8086 INSTRUCTIONS GROUPS :

- Data transfer instructions
- Arithmetic instructions
- Logic instructions
- Shift instructions
- Rotate instructions
- Flag control instructions
- Compare instruction
- Control flow and jump instructions
- Loop instruction
- String instruction

1- Data transfer instructions :

The 8086 microprocessor has a group of data-transfer instructions that are provided to move data either between its internal registers or between an internal register and a storage location in memory.
This group includes the 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, and load extra segment **(LES)** instruction.

- **MOV INSTRUCTION**

MOV instruction transfers data from a source operand to a destination operand. Mean that it used to copy the contents of a register or memory location into another register or contents of a register into a storage location in memory. In all of these cases, the original contents of the source location are preserved and the original contents of the destination are destroyed.

MOV D,S

the operands can be internal registers of the 8086 and storage locations in memory. the valid source and destination operand variations shown below:

Destination	Source
Memory	Accumulator
Accumulator	Memory
Register	Register
Register	Memory
Memory	Register
Register	Immediate
Memory	Immediate
Seg-reg	Reg16
Seg-reg	Mem16
Reg16	Seg-reg
Memory	Seg-reg

Note that:-

- the MOV instruction cannot transfer data directly between a source and a destination. which both reside in external memory. Instead, the data must first be moved from memory into an internal register, such as to the accumulator (AX), with one move instruction, and then moved to the new location in memory with a second move instruction.
- All transfers between general-purpose registers and memory can involve either a byte or word of data. The fact that the instruction corresponds to byte or word data is designated by the way in which its operands are specified. For instance, AL or AH is used to specify a byte operand, and AX, a word operand. On the other hand, data moved between one of the general-purpose registers and a segment register or between a segment register and a memory location must always be word - wide(16-bit).
- flag bits within the 8086 are not modified by execution of a MOV instruction.

Ex:

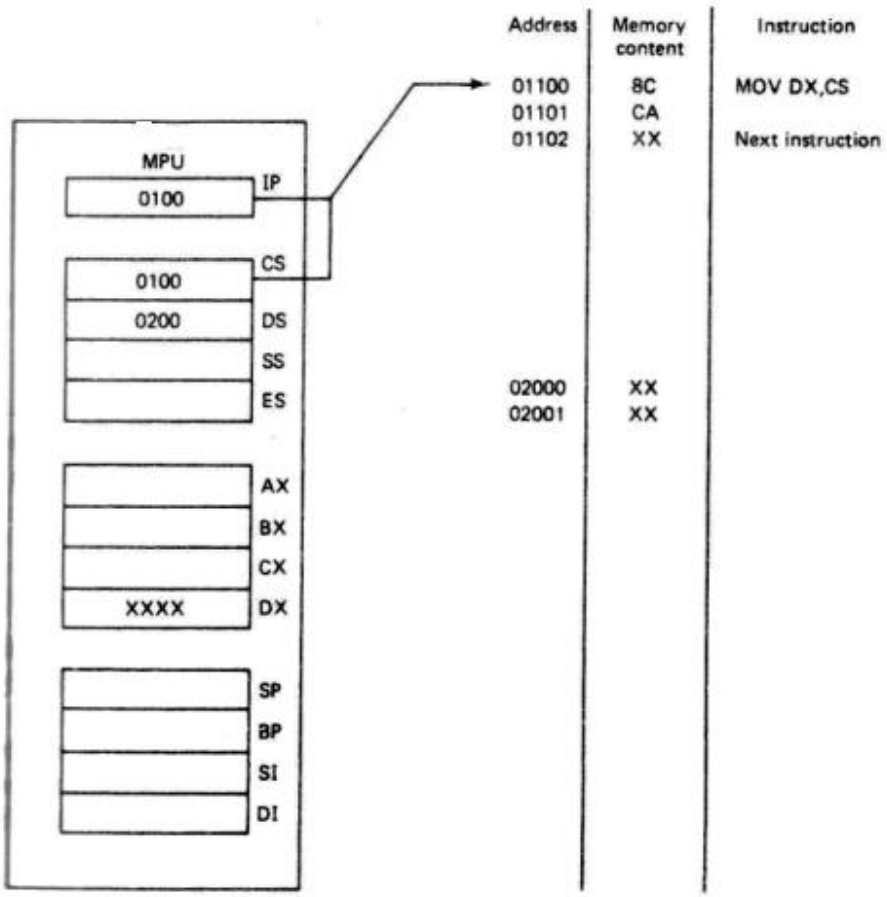
MOV DX, CS

In this instruction, the code segment register is the source operand, and the data register is the destination. It stands for "move the contents of CS into DX" That is,

(CS) → (DX)

For example, if the contents of CS are 0100H, execution of the instruction MOV DX, CS as shown in Fig.1. makes:

(DX) = (CS) = 0100 H


Address
Memory content
Instruction
01100
01101
01102
8C
CA
XX
MOV DX,CS
Next instruction
02000
02001
XX
XX
MPU
IP
0100
CS
0100
0200
DS
SS
ES
AX
BX
CX
XXXX
DX
SP
BP
SI
DI

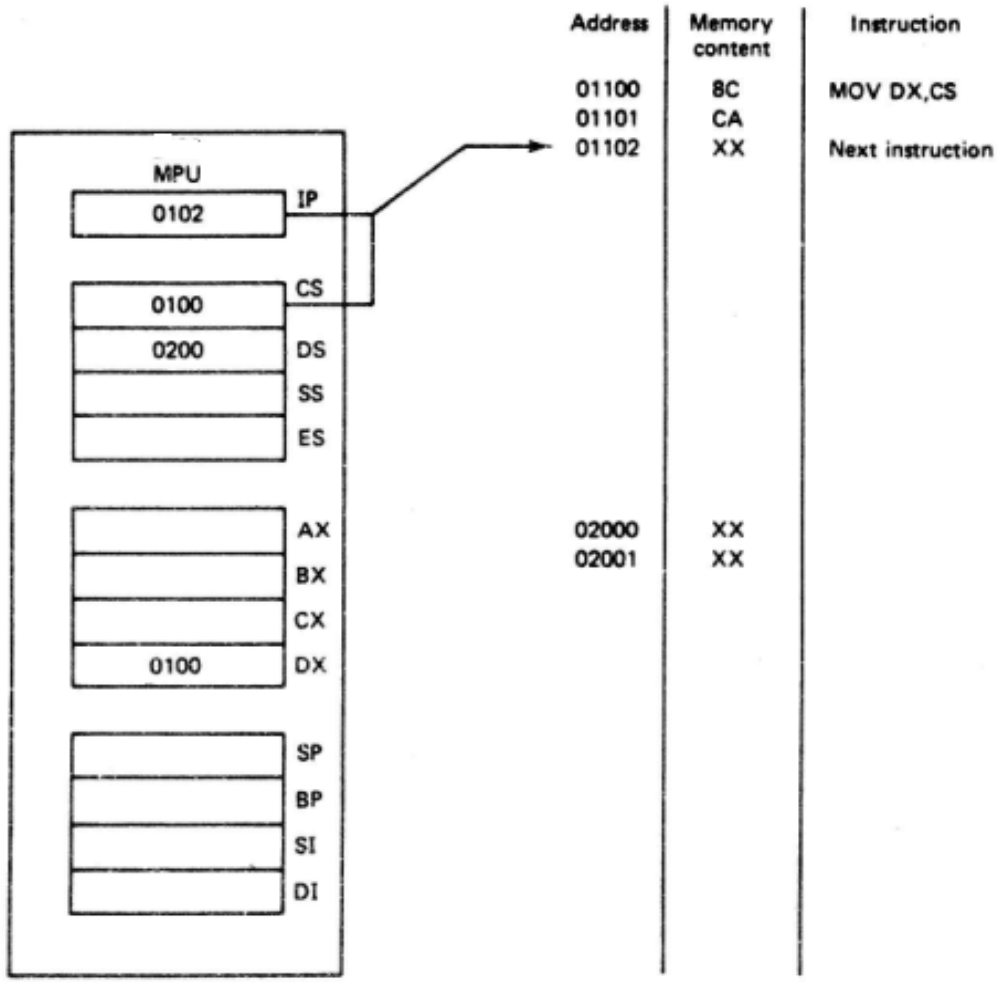


Fig-1- before/after execution

- **XCHG INSTRUCTION**

used to exchange (swap) data between two general-purpose registers or between a general-purpose register and a storage location in memory.
The forms of the XCHG instruction and its allowed operands are shown in Figures below:

Mnemonic	Meaning	Format	Operation	Flags affected
XCHG	Exchange	XCHG D,S	(D) ↔ (S)	None

(a)

Destination	Source
Accumulator	Reg16
Memory	Register
Register	Register
Register	Memory

(b)

Fig (a) XCHG data transfer instruction. fig (b) Allowed operands.

The XCHG instruction, used 16-bit AX register with another 16-bit register, is the most efficient exchange, it take one byte " code instruction" in memory. Other XCHG instructions require 2 or more bytes.

EX:

XCHG AX, DX

The 8086 swaps the contents of AX with that of DX

(AX original) → (DX)

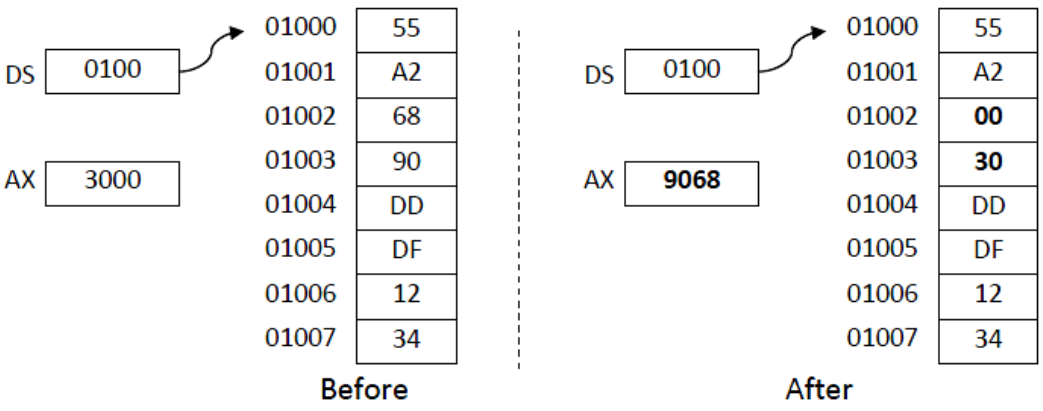
(DX original) → (AX)

(AX) ↔ (DX)

Ex:

What is the result of executing the following instruction?

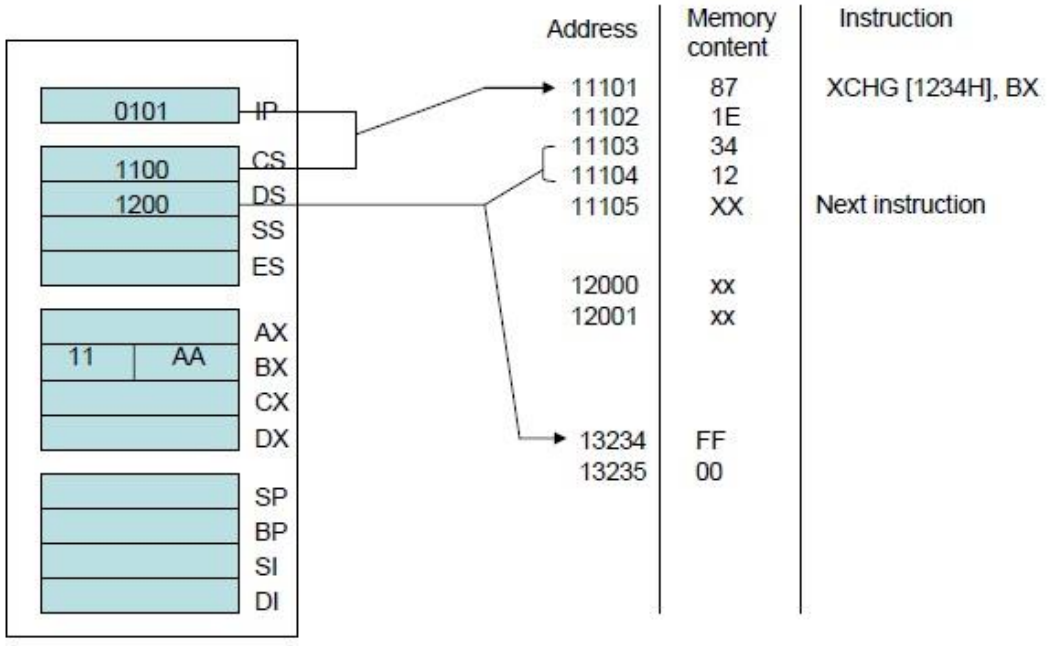
XCHG AX , [0002]



Ex:

For the data shown in the figure what is the result of executing the following Instruction

XCHG [1234H], BX



Execution of this instruction performs the function

(DSx10 +1234H)↔ (BX)

(DS) =1200H, EA or offset address =1234H,

The PA=12000H+1234H= 13234H

The data at this address is FFH and the address that follows contains 00H.

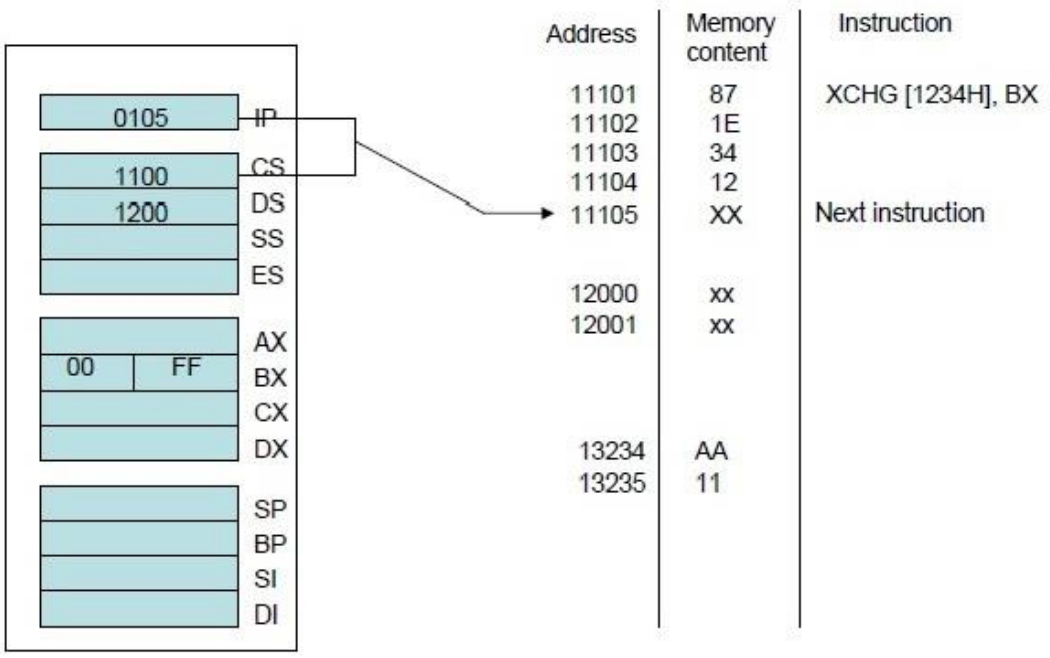
Execution of this instruction performs the following 16-bit swap:

(13234H)↔ (BL)

(13235H)↔(BH)

Thus we get **(BX)= 00FFH**

[DS:1234]= 11AAH



- **XLAT instruction**

XLAT instruction converts the contents of AL register into a number stored in a memory table, this instruction perform the direct table look up technique often used to convert one code to another.

Mnemonic	Meaning	Format	Operation	Flags affected
XLAT	Translate	XLAT	((AL) + (BX) + (DS) *10) →AL	none

XLAT data transfer instruction

XLAT is the only instruction that adds an 8 bit number to 16-bit number.

When using XLAT, the contents of register BX represent the offset of the starting address of the lookup table from the beginning of the current data segment. Also, the contents of AL represent the offset of the element to be accessed from the beginning of the lookup table. This 8-bit element address permits a table with 256 elements. The values in both of these registers must be initialized prior to execution of the XLAT instruction.
Execution of XLAT replaces the contents of AL by the contents of the accessed lookup table location. The physical address of this element in the table is derived as
PA = (DS)0 + (BX) + (AL)
An example of the use of this instruction is for software code conversions. Fig-1- below illustrates how an HEX code—to—BCD code conversion can be performed with the translate instruction.
As shown, DS:BX defines the starting address where the BCD code table is stored in memory. This gives the physical address
PA = (DS)0 + (BX) = 03000H+ 0100H = 03100H

The individual BCD codes are located in the table at element displacements equal to their equivalent HEX code values. For example, the BCD code 13, is positioned at displacement 0D H from the start
of the table.

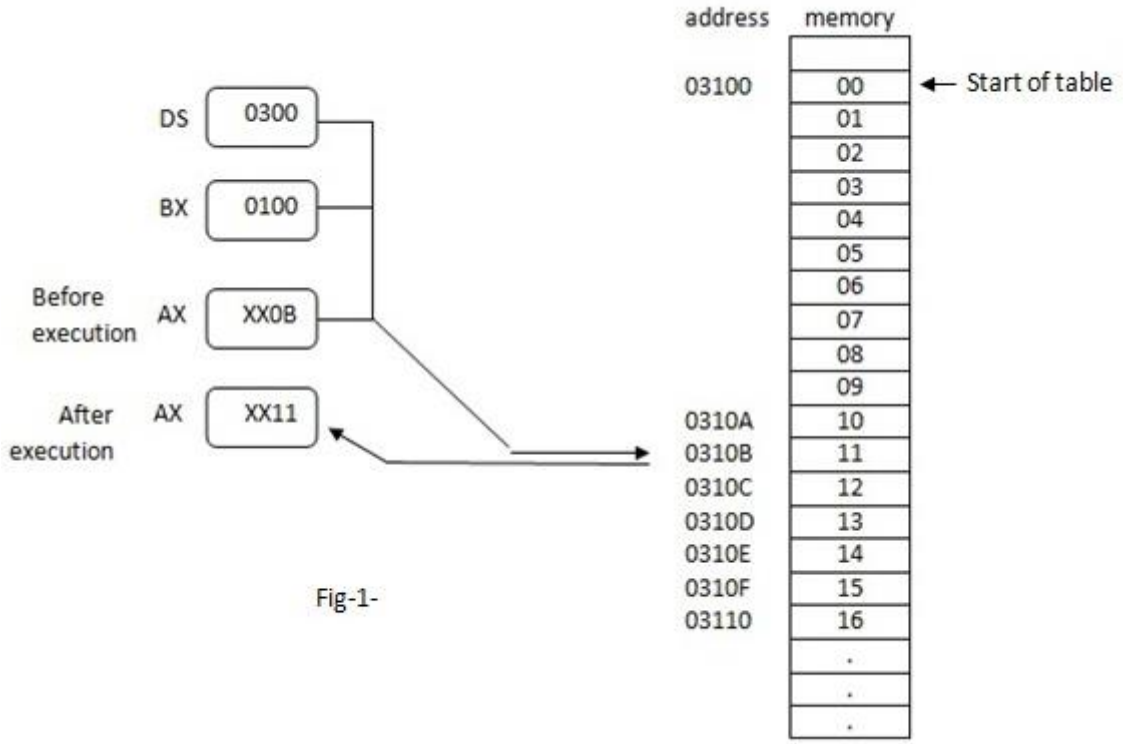


Fig-1-

As an illustration of XLAT, let us assume (AL) = 0B H. Execution of XLAT replaces the contents of AL by the contents of the memory location given by

PA = (DS)0 + (BX) + (AL)

= 03000H + 0100H+ 0B H =0310BH

Thus, the execution can be described by

(0310B H) → (AL)

This memory location contains 11 (BCD code) and this value is placed in AL:

(AL) = 11

- **LEA, LDS, and LES instructions**

An important type of data transfer operation is loading a segment and a general-purpose register with an address directly from memory. Special instructions are provided in the instruction set of the 8086 to give a programmer this capability. These instructions are :
load register with effective address (LEA),
load register and data segment register (LDS),
and load register and extra segment register (LES).
Looking at Figure below , we see that these instructions provide programmers with the ability to manipulate memory addresses by loading either a 16-bit offset address into a general-purpose register or a 16-bit offset address into a general-purpose register together with a 16-bit segment address into either DS or ES.

Mnemonic	Meaning	Format	Operation	Flags affected
LEA	Load effective address	LEA Reg16,EA	EA → (Reg16)	None
LDS	Load register and DS	LDS Reg16,EA	EA → (Reg16) EA+2 → (DS)	None
LES	Load register and ES	LES Reg16,EA	EA → (Reg16) EA+2 → (ES)	None

The LEA instruction is used to load a specified register with a l6-bit offset address.

Ex:

LEA SI , EA

When executed, it loads the SI register with an offset address value. The value of this offset is represented by the effective address EA.
The value of EA can be specified by any valid addressing mode. For instance, if the value in DI equals 1000H and that in BX is 20H, then executing the instruction
LEA SI, [DI + BX + 5H]
will load SI with the value
EA = 1000H + 20H + 5H = 1025H
That is,
(SI) = 1025H
The other two instructions, LDS and LES, are similar to LEA except that they load the specified register as well as the DS or ES segment register, respectively. That is, they are able to load a complete address pointer that is stored in memory. In this way, executing a single instruction can activate a new data segment.

Ex:
Assuming that the 8086 is initialized as shown in Fig. 1., what is the result of executing the following instruction?

LDS SI, [200H]

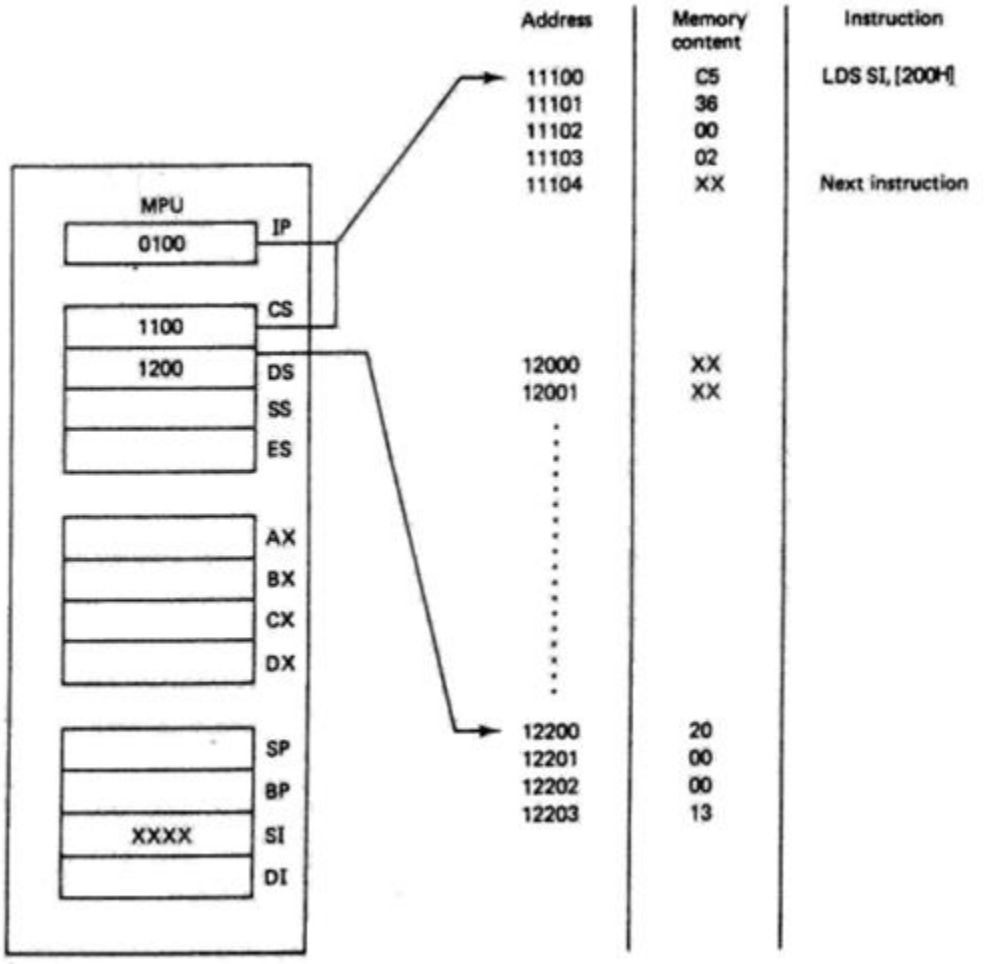


Fig.1.

Execution of the instruction loads the SI register from the word location in memory whose offset address with respect to the current data segment is 200H. Figure .1. shows that the contents of DS are 1200H. This gives a physical address

PA = 12000H + 0200H = 12200H

It is the contents of this location and the one that follows that are loaded into SI. Therefore, in Fig.2. we find that SI contains 0020H. The next two bytes-that is, the contents of addresses 12202H and 12203H—are loaded into the DS register, As shown, this defines a new data segment address of 13000H.

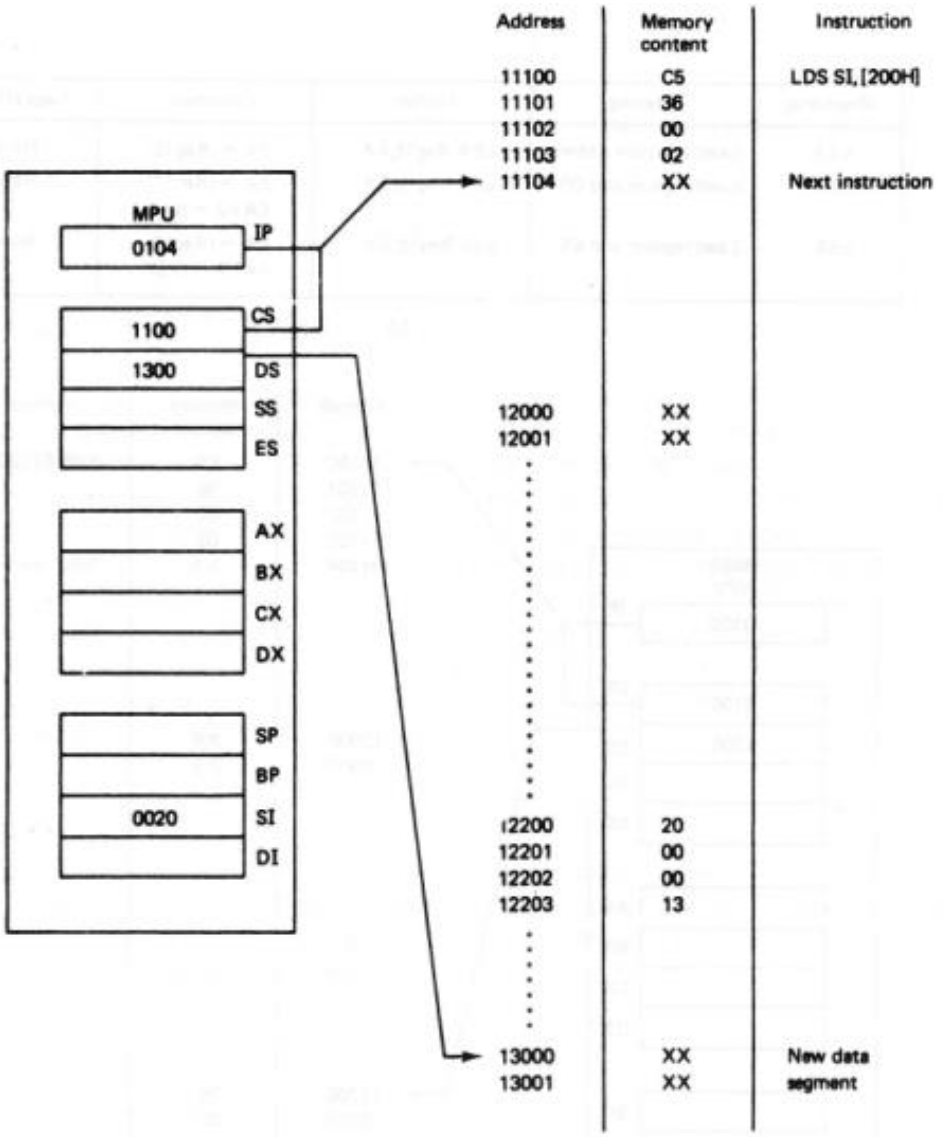


Fig.2.

NOTE: The instruction LES is similar to the instruction LDS except that it load the **Extra Segment Register** instead of **Data Segment Register**

Ex:

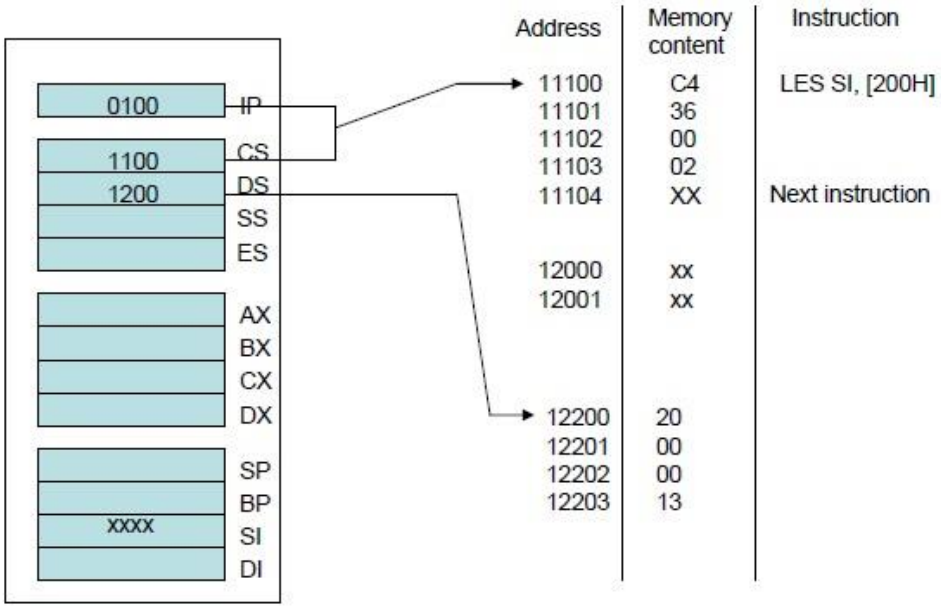
as shown in the following figure. What is the result of executing the following instruction?

LES SI, [200H]

This instruction loads SI from memory location at physical location

PA= 12000H + 200H = 12200H

and ES is loaded with the contents of the following two bytes at address 12202H



After execution of **LES** SI, [200H]