Lab Manual for Computer Organization and Assembly Language

Lab-10

Stack

Table of Contents

1. Introduction

Stack is an area of memory for keeping temporary data. Stack is used by CALL instruction to keep return address for procedure, RET instruction gets this value from the stack and returns to that offset. Quite the same thing happens when INT instruction calls an interrupt, it stores in stack flag register, code segment and offset. IRET instruction is used to return from interrupt call.
We can also use the stack to keep any other data.

2. Objective

- To know more about Assembly language, such as how to work with stack structure in assembly language.

3. Concept Map

This section provides you the overview of the concepts that will be discussed and implemented in this lab.

3.1 Stack

We can define a stack using the following method

```
Stack 10h
```

Details:
There are two instructions that work with the stack:

PUSH - stores 16-bit value in the stack.
POP - gets 16-bit value from the stack.

Syntax for **PUSH** instruction:

PUSH REG
PUSH SREG
PUSH memory
PUSH immediate
REG: AX, BX, CX, DX, DI, SI, BP, SP.

SREG: DS, ES, SS, CS.

memory: [BX], [BX+SI+7], 16-bit variable, etc...

immediate: 5, -24, 3Fh, 10001101b, etc...

Syntax for **POP** instruction:

POP REG
POP SREG
POP memory
REG: AX, BX, CX, DX, DI, SI, BP, SP.

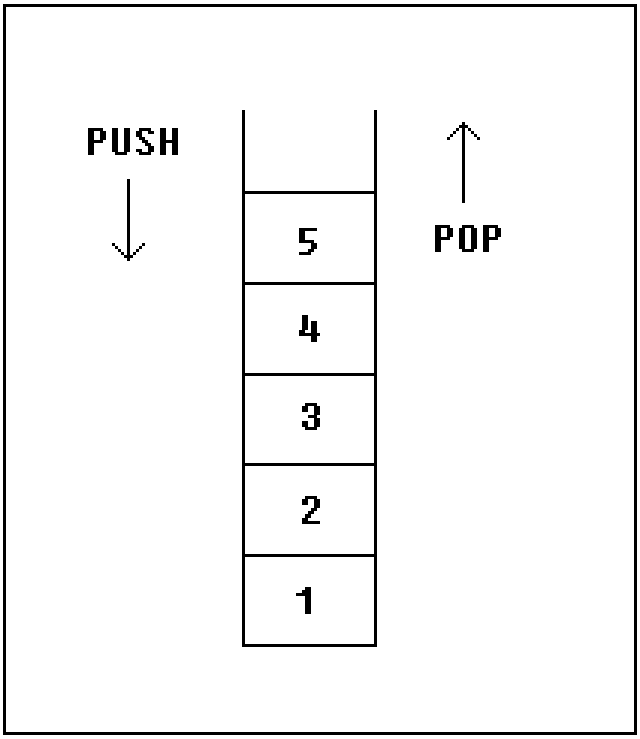
SREG: DS, ES, SS, (except CS).

memory: [BX], [BX+SI+7], 16-bit variable, etc...

Note:

- **PUSH** and **POP** work with 16-bit values only!
- **PUSH immediate** works only on 80186 CPU and later!

The stack uses **LIFO** (Last In First Out) algorithm, this means that if we push these values one by one into the stack: 1, 2, 3, 4, 5
The first value that we will get on pop will be **5**, then **4**, **3**, **2**, and only then **1**.



It is very important to do equal number of **PUSH**s and **POP**s, otherwise the stack maybe corrupted and it will be impossible to return to operating system. As you already know we use **RET** instruction to return to operating system, so when program starts there is a return address in stack (generally it's 0000h).

PUSH and **POP** instruction are especially useful because we don't have too much registers to operate with, so here is a trick:

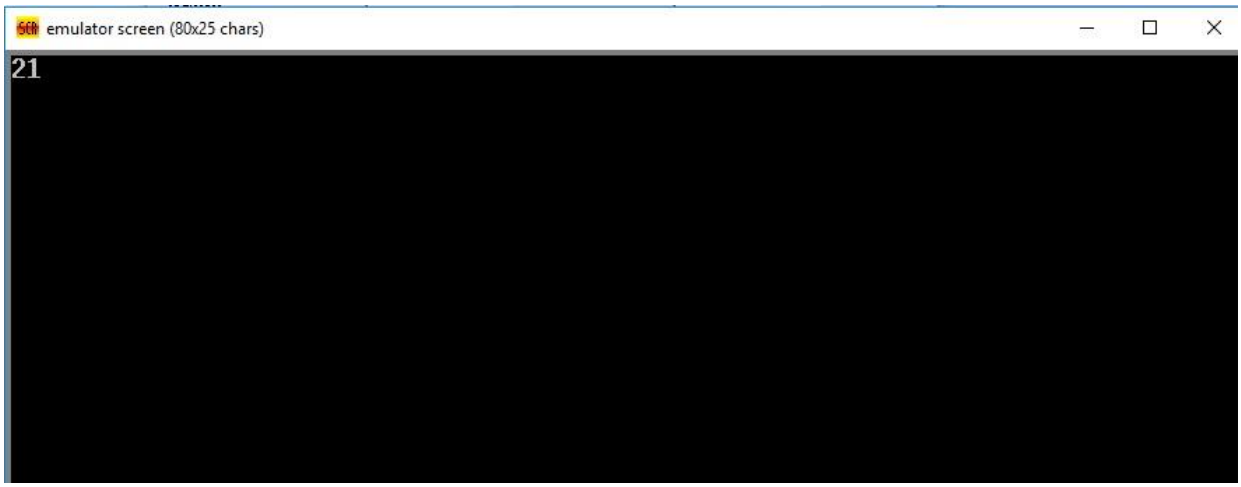
- Store original value of the register in stack (using **PUSH**).
- Use the register for any purpose.
- Restore the original value of the register from stack (using **POP**).

4. Walkthrough Task

The following program input two values in register ax and bx and push them in stack. After that the values are pop out, but the last value is pop out before the first because of the LIFO structure.

```
01 .model small
02 .data
03 .stack 10h
04 .code
05     mov ax, 1
06     mov bx, 2
07
08     push ax
09     push bx
10
11     pop ax
12     pop bx
13
14     mov dx, ax
15     add dx, 48
16     mov ah, 02
17     int 21h
18
19     mov dx, bx
20     add dx, 48
21     mov ah, 02
22     int 21h
```

The output of the program is shown below.



5. Procedure& Tools

In this section you will study how to setup and MASM Assembler.

5.1 Tools

- Download emu 8086 from (http://www.emu8086.com/files/emu8086v408r11.zip)
- Just extract the emu8086.15.zip on C
- Install emu8086

6. Practice Tasks

This section will provide more practice exercises which you need to finish during the lab. You need to finish the tasks in the required time. When you finish them, put these tasks in the following folder:

\\fs\assignments$\

6.1 Practice Task 1 [Expected time = 15mins]

Write a program to input 10 values in an array and push those values in stack. Reverse the values of array with the help of stack.

6.2 Practice Task 2 [Expected time = 15mins]

Write a program to input 10 values from user in variables. Find even values and push them in stack. Pop and display only those values which are less than 6.

6.3 Practice Task 3 [Expected time = 15mins]

Write a program to input 10 values from user in registers and use stack to handle values due to limitation of registers.

7. Out comes

After completing this lab, student will be able to understand and work with 1-Dimensional arrays in assembly language.

8. Evaluation Task (Unseen) [Expected time = 30mins for tasks]

The lab instructor will give you unseen task depending upon the progress of the class.

8.1 Evaluation criteria

The evaluation criteria for this lab will be based on the completion of the following tasks. Each task is assigned the marks percentage which will be evaluated by the instructor in the lab whether the student has finished the complete/partial task(s).

Table 3: Evaluation of Lab

Sr. No.	Task No	Description	Marks
1	4	Problem Modeling	20
2	6	Procedures and Tools	10
3	7	Practice tasks and Testing	35
4	8	Evaluation Tasks (Unseen)	20
5		Comments	5
6		Good Programming Practices	10

9. Further Reading

This section provides the references to further polish your skills.

9.1 Slides

The slides and reading material can be accessed from the folder of the class instructor available at \\fs\lectures$