OBJECTIVE:

Upon completion of this lab, you will become familiar with the following concepts:

1. Use of procedures to modularize your program;
2. Use of MACROs to make your program more readable;
3. Use of the stack;
4. Use of instructions to perform arithmetic/logic operations, and comparisons.

BACKGROUND:

1. Operations

In this lab, we will implement binary operations found in simple calculators. Your calculator should be able to perform the following five types of integer operations:

   a. addition (+), for example, 5 + 3 = 8
   b. subtraction (-), for example, 5 - 3 = 2
   c. multiplication (*), for example, 5 * 3 = 15
   d. division (/), for example, 5 / 3 = 1
   e. modulus (%), for example, 5 % 3 = 2

To perform the desired operations, an algebraic expression must be entered. The calculator will then display the result followed by a new prompt for the next algebraic expression. All the operands must be handled as signed decimal numbers in the range [-131,072 to +131,071]. (How many bytes do you need to handle these numbers?)

2. Procedures vs. Macros

Procedures

A procedure, or subroutine, is a sequence of code that can be executed in such a way as if such instructions were inserted at the point of the main program from which it branched to the subroutine. The branch to a procedure is referred to as the call, and the corresponding branch back is known as the return from the subroutine. Procedures provide the primary means of breaking the program code into modules, which can be easily and individually designed, tested, and documented. They can also be stored in libraries and shared by a variety of other programs. Procedures have one major disadvantage in that extra code is needed to join them together in such a way that they can communicate with each other. This extra code is referred to as linkage.
NOTE: The **MAIN procedure is always a FAR procedure, called by the Operating System, while the other procedures within your program can be NEAR or FAR** (in this assignment, you should use only NEAR).

**Macros**

A macro is a small segment of code that needs to be written repeatedly throughout a program. That is, some times, repetitive segments of code are not long enough to “deserve” its own subroutine – e.g. it would take longer to call the subroutine than to actually execute that piece of code. So, instead, we use a MACRO to define that sequence only once, and then we repeat that sequence as many times as we want by placing a simple statement (the name of the MACRO) at the point of each appearance. When a reference to a macro is encountered by the assembler, the assembler replaces the reference with the macro’s actual sequence of code. This replacement action is referred to as a macro expansion. To understand exactly what happens to a macro after assembling, try to look at your .LST file.

Once a MACRO has been defined in an .ASM file, you can include that entire sequence of statements simply by using its name as an operator. If you define the MACRO to have parameters, you can also pass operands to it. But keep in mind that a MACRO is **NOT** a function and therefore, the parameters are not actually passed to the MACRO, but simply replaced by the values in the “call”. An example of defining and using a macro is shown below:

```assembly
GetChar MACRO   mychar
    MOV     AH,1
    INT     21h
    MOV     mychar,al
ENDM

RetDOS MACRO
    MOV     AH,4Ch
    INT     21h
ENDM

; Example of how to use macros in your program
Cod SEGMENT
    ASSUME  CS:Cod

Begin:  GetChar BH  ; the above 3 instructions in the macro GetChar
        ; are inserted here with the word "mychar"
        ; replaced by "BH"
    RetDOS  ; again, the 2 lines in RetDOS are inserted here

Cod ENDS
END     Begin
```

If your macros are defined in a separate file, say DOS.MAC, you must add the following assembly directive at the beginning of your program:

```
INCLUDE DOS.MAC
```
**PRELAB:**

Design the pseudo code for this lab. That is, the pseudo code of a program that reads a line with the arithmetic expression in the form: "Operand1 Operator Operand2" without the quotes and checks for its validity – i.e. correct ranges, operator, etc. If the input is valid, it converts the ASCII operands to numbers, saves them into registers and then, calls the corresponding "OPERATE" procedure to perform the desired operation. Finally, it displays the result returned in some of the registers by converting it from Hex to ASCII. If the input is invalid, the program displays an error message and prompts for the next expression.

**REQUIREMENTS:**

1. Write a program in the EXE format.

2. The program should contain the following procedures:

   a. **INPUT**: Reads the arithmetic expression and checks for its validity.
   b. **OPERATE**: This is not the actual name of the functions. You should write separate procedures for each possible operator, e.g. **DO_ADDITION**, **DO_SUBTRACTION**, etc. Each of these functions performs the desired operation.
   c. **OUTPUT**: Converts the result to ASCII code and display it on the screen.
   d. **CONV_ASCII2HEX**: Performs data type conversion from ASCII to hexadecimal. Refer to your textbook for more details on ASCII codes.
   e. **CONV_HEX2ASCII**: Performs data type conversion from hexadecimal to ASCII.
   f. **ERROR**: Prints out different error messages for illegal operators and operands.

3. All the input operands must be handled as signed decimal numbers in the range [-131,072 to +131,071]. The results must be at least **twice as long as** the inputs (not twice the range!!). For the division, only the quotient should be displayed.

4. All DOS function calls used in this program must be defined as MACRO and saved in a file called **DOS.MAC**.

**SAMPLE EXECUTION:**

Assume the executable filename is **CALC.EXE**

```
C:/> CALC

Enter an algebraic command line: 255 + 2 <enter>
Result: 257
Again? Y

Enter an algebraic command line:
```

255 + 2 <enter>
Result: 257
Again? Y <enter>

Enter an algebraic command line:
255 + -2 <enter>
Result: 253
Again? Y <enter>

Enter an algebraic command line:
255 - 2 <enter>
Result: 253
Again? Y <enter>

Enter an algebraic command line:
255 * 2 <enter>
Result: 510
Again? Y <enter>

Enter an algebraic command line:
-255 / 2 <enter>
Result: -127
Again? Y <enter>

Enter an algebraic command line:
255 % 2 <enter>
Result: 1
Again? Y <enter>

Enter an algebraic command line:
255 <enter>
Error!!
Input format: Operand1 Operator Operand2
Operand: decimal numbers
Operator: + - * / %
Again? Y <enter>

Enter an algebraic command line:
255 / 9 + 1 <enter>
Error!!
Input format: Operand1 Operator Operand2
Operand: decimal numbers
Operator: + - * / %
Again? Y <enter>

Enter an algebraic command line:
25A / 9 <enter>
Error!!
Input format: Operand1 Operator Operand2
Operand: decimal numbers
Operator: + - * / %
Again? N <enter>

C:\>