Lab 10

Video Memory and String Instructions

After completing this lab

- Students will be able to write characters directly to video memory.
- Students will be able to translate screen coordinates into linear memory addresses.
- Students will be able to declare strings and display them on screen using loops.
- Students will be able to copy strings using string instructions.
- Students will know how OS displays messages on screen.

Video Memory

The text screen video memory for color monitors resides at **0xB8000**. The text screen resolution is **80 x 25**, which means that there are **25 rows**, and each row has 80 characters, therefore, there are 2000 characters. Each character requires **2 bytes** of information. One byte is for the **ASCII code of character** that is to be displayed on screen, and the other byte is for the **attributes associated with that character**. Attributes contain the foreground and background colors of the character. The **first nibble** (4 least significant bits) is reserved for **foreground (text) color**, and the **second nibble** (4 most significant bits) is **reserved for background color**.

Each character's background and foreground colors can be set or obtained by reading a specific byte from memory.

Attributes

The various fields of the byte reserved for attributes are shown below. RGB are the red, green, and blue colors. 'I' is the intensity of the color, and 'B' is for blink.

Background Color				Foreground Color			
B	R	G	B	I	R	G	B
Blink	Red	Green	Blue	Intensity	Red	Green	Blue

The following table shows the 4-bit color combination.

I/B	R	G	B	Color
0	0	0	0	Black
0	0	0	1	Blue
0	0	1	0	Green
0	0	1	1	Cyan
0	1	0	0	Red
0	1	0	1	Magenta
0	1	1	0	Brown
0	1	1	1	White
1	0	0	0	Gray
1	0	0	1	Light Blue
1	0	1	0	Light Green
1	0	1	1	Light Cyan
1	1	0	0	Light Red
1	1	0	1	Light Magenta
1	1	1	0	Yellow
1	1	1	1	Bright

Example#1:
Program to write "**HELLO**" on screen in red color with yellow background.

```
.model small

.stack 100h

.data

.code

Mov ax,0xb800
Mov es,ax
Mov si,0

Mov es:[si], 'H'
Inc si
Mov byte ptr es:[si], 11100100b
Inc si
Mov es:[si], 'E'
Inc si
Mov byte ptr es:[si], 11100100b
Inc si

Mov es:[si], 'L'
Inc si
Mov byte ptr es:[si], 11100100b
Inc si

Mov es:[si], 'L'
Inc si
Mov byte ptr es:[si], 11100100b
Inc si

Mov es:[si], 'O'
Inc si
Mov byte ptr es:[si], 11100100b
Inc si

.exit
```

Screen coordinates to linear address conversion

The screen is two-dimensional, having rows and columns, while the memory containing the screen's contents is linear. Each character on a screen has a row and column address, which is represented as (row, column), which can be translated to a linear memory address by using the following equations:

Equation 1 finds the linear address of the memory location where the ASCII code of the character to be shown on screen will be stored.

Equation 2 finds the linear address of the memory location where the character's screen attributes will be stored.

This linear address will be added to the base address, i.e., 0xB8000, to form the physical address.

Linear Address $_{\textbf{(ASCII)}}$ = 2 * (Row address * Columns per Row + Column address) **(eq. 1)**

Linear Address $_{\textbf{(Attributes)}}$ = 2 * (Row address * Columns per Row + Column address) + 1 **(eq. 2)**

Here,
Number of rows = 25
Columns per Row = 80
Since there are 80 columns in each row.

Example:

To display a text on the middle of the screen, we need to know its screen coordinates which are (12,40). These coordinates can be translated into physical address by using equation 1 as shown below.

Memory Address = 2 * (12 * 80 + 40) **(eq. 1)**
Memory Address = 2000

Similarly, the offset address of the attributes of center character is

Memory Address = 2 * (12 * 80 + 40) +1 **(eq. 2)**
Memory Address = 2000 + 1 = 2001

Example#2:
Program to write "**HELLO**" on center of the screen.

```
.model small

.stack 100h

.data

.code

Mov ax,0xb800
Mov es,ax
Mov si,2000

Mov es:[si], 'H'
add si,2

Mov es:[si], 'E'
add si,2

Mov es:[si], 'L'
add si,2

Mov es:[si], 'L'
add si,2

Mov es:[si], 'O'

.exit
```

Defining Strings

Strings are defined using a byte array as shown below.

Msg db “Hello World”

A string is terminated by a character which can be NULL or ‘$’ so that the procedure handling string could know that the string has been ended. Since we are going to define our own code to process strings, therefore, we will place 0, i.e., NULL character at the end of a string as shown below.

Msg db “Hello World”,0

Here ‘,’ is the concatenation operator. It will concatenate strings on its both sides to form a single string.

Example#3:
Program that defines a string and displays it through loop.

```
.model small
.stack 100h
.data

msg db "Hello World",0

.code
mov ax,@data
mov ds,ax

Mov ax,0xB800
Mov es,ax
Mov di,0

mov bx,offset msg
mov si,0

display:

mov al,[bx+si]          ;mov one character from array to al
cmp al,0                 ;compare is that character is null
je end                   ; if null, terminate loop
mov es:[di],al           ; otherwise, mov that character to video memory
add di,2                 ; add 2 to di, skipping the attribute part
inc si                  ; inc si to access the next character of array
jmp display             ; iterate loop


end:
.exit
```

String instructions

The following instructions facilitate programmer in moving strings from one memory location to another and comparing strings.

Instruction	Operands	Operation

LODSB
 	
Also see: LODSW
	No operands	• **AL = DS:[SI]**
		• if DF = 0 then
		  ○ SI = SI + 1
		else
		  ○ SI = SI – 1

STOSB
 	
Also See: STOSW
	No operands	• **ES:[DI] = AL**
		• if DF = 0 then
		  ○ DI = DI + 1
		else
		  ○ DI = DI – 1

MOVSB
 	
Also See: MOVSW
	No operands	• **ES:[DI] = DS:[SI]**
		• if DF = 0 then
		  ○ SI = SI + 1
		  ○ DI = DI + 1
		else
		  ○ SI = SI - 1
		  ○ DI = DI – 1

SCASB
 	
Also See: SCASW
	No operands	• **AL - ES:[DI]**
		• set flags according to result:
		OF, SF, ZF, AF, PF, CF
		• if DF = 0 then
		  ○ DI = DI + 1
		else
		  ○ DI = DI - 1

Example#4:
Program to write “Hello world” on screen using string movement instructions.

```
.model small

.stack 100h

.data

msg1 db "Hello World",0
msg2 db 12 dup(?)

.code

mov ax,@data
mov ds,ax
mov es,ax

Mov si,offset msg1        ; moving offset of msg1 to SI
Mov di,offset msg2        ; moving offset of msg2 to DI


mov cx,11                 ; initializing counter to iterate loop 11 times

copy:
movsb                     ; copying msg1 to msg2

loop copy


.exit
```

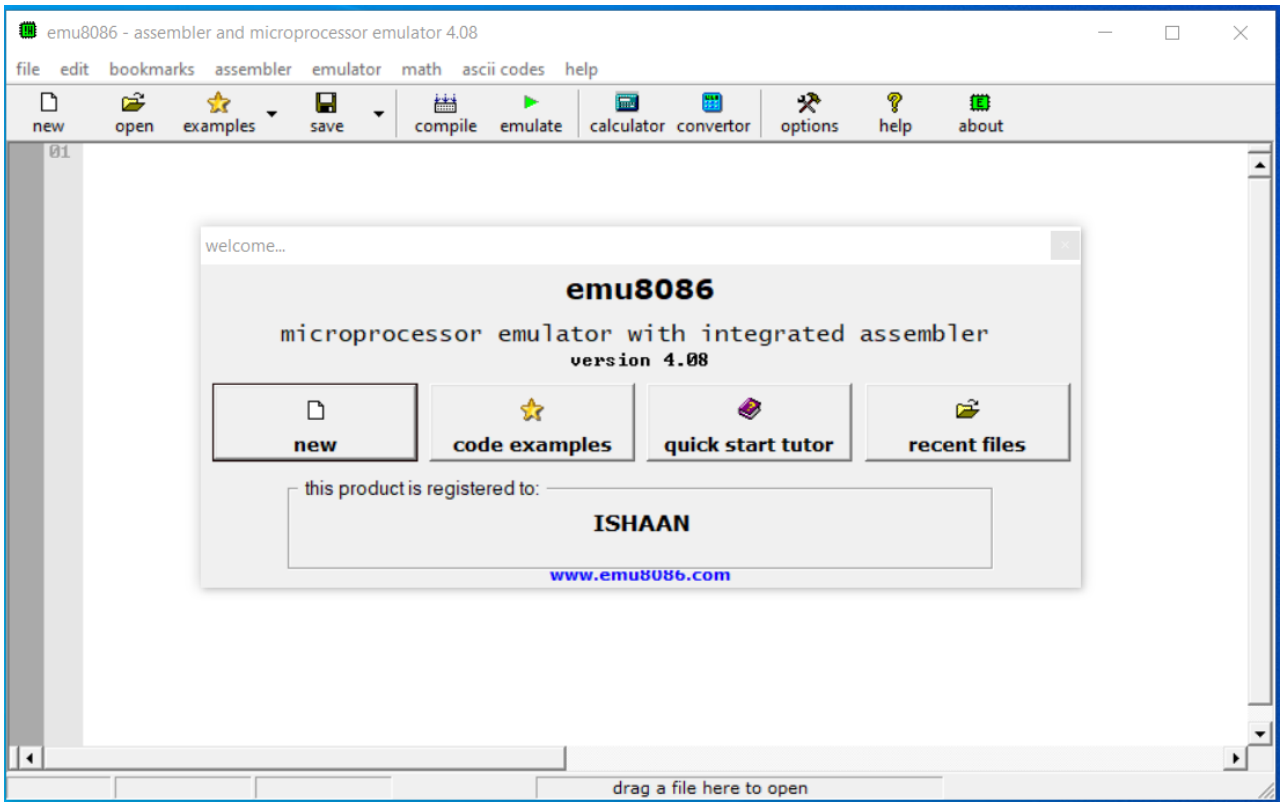
Emu8086 Tutorial Step by Step

Step-1

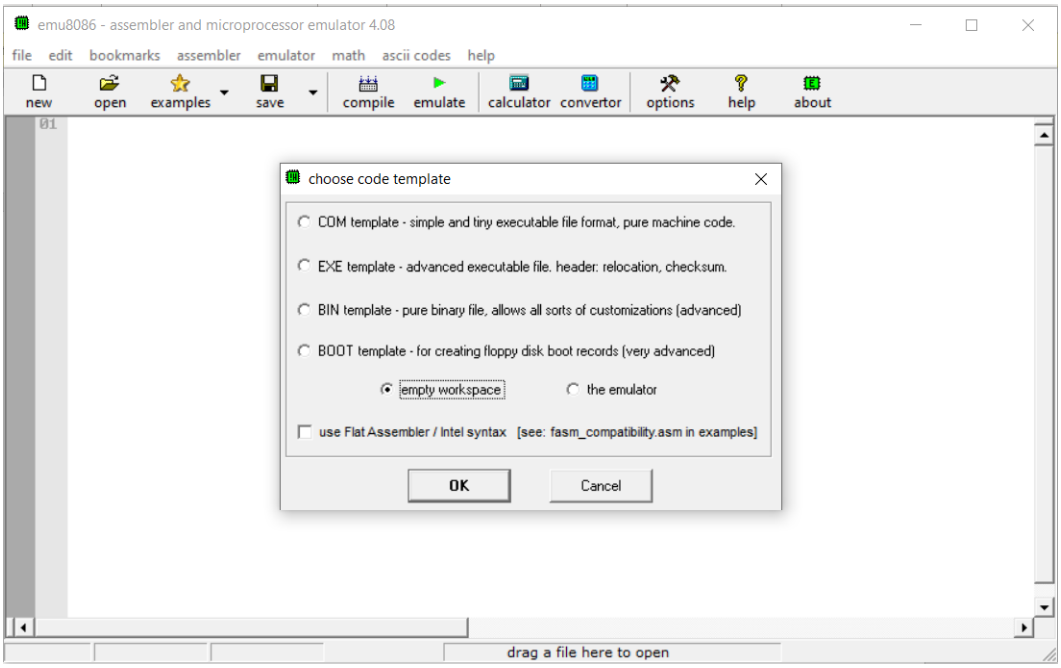
Double click on the icon on the desktop 

Step-2

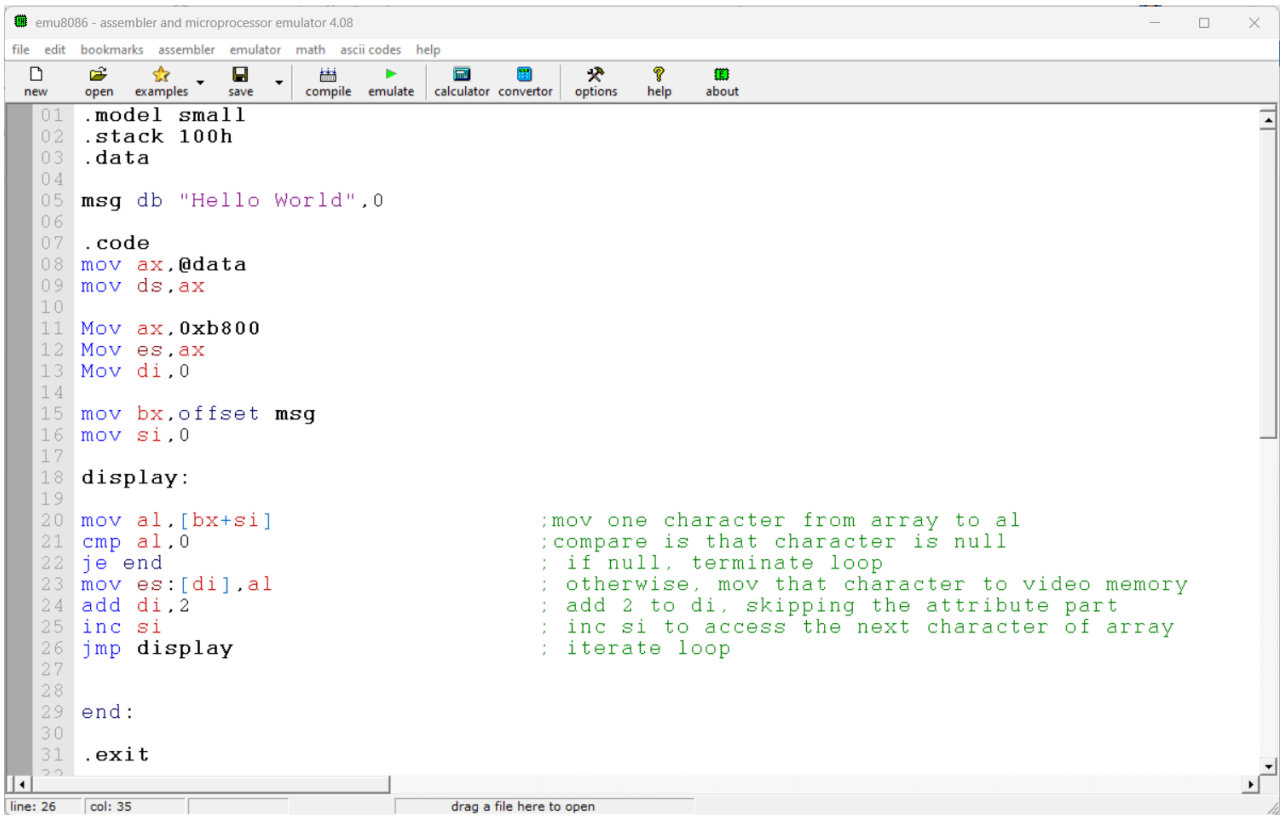
The following window will appear. Click on new.



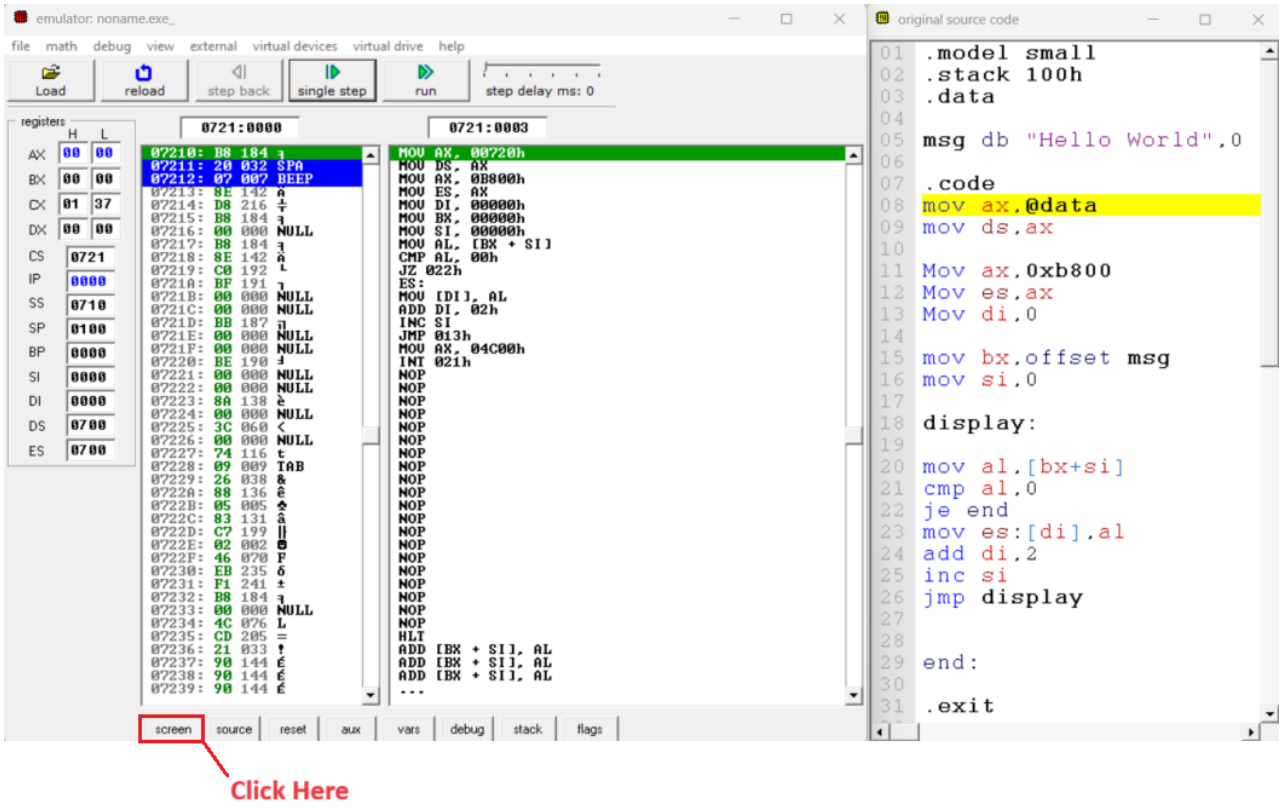
Step-3:
Click on empty workspace and press OK.



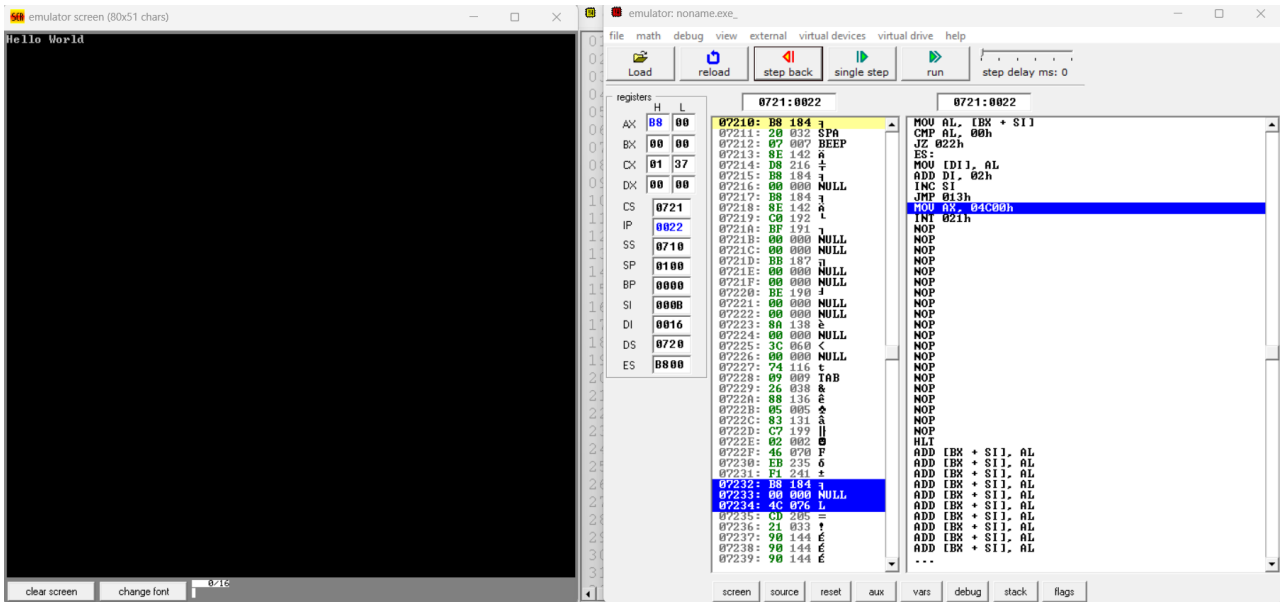
Step-4:
Type the code given in example#3 and click on emulate.



Step-5:
Click on the screen button.



Step-6:
Keep clicking on “Single step” and observe the working of the program.



Practice Exercise

Task-1

Write a procedure named "print" that takes a null-terminated string as a parameter and displays it on the screen at the position of the cursor. The cursor increments accordingly whenever a string is displayed. (Hint: You need to keep cursor information in a global variable.)

Task-2

Newline, carriage return, and tab characters are not displayed, but they instruct the display function to perform specific operations. The newline character moves the cursor to the next row in the same column, while carriage return moves the cursor to the first column of the current row. Similarly, the tab character inserts eight spaces.

Add the functionality of newline, cret, and tab to the print procedure you created in task 1. (Hint: See ASCII code of these characters from ASCII table)

Task-3

There are 25 rows from 0 to 24 and 80 columns in a row from 0 to 79. When a column address reaches 80, it is reset to 0, and the row is incremented by 1. When the row address reaches 25, the screen moves up one line, called "scrolling up."

Incorporate the functionality of scroll up to the print procedure you created in task 1.

The C++ code for scroll up is given below:

```
for(i=0;i<24;i++)
    for(j=0;j<80;j++)
        screen[i][j] = screen[i+1][j];

for(j=0;j<80;j++)
    screen[24][j] = " ";
```