Microprocessor and Interfacing LAB #10

Implementation of Stack Manipulation Instructions using EMU8086

During interrupt and subroutine operations, the contents of specific internal registers of the 8086 may be overwritten. If these registers contain data that are needed after the return, they should be PUSHed to a section of memory known as the Stack. Here they may be maintained temporarily. At the completion of the service routine or subroutine, these values are POPped off the stack in the reverse order of the PUSH, and the register contents are restored with their original data.

When an interrupt occurs, the 80x86 automatically PUSHes the current flags, the value in CS, and the value in IP onto the stack. As part of the service routine for the interrupt, the contents of other registers may be pushed onto the stack by executing PUSH instructions. An example is the instruction PUSH SI. It causes the contents of the Source Index register to be PUSHed onto the stack.

At the end of the service routine, POP instructions can be included to restore the values from the stack back into their corresponding internal registers. For example, POP SI causes the value at the top of the stack to be popped back into the source index register.

PUSH AX – the contents of a 16-bit register
SP <- SP - 1 **SP is decremented**

On the other hand, if the instruction is **POP AX**, its execution results in the following:

AL <- SS: SP **; A L is POPped from the Stack**
SP <- SP + 1 **; SP is incremented**

Regardless of which method we use, we must POP in the reverse order of the PUSH. Consider the following.

```
PUSH AX
PUSH BX
PUSH CX
PUSH DX
CALL SUB1
POP DX
POP CX
POP BX
POP AX
```

Example1:

Examine the contents of stack and the value of stack pointer by executing the following example.

Source Code:

```
MOV AX,24B6
MOV DI,85C2
MOV DX,5F93
MOV SP,1236

PUSH AX
PUSH DI
PUSH DX
POP DI
POP DX
POP AX
```

Example2:

Implement the following equation in EMU8086, Ax = (A+B) x (C+D)

Source Code:

```
.DATA
A DB 3
B DB 5
C DB 7
D DB 2

.code
mov al, A
mov bl, B
Add AL, BL
PUSH AX
Mov AL, C
Mov BL, D
SUB AL, BL

POP BX
MUL BL
```

Lab Tasks

Execute the following tasks.

TASK 1:

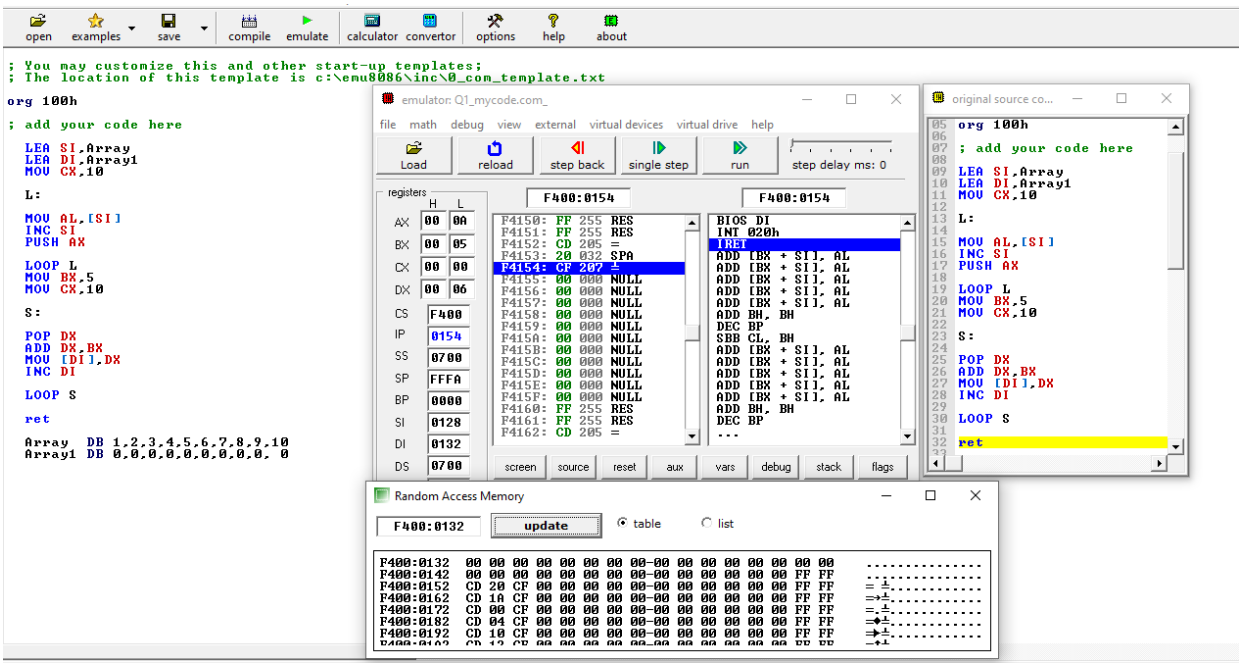
Insert numbers 1-10 in stack memory. Load register with the value 5, pop one of the item from the stack memory, add that item to the value 5 held in the register and save the result in another array/stack.

Source Code:

```
LEA SI,Array
LEA DI,Array1
MOV CX,10
```

```
L:
MOV AL,[SI]
INC SI
PUSH AX
LOOP L
MOV BX,5
MOV CX,10
S:
POP DX
ADD DX,BX
MOV [DI],DX
INC DI
LOOP S
ret
Array  DB 1,2,3,4,5,6,7,8,9,10
Array1 DB 0,0,0,0,0,0,0,0,0,0, 0
```

Output:



TASK 2:

Perform addition of numbers from 1-10 and save the result of each of the operation in stack.

hint:

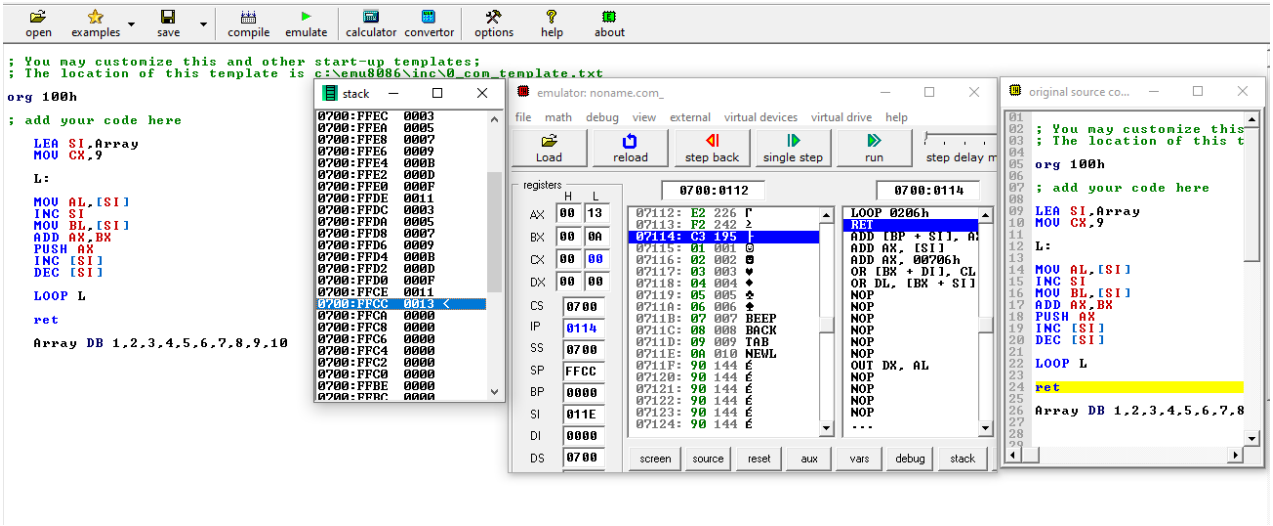
1+2= save in stack

2+3=save in stacks next location

Source Code:

```
org 100h

; add your code here

LEA SI,Array
MOV CX,9

L:

MOV AL,[SI]
INC SI
MOV BL,[SI]
ADD AX,BX
PUSH AX

INC [SI]
DEC [SI]

LOOP L

ret

Array DB 1,2,3,4,5,6,7,8,9,10
```

Output:



TASK 3:

Swap the following list using stack:

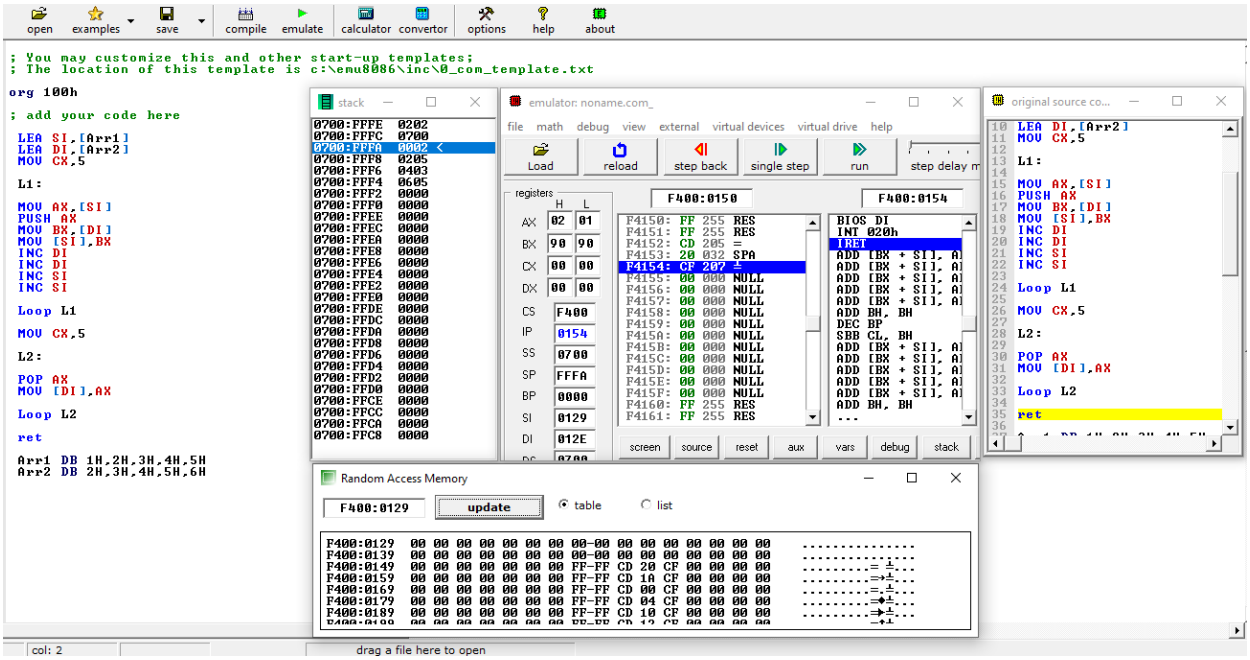
list 1	list 2
1	2
2	3
3	4
4	5
5	6

Source Code:

```
org 100h

; add your code here

 LEA SI,[Arr1]
 LEA DI,[Arr2]
 MOV CX,5
 L1:
 MOV AX,[SI]
 PUSH AX
 MOV BX,[DI]
```

```
MOV [SI],BX
INC DI
INC DI
INC SI
INC SI
Loop L1
MOV CX,5
L2:
POP AX
MOV [DI],AX
Loop L2
ret

Arr1 DB 1H,2H,3H,4H,5H
Arr2 DB 2H,3H,4H,5H,6H
```

Output:



TASK 4:

Considering the above list and add two numbers from the list, once the result is calculated, push that to stack, call the PROCEDURE FACT, perform factorial of the number and push the result of the factorial to stack as well. Perform this operation for the entire list. e.g. we added two numbers 1 and 2, the result is 3 in our case which needs to be pushed to the stack. Perform its factorial in procedure FACT and push its result to stack's next location.

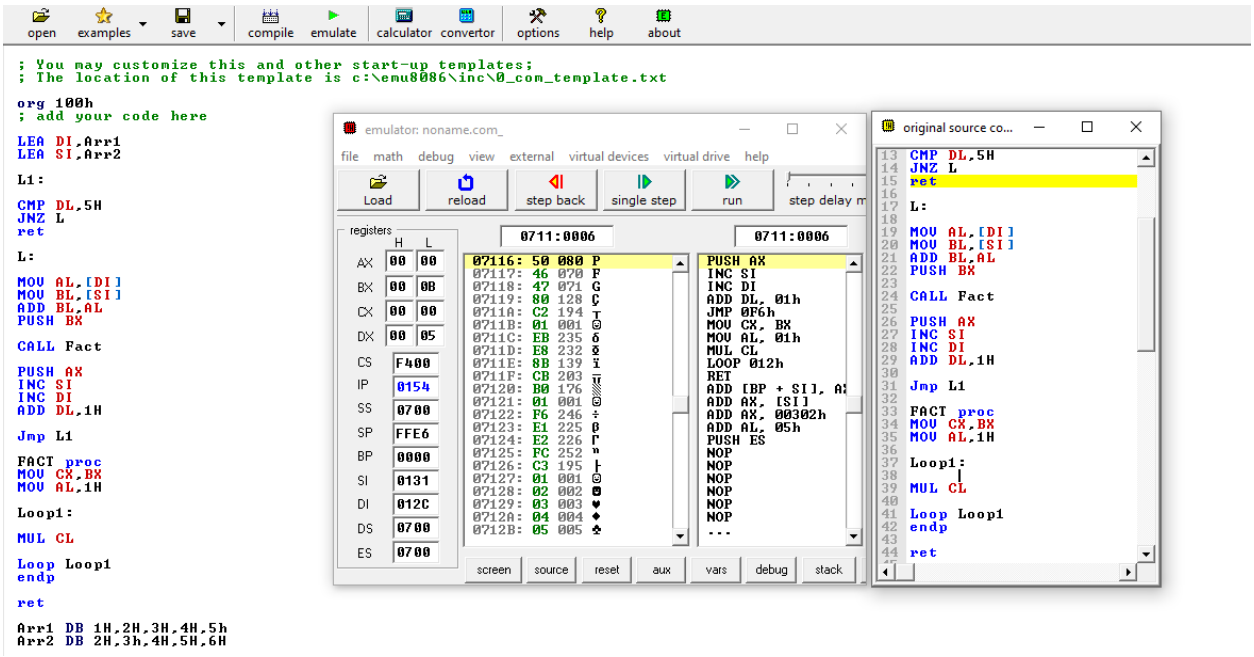
Note: In the end two consecutive stack locations will contain the sum of values and its factorial.

Source Code:

```
org 100h
; add your code here

LEA DI,Arr1
LEA SI,Arr2

L1:
CMP DL,5H
JNZ L
ret

L:
MOV AL,[DI]
MOV BL,[SI]
ADD BL,AL
PUSH BX
CALL Fact
PUSH AX
INC SI
INC DI
ADD DL,1H
```

```
Jmp L1

FACT proc
MOV CX,BX
MOV AL,1H

Loop1:
MUL CL

Loop Loop1
endp
ret

Arr1 DB 1H,2H,3H,4H,5h
Arr2 DB 2H,3h,4H,5H,6H
```

Output:



----------THE END----------