# Riphah School of Computing and Innovation (RSCI), Lahore



Computer Organization and Assembly Language (Lab)

## Lab Report # 2

**Submitted to:** Ms. Amna Bibi

**Submitted by:**

{Muhammad Abdullah}

{53457}

{BSCS 3B}

{09/11/2024}

## Task 1:

## Print a string such as Hello World using “Offset” keyword.

## Code:

ORG 100H

.DATA

msg DB "Hello World$"

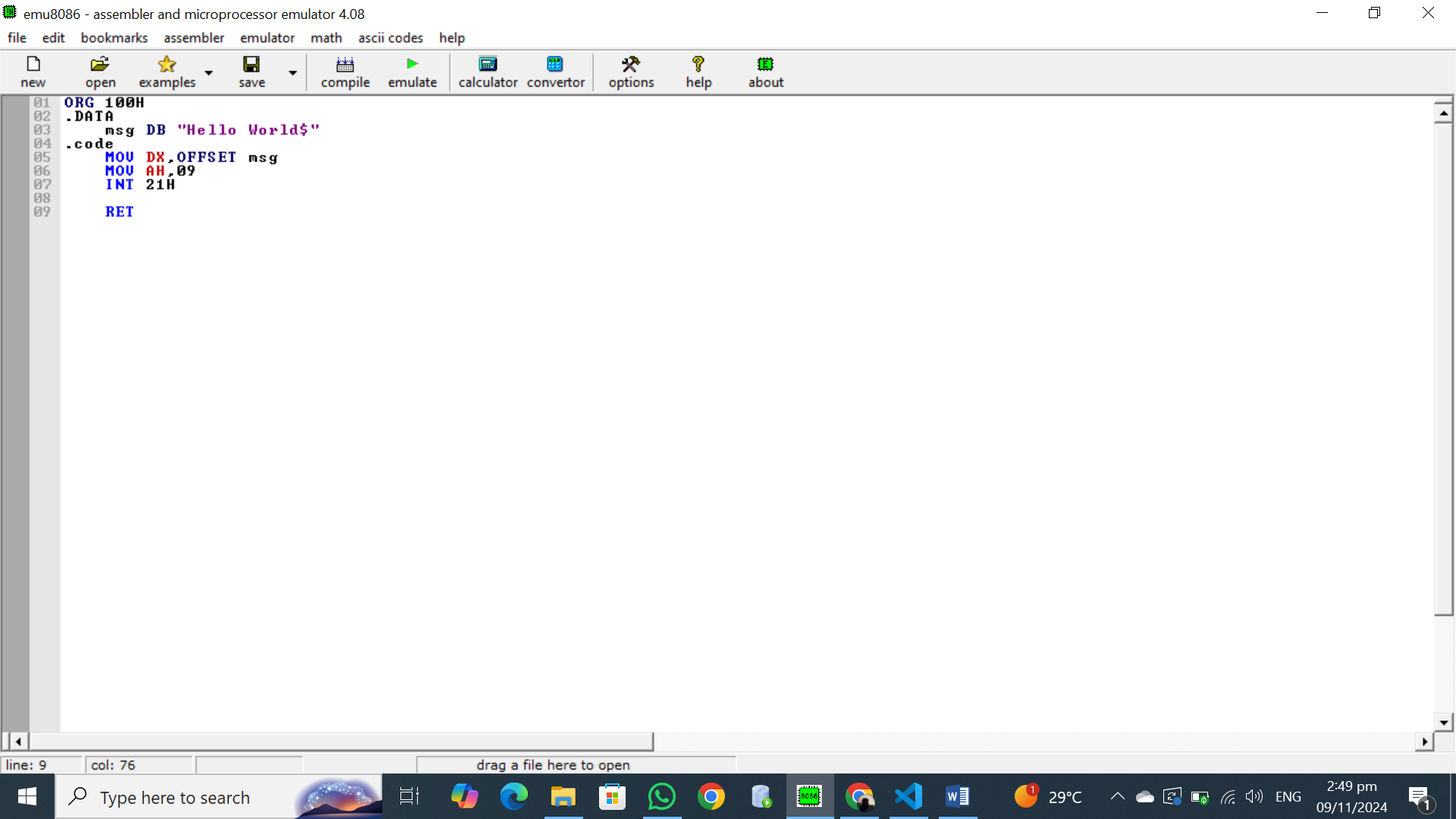
.code

MOV DX,OFFSET msg

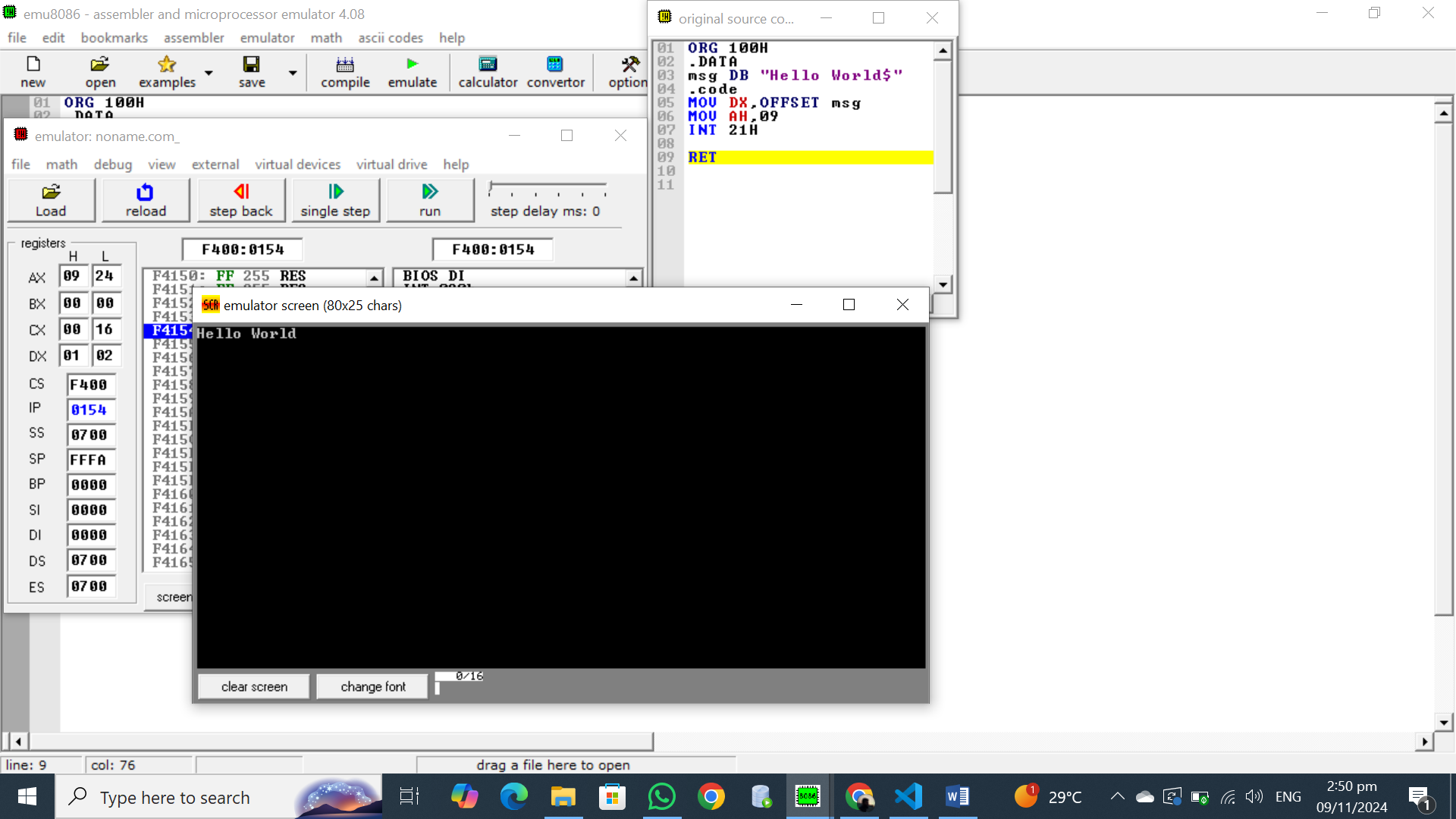
MOV AH,09

INT 21H

RET

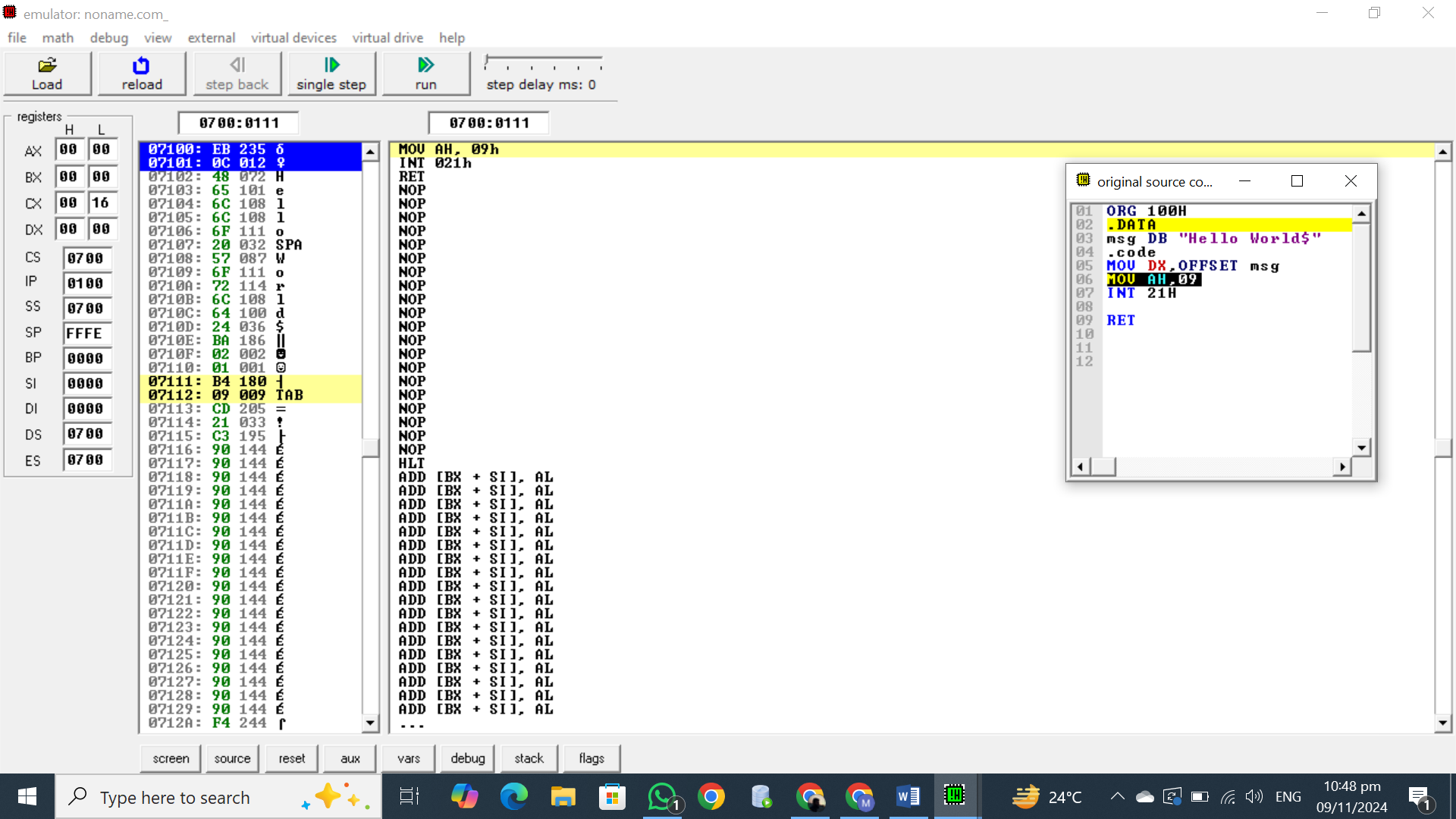


## OUTPUT



## Explanation:

**Binary Screen**



**Registers:**

* **AX:** Holds the value 00 00. The lower byte (AL) is 00, and the higher byte (AH) is 00.
* **BX:** Holds the value 00 00.
* **DX:** Holds the value 00 16.
* **CX:** Holds the value 07101.
* **CS:** Holds the value 0700. This is the Code Segment register, pointing to the current code segment.
* **IP:** Holds the value 0100. This is the Instruction Pointer, pointing to the next instruction to be executed.

**Memory Address:**

* The current memory address is 0700:0100. This is calculated by combining the contents of the CS and IP registers: (CS \* 16) + IP = (0700 \* 16) + 0100 = 07100.

**Current Instruction:**

* The current instruction is located at memory address 07100. This is the instruction MOV AH, 09h.

**Instruction Breakdown:**

* MOV AH, 09h: This instruction moves the value 09h (hexadecimal 9) into the AH register.

**Binary Representation:**

* The binary representation of the values in the registers is as follows:
  + AX: 00000000 00000000
  + BX: 00000000 00000000
  + DX: 00000000 00010110
  + CX: 00000111 00010001
  + CS: 00000111 00000000
  + IP: 00000000 00010000

**Other Instructions:**

* The NOP instructions do nothing. They are often used for padding or debugging purposes.
* The INT 21h instruction is a software interrupt that is used to call the operating system's services.
* The RET instruction returns from a procedure.

The registers store various values, and the CS:IP register pair points to the current instruction. The instruction at 07100 is currently being executed, which moves the value 09h into the AH register.

## Task 2:

## Print a string such as Hello World using LEA instruction.

## Code:

.MODEL SMALL

.STACK 100

.data

msg dw "Helo World$"

.code

MOV AX,@data

MOV DS,AX

lea DX,msg

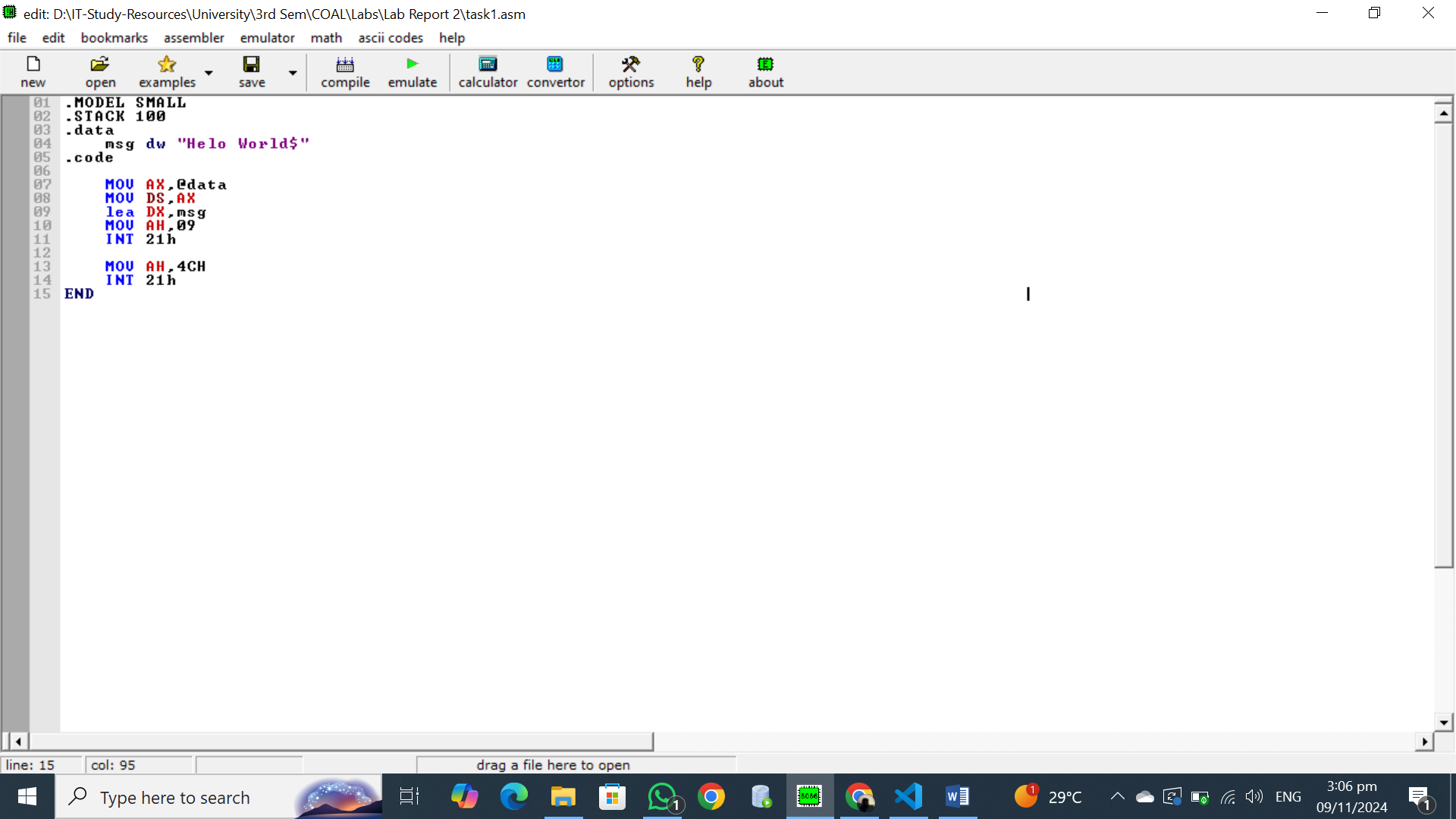
MOV AH,09

INT 21h

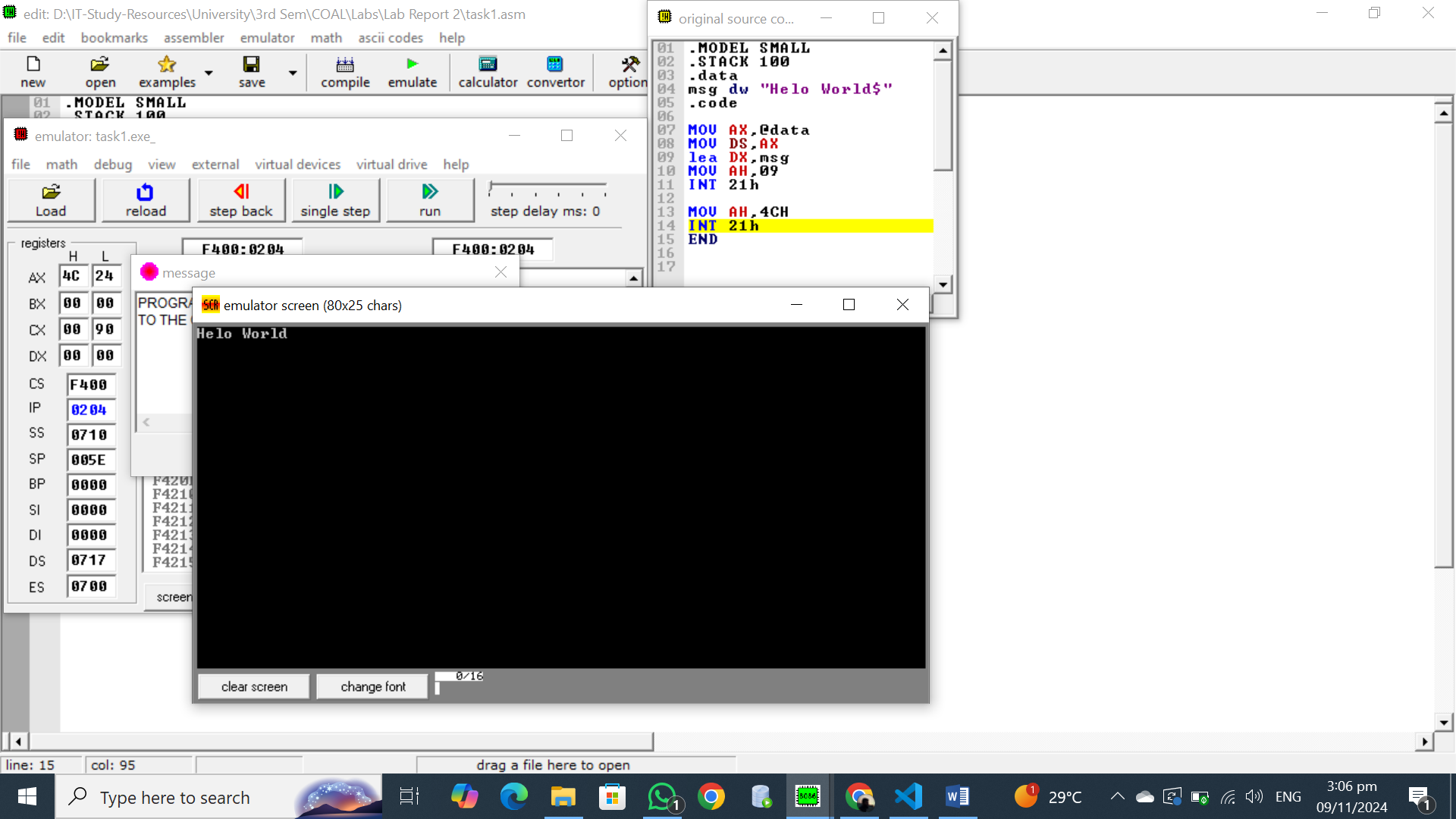
MOV AH,4CH

INT 21h

END

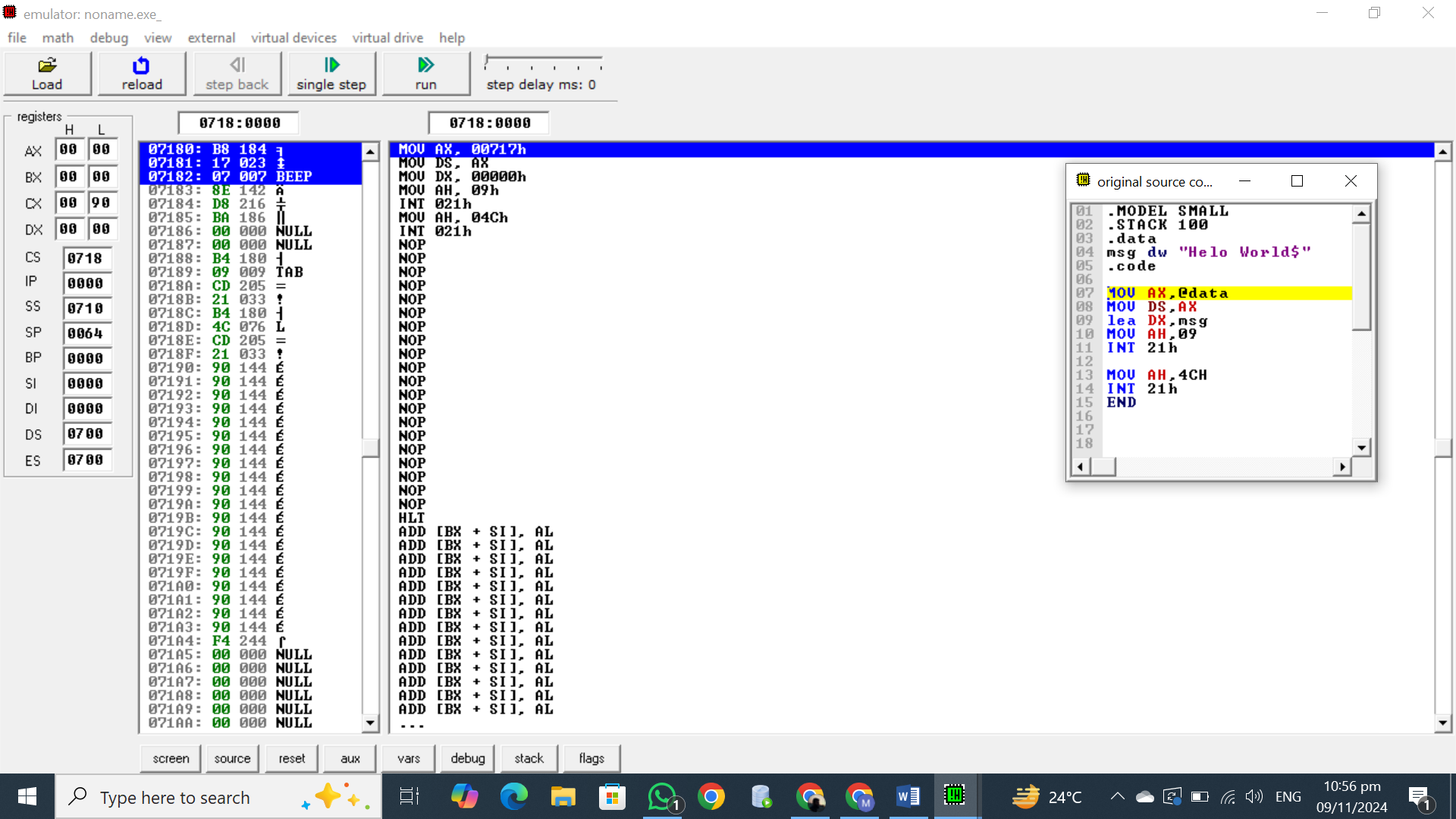


## OUTPUT



## Explanation:

**Binary Screen**



**Registers:**

* **AX:** Holds the value 0018. The lower byte (AL) is 18, and the higher byte (AH) is 00.
* **BX:** Holds the value 0023.
* **CX:** Holds the value 0000.
* **DX:** Holds the value 0000.
* **CS:** Holds the value 0718. This is the Code Segment register, pointing to the current code segment.
* **IP:** Holds the value 0000. This is the Instruction Pointer, pointing to the next instruction to be executed.

**Memory Address:**

* The current memory address is 0718:0000. This is calculated by combining the contents of the CS and IP registers: (CS \* 16) + IP = (0718 \* 16) + 0000 = 07180.

**Current Instruction:**

* The current instruction is located at memory address 07180. This is the instruction MOV AX, 00717h.

**Instruction Breakdown:**

* MOV AX, 00717h: This instruction moves the value 00717h (hexadecimal 717) into the AX register.

**Binary Representation:**

* The binary representation of the values in the registers is as follows:
  + AX: 00000000 00011000
  + BX: 00000000 00010011
  + CX: 00000000 00000000
  + DX: 00000000 00000000
  + CS: 00000111 00011000
  + IP: 00000000 00000000

**Other Instructions:**

* The NOP instructions do nothing. They are often used for padding or debugging purposes.
* The INT 21h instruction is a software interrupt that is used to call the operating system's services.
* The RET instruction returns from a procedure.

The registers store various values, and the CS:IP register pair points to the current instruction. The instruction at 07180 is currently being executed, which moves the value 00717h into the AX register.

## Task 3:

## Declare two strings in data section and print these using LEA instruction with second string displayed on next line.

## Code:

.MODEL SMALL

.STACK 100

.DATA

msg1 DW "Enter Your Name: $"

msg2 DW 10,13,"Enter Your SEMESTER : $"

msg3 DB 10,13,"YOUR NAME IS: $"

msg4 DB 10,13,"Your Semester is : $"

input\_name DB 20 DUP('$')

input\_sem DB 20 DUP('$')

.CODE

MOV AX,@DATA

MOV DS,AX

; Display the name prompt message

LEA DX,msg1

MOV AH,09H

INT 21H

; Read NAME into input\_name

LEA DX,input\_name

MOV AH,0AH

INT 21H

; Display the Semester prompt message

LEA DX,msg2

MOV AH,09H

INT 21H

; Read NAME into input\_sem

LEA DX,input\_sem

MOV AH,0AH

INT 21H

; Display the your name message

LEA DX,msg3

MOV AH,09H

INT 21H

; Display your name

LEA DX,input\_name+2

MOV AH,09H

INT 21H

;Display your semeter message

LEA DX,msg4

MOV AH,09H

INT 21H

; Display Semester

LEA DX,input\_sem+2

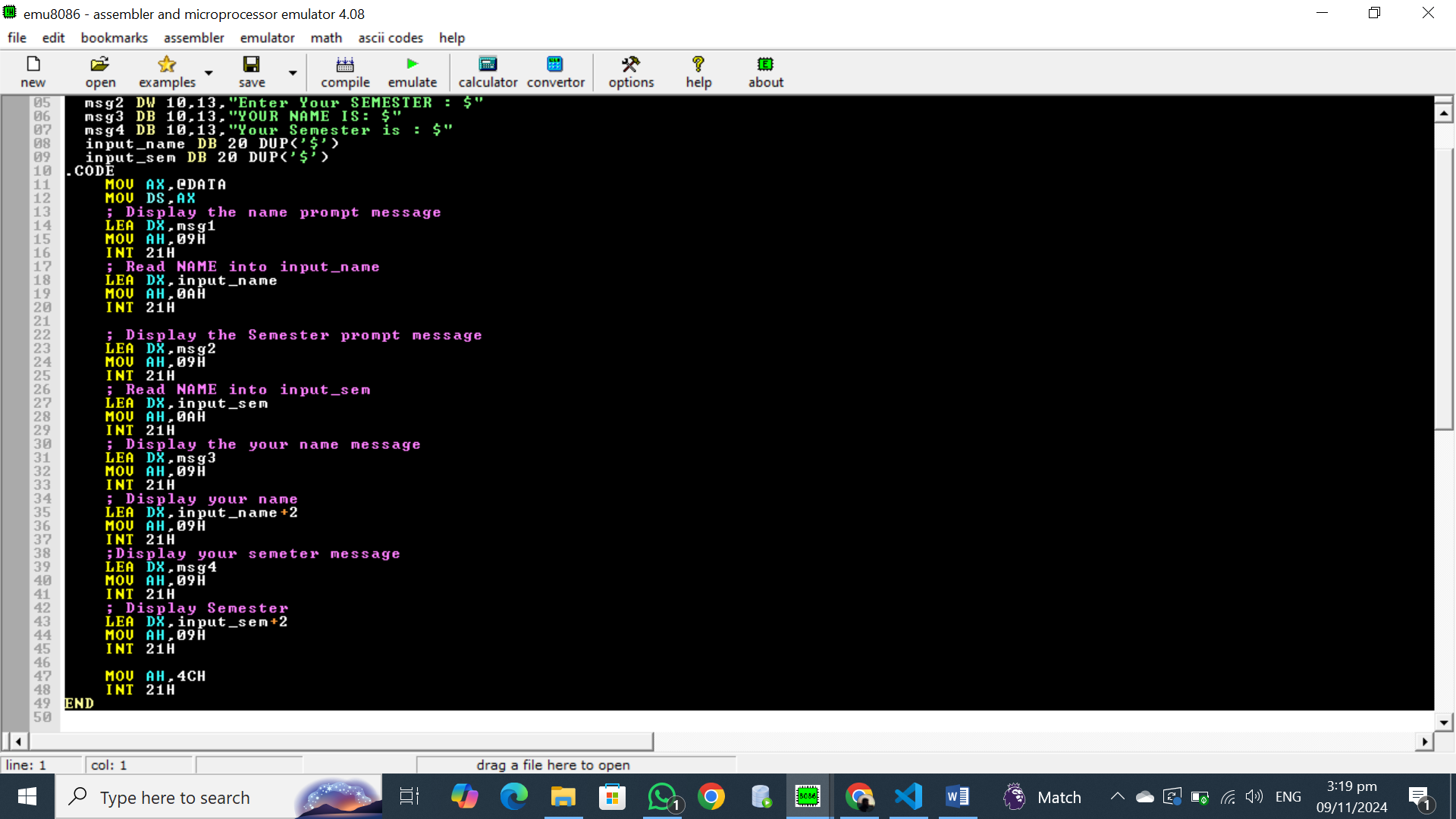
MOV AH,09H

INT 21H

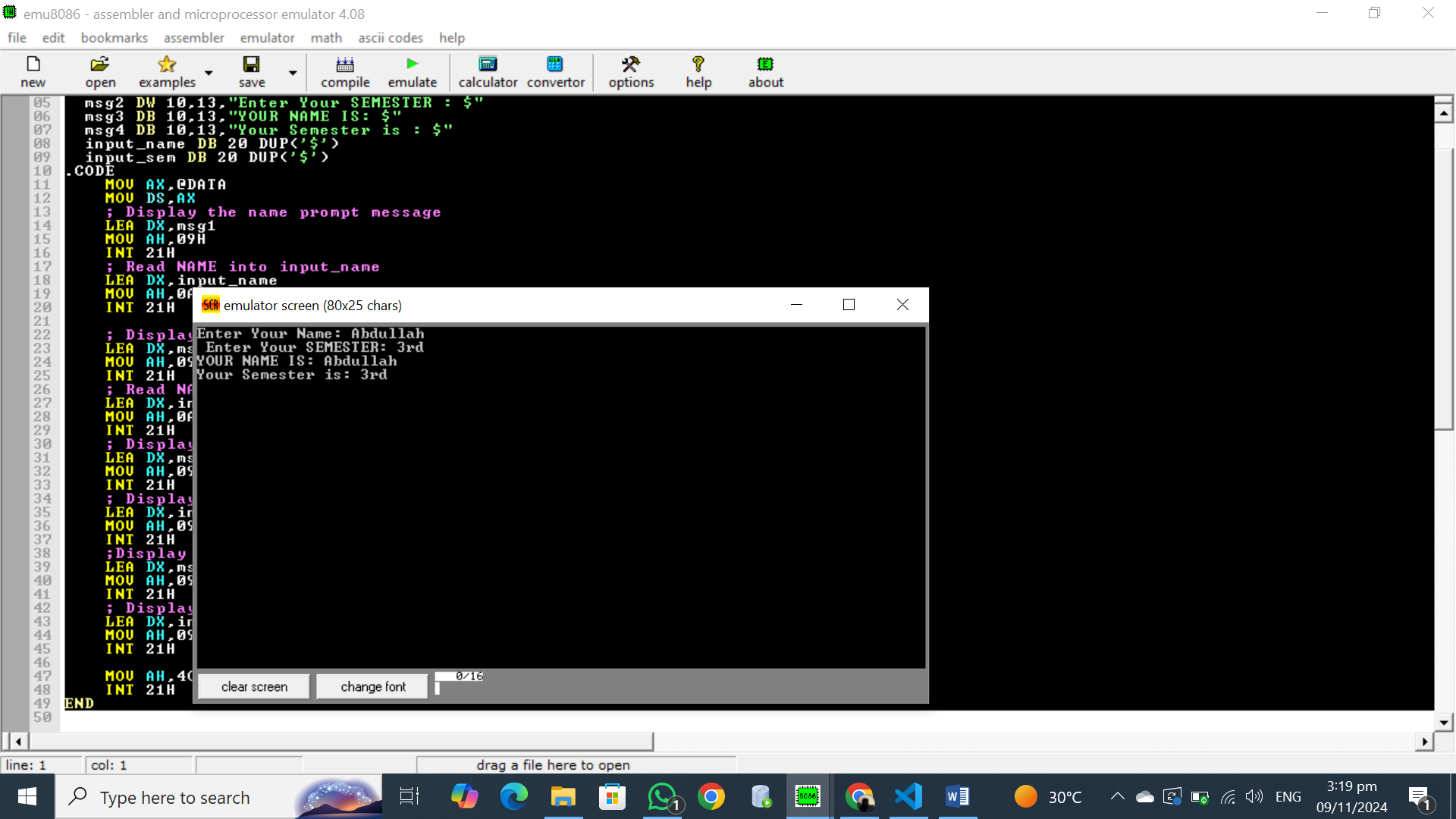
MOV AH,4CH

INT 21H

END



## OUTPUT:



## Explanation:

**Registers:**

* **AX:** Holds the value 0018. The lower byte (AL) is 18, and the higher byte (AH) is 00.
* **BX:** Holds the value 0023.
* **CX:** Holds the value 0000.
* **DX:** Holds the value 0000.
* **CS:** Holds the value 071F. This is the Code Segment register, pointing to the current code segment.
* **IP:** Holds the value 0000. This is the Instruction Pointer, pointing to the next instruction to be executed.

**Memory Address:**

* The current memory address is 071F:0000. This is calculated by combining the contents of the CS and IP registers: (CS \* 16) + IP = (071F \* 16) + 0000 = 071F0.

**Current Instruction:**

* The current instruction is located at memory address 071F0. This is the instruction MOV AX, 00717h.

**Instruction Breakdown:**

* MOV AX, 00717h: This instruction moves the value 00717h (hexadecimal 717) into the AX register.

**Binary Representation:**

* The binary representation of the values in the registers is as follows:
  + AX: 00000000 00011000
  + BX: 00000000 00010011
  + CX: 00000000 00000000
  + DX: 00000000 00000000
  + CS: 00000111 00011111
  + IP: 00000000 00000000

**Other Instrctions:**

* The NOP instructions do nothing. They are often used for padding or debugging purposes.
* The INT 21h instruction is a software interrupt that is used to call the operating system's services.
* The RET instruction returns from a procedure.

The registers store various values, and the CS:IP register pair points to the current instruction. The instruction at 071F0 is currently being executed, which moves the value 00717h into the AX register.

This code program takes input from the user and displays it on the screen. The data segment contains messages for prompting the user and displaying the input. The code segment contains instructions to move data between registers, call DOS interrupts to display messages and read input, and manipulate the stack.

## Task 4:

## Write a simple assembly code to take a character from user input.

## Code:

ORG 100H

.CODE

MOV AH,01

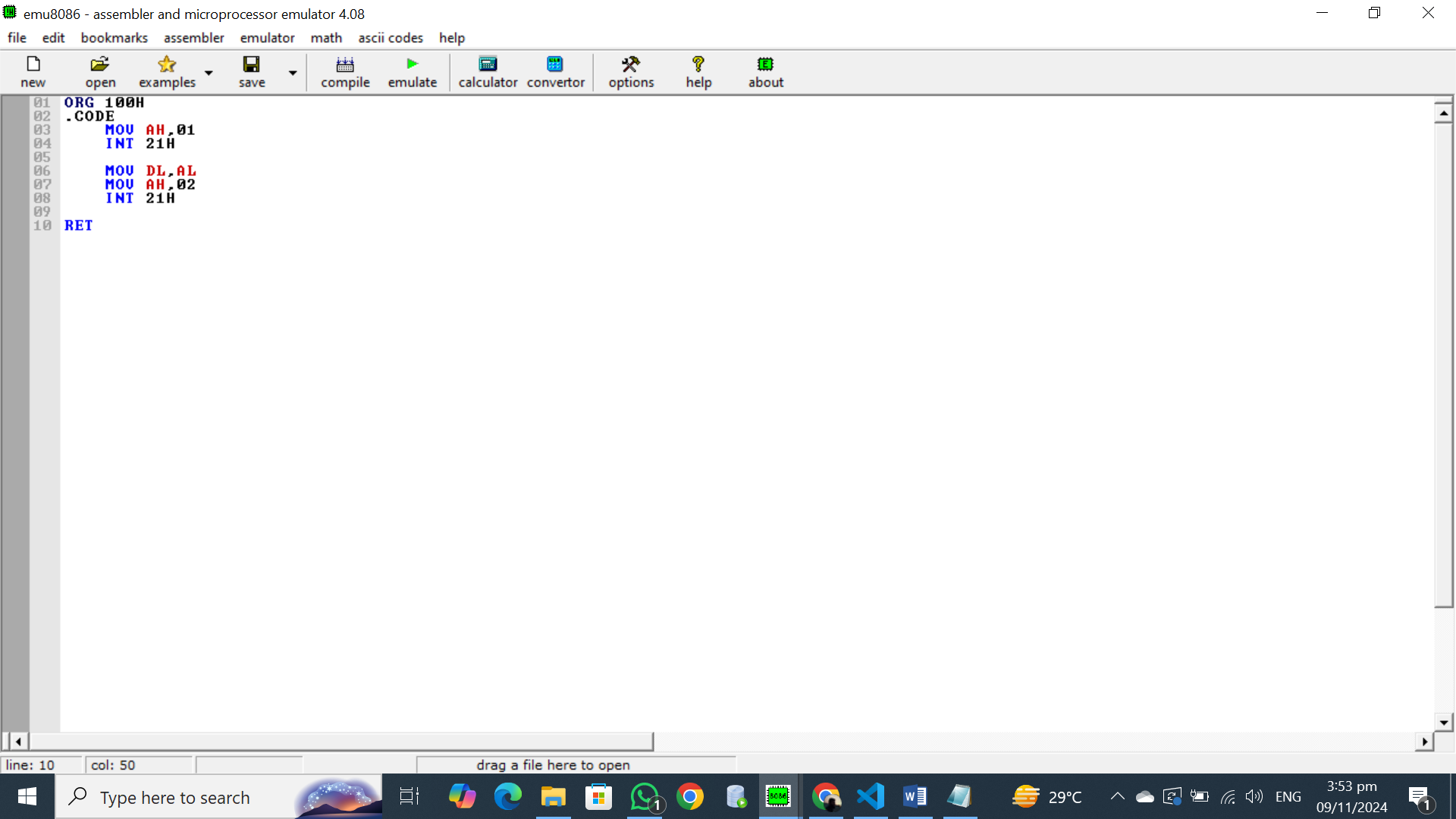
INT 21H

MOV DL,AL

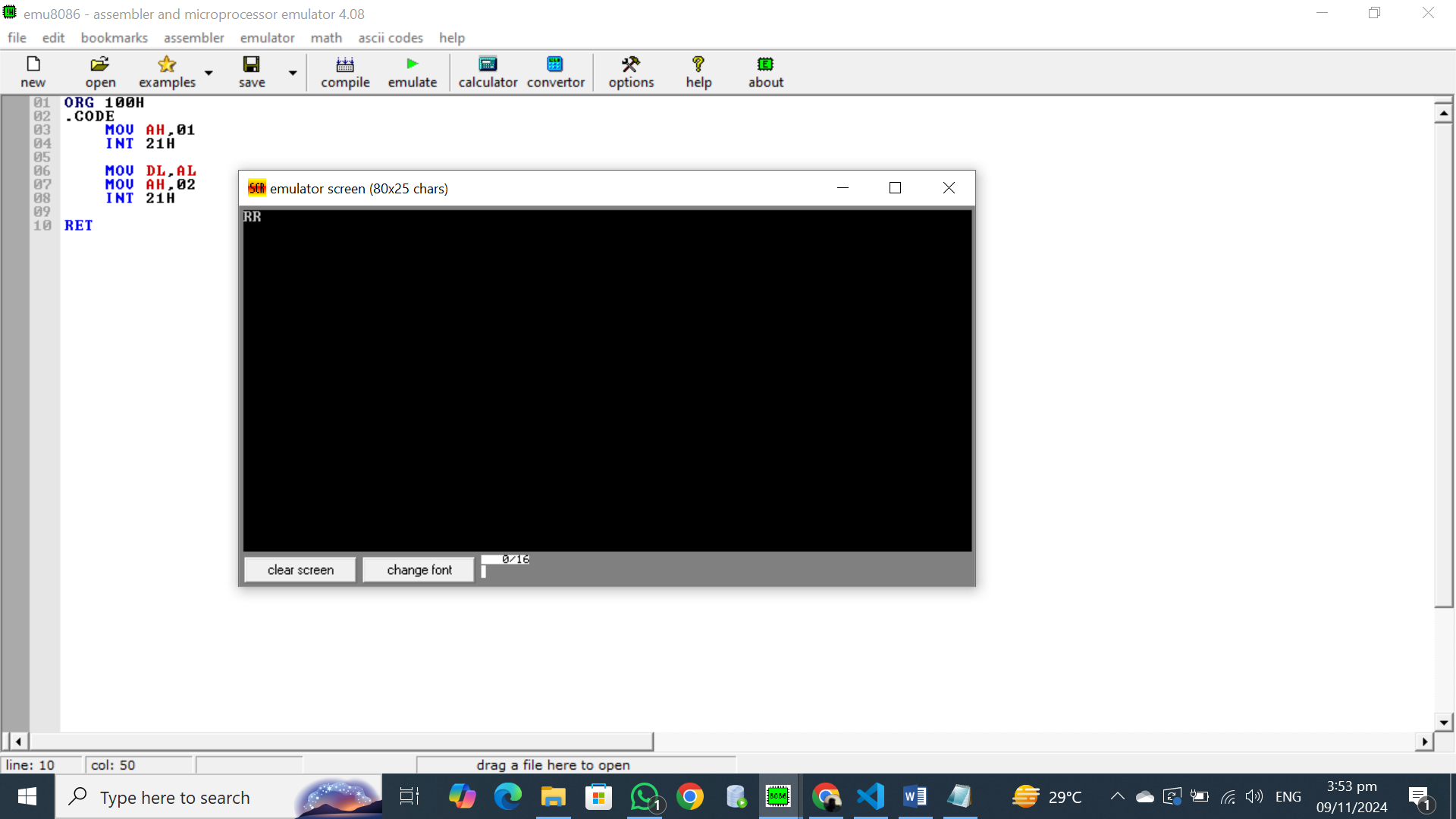
MOV AH,02

INT 21H

RET



## OUTPUT:



## Explanation:

**Input/Output of Characters in 8086**

In 8086 assembly, we typically use DOS interrupts to handle input and output operations. Here's a breakdown of how character input and output work:

**Input:**

1. **Setting the Function Code:**
   * We set the AH register to 01h. This indicates to the DOS interrupt handler that we want to read a character from the keyboard.
2. **Calling the Interrupt:**
   * We execute the INT 21h instruction. This interrupts the current program execution and transfers control to the DOS interrupt handler.
3. **Reading the Character:**
   * The DOS interrupt handler reads the character from the keyboard and stores it in the AL register.

**Output:**

1. **Setting the Function Code:**
   * We set the AH register to 02h. This indicates to the DOS interrupt handler that we want to display a character on the screen.
2. **Preparing the Character:**
   * We move the character to be displayed from the AL register (or any other register) to the DL register.
3. **Calling the Interrupt:**
   * We execute the INT 21h instruction. This interrupts the current program execution and transfers control to the DOS interrupt handler.
4. **Displaying the Character:**
   * The DOS interrupt handler displays the character stored in the DL register on the screen.

## Task 5:

## Write assembly code to take a character from user input and display it. Print messages before taking input and displaying it.

## Code:

ORG 100H

.DATA

msg1 DW "Enter a character: $"

msg2 DW "You entered: $"

input\_char DB ? ; space for the input character

.CODE

; Display the first message

MOV DX, offset msg1

MOV AH, 09H ; Function 09H to display a string

INT 21H

; Take character input from the user

MOV AH, 01H

INT 21H

MOV input\_char, AL

; Stored the entered character in memory

; Move to a new line after input

MOV DL, 10

MOV AH, 02H

INT 21H

MOV DL, 13

MOV AH, 02H

INT 21H

; Display the second message

MOV DX, offset msg2

MOV AH, 09H

INT 21H

; Display the stored character

MOV DL, input\_char

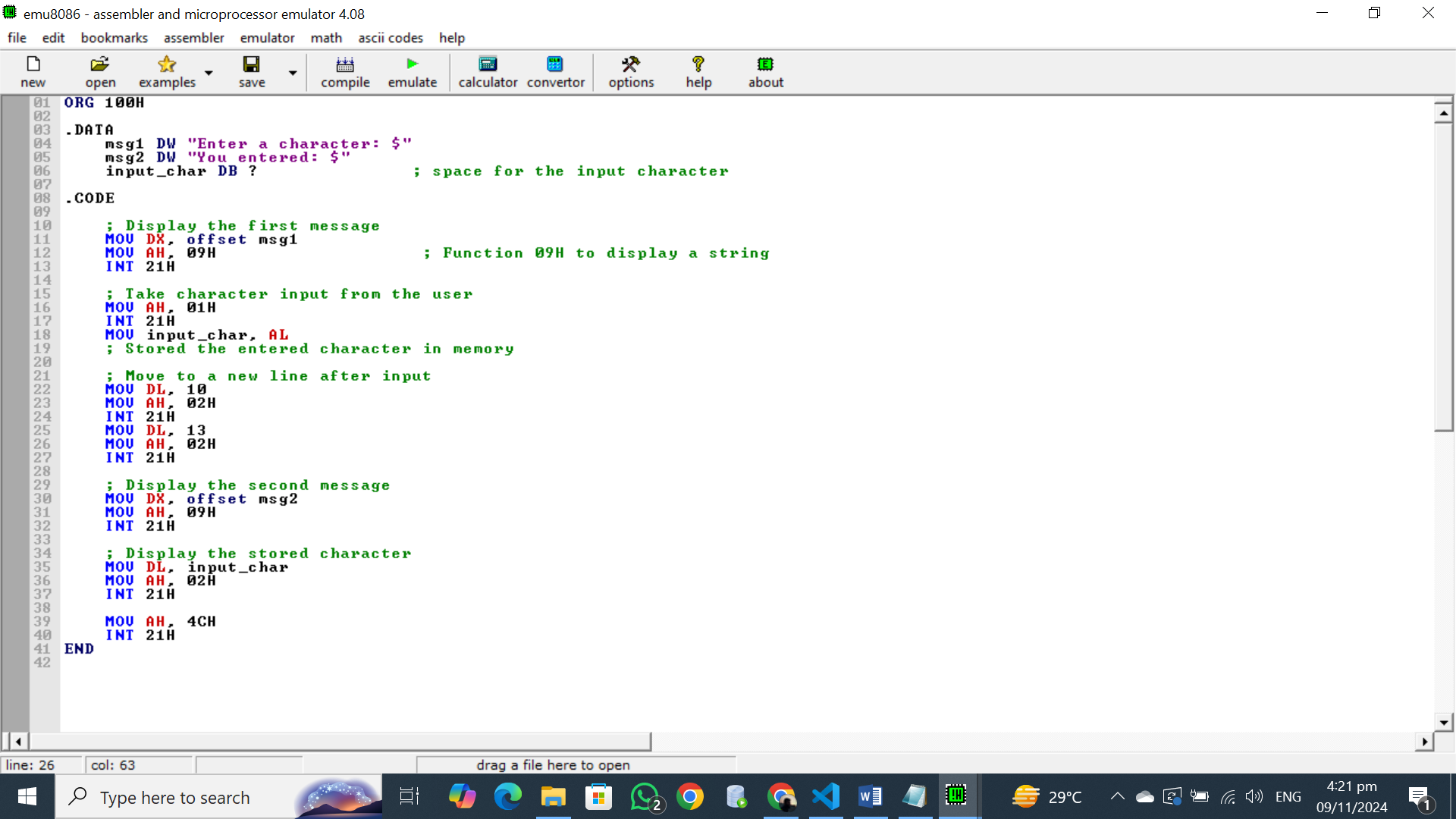
MOV AH, 02H

INT 21H

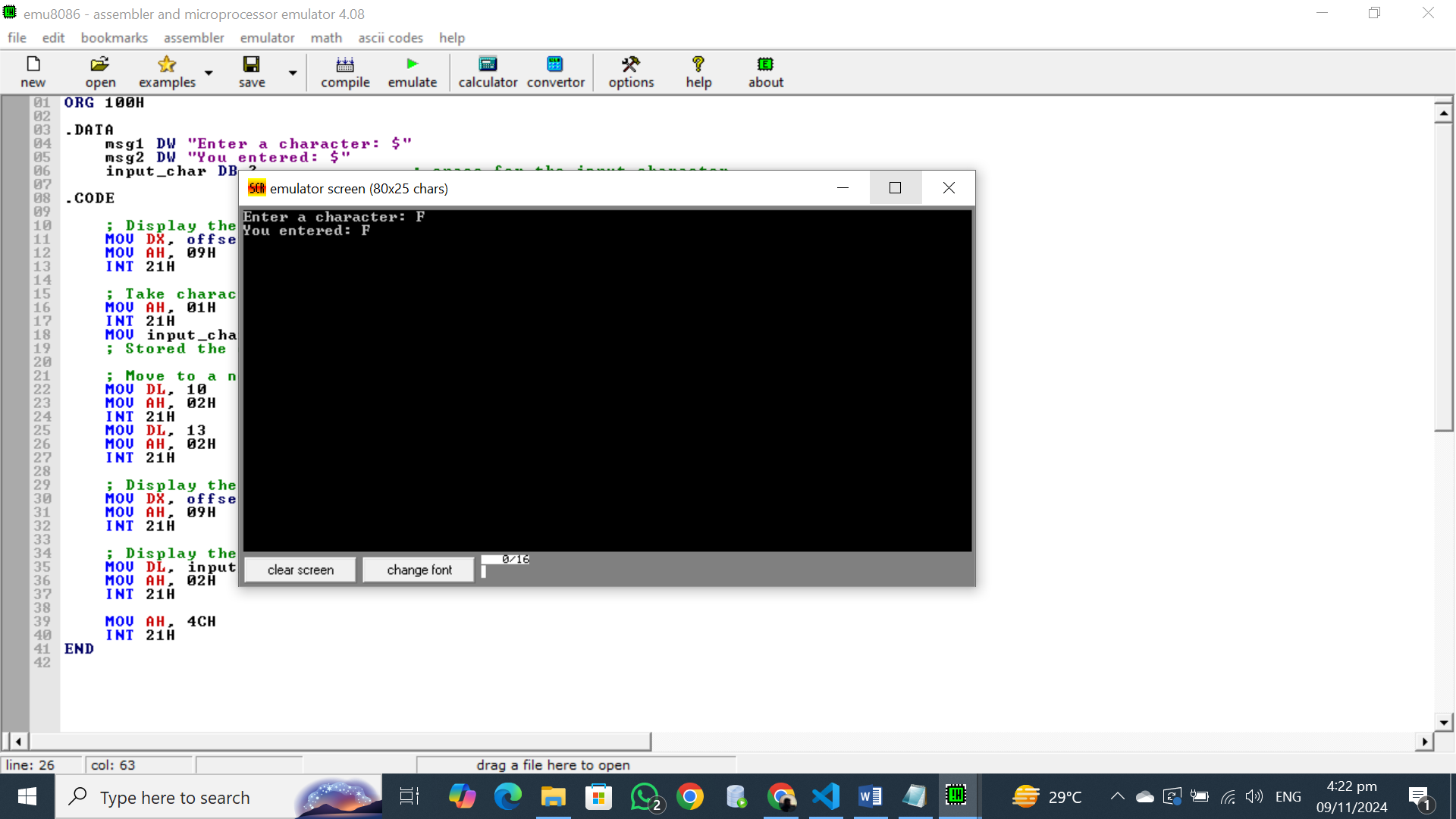
MOV AH, 4CH

INT 21H

END



## OUTPUT:



## Explanation:

**Input/Output of Characters in 8086**

In 8086 assembly, we typically use DOS interrupts to handle input and output operations. Here's a breakdown of how character input and output work:

**Input:**

1. **Setting the Function Code:**
   * We set the AH register to 01h. This indicates to the DOS interrupt handler that we want to read a character from the keyboard.
2. **Calling the Interrupt:**
   * We execute the INT 21h instruction. This interrupts the current program execution and transfers control to the DOS interrupt handler.
3. **Reading the Character:**
   * The DOS interrupt handler reads the character from the keyboard and stores it in the AL register.

**Output:**

1. **Setting the Function Code:**
   * We set the AH register to 02h. This indicates to the DOS interrupt handler that we want to display a character on the screen.
2. **Preparing the Character:**
   * We move the character to be displayed from the AL register (or any other register) to the DL register.
3. **Calling the Interrupt:**
   * We execute the INT 21h instruction. This interrupts the current program execution and transfers control to the DOS interrupt handler.
4. **Displaying the Character:**
   * The DOS interrupt handler displays the character stored in the DL register on the screen.

## Task 6:

## Write assembly code to print this output:

## 

## Code:

ORG 100H

.DATA

character DB '\*' ; Character to display

newLine DB 10, 13, '$'

.CODE

MOV CX, 1

OUTER\_LOOP:

PUSH CX ; Save the row counter

MOV BX, CX ; Set the number of asterisks to print in this row

INNER\_LOOP:

; Print an asterisk

MOV DL, character

MOV AH, 02H

INT 21H

DEC BX ; Decrement the counter for asterisks

JNZ INNER\_LOOP ; Repeat if there are still asterisks to print

; Print newline after each row

MOV DX, offset newLine

MOV AH, 09H

INT 21H

POP CX ; Restore the row counter

INC CX ; Increment the row counter

CMP CX, 6 ; Check if 5 rows have been printed

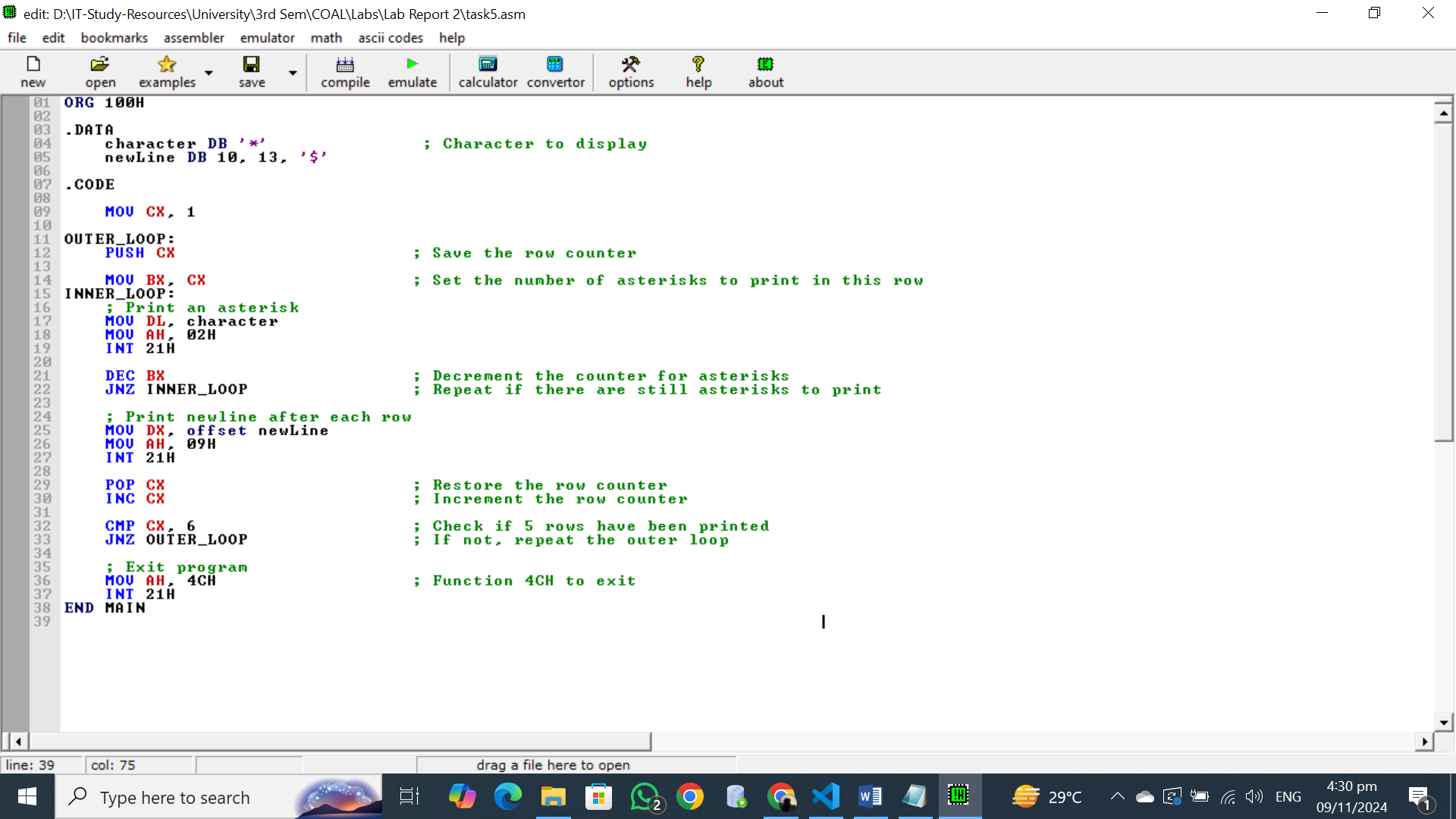
JNZ OUTER\_LOOP ; If not, repeat the outer loop

; Exit program

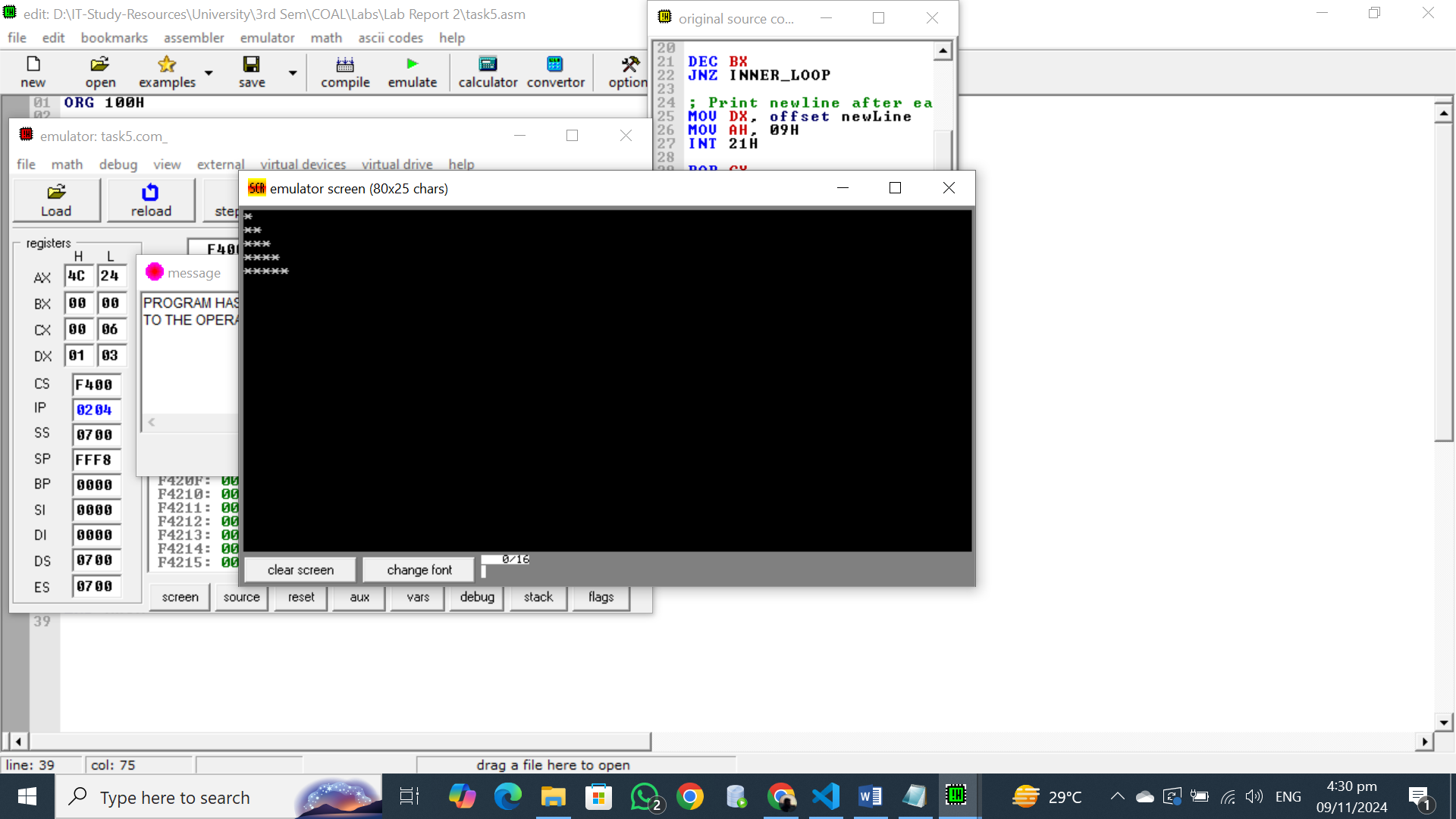
MOV AH, 4CH ; Function 4CH to exit

INT 21H

END



## OUTPUT:



## Explanation:

**Data Segment:**

* **character DB '\*':** Defines a byte variable named 'character' and initializes it with the asterisk character.
* **newLine DB 10, 13, '$':** Defines a byte array named 'newLine' containing ASCII codes for a newline character (10), a carriage return (13), and a dollar sign (termination character).

**Code Segment:**

* **MOV CX, 1:** Initializes the row counter (CX) to 1.
* **OUTER\_LOOP:** Starts an outer loop to control the number of rows.
  + **PUSH CX:** Pushes the current value of CX onto the stack to save it for later.
  + **MOV BX, CX:** Copies the row counter value to BX to use as a column counter.
  + **INNER\_LOOP:** Starts an inner loop to print asterisks in each row.
    - **MOV DL, character:** Loads the asterisk character into the DL register.
    - **MOV AH, 02H:** Sets the function code to 02h for displaying a character.
    - **INT 21H:** Calls the DOS interrupt 21h to display the asterisk.
    - **DEC BX:** Decrements the column counter.
    - **JNZ INNER\_LOOP:** Jumps back to the beginning of the inner loop if BX is not zero.
  + **MOV DX, offset newLine:** Loads the address of the 'newLine' array into the DX register.
  + **MOV AH, 09H:** Sets the function code to 09h for displaying a string.
  + **INT 21H:** Calls the DOS interrupt 21h to display the newline character.
  + **POP CX:** Restores the original value of CX from the stack.
  + **INC CX:** Increments the row counter.
  + **CMP CX, 6:** Compares the row counter with 6.
  + **JNZ OUTER\_LOOP:** Jumps back to the beginning of the outer loop if CX is less than 6.
* **MOV AH, 4CH:** Sets the function code to 4Ch for program termination.
* **INT 21H:** Calls the DOS interrupt 21h to exit the program.

Overall, the code prints a pyramid pattern of asterisks on the screen using nested loops and DOS interrupt functions.

## Task 7:

## Write assembly code to take character from user input and display your name using single characters.

## Code:

.MODEL SMALL

.STACK 100H

.DATA

newLine DB 10,13,'$'

.CODE

MOV AX,@DATA

MOV DS,AX

MOV AH, 01H

MOV AH, 08H

INT 21H

MOV DL, AL

MOV AH, 02H

INT 21H

LEA DX,newLine

MOV AH,09H

INT 21H

;Displaying Abdullah one character at a time

MOV DL, 'A'

MOV AH, 02H

INT 21H

LEA DX,newLine

MOV AH,09H

INT 21H

MOV DL, 'B'

MOV AH, 02H

INT 21H

LEA DX,newLine

MOV AH,09H

INT 21H

MOV DL, 'D'

MOV AH, 02H

INT 21H

LEA DX,newLine

MOV AH,09H

INT 21H

MOV DL, 'U'

MOV AH, 02H

INT 21H

LEA DX,newLine

MOV AH,09H

INT 21H

MOV DL, 'L'

MOV AH, 02H

INT 21H

LEA DX,newLine

MOV AH,09H

INT 21H

MOV DL, 'L'

MOV AH, 02H

INT 21H

LEA DX,newLine

MOV AH,09H

INT 21H

MOV DL, 'A'

MOV AH, 02H

INT 21H

LEA DX,newLine

MOV AH,09H

INT 21H

MOV DL, 'H'

MOV AH, 02H

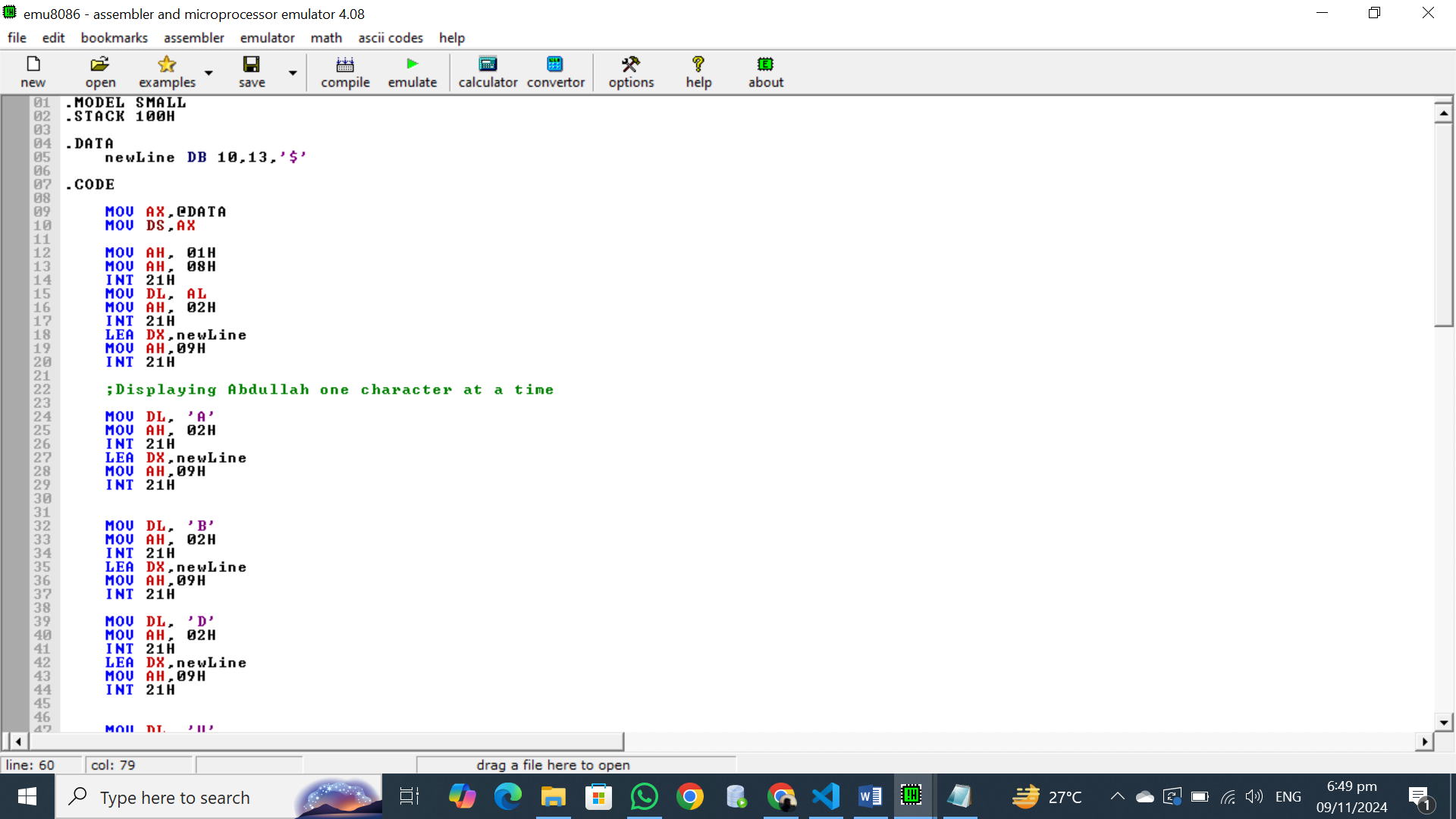
INT 21H

LEA DX,newLine

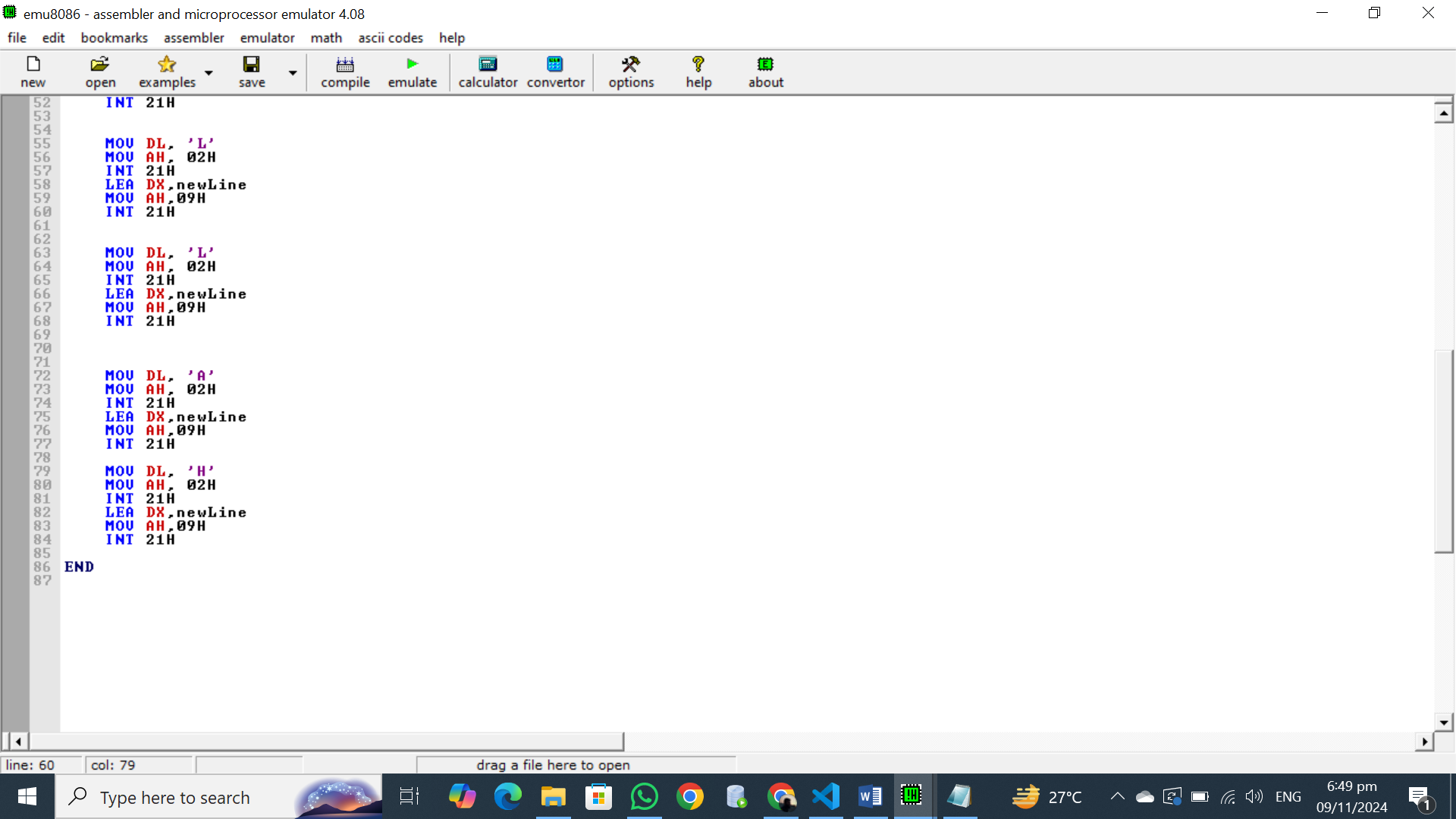
MOV AH,09H

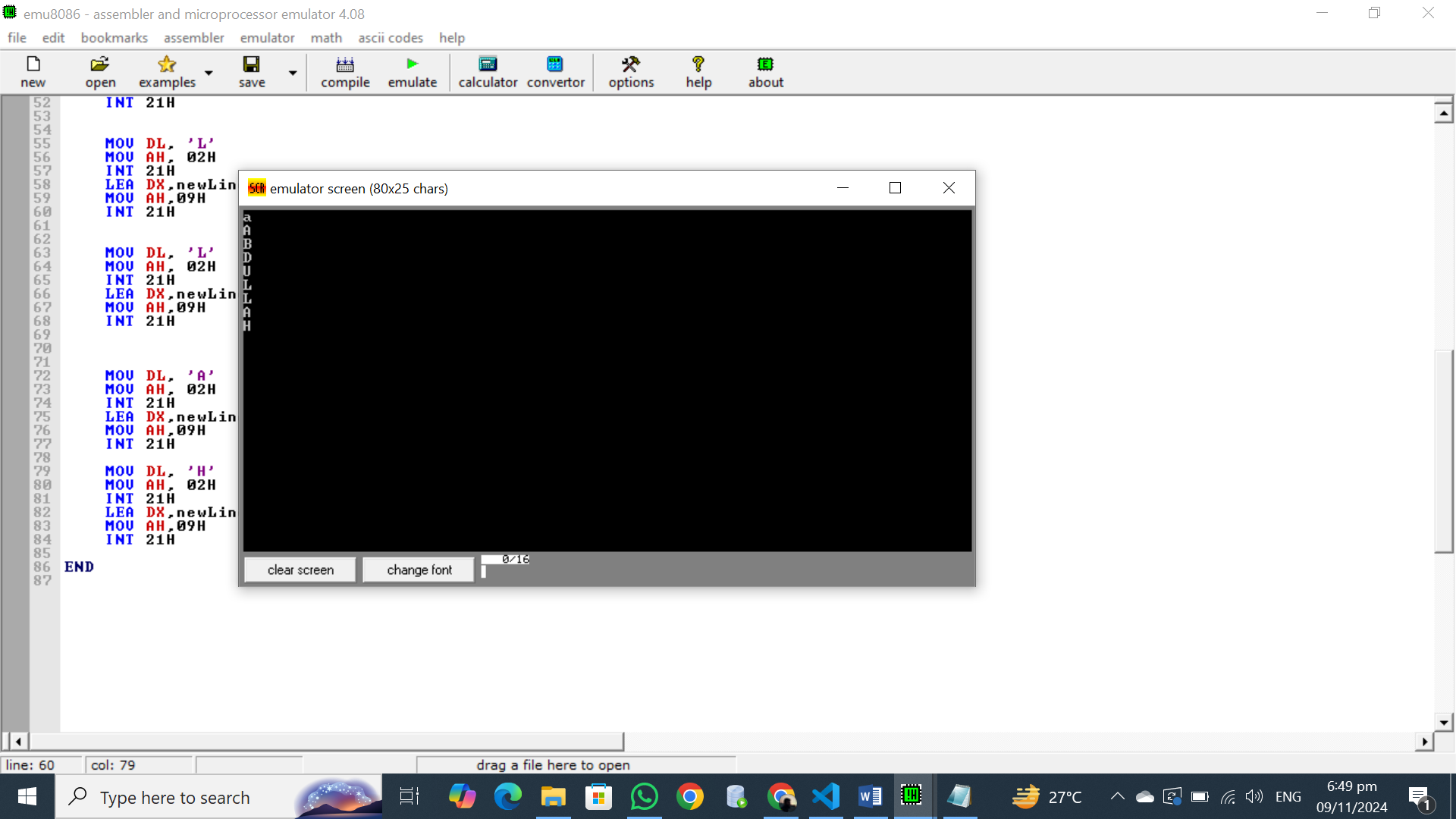
INT 21H

END



## OUTPUT:





## Explanation:

**Input/Output of Characters in 8086**

In 8086 assembly, we typically use DOS interrupts to handle input and output operations. Here's a breakdown of how character input and output work:

**Input:**

1. **Setting the Function Code:**
   * We set the AH register to 01h. This indicates to the DOS interrupt handler that we want to read a character from the keyboard.
2. **Calling the Interrupt:**
   * We execute the INT 21h instruction. This interrupts the current program execution and transfers control to the DOS interrupt handler.
3. **Reading the Character:**
   * The DOS interrupt handler reads the character from the keyboard and stores it in the AL register.

**Output:**

1. **Setting the Function Code:**
   * We set the AH register to 02h. This indicates to the DOS interrupt handler that we want to display a character on the screen.
2. **Preparing the Character:**
   * We move the character to be displayed from the AL register (or any other register) to the DL register.
3. **Calling the Interrupt:**
   * We execute the INT 21h instruction. This interrupts the current program execution and transfers control to the DOS interrupt handler.
4. **Displaying the Character:**
   * The DOS interrupt handler displays the character stored in the DL register on the screen.

## Task 8:

## Write a code read a character from the user without immediately displaying it, and then display that character afterward?

## Code:

ORG 100H

.DATA

msg1 DW "Enter a character: $"

msg2 DW "You entered: $"

input\_char DB ? ; space for the input character

.CODE

; Display the first message

MOV DX, offset msg1

MOV AH, 09H ; Function 09H to display a string

INT 21H

; Take character input from the user

MOV AH, 01H

MOV AH, 08H

INT 21H

MOV input\_char, AL

; Stored the entered character in memory

; Move to a new line after input

MOV DL, 10

MOV AH, 02H

INT 21H

MOV DL, 13

MOV AH, 02H

INT 21H

; Display the second message

MOV DX, offset msg2

MOV AH, 09H

INT 21H

; Display the stored character

MOV DL, input\_char

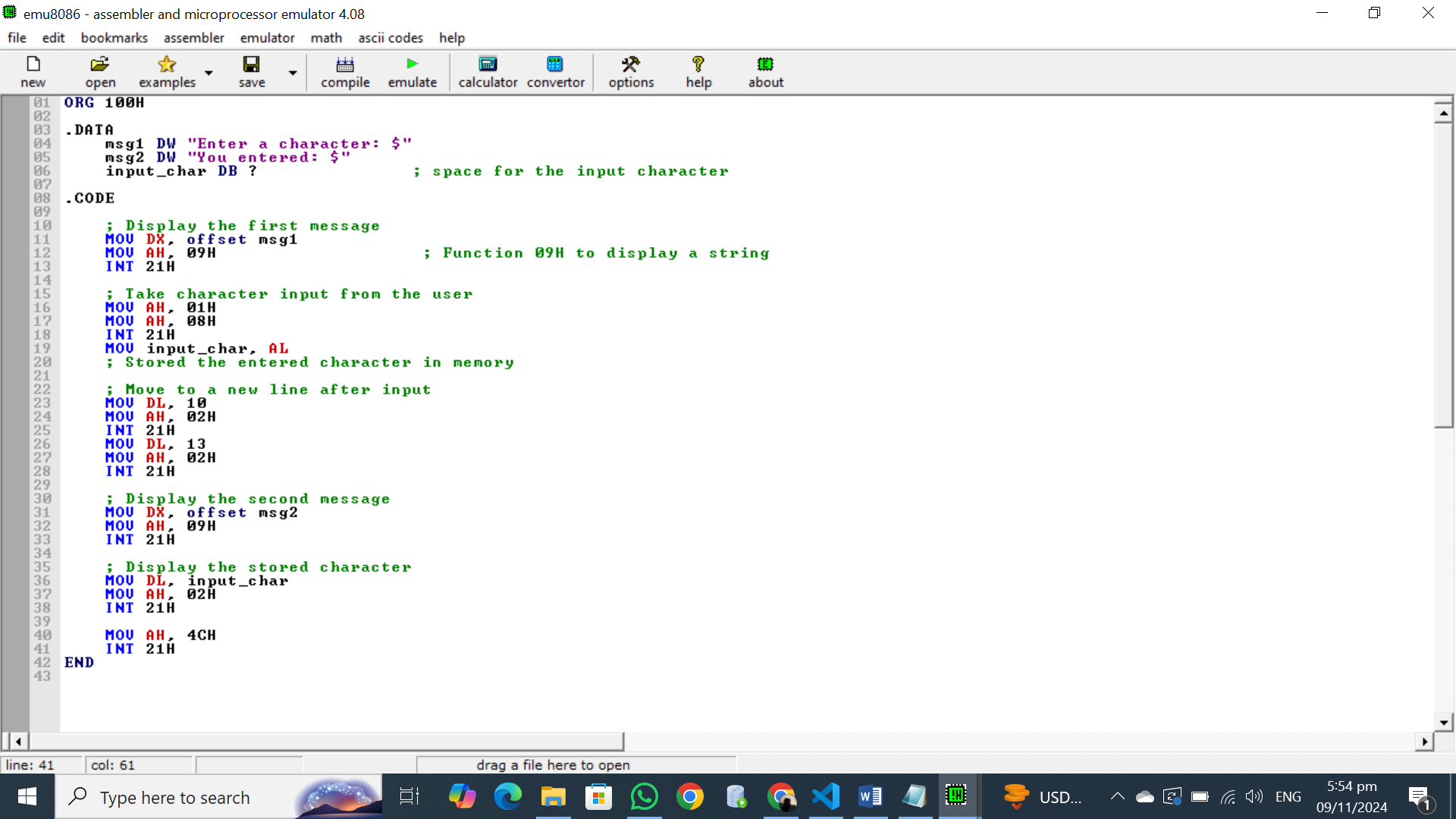
MOV AH, 02H

INT 21H

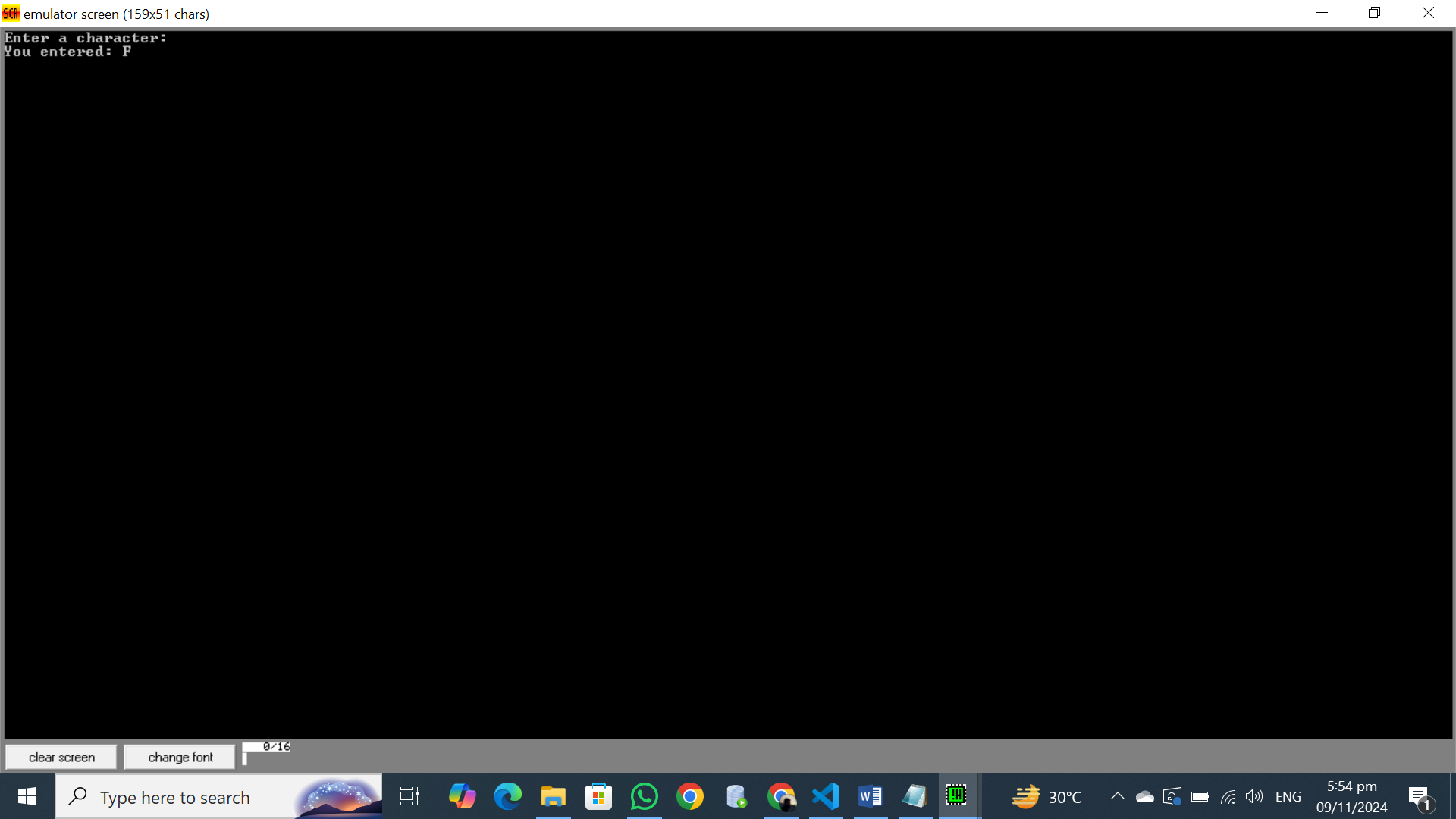
MOV AH, 4CH

INT 21H

END



## OUTPUT:



## Explanation:

**Input/Output of Characters in 8086**

In 8086 assembly, we typically use DOS interrupts to handle input and output operations. Here's a breakdown of how character input and output work:

**Input:**

1. **Setting the Function Code:**
   * We set the AH register to 01h. This indicates to the DOS interrupt handler that we want to read a character from the keyboard.
2. **Calling the Interrupt:**
   * We execute the INT 21h instruction. This interrupts the current program execution and transfers control to the DOS interrupt handler.
3. **Reading the Character:**
   * The DOS interrupt handler reads the character from the keyboard and stores it in the AL register.

**Output:**

1. **Setting the Function Code:**
   * We set the AH register to 02h. This indicates to the DOS interrupt handler that we want to display a character on the screen.
2. **Preparing the Character:**
   * We move the character to be displayed from the AL register (or any other register) to the DL register.
3. **Calling the Interrupt:**
   * We execute the INT 21h instruction. This interrupts the current program execution and transfers control to the DOS interrupt handler.
4. **Displaying the Character:**
   * The DOS interrupt handler displays the character stored in the DL register on the screen.

## Task 9:

## Write a code user to enter their name, store the input in a string array, and then display a message that includes the entered name.

## Code:

.MODEL SMALL

.STACK 100

.DATA

msg1 DW "Enter Your Name: $"

msg3 DB 10,13,"Entered NAME: $"

input\_name DB 20 DUP('$')

.CODE

MOV AX,@DATA

MOV DS,AX

; Display the name prompt message

LEA DX,msg1

MOV AH,09H

INT 21H

; Read NAME into input\_name

LEA DX,input\_name

MOV AH,0AH

INT 21H

; Display the your name message

LEA DX,msg3

MOV AH,09H

INT 21H

; Display your name

LEA DX,input\_name+2

MOV AH,09H

INT 21H

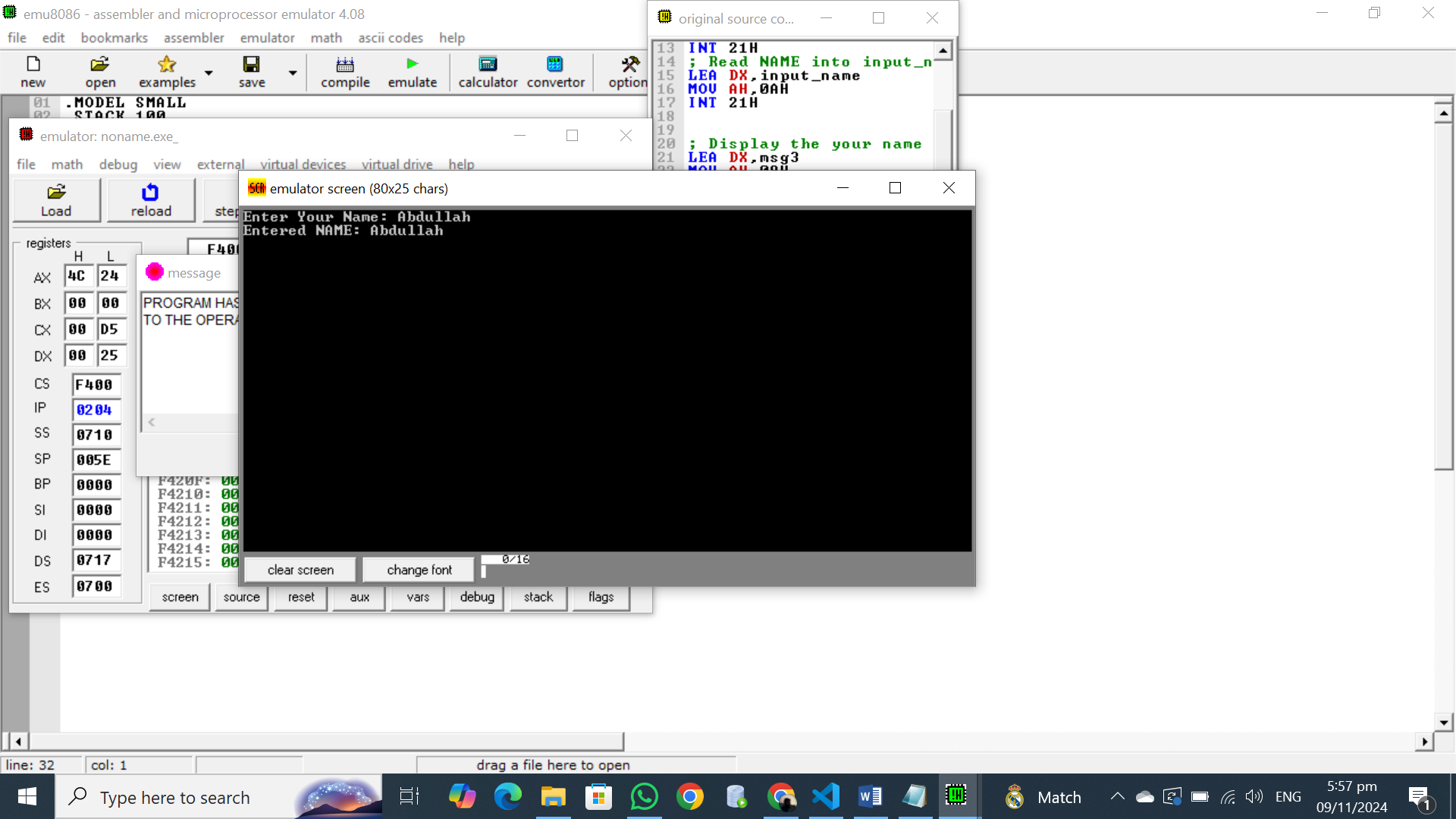
MOV AH,4CH

INT 21H

END



## OUTPUT:



## Explanation:

**Input and Output Functions:**

In 8086 assembly, we primarily use DOS interrupts to handle input and output operations. For string operations, we commonly use interrupts 21h with different function codes:

* **Input:**
  + **Function Code 0Ah:** Reads a string of characters from the keyboard.
  + The string is stored in a buffer, which is a memory location reserved for storing data.
  + The first byte of the buffer should contain the maximum number of characters to read.
  + The second byte will contain the actual number of characters read.
* **Output:**
  + **Function Code 09h:** Displays a string on the screen.
  + The string to be displayed must be terminated with a '$' character.
  + The address of the string is loaded into the DX register.

**Buffer Declaration and DUP:**

A buffer is a reserved memory area to store data temporarily. In the given code, input\_name DB 20 DUP('$') declares a buffer named input\_name of 20 bytes. The DUP directive is used to initialize each byte of the buffer with the '$' character.

**Code Explanation:**

1. **Input:**
   * LEA DX, input\_name: Loads the address of the input\_name buffer into the DX register.
   * MOV AH, 0Ah: Sets the function code to 0Ah for reading a string.
   * INT 21h: Calls the DOS interrupt 21h to read the input string.
2. **Output:**
   * LEA DX, msg1: Loads the address of the msg1 string into the DX register.
   * MOV AH, 09h: Sets the function code to 09h for displaying a string.
   * INT 21h: Calls the DOS interrupt 21h to display the prompt message.
   * LEA DX, msg3: Loads the address of the msg3 string into the DX register.
   * MOV AH, 09h: Sets the function code to 09h for displaying a string.
   * INT 21h: Calls the DOS interrupt 21h to display the "Entered NAME:" message.
   * LEA DX, input\_name+2: Loads the address of the input name (skipping the first two bytes) into the DX register.
   * MOV AH, 09h: Sets the function code to 09h for displaying a string.
   * INT 21h: Calls the DOS interrupt 21h to display the input name.

## Task 10:

## Write assembly code to prompt the user to enter their name, ID, and section. Once all inputs are gathered, the program will display the entered information on the screen. (Note: declare separate variables to take each user input)

## Code:

.MODEL SMALL

.STACK 100

.DATA

msg1 DW "Enter Your Name: $"

msg2 DW 10,13,"Enter Your ID : $"

msg3 DW 10,13,"Enter Your Section : $"

msg4 DB 10,13,"YOUR NAME IS: $"

msg5 DB 10,13,"Your ID is : $"

msg6 DB 10,13,"Your Section is : $"

input\_name DB 20 DUP('$')

input\_id DB 20 DUP('$')

input\_section DB 20 DUP('$')

.CODE

MOV AX,@DATA

MOV DS,AX

; Display the name prompt message

LEA DX,msg1

MOV AH,09H

INT 21H

; Read NAME into input\_name

LEA DX,input\_name

MOV AH,0AH

INT 21H

; Display the ID prompt message

LEA DX,msg2

MOV AH,09H

INT 21H

; Read NAME into input\_id

LEA DX,input\_id

MOV AH,0AH

INT 21H

; Display the Section prompt message

LEA DX,msg3

MOV AH,09H

INT 21H

; Read NAME into input\_section

LEA DX,input\_section

MOV AH,0AH

INT 21H

; Display the your name message

LEA DX,msg4

MOV AH,09H

INT 21H

; Display your name

LEA DX,input\_name+2

MOV AH,09H

INT 21H

;Display your id message

LEA DX,msg5

MOV AH,09H

INT 21H

; Display id

LEA DX,input\_id+2

MOV AH,09H

INT 21H

;Display your Section message

LEA DX,msg6

MOV AH,09H

INT 21H

; Display Section

LEA DX,input\_section+2

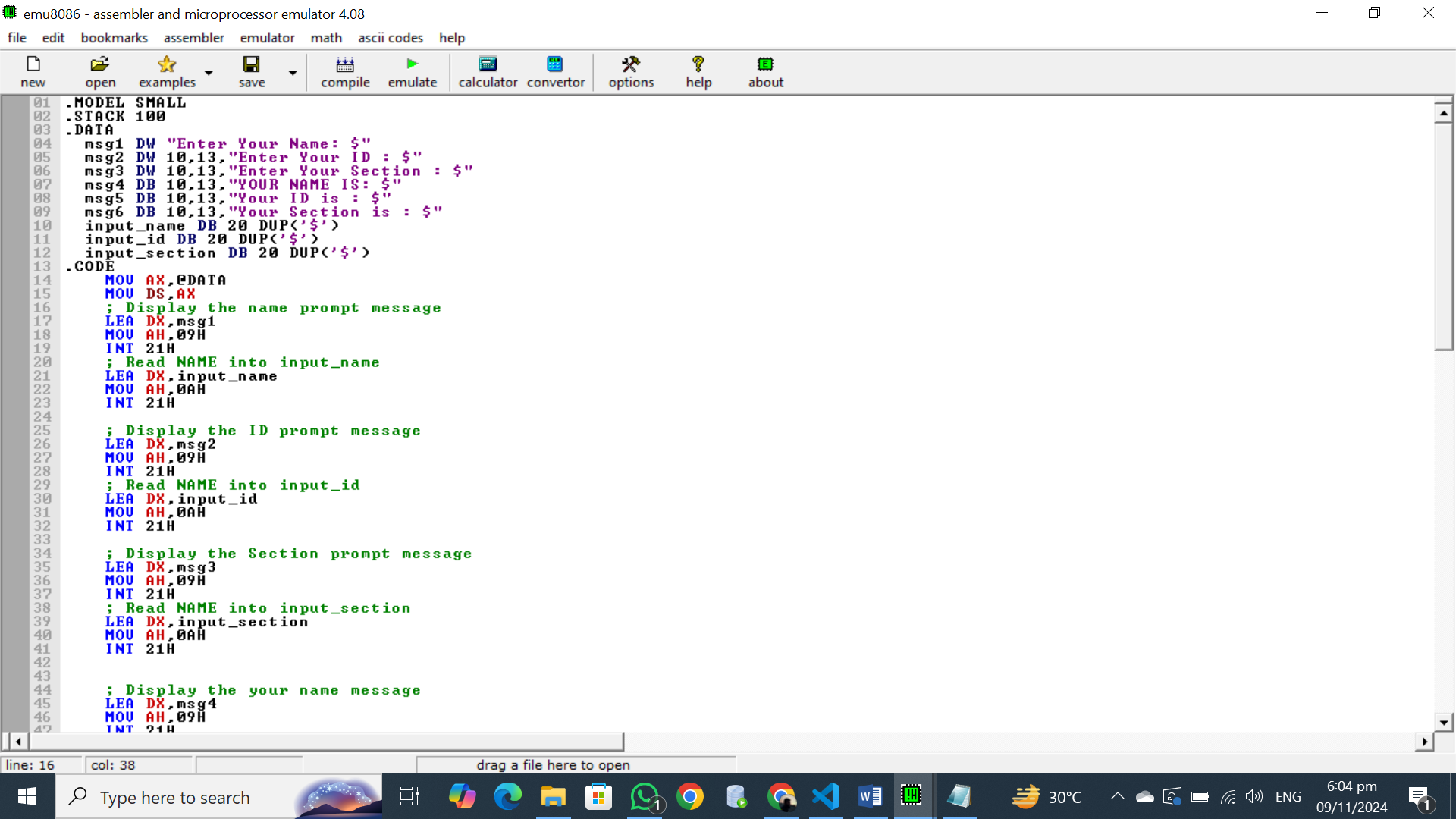
MOV AH,09H

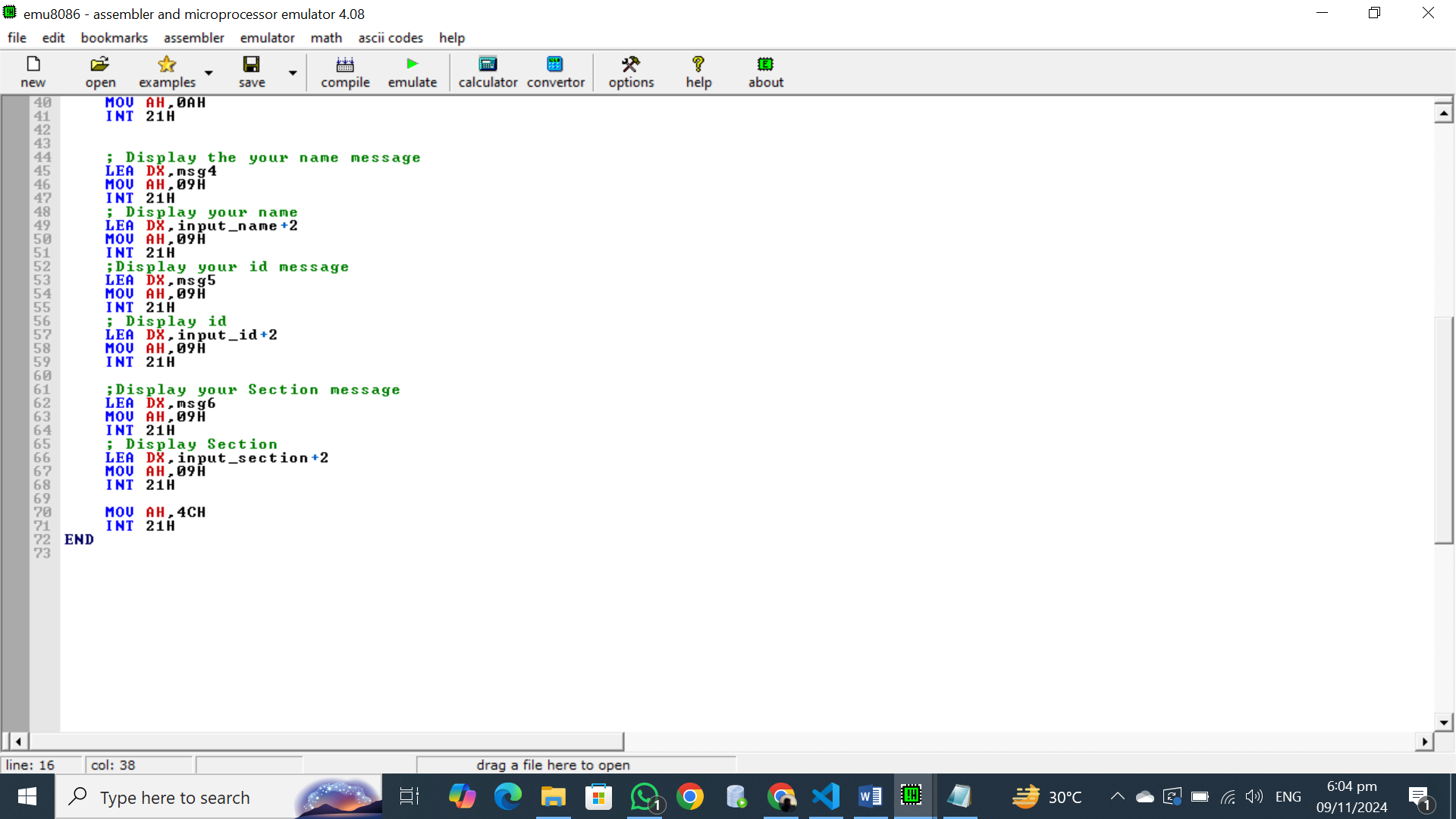
INT 21H

MOV AH,4CH

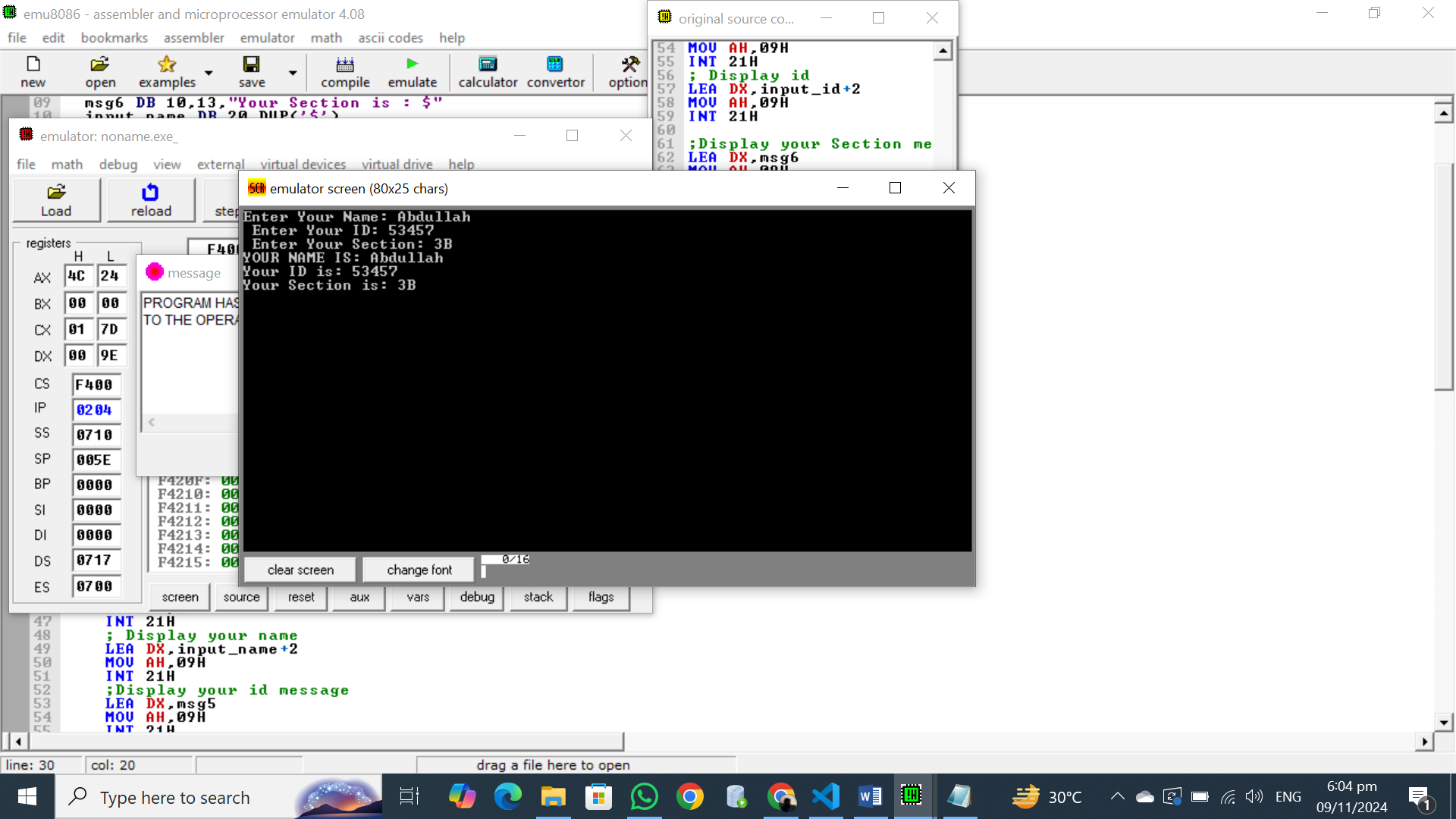
INT 21H

END





## OUTPUT:



## Explanation:

**Input and Output Functions:**

In 8086 assembly, we primarily use DOS interrupts to handle input and output operations. For string operations, we commonly use interrupts 21h with different function codes:

* **Input:**
  + **Function Code 0Ah:** Reads a string of characters from the keyboard.
  + The string is stored in a buffer, which is a memory location reserved for storing data.
  + The first byte of the buffer should contain the maximum number of characters to read.
  + The second byte will contain the actual number of characters read.
* **Output:**
  + **Function Code 09h:** Displays a string on the screen.
  + The string to be displayed must be terminated with a '$' character.
  + The address of the string is loaded into the DX register.

**Buffer Declaration and DUP:**

A buffer is a reserved memory area to store data temporarily. In the given code, input\_name DB 20 DUP('$') declares a buffer named input\_name of 20 bytes. The DUP directive is used to initialize each byte of the buffer with the '$' character.

## Task 11:

## Explain in your own words:

## What are different segments in assembly program and where their addresses are stored?

## What are ‘model’ and ‘stack’ assembler directives?

**Answer:**

**1. Segments in Assembly Programming**

In assembly programming, segments are logical divisions of memory used to organize code, data, and the stack. The most common segments are:

* **Code Segment (CS):**
  + Contains the executable instructions of the program.
  + Its address is stored in the CS register.
* **Data Segment (DS):**
  + Contains initialized and uninitialized data used by the program.
  + Its address is stored in the DS register.
* **Extra Segment (ES):**
  + Often used for additional data or as a temporary storage area.
  + Its address is stored in the ES register.
* **Stack Segment (SS):**
  + Contains the program's stack, used for storing function parameters, return addresses, and local variables.
  + Its address is stored in the SS register.

**2. Model and Stack Directives**

* **Model Directive:**
  + Specifies the memory model used for the program, which determines how memory is allocated for code, data, and the stack.
  + Common memory models include:
    - **SMALL:** A single 64KB segment for code and data.
    - **MEDIUM:** Two 64KB segments, one for code and one for data.
    - **COMPACT:** One 64KB segment for code and multiple 64KB segments for data.
    - **LARGE:** Multiple 64KB segments for both code and data.
    - **HUGE:** Multiple 64KB segments for code and data, with additional segment registers to access different segments.
  + The model directive influences the default segment registers and memory addressing modes.
* **Stack Directive:**
  + Specifies the size of the stack segment.
  + For example, .STACK 100 allocates 100 bytes for the stack.
  + The stack segment is used for function calls, local variable storage, and other temporary data.