Data Movement Instructions

Shamla Beevi A

Research Scholar, CSED

National Institute of Technology Calicut

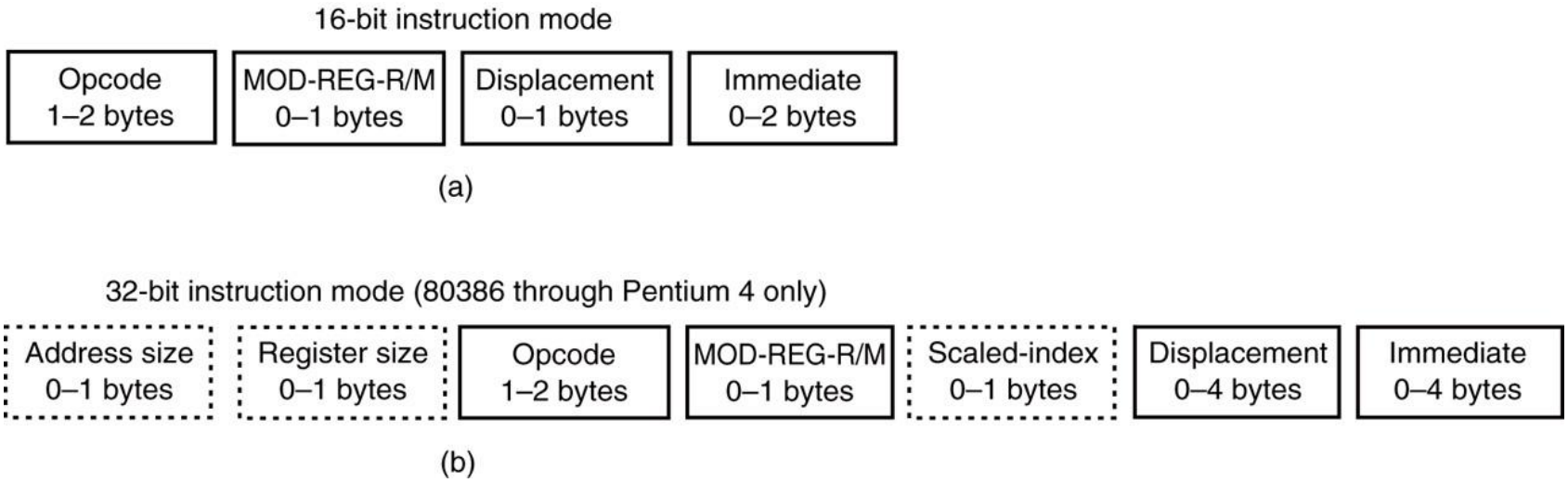
Introduction

- The microprocessor requires an **assembler program**, which generates machine language, because machine language instructions are too complex to efficiently generate by hand.
- 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

- Machine language is the native binary code that the microprocessor understands and uses as its instructions to control its operation.
- Machine language instructions for the 8086 vary in length from **1 to 13 bytes**
- Over 100,000 variations of machine language instructions
- There is no complete list of these variations

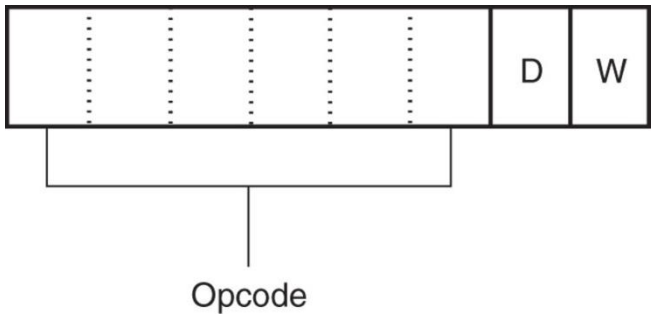
Formats of the 8086–Pentium 4 instructions



- Instructions for the 8086 through the 80286 are 16-bit mode 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)**



•D=0 → Data flows from REG field to R/M field

•D=1 → Data flows from R/M field to REG field

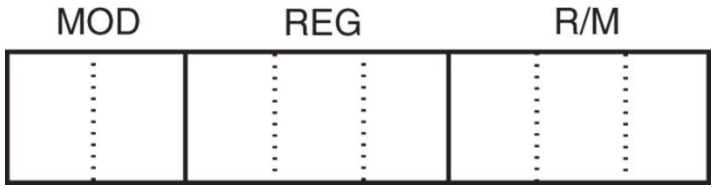
•D bit appears mostly with MOV instructions

•W=1 → Data size is *word* or *doubleword*

•W=0 → Data is always *byte*

MOD Field

- Location of the MOD (mode), REG (register), and R/M (register/memory) fields.
- The MOD field specifies ***addressing mode*** (MOD)for the selected instruction.
- Selects the type of addressing and whether a displacement is present with the selected type.
- Byte 2 of many machine language instructions, showing the position of the MOD, REG, and R/M fields.



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

✓MOV AL,[DI]

✓MOV AL,[DI+2]

✓MOV AL, [DI+1000H]

Register Assignments

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

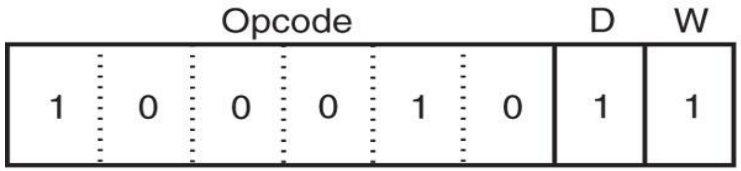
CODE	W=0 (byte)	W=1 (word)	W=1 (Double word)
000	AL	AX	EAX
001	CL	CX	ECX
010	DL	DX	EDX
011	BL	BX	EBX
100	AH	SP	ESP
101	CH	BP	EBP
110	DH	SI	ESI
111	BH	DI	EDI

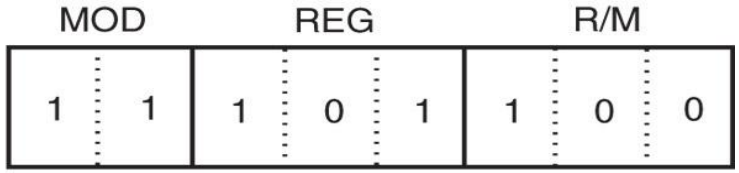
R/M Memory Addressing

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]

Register Assignments

- Suppose a 2-byte instruction, 8BECH in machine language
- 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**
- This instruction moves data from SP into BP and is written in symbolic form as a **MOV BP,SP** instruction

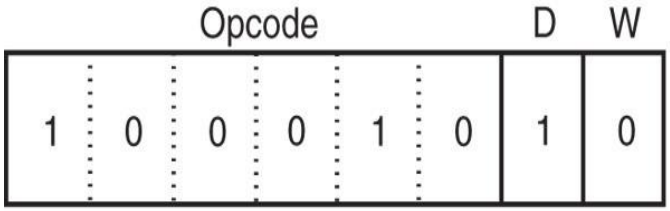


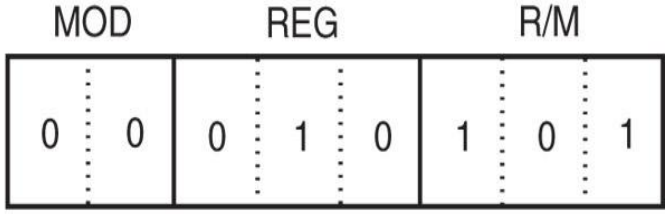


Opcode = MOV
D = Transfer to register (REG)
W = Word
MOD = R/M is a register
REG = BP
R/M = SP

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])

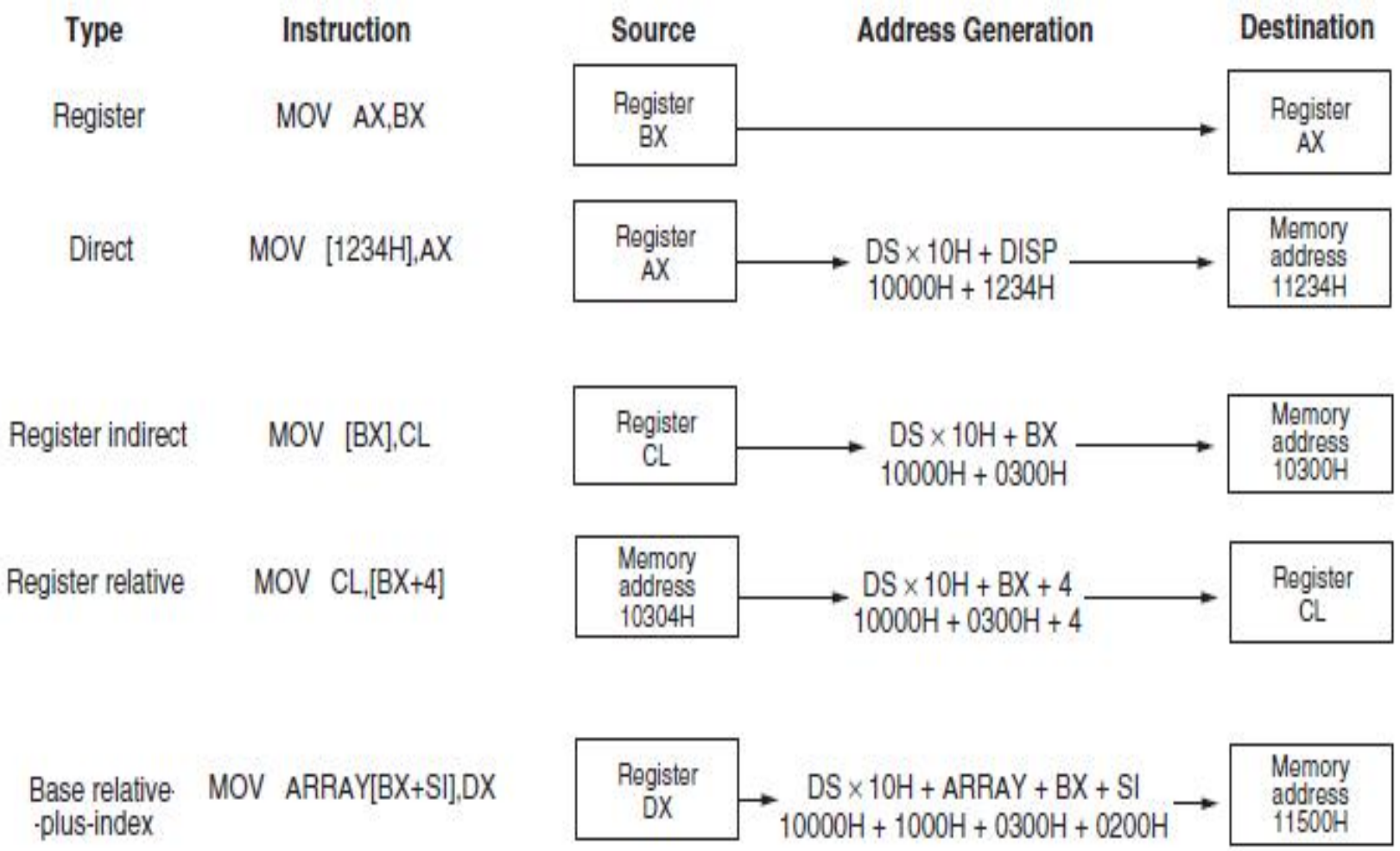




Opcode = MOV
D = Transfer to register (REG)
W = Byte
MOD = No displacement
REG = DL
R/M = DS:[DI]

•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

Common Operand Modifiers

Operator	*Example*	*Comment*
+	MOV AL,6+3	Copies 9 into AL
–	MOV AL,6–3	Copies 3 into AL
*	MOV AL,4*3	Copies 12 into AL
/	MOV AX,12/5	Copies 2 into AX (remainder is lost)
MOD	MOV AX,12 MOD 7	Copies 5 into AX (quotient is lost)
AND	MOV AX,12 AND 4	Copies 4 into AX (1100 AND 0100 = 0100)
OR	MOV EAX,12 OR 1	Copies 13 into EAX (1100 OR 0001 = 1101)
NOT	MOV AL,NOT 1	Copies 254 into AL (NOT 0000 0001 = 1111 1110 or 254)

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(*not available in 8086/8088*).

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

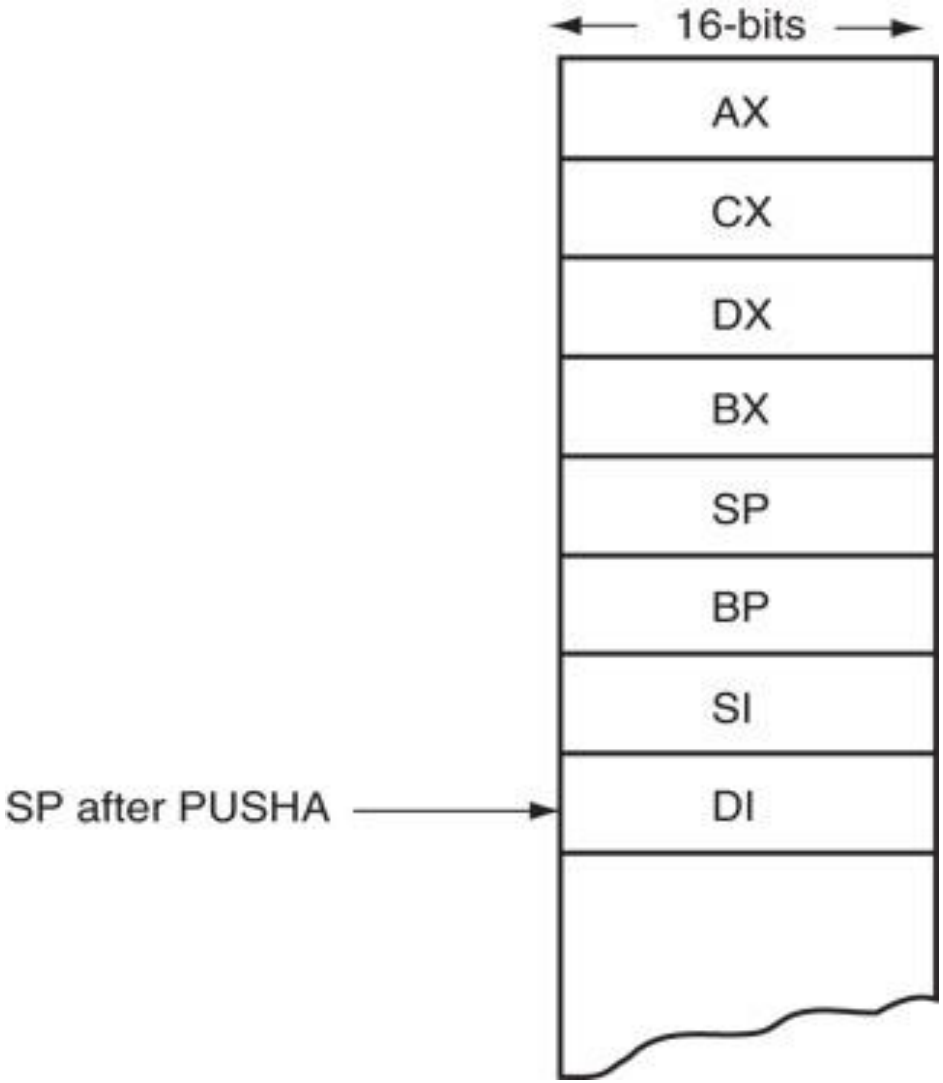
- Segment register addressing allows contents of any segment register to be pushed onto the stack or removed from the stack.
- ES may be pushed, but data from the stack may never be popped into ES
- The flags may be pushed or popped from the stack.
- contents of all registers may be pushed or popped

PUSH

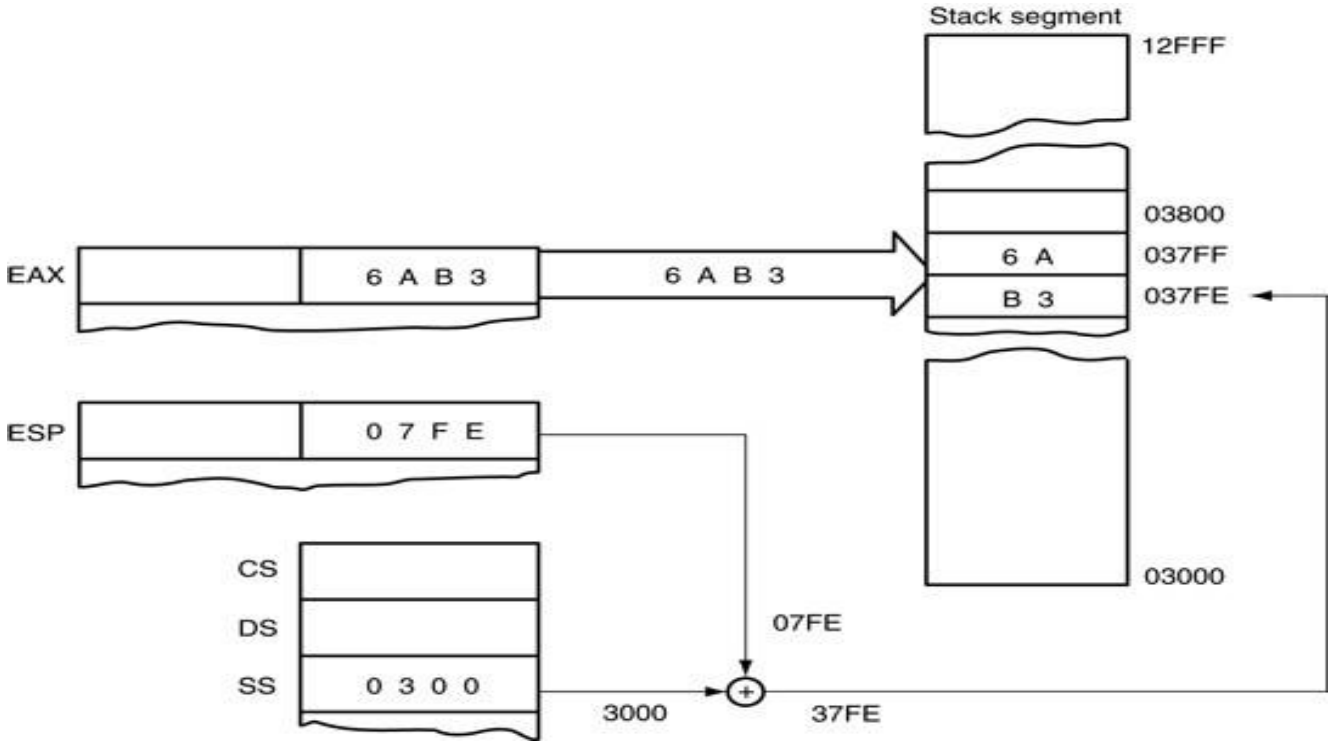
- Always transfers 2 bytes of data to the stack (8086-80286)
  - 80386 and above transfer 2 or 4 bytes
- PUSHA (**push all**) instruction copies the registers to the stack in the following order: AX, CX, DX, BX, SP, BP, SI, and DI.

  [16 bytes = 8 regs x 2 bytes]
- After all registers are pushed, the contents of the SP register is decremented by 16.

- 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.



•This instruction copies the contents of AX onto the stack where address SS:[SP-1]=AH, SS:[SP-2]=AL, and afterwards SP=SP-2 .

•Follows little endian format

PUSH

Symbolic	*Example*	*Note*
PUSH reg16	PUSH BX	16-bit register
PUSH reg32	PUSH EDX	32-bit register
PUSH mem16	PUSH WORD PTR[BX]	16-bit pointer
PUSH mem32	PUSH DWORD PTR[EBX]	32-bit pointer
PUSH mem64	PUSH QWORD PTR[RBX]	64-bit pointer (64-bit mode)
PUSH seg	PUSH DS	Segment register
PUSH imm8	PUSH 'R'	8-bit immediate
PUSH imm16	PUSH 1000H	16-bit immediate
PUSHD imm32	PUSHD 20	32-bit immediate
PUSHA	PUSHA	Save all 16-bit registers
PUSHAD	PUSHAD	Save all 32-bit registers
PUSHF	PUSHF	Save flags
PUSHFD	PUSHFD	Save EFLAGS

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

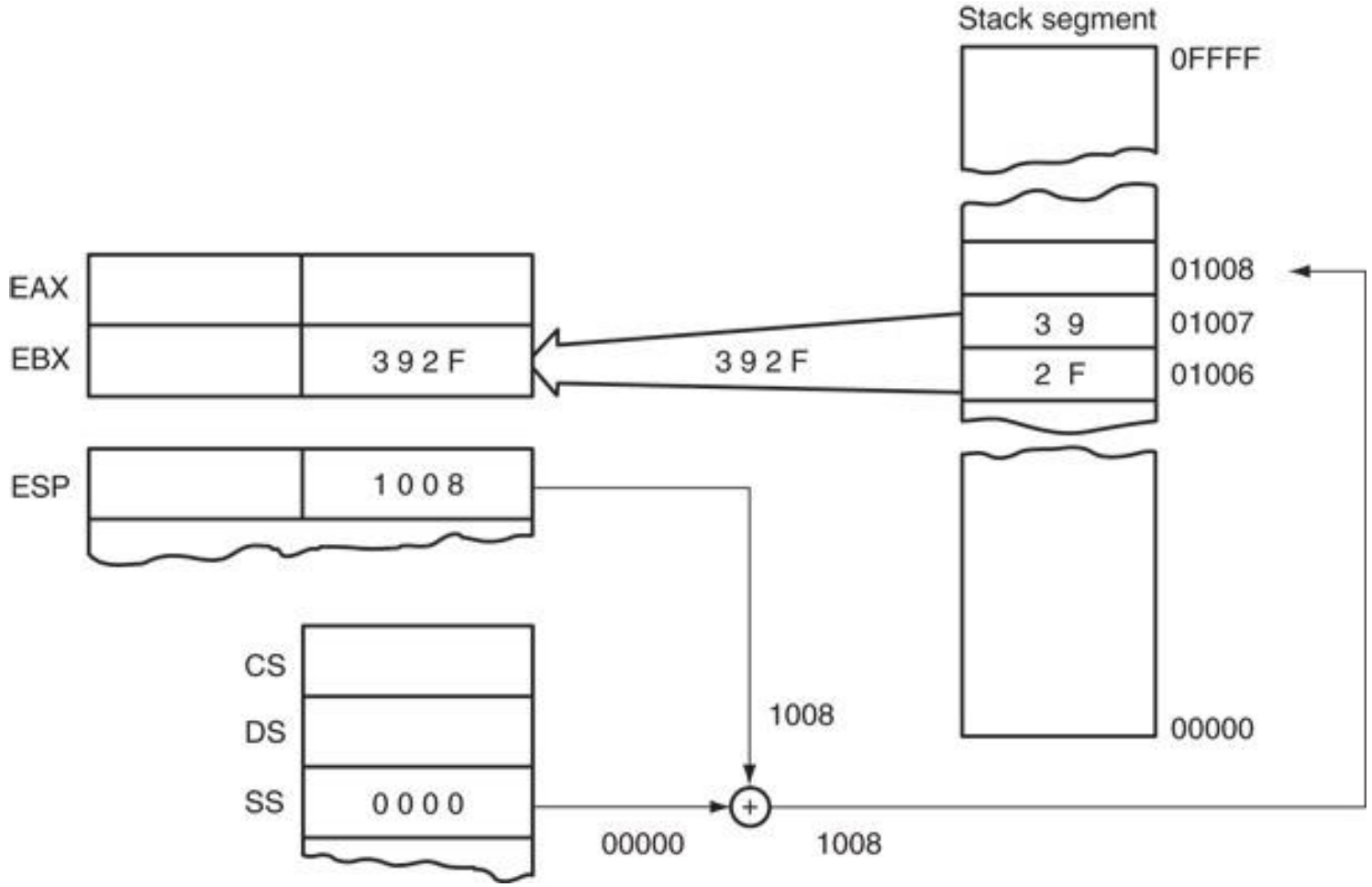
POP

- 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, 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.



POP

Symbolic	*Example*	*Note*
POP reg16	POP CX	16-bit register
POP reg32	POP EBP	32-bit register
POP mem16	POP WORD PTR[BX+1]	16-bit pointer
POP mem32	POP DATA3	32-bit memory address
POP mem64	POP FROG	64-bit memory address (64-bit mode)
POP seg	POP FS	Segment register
POPA	POPA	Pops all 16-bit registers
POPAD	POPAD	Pops all 32-bit registers
POPF	POPF	Pops flags
POPFD	POPFD	Pops EFLAGS

Load Effective Address

- LEA instruction loads any 16-bit register with the offset address, as determined by the addressing mode selected for the instruction.
- 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.

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

- LDS BX,[DI] : This instruction transfers the 32-bit number, addressed by DI in the data segment, into the BX and DS registers.

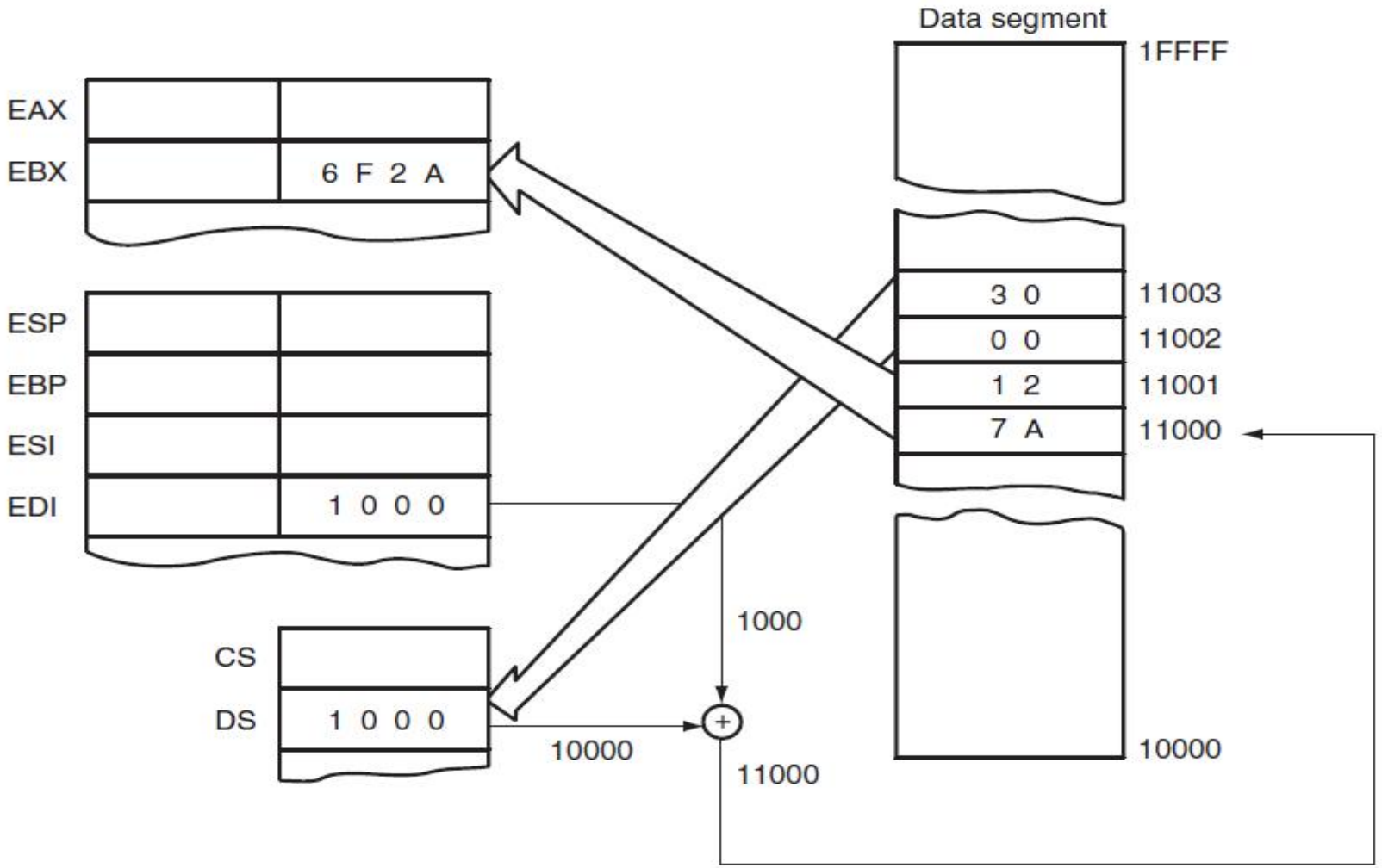


FIGURE 4–17 The LDS BX,[DI] instruction loads register BX from addresses 11000H and 11001H and register DS from locations 11002H and 11003H. This instruction is shown at the point just before DS changes to 3000H and BX changes to 127AH.

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**
- The direction flag (D, located in the flag register) selects the auto-increment(D=0) or the auto-decrement(D=1) operation for the DI and SI registers during string operations.
- D flag is 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.

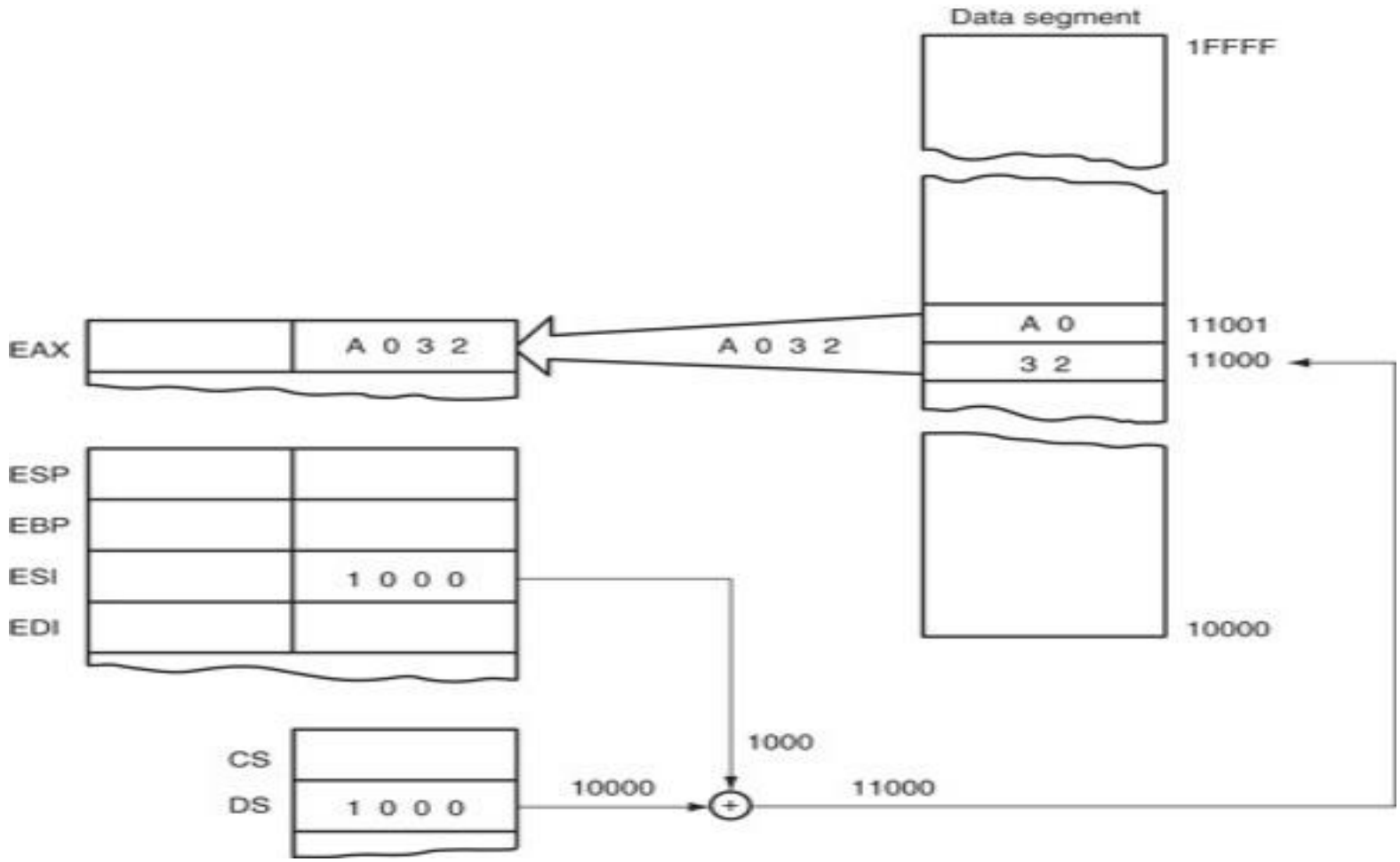
LODS

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

Assembly Language	*Operation*
LODSB	AL = DS:[SI]; SI = SI ± 1
LODSW	AX = DS:[SI]; SI = SI ± 2
LODSD	EAX = DS:[SI]; SI = SI ± 4
LODSQ	RAX = [RSI]; RSI = RSI ± 8 (64-bit mode)
LODS LIST	AL = DS:[SI]; SI = SI ± 1 (if LIST is a byte)
LODS DATA1	AX = DS:[SI]; SI = SI ± 2 (if DATA1 is a word)
LODS FROG	EAX = DS:[SI]; SI = SI ± 4 (if FROG is a doubleword)

Note: The segment register can be overridden with a segment override prefix as in LODS ES:DATA4.

- 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

Assembly Language	*Operation*
STOSB	ES:[DI] = AL; DI = DI ± 1
STOSW	ES:[DI] = AX; DI = DI ± 2
STOSD	ES:[DI] = EAX; DI = DI ± 4
STOSQ	[RDI] = RAX; RDI = RDI ± 8 (64-bit mode)
STOS LIST	ES:[DI] = AL; DI = DI ± 1 (if LIST is a byte)
STOS DATA3	ES:[DI] = AX; DI = DI ± 2 (if DATA3 is a word)
STOS DATA4	ES:[DI] = EAX; DI = DI ± 4 (if DATA4 is a doubleword)

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.

Forms of the MOVS instruction

Assembly Language	*Operation*
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)
MOVSQ	[RDI] = [RSI]; RDI = RDI ± 8; RSI = RSI ± 8 (64-bit mode)
MOVS BYTE1, BYTE2	ES:[DI] = DS:[SI]; DI = DI ± 1; SI = SI ± 1 (byte transferred if BYTE1 and BYTE2 are bytes)
MOVS WORD1,WORD2	ES:[DI] = DS:[SI]; DI = DI ± 2; SI = SI ± 2 (word transferred if WORD1 and WORD2 are words)
MOVS TED,FRED	ES:[DI] = DS:[SI]; DI = DI ± 4; SI = SI ± 4 (doubleword transferred if TED and FRED are doublewords)

INS

- Transfers a **byte, word,** or **doubleword** of data from an **I/O device** into the extra segment(ES) 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.

Assembly Language	*Operation*
INSB	ES:[DI] = [DX]; DI = DI ± 1 (byte transferred)
INSW	ES:[DI] = [DX]; DI = DI ± 2 (word transferred)
INSD	ES:[DI] = [DX]; DI = DI ± 4 (doubleword transferred)
INS LIST	ES:[DI] = [DX]; DI = DI ± 1 (if LIST is a byte)
INS DATA4	ES:[DI] = [DX]; DI = DI ± 2 (if DATA4 is a word)
INS DATA5	ES:[DI] = [DX]; DI = DI ± 4 (if DATA5 is a doubleword)

Note: [DX] indicates that DX is the I/O device address. These instructions are not available on the 8086 and 8088 microprocessors.

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

Assembly Language	*Operation*
OUTSB	[DX] = DS:[SI]; SI = SI ± 1 (byte transferred)
OUTSW	[DX] = DS:[SI]; SI = SI ± 2 (word transferred)
OUTSD	[DX] = DS:[SI]; SI = SI ± 4 (doubleword transferred)
OUTS DATA7	[DX] = DS:[SI]; SI = SI ± 1 (if DATA7 is a byte)
OUTS DATA8	[DX] = DS:[SI]; SI = SI ± 2 (if DATA8 is a word)
OUTS DATA9	[DX] = DS:[SI]; SI = SI ± 4 (if DATA9 is a doubleword)

Note: [DX] indicates that DX is the I/O device address. These instructions are not available on the 8086 and 8088 microprocessors.

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**.

Assembly Language	*Operation*
XCHG AL,CL	Exchanges the contents of AL with CL
XCHG CX,BP	Exchanges the contents of CX with BP
XCHG EDX,ESI	Exchanges the contents of EDX with ESI
XCHG AL,DATA2	Exchanges the contents of AL with data segment memory location DATA2
XCHG RBX,RCX	Exchange the contents of RBX with RCX (64-bit mode)

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
- ***2 forms of I/O device(port) addressing***
  - ***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

Assembly Language	*Operation*
IN AL,p8	8 bits are input to AL from I/O port p8
IN AX,p8	16 bits are input to AX from I/O port p8
IN EAX,p8	32 bits are input to EAX from I/O port p8
IN AL,DX	8 bits are input to AL from I/O port DX
IN AX,DX	16 bits are input to AX from I/O port DX
IN EAX,DX	32 bits are input to EAX from I/O port DX
OUT p8,AL	8 bits are output to I/O port p8 from AL
OUT p8,AX	16 bits are output to I/O port p8 from AX
OUT p8,EAX	32 bits are output to I/O port p8 from EAX
OUT DX,AL	8 bits are output to I/O port DX from AL
OUT DX,AX	16 bits are output to I/O port DX from AX
OUT DX,EAX	32 bits are output to I/O port DX from EAX

Note: p8 = an 8-bit I/O port number (0000H to 00FFH) and DX = the 16-bit I/O port number (0000H to FFFFH) held in register DX.

CMOV

- CMOV (**conditional move)**
- These instructions move the data only if the condition is true.

Assembly Language	*Flag(s) Tested*	*Operation*
CMOVB	C = 1	Move if below
CMOVAE	C = 0	Move if above or equal
CMOVBE	Z = 1 or C = 1	Move if below or equal
CMOVA	Z = 0 and C = 0	Move of above
CMOVE or CMOVZ	Z = 1	Move if equal or move if zero
CMOVNE or CMOVNZ	Z = 0	Move if not equal or move if not zero
CMOVL	S ! = O	Move if less than
CMOVLE	Z = 1 or S ! = O	Move if less than or equal
CMOVG	Z = 0 and S = O	Move if greater than
CMOVGE	S = O	Move if greater than or equal
CMOVS	S = 1	Move if sign (negative)
CMOVNS	S = 0	Move if no sign (positive)
CMOVC	C = 1	Move if carry
CMOVNC	C = 0	Move if no carry
CMOVO	O = 1	Move if overflow
CMOVNO	O = 0	Move if no overflow
CMOVP or CMOVPE	P = 1	Move if parity or move if parity even
CMOVNP or CMOVPO	P = 0	Move if no parity or move if parity odd

Thank you!!!!!!!!!!!!!!!