LAB 10

Advanced Procedures

STUDENT NAME _____ ROLL NO _____ SEC ___

LAB ENGINEER'S SIGNATURE & DATE _____

MARKS AWARDED: /

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES (NUCES), KARACHI

Prepared by: Qurat ul ain

Lab Session 10: Advanced Procedures

Learning Objectives

- Implementing procedures using stack frame
- Using stack parameters in procedures
- Passing value type and reference type parameters

Stack Applications

There are several important uses of runtime stacks in programs:

1. A stack makes a convenient temporary save area for registers when they are used for more than one purpose. After they are modified, they *can* be restored to their originalvalues.
2. When the CALL instruction executes, the CPU saves the current subroutine's return address on the stack.
3. When calling a subroutine, you pass input values called arguments by pushing them on the stack.
4. The stack provides temporary storage for local variables inside subroutines.

Stack Parameters

- **Passing by value**
  When an argument is passed by value, a copy of the value is pushed on the stack.

EXAMPLE # 01:

```
.data
var1    DWORD       5
var2    DWORD       6
.code
push var2
push var1
call AddTwo
exit
AddTwo PROC
push    ebp
mov     ebp, esp
mov     eax, [ebp + 12]
add     eax, [ebp + 8]
pop     ebp
ret
AddTwo ENDP
```

- **Explicit stack parameters**

  When stack parameters are referenced with expressions such as [ebp+8], we call themexplicit stack parameters.

Example 2:

```
.data
var1    DWORD       5
var2    DWORD       6
y_param     EQU     [ebp + 12]
x_param     EQU     [ebp+ 8]
.code
push var2
push var1
call AddTwo
exit
AddTwo PROC
push ebp
mov ebp, esp
mov eax, y_param
add eax, x_param
pop ebp
ret
AddTwo ENDP
```

- **Passing by reference**

  An argument passed by reference consists of the offset of an object to be passed.

EXAMPLE # 03:

```
.data
count = 10
arr     WORD count DUP (?)
.code
push OFFSET arr
push count
call ArrayFill
exit
ArrayFill PROC
push ebp
mov ebp, esp
pushad
```

```
mov esi, [ebp + 12]
mov ecx, [ebp + 8]
cmp ecx, 0
je L2
L1:
mov    eax, 100h
call RandomRange
mov [esi], ax
add esi, TYPE WORD
loop L1
L2:
popad
pop ebp
ret 8
ArrayFill ENDP
```

LEA Instruction

LEA instruction returns the effective address of an indirect operand. Offsets of indirectoperands are calculated at runtime.

EXAMPLE # 04:

```
.code
call      makeArray
exit
makeArray      PROC
push    ebp
mov     ebp, esp
sub     esp, 32
lea     esi, [ebp - 30]
mov ecx,30
L1:
mov     BYTE PTR [esi], '*'
inc     esi
loop    L1
add     esp, 32
pop     ebp ret
makeArray      ENDP
```

ENTER & LEAVE Instructions

Enter instruction automatically creates stack frame for a called Procedure. Leave instructionreverses the effect of enter instruction.

EXAMPLE # 05:

```
.data
var1    DWORD       5
var2    DWORD       6
.code
push var2
push var1
call AddTwo
exit
AddTwo PROC
enter 0, 0
mov     eax, [ebp + 12]
add     eax, [ebp + 8]
leave
ret
AddTwo ENDP
```

Local Variables

In MASM Assembly Language, local variables are created at runtime stack, below the basepointer (EBP).

EXAMPLE # 06:

```
.code
call     MySub
exit
MySub PROC
push    ebp
mov     ebp, esp
sub     esp, 8
mov     DWORD       PTR [ebp - 4], 10       ; first parameter
mov     DWORD       PTR [ebp - 8], 20       ; second parameter
mov     esp, ebp
pop     ebp
ret
MySub ENDP
```

LOCAL Directive

LOCAL directive declares one or more local variables by name, assigning them sizeattributes.

EXAMPLE # 07:

```
.code
call LocalProc
```

```
exit
LocalProc PROC
LOCAL  temp : DWORD
mov    temp, 5
mov    eax, temp
ret
LocalProc ENDP
```

Recursive Procedures

Recursive procedures are those that call themselves to perform some task.

EXAMPLE # 08:

```
.code
L1:
mov    ecx, 5
mov    eax, 0
call   CalcSum
call   WriteDec
call   crlf
exit
CalcSum        PROC
cmp    ecx, 0
jz     L2
add    eax, ecx
dec    ecx
call   CalcSum
L2:
ret
CalcSum        ENDP
```

- **INVOKE Directive**

The INVOKE directive pushes arguments on the stack and calls a procedure. INVOKE is a convenient replacement for the CALL instruction because it lets you pass multiple argumentsusing a single line of code.

Here is the general syntax:

```
INVOKE procedureName [, argumentList]
```

For example:

```
push TYPE array
push LENGTHOF array
```

```
push OFFSET array
call DumpArray
```

is equal to

```
INVOKE DumpArray, OFFSET array, LENGTHOF array, TYPE array
```

- **ADDR Operator**

The ADDR operator can be used to pass a pointer argument when calling a procedure usingINVOKE. The following INVOKE statement, for example, passes the address of myArrayto the FillArrayprocedure:

```
INVOKE FillArray, ADDR myArray
```

- **PROC Directive**

Syntax of the PROC Directive
The PROC directive has the following basic syntax:

```
Label PROC [attributes] [USES reglist], parameter_list
```

The PROC directive permits you to declare a procedure with a comma-separated list of namedparameters.
Example: The FillArray procedure receives a pointer to an array of bytes:

```
FillArray PROC,
pArray:PTR BYTE
. . .
FillArray ENDP
```

- **PROTO Directive**

The PROTO directive creates a prototype for an existing procedure. A prototype declares a procedure's name and parameter list. It allows you to call a procedure before defining it and toverify that the number and types of arguments match the procedure definition.

```
MySub PROTO  ; procedure prototype
.
INVOKE MySub  ; procedure call
.
MySub PROC              ; procedure implementation
.
MySub ENDP
```

Exercises:

1. Write a program which contains a procedure named **BubbleSort** that sorts an array which is passed through a stack using indirect addressing.

2. Write a program which contains a procedure named **TakeInput** which takes input numbers from user and call a procedure named **Armstrong** which checks either a numberis an Armstrong number or not and display the answer on console by calling another function **Display**. (Also show ESP values during nested function calls)

3. Write a program which contains a procedure named **Reverse** that reverse the string using recursion.

4. Write a program which contains a procedure named **LocalSquare** . The procedure mustdeclare a local variable. Initialize this variable by taking an input value from the user andthen display its square. Use **ENTER *& LEAVE*** instructions to allocate and de-allocate the local variable.

5. Write a program that calculates factorial of a given number ***n***. Make a recursive procedure named **Fact** that takes n as an input parameter.

6. Write a program to take 4 input numbers from the users. Then make two procedures **CheckPrime** and **LargestPrime**. The program should first check if a given number is a prime number or not. If all of the input numbers are prime numbers then the program should call the procedure LargestPrime.

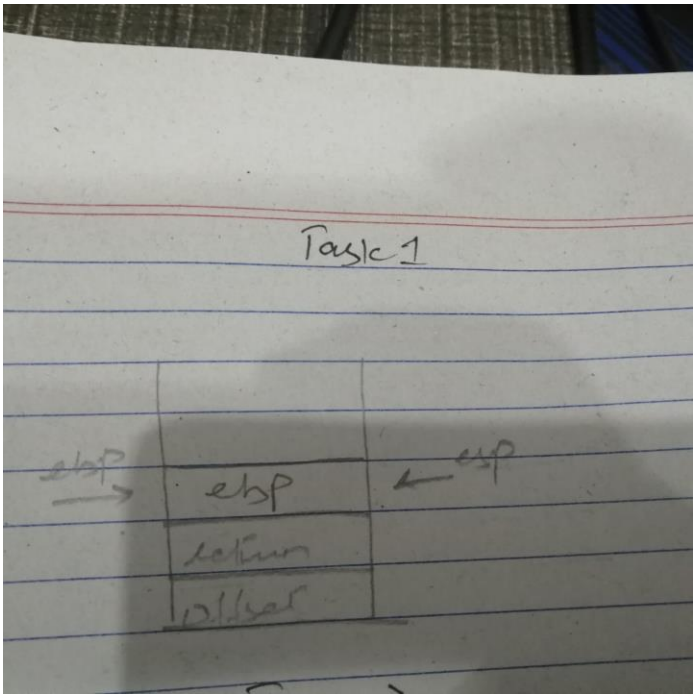
CheckPrime: This procedure tests if a number is prime or not

LargestPrime: This procedure finds and displays the largest of the four prime numbers.

TASK 1:

```
include irvine32.inc

.data
array dword 3,1,4,11,6,7
temp dword ?

.code
main proc
xor eax,eax
xor ebx,ebx
xor ecx,ecx
xor edx,edx
push offset array
call bubblesort
mov ecx,lengthof array
mov esi,offset array
l1:
    mov eax,[esi]
    call writedec
    call crlf
    add esi,4
loop l1
exit
main endp
bubblesort proc
push ebp
mov ebp,esp

mov ecx, lengthof array
dec ecx
;mov ebx,0
outerloop:
    mov esi,[ebp+8]
    mov temp,ecx
    mov ecx,lengthof array
    dec ecx
    innerloop:
        mov edx,[esi]
        mov eax,[esi+4]
        cmp edx,eax
        jb n_swap
        xchg edx,eax
        mov [esi],edx
        mov [esi+4],eax
        n_swap:
        add esi,4
        loop innerloop
    ;inc ebx
    mov ecx,temp
loop outerloop
mov esp,ebp
pop ebp
ret 4
bubblesort endp
end main
```

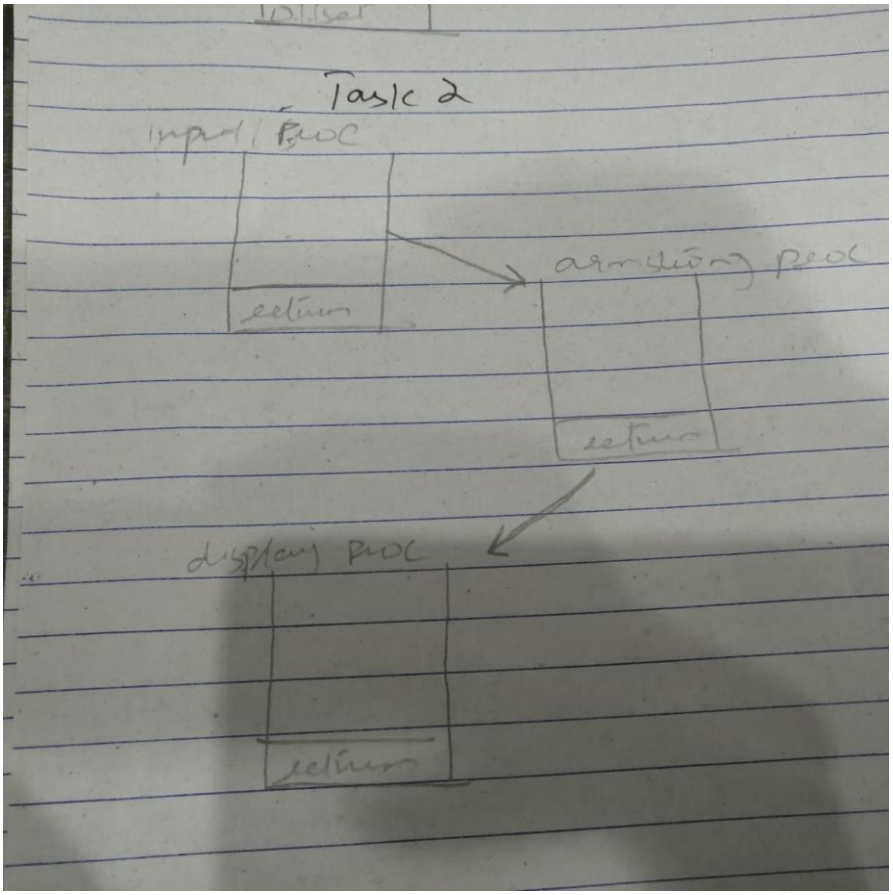
```
Microsoft Visual Studio Debug Console
1
3
4
6
7
11

C:\Users\pd\source\repos\COALmidpractice\Debug\COALmidpractice.exe (process 14504) exited with code 0.
Press any key to close this window . . .
```



TASK 2:

```
include irvine32.inc
include macros.inc
.data
n dword ?
var1 dword ?
var2 dword ?
var3 dword ?
divisor dword 10
.code
main proc
xor edx,edx
call input

exit
main endp

input proc
mwrite "Enter the number to check armstrong (the number mus
call readint
mov n,eax
call armstrong
ret
input endp

armstrong proc

div divisor
mov var1,edx
mov edx,0
div divisor
mov var2,edx
mov edx,0
div divisor
mov var3,edx
cmp eax,0
jne w_input
    mov eax,var1
    imul eax,var1
    imul eax,var1
    mov var1,eax
    mov eax,var2
    imul eax,var2
    imul eax,var2
    mov var2,eax
    mov eax,var3
    imul eax,var3
```

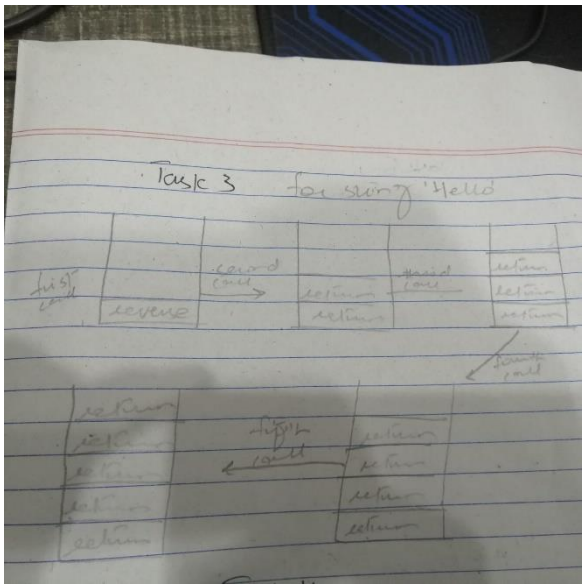
```
Microsoft Visual Studio Debug Console
Enter the number to check armstrong (the number must be 3 digits) 153
The entered number is an armstrong number
C:\Users\pd\source\repos\COALmidpractice\Debug\COALmidpractice.exe (process 5440) exited with code 0.
Press any key to close this window . . .
```



TASK 3:

```
include macros.inc
.data
str1 byte "I am programmer",0
str2 byte lengthof str1 dup(?)
.code
main proc
xor eax,eax
mov esi,0
add esi,lengthof str1
sub esi,2
mov edi,0
call reverse
mov edx,offset str2
call writestring
exit
main endp

reverse proc
cmp esi,-1
je end_proce
    mov al,str1[esi]
    mov str2[edi],al
    dec esi
    inc edi
    call reverse
end_proce:
ret
reverse endp

end main
```

```
Microsoft Visual Studio Debug Console
remmargorp ma I
C:\Users\pd\source\repos\COALmidpractice\Debug\COALmidpractice.exe (process 6064) exited with code 0.
Press any key to close this window . . .
```



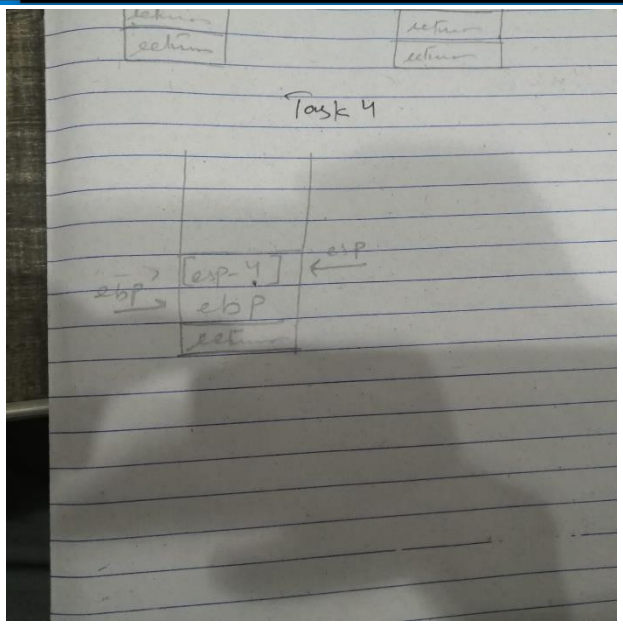
TASK 4:

```
include irvine32.inc
include macros.inc
.data

.code
main proc

xor eax,eax
call square

exit
main endp

square proc

enter 4,0

mwrite "Enter any number: "
call readint
mov [esp-4],eax
mov ebx,[esp-4]
imul ebx,ebx
mov [esp-4],ebx
mov eax,[esp-4]
call writedec

leave
ret

square endp
```

Solution Explorer

Microsoft Visual Studio Debug Console

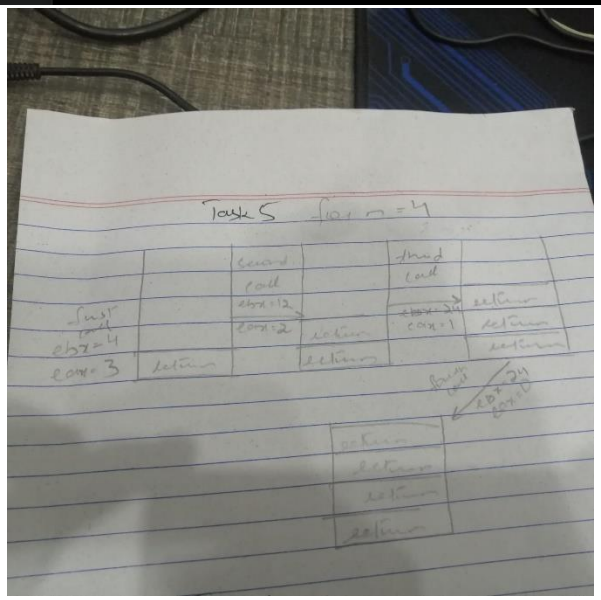
```
Enter any number: 6
36
C:\Users\pd\source\repos\COALmidpractice\Debug\COALmidpractice.exe (process 1756) exited with code 0.
Press any key to close this window . . .
```



TASK 5:

```
.code
main proc
xor ebx,ebx
mwrite "Enter the number to find factorial "
call readint
cmp eax,1
je cc
cmp eax,0
je cc
mov n,eax
dec eax
mov ebx,n
call factorial
mov eax,ebx
call writedec
exit
cc:
    mwrite "The factorial is 1"
exit
main endp

factorial proc
    cmp eax,0
    je end_prog
    imul ebx,eax
    dec eax
    call factorial
end_prog:
ret
factorial endp
```

```
Microsoft Visual Studio Debug Console
Enter the number to find factorial 6
720
C:\Users\pd\source\repos\COALmidpractice\Debug\COALmidpractice.exe (process 10808) exited with code 0.
Press any key to close this window . . .
```



TASK 6:

```
include irvine32.inc
include macros.inc
.data
array dword 3,5,31,37
max dword ?
flag byte ?
.code
main proc

xor eax,eax
mov esi,0
mov ecx,lengthof array
loop1:
    mov flag,0
    push array[esi]
    call is_prime
    cmp flag,0
    je somenp
    pop eax
    xor eax,eax
    add esi,4
loop loop1
call max_prime
jmp end_proce
somenp:
    mwrite "One of the numbers entered is not
    mov eax,array[esi]
    call writedec
end_proce:
exit

main endp

is_prime proc

xor edx,edx
mov ebx,2
L1:
```

```
Microsoft Visual Studio Debug Console
The max prime number is: 37
C:\Users\pd\source\repos\COALmidpractice\Debug\COALmidpractice.exe (process 12572) exited with code 0.
Press any key to close this window . . .
```