

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

21

LDS : Load Data Segment....

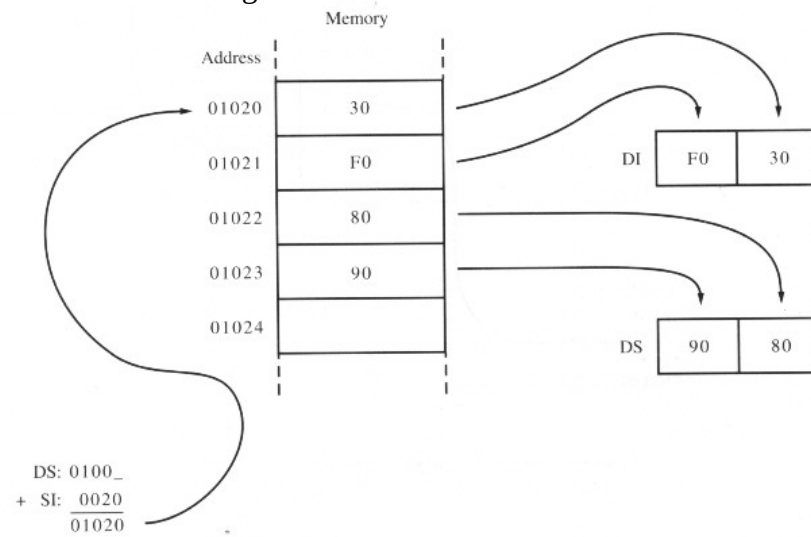
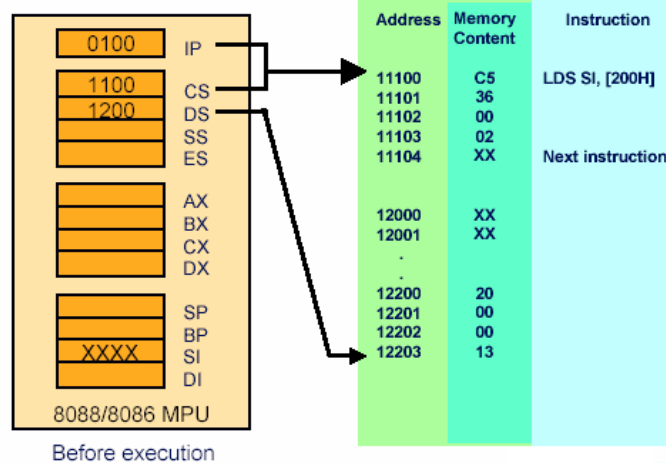


FIGURE 3.23 Execution of LDS DI,[SI]

■ The LEA, LDS, and LES Instructions

LDS SI, [200H]



EXAMPLE

Verify the following instruction using DEBUG program.

LDS SI, [200H]

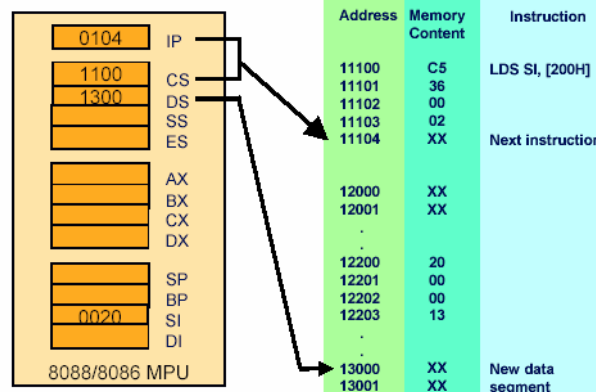
```

CONSOLE MODE - DEBUG
C:\>DEBUG
-R IP
IP 0100
:
-R CS
CS 0837
:1100
-R DS
DS 0837
:1200
-R SI
SI 0000
:
-A CS:100
1100:0100 LDS SI, [200]
1100:0102
-E 200 20 00 00 13
-T
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0020 DI=0000
DS=1300 ES=0837 SS=0837 CS=1100 IP=0104 NV UP EI PL NZ NA PO NC
1100:0104 0000 ADD [BX+SI],AL DS:0020=00

```

■ The LEA, LDS, and LES Instructions

LDS SI, [200H]



After..

■ 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	$((AL)+(BX)+(DS)0) \rightarrow (AL)$	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	$EA \rightarrow (Reg16)$	None
LDS	Load register and DS	LDS Reg16, Mem32	$(Mem32) \rightarrow (Reg16)$ $(Mem32+2) \rightarrow (DS)$	None
LES	Load register and ES	LES Reg16, Mem32	$(Mem32) \rightarrow (Reg16)$ $(Mem32+2) \rightarrow (ES)$	None

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

XLAT :Traslate instruction

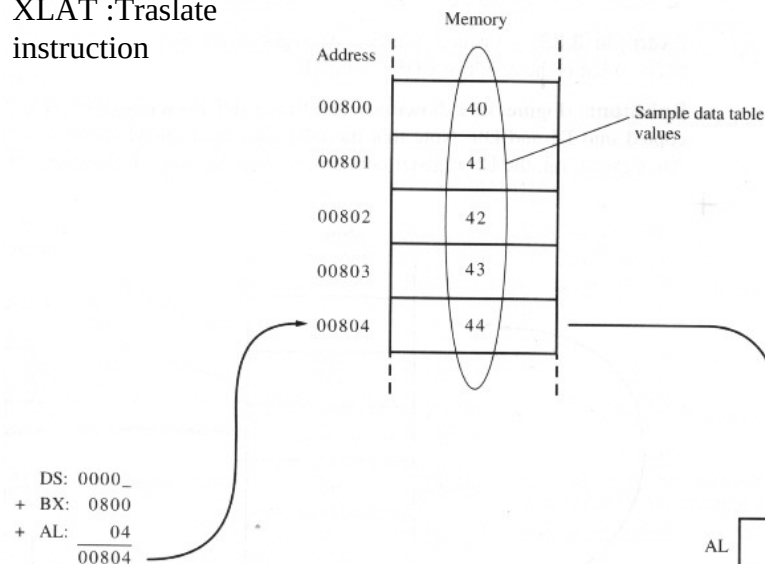


FIGURE 3.22 Execution of XLAT

Arithmetic : Addition ; Subtraction; Multiplication; Division	Decimal arithmetic AAA (adjust after addition) ASCII, unpacked decimal AAS (adjust after subtraction) ASCII, unpacked decimal AAM (adjust after multip..) unpacked decimal AAD (adjust <u>before</u> division) unpacked decimal DAA (adjust after addition) packed decimal DAS (adjust after subtraction) packed decimal
<pre> add ax, bx adc ax, bx inc ax aaa daa sub ax, bx sbb ax, bx dec ax neg ax cmp ax, bx das aas mul cx imul cx aam div cl idiv cx aad </pre>	
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		

Flag Register and ADD instruction

Ex: Show how the flag register is affected by the addition of 38H and 2FH.

Solution: MOV BH,38H ;BH=38H
 ADD BH,2FH ;BH = BH + 2F = 38 + 2F= 67H

38	0011	1000
+ 2F	0010	1111
67	0110	0111

CF = 0 since there is no carry beyond d7

PF = 0 since there is odd number of 1's in the result

AF = 1 since there is a carry from d3 to d4

ZF = 0 since the result is not zero

SF = 0 since d7 of the result is zero

Ex: Show how the flag register is affected by the following addition

Solution: MOV AX,34F5H ;AX =34F5H
 ADD AX,95EBH ;AX = CAE0H

34F5	0011	0100	1111	0101
+ 95EB	1001	0101	1110	1011
CAE0	1100	1010	1110	0000

CF = 0 since there is no carry beyond d15

PF = 0 since there is odd number of 1s in the lower byte

AF = 1 since there is a carry from d3 to d4

ZF = 0 since the result is not zero

SF = 1 since d15 of the result is 1

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	<u>0010 0101</u>
ADD AL, BL	; AL = 3CH	0110 1100 > 9
DAA	; Adjust BCD	+ <u>0110</u>
	; 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 AX=0034H BL=39H	EFLAGS(OSZAPC)= 000000 AX=006DH BL=39H
AAA	EFLAGS(OSZAPC)= 000000 AX=006DH BL=37H	EFLAGS(OSZAPC)= 000101 AX=0103H BL=39H
Instructions	Before Operation	After Operation
ADD AL, BL	EFLAGS(OSZAPC)=xxxxxx AL=79H BL=35H	EFLAGS(OSZAPC)= 110000 AL=AEH BL=35H
DAA	EFLAGS(OSZAPC)= 110000 AL=AEH BL=35H	EFLAGS(OSZAPC)= X00111 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		

Subtraction:

1. Take the 2's complement of the subtrahend (source operand)
2. Add it to the minuend (destination operand)
3. Invert the carry

```
MOV AL,3FH           ; load AL=3FH
MOV BH,23H           ; load BH=23H
SUB AL,BH             ; subtract BH from AL. Place result in AL
```

AL	3F	0011 1111	0011 1111	
-BH	-23	-0010 0011	+1100 0001	(2's complement)
	1C	0001 1100	1 0001 1100	(CF=0) Step 3

CF=0, ZF=0, PF=0, SF=0.

If the CF=0, the result is positive and the destination has the result.
 If the CF=1, the result is negative and the destination has the 2's complement of the result. NOT
 and INC increment instructions can be used to change it.

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

Ex:

;from the data segment:

DATA1 DB 4CH

DATA2 DB 6EH

RESULT DB ?

;from the code segment:

MOV DH,DATA1

;load DH with DATA1 value (4CH)

SUB DH,DATA2

;subtract DATA2 (6E) from DH (4C)

JNC NEXT

;if CF=0 jump to NEXT target

NOT DH

;if CF=1 take the 1's complement

INC DH

;and increment to get 2's complement

NEXT : MOV RESULT,DH

;save DH in RESULT

Analysis: Following the 3 steps for "SUB DH,DATA2"

4C	0100	1100		0100	1100	
- 6E	0110	1110	2's comp	+ 1001	0010	
- 22				0	1101	1110

CF=1(Step 3) the result is negative

AAS: ASCII adjust for subtraction

❖ Works only on AL

❖ Function similar to AAA

❖ Example: MOV BH, 00

; Clear BH

MOV AX, '15'

; 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

MOV AX,0105H ;

AX=0105 unpacked BCD for 15

MOV CL,06

; CL=06H

SUB AL,CL

; 5-6=-1 (FFH)

AAS

;FFH in AL is adjusted to 09 and

;AH is decremented leaving AX=0009

ASCII to Unpacked BCD Conversion

ASCII numbers:

Key	ASCII(Hex)	Binary	BCD (Unpacked)
0	30	011 0000	0000 0000
1	31	011 0001	0000 0001
2	32	011 0010	0000 0010
3	33	011 0011	0000 0011
4	34	011 0100	0000 0100
5	35	011 0101	0000 0101
6	36	011 0110	0000 0110
7	37	011 0111	0000 0111
8	38	011 1000	0000 1000
9	39	011 1001	0000 1001

```

Ex:  ASC      DB  '9562481273'
      ORG 0010H
      UNPACK DB  10 DUP(?)
      ...
      MOV CX,5                ;CX is the loop counter
      MOV BX,OFFSET ASC       ;BX points to ASCII data
      MOV DI,OFFSET UNPACK    ;DI points to unpacked BCD data
      AGAIN: MOV AX,WORD PTR [BX] ;move next 2 ASCII numbers to AX
            AND AX,0F0F        ;remove ASCII 3s (011)
            MOV WORD PTR [DI],AX ;store unpacked BCD
            ADD DI,2           ;point to next unpacked BCD data
            ADD BX,2           ;point to next ASCII data
            LOOP AGAIN

```

ASCII to packed BCD Conversion

<u>Key</u>	<u>ASCII</u>	<u>Unpacked BCD</u>	<u>Packed BCD</u>
4	34	00000100	
7	37	00000111	01000111 or 47H

```

      ORG 0010H
      VAL_ASC DB  '47'
      VAL_BCD DB  ?
      ...
;reminder: the DB will put 34 in 0010H location and 37 in 0011H.
      MOV     AX,WORD PTR VAL_ASC      ;AH=37     AL=34
      AND     AX,0F0FH                ;mask 3 to get unpacked BCD
      XCHG    AH,AL                   ;swap AH and AL
      MOV     CL,4                     ;CL=04 to shift 4 times
      SHL     AH,CL                    ;shift left AH to get AH=40H
      OR      AL,AH                    ;OR them to get packed BCD
      MOV     VAL_BCD,AL               save the result

```

Multiplication and Division

Multiplication (MUL or IMUL)	Multiplicand	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

- AX = 1h and BX = FFFFh, then:

Instruction	Result	DX	AX	CF/OF
mul bx	65535	0000	FFFF	0
imul bx	-1	FFFF	FFFF	0

- AX = FFFFh and BX = FFFFh, then:

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

- AL = 30h and BL = 4h, then:

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

- AL = 80h and BL = FFh, then

Instruction	Result	AH	AL	CF/OF
mul bl	32640	7F	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

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

Executing the CWD instruction, we get

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

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

Example: ADD

```
; This program adds 5 unsigned byte numbers.
.MODEL SMALL
.STACK 64
.DATA
COUNT    EQU    05
DATA       DB     125,235,197,91,48
           ORG     0008H
SUM        DW     ?
.CODE
MAIN:      MOV AX, @DATA
           MOV DS, AX
           MOV CX, COUNT           ;CX is the loop counter
           MOV SI, OFFSET DATA    ;SI is the data pointer
           MOV AX, 00              ;AX will hold the sum
BACK:      ADD AL, [SI]             ;add the next byte to AL
           ADC AH, 00              ;add 1 to AH if CF =1
           INC SI                  ;increment data pointer
           DEC CX                  ;decrement loop counter
           JNZ BACK               ;if not finished, go add next byte
           MOV SUM, AX            ;store sum
           MOV AH, 4CH
           INT  21H               ;go back to DOS
           END    MAIN
```

Overflow problem in signed number operations

When using signed numbers Overflow problem can arise after an operation. This problem arises if the result in a register after an operation is too large. In such a case CPU sets the OF (Overflow Flag). The programmer must consider the overflow case.

```
Ex:  DATA1    DB    +96
     DATA2    DB    +70
           ...
           MOV  AL, DATA1      ;AL=0110 0000 (60H)
           MOV  BL, DATA2      ;BL=0100 0110 (46H)
           ADD  AL, BL          ;AL=1010 0110 (AL=A6H=-90 invalid!)
```

```
+ 96  0110  0000
+ 70  0100  0110
-----
+166 1010  0110
```

According to the CPU this is -90, which is wrong. (OF=1, SF=1, CF=0)

When the OF is set in 8-bit operations

In 8-bit signed number operations, OF is set to 1 is either of the following two conditions occurs:

1. There is a carry out from D6 to D7, but no carry out from D7 (CF=0).
2. There is a carry out from D7 (CF=1), but no carry out from D6 to D7.

```
Ex:      MOV DL, -128           ;DL=1000 0000 (80H)
           MOV CH, -2           ;CH=1111 1110 (FEH)
           ADD DL, CH           ;DL=0111 1110 (DL=FEH=+126 invalid!)
```

```
-128 1000  0000
+  -2  1111  1110
-----
-130  0111  1110
```

OF=1, SF=0, CF=1

CBW ; Convert signed Byte to signed Word CBW will copy D7 (the sign flag) of **AL** to all bits of **AH**. Notice that the operand is assumed to be AL and the contents of AH is destroyed.

MOV AL,+96 ;AL = 0110 0000

CBW ;now AH= 0000 0000 and AL=0110 0000

MOV AL,-2 ;AL = 1111 1110

CBW ;now AH= 1111 1111 and AL=1111 1110

```

Ex:  DATA1    DB    +96
      DATA2    DB    +70
      RESULT    DW    ?
      ...
      MOV  AH,0          ;AH=0
      MOV  AL,DATA1      ;get operand 1
      MOV  BL,DATA2      ;get operand 2
      ADD  AL,BL         ;add them
      JNO  OVER          ;jump if there is no overflow (OF=0) to OVER
      MOV  AL,DATA2      ;otherwise get operand 2 to
      CBW               ;sign extend it
      MOV  BX,AX
      MOV  AL,DATA1      ; get back operand 1 to
      CBW               ;sign extend it
      ADD  AX,BX         ;add them
OVER:  MOV  RESULT,AX    ;save the result

```

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

```