University of Central Punjab
(Incorporated by Ordinance No. XXIV of 2002 promulgated by Government of the Punjab)
FACULTY OF INFORMATION TECHNOLOGY

Computer Organization and Assembly Language

Lab 07	
Topic	1. Stack operations 2. Implementation of subroutine

PART 1

Observe the values of SP, IP in each code after push, pop, call and ret instructions carefully.

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 without POP

```
MOV AX, 5
MOV BX, 7
MOV CX,8
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):

USING BP (Base Pointer)

```
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,[BP+2] ; DI=address of array
MOV CX,[BP+4] ;CX=10
Mov ax,0
L1:
Add ax,[DI]
Add di,2
LOOP L1
RET
START:
Push word [Count]
Mov bx,Array
PUSH bx ; PUSHING address in stack
CALL MYFUNCTION  ; CALLING THE FUNCTION
Mov [Result],ax
mov ax,0x4c00
int 21h
```

Subroutine Example(2):

WITHOUT USING BP (Base Pointer)

```
JMP START
Array: DW 1,2,3,4,5,6,7,8,9,10
Count: db 10
Result: dw 0
MYFUNCTION:
POP SI ; TOP OF THE STACK WILL HAVE RETURNING ADDRESS we have saved in a register :
POP DI ; DI=address of array
POP CX;CX=10
Mov ax,0
L1:
Add ax,[DI]
Add di,2
LOOP L1
PUSH SI ;pushing the IP value back into the stack which was pushed by the
        ;"Call" instruction that was saved by
    ;us in the SI register earlier.
RET ;now the SP is pointing to the IP value of line i.e mov[result],ax….
    ;ret updates the IP register and code continues.
START:
Push word [Count]
Mov bx,Array
PUSH bx ; PUSHING address in stack
CALL MYFUNCTION  ; CALLING THE FUNCTION
Mov [Result],ax
mov ax,0x4c00
int 21h
```

Simple Reading a character ascii example:

```
jmp start
sampleword: db 'UCP FALL 2017'
character: db 0
start:
mov bx,sampleword

mov al,[bx]     ;55 in hex is the ascii of 'U'
mov cl,[bx+12] ;37 in hex is the ascii of '7'
mov [character],cl

mov ax,0x4c00
int 21h
```

Subroutine Example(3):

```
[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:
pusha           ;=> pusha means push all registers values on stack to keep their values
                ;same after coming back from function.
xor ax,ax       ;=> clear all General purpose registers before commencing coding to
                ;remove garbage values.
xor bx,bx
xor cx,cx
xor dx,dx

mov bp,sp
add bp,16   ;=> 16 bytes is consumed by pusha so our parameter is way too back from sp.
add bp,2    ;=> 2 bytes(1 word) is the returning address of the line number 37
            ;pushed by call instruction.

mov bx,[bp] ;address at bp location is an address of array passed as a parameter.
mov cx,0


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
popa
ret

start:
mov bx,arr
push bx         ;parameter passing on stack
call calculate_size
mov dx,[size]

mov ax, 0x4c00
int 0x21
```

University of Central Punjab
(Incorporated by Ordinance No. XXIV of 2002 promulgated by Government of the Punjab)
FACULTY OF INFORMATION TECHNOLOGY

****Practice Tasks****

Task 1:
Write an assembly language program to add the values of ax,bx,cx and dx registers using a subroutine. Let AX=65,BX=0x9,CX=42,DX=60h. save the sum in AX.

Task 2:
Write a subroutine that takes address of an array of characters “I am a programmer” as parameter and counts vowels. Save the count in cx.

Task 3:
Write an assembly language program to invert the case of all the alphabets only within a string.
For example:
Array1: aB3C7d&E
After function calling….
Array1: Ab3c7D&e