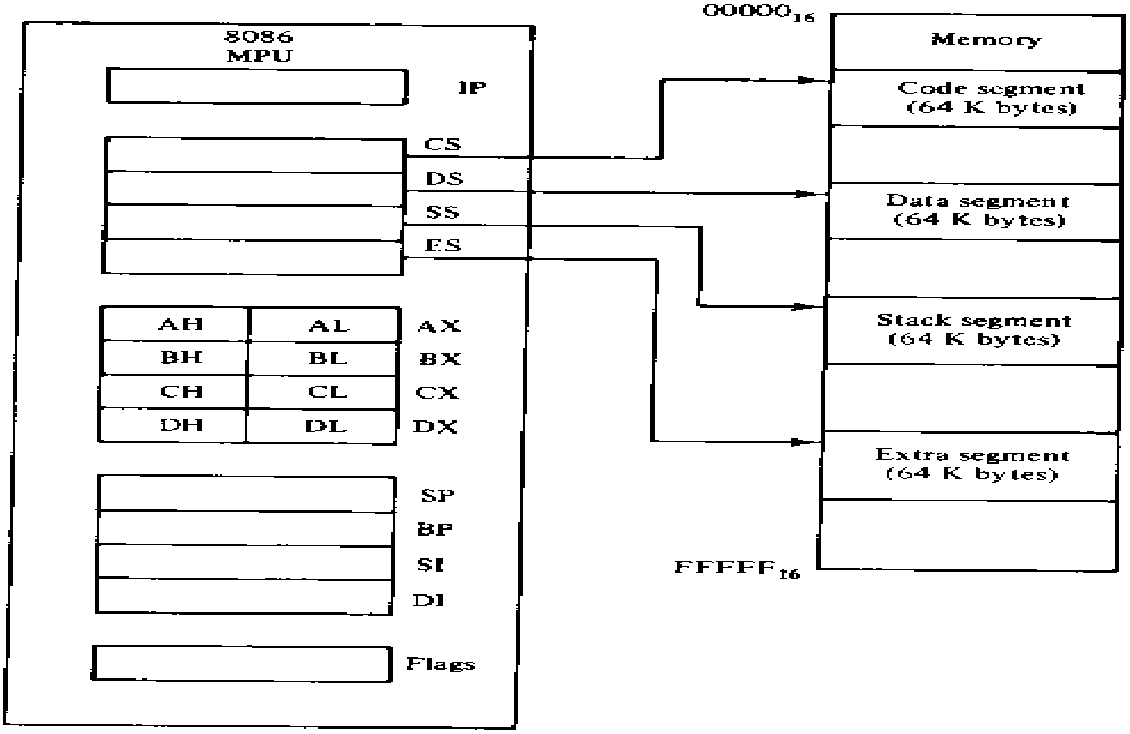
Topic 5: 8086 Assembly Language Programming(24 Marks)

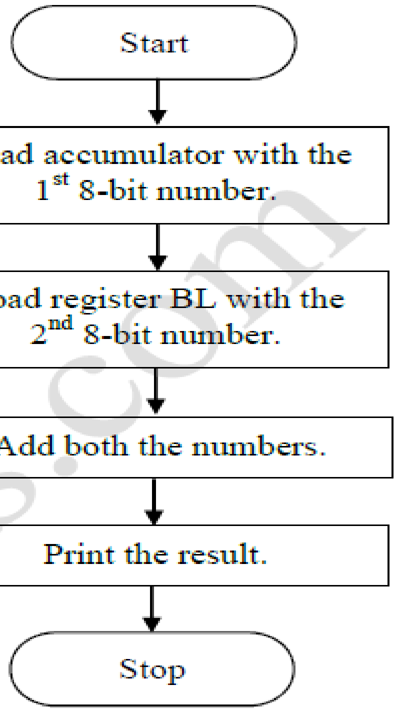
Figure 3.1 **Software model of the 8086 microprocessor**



The 8086 and 80286 Microprocessors:
Hardware, Software, and Interfacing
by Triebel/Singh

Add two 8bit numbers

Flowchart:


Start
Load accumulator with the 1st 8-bit number.
Load register BL with the 2nd 8-bit number.
Add both the numbers.
Print the result.
Stop

Program:

Instructions	Comments
include "emu8086.inc"	
ORG 100h	
MOV AL, 05H	Move 1st 8-bit number to AL.
MOV BL, 03H	Move 2nd 8-bit number to BL.
ADD AL, BL	Add BL with AL.
CALL PRINT_NUM	Print the result.
RET	Return.
DEFINE_PRINT_NUM	Declare function.
END	

Explanation:

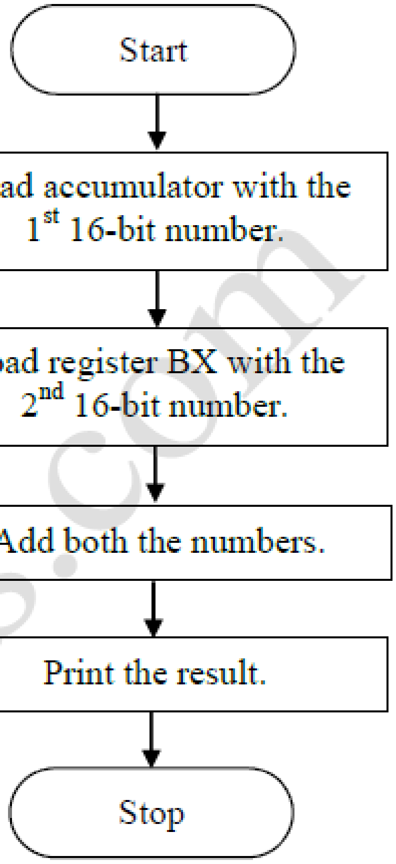
- This program adds two 8-bit numbers.
- The program has been developed using ***emu8086*** emulator available at: www.emu8086.com.
- ORG 100h is a compiler directive. It tells compiler how to handle the source code.
- It tells compiler that the executable file will be loaded at the offset of 100h (256 bytes).
- The 1st 8-bit number 05H is moved to accumulator AL.
- The 2nd 8-bit number 03H is moved to register BL.
- Then, both the numbers are added and the result is stored in AL.
- The result is printed on the screen.

Output:

Before Execution:	**After Execution:**
AL = 05H	AL = 08H
BL = 03H	

Add two 16 bit numbers

Flowchart:


Start
Load accumulator with the 1st 16-bit number.
Load register BX with the 2nd 16-bit number.
Add both the numbers.
Print the result.
Stop

Program:

Instructions	Comments
include "emu8086.inc"	
ORG 100h	
MOV AX, 0005H	Move 1st 16-bit number to AX.
MOV BX, 0003H	Move 2nd 16-bit number to BX.
ADD AX, BX	Add BX with AX.
CALL PRINT_NUM	Print the result.
RET	Return.
DEFINE_PRINT_NUM	Declare function.
END	

Explanation:

- This program adds two 16-bit numbers.
- The program has been developed using ***emu8086*** emulator available at: www.emu8086.com.
- ORG 100h is a compiler directive. It tells compiler how to handle the source code.
- It tells compiler that the executable file will be loaded at the offset of 100h (256 bytes).
- The 1st 16-bit number 0005H is moved to accumulator AX.
- The 2nd 16-bit number 0003H is moved to register BX.
- Then, both the numbers are added and the result is stored in AX.
- The result is printed on the screen.

Output:

Before Execution:

AX = 0005H

BX = 0003H

After Execution:

AX = 0008H

PROGRAMS FOR 16 BIT ARITHMETIC OPERATIONS (USING 8086)

ADDITION OF TWO 16-BIT NUMBERS

Address	Mnemonics	Op-Code	Commands
1000	MOV AX,[1100]	A1,00,11	Move the data to accumulator
1003	ADD AX,[1102]]	03,06,02,11	Add memory content with accumulator
1007	MOV [1200],AX	A3,00,12	Move accumulator content to memory
100A	HLT	F4	Stop

Input Address	Data	Output Address	Data
1100	**02**	**1200**	**04**
1101	**02**	**1201**	**04**
1102	**02**		
1103	**02**		

Second Way

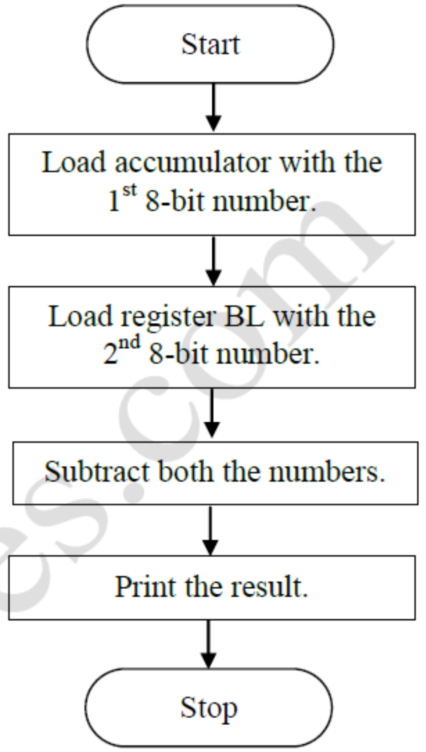
ALGORITHM:
I. Initialize the SI register to input data memory location
II. Initialize the DI register to output data memory location
III. Initialize the CL register to zero for carry
IV. Get the 1st data into accumulator.
V. Add the accumulator with 2nd data
VI. Check the carry flag, if not skip next line
VII. Increment carry(CL Reg)
VIII. Move the result from accumulator to memory.
IX. Also store carry register
X. Halt

Program

```
MOV SI, 2000H
MOV DI, 3000H
MOV CL, 00H
MOV AX, [SI]
ADD AX, [SI+2]
JNC STORE
INC CL
MOV [DI], AX
MOV [DI+2], CL
INT 3
```

Subtract two 8 bit numbers

Flowchart:


Start
Load accumulator with the 1st 8-bit number.
Load register BL with the 2nd 8-bit number.
Subtract both the numbers.
Print the result.
Stop

Program:

Instructions	Comments
include "emu8086.inc"	
ORG 100h	
MOV AL, 05H	Move 1st 8-bit number to AL.
MOV BL, 03H	Move 2nd 8-bit number to BL.
SUB AL, BL	Subtract BL from AL.
CALL PRINT_NUM	Print the result.
RET	Return.
DEFINE_PRINT_NUM	Declare function.
END	

Explanation:

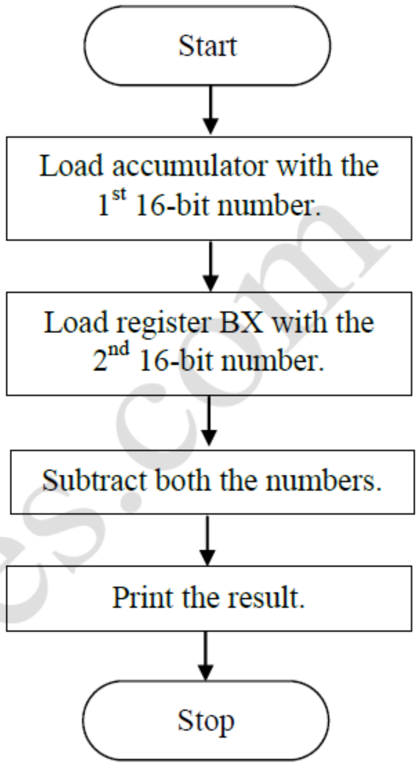
- This program subtracts two 8-bit numbers.
- The program has been developed using ***emu8086*** emulator available at: www.emu8086.com.
- ORG 100h is a compiler directive. It tells compiler how to handle the source code.
- It tells compiler that the executable file will be loaded at the offset of 100h (256 bytes).
- The 1st 8-bit number 05H is moved to accumulator AL.
- The 2nd 8-bit number 03H is moved to register BL.
- Then, both the numbers are subtracted and the result is stored in AL.
- The result is printed on the screen.

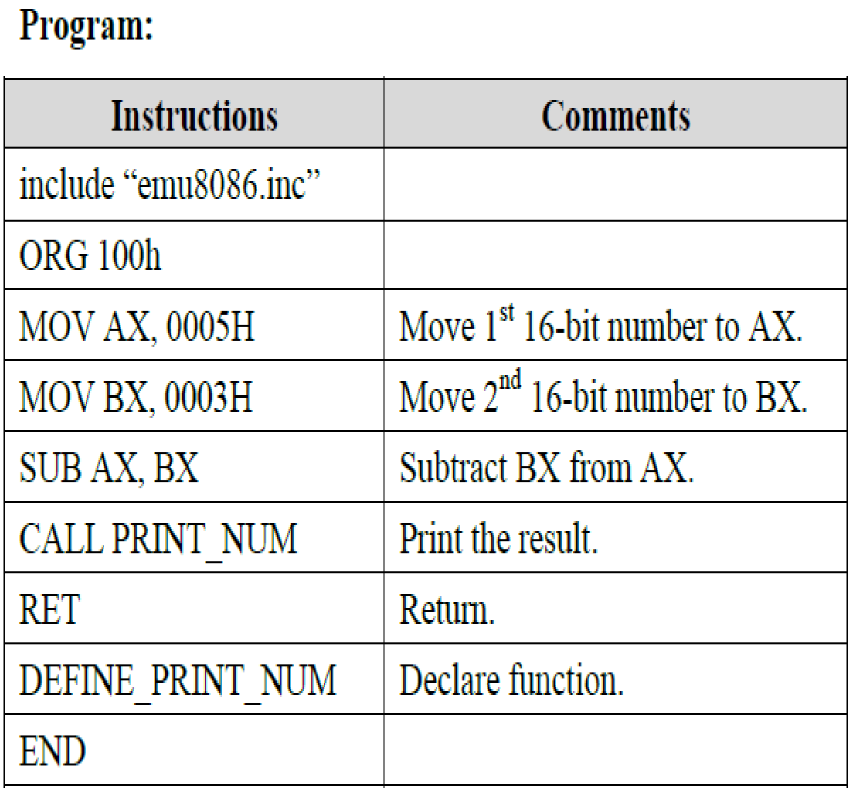
Output:

Before Execution:	After Execution:
AL = 05H	AL = 02H
BL = 03H	

Subtract two 16bit numbers

Flowchart:


Start
Load accumulator with the 1st 16-bit number.
Load register BX with the 2nd 16-bit number.
Subtract both the numbers.
Print the result.
Stop



Program:

Instructions	Comments
include "emu8086.inc"	
ORG 100h	
MOV AX, 0005H	Move 1st 16-bit number to AX.
MOV BX, 0003H	Move 2nd 16-bit number to BX.
SUB AX, BX	Subtract BX from AX.
CALL PRINT_NUM	Print the result.
RET	Return.
DEFINE_PRINT_NUM	Declare function.
END	

Explanation:

- This program subtracts two 16-bit numbers.
- The program has been developed using ***emu8086*** emulator available at: www.emu8086.com.
- ORG 100h is a compiler directive. It tells compiler how to handle the source code.
- It tells compiler that the executable file will be loaded at the offset of 100h (256 bytes).
- The 1st 16-bit number 0005H is moved to accumulator AX.
- The 2nd 16-bit number 0003H is moved to register BX.
- Then, both the numbers are subtracted and the result is stored in AX.
- The result is printed on the screen.

Output:

Before Execution:

AX = 0005H

BX = 0003H

After Execution:

AX = 0002H

SUBTRACTION OF TWO 16-BIT NUMBERS

Address	Mnemonics	Op-Code	Commands
1000	MOV AX,[1100]	A1,00,11	Move the data to accumulator
1003	SUB AX,[1102]]	2B,06,02,11	Subtract memory content with accumulator
1007	MOV [1200],AX	A3,00,12	Move accumulator content to memory
100A	HLT	F4	Stop

Input Address	**Data**	**Output Address**	**Data**
1100	**04**	**1200**	**04**
1101	**02**	**1201**	**04**
1102	**04**		
1103	**02**		

Second Way

ALGORITHM:

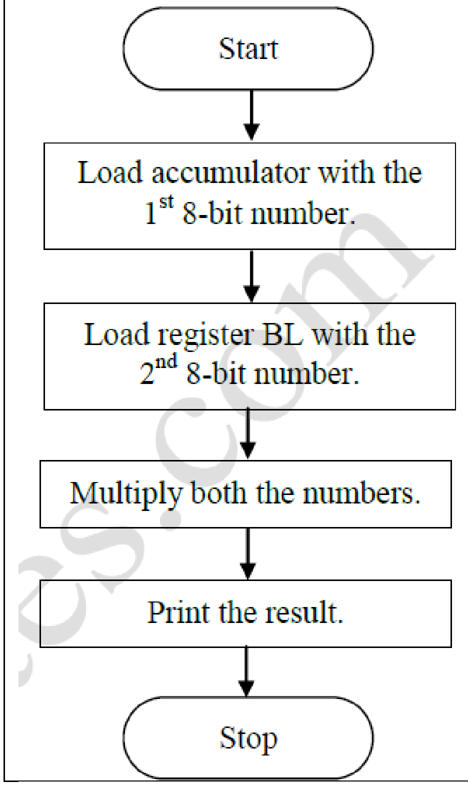
I. Initialize the SI register to input data memory location
II. Initialize the DI register to output data memory location
III. Initialize the CL register to zero for borrow
IV. Get the 1st data into accumulator.
V. Subtract the accumulator with 2nd data
VI. Check the carry flag, if not set skip next line
VII. Increment carry(CL Reg)
VIII. 2’s Compliment Accumalator
IX.Move the result from accumulator to memory.
X. Also store carry register
XI. Halt

Program

```
MOV SI, 2000H
MOV DI, 3000H
MOV CL, 00H
MOV AX, [SI]
SUB AX, [SI+2]
JNC STORE
INC CL
NEG AX
MOV [DI], AX
MOV
```

Multiply two 8 bit unsigned numbers

Flowchart:


Start
Load accumulator with the 1st 8-bit number.
Load register BL with the 2nd 8-bit number.
Multiply both the numbers.
Print the result.
Stop

Program:

Instructions	Comments
include "emu8086.inc"	
ORG 100h	
MOV AL, 04H	Move 1st 8-bit number to AL.
MOV BL, 02H	Move 2nd 8-bit number to BL.
MUL BL	Multiply BL with AL and the result will be in AX.
CALL PRINT_NUM	Print the result.
RET	Return.
DEFINE_PRINT_NUM	Declare function.
END	

Explanation:

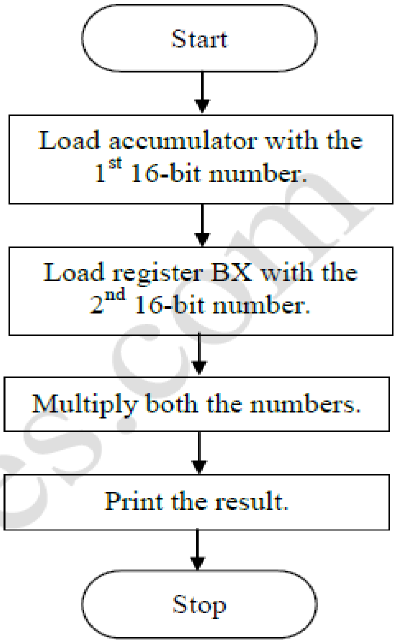
- This program multiplies two 8-bit unsigned numbers.
- The program has been developed using ***emu8086*** emulator available at: www.emu8086.com.
- ORG 100h is a compiler directive. It tells compiler how to handle the source code.
- It tells compiler that the executable file will be loaded at the offset of 100h (256 bytes).
- The 1st 8-bit number 04H is moved to accumulator AL.
- The 2nd 8-bit number 02H is moved to register BL.
- Then, both the numbers are multiplied.
- The multiplication of two 8-bit numbers may result into 16-bit number. So, the result is stored in AX register.
- The MSB is stored in AH and LSB is stored in AL.
- The result is printed on the screen.

Output:

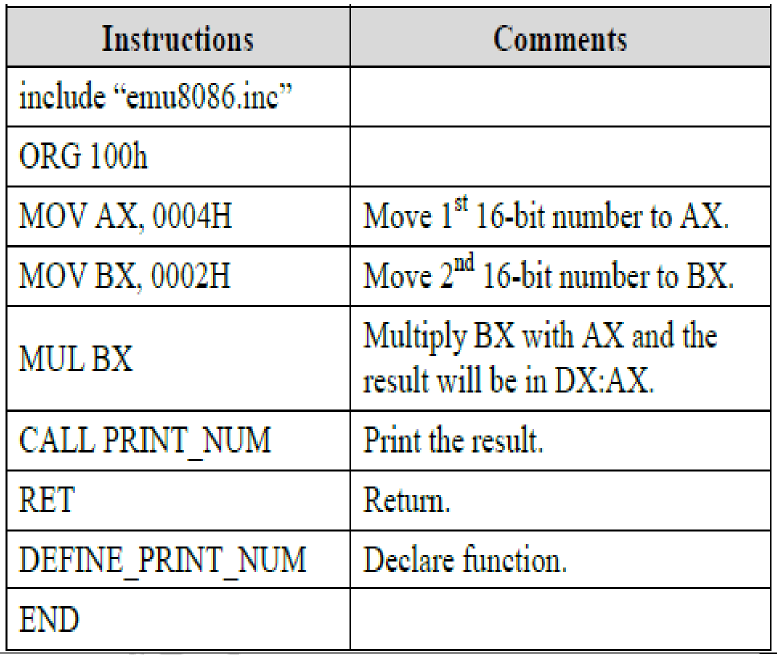
Before Execution:	**After Execution:**
AL = 04H	AX = 0008H
BL = 02H	

Multiply two 16 bit unsigned numbers

Flowchart:


Start
Load accumulator with the 1st 16-bit number.
Load register BX with the 2nd 16-bit number.
Multiply both the numbers.
Print the result.
Stop

Program:



Instructions	Comments
include "emu8086.inc"	
ORG 100h	
MOV AX, 0004H	Move 1st 16-bit number to AX.
MOV BX, 0002H	Move 2nd 16-bit number to BX.
MUL BX	Multiply BX with AX and the result will be in DX:AX.
CALL PRINT_NUM	Print the result.
RET	Return.
DEFINE_PRINT_NUM	Declare function.
END	

Explanation:

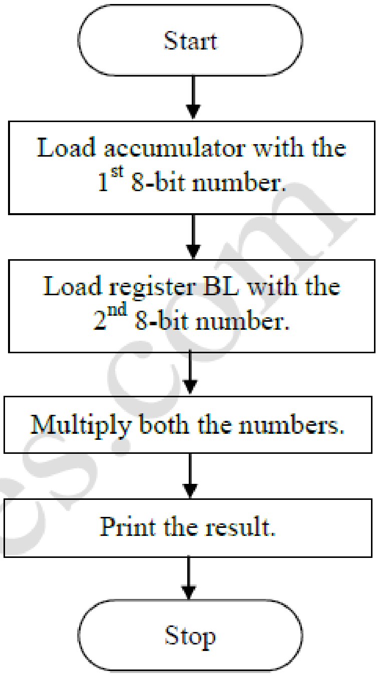
- This program multiplies two 16-bit unsigned numbers.
- The program has been developed using ***emu8086*** emulator available at: www.emu8086.com.
- ORG 100h is a compiler directive. It tells compiler how to handle the source code.
- It tells compiler that the executable file will be loaded at the offset of 100h (256 bytes).
- The 1st 16-bit number 0004H is moved to accumulator AX.
- The 2nd 16-bit number 0002H is moved to register BX.
- Then, both the numbers are multiplied.
- The multiplication of two 16-bit numbers may result into 32-bit number. So, the result is stored in the DX and AX register.
- The MSB is stored in DX and LSB is stored in AX.
- The result is printed on the screen.

Output:

Before Execution:	**After Execution:**
AX = 0004H	AX = 0008H
BX = 0002H	DX = 0000H

Multiply two 8 bit signed numbers

Flowchart:


Start
Load accumulator with the 1st 8-bit number.
Load register BL with the 2nd 8-bit number.
Multiply both the numbers.
Print the result.
Stop

Program:

Instructions	Comments
include "emu8086.inc"	
ORG 100h	
MOV AL, 04H	Move 1st 8-bit number to AL.
MOV BL, FEH	Move 2nd 8-bit number to BL.
IMUL BL	Signed multiply BL with AL and the result will be in AX.
CALL PRINT_NUM	Print the result.
RET	Return.
DEFINE_PRINT_NUM	Declare function.
END	

Explanation:

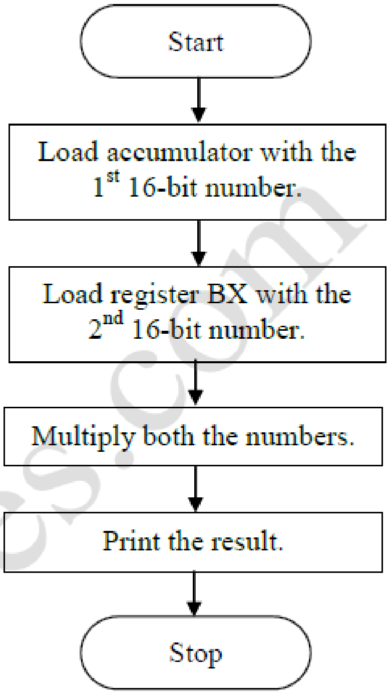
- This program multiplies two 8-bit signed numbers.
- The program has been developed using ***emu8086*** emulator available at: www.emu8086.com.
- ORG 100h is a compiler directive. It tells compiler how to handle the source code.
- It tells compiler that the executable file will be loaded at the offset of 100h (256 bytes).
- The 1st 8-bit number 04H is a positive number and is moved to accumulator AL.
- The 2nd 8-bit number FEH is a negative number (-2 in decimal) and is moved to register BL.
- Then, both the numbers are multiplied.
- The multiplication of two 8-bit numbers may result into 16-bit number. So, the result is stored in AX register.
- The MSB is stored in AH and LSB is stored in AL.
- The result is printed on the screen.

Output:

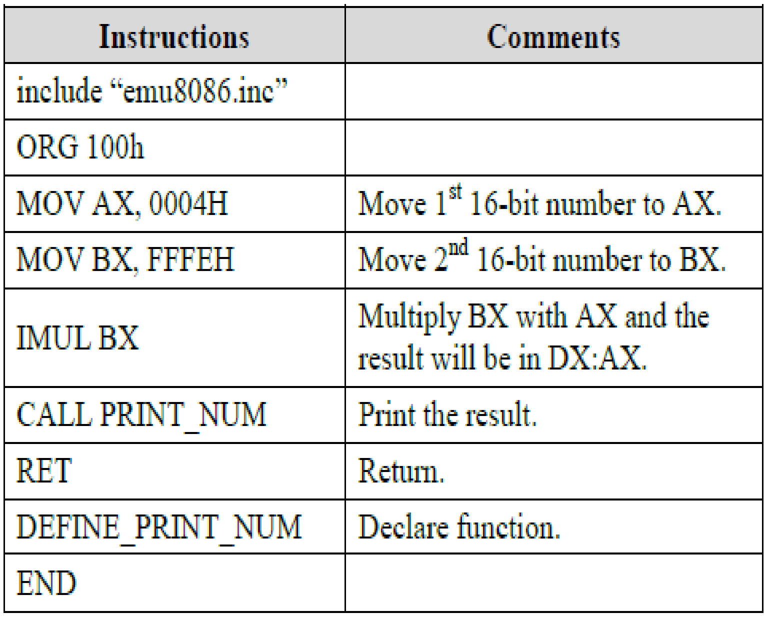
Before Execution:	**After Execution:**
AL = 04H	AX = FFF8H (-8 in decimal)
BL = FEH (-2 in decimal)	

Multiply two 16 bit signed numbers

Flowchart:


Start
Load accumulator with the 1st 16-bit number.
Load register BX with the 2nd 16-bit number.
Multiply both the numbers.
Print the result.
Stop

Program:



Instructions	Comments
include "emu8086.inc"	
ORG 100h	
MOV AX, 0004H	Move 1st 16-bit number to AX.
MOV BX, FFFEH	Move 2nd 16-bit number to BX.
IMUL BX	Multiply BX with AX and the result will be in DX:AX.
CALL PRINT_NUM	Print the result.
RET	Return.
DEFINE_PRINT_NUM	Declare function.
END	

Explanation:

- This program multiplies two 16-bit signed numbers.
- The program has been developed using ***emu8086*** emulator available at: www.emu8086.com.
- ORG 100h is a compiler directive. It tells compiler how to handle the source code.
- It tells compiler that the executable file will be loaded at the offset of 100h (256 bytes).
- The 1st 16-bit number 0004H is a positive number and is moved to accumulator AX.
- The 2nd 16-bit number FFFEH is a negative number (-2 in decimal) and is moved to register BX.
- Then, both the numbers are multiplied.
- The multiplication of two 16-bit numbers may result into 32-bit number. So, the result is stored in the DX and AX register.
- The MSB is stored in DX and LSB is stored in AX.
- The result is printed on the screen.

Output:

Before Execution:	**After Execution:**
AX = 0004H	AX = FFF8H (-8 in decimal)
BX = FFFEH (-2 in decimal)	DX = FFFFH

MULTIPLICATION OF TWO 16-BIT NUMBERS

Address	**Mnemonics**	**Op-Code**	**Commands**
1000	MOV AX,[1100]	A1,00,11	Move the data to accumulator
1003	MUL AX,[1102]]	F7,26,02,11	Multiply memory content with accumulator
1007	MOV [1200],DX	87,16,00,12	Move accumulator content to AX register
100B	MOV [1202],AX	A3,02,12	Move accumulator content to DX register
100E	HLT	F4	Stop

Input Address	**Data**	**Output Address**	**Data**
1100	**02**	**1200**	**00**
1101	**02**	**1201**	**04**
1102	**02**	**1202**	**08**
1103	**02**	**1203**	**04**

Second Way

ALGORITHM:

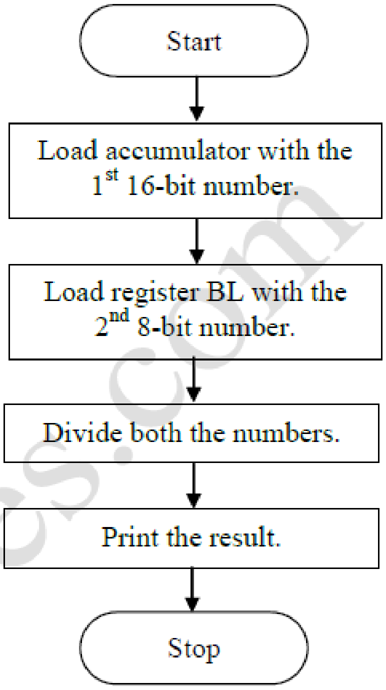
I. GET MULTIPLIER INTO ACCUMULATOR FROM MEMORY
II. GET MULTIPLICAND INTO BX REGISTER
III. MULTIPLY AX AND BX
IV. STORE LOWER ORDER WORD FORM ACCUMULATOR INTO MEMORY
V. STORE HIGHER ORDER WORD FROM DX INTO MEMORY
VI. HALT

Program

```
MOV AX, [2000H]
MOV BX, [2002H]
MUL BX
MOV [3000], AX
MOV [3002], DX
INT 3
```

Divide 16 bit unsigned bit using 8 bit unsigned number

Flowchart:


Start
Load accumulator with the 1st 16-bit number.
Load register BL with the 2nd 8-bit number.
Divide both the numbers.
Print the result.
Stop

Program:

Instructions	Comments
include "emu8086.inc"	
ORG 100h	
MOV AX, 0008H	Move 1st 16-bit number to AX.
MOV BL, 02H	Move 2nd 8-bit number to BL.
DIV BL	Divide AX with BL and the result will be in AX.
CALL PRINT_NUM	Print the result.
RET	Return.
DEFINE_PRINT_NUM	Declare function.
END	

Explanation:

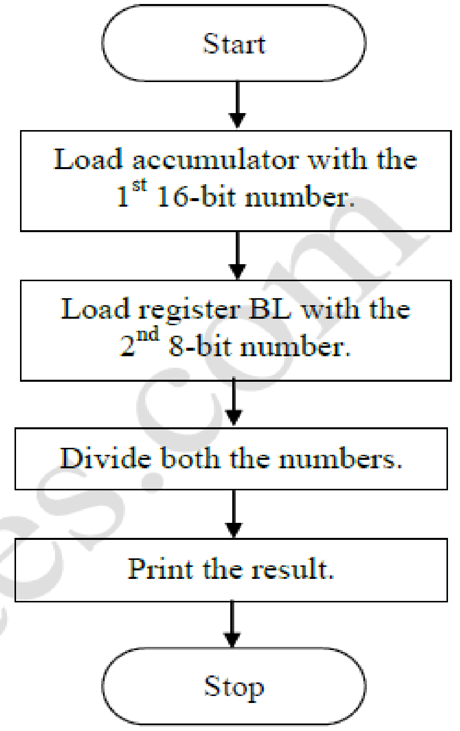
- This program divides a 16-bit unsigned number by an 8-bit unsigned number.
- The program has been developed using ***emu8086*** emulator available at: www.emu8086.com.
- ORG 100h is a compiler directive. It tells compiler how to handle the source code.
- It tells compiler that the executable file will be loaded at the offset of 100h (256 bytes).
- The 1st 16-bit number 0008H, i.e. dividend, is moved to accumulator AX.
- The 2nd 8-bit number 02H, i.e. divisor, is moved to register BL.
- Then, both the numbers are divided.
- The result of division is stored in AX. AL contains the quotient and AH contains the remainder.
- The result is printed on the screen.

Output:

Before Execution:	After Execution:
AX = 0008H	AL = 04H (Quotient)
BL = 02H	AH = 00H (Remainder)

Divide 16 bit signed bit using 8 bit signed number

Flowchart:


Start
Load accumulator with the 1st 16-bit number.
Load register BL with the 2nd 8-bit number.
Divide both the numbers.
Print the result.
Stop

Program:

Instructions	Comments
include "emu8086.inc"	
ORG 100h	
MOV AX, 0008H	Move 1st 16-bit number to AX.
MOV BL, FEH	Move 2nd 8-bit number to BL.
IDIV BL	Divide AX with BL and the result will be in AX.
CALL PRINT_NUM	Print the result.
RET	Return.
DEFINE_PRINT_NUM	Declare function.
END	

Explanation:

- This program divides a 16-bit signed number by an 8-bit signed number.
- The program has been developed using ***emu8086*** emulator available at: www.emu8086.com.
- ORG 100h is a compiler directive. It tells compiler how to handle the source code.
- It tells compiler that the executable file will be loaded at the offset of 100h (256 bytes).
- The 1st 16-bit number 0008H, i.e. dividend, is moved to accumulator AX.
- The 2nd 8-bit number FEH (-2 in decimal), i.e. divisor, is moved to register BL.
- Then, both the numbers are divided.
- The result of division is stored in AX. AL contains the quotient and AH contains the remainder.
- The result is printed on the screen.

Output:

Before Execution:	**After Execution:**	
AX = 0008H	AL = FCH (-4 in decimal)	(Quotient)
BL = FEH (-2 in decimal)	AH = 00H	(Remainder)

DIVISION OF TWO 16-BIT NUMBERS

Address	Mnemonics	Op-Code	Commands
1000	MOV AX,[1100]	A1,00,11	Move the data to accumulator
1003	DIV AX,[1102]]	F7,26,02,11	Divide memory content with accumulator
1007	MOV [1200],DX	87,16,00,12	Move accumulator content to AX register
100B	MOV [1202],AX	A3,02,12	Move accumulator content to DX register
100E	HLT	F4	Stop

Input Address	**Data**	**Output Address**	**Data**
1100	**04**	**1200**	**02**
1101	**04**	**1201**	**02**
1102	**02**	**1202**	**00**
1103	**02**	**1203**	**00**

Second Way

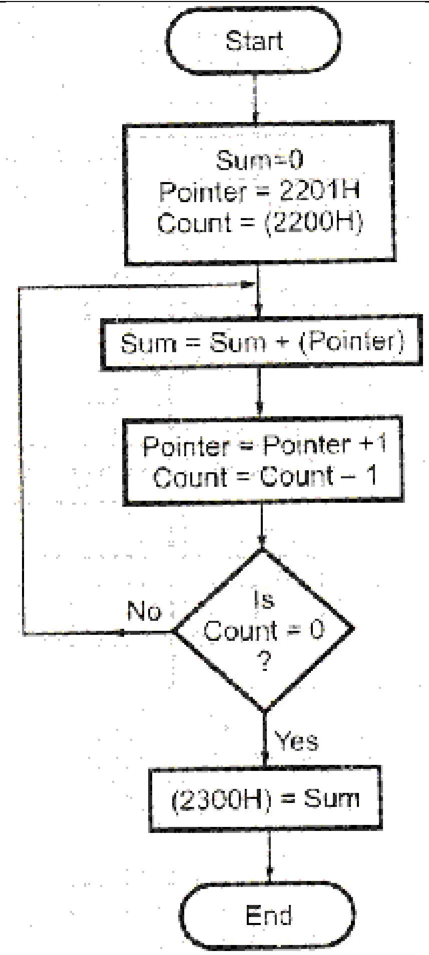
ALGORITHM:
I. GET DIVIDEND INTO ACCUMULATOR FROM MEMORY
II. GET DIVISOR INTO BX REGISTER
III. DIVIDE AX BY BX
IV. STORE QUOTIENT FORM ACCUMULATOR INTO MEMORY
V. STORE REMAINDER FROM DX INTO MEMORY
VI. HALT

Program

```
MOV AX, [2000H]
MOV BX, [2002H]
DIV BX
MOV [3000], AX
MOV [3002], DX
INT 3
```

8086 Microprocessor SUM OF N-NUMBERS Program

SUM OF N-NUMBERS:

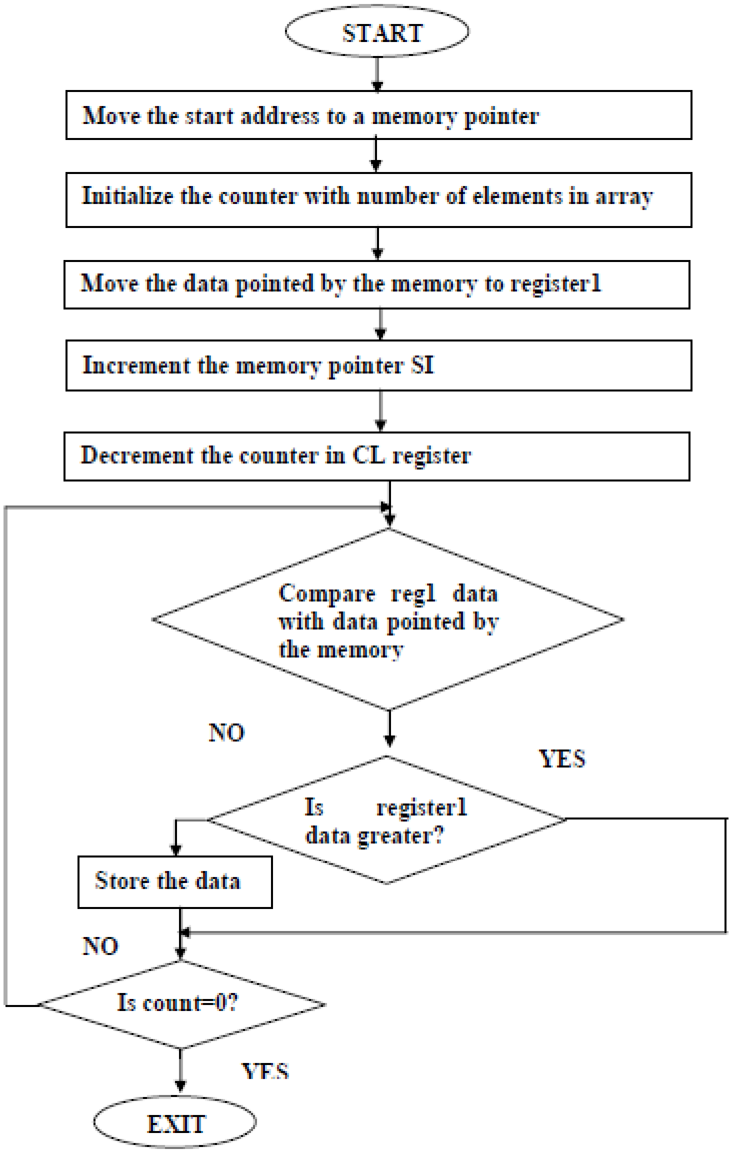


```
assume cs:code,ds:data
data segment
org 2000h
series dw 1234h,2345h,0abcdh,103fh,5555h
sum dw 00h
carry dw 00h
data ends
code segment
start:mov ax,data
mov ds,ax
mov ax,00h
mov bx,00h
mov cx,05h
mov si,2000h
go:add ax,[si]
adc bx,00h
inc si
inc si
dec cx
jnz go
mov sum,ax
mov carry,bx
mov ah,4ch
int 21h
code ends
end start
```

Code for Program to find the largest and smallest number from an array of n 16 bit nos in Assembly Language

```
DATA SEGMENT
A DW 8,2,5,6,1,3
DATA ENDS
CODE SEGMENT
      ASSUME DS:DATA,CS:CODE
START:
      MOV AX,DATA
      MOV DS,AX
      MOV CX,0000
      MOV CL,06
      LEA BX,A
      MOV DX,WORD PTR[BX]
      MOV AX,0000
   L1:CMP AX,WORD PTR[BX]
      JNC L2
      MOV AX,WORD PTR[BX]
   L2:CMP DX,WORD PTR[BX]
      JC L3
      MOV DX,WORD PTR[BX]
   L3:ADD BX,02
      DEC CL
      CMP CL,00
      JNZ L1
      MOV AH,4CH
      INT 21H
CODE ENDS
END START
```

FIND THE LARGEST NUMBER IN AN ARRAY


START
Move the start address to a memory pointer
Initialize the counter with number of elements in array
Move the data pointed by the memory to register1
Increment the memory pointer SI
Decrement the counter in CL register
Compare reg1 data with data pointed by the memory
NO
YES
Is register1 data greater?
Store the data
NO
Is count=0?
YES
EXIT

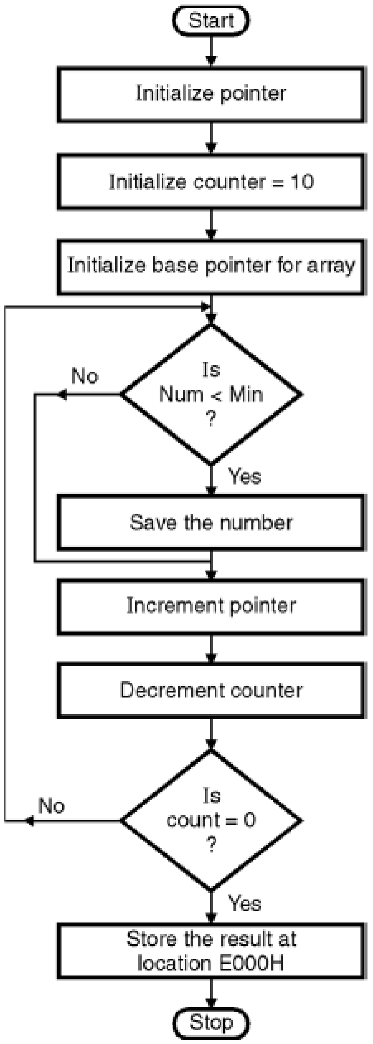
ALGORITHM:

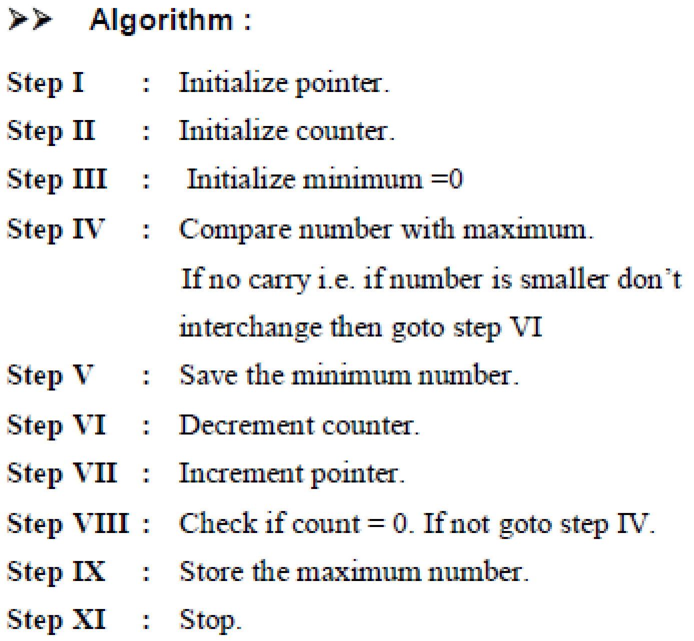
1. Take the first number of the array.
2. Compare with next number.
3. Take the bigger one of the them.
4. Decrement the count in CL register.
5. If the count is not zero then continue from step 2.
6. Store the result into Memory address 9500.

MNEMONICS	COMMENTS
MOV SI,9000	Load 9000 address into SI
MOV CL,[SI]	Load SI value into CL

INC SI	Increment SI
MOV AL,[SI]	Move the first data in AL
DEC CL	Reduce the count
INC SI	Increment SI
CMP AL,[SI]	if AL> [SI] then go to jump1 (no swap)
JNB 1111	If count is zero then jump into 1111
MOV AL,[SI]	Else store large no in to AL
DEC CL	Decrement the count
JNZ 110A	If count is not zero then jump into 110A
MOV DI,9500	Else store the biggest number at 9500
MOV [DI],AL	Store the AL value into DI
INT3	Break point

Smallest and Largest numbers from array





➤➤ **Algorithm :**

Step I : Initialize pointer.

Step II : Initialize counter.

Step III : Initialize minimum =0

Step IV : Compare number with maximum.
If no carry i.e. if number is smaller don't interchange then goto step VI

Step V : Save the minimum number.

Step VI : Decrement counter.

Step VII : Increment pointer.

Step VIII : Check if count = 0. If not goto step IV.

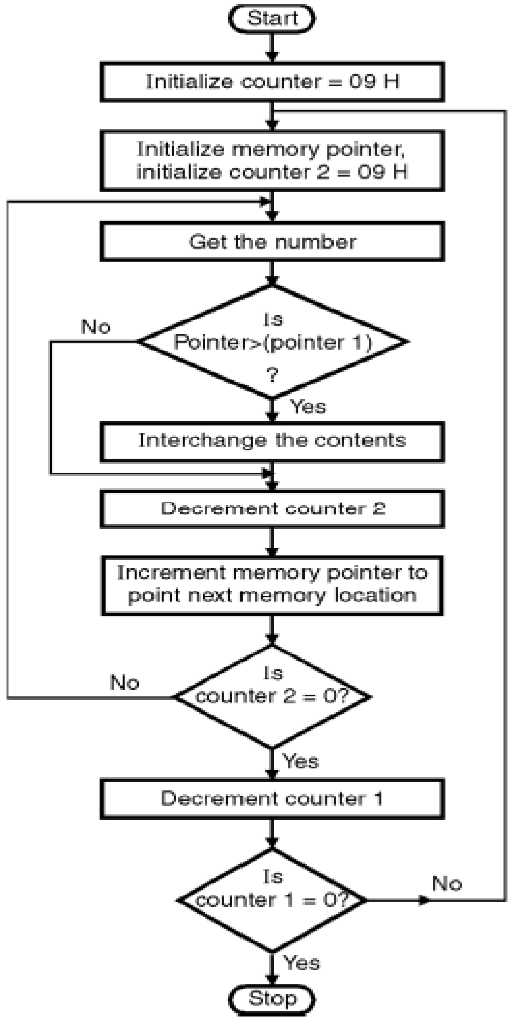
Step IX : Store the maximum number.

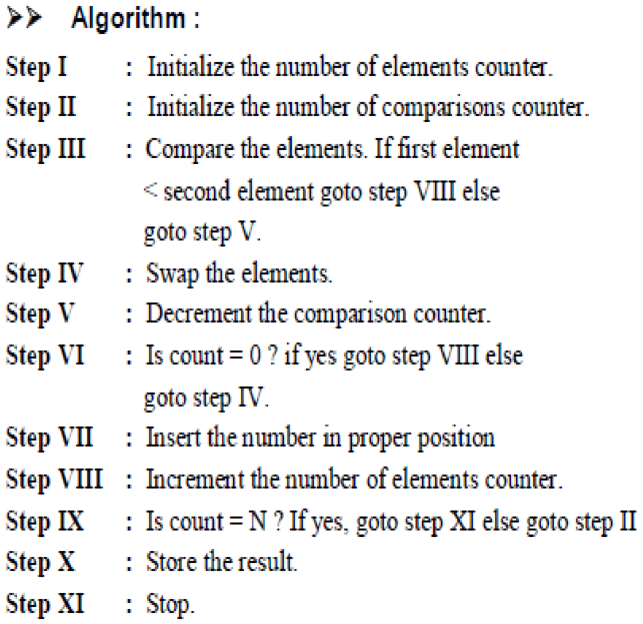
Step XI : Stop.

Program :

```
        Instruction            Comment
         LDA    D000H
         MOV    C, A           ; Initialize counter
         LXI    H, D001H       ; Initialize pointer
           MOV         A, M
           INX  M
BACK:    CMP    M              ; Is number < miniumum
         JC     SKIP
         MOV    A, M           ; If number < minimum
                               ; then interchange.
SKIP:    INX    H
         DCR    C
         JNZ    BACK
         STA    E000H ; Store minimum number
         HLT                   ; Terminate program execution
```

Ascending/Descending order





➤➤ **Algorithm :**

Step I : Initialize the number of elements counter.

Step II : Initialize the number of comparisons counter.

Step III : Compare the elements. If first element < second element goto step VIII else goto step V.

Step IV : Swap the elements.

Step V : Decrement the comparison counter.

Step VI : Is count = 0 ? if yes goto step VIII else goto step IV.

Step VII : Insert the number in proper position

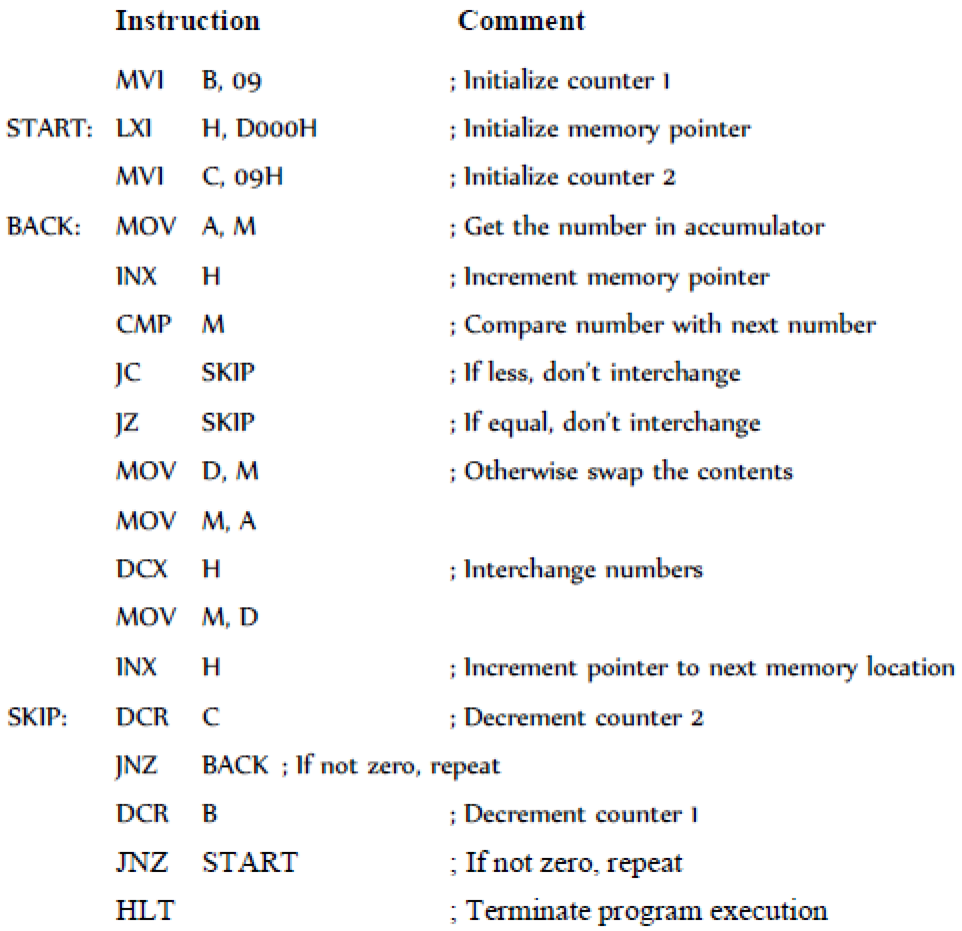
Step VIII : Increment the number of elements counter.

Step IX : Is count = N ? If yes, goto step XI else goto step II

Step X : Store the result.

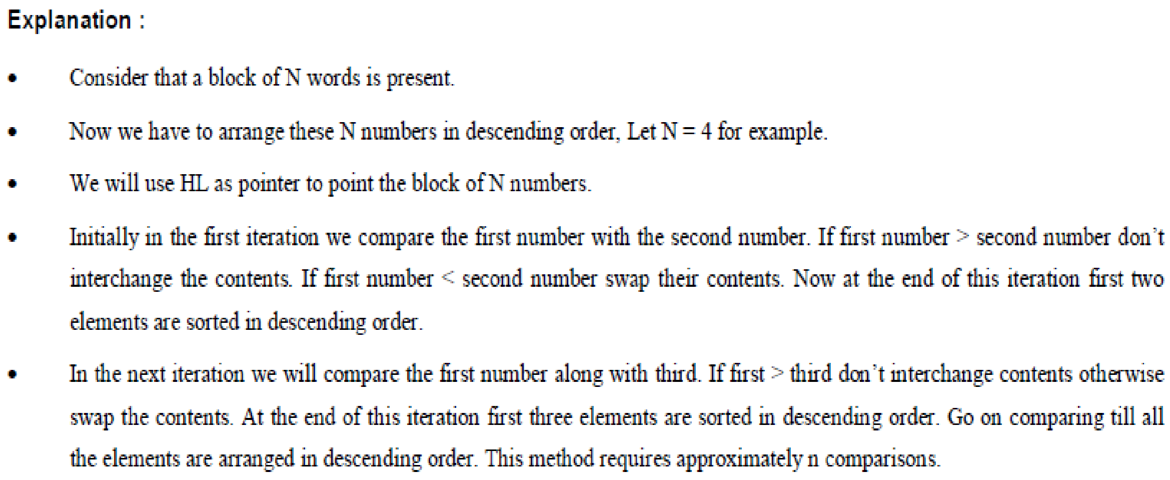
Step XI : Stop.

Program



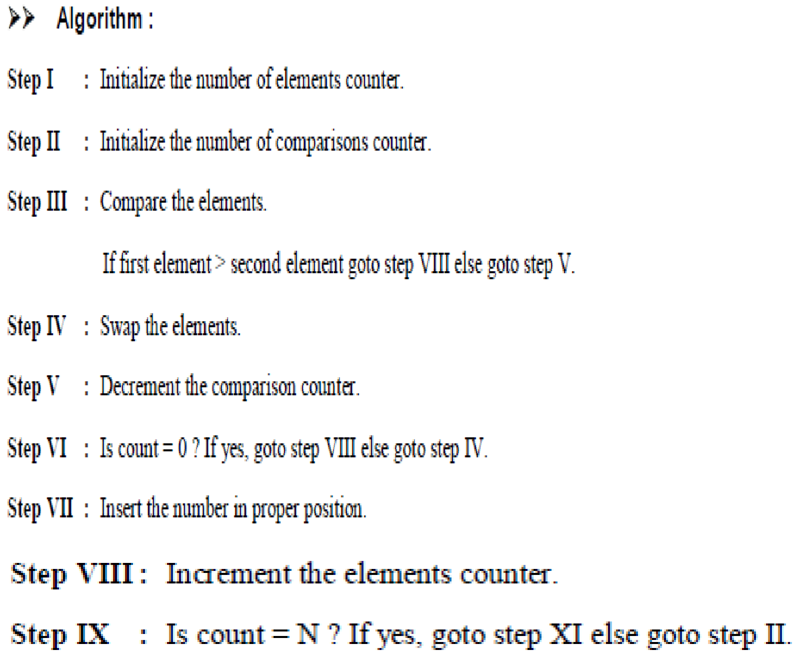
```
         Instruction          Comment
         MVI   B, 09          ; Initialize counter 1
START:   LXI   H, D000H       ; Initialize memory pointer
         MVI   C, 09H         ; Initialize counter 2
BACK:    MOV   A, M           ; Get the number in accumulator
         INX   H              ; Increment memory pointer
         CMP   M              ; Compare number with next number
         JC    SKIP           ; If less, don't interchange
         JZ    SKIP           ; If equal, don't interchange
         MOV   D, M           ; Otherwise swap the contents
         MOV   M, A
         DCX   H              ; Interchange numbers
         MOV   M, D
         INX   H              ; Increment pointer to next memory location
SKIP:    DCR   C              ; Decrement counter 2
         JNZ   BACK  ; If not zero, repeat
         DCR   B              ; Decrement counter 1
         JNZ   START          ; If not zero, repeat
         HLT                  ; Terminate program execution
```

Descending Order Program



Explanation :

- Consider that a block of N words is present.
- Now we have to arrange these N numbers in descending order, Let N = 4 for example.
- We will use HL as pointer to point the block of N numbers.
- Initially in the first iteration we compare the first number with the second number. If first number > second number don't interchange the contents. If first number < second number swap their contents. Now at the end of this iteration first two elements are sorted in descending order.
- In the next iteration we will compare the first number along with third. If first > third don't interchange contents otherwise swap the contents. At the end of this iteration first three elements are sorted in descending order. Go on comparing till all the elements are arranged in descending order. This method requires approximately n comparisons.



>> Algorithm :

Step I : Initialize the number of elements counter.

Step II : Initialize the number of comparisons counter.

Step III : Compare the elements.

If first element > second element goto step VIII else goto step V.

Step IV : Swap the elements.

Step V : Decrement the comparison counter.

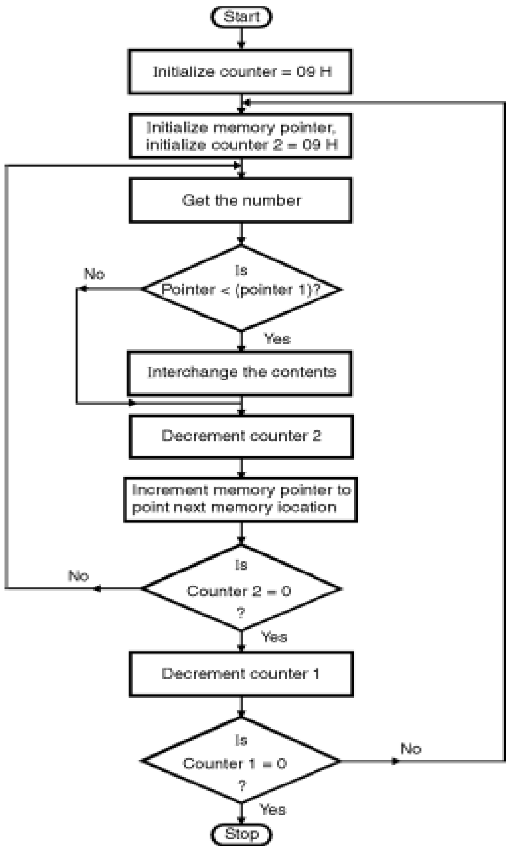
Step VI : Is count = 0 ? If yes, goto step VIII else goto step IV.

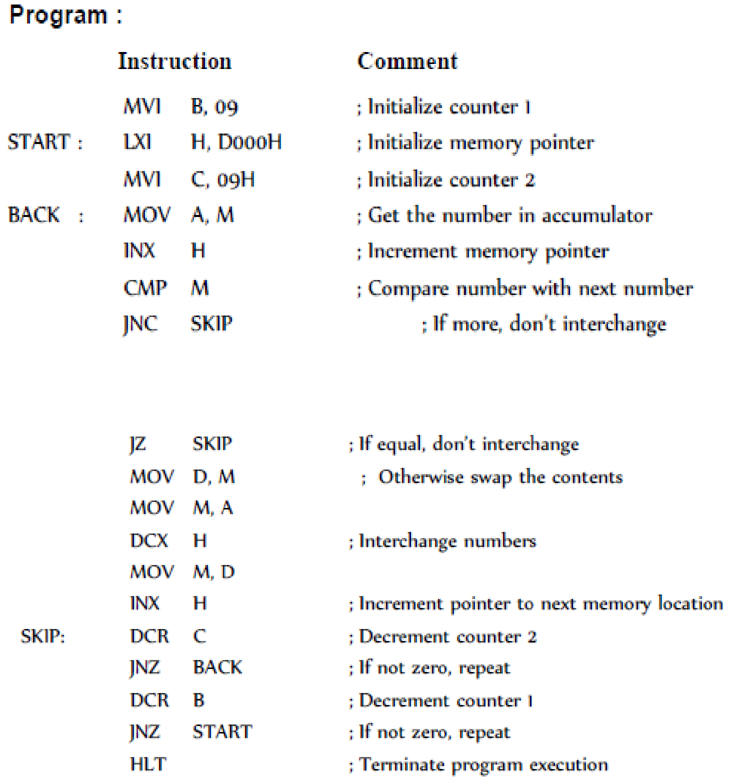
Step VII : Insert the number in proper position.

Step VIII : Increment the elements counter.

Step IX : Is count = N ? If yes, goto step XI else goto step II.

Step X : Stop


Start
Initialize counter = 09 H
Initialize memory pointer, initialize counter 2 = 09 H
Get the number
Is Pointer < (pointer 1)?
No
Yes
Interchange the contents
Decrement counter 2
Increment memory pointer to point next memory location
Is Counter 2 = 0 ?
No
Yes
Decrement counter 1
Is Counter 1 = 0 ?
No
Yes
Stop



Program :

```
            Instruction              Comment
            MVI    B, 09             ; Initialize counter 1
START :     LXI    H, D000H          ; Initialize memory pointer
            MVI    C, 09H            ; Initialize counter 2
BACK  :     MOV    A, M              ; Get the number in accumulator
            INX    H                 ; Increment memory pointer
            CMP    M                 ; Compare number with next number
            JNC    SKIP                      ; If more, don't interchange



            JZ     SKIP              ; If equal, don't interchange
            MOV    D, M                ; Otherwise swap the contents
            MOV    M, A
            DCX    H                 ; Interchange numbers
            MOV    M, D
            INX    H                 ; Increment pointer to next memory location
SKIP:       DCR    C                 ; Decrement counter 2
            JNZ    BACK              ; If not zero, repeat
            DCR    B                 ; Decrement counter 1
            JNZ    START             ; If not zero, repeat
            HLT                      ; Terminate program execution
```

Algorithm:

i. Load SI reg with pointer to array
ii. Load array length to CL & CH for two counters (CL for repetitions & CH for comparisons)
iii. REPEAT : Get an element into accumulator
iv. NEXT: Compare with next element
v. Check carry flag, if set goto SKIP
vi. Swap elements of array
vii. SKIP: Decrement CH and if not zero go to NEXT
viii. Decrement CL , if not zero go to REPEAT
ix. Halt

Program

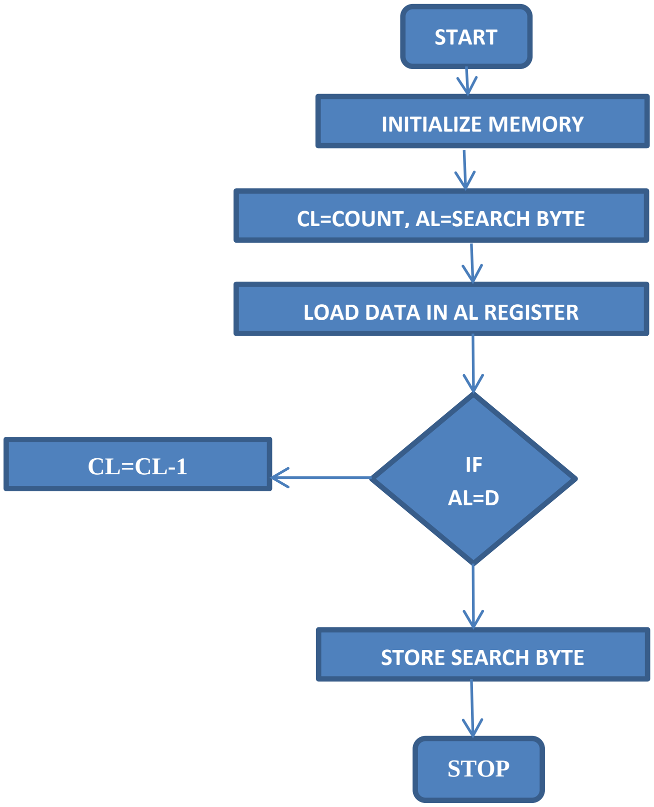
Label	Mnemonics
	MOV SI, 1500H
	MOV CL, [SI]
	DEC CL
REPEAT:	**MOV SI, 1500H**
	MOV CH, [SI]
	DEC CH

	INC SI
NEXT:	MOV AL, [SI]
	INC SI
	CMP AL, [SI]
	JC *SKIP/JNC SKIP*
	XCHG AL, [SI]
	XCHG AL, [SI - 1]
SKIP:	DEC CH
	JNZ *NEXT*
	DEC CL
	JNZ *REPEAT*
	INT 3

To write a program to search a number in a given array using 8086 microprocessor

ALOGORITHM:

1. Initialize the counter to the memory for staring the data and result.
2. Load DI with search byte
3. Load CL with count
4. Load AC with data from memory
5. Compare AC with DL if its equal
6. Store result else go to 2
7. Store the result
8. Stop the program.



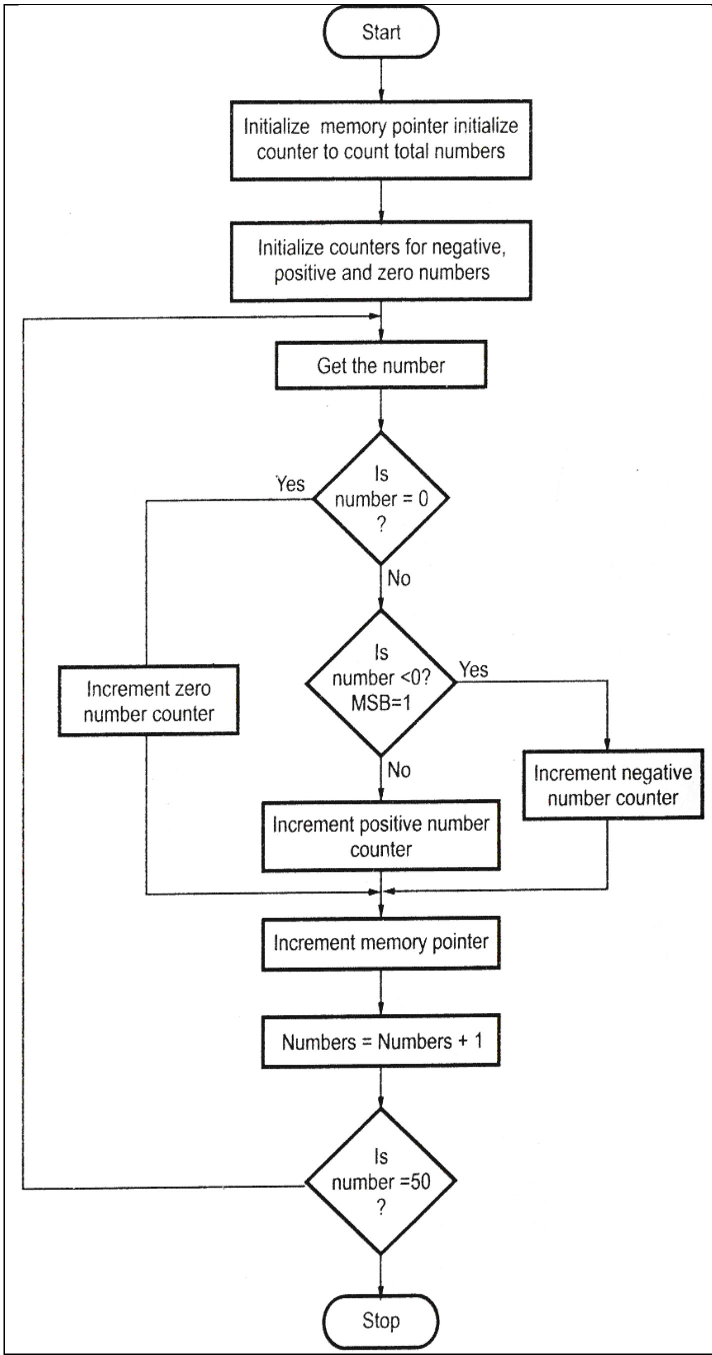
Label	**Mnemonics**	**Comments**
START	MOV SI,1100	Set SI reg for array
	MOV DI,1200	Load address of data to be searched
	MOV DI,[DL]	Get the data to search in DL reg
	MOV BL,01	Set BL reg as want
	MOV AL,[SI]	Get first element
AGAIN	CMP AL,DL	Compare an element of array
	JZ AVAIL	If data are equal then jump to avail
	INC SI	Increment SI
	INC BL	Increment BL count
	MOV AL,[SI]	
	CMP AL,20	Check for array
	JNZ AGAIN	If not JZ to again
NOT AVAIL	MOV CX,0000	Initialize CX to zero
	MOV [DI+1],CX	
	MOV [DI+3],CX	
	JMP 102F	
AVAIL	MOV BH,FF	
	MOV [DI+1],BH	Store FF to result
	MOV [DI+2],BL	Availability of data
	MOV [DI+3],SI	Store the address of data
	INT 3	Stop the program

Program to find the total no. of even and odd nos. from an array in Assembly Language

```
DATA SEGMENT
A DW 1,2,3,4,5,6,7,8,9,10
DATA ENDS
CODE SEGMENT
      ASSUME DS:DATA,CS:CODE
START:
    MOV AX,DATA
    MOV DS,AX
    LEA SI,A
    MOV DX,0000
    MOV BL,02
    MOV CL,10
  L1:MOV AX,WORD PTR[SI]
    DIV BL
    CMP AH,00
    JNZ L2
    INC DH
    JMP L3
  L2:INC DL
  L3:
    ADD SI,2
```

```
      DEC CL
      CMP CL,00
      JNZ L1
      MOV AH,4CH
      INT 21H
CODE ENDS
END START
```

Finding Positive and Negative Numbers in array


Start
Initialize memory pointer initialize counter to count total numbers
Initialize counters for negative, positive and zero numbers
Get the number
Is number = 0 ?
Yes
No
Increment zero number counter
Is number <0? MSB=1
Yes
No
Increment negative number counter
Increment positive number counter
Increment memory pointer
Numbers = Numbers + 1
Is number =50 ?
Stop

```
.MODEL SMALL
.DATA
        cr equ 0dh
        lf equ 0ah
        LIST dW 2579H, 0A500H,
0C009H,0159H,0B900H
    COUNT EQU 05H
.CODE
        mov ax,@data
        mov ds,ax

        XOR BX,BX
    XOR DX,DX
    MOV CL,COUNT
    MOV SI, OFFSET LIST
AGAIN:    MOV AX,[SI]
    SHL AX,01
        JC NEGAT
    INC BX
    JMP NEXT
NEGAT:  INC DX
NEXT:    ADD SI,02
    DEC CL
    JNZ AGAIN

    MOV ah,4ch
        INT 21h

        end
```

Block transfer

The 8 data bytes are stored from memory location E000H to E007H. Write 8086 ALP to transfer the block of data to new location B001H to B008H

	MOV BL, 08H MOV CX, E000H MOV EX, B001H Loop: MOV DL, [CX] MOV [EX], DL DEC BL JNZ loop HLT

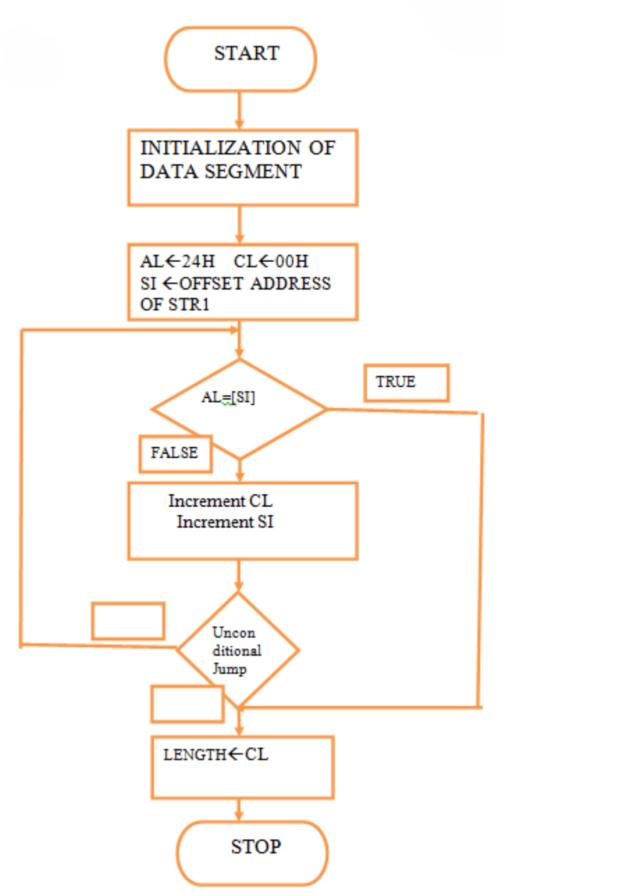
Write algorithm to transfer block of data from source address to destination address and vice versa [overlapping block transfer].

The concept of swapping blocks by including a third temporary block is used in the following algorithm.

Algorithm:

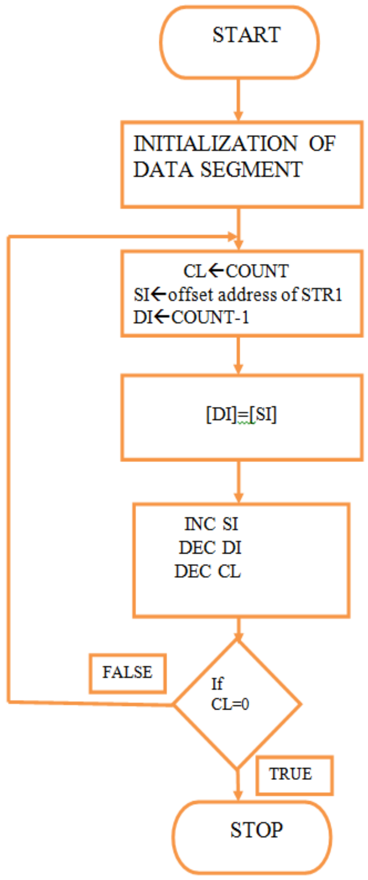
1. Initialize data segment
2. Initialize word counter.
3. Initilize memory pointers for destination and temporary array.
4. Read numbers from destination array.
5. Copy it to temporary array.
6. Increment memory pointer for destination and temporary array for next number.
7. Decrement word counter by one.
8. If word counter not equal to zero then go to step 4.
9. Initilize memory pointers for source and destination array.
10. Read numbers from source array.
11. Copy it to destination array.
12. Increment memory pointer for source and destination array for next number.
13. Decrement word counter by one.
14. If word counter not equal to zero then go to step 10.
15. Initilize memory pointers for temporary and source array.
16. Read numbers from temporary array.
17. Copy it to source array.
18. Increment memory pointer for temporary and source array for next number.
19. Decrement word counter by one.
20. If word counter not equal to zero then go to step 16.
21. Stop.

To write an assembly language program to find the length of the given string.



```
ASSUME CS : CODE, DS : DATA
CODE SEGMENT
MOV AX, DATA
MOV DS, AX
MOV AL, ’$’
MOV CX, 00H
MOV SI, OFFSET STR1
BACK : CMP AL, [SI]
JE GO
INC CL
INC SI
JMP BACK
GO : MOV LENGTH, CL
HLT
CODE ENDS
DATA SEGMENT
STR1 DB ‘STUDENT BOX OFFICE$’
LENGTH DB ?
DATA ENDS
END
```

To write an assembly language program to reverse the given string.



```
ASSUME CS : CODE, DS : DATA
CODE SEGMENT
MOV AX, DATA
MOV DS, AX
MOV CL, COUNT
MOV SI, OFFSET STR1
MOV DI, COUNT - 1
BACK:MOV AL, [SI]
XCHG [DI], AL
MOV [SI], AL
INC SI
DEC DI
DEC CL
JNZ BACK
HLT
CODE ENDS
DATA SEGMENT
STR1 DB ‘MPMC$’
COUNT EQU 04H
STR2 DB DUP (0)
DATA ENDS
END
```

Write ALP to concatenate two strings with algorithm
String 1 : “Maharashtra board”
String 2 : “ of technical Education”

Algorithm:

1. Initialize data segment.
2. Initialize memory pointers for source and destination string.
3. Move memory pointer of source string to the end of string.
4. Move memory pointer of destination string to the end of string.
5. Copy characters from destination string to source string.
6. Stop.

ALP:

```
.MODEL SMALL
.DATA
STR_S DB ‘Maharashtra board $’
STR_D DB ‘of technical Education $’
.CODE
MOV AX, @DATA
MOV DS, AX
MOV SI, OFFSET STR_S
NEXT:
MOV AL, [SI]
CMP AL, ’$’
JE EXIT
INC SI
JMP NEXT
EXIT:
MOV DI, OFFSET STR_D
UP: MOV AL, [DI]
CMP AL, ’$’
JE EXIT1
MOV [SI], AL
INC SI
INC DI
JMP UP
EXIT1:
MOV AL, ’$’
MOV [SI], AL
MOV AH, 4CH
INT 21H
ENDS
END
```

(OR)

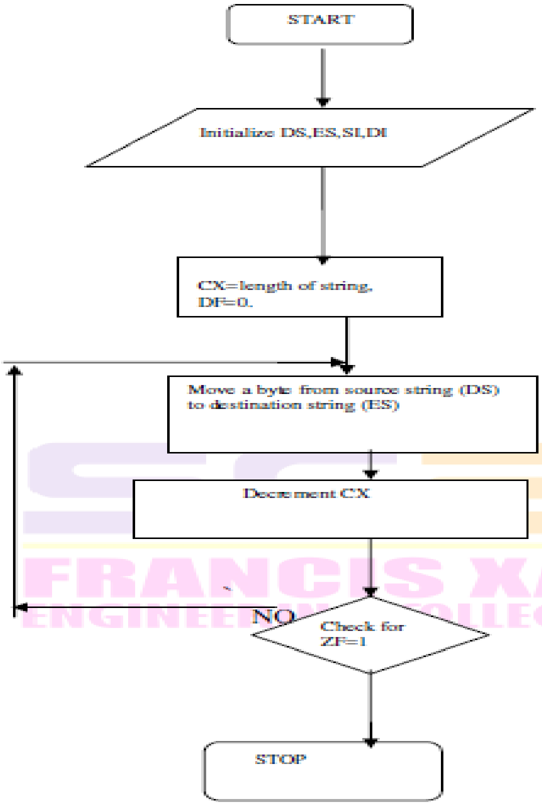
```
DATA SEGMENT
ST1 DB " Maharashtra board$"
LEN1 DB 0
ST2 DB " of technical Education$"
LEN2 DB 0
R DB ?
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE, ,DS:DATA, ES:DATA
START: MOV AX,DATA
MOV DS,AX
```

	MOV ES,AX MOV SI, OFFSET ST1 ; Length of the first string in LEN1 MOV AL,'$' NEXT1: CMP AL,[SI] JE EXIT1 INC LEN1 INC SI JMP NEXT1 EXIT1: MOV SI, OFFSET ST2 ; Length of the second string in LEN2 NEXT2: CMP AL,[SI] JE EXIT2 INC LEN2 INC SI JMP NEXT2 EXIT2: MOV SI, OFFSET ST1 ; copy first string to R MOV DI, OFFSET R MOV CL, LEN1 REP MOVSB MOV SI, OFFSET ST2 ; Concat second string to R MOV CL, LEN2 REP MOVSB MOV AH,4CH INT 21H CODE ENDS END START

COPYING A STRING

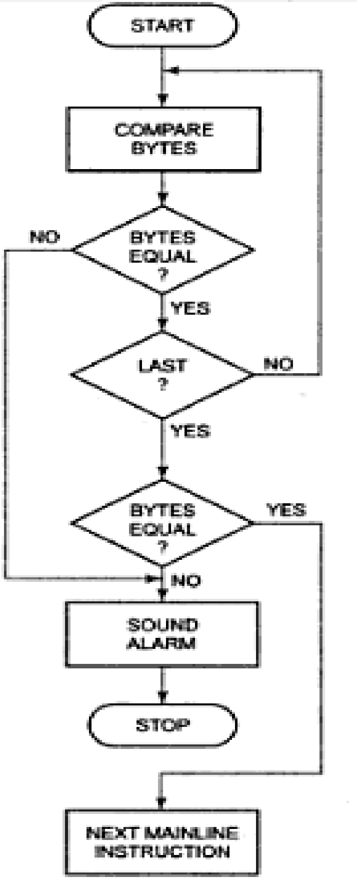
ALGORITHM:

a. Initialize the data segment .(DS)
b. Initialize the extra data segment .(ES)
c. Initialize the start of string in the DS. (SI)
d. Initialize the start of string in the ES. (DI)
e. Move the length of the string(FF) in CX register.
f. Move the byte from DS TO ES, till CX=0.



```
PROGRAM          COMMENTS
MOV SI,1200H     ;Initialize destination address
MOV DI,1300H     ;Initialize starting address
MOV CX,0006H     ;Initialize array size
CLD              ;Clear direction flag
REP MOVSB        ;Copy the contents of source into destination until count reaches ;zero
HLT               ;Stop
```

Using compare string byte to check password-flowchart



ADDRESS	MNEMONICS	OR	**Strings Comparison:**
1100	LEA SI, [1200]		**ASM CODE:** . Model small . Stack . Data Strg1 db ‘lab’,’$’ Strg 2 db ‘lab’, $’ Res db ‘strg are equal’,’$’ Res db ‘strg are not equal’,’$’ Count equ 03h . Code
1104	LEA DI, [1300]		
1108	MOV CX, 0003H		
110b	CLD		
110c	REPE CMPSB		
110e	JNZ NOTEQUAL		
1110	MOV AL, 01		
1112	MOV [1400], AL		
1115	HLT		
1116	NOTEQUAL: MOV AL, 00		
1118	MOV [1400], AL		
111b	HLT		

			Mov ax, @data Mov ds, ax Mov es, ax Lea si, strg1 Lea di, strg2 Cld Rep cmpsb Jnz loop1 Mov ah, 09h Lea dx, res Int 21h Jmp a1 Loop1: mov ah, 09h Lea dx, re1 Int 21h A1: mov ah, 4ch Int 21h End

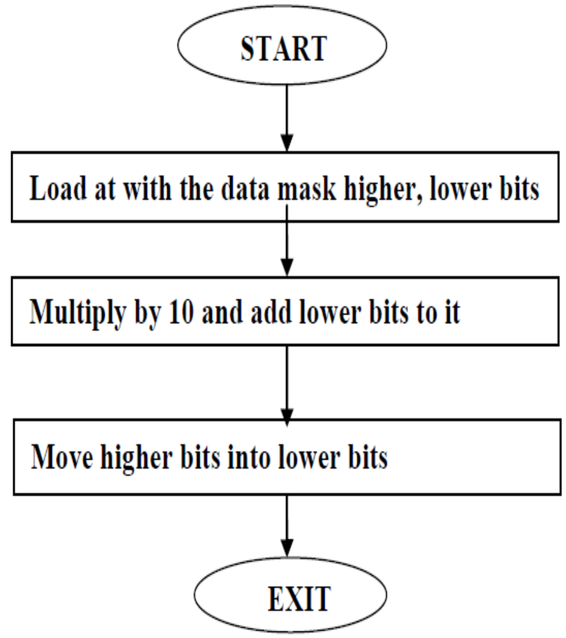
Write an ALP to count the number of „1“ in a 16 bit number.
Assume the number to be stored in BX register. Store the result in CX register.

```
.MODEL SMALL
.DATA
NUM DW 0008H
ONES DB 00H
.CODE
MOV AX, @DATA ; initialize data segment
MOV DS, AX
MOV CX, 10H ; initialize rotation counter by 16
MOV BX, NUM ;load number in BX
UP: ROR BX, 1 ; rotate number by 1 bit right
```

```
JNC DN ; if bit not equal to 1 then go to dn
INC ONES ; else increment ones by one
DN: LOOP UP
;decrement rotation counter by 1 anf if not zero then go to up
MOV CX, ONES ;move result in cx register.
MOV AH, 4CH
INT 21H

ENDS
END ; end of program.
```

BCD TO HEXA DECIMAL CONVERSION


START
Load at with the data mask higher, lower bits
Multiply by 10 and add lower bits to it
Move higher bits into lower bits
EXIT

ALGORITHM:

1. Load the data in AL register.

2. Separate higher nibbles and (in) lower nibbles.

3. Move the higher nibbles (in) to lower nibbles position.

4. Multiply AL by 10.

5. Add lower nibbles.

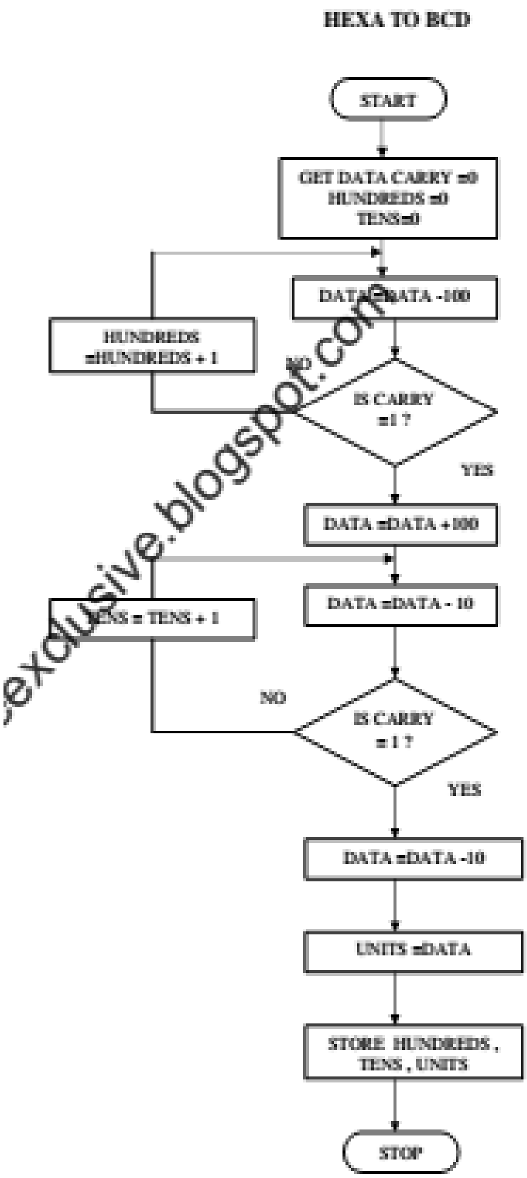
6. Store the result into Memory.

MNEMONICS	COMMENTS
MOV AL,10	Load register AL with the data 10
MOV AH,AL	Load AL value into AH
AND AH,0F	Mask higher bits
MOV BL,AH	Load AH value into BL
AND AL,F0	Mask lower bits
MOV CL,04	Load 04 value into CL
ROR AL,CL	Rotate the data from last 4bits to first 4 bits
MOV BH,0A	Load 10 value into BH
MUL BH	Multiply by 10
ADD AL,BL	Add lower nibble to the multiplied data
INT3	Break point

Hex to BCD number conversion

ALGORITHM-

1. Start
2. Load the hexadecimal number into a memory location
3. Divide the number separately by 64 (H) and 0A(H) and store the quotients separately in memory location
4. The last unit digit remainder is stored separately in successive memory location
5. Stop



```
DATA SEGMENT
HEX DB 0AFH
BCD DW 0
CNT DB 0
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE,DS:DATA
START:
MOV AX,DATA
MOV DS,AX
MOV AL,HEX
CMP AL,00
JZ LAST

LOOP1:
MOV AH,00
MOV BL,0AH
DIV BL
MOV DH,00
MOV DL,AH
MOV BL,AL
MOV AL,04
MUL CNT
MOV CL,AL
ROL DX,CL
OR BCD,DX
MOV AL,BL
INC CNT
CMP AL,0
JNZ LOOP1
LAST:INT 3
CODE ENDS
END START
END
```