



EEE 302

Microprocessor and Interfacing

Experiment 5

Name of the experiment:  
Procedures, Stacks and Arrays in  
assembly language

Department of Electrical and Electronic  
Engineering, EWU.

**Objective:**

- To be familiar with stack operations.
- Calling and executing functions within assembly language programs.
- Arrays in assembly language programming.

**Introduction to stack:**

Stack is a segment where some register values can be stored so that it is not lost. In assembly language programming we have only four registers with which we can perform operations. These are AX, BX, CX and DX. But in many problems we may need a lot of registers. So we need to reuse these registers again and again. This can be done by storing present value of AX in stack segment. Now we can perform other tasks with AX. This will certainly change content of AX. After this getting the value of AX from stack and restoring its previous value. The first job (storing value of AX in stack) is done by PUSH command. Second job (restoring value of AX from stack) is done by POP command in assembly language. A simple program can clarify this.

MOV	AX, 3256H	;Stores 3256H in AX. This is AX's original value
PUSH	AX	;Puts AX's value in stack
MOV	AX, 1254H	;AX is assigned to an intermediate value 1254H
DEC	AX	;New value of AX is 1253H. Certainly AX's value is changed.
POP	AX	;This restores AX's previous value and now AX is 3256H

Now how stack segment is organized. Stack segment is a different segment other than default segment 0000 (in which we use to work). It is an array of memory that can store values. Obviously every memory content has an address. Each time a PUSH instruction is called, value of corresponding register is stored in stack memory. If another PUSH is called that corresponding registers value is stored in a memory that has address 2 bytes lower than previous one. All these can be shown as,

1. PUSH AX
2. PUSH BX
3. PUSH CX
4. ...
5. ...
6. ...
7. POP CX
8. POP BX
9. POP AX

Here the numbers are simply program line numbers. In line 1 AX is PUSHed. Now suppose stack starts with address value 100H. This is called Stack Pointer (SP this indicates where to put a value if something is PUSHed). Now AX's value is stored in 100H address. Stack pointer is changed to 0FEH. This means it is decremented by 2. If BX is PUSHed then it

value is stored in 0FEH and new value of SP is 0FCH. Finally if CX is PUSHed its value is stored in 0FCH. All these are shown graphically below.

At the beginning.

00F2H		
00F4H		
00F6H		
00F8H		
00FAH		
00FCH		
00FEH		
0100H		← SP

After the call, PUSH AX

00F2H		
00F4H		
00F6H		
00F8H		
00FAH		
00FCH		
00FEH		← SP
0100H	AX	

After the call, PUSH BX

00F2H		
00F4H		
00F6H		
00F8H		
00FAH		
00FCH		← SP
00FEH	BX	
0100H	AX	

After the call, PUSH CX

00F2H		
00F4H		
00F6H		
00F8H		
00FAH		← SP
00FCH	CX	
00FEH	BX	
0100H	AX	

When POP command is executed the value stored last in stack, is freed first. Here POP CX has to be called before BX or AX can be POPed. POP command automatically increases SP value by 2. So total operation of POP is opposite to PUSH.

After the call, POP CX

00F2H		
00F4H		
00F6H		
00F8H		
00FAH		
00FCH		← SP
00FEH	BX	
0100H	AX	

After the call, POP BX

00F2H		
00F4H		
00F6H		
00F8H		
00FAH		
00FCH		
00FEH		← SP
0100H	AX	

After the call, POP AX

00F2H		
00F4H		

00F6H	
00F8H	
00FAH	
00FCH	
00FEH	
0100H	

← SP

### Exercise Part 1:

**(a) Program 1:** This program stores two values in AX and BX. It finds remainder of division operation between AX and BX, and stores it to BX. AX's value need not to be altered. Clearly observe all the steps and write what happens.

```
CODE SEGMENT
    ASSUME CS:CODE, DS:CODE
    MOV AX, 3256H
    MOV BX, 125AH
    PUSH AX

LEV:  SUB AX, BX
      CMP AX, BX
      JG  LEV
      MOV BX, AX
      POP AX
      HLT

CODE ENDS
      END
```

**(b) Program 2:** Here BX is added with AX for CX times. It is desired that values of AX, BX, CX will be regained after the complete operation. Clearly observe the program and state what happens in every line.

```
CODE SEGMENT
    ASSUME CS:CODE, DS:CODE
    MOV AX, 42A6H
    MOV BX, 2E5AH
    MOV CX, 12H
    PUSH AX
    PUSH BX
    PUSH CX
```

```

LEV:  ADD AX,BX
      LOOP LEV
      POP CX
      POP BX
      POP AX
      HLT

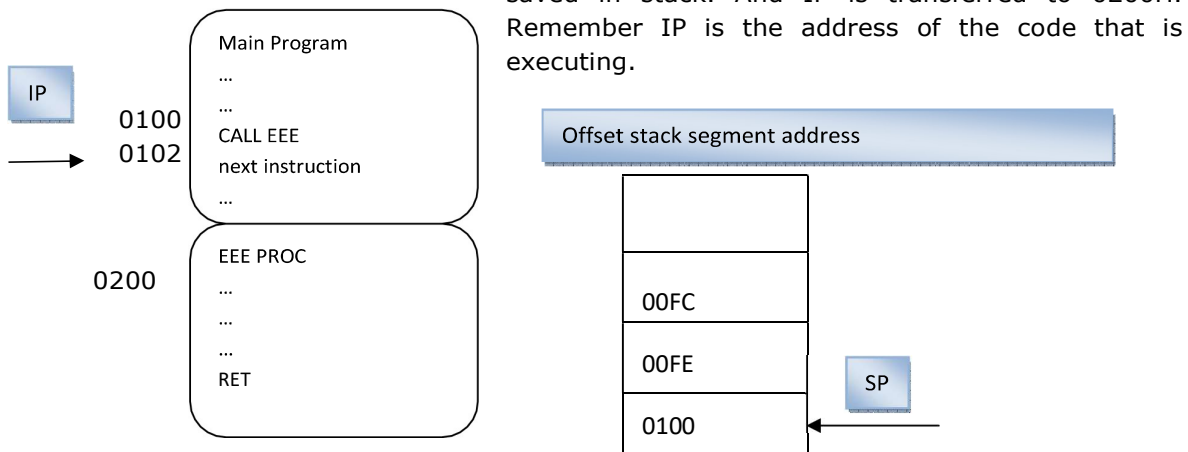
CODE ENDS
      END

```

### Introduction to CALL and RET:

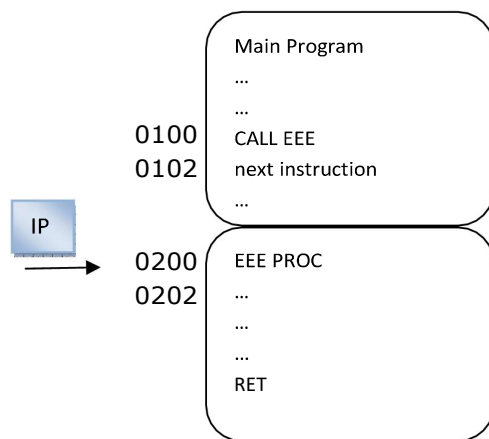
Like 'C' or 'MATLAB', in assembly language we can create functions that can be accessed from main program. CALL instruction is needed to access a function. Each function is ended with a RET to go back to main program. Suppose EEE is the name of a procedure (in assembly language functions are called procedures). It is called from the main program. What happens is stated step by step below,

At the instant of procedure calling,

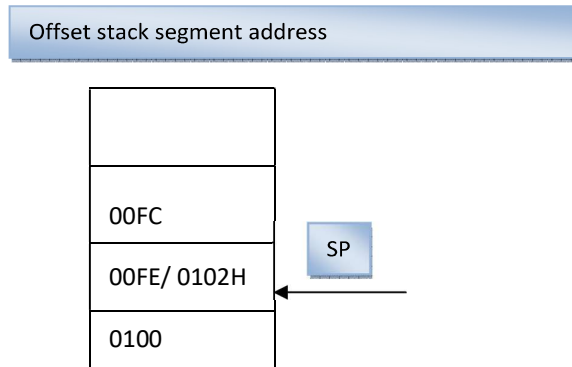


When EEE is called next instruction address 0102H is saved in stack. And IP is transferred to 0200H. Remember IP is the address of the code that is executing.

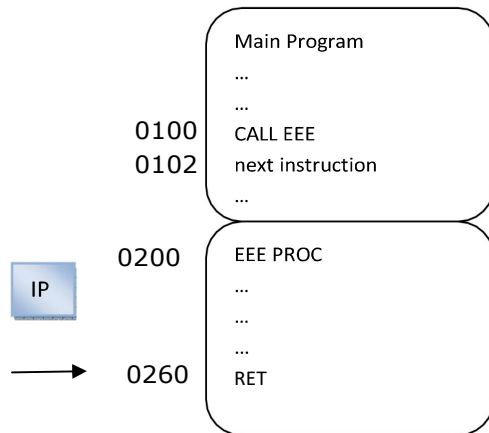
At the instant of procedure calling,



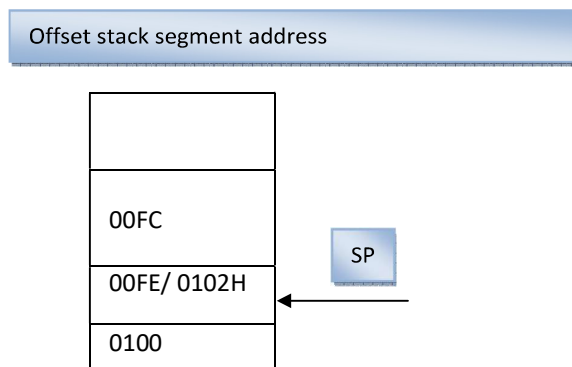
When EEE is called next instruction address 0102H is saved in stack. IP is transferred to 0202H.



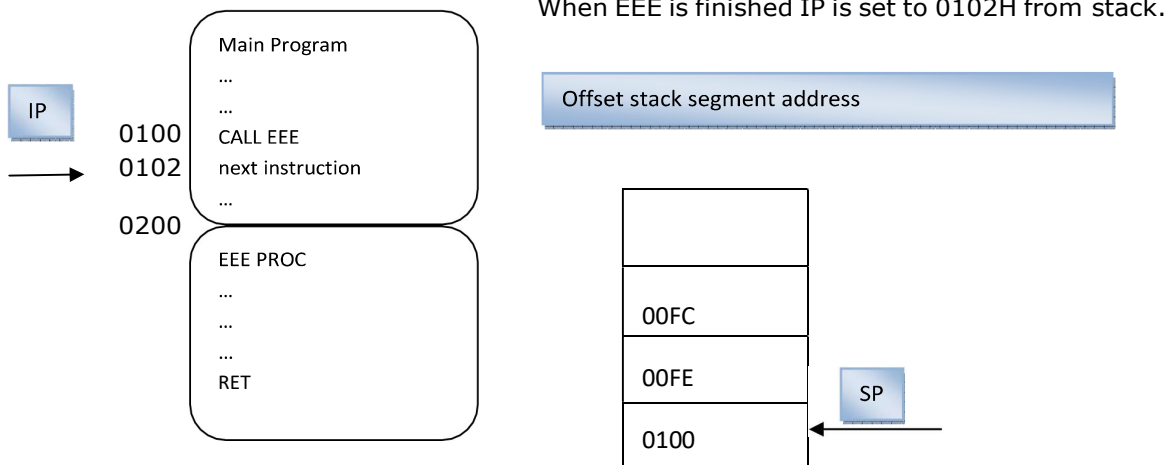
At the instant of procedure finishing,



When EEE is finishing IP is set to 0260H



At the time back in the main program,



## Exercise part 2:

**(a) Multiplication program:** Here multiplication is done by shifting and then adding a numbers. This is done as follows.

```

      1101
    X1001
    -----
      1101
     0000X
    0000XX
   1101XXX
   -----
  1110101

```

```

CODE SEGMENT
    ASSUME CS:CODE, DS:CODE

```

```

MAIN PROC
    MOV AX, 123H
    MOV BX, 16H
    CALL MULTIPLY

```

```

MULTIPLY PROC
    PUSH AX

```



```

        PUSH BX
        XOR DX, DX
REPEAT:
        TEST BX, 1
        JZ END_IF
        ADD DX, AX
END_IF:
        SHL AX, 1
        SHR BX, 1
        JNZ REPEAT

        POP BX
        POP AX

RET
MULTIPLY ENDP
END MAIN

CODE ENDS
END

```

**(b) Program 2:** This calculates GCD of three or more numbers. It uses a procedure that calculated GCD of two numbers, as discussed in the previous experiment.

```

CODE SEGMENT
    ASSUME CS: CODE, DS: CODE

MAIN PROC
    MOV AX, 9H
    MOV BX, 5H
    MOV CX, 4H
    MOV DX, 3H
    CALL GCD
    MOV BX, CX
    XCHG AX, BX
    CALL GCD
    MOV BX, DX
    XCHG AX, BX
    CALL GCD
    HLT

GCD PROC
    PUSH CX
    PUSH DX
    MOV DX, 0H
LEV:   SUB AX, BX
    CMP AX, BX

```

```

JG LEV
MOV CX,AX
MOV AX,BX
MOV BX,CX
CMP BX,DX
JNE LEV
POP CX
POP DX
RET
GCD ENDP
END MAIN

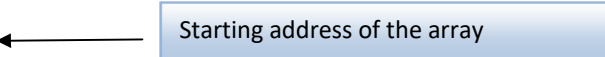
CODE ENDS
END

```

- **Introduction to Arrays in assembly language:**

Array is nothing but series of data stored in successive locations in memory. It can be showed graphically as,

Location	Content
200H	1
202H	2
204H	3
206H	4
208H	5
20AH	6
20CH	7



Simple command to insert an array in assembly language is,

```
W DW 10,20,30,40,50,60
```

Or

```
MSG DB 'abcde'
```

Here DW is data word or word array, DB is data byte or byte array. W and MSG are array names. [W] indicates the starting address. DUP command is used to create an array of same numbers. For example,

```
ALPHA DW 100 DUP(0)
```

Creates an array containing 100 0s. DUP can also be nested.

```
GAMMA DB 5,4, 3 DUP (2, 3 DUP (0), 1)
```

Creates an array of 5,4,2,0,0,0,1,2,0,0,0,1,2,0,0,0,1

Memory access for array is simple. Suppose we have W, an array of 50 elements. We want to swap its 25'th value with 31'st value. Now W indicates first element of array. W+2 second, W+4 third. So W+48 25'th and W+60 31'st. So code will be like this.

```
MOV AX, W+48
XCHG W+60, AX
MOV W+48, AX
```

**Report:**

Find Least Common Multiplier of four numbers.