Lab 7

Stack & Procedures

Objectives

After completing this lab,

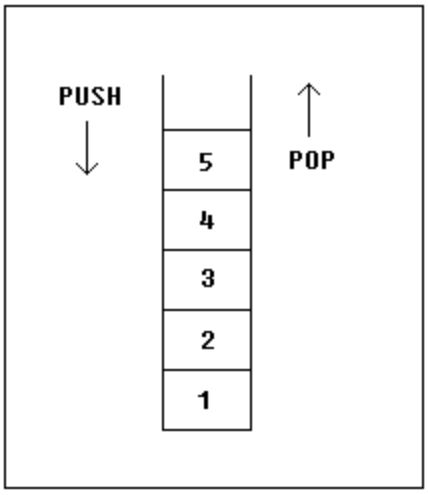
- Students will be able to use real-time Stack using push and pop instructions.
- Students will be able to access Stack directly without using push and pop instructions.
- Students will be able to use Stack for various applications.
- Students will be able to define procedures, pass parameters to them, and return values from them.

The Stack

A stack is a Last-In-First-Out (LIFO) list that is attached to a program when it is loaded into memory for execution. This stack is managed by the hardware. The Stack Segment (SS) register points to the base of the stack, and the Stack Pointer (SP) register points to the top of the stack (TOS).

In 8086, a 16-bit register, or variable is pushed on the stack using PUSH instructions. The POP instruction, on the other hand, removes the values pointed to by SP in some register.

If we push 1, 2, 3, 4, 5 one by one into the stack, the first value that we will get on pop will be 5, then 4, 3, 2, and then 1 as shown in the following figure.



The instructions that push the operand to stack and pop it from stack are shown in the table below.

Instruction	Operation	Description
PUSH source	▪ SP ← SP – 2 ▪ SS:[SP] ← source	Source: ▪ All general-purpose register, ▪ All segment registers. ▪ Variables ▪ Immediate Value ▪ IP register cannot be pushed
POP destination	▪ Destination ← SS:[SP] ▪ SP ← SP + 2	Destination: ▪ All general-purpose register, ▪ All segment registers except CS & IP ▪ Word-type Variables

The following instructions push and pop flag registers into the stack.

Instruction	Operation	Description
PUSHF	▪ SP ← SP – 2 ▪ SS:[SP] ← Flag register	▪ Stores flag registers on top of the stack
POPF	▪ Flag register ← SS:[SP] ▪ SP ← SP + 2	▪ Mov value at top of the stack to flag register ▪

Applications of Stack

Reusing registers simultaneously

There are a few registers in a processor to operate. Therefore, to use a register for another purpose, the contents of the register are temporarily pushed to the stack and popped later.

- Store the original value of the register in a stack (using PUSH).
- Use the register for any purpose.
- Restore the original value of the register from the stack (using POP).

Storing Return Address

When a function is called, the return address is pushed to the stack.

Declaring local variables

Global variables are declared and defined in data segments. However, local variables are stored on stack.

Accessing stack without using pop instructions

The Stack Segment can be accessed directly without using the POP instruction. We know that the stack segment is accessed when we use the BP register in register indirect addressing, i.e., [BP]. The contents of the SP register are moved to the BP register to gain access to the top of the stack. And then, using register indirect mode, the value at the top of the stack can be read or written.

Example: Accessing top of the stack

```
Mov BP, SP
MOV AX, [BP]
```

Example: Accessing the second value from the top of the stack

```
Mov BP, SP
MOV AX, [BP+2]
```

Example: Accessing the third value from the top of the stack

```
Mov BP, SP
MOV AX, [BP+4]
```

Procedures

A procedure is a part of code that can be called from your program to perform some specific tasks. Procedures make programs easier to understand. Generally, the procedure returns to the same point from where it was called.

The syntax for a procedure declaration is:

```
name PROC

    ; here goes the code
    ; of the procedure

RET
name ENDP
```

name- is the procedure name. The same name should appear at the top and bottom. This is used to ensure that procedures are properly closed.

RET- is requires at the end of the procedure to return the program control back to where it came from.

The following tables shows the instructions that are used to call a procedure and return control back.

Instruction	Operation	Description
Call label/ offset (2 bytes)	Push IP IP ← Label	Calling procedure that is defined in the current code segment
Call label/ segment:offset (4 bytes)	Push CS Push IP IP ← offset CS← segment	Calling procedure that is defined outside the current code segment
Ret	Pop IP	Returns control from procedure defined in the current code segment.
Retf	POP IP POP CS	Returns control from procedure defined outside the current code segment.

So far, we have not defined any procedure in a code segment. However, there should be a procedure inside the code segment, just as there is a main function in C/C++. From now on, we will write all code in procedures. The main function is called by an operating system when we run a program.

Program#1: Program that defines the main function in a program.

```
.model small

.stack

.data


.code
Main proc

Mov ax,@data
Mov ds,ax



.exit
Main endp
```

Parameter and Return value to/ from Procedures.

Procedures can be given data to process, called parameters. These parameters can be passed via registers or through the stack. However, the procedures return values through the AL or AX register.

Program#2: Program that defines the procedure “addition” to add two numbers, passed through registers, and return their sum.

```
.model small

.stack

.data


.code
Main proc                ;Defining main procedure

Mov ax,@data
Mov ds,ax

Call addition            ;Calling procedure

.exit
Main endp                ;main procedure ends here


Addition proc            ;defining procedure

Add ax,bx                ;adding two registers


Ret                      ;return control back to calling procedure
Addition endp
```

Program#3: Program that defines the procedure “addition” to add two numbers, passed through stack, and return their sum.

```
.model small

.stack

.data


.code
Main proc

Mov ax,@data
Mov ds,ax

Mov ax,5
Mov bx,10
Push ax                 ;pushing value of ax, which is 5 to stack
Push bx                 ;pushing value of bx, which is 10 to stack

Call addition           ; calling procedure

Pop bx                  ; removing number 10 from stack
Pop bx                  ; removing number 5 from stack

.exit
Main endp


Addition proc           ;defining procedure

Push bp                 ;storing value of BP on stack so that we can restore it later

Mov bp,sp               ;to access stack without pop instruction—moving TOS to bp
Mov ax,0                ;no need to push AX as it is safe to use AX register

Add ax,[bp+4]           ; [bp+4] contains value 10
Add ax,[bp+6]           ; [bp+6] contains value 5

Pop bp                  ; restoring value of BP from stack

Ret                     ; transferring control back to the calling procedure
Addition endp
```

Program#4: Program that defines the procedure “addition” to sum up an array of 5 elements and return its sum. Note: Array is always passed by reference.

```
.model small
.data
array db 1,2,3,4,5
.code
Main proc

Mov ax,@data
Mov ds,ax

Mov bx,offset array ;base address of array
mov ax,5            ;size of array

Push bx
Push ax

Call addition       ;calling procedure

pop bx              ;removing parameters from stack
pop bx              ;removing parameters from stack

.exit
Main endp


Addition proc  ;Defining procedure

push bp     ;saving the value of BP on stack before using it
push cx     ;saving the value of CX on stack before using it
push si     ;saving the value of SI on stack before using it

mov bp,sp       ;moving top of the stack to bp
mov cx,[bp+8]   ;reading size of array
mov ax,[bp+10]  ;reading base address of array
mov bp,ax       ;moving base address to BP
mov ax,0        ; it is safe to use ax without pushing it
mov si,0        ;assigning index register with 0

l1:
Add al,ds:[bp+si]  ;accessing value 10
inc si          ;incrementing index by 1 as the array is of a byte type
loop l1
pop si          ;Restoring original value of SI from stack
pop cx          ;Restoring original value of CX from stack
pop bp          ;Restoring original value of BP from stack

ret             ;returning control back to calling procedure
Addition endp
```

Program#5: Program that creates local variables.

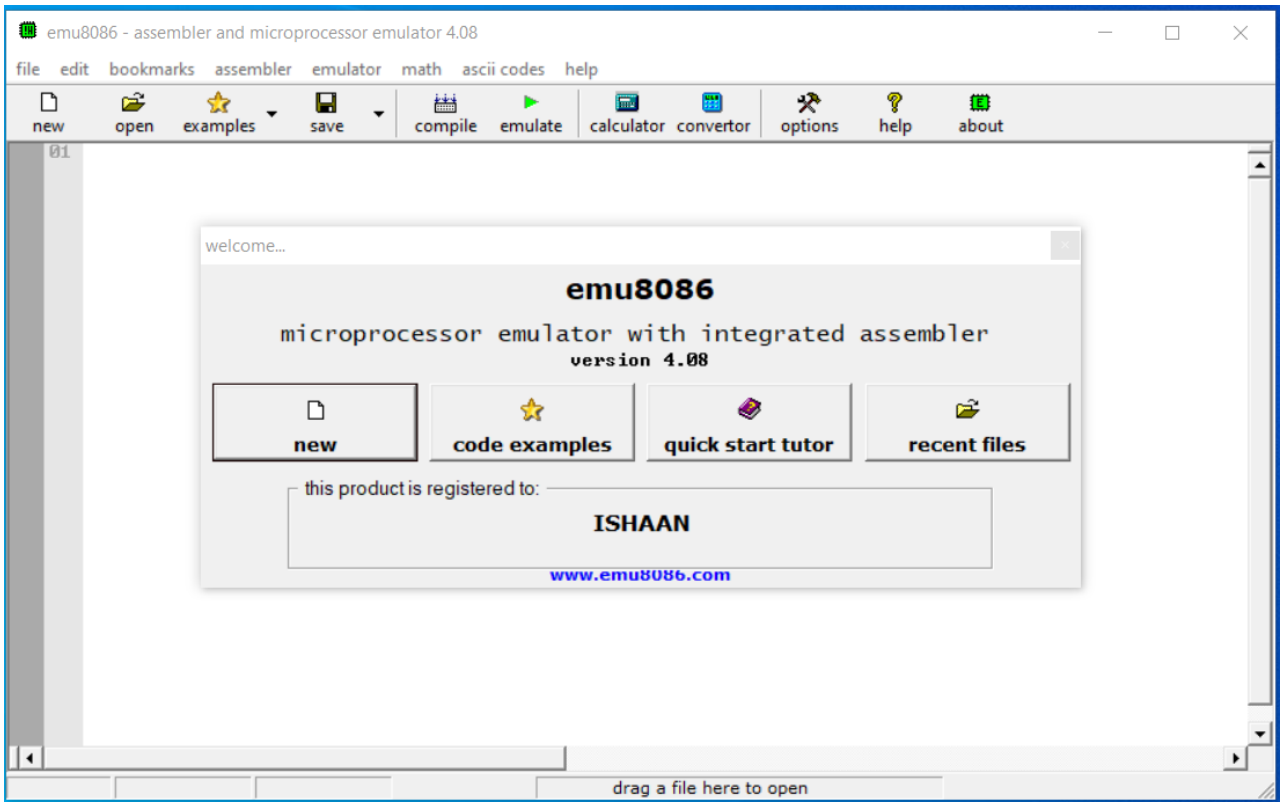
```
.model small

.data

.code

Main proc

        push    bp                      ; storing value of bp on stack
        mov     bp,sp
        sub     sp,4                    ;creating space on stack for two variables

        mov     word ptr [bp-2],5       ; assigning value to local variable
        mov     word ptr [bp-2],10      ; assigning value to local variable

       ; now you may PUSH and POP equal number of times here


        mov     sp,bp                   ;destroying local variables before returning
        pop     bp                      ; restoring value of bp from stack
        ret



.exit

Main endp
```

Emu8086 Tutorial Step by Step

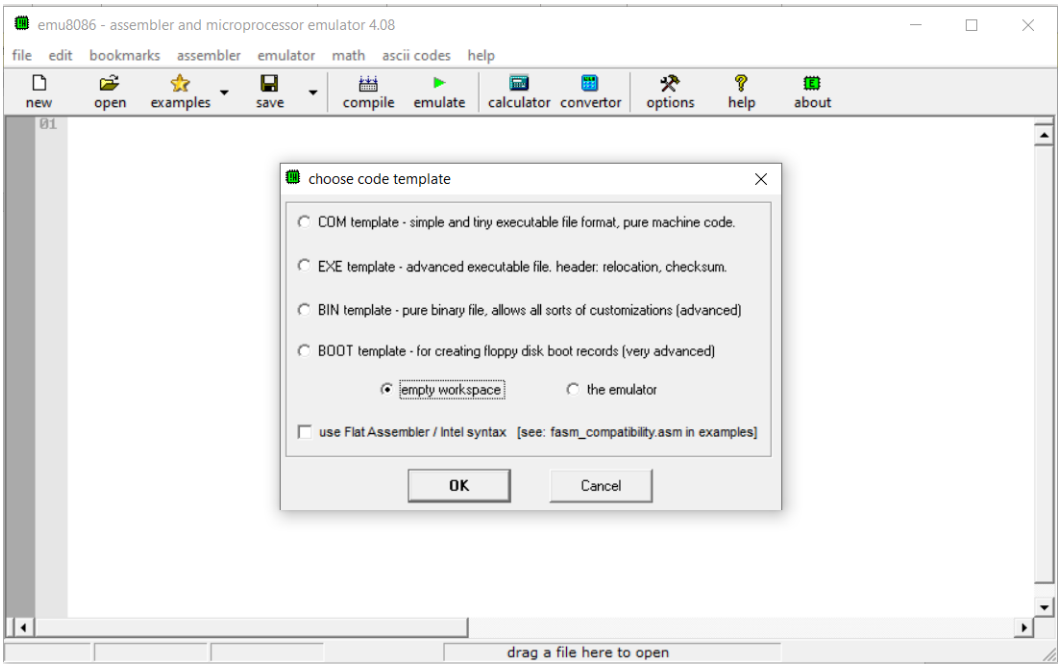
Step-1:

Double click on the icon on the desktop 

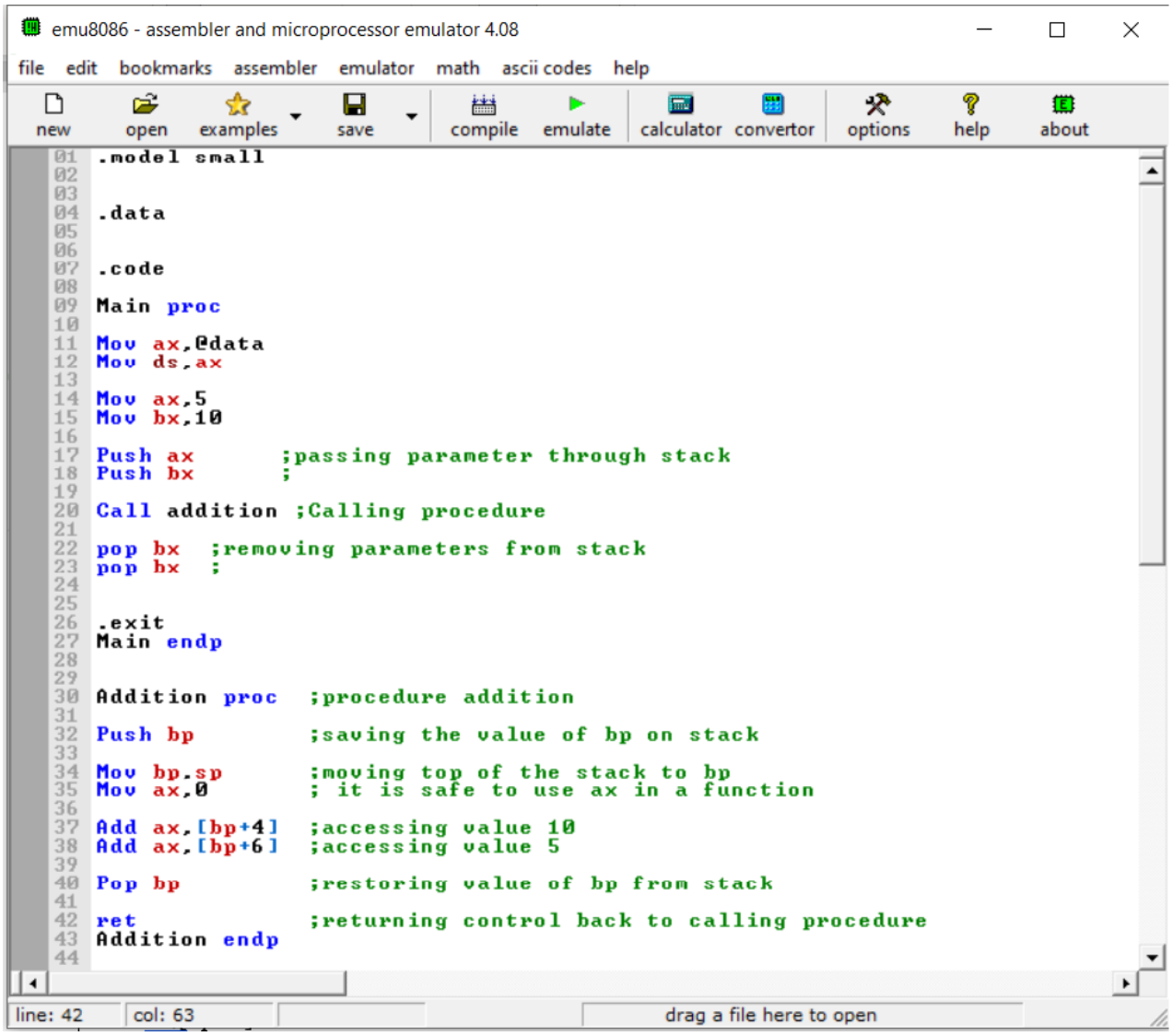
Step-2:
The following window will appear. Click on new.



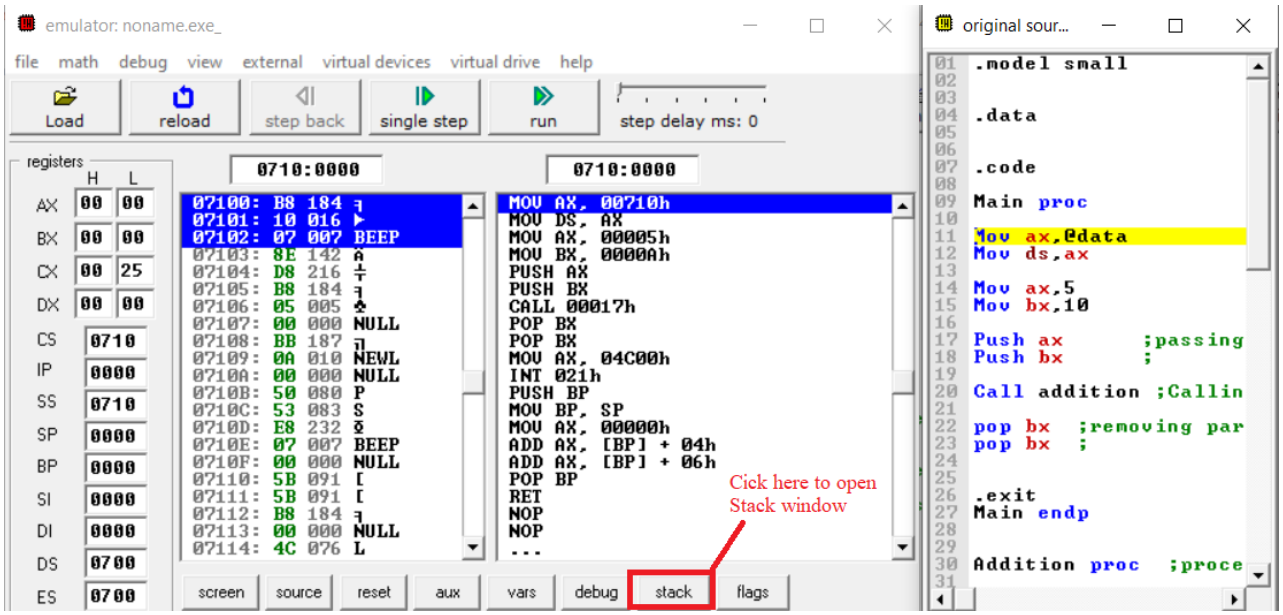
Step-3:
Click on empty workspace and press OK.



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

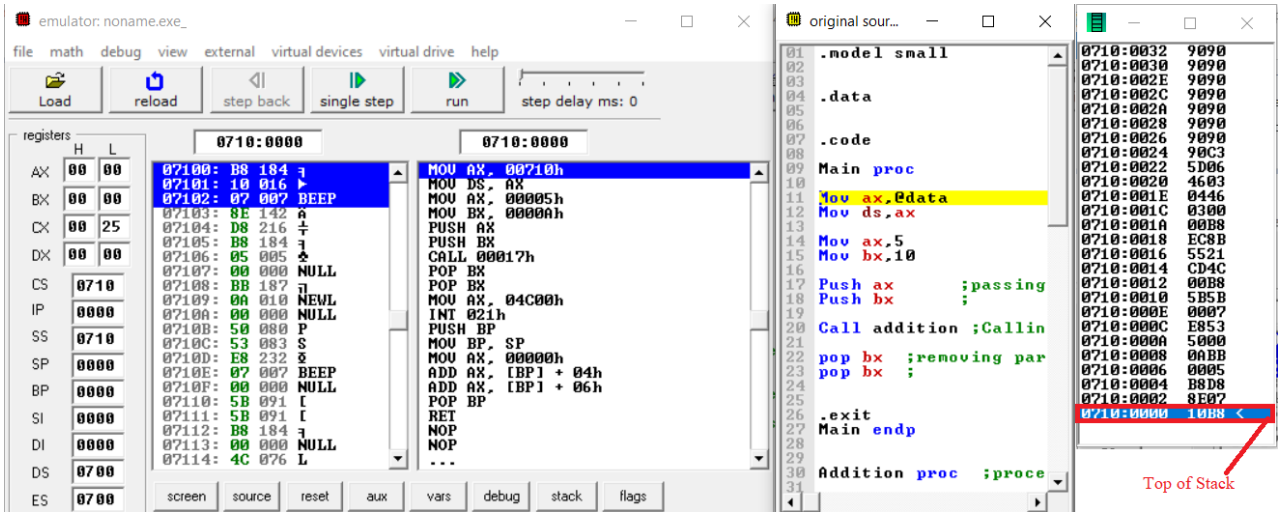


Step-5:
Click on the Stack to view stack window.



Step-6:

Keep clicking on "Single Step" to execute program instructions one by one and observe the top of the Stack side by side.



Observations

Which address is pushed to stack when a procedure is called?
At what address is control transferred when a procedure returns?
How have [BP+4] and [BP+6] been calculated for the desired locations of parameters?

Practice Exercise

Task-1

Write a program that defines the procedure "SUM." The procedure receives three arrays: A, B, and C, and the sizes of the arrays through a parameter. The procedure adds corresponding elements from arrays A and B and stores the sum in array C.

Task-2

Write a program that declares a byte array and stores an English word into it. The program then checks if the array contains a palindrome. It stores 1 in the DL register in the case of palindromes and 0 otherwise.