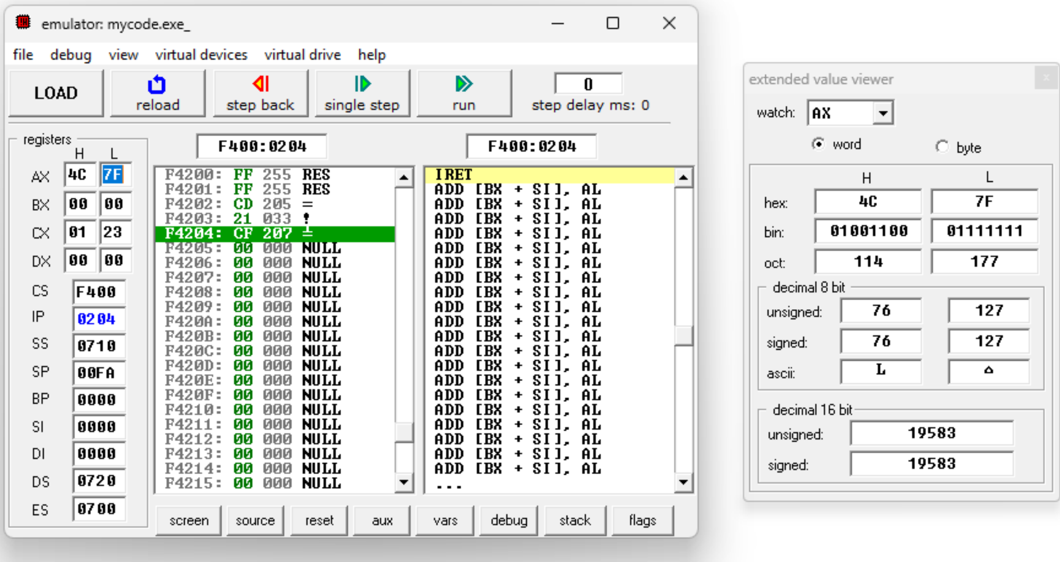
Experiment No. 05

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:

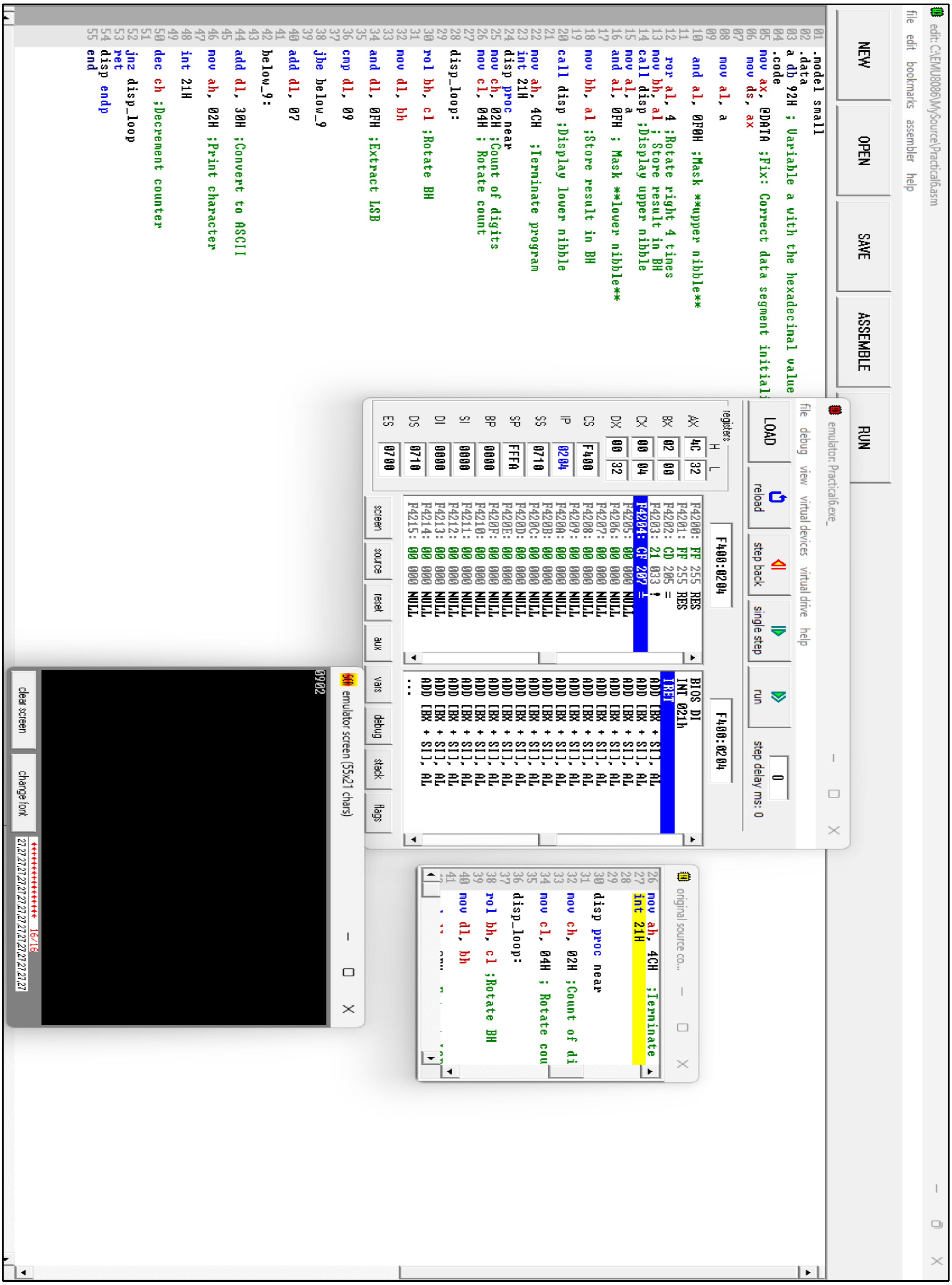


emulator: mycode.exe_
file
debug
view
virtual devices
virtual drive
help
LOAD
reload
step back
single step
run
0
step delay ms: 0
registers
F400:0204
F400:0204
AX 4C 7F
BX 00 00
CX 01 23
DX 00 00
CS F400
IP 0204
SS 0710
SP 00FA
BP 0000
SI 0000
DI 0000
DS 0720
ES 0700
F4200: FF 255 RES
F4201: FF 255 RES
F4202: CD 205 =
F4203: 21 033 !
F4204: CF 207 ±
F4205: 00 000 NULL
IRET
ADD [BX + SI], AL
screen
source
reset
aux
vars
debug
stack
flags
extended value viewer
watch: AX
word
byte
H
L
hex: 4C 7F
bin: 01001100 01111111
oct: 114 177
decimal 8 bit
unsigned: 76 127
signed: 76 127
ascii: L ⌂
decimal 16 bit
unsigned: 19583
signed: 19583

z

Experiment No. 06

Output:



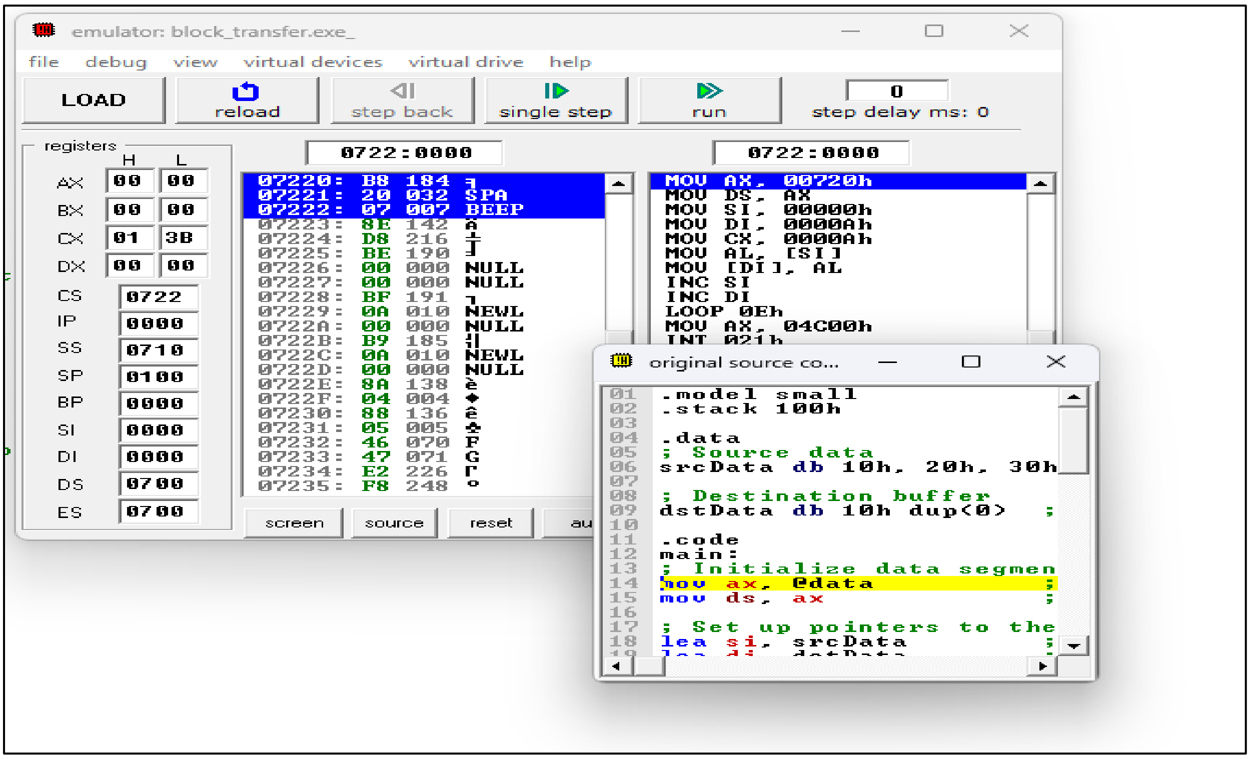
Experiment No. 07

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:



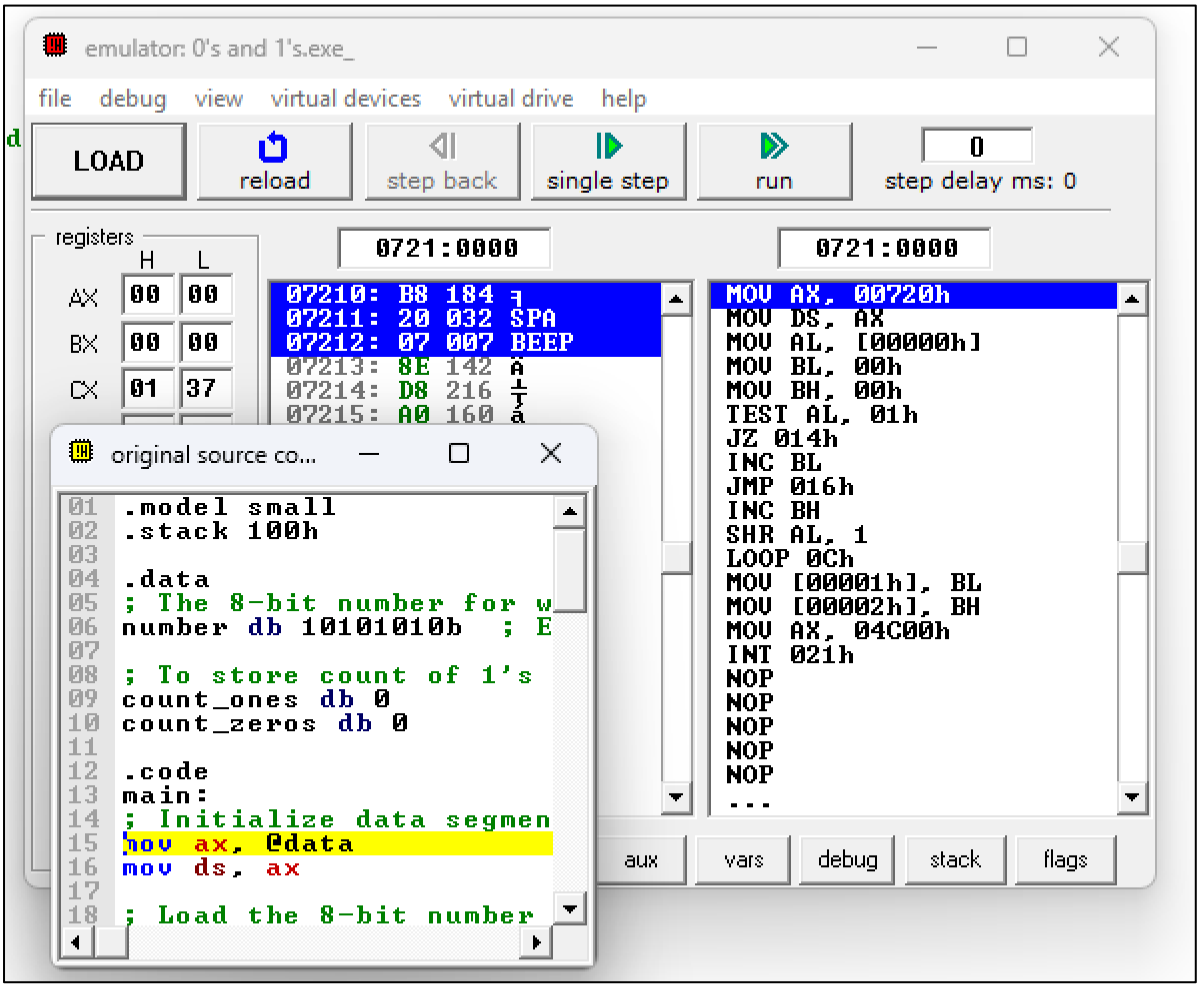
Experiment No. 08

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:



emulator: 0's and 1's.exe_
file debug view virtual devices virtual drive help
LOAD
reload
step back
single step
run
0
step delay ms: 0
registers
H L
AX 00 00
BX 00 00
CX 01 37
0721:0000
0721:0000
07210: B8 184
07211: 20 032 SPA
07212: 07 007 BEEP
07213: 8E 142
07214: D8 216
07215: A0 160
MOV AX, 00720h
MOV DS, AX
MOV AL, [00000h]
MOV BL, 00h
MOV BH, 00h
TEST AL, 01h
JZ 014h
INC BL
JMP 016h
INC BH
SHR AL, 1
LOOP 0Ch
MOV [00001h], BL
MOV [00002h], BH
MOV AX, 04C00h
INT 021h
NOP
NOP
NOP
NOP
NOP
...
original source co...
01 .model small
02 .stack 100h
03
04 .data
05 ; The 8-bit number for w
06 number db 10101010b ; E
07
08 ; To store count of 1's
09 count_ones db 0
10 count_zeros db 0
11
12 .code
13 main:
14 ; Initialize data segmen
15 mov ax, @data
16 mov ds, ax
17
18 ; Load the 8-bit number
aux
vars
debug
stack
flags

Practical No. 9

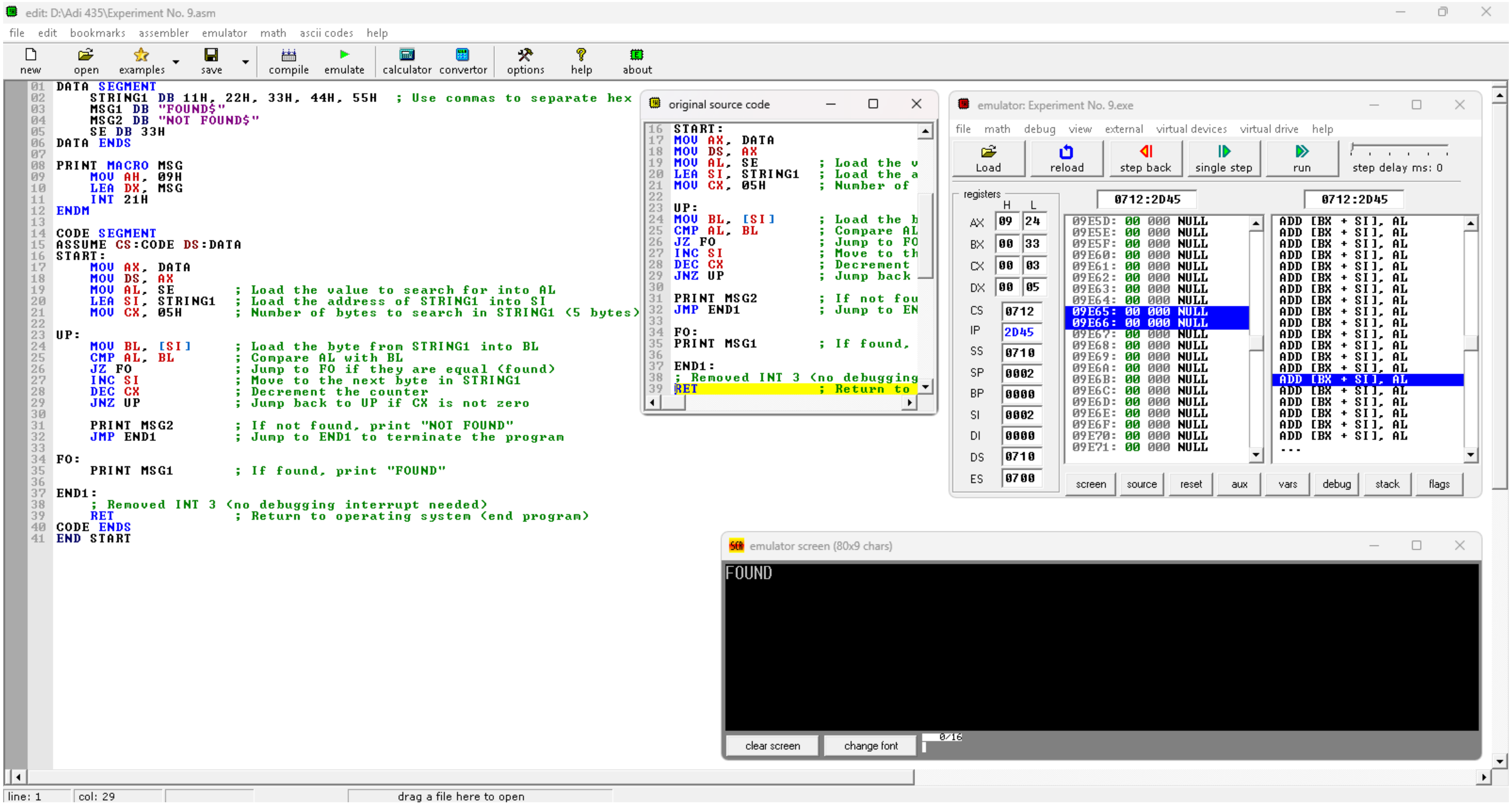
AIM: 8086 Assembly Program to Find Smallest Number from Given Numbers.

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
DATA SEGMENT
STRING1 DB 11H, 22H, 33H, 44H, 55H ; Use commas to separate hex
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
emulator: Experiment No. 9.exe
emulator screen (80x9 chars)
FOUND

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

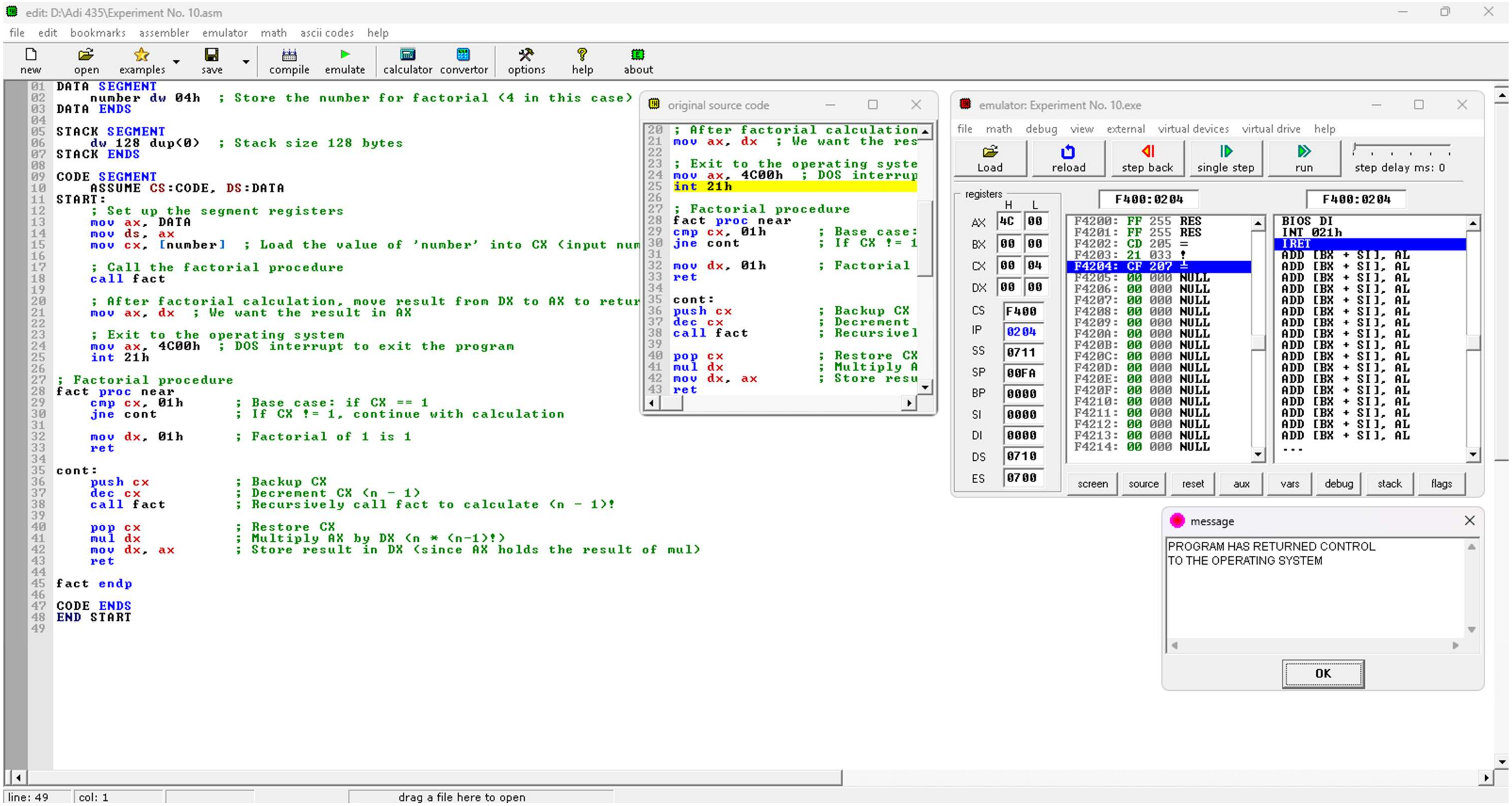
AIM: Compute the factorial of a positive integer 'n' using recursive procedure.

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
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 num
; Call the factorial procedure
call fact
; After factorial calculation, move result from DX to AX to retur
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
; Factorial procedure
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
original source code
emulator: Experiment No. 10.exe
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
BIOS DI
INT 021h
IRET
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 works.

Experiment No: 11

Code:

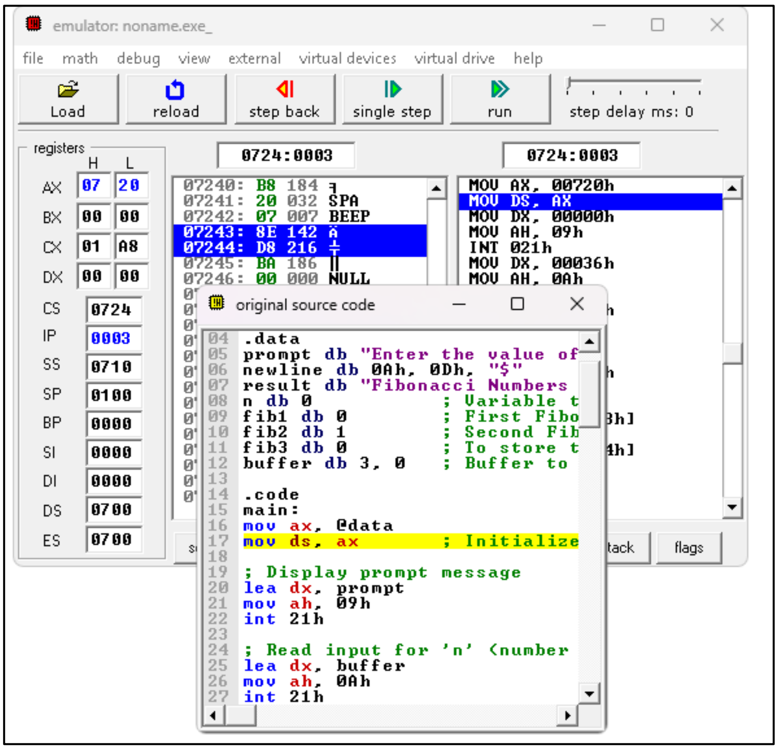
```
.model small
.stack 100h

.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
```

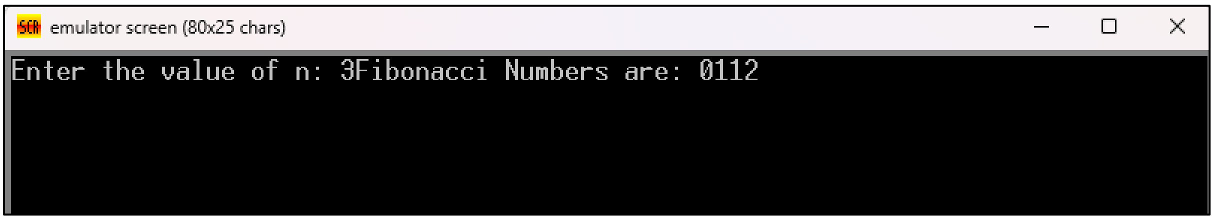
```
    mov ah, 0Ah
    int 21h
    lea si, buffer + 1   ; SI points to the first digit of the input

    ; 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
```

$S = A \oplus B \oplus C\text{-}IN$

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:

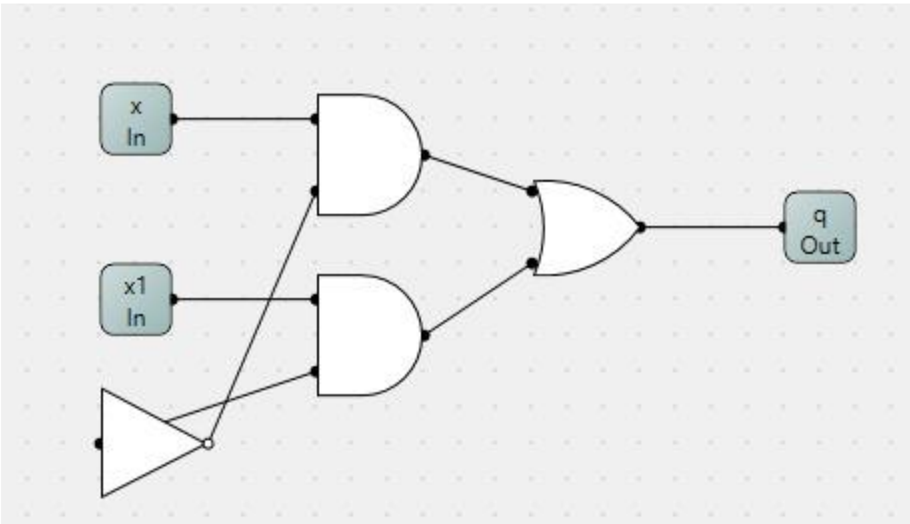
$$C\text{-}OUT = A'\,B\,C\text{-}IN + A\,B'\,C\text{-}IN + A\,B\,C\text{-}IN' + A\,B\,C\text{-}IN$$

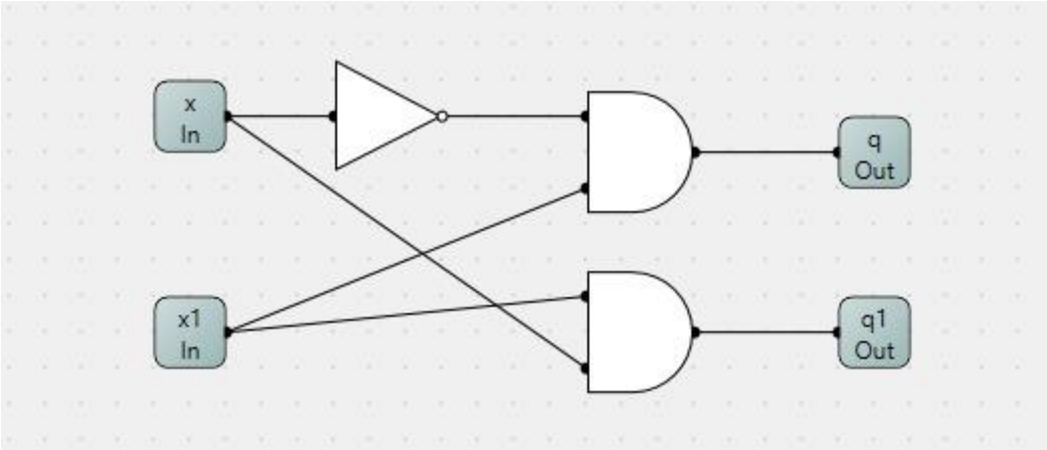
This simplifies to:

$$C\text{-}OUT = A\,B(C\text{-}IN' + C\text{-}IN) + C\text{-}IN(A'B + AB')$$

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

$$C\text{-}OUT = A\,B + C\text{-}IN\,(A \oplus B)$$




x
In
x1
In
q
Out
q1
Out

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
```