

Experiment No. 05

Code:

```
.model small
.stack 100h

.data
    num1 dw 0011h ; First 16-bit number
    num2 dw 0011h ; Second 16-bit number
    result dw 0 ; To store the result

.code
main:
    ; Initialize data segment
    mov ax, @data
    mov ds, ax

    ; Load num1 into AX
    mov ax, num1

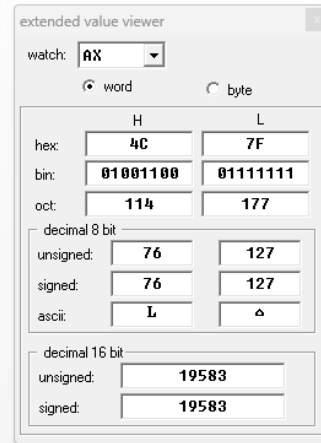
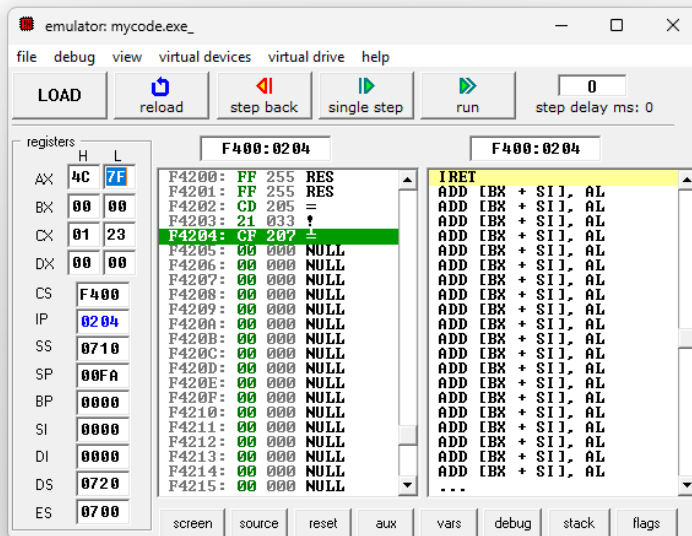
    ; Add num2 to AX
    add ax, num2

    ; Store the result in 'result'
    mov result, ax

    ; Exit the program
    mov ah, 4Ch
    int 21h

end main
```

Output:

 Σ

Experiment No. 06

Output:

The screenshot displays a Windows desktop environment with three overlapping windows:

- Text Editor (Top):** Shows an assembly program named `model snail.asm`. The code includes comments and instructions for rotating a value, displaying it, and terminating the program. The assembly code is as follows:

```
01 .model small
02 .data
03 a db 92H ;Variable a with the hexadecimal value
04 .code
05 mov ax, 0D01H ;Fix: Correct data segment initial:
06 mov ds, ax
07
08 mov al, a
09
10 and al, 0F0H ;Mask *upper nibble**
11
12 rot al, 4 ;Rotate right 4 times
13 mov bh, al ;Store result in BH
14 call disp ;Display upper nibble
15
16 mov al, a
17 and al, 0FH ; Mask *lower nibble**
18
19 mov bh, al ;Store result in BH
20
21 call disp ;Display lower nibble
22
23 mov ah, 4CH ;Terminate program
24 int 21H
25
26 disp proc near
27     mov ch, 02H ;Count of digits
28     mov cl, 04H ; Rotate count
29
30 disp_loop:
31     rol bh, cl ;Rotate BH
32
33     mov dl, bh
34     and dl, 0FH ;Extract LSB
35
36     cmp dl, 09
37
38     jbe below_9
39
40     add dl, 07
41
42     below_9:
43     add dl, 30H ;Convert to ASCII
44
45     mov ah, 02H ;Print character
46
47     int 21H
48
49     dec ch ;Decrement counter
50
51     jnz disp_loop
52
53     ret
54 disp endp
55 end
```
- Emulator Window (Middle):** Titled "emulator: Practical.exe", it shows the execution state. The registers window displays the following values:

Register	Value
AX	4C 32
BX	02 00
CX	00 04
DX	00 32
ES	07 00

The memory window shows the instruction at address `F4207: 0F 20 7` being executed, which corresponds to the `and al, 0F0H` instruction in the source code.
- Debugger Window (Bottom):** Titled "original source co...", it shows the source code with the instruction `and al, 0F0H ;Mask *upper nibble**` highlighted in yellow, indicating the current execution point.

Experiment No. 07

Code:

```
.model small
.stack 100h

.data
    ; Source data
    srcData db 10h, 20h, 30h, 40h, 50h, 60h, 70h, 80h, 90h, 0Ah ; Example source data (10
bytes)

    ; Destination buffer
    dstData db 10h dup(0) ; Allocate space for 10 bytes

.code
main:
    ; Initialize data segment
    mov ax, @data      ; Load address of data segment into AX
    mov ds, ax         ; Set DS to the value in AX

    ; Set up pointers to the source and destination
    lea si, srcData    ; Load address of source data into SI
    lea di, dstData    ; Load address of destination buffer into DI

    ; Set the counter for the number of bytes to transfer
    mov cx, 10         ; We want to transfer 10 bytes

transferLoop:
    ; Move data from source to destination
```

mov al, [si] ; Load byte from source (pointed by SI) into AL

mov [di], al ; Store byte in destination (pointed by DI)

; Increment the pointers

inc si ; Move SI to the next byte in source

inc di ; Move DI to the next byte in destination

; Decrement the counter

loop transferLoop ; Decrement CX and loop if CX != 0

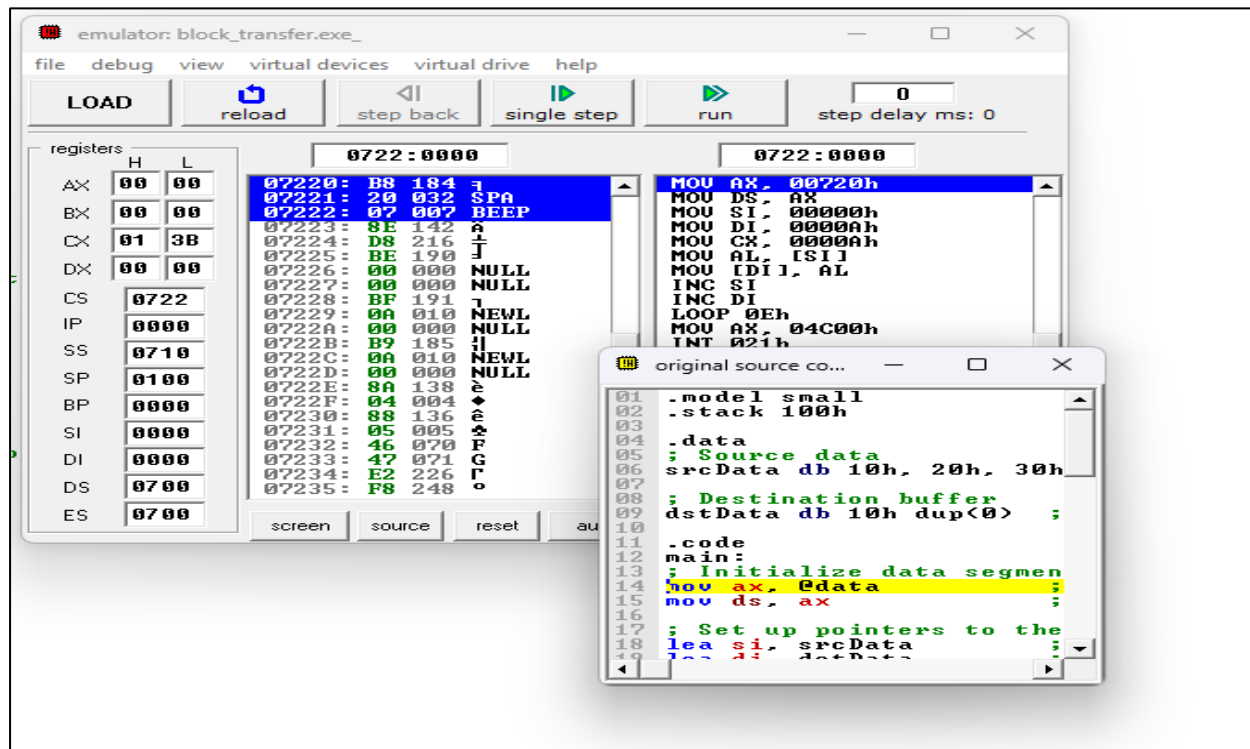
; Exit program (for DOS interrupt 21h)

mov ax, 4C00h ; Prepare to terminate program

int 21h ; Call DOS interrupt 21h to exit

end main

Output:



Experiment No. 08

Code:

```
.model small
```

```
.stack 100h
```

```
.data
```

```
    ; The 8-bit number for which we want to count 1's and 0's
```

```
number db 10101010b ; Example 8-bit number: 10101010 (170 in decimal)
```

```
    ; To store count of 1's and 0's
```

```
count_ones db 0
```

```
count_zeros db 0
```

```
.code
```

```
main:
```

```
    ; Initialize data segment
```

```
    mov ax, @data
```

```
    mov ds, ax
```

```
    ; Load the 8-bit number into AL register
```

```
    mov al, [number]
```

```
    ; Clear the count registers
```

```
    mov bl, 0 ; bl will hold the count of 1's
```

```
    mov bh, 0 ; bh will hold the count of 0's
```

```
count_loop:
```

```
    ; Test the least significant bit of AL
```

```
test al, 1 ; Test the least significant bit of AL (AL AND 1)
```

```
jz zero_bit ; If the bit is 0, jump to zero_bit
```

```
; If the bit is 1, increment the count of ones
```

```
inc bl
```

```
jmp next_bit
```

```
zero_bit:
```

```
; If the bit is 0, increment the count of zeros
```

```
inc bh
```

```
next_bit:
```

```
; Shift AL to the right to check the next bit
```

```
shr al, 1
```

```
; Repeat the loop for all 8 bits
```

```
loop count_loop
```

```
; Store the result in the count variables
```

```
mov [count_ones], bl
```

```
mov [count_zeros], bh
```

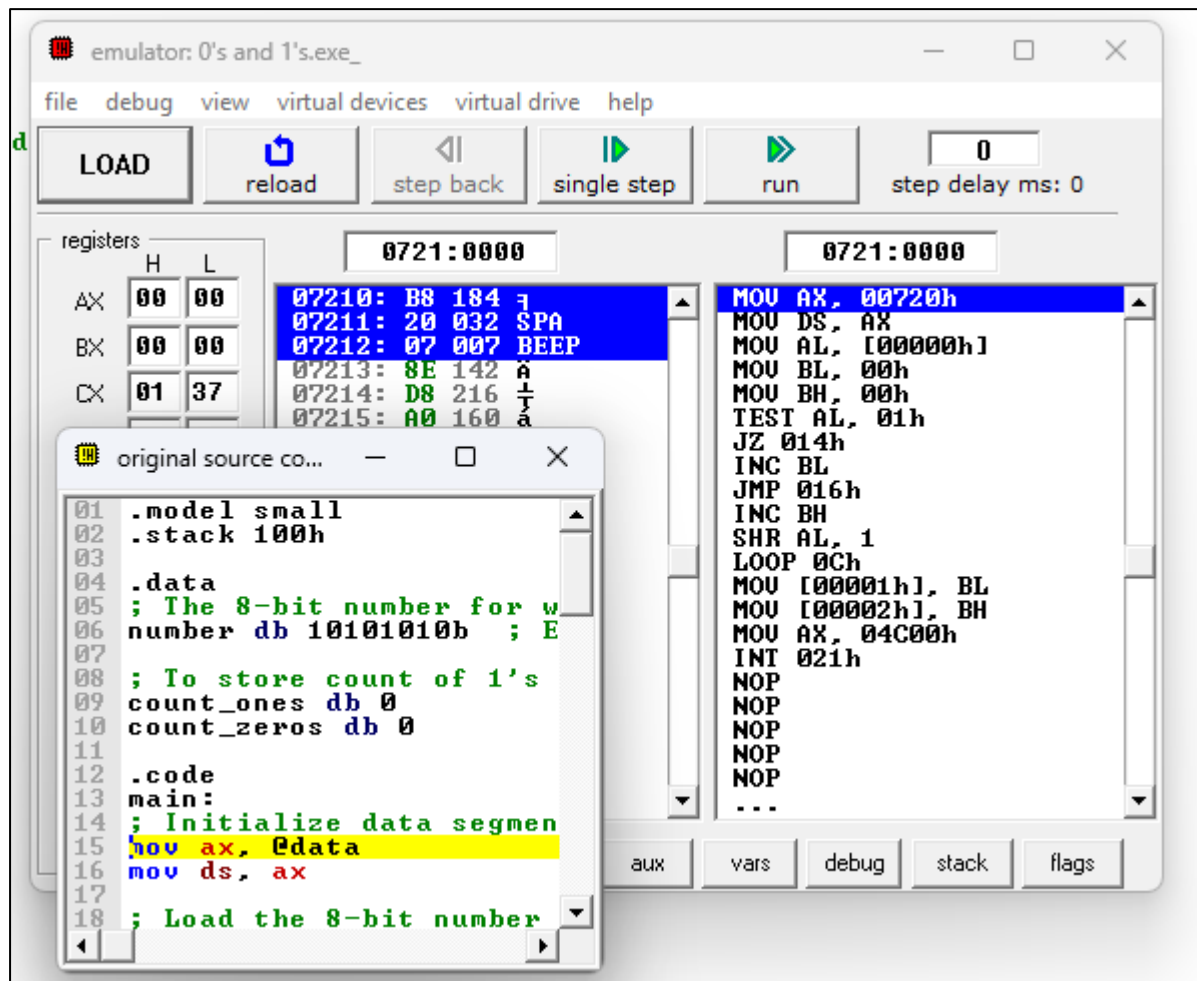
```
; Exit program (DOS interrupt)
```

```
mov ax, 4C00h
```

```
int 21h
```

```
end main
```

Output:



Practical No. 9

AIM: 8086 Assembly Program to Find Smallest Number from Given Numbers.

PROGRAM:

DATA SEGMENT

STRING1 DB 11H, 22H, 33H, 44H, 55H ; Use commas to separate hex values

MSG1 DB "FOUND\$"

MSG2 DB "NOT FOUND\$"

SE DB 33H

DATA ENDS

PRINT MACRO MSG

MOV AH, 09H

LEA DX, MSG

INT 21H

ENDM

CODE SEGMENT

ASSUME CS:CODE DS:DATA

START:

MOV AX, DATA

MOV DS, AX

MOV AL, SE ; Load the value to search for into AL
LEA SI, STRING1 ; Load the address of STRING1 into SI
MOV CX, 05H ; Number of bytes to search in STRING1 (5 bytes)

UP:

MOV BL, [SI] ; Load the byte from STRING1 into BL
CMP AL, BL ; Compare AL with BL
JZ FO ; Jump to FO if they are equal (found)
INC SI ; Move to the next byte in STRING1
DEC CX ; Decrement the counter
JNZ UP ; Jump back to UP if CX is not zero

PRINT MSG2 ; If not found, print "NOT FOUND"
JMP END1 ; Jump to END1 to terminate the program

FO:

PRINT MSG1 ; If found, print "FOUND"

END1:

; Removed INT 3 (no debugging interrupt needed)
RET ; Return to operating system (end program)

CODE ENDS

END START

OUTPUT:

edit: D:\Adi 435\Experiment No. 9.asm

file edit bookmarks assembler emulator math ascii codes help

new open examples save compile emulate calculator convertor options help about

```
01 DATA SEGMENT
02 STRING1 DB 11H, 22H, 33H, 44H, 55H ; Use commas to separate hex
03 MSG1 DB "FOUND$"
04 MSG2 DB "NOT FOUND$"
05 SE DB 33H
06 DATA ENDS
07
08 PRINT MACRO MSG
09     MOV AH, 09H
10     LEA DX, MSG
11     INT 21H
12 ENDM
13
14 CODE SEGMENT
15 ASSUME CS:CODE DS:DATA
16 START:
17     MOV AX, DATA
18     MOV DS, AX
19     MOV AL, SE ; Load the value to search for into AL
20     LEA SI, STRING1 ; Load the address of STRING1 into SI
21     MOV CX, 05H ; Number of bytes to search in STRING1 (<5 bytes>)
22
23 UP:
24     MOV BL, [SI] ; Load the byte from STRING1 into BL
25     CMP AL, BL ; Compare AL with BL
26     JZ FO ; Jump to FO if they are equal <found>
27     INC SI ; Move to the next byte in STRING1
28     DEC CX ; Decrement the counter
29     JNZ UP ; Jump back to UP if CX is not zero
30
31     PRINT MSG2 ; If not found, print "NOT FOUND"
32     JMP END1 ; Jump to END1 to terminate the program
33
34 FO:
35     PRINT MSG1 ; If found, print "FOUND"
36
37 END1:
38     ; Removed INT 3 <no debugging interrupt needed>
39     RET ; Return to operating system <end program>
40 CODE ENDS
41 END START
```

original source code

```
16 START:
17     MOV AX, DATA
18     MOV DS, AX
19     MOV AL, SE ; Load the v
20     LEA SI, STRING1 ; Load the a
21     MOV CX, 05H ; Number of
22
23 UP:
24     MOV BL, [SI] ; Load the b
25     CMP AL, BL ; Compare AL
26     JZ FO ; Jump to FO
27     INC SI ; Move to th
28     DEC CX ; Decrement
29     JNZ UP ; Jump back
30
31     PRINT MSG2 ; If not fou
32     JMP END1 ; Jump to EN
33
34 FO:
35     PRINT MSG1 ; If found,
36
37 END1:
38     ; Removed INT 3 <no debugging
39     RET ; Return to
```

emulator: Experiment No. 9.exe

file math debug view external virtual devices virtual drive help

Load reload step back single step run step delay ms: 0

registers	H	L	0712:2D45	0712:2D45
AX	09	24	09E5D: 00 000 NULL	ADD [BX + SI], AL
BX	00	33	09E5E: 00 000 NULL	ADD [BX + SI], AL
CX	00	03	09E5F: 00 000 NULL	ADD [BX + SI], AL
DX	00	05	09E60: 00 000 NULL	ADD [BX + SI], AL
CS	0712		09E61: 00 000 NULL	ADD [BX + SI], AL
IP	2D45		09E62: 00 000 NULL	ADD [BX + SI], AL
SS	0710		09E63: 00 000 NULL	ADD [BX + SI], AL
SP	0002		09E64: 00 000 NULL	ADD [BX + SI], AL
BP	0000		09E65: 00 000 NULL	ADD [BX + SI], AL
SI	0002		09E66: 00 000 NULL	ADD [BX + SI], AL
DI	0000		09E67: 00 000 NULL	ADD [BX + SI], AL
DS	0710		09E68: 00 000 NULL	ADD [BX + SI], AL
ES	0700		09E69: 00 000 NULL	ADD [BX + SI], AL
			09E6A: 00 000 NULL	ADD [BX + SI], AL
			09E6B: 00 000 NULL	ADD [BX + SI], AL
			09E6C: 00 000 NULL	ADD [BX + SI], AL
			09E6D: 00 000 NULL	ADD [BX + SI], AL
			09E6E: 00 000 NULL	ADD [BX + SI], AL
			09E6F: 00 000 NULL	ADD [BX + SI], AL
			09E70: 00 000 NULL	ADD [BX + SI], AL
			09E71: 00 000 NULL	...

emulator screen (80x9 chars)

FOUND

clear screen change font 8/16

line: 1 col: 29 drag a file here to open

CONCLUSION:

In these assembly language programs, we successfully implemented algorithms to determine the largest and smallest numbers from a given series of hexadecimal values stored in the data segment.

Finding the Largest Number:

The program initializes a data series and compares each element sequentially. It stores the largest value found and displays it as output.

Finding the Smallest Number:

A similar approach is used to determine the smallest number. The program iterates through the data series, updating the smallest value found.

Practical No. 10

AIM: Compute the factorial of a positive integer 'n' using recursive procedure.

PROGRAM:

DATA SEGMENT

 number dw 04h ; Store the number for factorial (4 in this case)

DATA ENDS

STACK SEGMENT

 dw 128 dup(0) ; Stack size 128 bytes

STACK ENDS

CODE SEGMENT

 ASSUME CS:CODE, DS:DATA

START:

 ; Set up the segment registers

 mov ax, DATA

 mov ds, ax

 mov cx, [number] ; Load the value of 'number' into CX (input number)

 ; Call the factorial procedure

 call fact

; After factorial calculation, move result from DX to AX to return control

mov ax, dx ; We want the result in AX

; Exit to the operating system

mov ax, 4C00h ; DOS interrupt to exit the program

int 21h

fact proc near

cmp cx, 01h ; Base case: if CX == 1

jne cont ; If CX != 1, continue with calculation

mov dx, 01h ; Factorial of 1 is 1

ret

cont:

push cx ; Backup CX

dec cx ; Decrement CX (n - 1)

call fact ; Recursively call fact to calculate (n - 1)!

pop cx ; Restore CX

mul dx ; Multiply AX by DX ($n * (n-1)!$)

mov dx, ax ; Store result in DX (since AX holds the result of mul)

ret

fact endp

CODE ENDS

END START

OUTPUT:

edit: D:\Adi 435\Experiment No. 10.asm

file edit bookmarks assembler emulator math ascii codes help

new open examples save compile emulate calculator convertor options help about

```
01 DATA SEGMENT
02     number dw 04h ; Store the number for factorial (4 in this case)
03 DATA ENDS
04
05 STACK SEGMENT
06     dw 128 dup(0) ; Stack size 128 bytes
07 STACK ENDS
08
09 CODE SEGMENT
10     ASSUME CS:CODE, DS:DATA
11 START:
12     ; Set up the segment registers
13     mov ax, DATA
14     mov ds, ax
15     mov cx, [number] ; Load the value of 'number' into CX (input num
16
17     ; Call the factorial procedure
18     call fact
19
20     ; After factorial calculation, move result from DX to AX to retur
21     mov ax, dx ; We want the result in AX
22
23     ; Exit to the operating system
24     mov ax, 4C00h ; DOS interrupt to exit the program
25     int 21h
26
27 ; Factorial procedure
28 fact proc near
29     cmp cx, 01h ; Base case: if CX == 1
30     jne cont ; If CX != 1, continue with calculation
31
32     mov dx, 01h ; Factorial of 1 is 1
33     ret
34
35 cont:
36     push cx ; Backup CX
37     dec cx ; Decrement CX (n - 1)
38     call fact ; Recursively call fact to calculate (n - 1)!
39
40     pop cx ; Restore CX
41     mul dx ; Multiply AX by DX (n * (n-1)!)
42     mov dx, ax ; Store result in DX (since AX holds the result of mul)
43     ret
44
45 fact endp
46
47 CODE ENDS
48 END START
49
```

original source code

```
20 ; After factorial calculation
21 mov ax, dx ; We want the res
22
23 ; Exit to the operating syste
24 mov ax, 4C00h ; DOS interrup
25 int 21h
26
27 ; Factorial procedure
28 fact proc near
29     cmp cx, 01h ; Base case:
30     jne cont ; If CX != 1
31
32     mov dx, 01h ; Factorial
33     ret
34
35 cont:
36     push cx ; Backup CX
37     dec cx ; Decrement
38     call fact ; Recursive1
39
40     pop cx ; Restore CX
41     mul dx ; Multiply A
42     mov dx, ax ; Store resu
43     ret
```

emulator: Experiment No. 10.exe

file math debug view external virtual devices virtual drive help

Load reload step back single step run step delay ms: 0

registers	H	L
AX	4C	00
BX	00	00
CX	00	04
DX	00	00
CS	F400	
IP	0204	
SS	0711	
SP	00FA	
BP	0000	
SI	0000	
DI	0000	
DS	0710	
ES	0700	

F400:0204		F400:0204	
F4200:	FF 255 RES	BIOS DI	
F4201:	FF 255 RES	INT 021h	
F4202:	CD 205 =	IRET	
F4203:	21 033 !	ADD [BX + SI], AL	
F4204:	CF 207 ±	ADD [BX + SI], AL	
F4205:	00 000 NULL	ADD [BX + SI], AL	
F4206:	00 000 NULL	ADD [BX + SI], AL	
F4207:	00 000 NULL	ADD [BX + SI], AL	
F4208:	00 000 NULL	ADD [BX + SI], AL	
F4209:	00 000 NULL	ADD [BX + SI], AL	
F420A:	00 000 NULL	ADD [BX + SI], AL	
F420B:	00 000 NULL	ADD [BX + SI], AL	
F420C:	00 000 NULL	ADD [BX + SI], AL	
F420D:	00 000 NULL	ADD [BX + SI], AL	
F420E:	00 000 NULL	ADD [BX + SI], AL	
F420F:	00 000 NULL	ADD [BX + SI], AL	
F4210:	00 000 NULL	ADD [BX + SI], AL	
F4211:	00 000 NULL	ADD [BX + SI], AL	
F4212:	00 000 NULL	ADD [BX + SI], AL	
F4213:	00 000 NULL	ADD [BX + SI], AL	
F4214:	00 000 NULL	...	

screen source reset aux vars debug stack flags

message

PROGRAM HAS RETURNED CONTROL TO THE OPERATING SYSTEM

OK

line: 49 col: 1 drag a file here to open

Conclusion:

In this experiment, we successfully implemented a recursive procedure to compute the factorial of a positive integer using assembly language.

1. Recursive Function Implementation:

The factorial function (fact) calls itself until $n = 1$, at which point it returns 1. The result of $n * (n-1)!$ is computed using stack operations (push and pop).

2. Stack Utilization:

The program utilizes the stack to store intermediate values, ensuring that the function operates correctly through recursive calls.

3. Factorial Computation:

The final result is stored in the DX register, showcasing correct multiplication and recursive execution.

This experiment demonstrates the application of recursion in assembly language and provides insight into how function calls work.

Experiment No: 11

Code:

```
.model small
```

```
.stack 100h
```

```
.data
```

```
prompt db "Enter the value of n: $"
```

```
newline db 0Ah, 0Dh, "$"
```

```
result db "Fibonacci Numbers are: $"
```

```
n db 0 ; Variable to store the number of Fibonacci numbers to generate
```

```
fib1 db 0 ; First Fibonacci number (0)
```

```
fib2 db 1 ; Second Fibonacci number (1)
```

```
fib3 db 0 ; To store the next Fibonacci number
```

```
buffer db 3, 0 ; Buffer to store a number for input
```

```
.code
```

```
main:
```

```
mov ax, @data
```

```
mov ds, ax ; Initialize the data segment
```

```
; Display prompt message
```

```
lea dx, prompt
```

```
mov ah, 09h
```

```
int 21h
```

```
; Read input for 'n' (number of Fibonacci numbers)
```

```
lea dx, buffer
```

```

mov ah, 0Ah
int 21h
lea si, buffer + 1 ; SI points to the first digit of the input

; Convert ASCII to integer
mov al, [si]
sub al, '0'
mov bl, al ; Store the value of 'n' in BL (number of Fibonacci numbers to
generate)

; Display "Fibonacci Numbers are: "
lea dx, result
mov ah, 09h
int 21h

; Display the first Fibonacci number (fib1 = 0)
mov al, fib1
call PrintNumber

; Display the second Fibonacci number (fib2 = 1)
mov al, fib2
call PrintNumber

; Loop to generate the next Fibonacci numbers, starting from the 3rd
Fibonacci number
mov cl, bl ; Store the count of Fibonacci numbers to print in CL
sub cl, 2 ; Since the first two numbers are already printed, we start
with n-2

```

fibonacci_loop:

; Calculate the next Fibonacci number

mov al, fib1

add al, fib2 ; fib3 = fib1 + fib2

mov fib3, al

; Display the next Fibonacci number

mov al, fib3

call PrintNumber

; Update fib1 and fib2 for next iteration

mov al, fib2

mov fib1, al ; Store fib2 in fib1

mov al, fib3

mov fib2, al ; Store fib3 in fib2

dec cl ; Decrement the count

jnz fibonacci_loop ; If cl > 0, repeat the loop

; Display a new line

lea dx, newline

mov ah, 09h

int 21h

; Exit program

mov ah, 4Ch

int 21h

;-----

PrintNumber proc

; Input: AL = number to print

; Output: prints the number in AL as ASCII

add al, '0' ; Convert number to ASCII

mov dl, al

mov ah, 02h

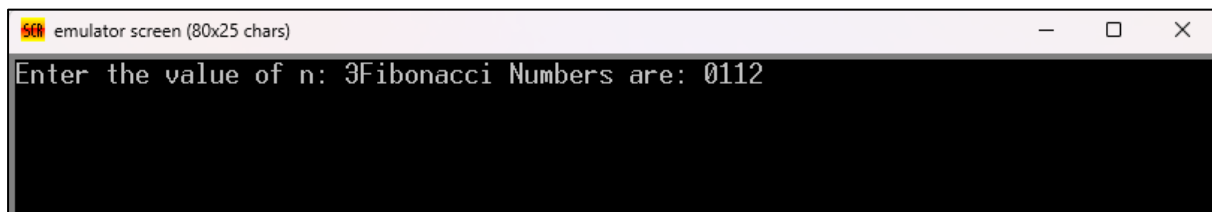
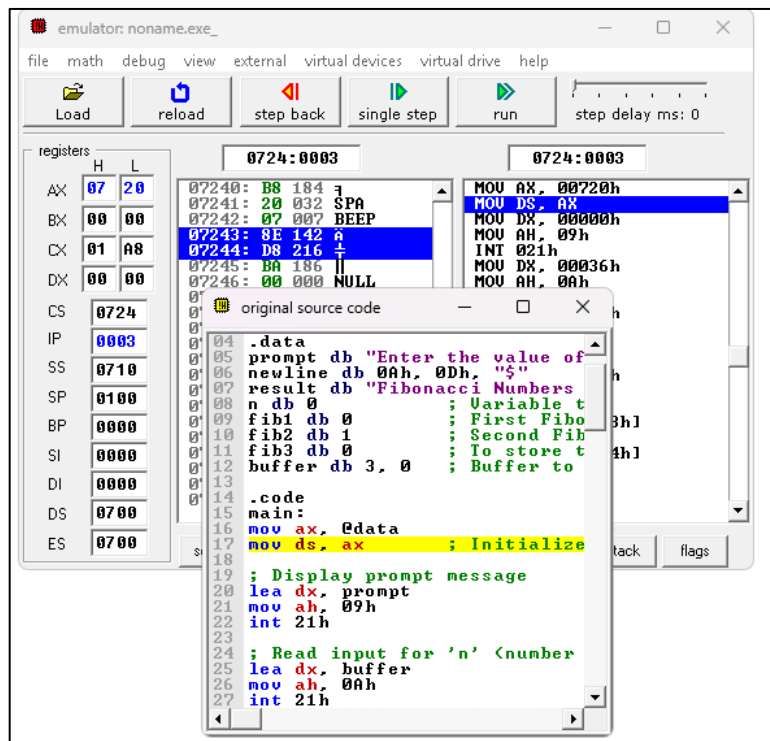
int 21h

ret

PrintNumber endp

end main

Output:



Fibonacci

.model small

.stack 100h

.data

prompt db "Enter the value of n: \$"

newline db 0Dh, 0Ah, "\$"

result db "Fibonacci Numbers are: \$"

fib1 db 0

fib2 db 1

fib3 db 0

buffer db 2, ?, ?, '\$' ; 2 bytes for input: max 1 digit + CR

.code

main:

mov ax, @data

mov ds, ax

; Display prompt

lea dx, prompt

mov ah, 09h

int 21h

; Read user input

lea dx, buffer

mov ah, 0Ah

int 21h

; Convert ASCII input to integer

lea si, buffer + 2

mov al, [si]

sub al, '0'

mov bl, al

; Display header

lea dx, newline

mov ah, 09h

int 21h

lea dx, result

mov ah, 09h

int 21h

; Print first number (0)

mov al, fib1

call PrintNumber

; Check if n == 1, then exit

cmp bl, 1

jbe exit_program

; Print second number (1)

```

    mov al, fib2
    call PrintNumber

; Loop for remaining Fibonacci numbers
    mov cl, bl
    sub cl, 2

fibonacci_loop:
    ; fib3 = fib1 + fib2
    mov al, fib1
    add al, fib2
    mov fib3, al

    ; Print fib3
    mov al, fib3
    call PrintNumber

    ; Update fib1 = fib2, fib2 = fib3
    mov al, fib2
    mov fib1, al
    mov al, fib3
    mov fib2, al

    dec cl
    jnz fibonacci_loop

exit_program:
    ; New line
    lea dx, newline
    mov ah, 09h
    int 21h

    ; Exit
    mov ah, 4Ch
    int 21h

;-----
PrintNumber proc
    ; Input: AL = number (0-9)
    ; Output: displays number followed by space
    add al, '0'
    mov dl, al
    mov ah, 02h
    int 21h

    ; Print space
    mov dl, ' '
    mov ah, 02h
    int 21h
    ret
PrintNumber endp
end main

```

$$S = A \oplus B \oplus C-IN$$

Thus, the sum output is the XOR of **A**, **B**, and **C-IN**.

Logical Expression for C-OUT

From the truth table, the logical expression for **C-OUT** (carry-out) in a full adder is:

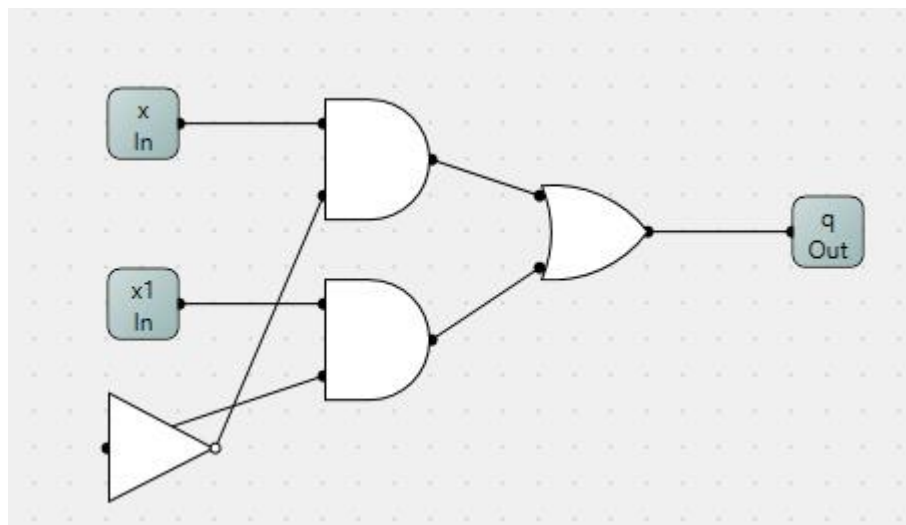
$$C-OUT = A' B C-IN + A B' C-IN + A B C-IN' + A B C-IN$$

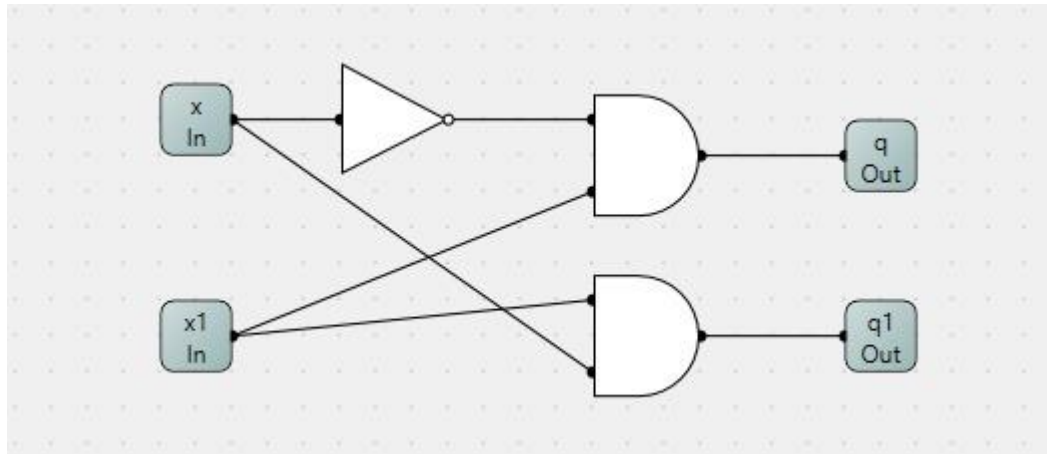
This simplifies to:

$$C-OUT = A B(C-IN' + C-IN) + C-IN(A'B + AB')$$

Since $C-IN' + C-IN = 1$ and $A'B + AB' = A \oplus B$. Thus, the final simplified expression is:

$$C-OUT = A B + C-IN (A \oplus B)$$





Practical 6

```
.model small
.data
a db 92H; Variable a with the hexadecimal v
.code
mov ax, @DATA; Fix: Correct data segment ini
mov ds, ax
mov al, a
and al, 0F0H; Mask *upper nibble*
ror al, 4; Rotate right 4 times mov bh, al; Store result in BH call disp; Display upper nibble
mov bh, al ;Store result in BH
call disp; Display lower nibble
mov al, a
and al, 0FH; Mask *lower nibble*
    mov bh,al
call disp
mov ah, 4CH ; Terminate program
int 21H
disp proc near
mov ch, 02H; Count of digits p??
mov cl, 04H; Rotate count

disp_loop:

rol bh, cl; Rotate BH

mov dl, bh

and dl, 0FH; Extract LSB

cmp dl, 09

jbe below_9

add dl, 07

below_9:

add dl, 30H; Convert to ASCII

mov ah, 02H; Print character

int 21H

dec ch; Decrement counter

jnz disp_loop
ret

disp endp

end
```