of port whose address is C8H to the AL register.

- The IN instruction has two possible formats

- Fixed port
- Variable port.

- For the **Fixed port**, the port address is specified directly in the instruction. The port numbers are from 00 to FF i.e. 8 bit address is directly specified. Thus addressing mode is Direct port addressing The above example is an example of direct port addressing.
- For the **Variable port** IN instruction, the port address is loaded into the DX register before the IN instruction. Since DX is a 16 bit register, the port address can be any number between 0000H and FFFFH. Therefore it is possible to address up to 65, 536 ports in this mode. → Port Address

e.g. MOV DX, 0FF0H ; Initialize DX to point to port.
IN AL, DX; Input an 8 bit data from port 0FF0 to AL i.e. contents of port 0FF0 are copied to the AL register. The addressing mode is Indirect port addressing mode .

Note : This instruction is basically related to communication with I/O devices which are mapped in the I/O mapped I/O .

2. OUT Instruction – Output byte or word to a port

Mnemonic OUT port address, Accumulator. **Flags** No flags are affected

Algorithm —

Addr. Mode Indirect port addressing mode

Operation Contents of AX → Port address / Contents of AL → port address.

This Instruction copies a byte from AL or a word from AX to the specified port.

Example MOV DX, FFF8H
OUT DX, AL

Load desired port address in DX.
Copies contents of AL to given port address in register DX .

- It has two possible forms.

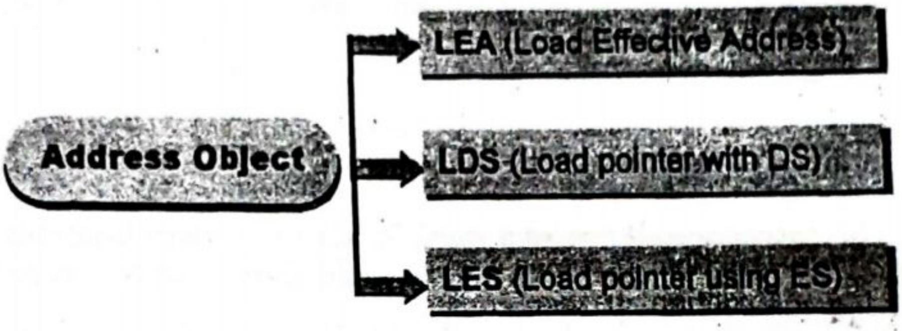
- Fixed port
- Variable port.

- In the **fixed port** form, the 8 bit port address is specified directly in the instruction with this form, any one of 256 possible ports can be addressed.

  e.g. OUT 3B H, AL ; copies the contents of AL to port 3B H.
- The addressing mode is Direct port addressing mode.
- In the **variable port** form, the contents of AL or AX will be copied to the port at an address contained in DX. The DX register should be loaded with the desired port address. The above example in table is an example of indirect port addressing.

12.6.3 Address Object

These instructions manipulate the addresses of the variables, rather than the contents or values of the variables. These are mainly used for list processing, based variables and string operation.


Address Object
LEA (Load Effective Address)
LDS (Load pointer with DS)
LES (Load pointer using ES)

1. LEA – Load Effective Address

Mnemonic LEA register , source. **Flags** No flags are affected

Algorithm REG = Address of memory (offset)

Addr. Mode Register Direct Addressing

Operation REG ← source.

- This instruction determines the offset of variables or memory location named as source and puts the offset in the indicated 16 bit register.
- Generally this instruction is replaced by MOV when assembling is possible.
- Normally the offset is loaded into index register or base pointer registers such as SI, DI, BX, BP.

Example LEA AX, COUNT — offset

This instruction loads AX with the offset of COUNT in DS.

2. LDS – Load pointer with DS

(Load Register and DS with words from memory)

Mnemonic LDS register , source. **Flags** No flags are affected

Algorithm REG = First word, DS = Second word

Addr. Mode Register Direct Addressing.

Operation REG ← Source, DS ← (Source + 2)

- The source is always a memory location. DS is used as a segment register for memory.
- This instruction is a 2 byte instruction. It copies a word from two memory locations into register specified in the instruction. It then copies a word from the next two memory locations into the DS register.

3. LES – Load pointer using ES

(Load register and ES with words from the memory)

Mnemonic LES register , source. **Flags** No flags are affected

Algorithm REG = First word, ES = Second word

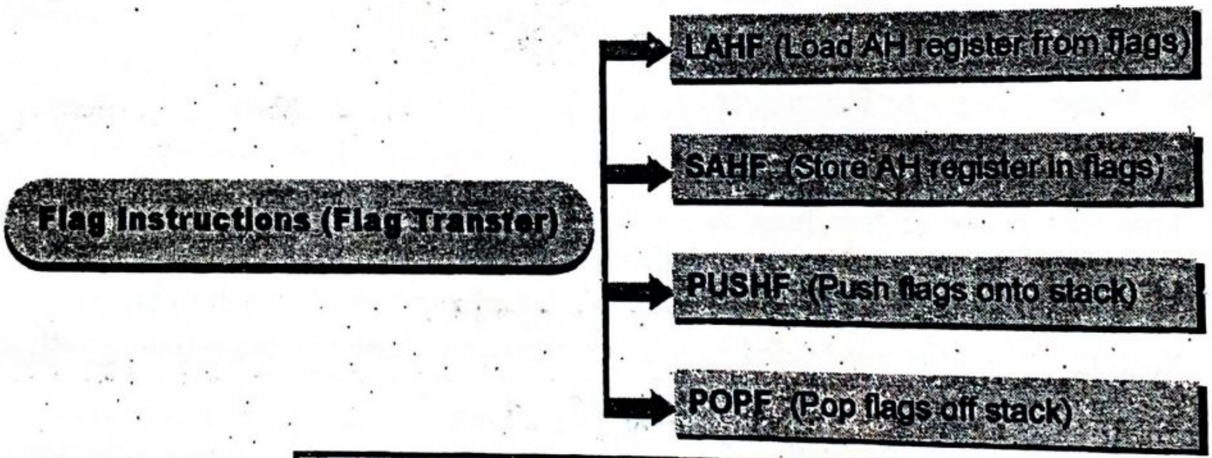
Addr. Mode Register Direct Addressing

Operation REG ← Source, ES ← (source + 2)

- The source is always a memory location.
- This is a 2 byte instruction. It copies a word from two memory locations into register specified in the instruction. It then copies a word from next two memory locations into the ES register.

12.6.4 Flag Instructions (Flag Transfer)

These instructions are related to movement of flag register to/from a register and memory.


Flag Instructions (Flag Transfer)
LAHF (Load AH register from flags)
SAHF (Store AH register in flags)
PUSHF (Push flags onto stack)
POPF (Pop flags off stack)

1. LAHF – Load AH register from flags

(Copy lower byte of flag register to AH)

Mnemonic LAHF **Flags** No flags are affected.

Algorithm AH = flag register's lower byte

Addr. Mode Implied Addressing mode

Operation AH ← Lower byte of flag register

The lower byte of 8086 flag register is copied to the AH register

2. SAHF – Store AH register in flags

(Copy contents of AH to lower byte of flag register)

Mnemonic SAHF **Flags** All the flags are changed.

Algorithm AH = flag register