BITS Pilani

Microprocessors & Interfacing

INSTRUCTION SET

Dr. Gargi Prabhu
Department of CS & IS

LEA- LOAD-EFFECTIVE ADDRESS

- The LEA instruction loads a 16- or 32-bit register with the offset address of the data specified by the operand.

E.g. LEA AX,NUMB

- Loads AX with the offset address of NUMB

- MOV BX,OFFSET LIST is same as LEA BX,LIST

Why is the LEA instruction available if the OFFSET directive accomplishes the same task?

- OFFSET only functions with simple operands such as LIST. It may not be used for an operand such as [DI], LIST [SI], and so on.
- OFFSET directive is more efficient than the LEA instruction for simple operands.
- It takes the microprocessor longer to execute the LEA BX,LIST instruction than the MOV BX,OFFSET LIST.
- It is because the assembler calculates the offset address of LIST, whereas the microprocessor calculates the address for the LEA instruction.
- The MOV BX,OFFSET LIST instruction is actually assembled as a move immediate instruction and hence is more efficient.

Example

E.g. LEA BX, [DI]

DI=1000H -> BX=1000H

Can this be done using MOV BX, DI ?

E.g. LEA SI,[BX+DI]

BX=3000H, DI=2000H -> SI=3000H (Sum= modulo-64)

If BX=1000H DI = FF00H , what is SI?

Yes.
LEA BX, [DI] is same as
MOV BX, DI
Both transfer a copy of
offset address(DI) in BX

DI – Data stored at DI
[DI] – Data stored at the
address of DI

Example on DosBox

```
-A 100
0859:0100 MOV DI,0101
0859:0103 MOV AX,DI
0859:0105 LEA BX,[DI]
0859:0107
-T
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0859 ES=0859 SS=0859 CS=0859 IP=011E NV UP EI NG NZ NA PO NC
0859:011E 0000          ADD     [BX+SI],AL                  DS:0000=CD
-T=100
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0101
DS=0859 ES=0859 SS=0859 CS=0859 IP=0103 NV UP EI NG NZ NA PO NC
0859:0103 89F8          MOV     AX,DI
-
AX=0101 BX=0000 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0101
DS=0859 ES=0859 SS=0859 CS=0859 IP=0105 NV UP EI NG NZ NA PO NC
0859:0105 8D1D          LEA     BX,[DI]                     DS:0101=0101
-
AX=0101 BX=0101 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0101
DS=0859 ES=0859 SS=0859 CS=0859 IP=0107 NV UP EI NG NZ NA PO NC
0859:0107 89F8          MOV     AX,DI
-
```

Example

```
        .DATA                   ;start data segment
DATA1   DW      2000H           ;define DATA1
DATA2   DW      3000H           ;define DATA2
        .CODE                   ;start code segment
        .STARTUP                ;start program
        LEA SI,DATA1            ;address DATA1 with SI
        MOV DI,OFFSET DATA2     ;address DATA2 with DI
        MOV BX,[SI]             ;exchange DAT1 with DATA2
        MOV CX,[DI]
        MOV [SI],CX
        MOV [DI],BX
        .EXIT
        END
```

LDS, LES

- LDS (Load DS), LES (Load ES)
- Load any 16-bit or 32-bit register with an offset address, and the DS, ES, or SS segment register with a segment address.
- Used when working with data structures that include a far pointer, which consists of both a segment and an offset.

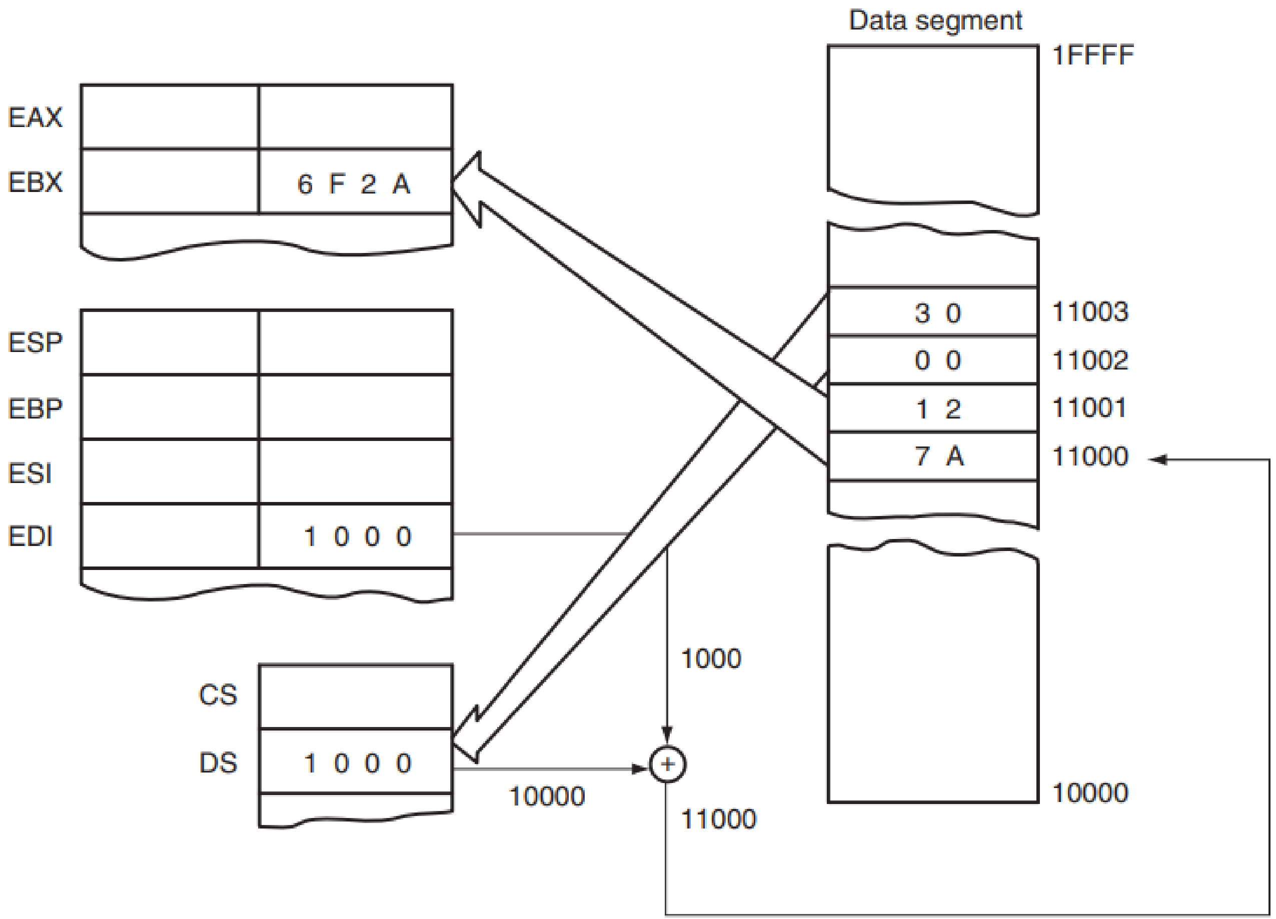
- E.g. LDS BX,[DI] – transfer 32 bit number addressed by DI in data segment into BX and DS registers.

Far Pointer

- A far pointer combines both the segment and offset into a 32-bit value. It is used to address memory locations beyond the 64 KB limit imposed by a single segment in the x86 architecture.
- The format of a far pointer is typically:

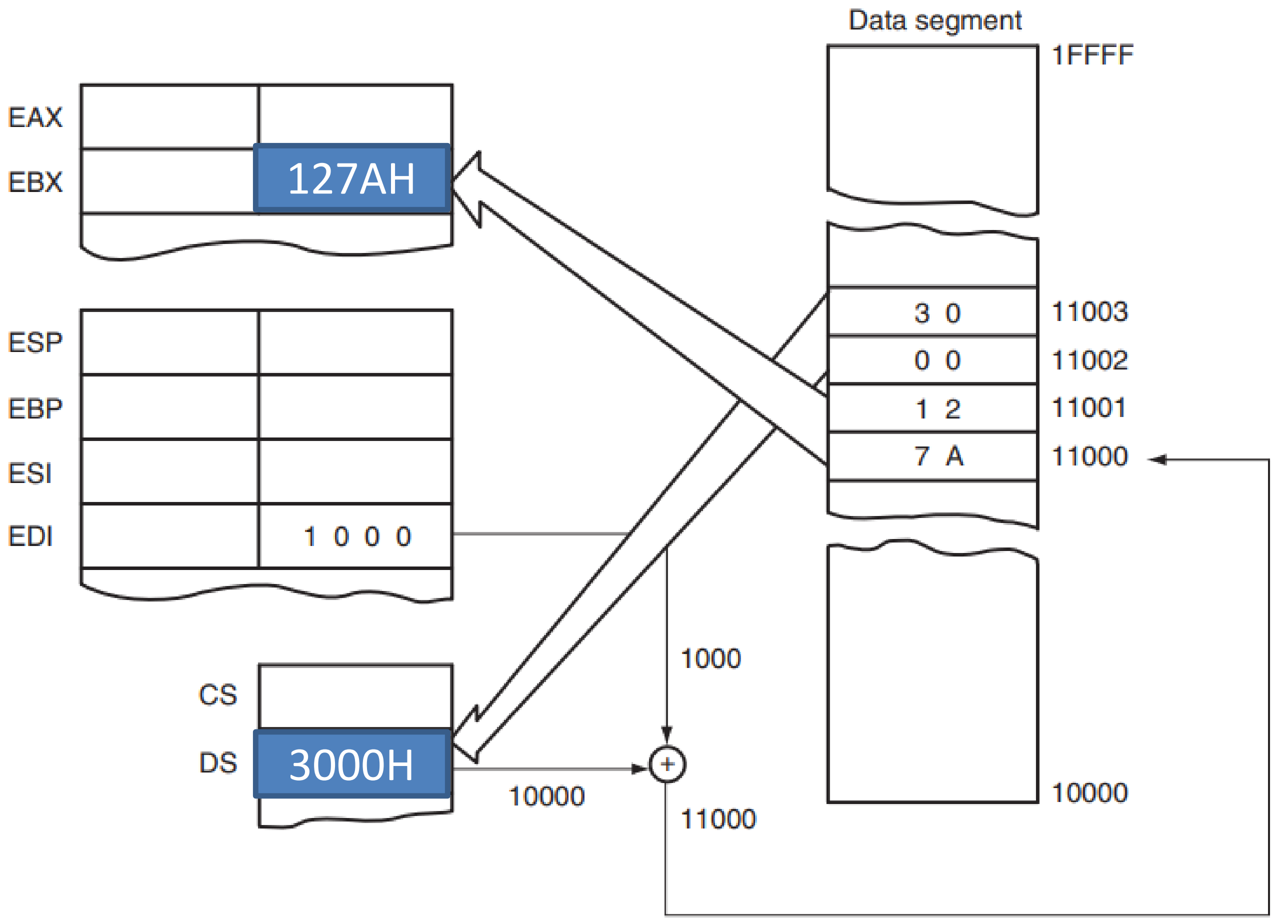
$$\text{Far Pointer} = (\text{Segment} \ll 4) + \text{Offset}$$

LDS Example



Data segment
1FFFF
EAX
EBX
6 F 2 A
ESP
EBP
ESI
EDI
1 0 0 0
CS
DS
1 0 0 0
10000
1000
11000
3 0
0 0
1 2
7 A
11003
11002
11001
11000
10000

LDS Example



Data segment
1FFFF
EAX
EBX
127AH
ESP
EBP
ESI
EDI
1 0 0 0
3 0
0 0
1 2
7 A
11003
11002
11001
11000
1000
CS
DS
3000H
10000
11000
10000

STRING DATA TRANSFERS

- Five string data transfer instructions: LODS, STOS, MOVS, INS, and OUTS.
- Each string instruction allows data transfers that are either a single byte, word, or doubleword (or if repeated, a block of bytes, words, or doublewords).
- String instructions use
  - D flag-bit (direction)
  - DI and SI registers

Direction Flag

- **The Direction Flag:** (D, located in the flag register) selects the auto-increment or the auto-decrement operation for the DI and SI registers during string operations.

- CLD instruction clears the D flag and the STD instruction sets it .
- CLD instruction selects the auto-increment mode and STD selects the auto-decrement mode .

DI and SI

- Memory accesses occur through either or both of the DI and SI registers.
- The DI offset address accesses data in the extra segment for all string instructions that use it.
- The SI offset address accesses data, by default, in the data segment.
- Transferring a byte, the contents of DI and/or SI are **incremented or decremented by 1**.
- Transferring a word, the contents of DI and/or SI are **incremented or decremented by 2.**
- Transferring a Doubleword cause DI and/or SI to **increment or decrement by 4.**

LODS

- The LODS instruction loads AL, AX, or EAX with data stored at the data segment offset address indexed by the SI register.

E.g.

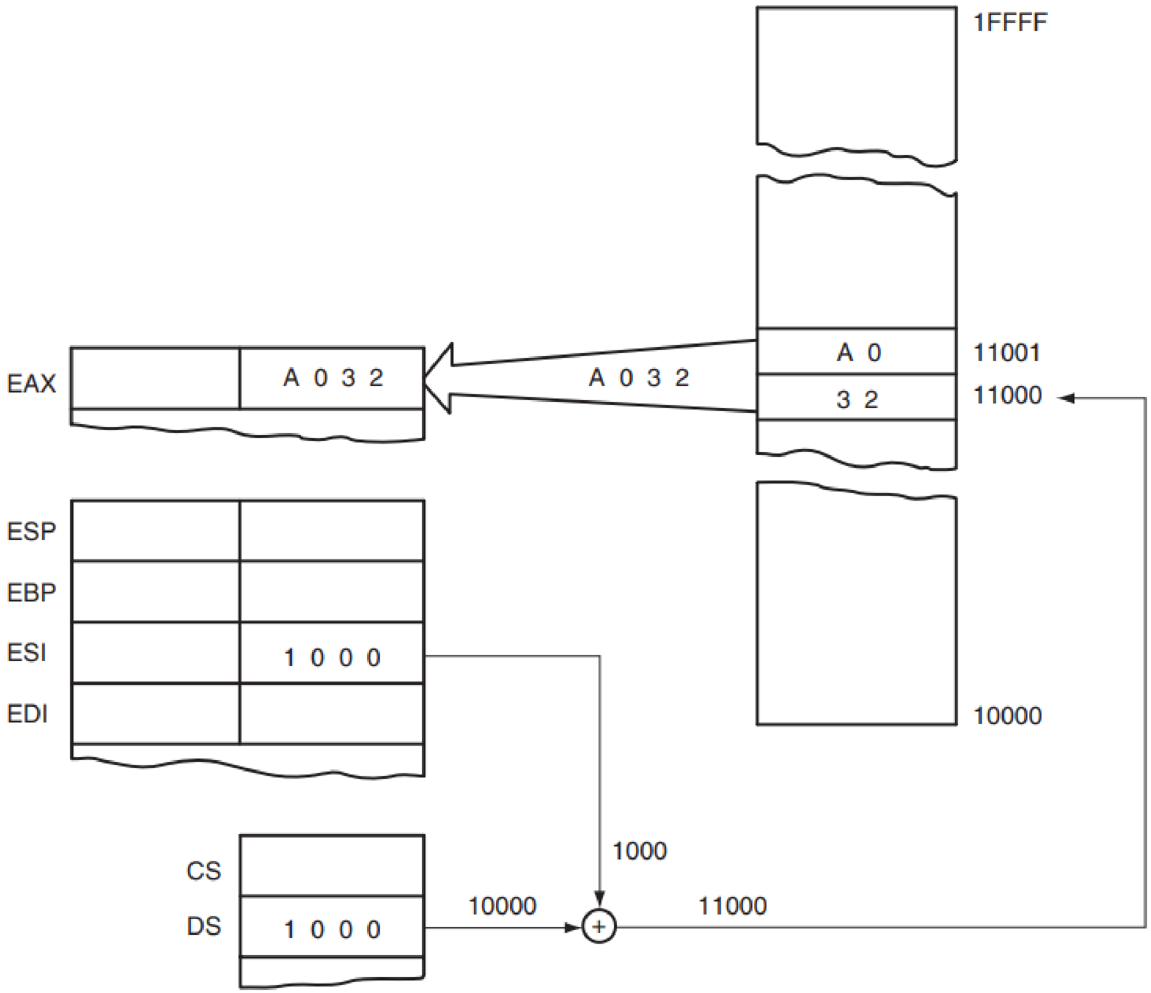
LODSB	AL = DS:[SI]; SI = SI $\pm$ 1
LODSW	AX = DS:[SI]; SI = SI $\pm$ 2

After LODSB , content of SI increment if D=0

Content of SI decrement if D=1

Example: LODSW

FIGURE 4–18 The operation of the LODSW instruction if DS = 1000H, D = 0, 11000H = 32, and 11001H = A0. This instruction is shown after AX is loaded from memory, but before SI increments by 2.



STOS

- Stores AL, AX, or EAX at the extra segment memory location addressed by the DI register.

E.g. STOSB $\quad ES:[DI] = AL; DI = DI \pm 1$

STOSW $\quad ES:[DI] = AX; DI = DI \pm 2$

- The STOSB (stores a byte) instruction stores the byte in AL at the extra segment memory location addressed by DI.
- The STOSW (stores a word) instruction stores AX in the extra segment memory location addressed by DI.

Example

- Suppose that the STOSW instruction is used to clear an area of memory called Buffer using a count called Count and the program is to function call Clear Buffer in the environment using the inline assemble.

```
void ClearBuffer (int count, short* buffer)
{
   _asm{
         push edi              ;save registers
         push es
         push ds
         mov  ax,0
         mov  ecx, count
         mov  edi, buffer
         pop  es               ;load ES with DS
         rep  stosw            ;clear Buffer
         pop  es               ;restore registers
         pop  edi
   }
}
```

The repeat prefix (REP) is added to any string data transfer instruction, except the LODS instruction. The REP prefix causes CX to decrement by 1 each time the string instruction executes. After CX decrements, the string instruction repeats. If CX reaches a value of 0, the instruction terminates and the program continues with the next sequential instruction.

MOVS

- Transfers data from one memory location to another.
- This is the only memory-to-memory transfer allowed in the 8086–Pentium 4 microprocessors.
- The MOVS instruction transfers a byte, word, or doubleword from the data segment location addressed by SI to the extra segment location addressed by DI.
- As with the other string instructions, the pointers then are incremented or decremented, as dictated by the direction flag.

E.g.

MOVSB	ES:[DI] = DS:[SI]; DI = DI ± 1; SI = SI ± 1 (byte transferred)
MOVSW	ES:[DI] = DS:[SI]; DI = DI ± 2; SI = SI ± 2 (word transferred)
MOVSD	ES:[DI] = DS:[SI]; DI = DI ± 4; SI = SI ± 4 (doubleword transferred)

innovate achieve lead

BITS Pilani
Pilani Campus

Thank You