Data Transfer Instructions

1) MOV Destination, Source

Moves a byte/word **from** the **source to** the **destination** specified in the instruction.
Source: *Register, Memory Location, Immediate Number*
Destination: Register, Memory Location
Both, source and destination cannot be memory locations.

Eg: **MOV CX, 0037H** ; *CX ← 0037H*
MOV BL, [4000H] ; *BL ← DS:[4000H]*
MOV AX, BX ; *AX ← BX*
MOV DL, [BX] ; *DL ← DS:[BX]*
MOV DS, BX ; *DS ← BX*

2) PUSH Source

Push the **source** (word) **into** the **stack** and decrement the stack pointer by two.
The source MUST be a **WORD** (**16 bits**).
Source: *Register, Memory Location*

Eg: **PUSH CX** ; *SS:[SP-1] ← CH, SS:[SP-2] ← CL*
; *SP ← SP – 2*
PUSH DS ; *SS:[SP-1, SP-2] ← DS*
; *SP ← SP – 2*

3) POP Destination

POP a **word from** the **stack** into the given destination and increment the Stack Pointer by 2. The destination MUST be a **WORD** (16 bits).
Destination: *Register [EXCEPT CS], Memory Location*

Eg: **POP CX** ; *CH ← SS:[SP], CL ← SS:[SP+1]*
; *SP ← SP + 2*
POP DS ; *DS ← SS:[SP, SP+1]*
; *SP ← SP + 2*

Please Note: **MOV, PUSH, POP** are the ONLY instructions that use the Segment Registers as operands {except CS}.

4) PUSHF

Push value of **Flag Register into stack** and decrement the stack pointer by 2.
Eg: **PUSHF** ; *SS:[SP-1] ← $Flag_H$, SS:[SP-2] ← $Flag_L$, SP ← SP – 2*

5) POPF

POP a **word from** the **stack into** the **Flag register**.
Eg: **POPF** ; *$Flag_L$ ← SS:[SP], $Flag_H$ ← SS:[SP+1], SP ← SP + 2*

6) XCHG Destination, Source

Exchanges a byte/word between the **source and** the **destination** specified in the instruction.
Source: *Register, Memory Location*
Destination: Register, Memory Location
Even here, both operands cannot be memory locations.

Eg: **XCHG CX, BX** ; *CX ←→ BX*
XCHG BL, CH ; *BL ←→ CH*

7) **XLATB / XLAT** (very important)
Move into AL, the **contents of** the **memory location** in Data Segment, **whose** effective **address** is **formed by** the **sum of BX and AL**.

Eg: **XLAT** ; *AL ← DS:[BX + AL]*
; i.e. if DS = 1000H; BX = 0200H; AL = 03H
; ∴ 10000 … DS × 16
; + 0200 … BX
; + 03 … AL
; =**10203** ∴ **AL ← [10203H]**

Note: the difference between XLAT and XLATB

In XLATB there is no operand in the instruction.
E.g.:: XLATB
It works in an implied mode and does exactly what is shown above.

In XLAT, we can specify the name of the look up table in the instruction
E.g.:: XLAT SevenSeg
This will do the translation form the look up table called SevenSeg.
In any case, the base address of the look up table must be given by BX.

8) LAHF
Loads AH with **lower byte** of the **Flag** Register.

9) SAHF
Stores the contents of **AH into** the **lower byte** of the **Flag** Register.

10)LEA register, source
Loads Effective Address (offset address) **of** the **source into** the **given register**.

Eg: **LEA BX, Total** ; *BX ← offset address of Total in Data Segment.*

11)LDS destination register, source
Loads the **destination register and DS** register **with offset** address **and segment address** specified by the **source**.

Eg: **LDS BX, Total** ; *BX ← {DS:[Total], DS:[Total + 1]},*
; *DS← {DS: [Total + 2], DS:[Total + 3]}*

12)LES destination register, source
Loads the **destination register and ES** register with the **offset address and** the **segment address** indirectly specified by the **source**.

Eg: **LES BX, Total** ; *BX ← {DS:[Total], DS:[Total + 1]},*
; *ES← {DS: [Total + 2], DS:[Total + 3]}*

I/O ADDRESSING MODES OF 8086 (5m – Important Question)

I/O addresses in 8086 can be either 8–bit or 16-bit

Direct Addressing Mode:

If we use **8-bit I/O address** we get a **range of 00H… FFH**.
This gives a total of **256 I/O ports.**
Here we use Direct addressing Mode, that is, the **I/O address is specified in the instruction.**

```
E.g.:: IN AL, 80H                ; AL gets data from I/O port address 80H.
```

This is also called **Fixed Port Addressing**.

Indirect Addressing Mode:

If we use **16-bit I/O address** we get a **range of 0000H… FFFFH**.
This gives a total of **65536 I/O ports.**
Here we use Indirect addressing Mode, that is, the **I/O address is specified by DX register.**

```
E.g.::  MOV DX, 2000H
        IN AL, DX                ; AL gets data from I/O port address 2000H given by DX.
```

This is also called **Variable Port Addressing**.

13)IN destination register, source port
Loads the **destination register with** the contents of the **I/O port** specified by the source.

Source: It is an I/O port address.
If the address is 8-bit it will be given in the instruction by **Direct addressing mode.**
If it is a 16 bit address it will be given by DX register using **Indirect addressing mode.**

Destination: It has to be some form of "A" register, in which we will get data from the I/O device.
If we are getting 8-bit data, it will be AL or AH register.
If we are getting 16-bit data, it will be AX register.

```
Eg: IN AL, 80H              ; AL gets 8-bit data from I/O port address 80H
    IN AX, 80H              ; AX gets 16-bit data from I/O port address 80H
    IN AL, DX               ; AL gets 8-bit data from I/O port address given by DX.
    IN AX, DX               ; AX gets 16-bit data from I/O port address given by DX.
```

14)OUT destination port, source register
Loads the destination I/O port with the contents of the source register.

```
Eg: OUT 80H, AL             ; I/O port 80H gets 8-bit data from AL
    OUT 80H, AX             ; I/O port 80H gets 16-bit data from AX
    OUT DX, AL              ; I/O port whose address is given by DX gets 8-bit data from AL
    OUT DX, AX              ; I/O port whose address is given by DX gets 16-bit data from AX
```