BRAC UNIVERSITY

Inspiring Excellence

Department of Computer Science and Engineering

Course Code: CSE341	**Credits: 1.5**
Course Name: Microprocessors	**Semester: Fall 2018**

Lab 07

STACK

I. Topic Overview:

A stack is an array-like data structure in the memory in which data can be stored and removed from a location called the 'top' of the stack. The data that needs to be stored is 'pushed' into the stack and data to be retrieved is 'popped' out from the stack. Stack is a LIFO data structure, i.e., the data stored first is retrieved last.

The memory space reserved in the stack segment is used for implementing stack. The registers SS and ESP (or SP) are used for implementing the stack. The top of the stack, which points to the last data item inserted into the stack is pointed to by the SS:ESP register, where the SS register points to the beginning of the stack segment and the SP (or ESP) gives the offset into the stack segment.

II. Lesson Fit:

There is prerequisite to this lab. Students must have a basic idea on the following concepts:

a. Registers
b. Details of flags
c. Character encoding using ASCII
d. Interrupt in assembly

II. Learning Outcome:

After this lecture, the students will be able to:

a. Perform basic push pop operations using stack
b. Solve different problems in more effective way using stack

III. Anticipated Challenges and Possible Solutions

a. Students may get confused in push pop operations, if source of push or destination of pop is given a data byte register it will show error.

Solutions:

i. In push/pop operations data word registers are to be used..

IV. Acceptance and Evaluation

Students will show their progress as they complete each problem. They will be marked according to their class performance. There may be students who might not be able to finish all the tasks, they will submit them later and give a viva to get their performance mark. A deduction of 30% marks is applicable for late submission. The marks distribution is as follows:

Code: 50%

Viva: 50%

V. Activity Detail

1. Hour: 1

Discussion: Basics of stack

- The stack segment register holds the starting address of the stack segment in the memory. Majorly SS is used for the following purposes namely:

o To hold the temporary results.

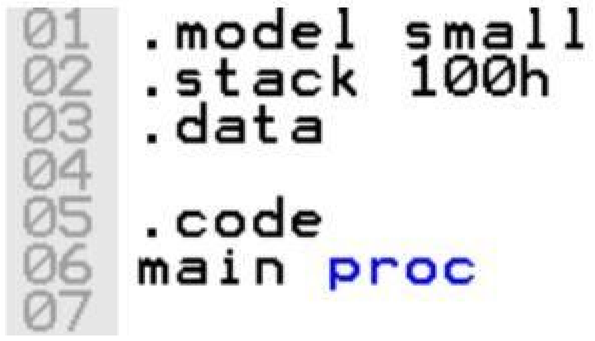
o To hold the return address of the subroutine.

o Finally to handle the interrupts

- SS (Stack Segment) Register: holds stack Segment Address
- SP (Stack Pointer) Register: initialized to the value specified with the .STACK directive, represents empty stack position
- When stack is not empty, SP represents the top of the Stack in order to save the contents of a register during the execution, so that it can be used later for purposes.
- To do so, the microprocessor has an area of memory where the contents of specified registers or memory location can be saved temporarily, and is called STACK.
- It is a top-down data structure whose elements are accessed using the pointer that is implemented using the SP and SS registers.
- As we go on storing the data words onto the stack, the pointer goes on decrementing.

Declaring a STACK:

STACK is declared at the beginning of the program:



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

.code
main proc
```

This portion of code will create a stack with 100 memory byte. In each push or pop operations the stack pointer will decrease or increase by 2.

Practise problems: Problem 1 and 2. (page 07)

2. Hour 2

Discussion: PUSH and POP OPERATIONS

There are mainly two operations **PUSH** and **PUSHF** in stack for push operations. In case

PUSH command the new value is added at the top of the stack.

PUSH: Used to add a new source (16-bit register or memory word) on stack

Syntax:

PUSH source

i.e: ***PUSH AL*** ;this push the content of AL register to STACK

Execution of PUSH causes:

a. SP is decreased by 2

b. A copy of source contents is moved to the address specified by SS:SP.

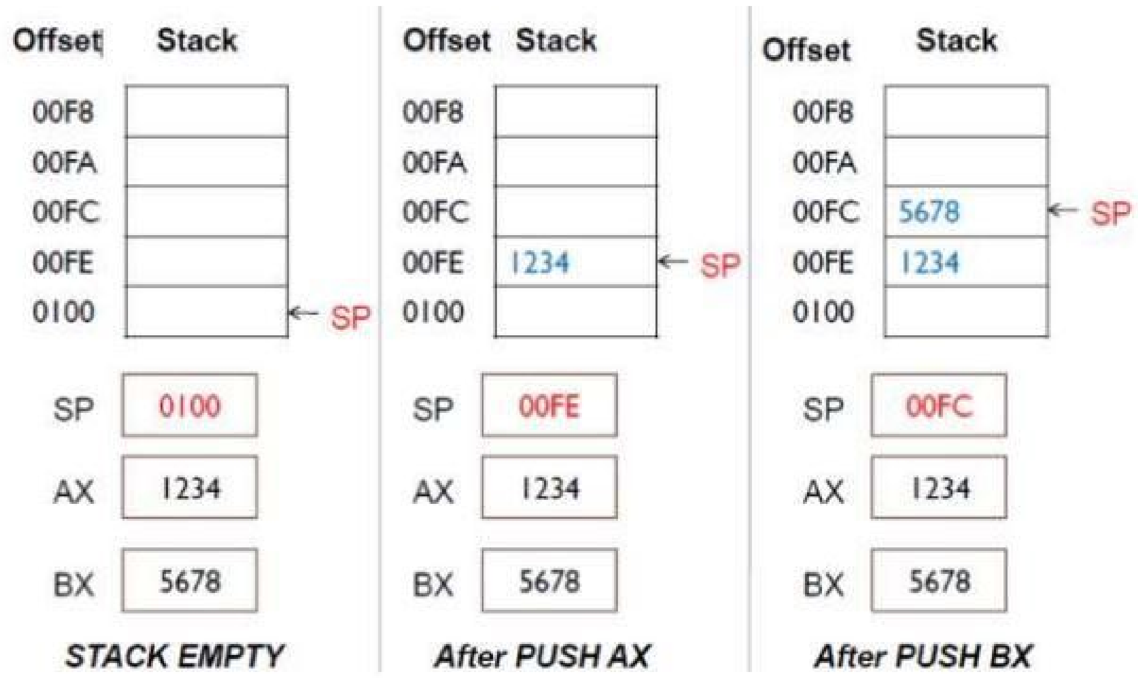
c. Source remains unchanged

PUSHF: has no operand and it pushes the contents of FLAG register onto the stack.

Usually this is done whenever the processor is interrupted

Syntax: ***PUSHF***

In case of ***PUSHF*** all the flags are stored in the stack. This is mainly used when any Procedure is called. In such case the current condition of the program is saved in to stack by **PUSHF** command. This is automatically called when any procedure or macro is called inside the program.



PUSH OPERATION IN STACK

POP OPERATIONS

There are mainly two operations **POP** and **POPF** in stack for push operations. In case POP command the top element of the stack is removed from the stack and stored in the destination register.

POP is used to remove top item from stack to destination (i.e. a 16-bit register or memory word).

Syntax:

POP destination

i.e **POP BX** ;this will pop the top element of the stack and save it to BX register

Execution of POP causes:

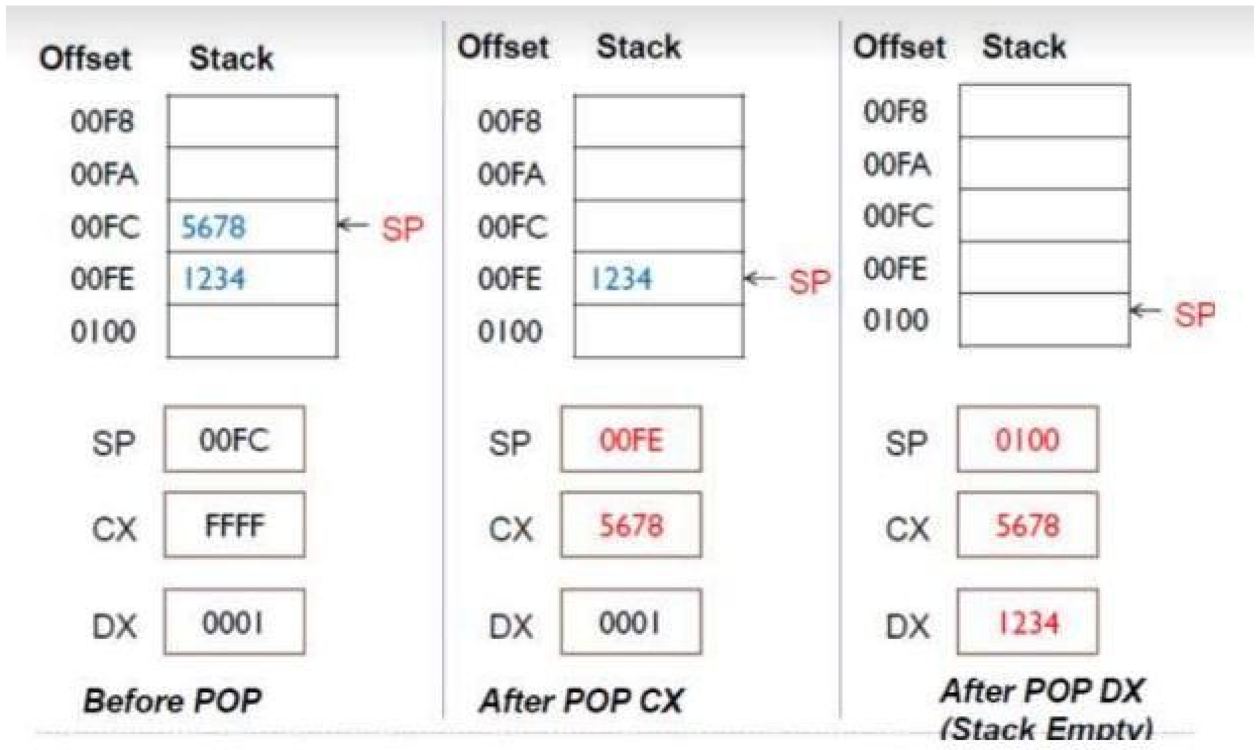
a. The contents of SS:SP (top of the stack) is moved to the destination.

b. SP is increased by 2

POPF has no operand and pops the top of the stack into FLAG register. SP is incremented by 2 after executing this instant.

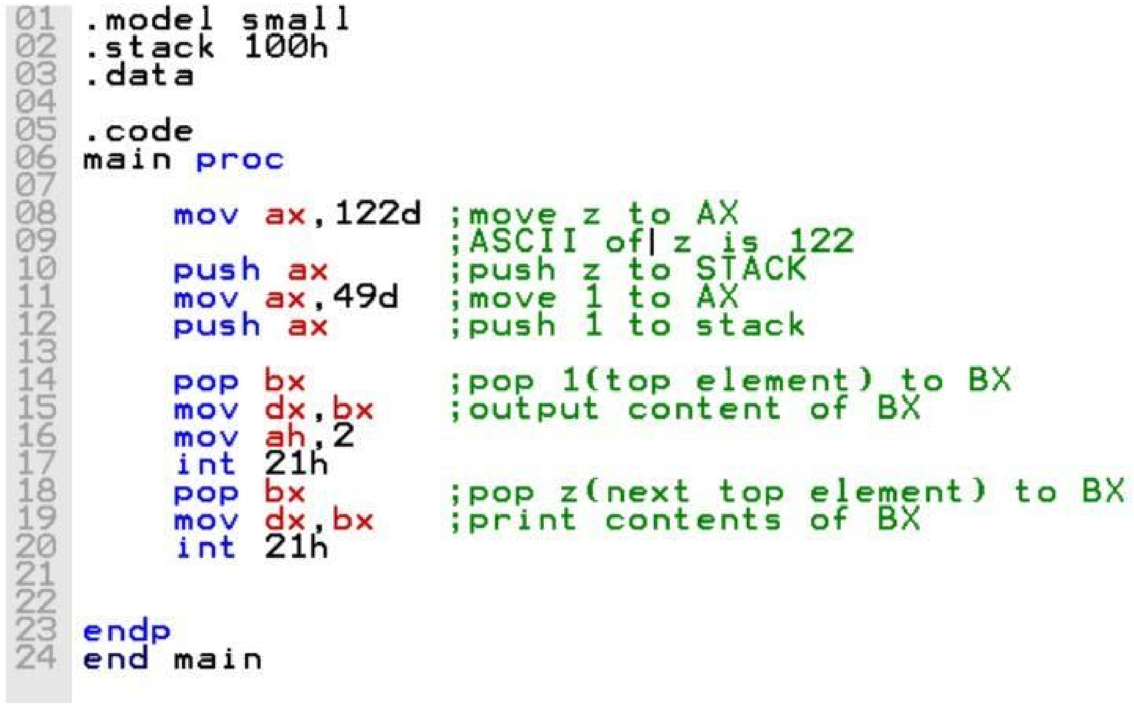
Syntax: ***POPF***

POPF is just opposite of **PUSHF.** This command restore all the flags to the flag register. Thus the previous state of the system is restored. This is automatically called while returning from a procedure or macro.



POP OPERATION IN STACK

Example code:



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

.code
main proc

     mov ax,122d   ;move z to AX
                   ;ASCII of z is 122
     push ax       ;push z to STACK
     mov ax,49d    ;move 1 to AX
     push ax       ;push 1 to stack

     pop bx        ;pop 1(top element) to BX
     mov dx,bx     ;output content of BX
     mov ah,2
     int 21h
     pop bx        ;pop z(next top element) to BX
     mov dx,bx     ;print contents of BX
     int 21h


endp
end main
```

Use of stack in function call:

STACK is internally used during any procedure call.

1. When any procedure is called, current memory location, values of all registers and local variables are pushed in stack. This is done internally.
2. When returning from a procedures, the saved elements which were pushed in stack during calling the procedure are popped from the stack. And in this way the execution can start from previous point.

When to use stacks

• Temporary save area for registers

• To save return address for CALL

• To pass arguments

• Local variables

• Applications which have LIFO nature such as Applications which have LIFO nature, such as reversing a string

HOUR 3: PROBLEM SOLVING

PROBLEMS:

1. Declare a stack with length 100h. Declare an array with 10 elements. Now if we push all the elements of the array to the stack how the value of **SP (stack pointer)** will change? Find the values manually in your script. Then run your program step by step and find whether the values you found are correct or not.
2. Now if we pop all the elements of the stack how the value of **SP (stack pointer)** will change? Find the values manually in your script. Then run your program step by step and find whether the values you found are correct or not.
3. Input a string and check whether all the letters of the word are unique or not.
4. Take five elements in an array and print them in reverse order using stack.
5. Input a number in a register. Print the number in reverse order using stack.
6. Input a word and check whether the word is palindrome or not.

BRAC UNIVERSITY

Inspiring Excellence

Home work:

1. Input a number. Find all the factors of that number and store them in stack.
2. Input a string and reverse it using stack.
3. Input a number and check whether all the digits of that number are unique or not.
4. Input a number. Starting from 1 to that number find all the prime numbers.
5. Write a program that lets the user type some text, consisting of words separated by blanks, ending with a carriage return, and displays the text in the same word order as entered, but with the letters ·in each word reversed. For example, "this is a test" becomes "siht si a tset".