Computer Organization and Assembly Language

Lab 7	
Topic	1. Stack operations 2. Subroutine

PART 1

PUSH

PUSH decrements SP (the stack pointer) by two and then transfers a word from the source operand to the top of stack now pointed to by SP. PUSH often is used to place parameters on the stack before calling a procedure; more generally, it is the basic means of storing temporary data on the stack. For example "push ax" will push the current value of the AX register on the stack. The operation of PUSH is shown below.

```
SP ← SP - 2
[SP] ← AX
```

POP

POP transfers the word at the current top of stack (pointed to by SP) to the destination operand and then increments SP by two to point to the new top of stack. POP can be used to move temporary variables from the stack to registers or memory. Observe that the operand of PUSH is called a source operand since the data is moving to the stack from the operand, while the operand of POP is called destination since data is moving from the stack to the operand. The operation of "pop ax" is shown below.

```
AX ← [SP]
SP ← SP + 2
```

CALL

CALL activates an out-of-line procedure, saving information on the stack to permit a RET (return) instruction in the procedure to transfer control back to the instruction following the CALL. For an intra segment direct CALL, SP is decremented by two and IP is pushed onto the stack. The target procedure's relative displacement from the CALL instruction is then added to the instruction pointer.

RET

RET (Return) transfers control from a procedure back to the instruction following the CALL that activated the procedure. RET pops the word at the top of the stack (pointed to by register SP) into the instruction pointer and increments SP by two. If RET is used the word at the top of the stack is popped into the IP register and SP is incremented by two. If an optional pop value has been specified, RET adds that value to SP. This feature may be used to discard parameters pushed onto the stack before the execution of the CALL instruction.

Stack Example:

ADD Two Numbers that are pushed in stack with POP.

```
;initializing
MOV AX, 55495
MOV BX, 9876
;pushing values on stack
PUSH AX
PUSH BX
;pop values from stack and observe that values of ax and bx are now swaped
Pop ax
Pop bx
Add ax,bx
jc carry_occur
jmp exit
carry_occur:
exit:
mov ax,0x4c00
int 21h
```

ADD Two Numbers that are pushed in stack without POP

```
MOV AX, 5
MOV BX, 7
PUSH AX
PUSH BX
MOV BP, SP ; SP CURRENT ADDREESS IS STORED IN BP
MOV AX, [BP]
add AX, [BP+2]
mov ax,0x4c00
int 21h
```

Subroutine Example 1:

Add all the elements of array and save the result in ax.

```
 [org 0x100]

JMP START

Array: DW 1,2,3,4,5,6,7,8,9,10

Count: dw 10

Result: dw 0

MYFUNCTION:

MOV BP,SP ; TOP OF THE STACK WILL HAVE RETURNING ADDRESS.

MOV DI,Array; DI=address of array

MOV CX,[Count] ;CX=10

Mov ax,0

L1:

Add ax,[DI]

Add di,2

LOOP L1

RET

START:

CALL MYFUNCTION  ; CALLING THE FUNCTION

Mov [Result],ax

mov ax,0x4c00

int 21h
```

Subroutine Example 2:

```
[org 0x0100]

jmp start

arr: db 'calculate the size of the string',0  ;adding a 0 as a null to the end of string

size: dw 0

calculate_size:

mov bx,arr

Continue:

cmp byte [bx],0 ;comparing null in a string in order to calculate size.

jne add_size

je exit

add_size:

inc bx   ;for next index of array

inc cx

jmp Continue

exit:

mov [size],cx

ret

start:

mov cx,0

call calculate_size

mov dx,[size]


mov ax, 0x4c00

int 0x21
```