Data-Transfer Instructions

- The data-transfer functions provide the ability to move data either between its internal registers or between an internal register and a storage location in memory.
- The data-transfer functions include
  - MOV (Move byte or word)
  - XCHG (Exchange byte or word)
  - XLAT (Translate byte)
  - LEA (Load effective address)
  - LDS (Load data segment)
  - LES (Load extra segment)

Data Transfer Instructions - MOV

Mnemonic	Meaning	Format	Operation	Flags Affected
MOV	Move	MOV D, S	(D) →(S)	None

Destination	Source
Memory	Accumulator
Accumulator	Memory
Register	Register
Register	Memory
Memory	Register
Register	Immediate
Memory	Immediate
Seg reg	Reg16
Seg reg	Mem16
Reg 16	Seg reg
Memory	Seg reg

Memory to memory is not allowed

Data Transfer Instructions - XCHG

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

Destination	Source
Accumulator	Reg16
Memory	Register
Register	Register
Register	Memory

Example. XCHG [1234h], BX

Data Transfer Instructions – LEA, LDS, LES

An important type of data transfer operation is loading a segment and a general purpose register with an address directly from memory

Mnemonic	Meaning	Format	Operation	Flags Affected
LEA	Load Effective Address	LEA Reg16,EA	EA → (Reg16)	None
LDS	Load Register and DS	LDS Reg16, MEM32	(Mem32) → (Reg16) (Mem32 + 2) → (DS)	None
LES	Load Register and ES	LES Reg16, MEM32	(Mem32) → (Reg16) (Mem32 + 2) → (ES)	None

Example. LEA SI, DATA or MOV SI, OFFSET DATA

The XLAT Instruction

The translate (XLAT) instruction is used to simplify implementation of the lookup-table operation. Execution of the XLAT replaces the contents of AL by the contents of the accessed lookup-table location.

Mnemonic	Meaning	Format	Operation	Flags affected
XLAT	Translate	XLAT	((AL)+(BX)+(DS)0) → (AL)	None

e.g. PA = (DS)0 + (BX) + (AL)

$= 03000_{16} + 0100_{16} + 0D_{16} = 0310D_{16}$

$(0310D_{16}) \rightarrow$ **(AL)**

The LEA, LDS, and LES Instructions

The LEA, LDS, LES instructions provide the ability to manipulate memory addresses by loading either a 16-bit offset address into a general-purpose register or a register together with a 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, Mem32	(Mem32) →(Reg16) (Mem32+2) →(DS)	None
LES	Load register and ES	LES Reg16,Mem32	(Mem32) →(Reg16) (Mem32+2) →(ES)	None

e.g. LEA SI, [DI+BX+5H]

XLAT :Traslate instruction

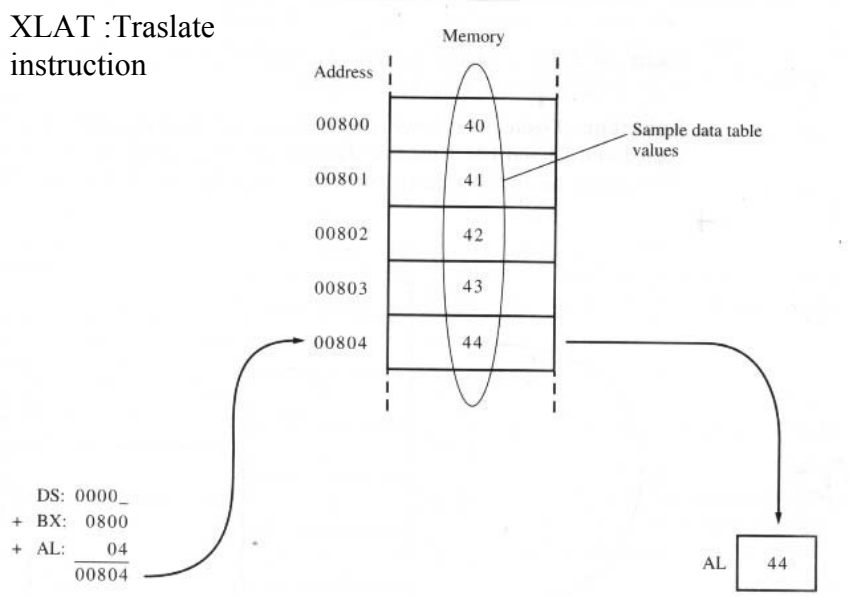


FIGURE 3.22 Execution of XLAT

LDS : Load Data Segment....

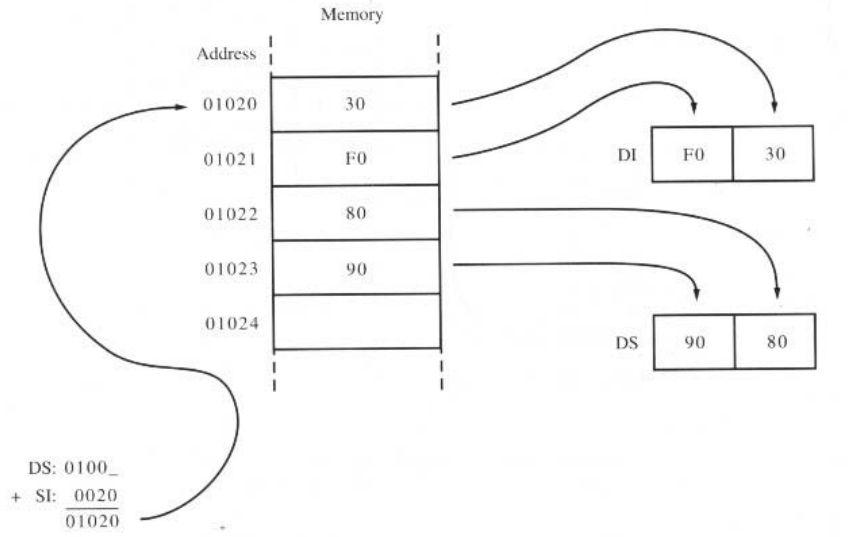
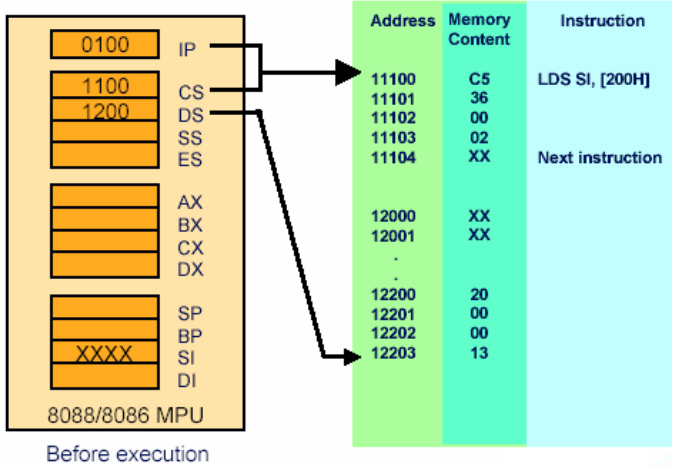


FIGURE 3.23 Execution of LDS DI,[SI]

The LEA, LDS, and LES Instructions

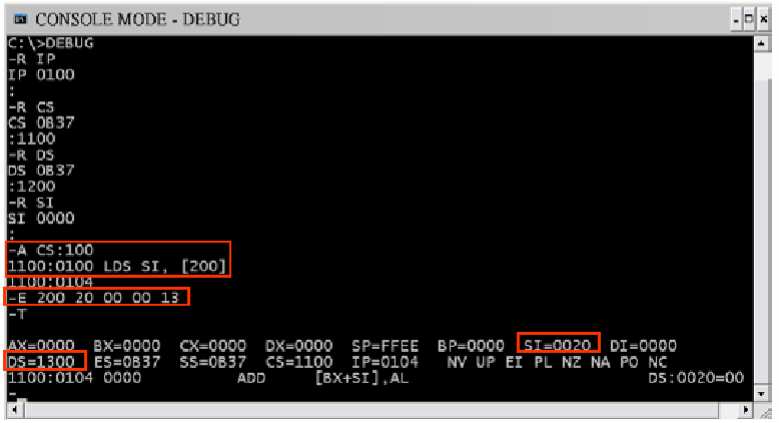
LDS SI, [200H]



EXAMPLE

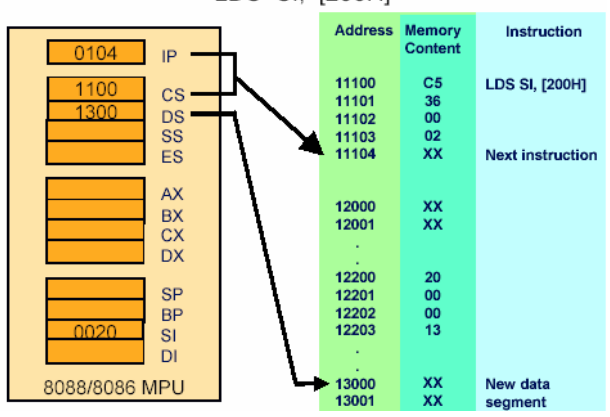
Verify the following instruction using DEBUG program.

LDS SI, [200H]



- The LEA, LDS, and LES Instructions

LDS SI, [200H]



After..

Arithmetic

- Addition ;Subtraction; Multiplication; Division

Arithmetic Instruction Summary

```
add     ax,   bx              ;ax<-ax+bx and set flags
adc     ax,   bx              ;ax<-ax+bx+CF(lsb) and set flags
inc     ax                    ;ax<-ax+1 and set flags
aaa                           ;ASCII Adjust after Addition
daa                           ;Decimal (BCD) Adjust after Addition
sub     ax,   bx              ;ax<-ax-bx and set flags
sbb     ax,   bx              ;ax<-(ax-CF)-bx and set flags
dec     ax                    ;ax<-ax-1
neg     ax                    ;ax<-(-1)*(ax) - 2's Complement
cmp     ax,   bx              ;ZF is set according to ax-bx
das                           ;Decimal (BCD) Adjust after Subtraction
aas                           ;ASCII Adjust after Subtraction
mul     cx                    ;dx:ax<- ax * cx (unsigned)
imul    cx                    ;dx:ax<- ax * cx (2's complement)
aam                           ;ASCII Adjust after Multiplication
div     cl                    ;al<-ax/cl Quot. AND ah<-ax/cl Rem.
idiv    cx                    ;ax<-(dx:ax)/ax Quot. AND dx <- Rem.
aad                           ;ASCII Adjust after Division
```

Decimal arithmetic

- Many computers can do decimal arithmetic.
- Storing decimal numbers 12345
  - ASCII format: 31 32 33 34 35
  - unpacked BCD: 01 02 03 04 05
  - packed decimal: 01 23 45

Arithmetic Instructions – ADD, ADC, INC, AAA, DAA

Mnemonic	Meaning	Format	Operation	Flags Affected
ADD	Addition	ADD D, S	(S) + (D) → (D) Carry → (CF)	All
ADC	Add with carry	ADC D, S	(S) + (D) + (CF) → (D) Carry → (CF)	All
INC	Increment by one	INC D	(D) + 1 → (D)	All but the carry flag
AAA	ASCII adjust for addition	AAA		
DAA	Decimal adjust for addition	DAA		

Decimal arithmetic

- AAA (adjust after addition)
  ASCII, unpacked decimal
- AAS (adjust after subtraction)
  ASCII, unpacked decimal
- AAM (adjust after multiplication)
  unpacked decimal
- AAD (adjust before division)
  unpacked decimal
- DAA (decimal adjust after addition)
  packed decimal
- DAS (decimal adjust after subtraction)
  packed decimal

AAA: ASCII adjust for addition

- Add ASCII numbers and if the result is more than 9, it will convert it and pass the extra bit to AH and the CF
- Example 1:

```
MOV AL, '5'     ; AL = 35H          0011 0101
MOV BL, '2'     ; BL = 32H          0011 0010
ADD AL, BL      ; AL = 67H          0110 0111
AAA             ; Change 67 → 07 in AL
OR  AL, 30      ; AL = 37 → ASCII for 7
```

- Example 2:

```
MOV AL, '7'     ; AL =37H           0011 0111
MOV BL, '5'     ; BL = 35H          0011 0101
ADD AL, BL      ; AL = 6CH          0110 1100
AAA             ; Change 6C → 12 where
                ; CF = AH = 01 and AL = 02
OR  AX, 3030    ; AH = 31 → ASCII for 1
                ; and AL = 32 → ASCII for 2
```

ASCII Adjust after Addition

- **For Addition Using ASCII Encoded Numbers**

  `30h` **through** `39h` **Represent** `'0'` **through** `'9'`

- `ax` **is Default Source and Destination for** `aaa`

```
  31   '1'
 +39   '9'
  6a   '10' →should be 3130h
```

(6ah is incorrect ASCII result 'j')

```
mov   ax,   31h   ;ax<--0031h='1'
add   al,   39h   ;ax<--31h+39h=006ah='<nul>j'
aaa               ;ax<--0100h (this is BCD of result)
add   ax,   3030h ;Convert from BCD to ASCII
                  ;ax<--0100h+3030h=3130h='10'
```

- **Example**

```
mov  ah, 0          ;  AX = 00 ??
mov  al, "9"    ;  AX = 00 39
add  al, "3"    ;  AX = 00 6C
aaa             ;  AX = 01 02
or   ax, 3030h ;  AX = 31 32
```

- **We have the answer in ASCII decimal**
- **Slower than normal binary adds**
- **May be faster than converting ASCII decimal to binary, adding, and converting back to ASCII decimal**

DAA: Decimal adjust for addition (Packed BCD)

- If after an ADD or ADC instruction the lower nibble is greater than 9, or if AF =1, add 0110 to the lower 4 bits
- If the upper nibble is greater than 9, or if CF=1, add 0110 to the upper nibble
- Example:

```
MOV AL, 47H    ; first BCD          0100 0111
MOV BL, 25H    ; Second BCD         0011 0010
ADD AL, BL     ; AL = 3CH           0110 0111 > 9
DAA            ; Adjust BCD     +       0110
               ; AL = 72 H          0111 0010
```

- Example

Instructions	Before Operation	After Operation
MOV AX, 0034H	EFLAGS(OSZAPC)=xxxxxx	EFLAGS(OSZAPC)=xxxxxx
MOV BL, 39H	AX=xxxx BL=xx	AX=0034H BL=39H
ADD AL, BL	EFLAGS(OSZAPC)=xxxxxx	EFLAGS(OSZAPC)=**000000**
	AX=0034H BL=39H	AX=006DH BL=39H
AAA	EFLAGS(OSZAPC)=**000000**	EFLAGS(OSZAPC)=**000101**
	AX=006DH BL=37H	AX=0103H BL=39H

Instructions	Before Operation	After Operation
ADD AL, BL	EFLAGS(OSZAPC)=xxxxxx	EFLAGS(OSZAPC)=**110000**
	AL=79H BL=35H	AL=AEH BL=35H
DAA	EFLAGS(OSZAPC)=**110000**	EFLAGS(OSZAPC)=**X00111**
	AL=AEH BL=35H	AL=14H BL=35H

Arithmetic Instructions – SUB, SBB, DEC, AAS, DAS, NEG

Mnemonic	Meaning	Format	Operation	Flags Affected
SUB	Subtract	SUB D, S	(D) - (S) → (D) Borrow → (CF)	All
SBB	Subtract with borrow	SBB D, S	(D) - (S) - (CF) → (D)	All
DEC	Decrement by one	INC D	(D) - 1 → (D)	All but the carry flag
NEG	Negate	NEG D		
DAS	Decimal adjust for subtraction	DAS		
AAS	ASCII adjust for subtraction	AAS		

2. Subtraction

SUB: Unsigned subtraction → done in 2's complement

CF indicated the sign of the result ….. if CF= 1 it is negative

```
Example:  MOV   DH, 4CH   ; DH = 4C    = 0100 1100                0100 1100
          MOV   BH, 6EH   ; BH = 6E    = 0110 1110   2's comp  + 1001 0010
          SUB   DH, BH    ; DH = E7 and CF = 1       2's comp    1101 1110
```

Note: To take the 2's complement of the result use the NEG instruction

neg Two's complement negate Syntax: neg op

register or memory Action: op = 0 - op

Flags Affected: OF, SF, ZF, AF, PF, CF

AAS: ASCII adjust for subtraction

- Works only on AL
- Function similar to AAA
- Example:

```
MOV BH, 00          ; Clear BH
MOV  AX, '105'      ; AX = 3135
MOV  BL, '06'       ; BL = 36
SUB  AL, BL         ; AL                    0011 0001 0011 0101
                    ; BL (2's comp) +       1111 1111 1100 1010
                                            ---------------------
                                            0011 0000 1111 1111
                    ; AX = 31FF (2's complement)
AAS                 ; AH= 00 and AL = 09
```

Multiplication and Division

Multiplication (MUL or IMUL)	Multiplicant	Operand (Multiplier)	Result
Byte * Byte	AL	Register or memory	AX
Word * Word	AX	Register or memory	DX :AX
Dword * Dword	EAX	Register or Memory	EDX :EAX

Division (DIV or IDIV)	Dividend	Operand (Divisor)	Quotient : Remainder
Word / Byte	AX	Register or memory	AL : AH
Dword / Word	DX:AX	Register or memory	AX : DX
Qword / Dword	EDX: EAX	Register or Memory	EAX : EDX

3. Unsigned Multiplication [MUL]:

Multiplication	Operand 1	Operand 2	Result
byte x byte	AL	Register or memory	AX
word x word	Ax	Register or memory	DX AX
word x byte	AL = byte, AH=0	Register or memory	DX AX

4. Unsigned Division [DIV]:

Multiplication	Operand 1	Operand 2	Quotient	Rem
byte / byte	AL = byte, HA = 0	Register or memory	AL	AH
word / word	AX = word, DX = 0	Register or memory	AX	DX
word / byte	AX = word	Register or memory	AL	AH
dw / word	DX AX = dw	Register or memory	AX	DX

5. Signed Multiplication [IMUL]:

Multiplication	Operand 1	Operand 2	Result
byte x byte	AL	Register or memory	AX
word x word	Ax	Register or memory	DX AX
word x byte	AL = byte, CBW	Register or memory	DX AX

6. Signed Division [IDIV]:

Multiplication	Operand 1	Operand 2	Quotient	Rem
byte / byte	AL = byte, CBW	Register or memory	AL	AH
word / word	AX = word, CWD	Register or memory	AX	DX
word / byte	AX = word	Register or memory	AL	AH
dw / word	DX AX = dw	Register or memory	AX	DX

```
mov BL,0FEh
mov AL,0E5h
mul BL
```

- BL contains 254, AL contains 229
- After MUL
  - AX contains 0E336h (58166d)
  - CF=OF=1 (result requires a word)
  - SF, AF, PF, and ZF are undefined

```
mov BL,0FEh
mov AL,0E5h
imul BL
```

- BL contains -2, AL contains -27
- After IMUL
  - AX contains 0036h (54d)
  - CF=OF=0 (result still fits in byte)
  - SF, AF, PF, and ZF are undefined

AL=20 H……..32

CL= 80 H…….128

MUL CL ……AX= 1000

IMUL CL ……AX=F000

AL=95

BL=10

DIV BL

AL=09 ; AH=05

AX=2711 H……10001 d

BL=32 H……..50 d

Div BL….. …….. AX=01C8

Examples of MUL and IMUL

- Say that AX = 1h and BX = FFFFh, then:
  - Instruction Result DX AX CF/OF

Instruction	Result	DX	AX	CF/OF
mul bx	655350000	FFFF	0	
imul bx	-1	FFFF	FFFF	0

- Say that AX = FFFFh and BX = FFFFh, then:
  - Instruction Result DX AX CF/OF

Instruction	Result	DX	AX	CF/OF
mul bx	4294836225	FFFE	0001	1
imul bx	1	0000	0001	0

Examples of MUL and IMUL (cont.)

- AL = 30h and BL = 4h, then:
  - Instruction ResultAH AL CF/OF

Instruction	ResultAH	AL	CF/OF	
mul bl	192	00	C0	0
imul bl	192	00	C0	1

- AL = 80h and BL = FFh, then
  - Instruction ResultAH AL CF/OF

Instruction	ResultAH	AL	CF/OF	
mul bl	326407F	80	1	
imul bl	128	00	80	1

Examples of DIV and IDIV

- DX = 0000h, AX = 0005h, BX = FFFEh:
  - Instruction Quot. Rem. AX DX

div bx	0	5	0000	0005
idiv bx	-2	1	FFFE	0001

- DX = FFFFh, AX = FFFBh, BX = 0002h:
  - Instruction Quot. Rem. AX DX

div bx	Divide Overflow			
idiv bx	-2	-1	FFFE	FFFF

- Example 1

  Before the instruction executes:

  DX:???? AX: 0002 BX: 0010 (16) CF/OF:?

  IMUL BX multiplies the contents of AX by BX
  placing the 32 bit answer in DX:AX

  After the instruction executes:

  DX:0000 AX: 0020 (32) BX: 0010 CF/OF:0 (Result fits in AX)

- Example 2

  Before the instruction executes:

  DX:???? AX: 0003 BX: 7000 (28672) CF/OF:?

  IMUL BX multiplies the contents of AX by BX
  placing the 32 bit answer in DX:AX

  After the instruction executes:

  DX:0001 AX: 5000 BX: 7000 CF/OF:1

  (Result [86016] does not fit in AX)

CBW Instruction

- This instruction converts the signed 8-bit number in AL into a signed 16-bit number in AX
- *NOTE: Use this instruction only with signed arithmetic!!*
- Syntax: CBW

  If bit 7 of AL is a 1, the instruction will place FF in AH.

  If bit 7 of AL is a 0, the instruction will place 00 in AH.

  No flags are affected by this instruction.
- Examples:

  Before: AX:??40 (64)

  CBW converts the signed 8-bit number in AL to a 16-bit signed number in AX

  After: AX:0040 (64)

What is the result of executing the following instructions?

```
MOV  AL, 0A1H
CBW
CWD
```

Solution:

$(AL) = A1_{16} = 10100001_2$

Executing the CBW instruction extends the MSB of AL

$(AH) = 11111111_2 = FF_{16}$ or $(AX) = 1111111110100001_2$

Executing the CWD instruction, we get

$(DX) = 1111111111111111_2 = FFFF_{16}$

That is, $(AX) = FFA1_{16}$ $(DX) = FFFF_{16}$

Compare (CMP)

Write a program to find the highest among 5 grades and write it in DL

```
DATA      DB       51, 44, 99, 88, 80          ; 13h,2ch,63h,58h,50h
          MOV      CX,5                        ;set up loop counter
          MOV      BX, OFFSET DATA  ;BX points to GRADE data
          SUB      AL,AL                       ;AL holds highest grade found so far
AGAIN:    CMP      AL,[BX]                     ;compare next grade to highest
          JA       NEXT                        ;jump if AL still highest
          MOV      AL,[BX]                     ;else AL holds new highest
NEXT:     INC      BX                          ;point to next grade
          LOOP AGAIN                           ;continue search
          MOV DH, AL
```