LAB 05

LOOP INSTRUCTION & PROCEDURES

STUDENT NAME _____ ROLL NO _____ SEC ____

SIGNATURE & DATE _____

MARKS AWARDED: _____

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

Prepared by: Aamir Ali

Version: 1.0
Date: 06th Oct 2021

EL 2003 COMP ORG & ASSEMBLY LANGUAGE LAB

LABORATORY MANUAL • Fall 2021 • Instructor: Aamir Ali, Aashir Mehboob, Amin Sadiq, Qurat-ul-Ain, M Kariz, Rabia

Lab Session 05: LOOP INSTRUCTION & PROCEDURES

Objectives:

- Loop Instruction
- Built-in-Procedure

Branching Instructions:

Branching is the most direct method of modifying the instruction flow. A transfer of control, or branch, is a way of altering the order in which statements are executed. There are two basic types of transfers:

- Unconditional
- Conditional

Unconditional Transfer:

The unconditional jump instruction (jmp) unconditionally transfers control to the instruction located at the target address i.e. there is no need to satisfy any condition for the jump to take place. The general format is:

JMP *destination*

When the CPU executes an unconditional transfer, the offset of destination is moved into the instruction pointer, causing execution to continue at the new location.

Syntax:

Label:

…….

…….

…….

JMP *Label*

Conditional Transfer:

In these types of instructions, the processor must check for the particular condition. If it is true, then only the jump takes place else the normal flow in the execution of the statements is maintained. There are many instructions for conditional jumping, that we will explore in later labs. For this lab, our focus in only on LOOP instruction.

Loop Instruction:

The LOOP instruction, formally known as Loop According to ECX Counter, repeats a block of statements a specific number of times. ECX is automatically used as a counter and is decremented each time the loop repeats.

Syntax:

LOOP *destination*

The execution of the LOOP instruction involves two steps: First, it subtracts 1 from ECX. Next, it compares ECX to zero. If ECX is not equal to zero, a jump is taken to the label identified by destination. Otherwise, if ECX equals zero, no jump takes place, and control passes to the instruction following the loop.

Syntax:

MOV ECX, #COUNT

Label:

……..

……..

```
LOOP Label
```

Example 01:

```
INCLUDE Irvine32.inc
.code
main PROC
    mov ax,0
    mov ecx,5
    L1:
        Inc ax
        call dumpregs
    loop L1
    exit
main ENDP
END main
```

Example 02:

```
INCLUDE Irvine32.inc
.data
    intArray WORD 100h, 200h, 300h, 400h, 500h
.code
main PROC
    mov esi, 0
    mov eax, 0
    mov ecx, LENGTHOF intArray
    call dumpregs
    L1:
        mov ax, intArray[esi]
        add esi, TYPE intArray
        call dumpregs
    loop L1
    exit
main ENDP
END main
```

Nested Loops

When creating a loop inside another loop, special consideration must be given to the outer loop counter in ECX. You can save it in a variable.

Syntax:

```
MOV ECX, #COUNT1
LABEL1:
    MOV VAR1, ECX
    ……….
    MOV ECX, #COUNT2
    LABEL2:
        MOV VAR2, ECX
        ……..
        MOV ECX, VAR2
    LOOP LABEL2
```

```
        ……..
        MOV ECX,VAR1
    LOOP LABEL1
```

Example 03:

```
    INCLUDE Irvine32.inc
    .code
    main PROC
    mov eax, 0
    mov ebx, 0
    mov ecx, 5
    L1:
        inc eax
        mov edx, ecx
        call dumpregs
        mov ecx, 10
        L2:
            inc ebx
            call dumpregs
        loop L2
        mov ecx, edx
    loop L1
    call DumpRegs
    exit
    main ENDP
    END main
```

Procedure in Irvine32 Library:

Some of the procedures available in Irvine32 library are:

1. **Clrscr:**
   Clears the console window and locates the cursor at the above left corner.
2. **Crlf:**
   Writes the end of line sequence to the console window.
3. **DumpRegs:**
   Displays the EAX, EBX, ECX, EDX, ESI, EDI, ESP:EIP and EFLAG registers.
4. **DumpMem (ESI=Starting OFFSET, ECX=LengthOf, EBX=Type):**
   Writes the block of memory to the console window in hexadecimal.
5. **WriteBin:**
   Writes an unsigned 32-bit integer to the console window in ASCII binary format.
6. **WriteChar:**
   Writes a single character to the console window.
7. **WriteDec:**
   Writes an unsigned 32-bit integer to the console window in decimal format.
8. **WriteHex:**
   Writes a 32-bit integer to the console window in hexadecimal format.
9. **WriteInt:**
   Writes a signed 32-bit integer to the console window in decimal format.

10. **WriteString (EDX= OFFSET String):**
    Write a null-terminated string to the console window.
11. **ReadChar:**
    Waits for single character to be typed at the keyboard and returns that character.
12. **ReadDec:**
    Reads an unsigned 32-bit integer from the keyboard.
13. **ReadHex:**
    Reads a 32-bit hexadecimal integers from the keyboard, terminated by the enter key.
14. **ReadInt:**
    Reads a signed 32-bit integer from the keyboard, terminated by the enter key.
15. **ReadString (EDX=OFFSET String, ECX=SIZEOF):**
    Reads a string from the keyboard, terminated by the enter key.
16. **SetTextColor (Background= Upper AL, Foreground= Lower AL):**
    Sets the foreground and background colors of all subsequent text output to the console.
17. **GetTextColor (Background= Upper AL, Foreground= Lower AL):**
    Returns the active foreground and background text colors in the console window.
18. **MsgBox (EDX=OFFSET String, EBX= OFFSET Title):**
    Displays a pop-up message box.
19. **MsgBoxAsk (EDX=OFFSET String, EBX= OFFSET Title):**
    Displays a yes/no question in a pop-up message box.
20. **WaitMsg:**
    Display a message and wait for the Enter key to be pressed.
21. **Delay:**
    Pauses the program execution for a specified interval (in milliseconds).
22. **getDateTime:**
    Gets the current date and time from system
23. **GetMaxXY (DX=col, AX=row):**
    Gets the number of columns and rows in the console window buffer.
24. **Gotoxy (DH=row , DL=col):**
    Locates the cursor at a specific row and column in the console window. By default X coordinate range is 0-79 and Y coordinate range is 0-24.
25. **Randomize:**
    Seeds the random number generator with a unique value.

Color and Its Value

Color	Value	Color	Value	Color	Value	Color	Value
Black	0	Red	4	Gray	8	Light Red	C
Blue	1	Magneta	5	Light Blue	9	Light Magenta	D
Green	2	Brown	6	Light Green	A	Yellow	E
Cyan	3	Light Gray	7	Light Cyan	B	White	h

Example 04:

WriteDec: The integer to be displayed is passed in EAX
WriteString: The offset of string to be written is passed in EDX
WriteChar: The character to be displayed is passed in AL

```
INCLUDE Irvine32.inc
.data
Dash BYTE " - ", 0
```

```
.code
main PROC
    mov ecx, 1FFh
    mov eax,1
    mov edx, OFFSET Dash
    L1:
        call WriteDec           ; EAX is writen as a decimal number
        call WriteString        ; EDX points to string
        call WriteChar          ; AL is the character
        call Crlf
        inc EAX                 ; next character
    Loop L1
    exit
main ENDP
END main
```

Example 05:

DumpMem: Pass offset of array in ESI, length of array in ECX & type in EBX

```
INCLUDE Irvine32.inc
.data
    arrayD SDWORD 12345678h, 8A4B2000h, 3434h, 7AB9h
.code
main PROC
    ; Display an array using DumpMem.
    mov esi, OFFSET arrayD          ; starting OFFSET
    mov ebx, TYPE arrayD            ; doubleword = 4 bytes
    mov ecx, LENGTHOF arrayD        ; number of units in arrayD
    call DumpMem                    ; display memory
    call Crlf                       ; new line
    call DumpRegs
    exit
main ENDP
END main
```

Example 06:

ReadInt: Reads the signed integer into EAX

WriteInt: Signed integer to be written is passed in EAX

WriteHex: Hex value to be written is passed in EAX

WriteBin: Binary value to be written is passed in EAX

```
INCLUDE Irvine32.inc
.data
    COUNT = 4
    prompt BYTE "Enter a 32-bit signed integer: ", 0
.code
main PROC
    ; Ask the user to input a sequence of signed integers
    mov ecx, COUNT
```

```
        L1:
                mov edx, OFFSET prompt
                call WriteString
                call ReadInt                    ; input integer into EAX
                call Crlf                       ; new line
                ; Display the integer in decimal, hexadecimal, and binary
                call WriteInt                   ; display in signed decimal
                call Crlf
                call WriteHex                   ; display in hexadecimal
                call Crlf
                call WriteBin                   ; display in binary
                call Crlf
                call Crlf
        Loop L1                                 ; repeat the loop
        exit
main ENDP
END main
```

Example 07:

SetTextColor: Background & foreground colors are passed to EAX

```
INCLUDE Irvine32.inc
.data
    str1 BYTE "Sample string in color", 0
.code
main PROC
    mov eax, yellow + (blue*16)
    call SetTextColor
    mov edx, OFFSET str1
    call WriteString
    call DumpRegs
    exit
main ENDP
END main
```

Example 08:

MsgBox: Offset of content string is passed in EDX. Offset of caption is passed in EBX.

```
INCLUDE Irvine32.inc
.data
    caption BYTE "Dialog Title", 0
    HelloMsg BYTE "This is a pop-up message box.", 0ah
    BYTE "Click OK to continue...", 0
.code
main PROC
    mov ebx, 0                          ; no caption
    mov edx, OFFSET HelloMsg            ; contents
    call MsgBox
```

```
        mov ebx, OFFSET caption          ; caption
        mov edx, OFFSET HelloMsg         ; contents
        call MsgBox
        exit
main ENDP
END main
```

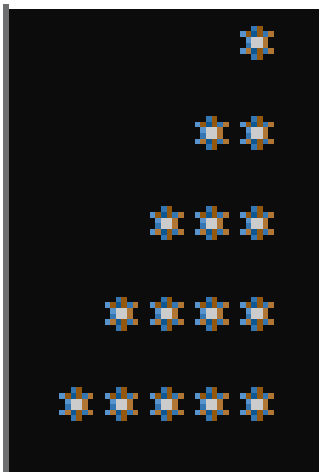
Example 09:

MsgBoxAsk: Offset of question string is passed in EDX. Offset of caption is passed in EBX. Selected value is returned in EAX (If : YES equal to 6 OR If: NO equal to 7)

```
INCLUDE Irvine32.inc
.data
        caption BYTE "Survey Completed",0
        question BYTE "Thank you for completing the survey.", 0ah
        BYTE "Would you like to receive the results?", 0
.code
main PROC
        mov ebx, OFFSET caption
        mov edx, OFFSET question
        call MsgBoxAsk
        ;(check return value in EAX)
        call DumpRegs
        mov ebx, OFFSET caption
        mov edx, OFFSET question
        call MsgBoxAsk
        ;(check return value in EAX)
        call DumpRegs
exit
main ENDP
END main
```

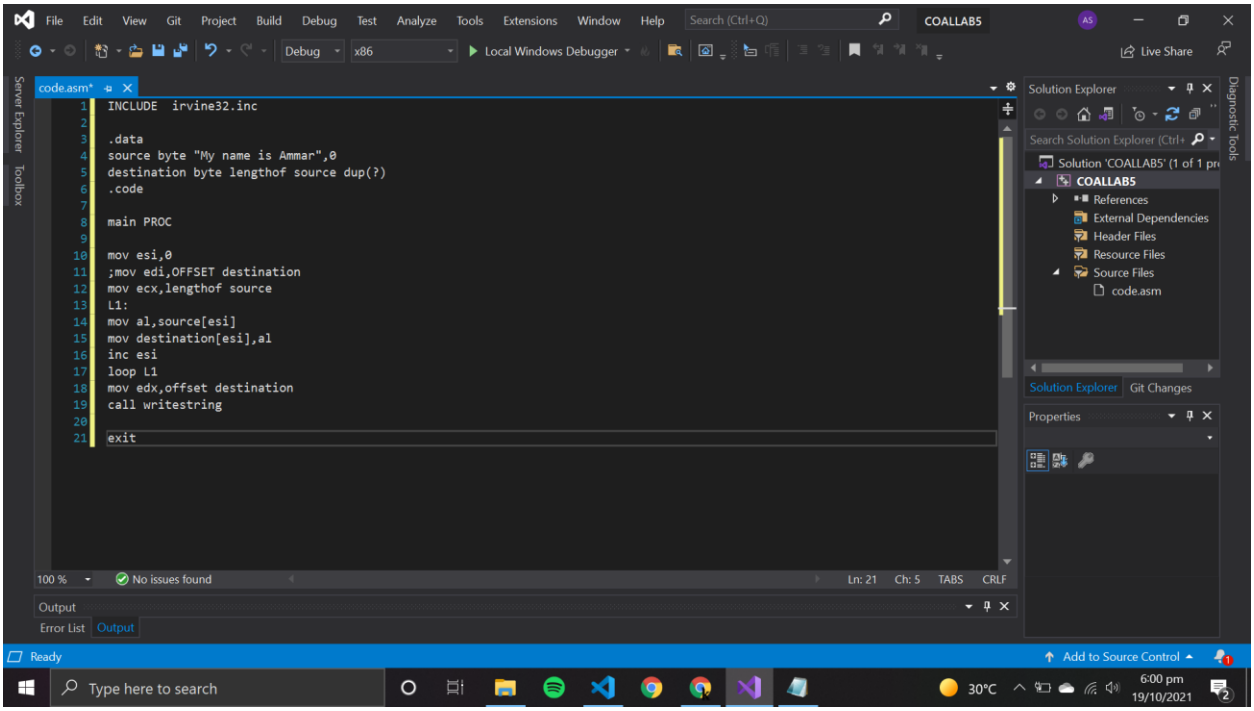
Lab Exercise:

1. Initialize an array named Source and use a loop with indexed addressing to copy a string represented as an array of bytes with a null terminator value in an array named as target.
2. Use a loop with direct or indirect addressing to reverse the elements of an integer array in place. Do not copy elements to any other array. Use SIZEOF, TYPE and LENGTHOF operators to make program flexible.
3. Write a program that uses a loop to calculate the first ten numbers of Fibonacci sequence.
4. Write a nested Loop Program that give following output.

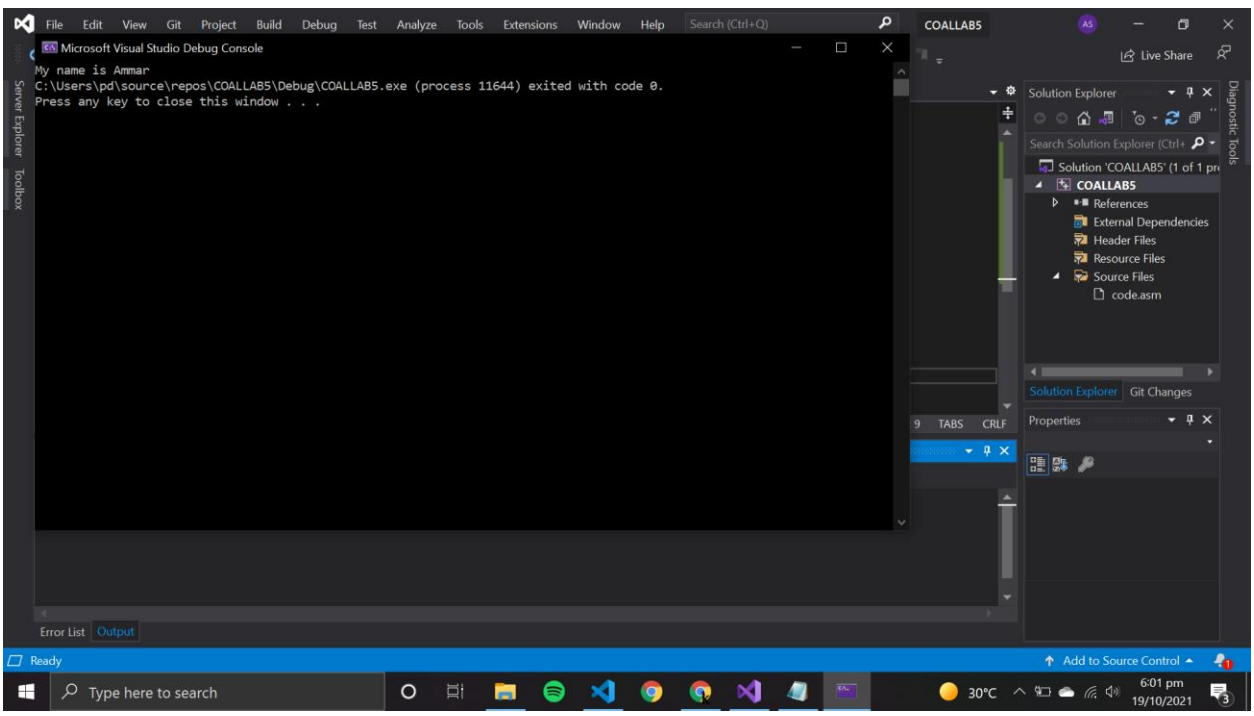


5. Write a program that enquire user about the quantity of Fibonacci sequence numbers to be display.
6. Implement task4 but user give input for number of lines for that triangle.

TASK 1:



OUTPUT:



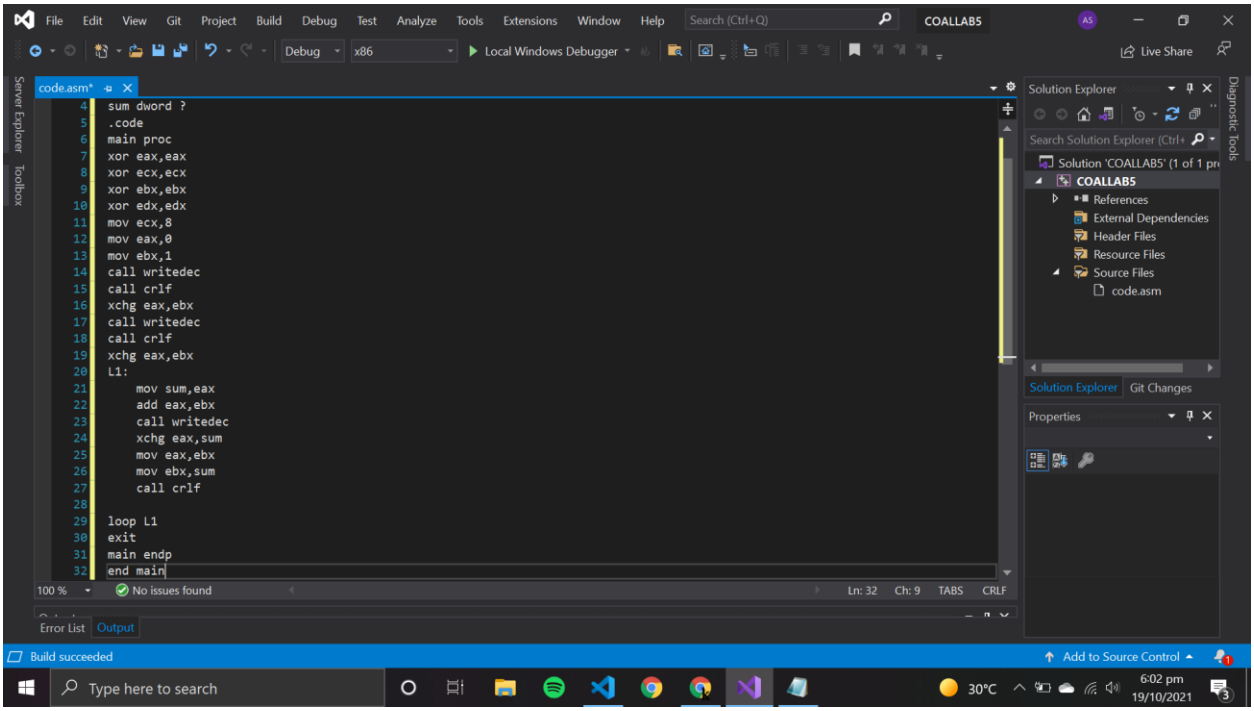
TASK 2:

```
source byte 10,20,30,40,50
.code

main PROC

mov esi,0
mov edi,(lengthof source) - 1
mov ecx,(lengthof source)/2

L1:
    mov al,source[esi]
    xchg al,source[edi]
    mov source[esi],al
    inc esi
    dec edi
loop L1
mov esi,0
mov ecx,lengthof source
xor eax,eax
L2:
    mov al,source[esi]
    call writedec
    call crlf
    inc esi
loop L2



exit
```

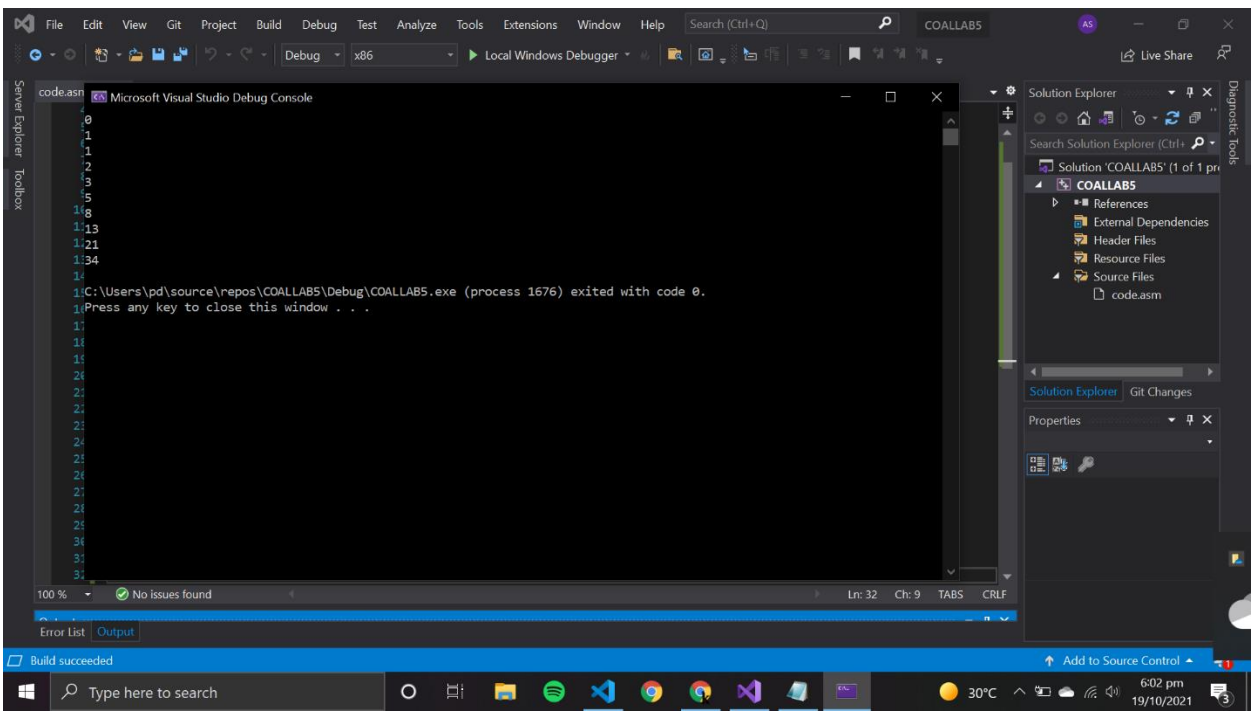
OUTPUT:

```
50
40
30
20
10

C:\Users\pd\source\repos\COALLAB5\Debug\COALLAB5.exe (process 3540) exited with code 0.
Press any key to close this window . . .
```

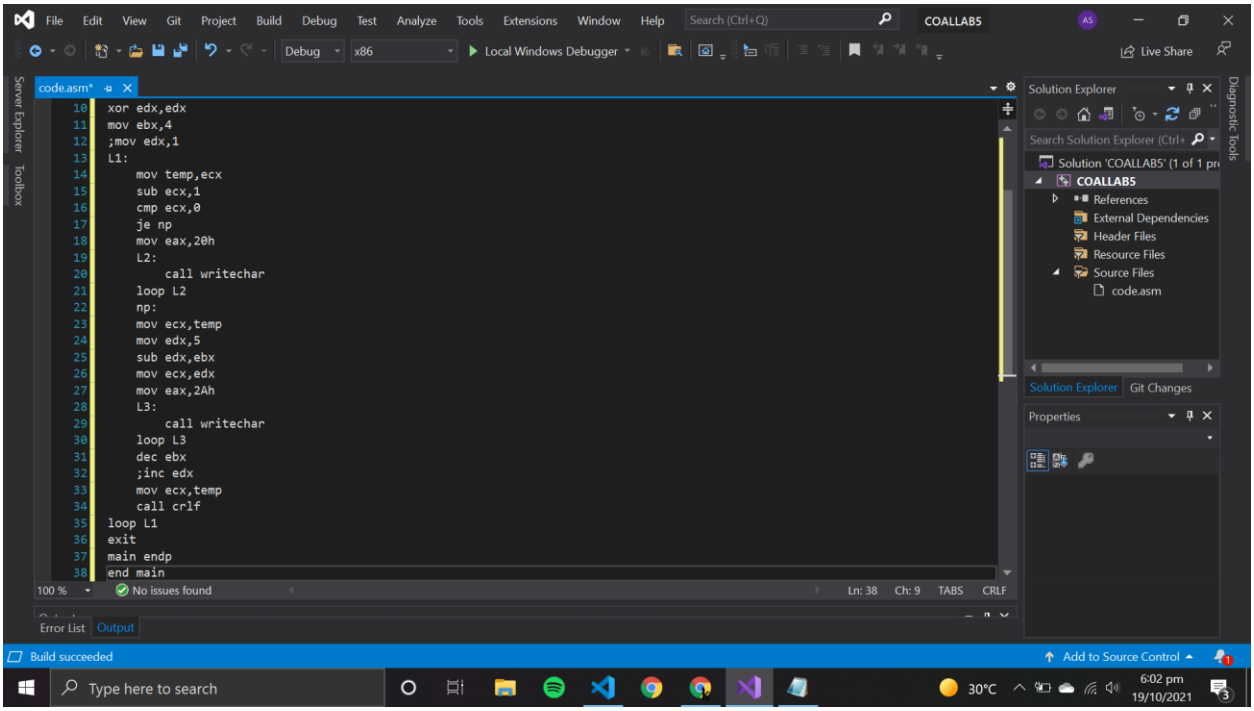
TASK 3:



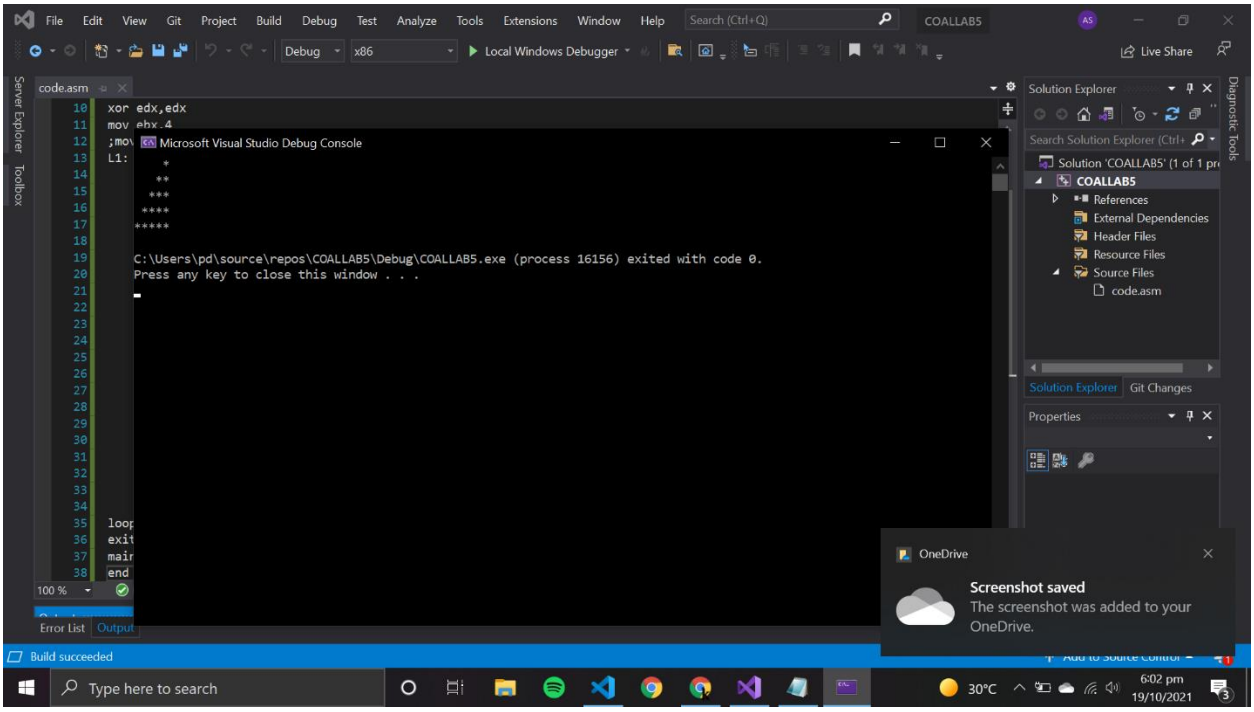
OUTPUT:



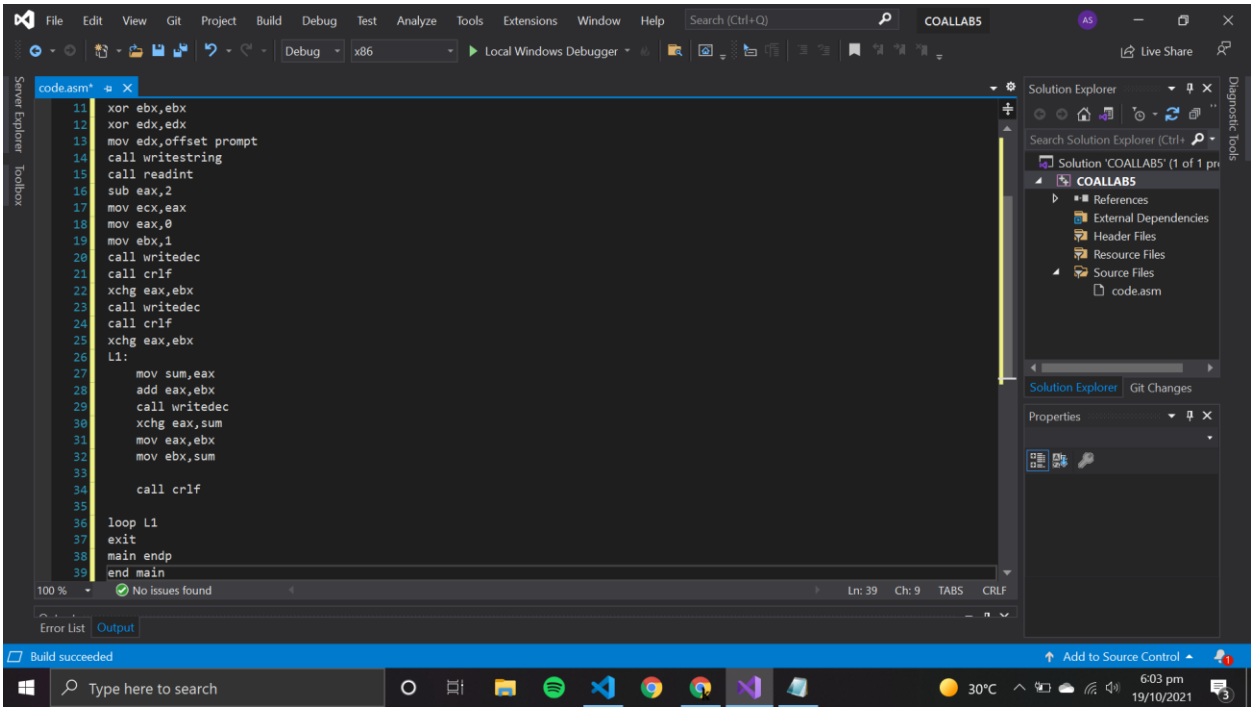
TASK 4:



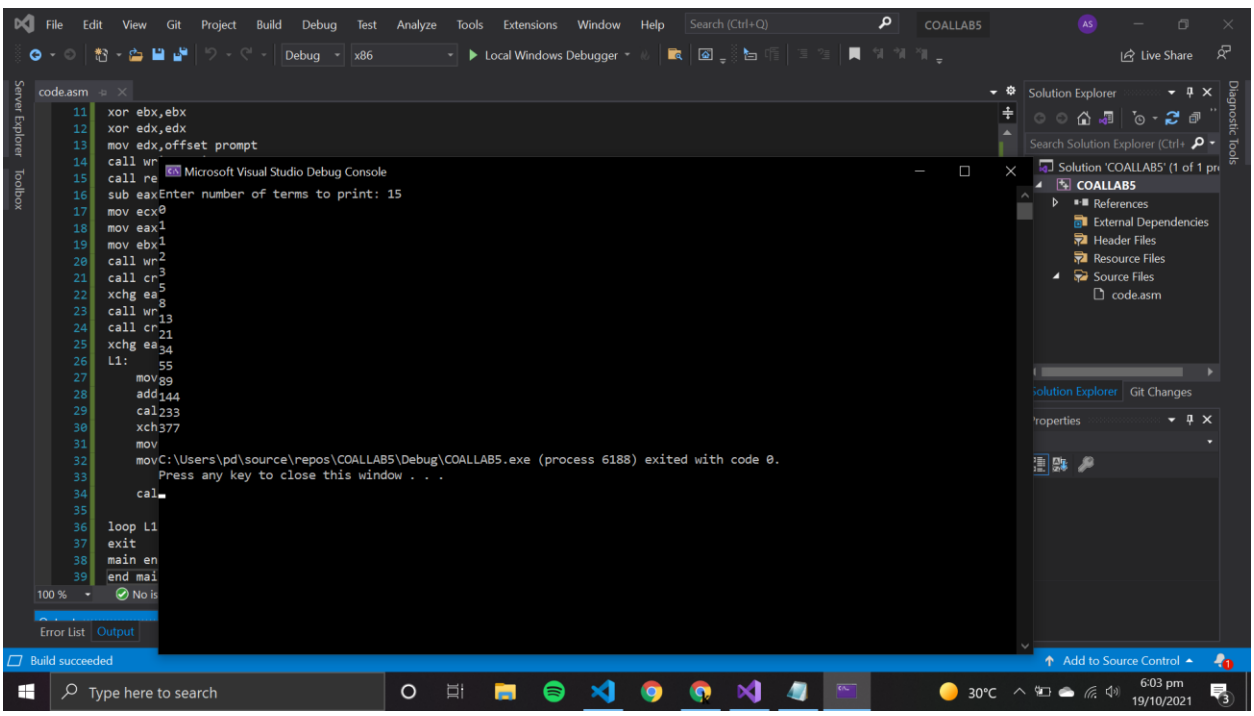
OUTPUT:



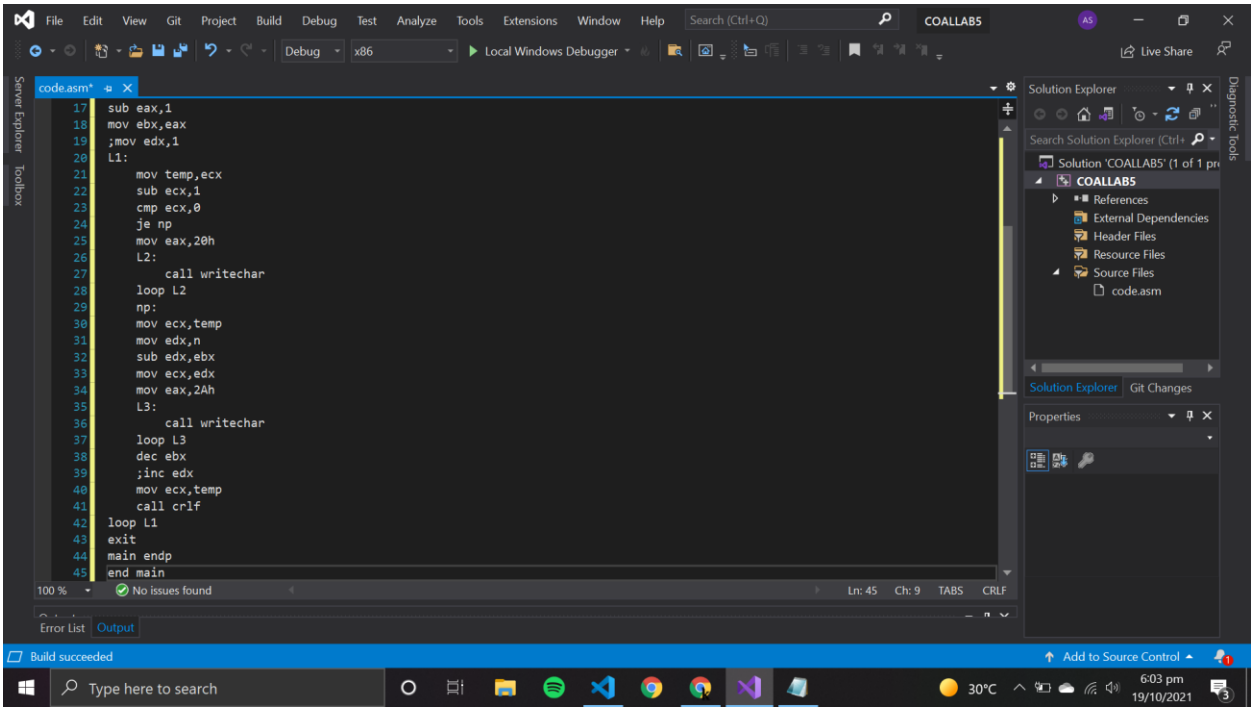
TASK 5:



OUTPUT:



TASK 6:



OUTPUT: