Data movement instructions

Hardware Lab

Introduction

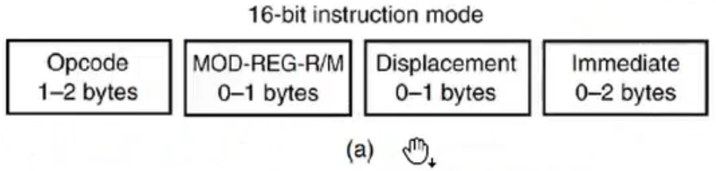
- The data movement instructions include MOV, MOVSX, MOVZX, PUSH, POP, BSWAP, XCHG, XLAT, IN, OUT, LEA, LDS, LES, LFS, LGS, LSS, LAHF, SAHF.

- String instructions: MOVS, LODS, STOS, INS, and OUTS.

Machine Language

- Native binary code microprocessor uses as its instructions to control its operation.
  - instructions vary in length from 1 to 13 bytes
- Over 100,000 variations of machine language instructions.
  - there is no complete list of these variations
- Some bits in a machine language instruction are given; remaining bits are determined for each variation of the instruction.

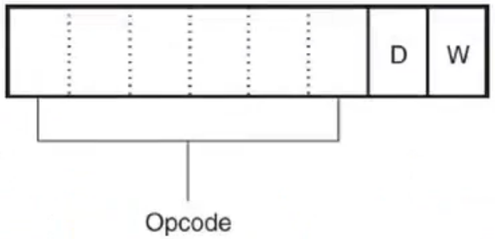
Formats of the 8086–Pentium 4 instructions



- 80386 and above assume all instructions are 16-bit mode instructions when the machine is operated in the *real mode*.
- In *protected mode*, the upper byte of the descriptor contains the D-bit that selects either the 16- or 32-bit instruction mode

The Opcode

- Selects the operation (addition, subtraction, etc.,) performed by the microprocessor.
  - either 1 or 2 bytes long for most instructions
  - first 6 bits of the first byte are the binary opcode
  - remaining 2 bits indicate the **direction** (D) of the data flow, and indicate whether the data are a byte or a word (W)



Byte 1 of many machine language instructions (opcode)

The Opcode

- D=0 → Data flows from REG field to R/M field
- D=1 → Data flows from R/M field to REG field
- D bit appears mostly with MOV instructions

- W=1 → Data is word
- W=0 → Data is byte

MOD Field

- Specifies addressing mode (MOD) and whether a displacement is present with the selected type.

MOD	Function
00	No displacement
01	8-bit sign extended displacement
10	16-bit displacement
11	R/M is a register (Reg. addressing mode)

Byte 2 of many machine language instructions, showing the position of the MOD, REG, and R/M fields.

- All 8-bit displacements are sign-extended into 16-bit displacements when the processor executes the instruction.
- Some assembler programs do not use the 8-bit displacements and in place default to all 16-bit displacements.

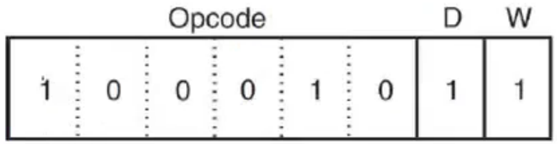
Register Assignments

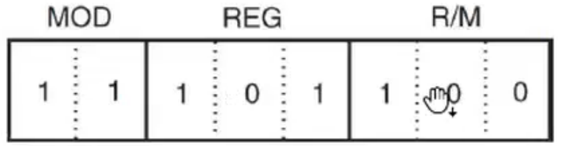
REG and R/M when MOD is 11 (Register addressing)

Code	W=0 (byte)	W=1 (word)	W=1 (Double word)
000	AL	AX	EAX
001	CL	CX	ECX
010	DL	DX	EDX
011	BL	BX	EBX
100	AH	SP	ESP
101	CH	BP	EBP
110	DH	SI	ESI
111	BH	DI	EDI

Register Assignments

- Suppose a 2-byte instruction, 8BECH in machine language





Opcode = MOV
D = Transfer to register (REG)
W = Word
MOD = R/M is a register
REG = BP
R/M = SP

Instruction 8BECH in 16-bit mode in two bytes instruction format (MOV BP,SP)

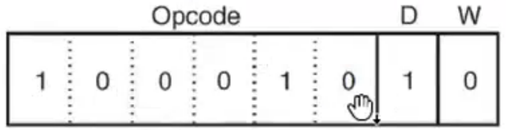
- Opcode is 100010, a MOV instruction
- D and W bits are 1, so a word moves into the destination register specified in the REG field
- REG field contains 101, indicating register BP, so the MOV instruction moves data into register BP

R/M Memory Addressing

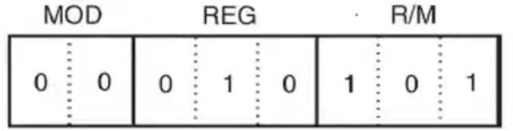
R/M Code	*Addressing Mode*
000	DS:[BX+SI]
001	DS:[BX+DI]
010	SS:[BP+SI]
011	SS:[BP+DI]
100	DS:[SI]
101	DS:[DI]
110	SS:[BP]*
111	DS:[BX]

R/M Memory Addressing

- MOV DL, [DI] or instruction (8AI5H).
- D=1 (to REG from R/M), W=0 (byte)
- MOD=00 (no displacement)
- REG=010 (DL), and R/M=101 ([DI])

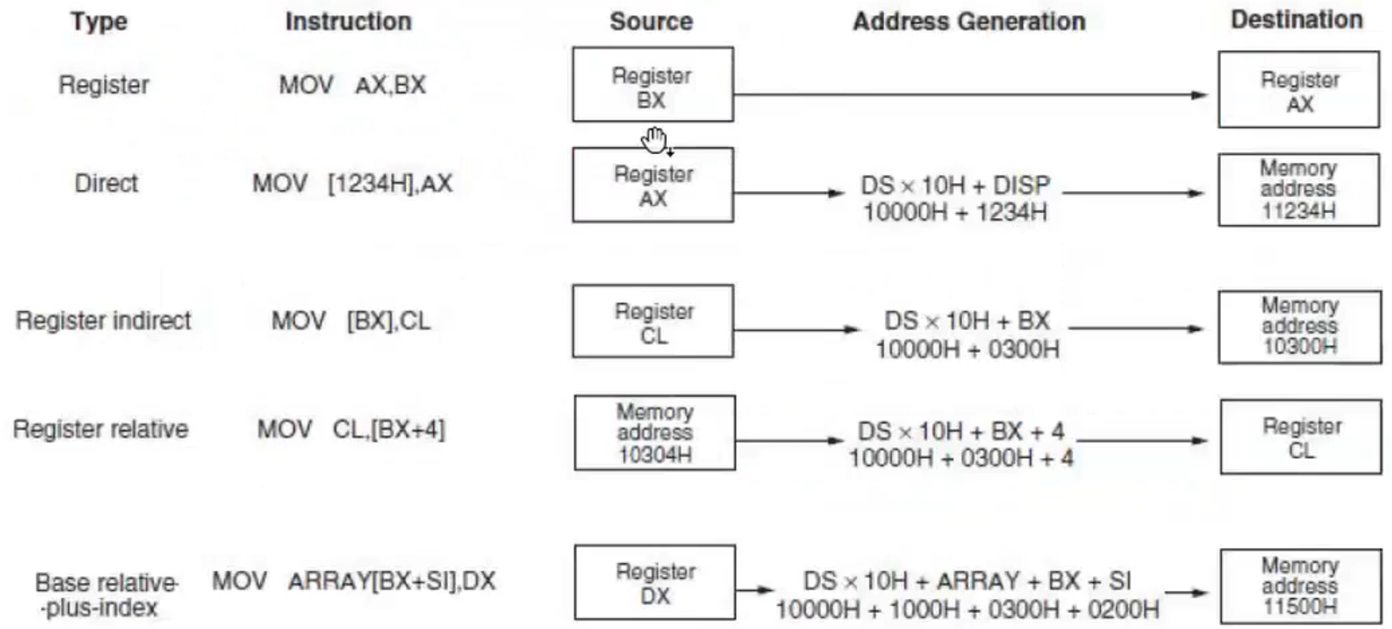


Opcode = MOV
D = Transfer to register (REG)
W = Byte
MOD = No displacement
REG = DL
R/M = DS:[DI]



Instruction 8AI5H in 16-bit mode in two bytes instruction format

MOV


Type
Instruction
Source
Address Generation
Destination
Register
MOV AX,BX
Register BX
Register AX
Direct
MOV [1234H],AX
Register AX
DS × 10H + DISP
10000H + 1234H
Memory address 11234H
Register indirect
MOV [BX],CL
Register CL
DS × 10H + BX
10000H + 0300H
Memory address 10300H
Register relative
MOV CL,[BX+4]
Memory address 10304H
DS × 10H + BX + 4
10000H + 0300H + 4
Register CL
Base relative-plus-index
MOV ARRAY[BX+SI],DX
Register DX
DS × 10H + ARRAY + BX + SI
10000H + 1000H + 0300H + 0200H
Memory address 11500H

PUSH/POP

- Important instructions that *store* and *retrieve* data from the LIFO (last-in, first-out) stack memory.
- Six forms of the PUSH and POP instructions:
  - register, memory, immediate, segment register, flags, all registers
- The PUSH and POP immediate & PUSHA and POPA (all registers) available 80286 - Core2.

PUSH and POP instructions

Assembly Language	*Operation*
POPF	Removes a word from the stack and places it into the flag register
POPFD	Removes a doubleword from the stack and places it into the EFLAG register
PUSHF	Copies the flag register to the stack
PUSHFD	Copies the EFLAG register to the stack
PUSH AX	Copies the AX register to the stack
POP BX	Removes a word from the stack and places it into the BX register
PUSH DS	Copies the DS register to the stack
PUSH 1234H	Copies a word-sized 1234H to the stack
POP CS	This instruction is illegal
PUSH WORD PTR[BX]	Copies the word contents of the data segment memory location addressed by BX onto the stack
PUSHA	Copies AX, CX, DX, BX, SP, BP, DI, and SI to the stack
POPA	Removes the word contents for the following registers from the stack: SI, DI, BP, SP, BX, DX, CX, and AX
PUSHAD	Copies EAX, ECX, EDX, EBX, ESP, EBP, EDI, and ESI to the stack
POPAD	Removes the doubleword contents for the following registers from the stack: ESI, EDI, EBP, ESP, EBX, EDX, ECX, and EAX
POP EAX	Removes a doubleword from the stack and places it into the EAX register
POP RAX	Removes a quadword from the stack and places it into the RAC register (64-bit mode)
PUSH EDI	Copies EDI to the stack
PUSH RSI	Copies RSI into the stack (64-bit mode)
PUSH QWORD PTR[RDX]	Copies the quadword contents of the memory location addressed by RDX onto the stack

PUSH/POP

- Register addressing allows contents of any 16-bit register to transfer to & from the stack.
- Memory-addressing PUSH and POP instructions store contents of a 16- or 32 bit memory location on the stack or stack data into a memory location.
- Immediate addressing allows immediate data to be pushed onto the stack, but not popped off the stack.

PUSH/POP

- Segment register addressing allows contents of any segment register to be pushed onto the stack or removed from the stack.
  - ES may be pushed, but data from the stack may never be popped into ES
- The flags may be pushed or popped from the stack.
- contents of all registers may be pushed or popped

PUSH

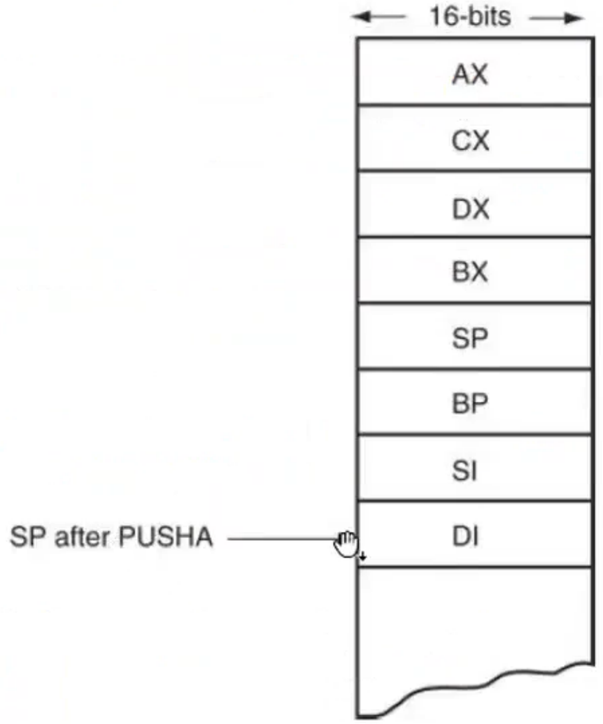
- Always transfers 2 bytes of data to the stack (8086-80286)
  - 80386 and above transfer 2 or 4 bytes
- PUSHA (**push all**) instruction copies the registers to the stack in the following order: AX, CX, DX, BX, SP, BP, SI, and DI.

  [16 bytes = 8 regs x 2 bytes]

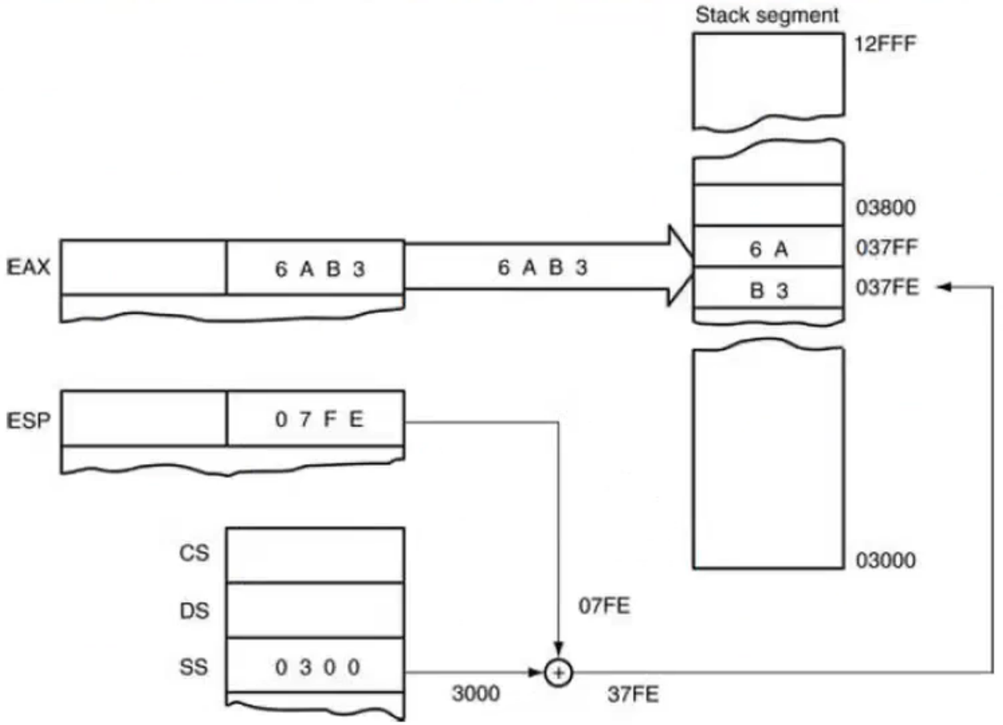
- After all registers are pushed, the contents of the SP register is decremented by 16.

- PUSHA is useful when the entire register set of 80286 and above must be saved.

The operation of the PUSHA instruction, showing the location and order of stack data.



The effect of the PUSH AX instruction on ESP and stack memory locations 37FFH and 37FEH. This instruction is shown after execution.



PUSH

- PUSHF (**push flags**) instruction copies the contents of the flag register to the stack.
- PUSHAD and POPAD instructions push and pop the contents of the 32-bit register set in 80386 - Pentium 4.
- PUSHA and POPA instructions do not function in the 64-bit mode of operation for the Pentium 4

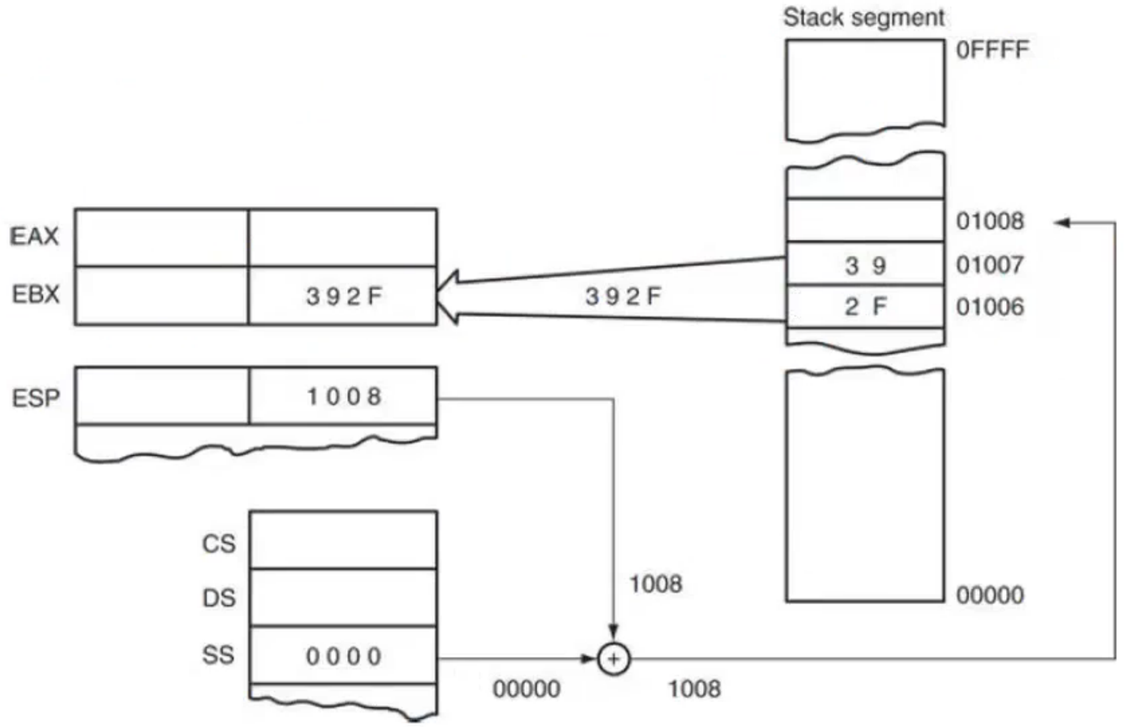
POP

- Performs the inverse operation of PUSH.
- POP removes data from the stack and places it in a target 16-bit register, segment register, or a 16-bit memory location.
  - not available as an immediate POP
- POPF (pop flags) removes a 16-bit number from the stack and places it in the flag register;
  - POPFD removes a 32-bit number from the stack and places it into the extended flag register

POP

- POPA (pop all) removes 16 bytes of data from the stack and places them into the following registers, in the order shown: DI, SI, BP, SP, BX, DX, CX, and AX.
  - reverse order from placement on the stack by PUSHA instruction, causing the same data to return to the same registers

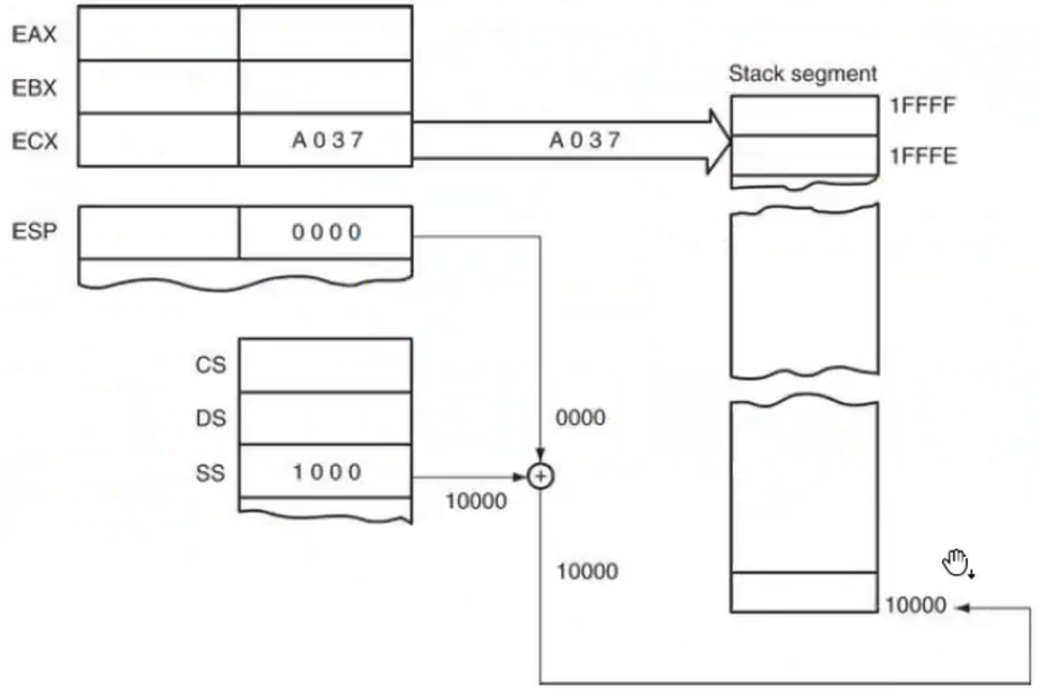
The POP BX instruction, showing how data are removed from the stack. This instruction is shown after execution.



Initializing the Stack

- When the stack area is initialized, load both the stack segment (SS) register and the stack pointer (SP) register.
- All segments are cyclic in nature
  - the top location of a segment is contiguous with the bottom location of the segment
- Assembler and linker programs place correct stack segment address in SS and the length of the segment (top of the stack) into SP.

The PUSH CX instruction, showing the cyclical nature of the stack segment. This instruction is shown just before execution, to illustrate that the stack bottom is contiguous to the top.



LOAD EFFECTIVE ADDRESS

- LEA instruction loads any 16-bit register with the offset address
  - determined by the addressing mode selected
- LDS and LES
  1. load a 16-bit register with offset address retrieved from a memory location
  2. then load either DS or ES with a segment address retrieved from memory
- In 80386 and above, LFS, LGS, and LSS are added to the instruction set.

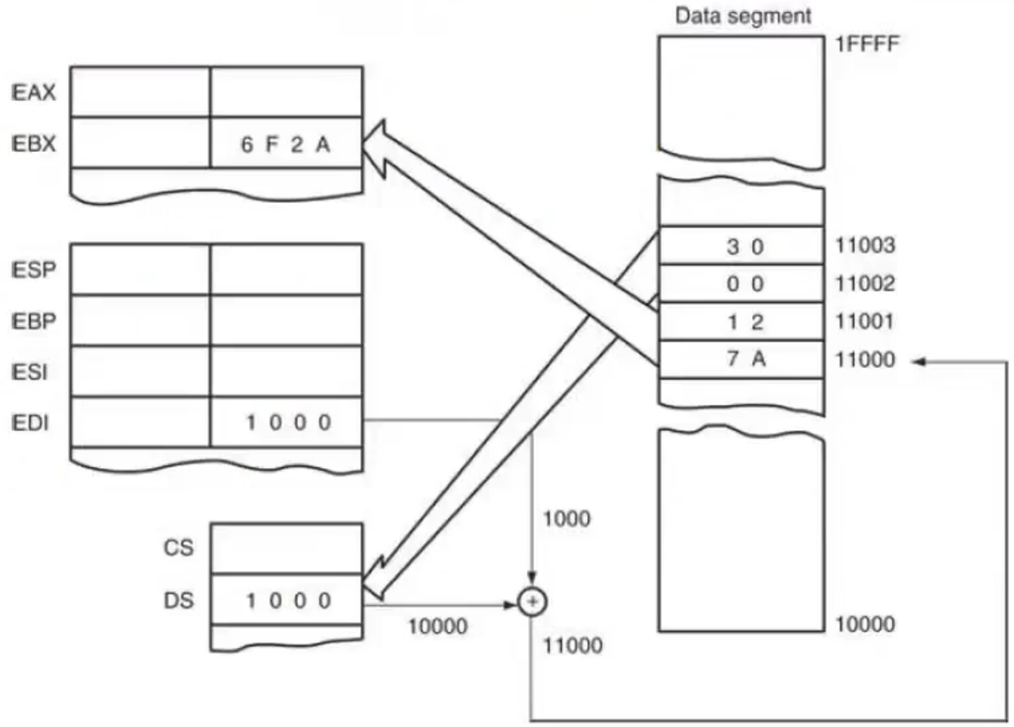
LDS, LES, LFS, LGS, and LSS

- Load any 16- or 32-bit register with an offset address, and the DS, ES, FS, GS, or SS segment register with a segment address.

TABLE 4–10 Load-effective address instructions.

Assembly Language	*Operation*
LEA AX,NUMB	Loads AX with the offset address of NUMB
LEA EAX,NUMB	Loads EAX with the offset address of NUMB
LDS DI,LIST	Loads DS and DI with the 32-bit contents of data segment memory location LIST
LDS EDI,LIST1	Loads the DS and EDI with the 48-bit contents of data segment memory location LIST1
LES BX,CAT	Loads ES and BX with the 32-bit contents of data segment memory location CAT
LFS DI,DATA1	Loads FS and DI with the 32-bit contents of data segment memory location DATA1
LGS SI,DATA5	Loads GS and SI with the 32-bit contents of data segment memory location DATA5
LSS SP,MEM	Loads SS and SP with the 32-bit contents of data segment memory location MEM

The LDS BX,[DI] instruction loads register BX from addresses 11000H and 11001H and register DS from locations 11002H and 11003H. This instruction is shown at the point just before DS changes to 3000H and BX changes to 127AH.



offset address appears first, followed by the segment address

STRING DATA TRANSFERS

- Five string data transfer instructions:
  - LODS, STOS, MOVS, INS, and OUTS.
- Each allows data transfers as a single byte, word, or doubleword.
- The operation of the D flag-bit (direction), DI, and SI must be understood as they apply to the string instructions.

The 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.
  - used only with the string instructions

- **CLD** instruction clears the D flag (D=0) -> auto-increment mode
- **STD** instruction sets the D flag (D=1) -> auto-decrement mode

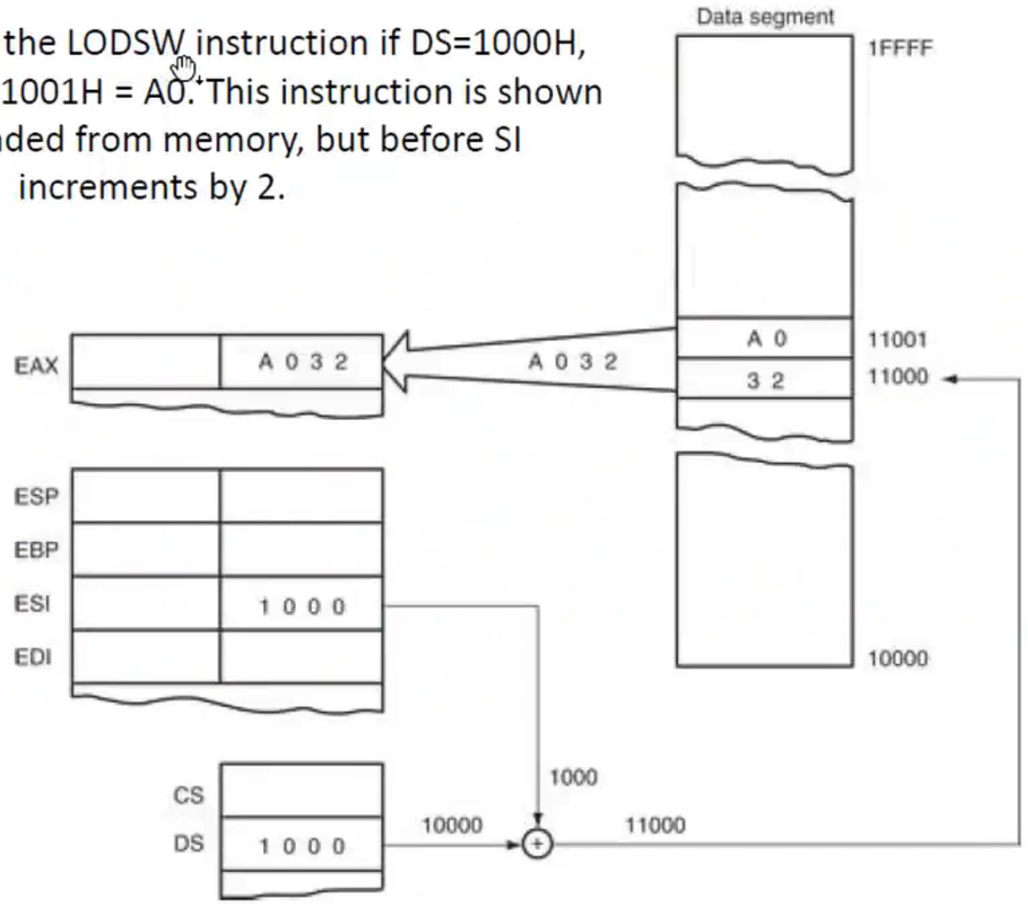
DI and SI

- During execution of string instruction, memory accesses occur through DI and SI registers.
  - DI offset address accesses data in the extra segment for all string instructions that use it
  - SI offset address accesses data by default in the data segment
- Operating in 32-bit mode EDI and ESI registers are used in place of DI and SI.

LODS

- Loads AL, AX, or EAX with data at segment offset address indexed by the SI register.
- A 1 is added to or subtracted from SI for a byte-sized LODS
- A 2 is added or subtracted for a word-sized LODS.
- A 4 is added or subtracted for a doubleword-sized LODS.
- LODSW ➔ AX = DS:[SI]; SI = SI ± 2

The operation of the LODSW instruction if DS=1000H, D=0,11000H=32, 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.
- STOSB (**stores a byte**) stores the byte in AL at the extra segment memory location addressed by DI.
- STOSW (**stores a word**) stores AX in the memory location addressed by DI.
- After the byte (AL), word (AX), or doubleword (EAX) is stored, contents of DI increment or decrement.

STOS with a REP

- The **repeat prefix** (REP) is added to any string data transfer instruction except LODS.
  - 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.
- If CX is loaded with 100 and a REP STOSB instruction executes, the microprocessor automatically repeats the STOSB 100 times.

MOVS

- Transfers a byte, word, or doubleword a data segment addressed by SI to extra segment location addressed by DI
  - pointers are incremented or decremented, as dictated by the direction flag
- Only the source operand (SI), located in the data segment may be overridden so another segment may be used.
- The destination operand (DI) must always be located in the extra segment.
- MOVSB ➔ ES:[DI] = DS:[SI]; DI = DI ± 1; SI = SI ± 1

  (byte transferred)

INS

- Transfers a byte, word, or doubleword of data from an I/O device into the extra segment memory location addressed by the DI register.
  - I/O address is contained in the DX register
- Useful for inputting a block of data from an external I/O device directly into the memory.

- INSW ➔ ES:[DI] = [DX]; DI = DI ± 2 (word transferred)

- Three basic forms of the INS.
- INSB inputs data from an 8-bit I/O device and stores it in a memory location indexed by DI.
- INSW instruction inputs 16-bit I/O data and stores it in a word-sized memory location.
- INSD instruction inputs a doubleword.
- These instructions can be repeated using the REP prefix
  - allows an entire block of input data to be stored in the memory from an I/O device

OUTS

- Transfers a byte, word, or doubleword of data from the data segment memory location address by SI to an I/O device.
  - I/O device addressed by the DX register as with the INS instruction

OUTSB ➔ [DX] = DS:[SI]; SI = SI ± 1 (byte transferred)

MISCELLANEOUS DATA TRANSFER INSTRUCTIONS

XCHG

- Exchanges contents of a register with any other register or memory location.
  - cannot exchange segment registers or memory-to-memory data
- Exchanges are byte-, word-, or doubleword and use any addressing mode except immediate addressing.

- XCHG AL,CL

LAHF and SAHF

- Seldom used bridge instructions (to translate 8085 instructions to 8086).
- LAHF instruction transfers the rightmost 8 bits of the flag register into the AH register.
- SAHF instruction transfers the AH register into the rightmost 8 bits of the flag register.

IN and OUT

- Contents of AL, AX, or EAX are transferred only between I/O device and microprocessor.
  - an IN instruction transfers data from an external I/O device into AL, AX, or EAX
  - an OUT transfers data from AL, AX, or EAX to an external I/O device
- *Fixed-port addressing* allows data transfer between AL, AX, or EAX using an 8-bit I/O port address.
  - port number follows the instruction's opcode
  - IN AL,6AH
  - OUT 19H,AX

IN and OUT

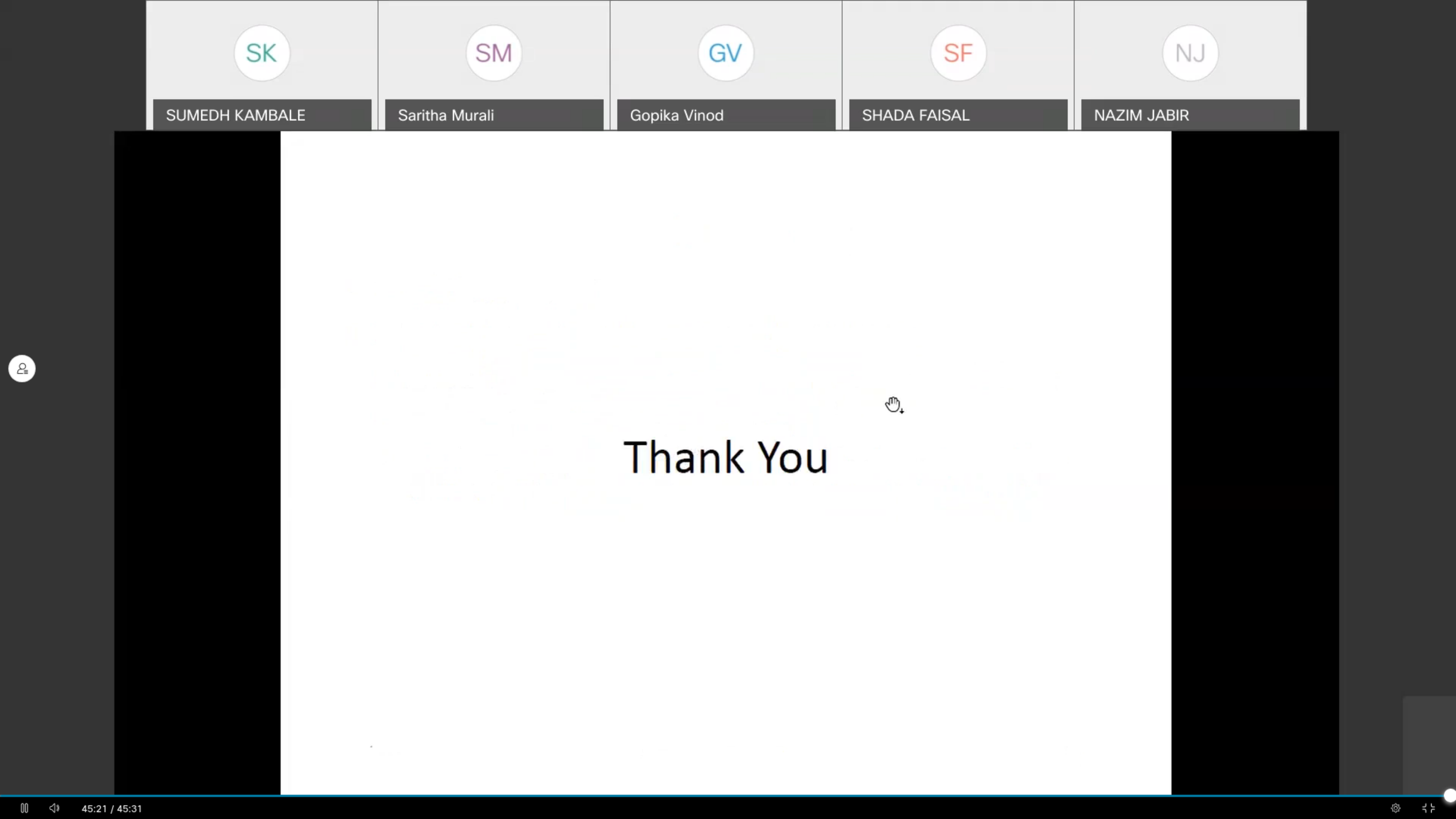
- *Variable-port addressing* allows data transfers between AL, AX, or EAX and a 16-bit port address.
  - the I/O port number is stored in register DX, which can be changed (varied) during the execution of a program.
  - IN AX,DX – 16-bits are input to AX from I/O port DX
  - OUT DX,AX - 16-bits are output from AX to the I/O port DX

BSWAP

- Takes the contents of any 32-bit register and swaps the first byte with the fourth, and the second with the third.
  - BSWAP (**byte swap**) is available only in 80486–Pentium 4 microprocessors
- This instruction is used to convert data between the big and little endian forms.
- In 64-bit operation for the Pentium 4, all 8 bytes in the selected operand are swapped.

CMOV

- Conditional MOV
- Moves the data only if the condition is true
- CMOVZ instruction moves data only if the result from some prior instruction was a zero.
  - destination is limited to only a 16- or 32-bit register, but the source can be a 16- or 32-bit register or memory location


SK
SUMEDH KAMBALE
SM
Saritha Murali
GV
Gopika Vinod
SF
SHADA FAISAL
NJ
NAZIM JABIR
Thank You
45:21 / 45:31