



**Daffodil**  
*International*  
**University**

# 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 select a standard memory model, set 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:

The screenshot displays an x86 emulator interface with two main windows: 'emulator screen (80x25 chars)' and 'edit: C:\emu8086\MySource\lab 4 emu.asm'.

The emulator screen shows a black display area with controls for 'clear screen', 'change font', and '0/16'. Below the screen are buttons for 'Load', 'reload', 'step back', 'single step', 'run', and a 'step delay ms: 0' slider. A 'registers' panel on the left lists registers AX through ES with their high (H) and low (L) bytes. The IP register is highlighted at 0204.

A message box titled 'message' is open, displaying the text: 'PROGRAM HAS RETURNED CONTROL TO THE OPERATING SYSTEM'. It has an 'OK' button.

The assembly editor window shows the following code:

```
01 .MODEL SMALL
02 .STACK 100H
03 .CODE
04
05 MAIN PROC
06
07     MOV AH,1    ;Input Function
08     INT 21H     ;Interrupt
09     MOV BL,AL   ;Putting input into BL for Output
10
11     Sub BL,32
12
13     MOV AH,2
14     MOV DL,10
15     INT 21H
16
17     MOV AH,2    ;Output Function
18     MOV DL,13   ;OUTPUT
19     INT 21H     ;Interrupt
20
21     MOV AH,2    ;Output Function
22     MOV DL,BL   ;OUTPUT
23     INT 21H     ;Interrupt
24
25
26 EXIT:
27     MOV AH,4CH
28     INT 21H
29
30 MAIN ENDP
31
32 END MAIN
33
34
```

A smaller window titled 'original source co...' shows a zoomed-in view of the assembly code, highlighting the 'EXIT:' section (lines 26-28):

```
26 EXIT:
27     MOV AH,4CH
28     INT 21H
```

The status bar at the bottom indicates 'line: 9' and 'col: 14'.

## 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

Jump L

display:

Add Dx, 48

MOV AH,2

Int 21h

Jump End\_if

else:

MOV Dx,Ax

jmp display

END\_if:

MAIN ENDP

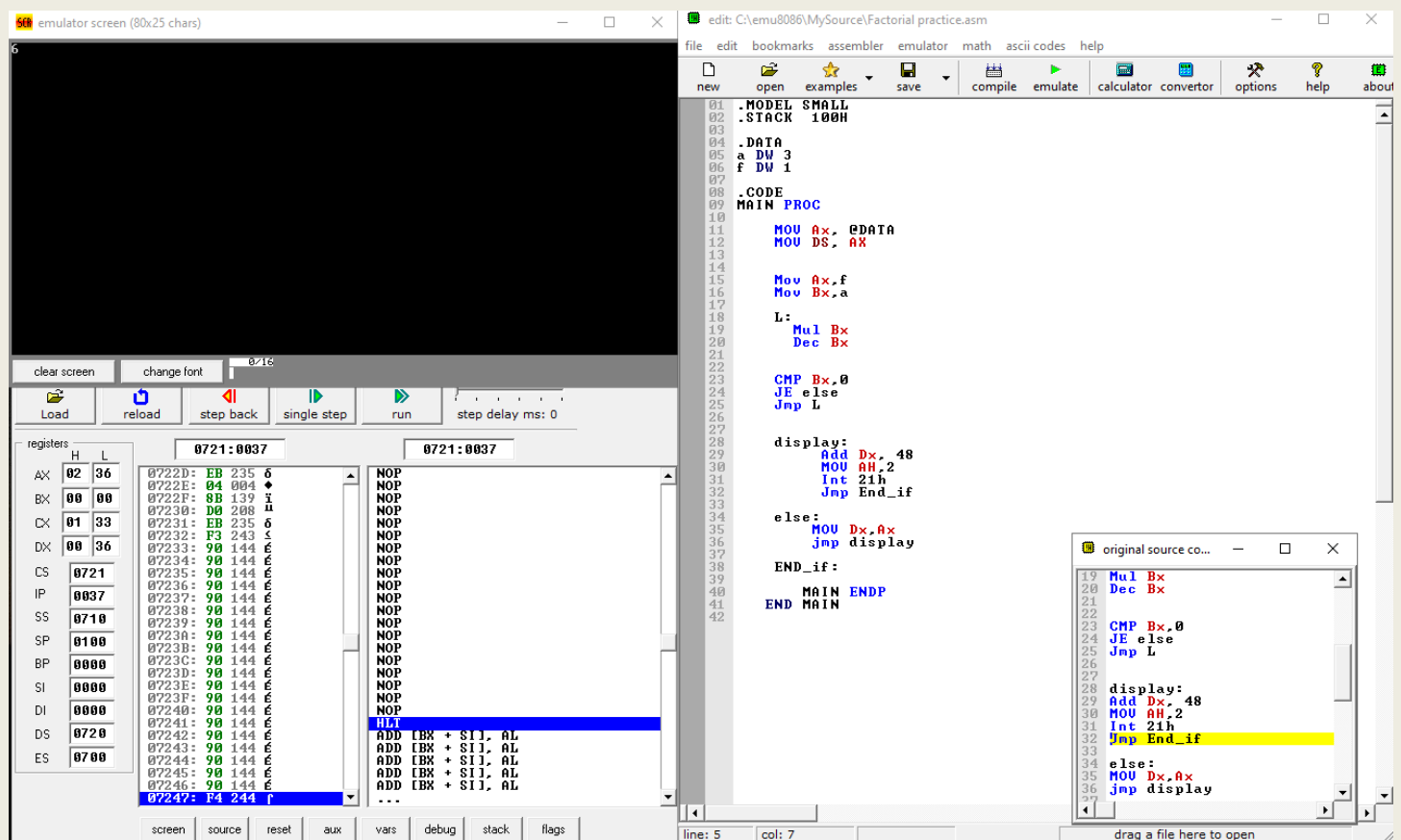
END MAIN

### Description:

First of all we select a standard memory model, set 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 label named "L" where Bx will be multiplied by the value of "a"

and decrease the value. Then compare if the value of Bx is equal 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:



The screenshot displays an x86 emulator interface with three main windows:

- emulator screen (80x25 chars):** Shows a black screen with a cursor at the top left.
- edit: C:\emu8086\MySource\Factorial practice.asm:** Contains the assembly code for calculating the factorial of 3. The code includes a main procedure that initializes registers, performs a loop to calculate the factorial, and a display routine to output the result.
- original source co...:** A smaller window showing a portion of the assembly code, highlighting the loop and display sections.

The assembly code in the main window is as follows:

```
01 .MODEL SMALL
02 .STACK 100H
03
04 .DATA
05 a DW 3
06 f DW 1
07
08 .CODE
09 MAIN PROC
10
11     MOV AX, @DATA
12     MOV DS, AX
13
14
15     MOV AX, f
16     MOV BX, a
17
18     L:
19         MUL BX
20         DEC BX
21
22
23     CMP BX, 0
24     JE else
25     JMP L
26
27
28     display:
29         ADD DX, 48
30         MOV AH, 2
31         INT 21h
32         JMP End_if
33
34     else:
35         MOV DX, AX
36         JMP display
37
38     END_if:
39
40     MAIN ENDP
41
42     END MAIN
```

The registers window shows the following values:

Register	H	L
AX	02	36
BX	00	00
CX	01	33
DX	00	36
CS	0721	
IP	0037	
SS	0710	
SP	0100	
BP	0000	
SI	0000	
DI	0000	
DS	0720	
ES	0700	

# 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

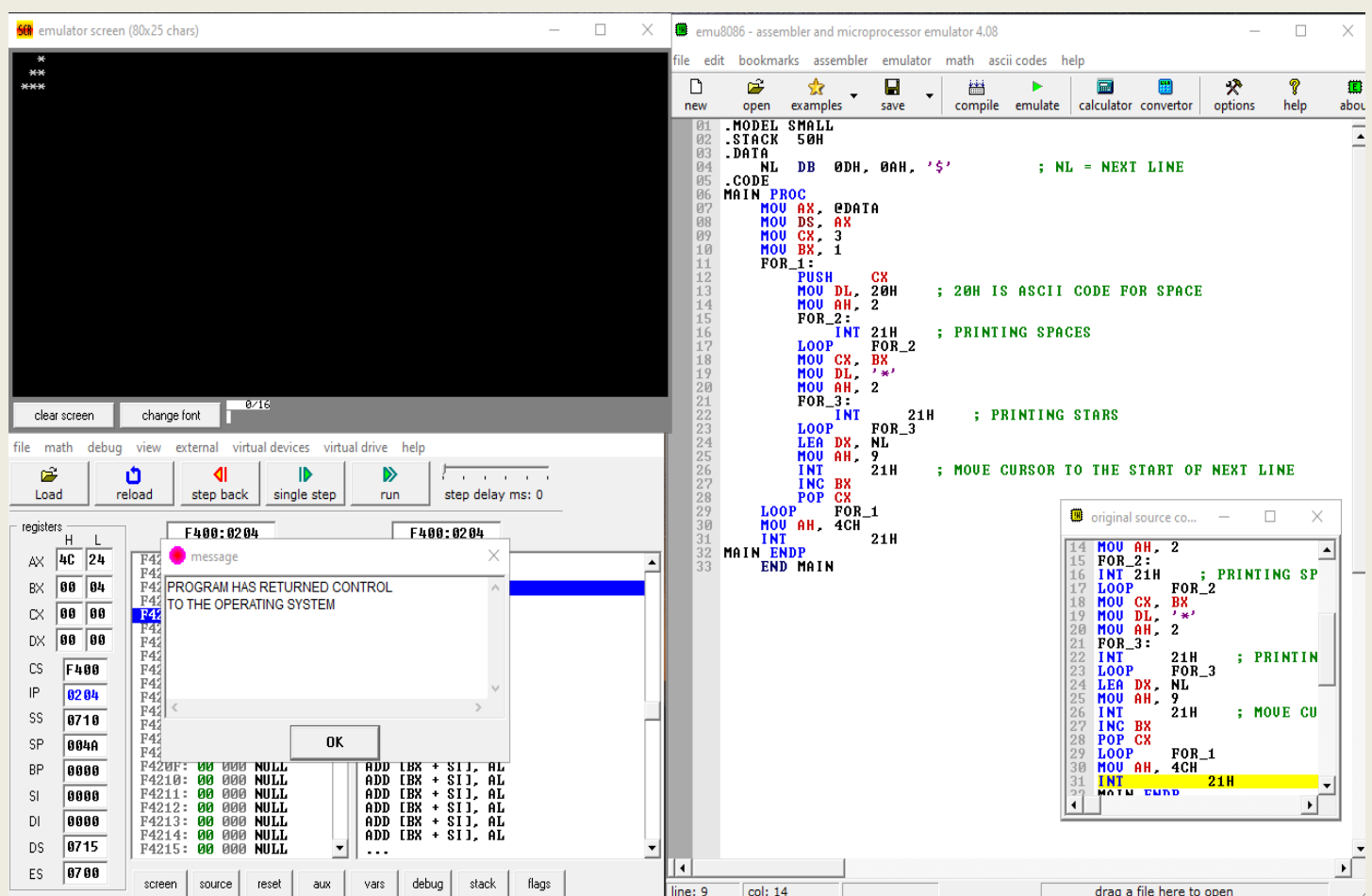
```

### Description:

First of all we select a standard memory model, set 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 Conversion=\$'

.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:

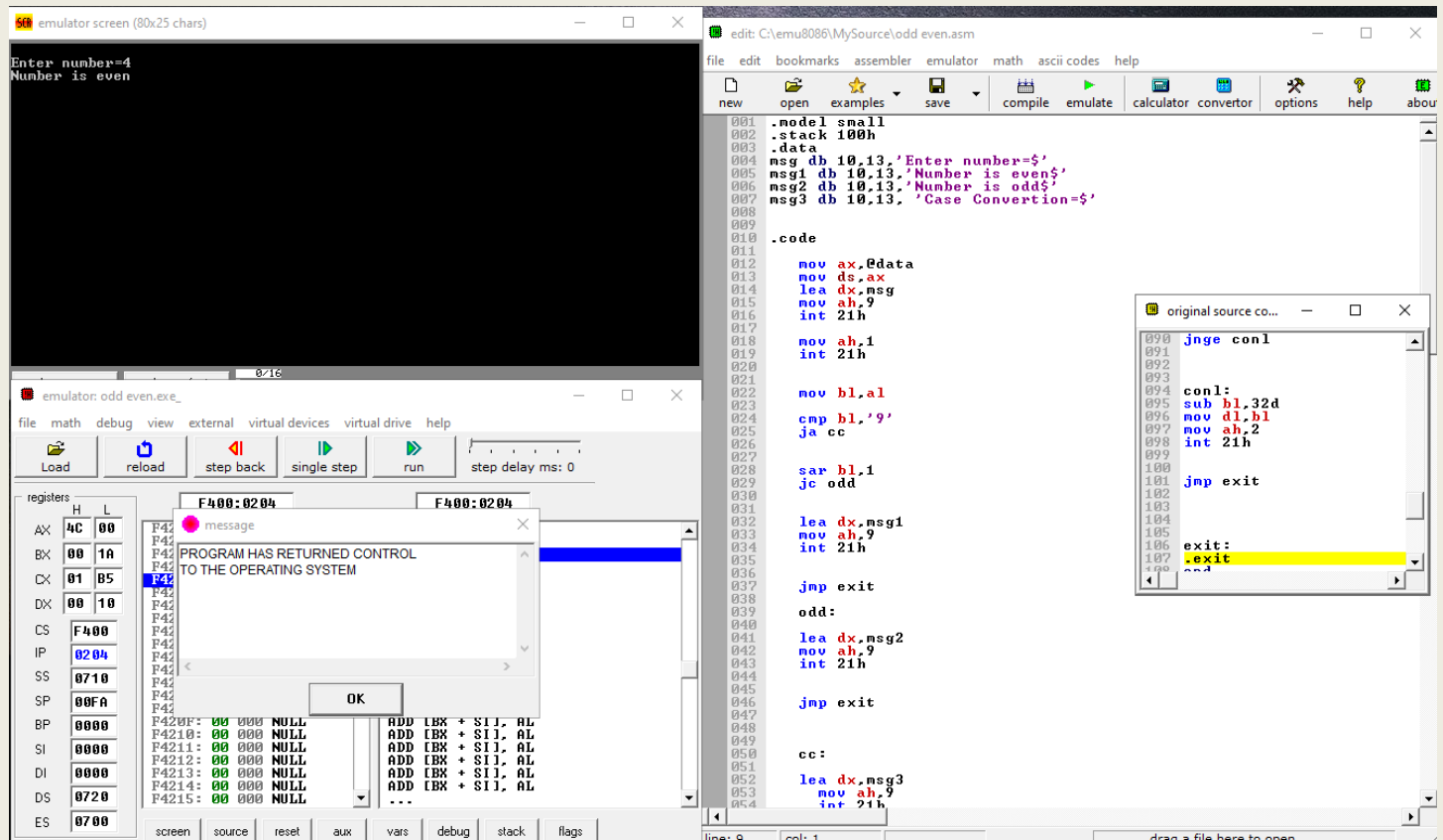
.exit

End

### Description:

First of all we select a standard memory model, set 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 select a standard memory model, set 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:

