REPORT ON IOT

TEAM MEMBERS:

1) Sumaya Islam (201-15-13675)

2) Dipanita Mondol (201-15-13617)

3) Tushar Ahmed Nirma (201-15-13701)

SUBMITTED TO:

Syada Tasmia Alvi

Lecturer

Department: CSE

Daffodil International University

1. STRING: Lower Case To Upper Case

The Code is:

```
.MODEL SMALL
.STACK 100H
.CODE

MAIN PROC

   MOV AH, 1      ;Input Function
   INT 21H        ;Interrupt
   MOV BL, AL     ;Putting input into BL for Output

   Sub BL, 32

   MOV AH, 2
   MOV DL, 10     ;New line
   INT 21H

   MOV AH, 2          ;Output Function
   MOV DL, 13         ;OUTPUT
   INT 21H            ;Interrupt
```

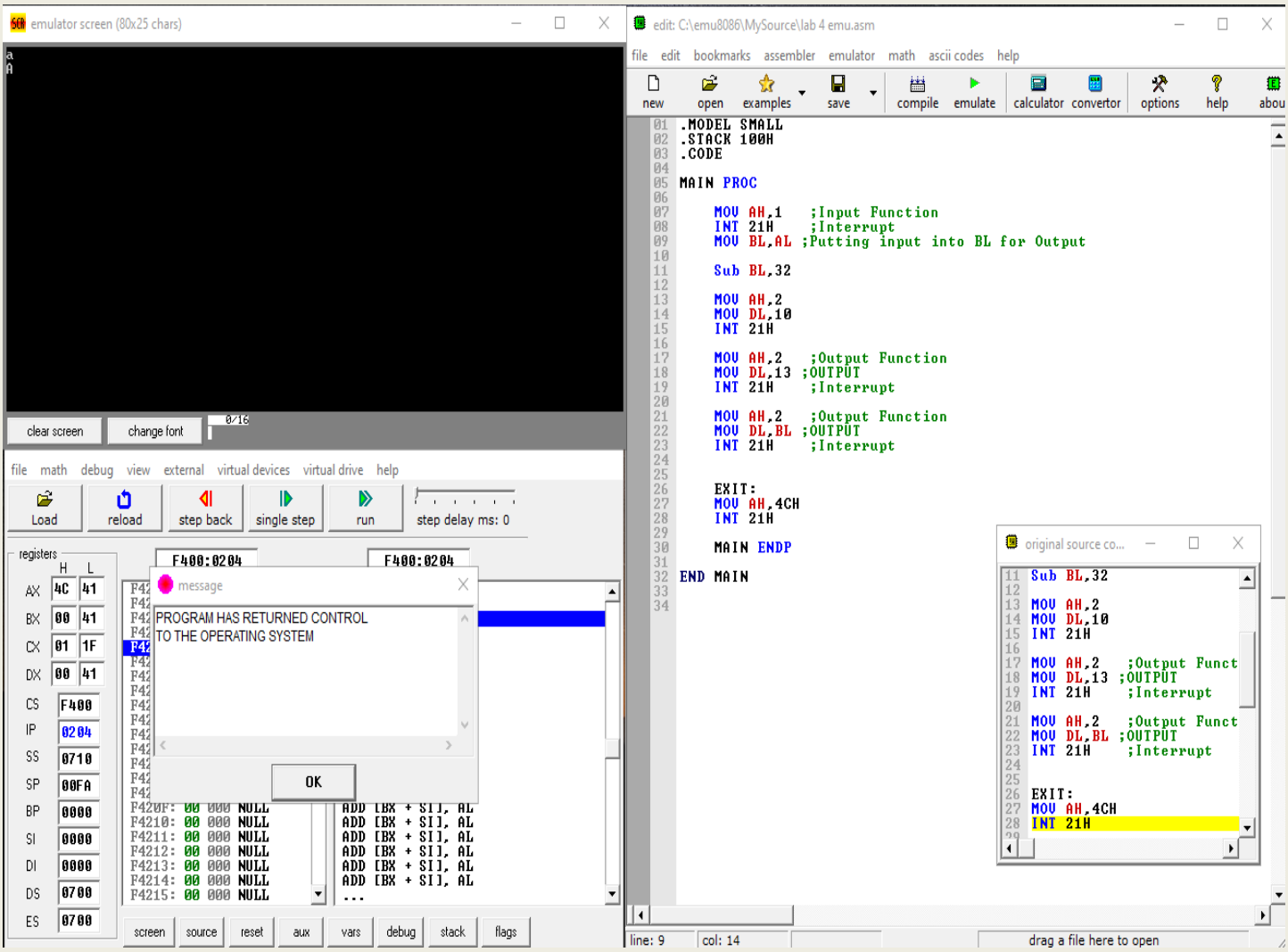
```
    MOV AH, 2        ;Output Function
    MOV DL, BL       ;OUTPUT
    INT 21H          ;Interrupt
    EXIT:
    MOV AH, 4CH
    INT 21H

    MAIN ENDP

END MAIN
```

Description:

First of all we selects a standard memory model, sets the size of the program stack for the programs. Then we use .CODE which identifies that contains instructions. Next create a name and an address for the beginning of a procedure. For single key input and interrupt we use MOV AH,1 and INT 21H. Then move the value of AL to BL. Subtract the value with 32 for getting the decimal number. For a new line used MOV DL,10 and for remove the space MOV DL,13 is used. For displaying output use MOV AH,2. Indicates the end of the procedure and terminates we use MAIN ENDP and END MAIN. For this code if we input a lower case or small letter then we will get the upper case or capital letter of it.

.

The Output is:



2. Factorial

The Code is:

```
.MODEL SMALL
.STACK  100H

.DATA
a DW 3
f DW 1

.CODE
MAIN PROC
   MOV Ax, @DATA
   MOV DS, AX

   Mov Ax,f
   Mov Bx,a

   L:
    Mul Bx
    Dec Bx


   CMP Bx, 0
```

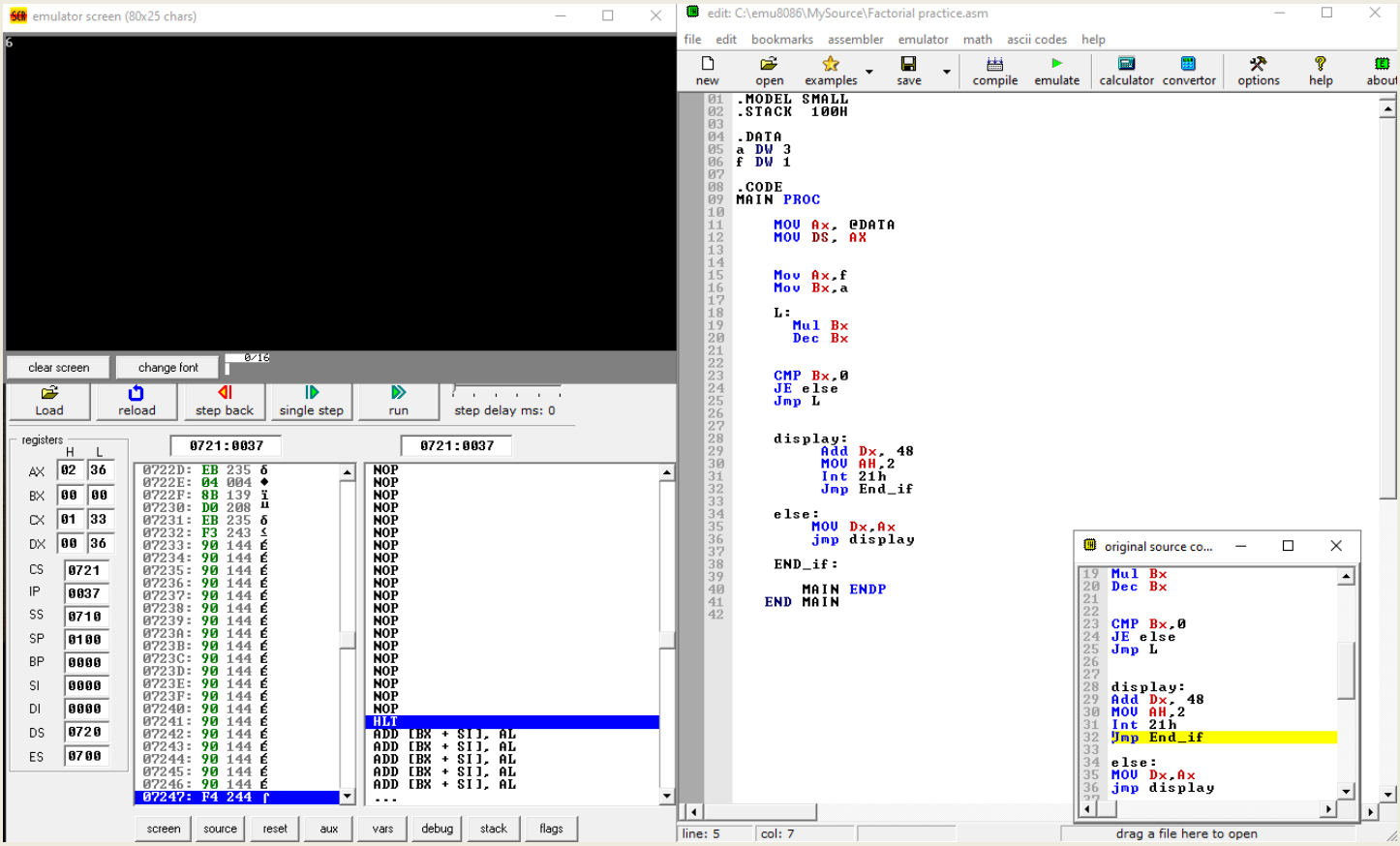
```
JE else
Jmp L



display:
    Add Dx, 48
    MOV AH,2
    Int 21h
    Jmp End_if

else:
   MOV Dx,Ax
   jmp display

END_if:

  MAIN ENDP
END MAIN
```

Description:

First of all we selects a standard memory model, sets the size of the program stack for the programs. Pertain all variables to the program we use .DATA. In it we declare two variables. Then we use .CODE which identifies that contains instructions. Next create an address for the beginning of a procedure. Then move variables “a” & “f” respectively Ax & Bx. Next we create a level named “L” where Bx will multiplied by the value of “a”

and decrease the value. Then compare if the value of Bx is eual to 0 or not. If yes then jump the level again. If the result is equal then jump to else level. In “display” level, add 48 with the value of Dx, display it and then jump to end. In level “else” move the value of Ax to Dx and jump to display level. Indicating the end of the procedure and terminates we use MAIN ENDP and END MAIN. By this code we will get the factorial of 3.

The Output is:



3. Star Pattern

The Code is:

```
.MODEL SMALL
.STACK  50H
.DATA
   NL  DB  0DH, 0AH, '$'        ; NL = NEXT LINE
.CODE
MAIN PROC

   MOV AX, @DATA
   MOV DS, AX

   MOV CX, 3
   MOV BX, 1

   FOR_1:
      PUSH    CX
      MOV DL, 20H              ; 20H IS ASCII CODE FOR SPACE
      MOV AH, 2

   FOR_2:
       INT 21H                 ; PRINTING SPACES
```

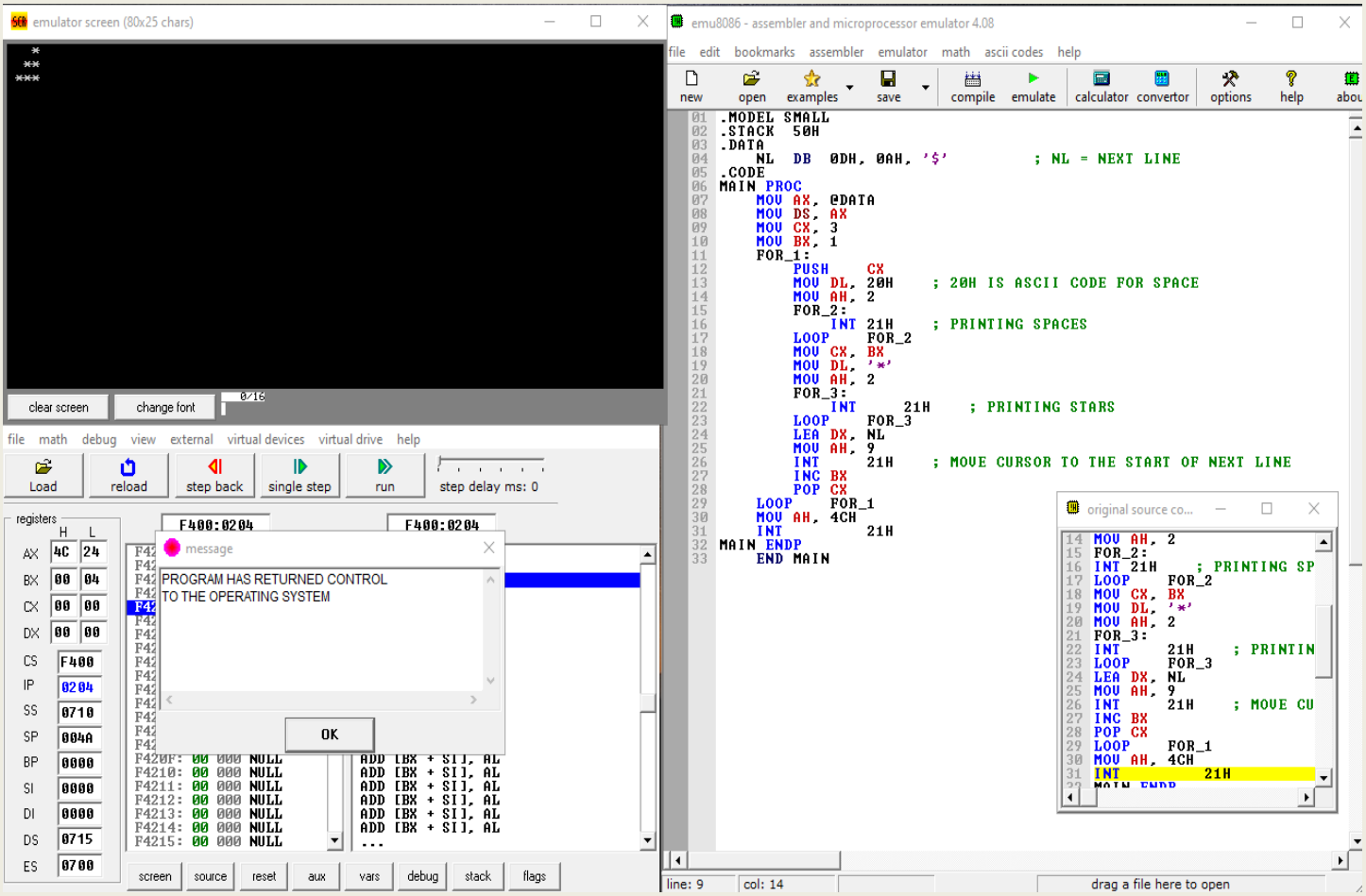
```
    LOOP    FOR_2
    MOV CX, BX
    MOV DL, '*'
    MOV AH, 2

 FOR_3:
    INT     21H                 ; PRINTING STARS
    LOOP    FOR_3
    LEA DX, NL

    MOV AH, 9
    INT     21H                 ; MOVE CURSOR TO THE START OF NEXT LINE

    INC BX
    POP CX

   LOOP    FOR_1
   MOV AH, 4CH
   INT       21H
MAIN ENDP
   END MAIN
```

Description:

First of all we selects a standard memory model, sets the size of the program stack for the programs. Pertain all variables to the program we use .DATA. Then we use .CODE

which identifies that contains instructions. Next create a name and an address for the beginning of a procedure. Addressing data segment use MOV AX, @DATA and MOV DS, AX. Then we move 3 and 1 to the register CX and BX. Then we create a level named FOR_1. In this level we push CX and for a space we use 20H ASCII value and display it. In FOR_2 level we also print a space by using INT 21H and create a loop with level FOR_2. Then we move BX in CX, Star sign "*" in DL register and display it. In level For_3, print the stars and return to level FOR_2 again. By using load effective address or lea, copy the source offset address into the destination. Then display the string by MOV AH, 9. Then increase the value of BX and pop the CX and go to the FOR_1 level again. Indicates the end of the procedure and terminates we use MAIN ENDP and END MAIN. By this code we can get the star pattern as an output.

The Output is:



Reference for Star Pattern

4. Even And Odd

The Code is:

```
.MODEL SMALL
.STACK 100H

.DATA
msg DB 10,13,'Enter number=$'
msg1 DB 10,13,'Number is even$'
msg2 DB 10,13,'Number is odd$'
msg3 DB 10,13, 'Case Convertion=$'

.CODE
  MOV AX,@DATA
  MOV DS,AX
  lea DX, msg
  MOV AH, 9
  INT 21H

  MOV AH,1
  INT 21H

  MOV BL,AL
```

```
CMP BL,'9'
JA cc
SAR bl,1
JC odd
lea DX, msg1
MOV AH,9
INT 21H
JMP exit

odd:
lea DX, msg2
MOV AH, 9
INT 21h

JMP exit

cc:
lea DX, msg3
  MOV AH, 9
  INT 21h

CMP bl,'A'
JNLE next

next:
```

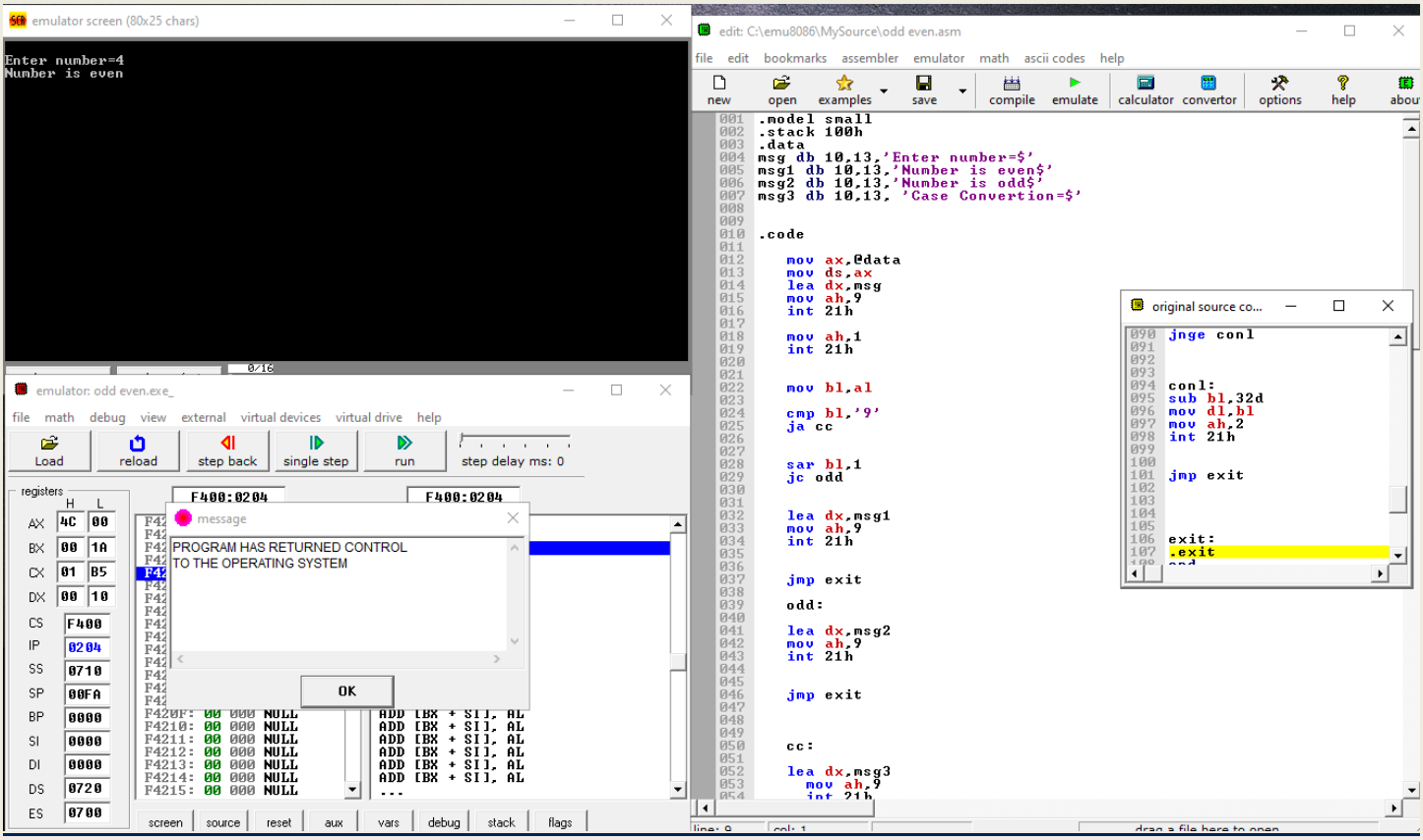
```
 CMP BL,'Z'
 JNGE con

 JMP lower

con:
  ADD BL, 32d
  MOV DL, BL
  MOV AH, 2
  INT 21h

  JMP exit

lower:
   CMP BL,'a'
   JNLE ln

 ln:
    CMP BL,'z'
    JNGE conl

conl:
  SUB BL,32d
  MOV DL,BL
  MOV AH,2
```

```
INT 21h


JMP exit


exit:

.exit

End
```

Description:

First of all we selects a standard memory model, sets the size of the program stack for the programs. Pertain all variables to the program we use .DATA and pass some string which will give some input command to user. Then we use .CODE which identifies that contains instructions. Addressing data segment use MOV AX, @DATA and MOV DS, AX. By using load effective address or lea, copy the source offset address into the destination and display it by move ah,9 and INT 21h. For single key input and interrupt we use MOV AH,1 and INT 21H. After move bl,al for characters compare that the input is less than or equal to '9'. If it is not then jump cc if there is carries. Then we shift and rotate bl and 1 and jump if there is carry. If there is no carry then, load the message 1 and display. In level “cc”, load the message 3 and compare bl and “A”. Then jump next if it is greater or equal to “A” otherwise jump lower. In “con” level, we add 32 decimal in bl register and move it in dl then display it. In “lower” and “ln” level again we compare bl with “A” & “Z” and respectively jump level “ln” if it is less than or equal and jump level “conl” if it is greater than or equal. In “conl” level, we subtract 32 from Bl, move it to DL and display. At the last we jump exit and end the procedure. By this code we can get that the given number is even or odd.

The Output is:



Reference for Odd Even Number

5. Check Number

The Code is:

```
.MODEL SMALL

.STACK 100H


.DATA

a dw "positive$"

b dw "negative$"

c dw "zero$"


.CODE

MAIN PROC

    mov ax,@data

    mov ds,ax


    mov AH,1

    int 21h

    mov BX,AX


    cmp BX,0
```

```
    JL NEGA
    JE ZERO
    JG POST

NEGA:
    lea dx,b
    mov ah,9
    int 21h
    JMP exit

ZERO:
    lea dx,c
    mov ah,9
    int 21h
    JMP exit

POST:
    lea dx,a
    mov ah,9
    int 21h
    JMP exit

   exit:
   MAIN ENDP
```

END MAIN

Description:

First of all we selects a standard memory model, sets the size of the program stack for the programs. Pertain all variables to the program we use .DATA and pass some string which will give some output. Then we use .CODE which identifies that contains instructions. Addressing data segment we use MOV AX, @DATA and MOV DS. For single key input and interrupt we use MOV AH,1 and INT 21H. After moving the address al to bl, compare that the input with '0'. If value is less than, then jump to level “NEGA”. If value is equal then jump level “ZERO” and if it is greater than then jump level “POST”. In “NEGA” level we load the variable b means negative will show in output. In “ZERO” level we load the variable c means zero will show in output. In “POST” level we load the variable a means positive will show in output. By this code we can identify the input number is zero, positive or negative.

The Output is: