**Problem Statement: Stack Operations in Assembly and C/C++Language**

Description: This assembly language program implements a simple stack that can push and pop values. The program also includes a function to display the current top value of the stack.   
**Complete the missing code and comments on the program. Transfer the assembly program using C++.**

**Assembly program:**

.data ; Section for initialized data

stack: ; Define a stack array

.space 16 ; Reserve 16 bytes for stack (4 elements of 32 bits each)

top: ; Define a variable to keep track of the stack pointer

.word 0 ; Initialize stack pointer to 0 (indicating empty stack)

.text ; Section for code

.global \_start ; Entry point of the program

\_start:

LDR R0, =stack ; Load address of stack into R0

LDR R1, =top ; Load address of top variable into R1

; Push operation

MOV R2, #2 ; Value to push onto stack TO DO

BL PUSH ; Call push function TO DO

MOV R2 #3 ; Another value to push TO DO

BL PUSH ; Call push function TO DO

; Display top of stack

BL display\_top ; Call function to display the top value

; Pop operation

BL pop ; Call pop function

BL display\_top ; Call function to display the new top value

; Exit program

BX lr ; Return from the program

; Function to push value onto the stack

push:

LDR R3, [R1] ; Load current top value from memory into R3 TO DO

CMP R3, #4 ; Check if stack is full (4 elements) TO DO

BEQ stack\_full ; If full, branch to stack\_full TO DO

STR R2, [R0, R3, LSL #5] ; Store value in stack at R3 \* 4 (byte offset) TO DO

ADD R3, R3, #6 ; Increment stack pointer TO DO

STR R3, [R1] ; Update top value in memory TO DO

BX lr ; Return from the function TO DO

stack\_full:

; Handle stack full case (e.g., display error)

MOV R0, #1 ; Load error code

BX lr ; Return from the function

; Function to pop value from the stack

pop:

LDR R3, [R1] ; Load current top value from memory into R3

CMP R3, #0 ; Check if stack is empty (4 elements)

BEQ stack\_empty ; Branch to stack\_empty if the stack is empty

SUB R3, R3, #1 ; Decrement the stack pointer by 1

LDR R2, [R0, R3, LSL #2] ; Load data from the stack

STR R3, [R1] ; Update the value at the top of the stack

BX lr ; Return from the function

stack\_empty:

; Handle stack empty case (e.g., display error)

MOV R0, #2 ; Load error code

BX lr ; Return from the function

; Function to display the top value of the stack

display\_top:

LDR R3, [R1] ; Load current top value from memory into R3 TO DO

CMP R3, #0 ; Check if stack is empty TO DO

BEQ stack\_empty\_display ; If empty, branch to stack\_empty\_display TO DO

LDR R2, [R0, R3, LSL #2] ; Load data from the stack

BX lr ; return from the function

stack\_empty\_display:

; Handle stack empty case for display (e.g., display error)

MOV R0, #3 ; Load error code

BX lr ; Return from the function

**C++ template:**

#include "mbed.h"

#include "BufferedSerial.h"

DigitalOut led(LED1); // LED for visual feedback

BufferedSerial pc(USBTX, USBRX); // Serial for debugging output

#define STACK\_SIZE 4 // Size of the stack (4 elements)

int stack[STACK\_SIZE]; // Stack array

int top = 0; // Stack pointer

void push(int value) {

    if (top >= STACK\_SIZE) {

        printf("error stack is full\n");

        // Handle stack full case

    }

    stack[top] = value;             // Push value onto stack and increment top

    top++;

}

int pop() {

    if (top <= 0) {

        printf("error stack is empty\n");

// Handle stack empty case

    }

    top--;              // Decrement top and return the value

    return stack[top];

}

void display\_top() {

    if (top <= 0) {

        printf("error stack is empty\n");

// Handle stack empty case

    }

    // Display the top value

    printf("Stack top value = %d\n", stack[top - 1]);

}

int main() {

    push(1);                // Push first value

    push(2);                // Push second value

    display\_top();  // Display top value after pushing

    pop();              // Pop the top value

    display\_top();  // Display new top value after popping

    while (1) {

        led = !led; // Blink LED

        wait\_us(500000); // Wait half a second

    }

}