Kennesaw State University

Southern Polytechnic College of Engineering and Engineering Technology

EE 3501

Embedded Systems

Project Report

Arithmetic Calculator Using the Keypad

By

John Yare

Date: 12 – 4 – 2023

Table of Contents

Table of Figures

Objectives

The objective of this project is to design a simple arithmetic calculator that performs all four arithmetic operations (addition, subtraction, multiplication, and division), on two user-supplied integer operands and displays the result on the LCD screen. The calculator will interact with the user through a keypad and display the results on an LCD screen. The project is aimed at providing a user-friendly and error-tolerant interface for basic arithmetic operations.

Apparatus List

1. Microcontroller (NUCLEO-F401RE).
2. Jumper Cables.
3. Breadboard.
4. USB connector.
5. 4x4 Keypad.
6. LCD display screen.
7. 10kOhm Potentiometer.
8. Assembler for assembly language programming

Engineering Design

Overview

The design approach meets the project design requirements by implementing each operation using an assembly language-based function. The arithmetic calculator is programmed to add, subtract, multiply, and divide integer numbers and display the result on the LCD screen. The design of the arithmetic calculator involves breaking down the functionality into modular components. The modules include operand input, operator selection, arithmetic operations, and result display. These modules work together to fulfill the project objectives/specifications.

Operand Input Module

This module handles the input of two integer operands from the user through the keypad. The user will utilize the keypad to enter the operands and select the operation. The first step of the calculator operation will begin by displaying the text “NO 1” on the screen and wait for the user to use the keypad to type the first integer operand followed by pressing the ‘#’ key. The user input of the 1st operand will be mirrored on the screen. If a non-integer value is entered, the calculator will signal an error to the user by displaying the text “ERROR” for 2 seconds and then restart step 1. Next, the text “NO 2” will display on the screen and the calculator will wait for the user to use the keypad to type the second integer followed by pressing the ‘#’ key. The user input of the 2nd operand will be mirrored on the screen. If a non-integer value is entered, the calculator will signal an error to the user by displaying the text “ERROR” for 2 seconds and then restart step 2.

Operator Selection Module

Upon the entry of the second operand, the calculator will display the text “SELECT OPERATION” on the screen. The keys A to D are mapped to the four arithmetic operations as follows: A – add, B – subtract, C – multiply, D – divide. The calculator will wait for entry of the operator selection, which is signaled by entering one of the four letters. The user input of the operation selected will be mirrored on the screen for 3 seconds. If the entry is not equal to letters between A to D, the calculator will signal an error to the user by displaying the text “ERROR” for 3 seconds and then restart step 3.

Arithmetic Operations Module

Each arithmetic operation (addition, subtraction, multiplication, division) is implemented as a separate assembly language function (_asm.s file). These functions take the two operands and perform the specified operation, returning the result.

Result Display Module

After the arithmetic operation is completed, the result is displayed on the LCD screen. The calculator then returns to the operand input module, starting a new calculation cycle.

Call Graphs and Flowcharts

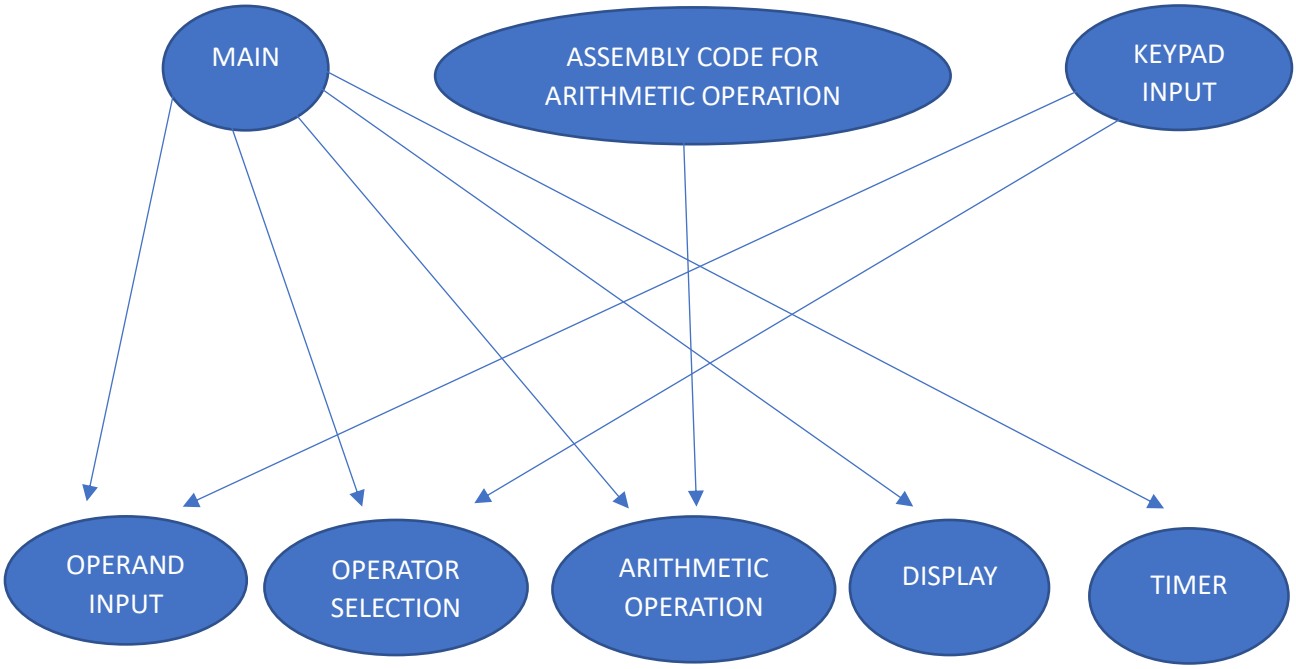

Figure 1: Call Graph showing structural design of arithmetic calculator.

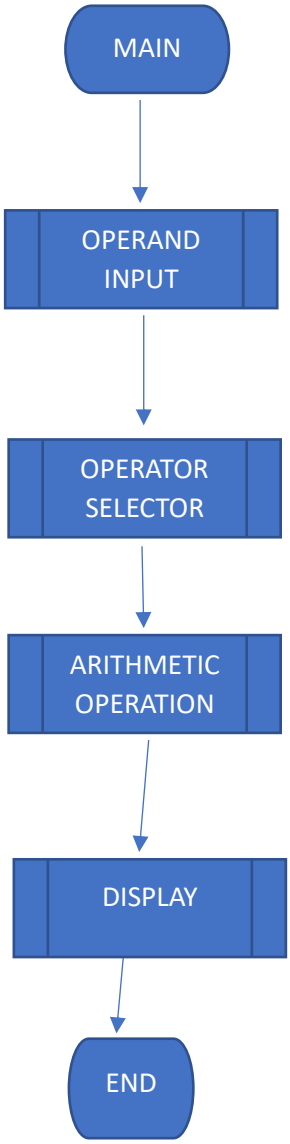

Figure 2: Main Module Flowchart

Test Plan

Operand Input Testing

1. Verify that “NO 1”, and “NO 2” are displayed correctly to allow the user input an integer.
2. Verify that integer input is accepted and mirrored correctly on the screen.
3. Confirm that errors are displayed correctly for non-integer inputs by displaying the text “ERROR” for 3 seconds and returning to the last user input.
4. Check that multi-digit operands up to 4 digits are supported.

Operator Selection Testing

1. Ensure the correct mapping of keys A to D for each operation (addition, subtraction, multiplication, and division).
2. Confirm that errors are displayed correctly for invalid inputs by displaying the text “ERROR” on the LCD and retuning to the “SELECT OPERATION” input.

Arithmetic Operations Testing

1. Test each arithmetic operation function separately at least twice.
2. Verify the correctness of results for various operand combinations.

Result Display Testing

1. Confirm that the result is displayed correctly on the LCD for 3 seconds. Display then restarts operation again from the first operand input.
2. Check the support for multi-digit results up to 6 digits.

Source Code

Main.cpp

```
#include "mbed.h"                /*This section includes the necessary libraries*/

#include "TextLCD.h"
```

```
#include <cstdio>
#include <string>
#include <iostream>
#include <sstream>

DigitalOut ROW1 (PA_6);              /*This section defines digital input/output
DigitalOut ROW2 (PA_7);               pins for the keypad rows and columns*/
DigitalOut ROW3 (PB_6);
DigitalOut ROW4 (PC_7);
DigitalIn COL1 (PA_9, PullUp);
DigitalIn COL2 (PA_8, PullUp);
DigitalIn COL3 (PB_10, PullUp);
DigitalIn COL4 (PB_4, PullUp);


int col_scan(void);                  /*functions 'col_scan' and 'keypad_scan'
char keypad_scan (void);              are declared*/

//TextLCD lcd(RS, E, D4, D5, D6, D7);                    // RS, E, D4-D7,
LCDType=LCD16x2, BL=NC, E2=NC, LCDTCtrl=HD44780
TextLCD lcd(PA_0, PA_1, PA_4, PB_0, PC_1, PC_0);

extern "C" int add_asm(int operand_1, int operand_2);   /*External declarations
extern "C" int sub_asm(int operand_1, int operand_2);    for assembly functions
extern "C" int mul_asm(int operand_1, int operand_2);    that perform arithmetic
extern "C" int div_asm(int operand_1, int operand_2);    operations*/
```

```
int main() {
    bool valid_number = false;                  /*This section contains variable
    string numString = "";                        declarations used in the main
    stringstream convertToInt;                    function*/
    int numInt;
    char key = 'n';
    int operand_1;
    int operand_2;

    while(1) {
        valid_number = false;                   /*This section of code lets the user
        // Get the first operand                  input the first operand by displaying
        while (valid_number == false) {           'NO 1' and storing the keypad
            lcd.cls();                            input into an integer variable
            lcd.printf("NO 1: ");                 'operand_1' using the '#' key*/
                                                /*Displays 'NO 1' on the LCD while
                                                waiting for user input*/
            while (key != '#') {
                key = keypad_scan();
                wait(0.3);

                if (key != 'n' && key != '#'){
                    if (key != 'A' && key != 'B' && key != 'C' && key != 'D' && key !=
'*') {
                        numString += key;
```

```
            lcd.printf("%c", key);
            wait(0.3);
        }                           /*Stores user's 'NO 1' keypad input as a
                                    string object*/
        else {
            lcd.cls();
            lcd.printf("ERROR");
            wait(1);
            numString = "";
            break;
        }                           /*Display 'ERROR' if key pressed is not a
                                    number or '#' and repeats 'NO 1' input*/
    }

  }
if (numString.size() > 0) {
convertToInt.clear(); //clears the content of stringstream object
convertToInt.str(numString); //transfers string content into stringstream object
convertToInt >> numInt; //outputs integer from stringstream object
valid_number = true;
operand_1 = numInt;             /*This section of code converts NO 1 string
}                                 input to integer*/
}

// Get the second operand       /* This section of code lets the user input the
```

```
valid_number = false;                second operand by displaying ‘NO 2’ and
key = 'n';                           storing the keypad input into an integer
numString = "";                      variable ‘operand_2’ using the ‘#’ key*/

while (valid_number == false) {
   lcd.cls();
   lcd.printf("NO 2: ");             /*Displays ‘NO 2’ on the LCD while
                                     waiting for user input*/
   while (key != '#') {
      key = keypad_scan();
      wait(0.3);

      if (key != 'n' && key != '#'){
         if (key != 'A' && key != 'B' && key != 'C' && key != 'D' && key !=
'*') {
            numString += key;
            lcd.printf("%c", key);
            wait(0.3);
         }                           /*Stores user’s ‘NO 2’ keypad input as a
                                     string object*/
         else {
            lcd.cls();
            lcd.printf("ERROR");
            wait(1);
            numString = "";
            break;
```

```
            }                       /*Display 'ERROR' if key pressed is not a
                                    number or '#' and repeats 'NO 2' input*/
          }
        }
        if (numString.size() > 0) {
        convertToInt.clear(); //clears the content of stringstream object
        convertToInt.str(numString); //transfers string content into stringstream
object
        convertToInt >> numInt; //outputs integer from stringstream object
        valid_number = true;        /*This section of code converts NO 2 string
        operand_2 = numInt;         input to integer*/
        }
      }


      // Get the operation                    /*This section of code performs the
      valid_number = false;                  user's desired arithmetic operation
      key = 'n';                             by displaying 'SELECT
      numString = "";                        OPERATION' and converting each
      while (valid_number == false) {        key (A-D) pressed to an operation
        lcd.cls();                           sign then performing the
        lcd.printf("SELECT OPERATOR: "); corresponding arithmetic operation,
        key = keypad_scan();                 displaying the result on the LCD for
        wait (0.3);                          3 seconds*/
        if (key != 'n') {
          if (key == 'A'){          /*If A is pressed, performs NO 1 + NO 2*/
```

```
            lcd.cls();
            lcd.printf("%d + %d = %d", operand_1, operand_2,
add_asm(operand_1, operand_2));

            wait(2);
            valid_number = true;
        }

        if (key == 'B'){            /*If B is pressed, performs NO 1 - NO 2*/
            lcd.cls();
            lcd.printf("%d - %d = %d", operand_1, operand_2,
sub_asm(operand_1, operand_2));
            wait(2);
            valid_number = true;
        }

        if (key == 'C'){            /*If C is pressed, performs NO 1 * NO 2*/
            lcd.cls();
            lcd.printf("%d * %d = %d", operand_1, operand_2,
mul_asm(operand_1, operand_2));
            wait(2);
            valid_number = true;
        }

        if (key == 'D'){            /*If D is pressed, performs NO 1 / NO 2*/
            lcd.cls();
```

```
            lcd.printf("%d / %d = %d", operand_1, operand_2,
div_asm(operand_1, operand_2));
            wait(2);
            valid_number = true;
          }

          if (key != 'A' && key != 'B' && key != 'C' && key != 'D'){
            lcd.cls();                    /*If any other key asides A-D is takes
            lcd.printf("ERROR");          pressed, code returns 'ERROR' and
            wait(2);                      user back to 'SELECT
            numString = "";               OPERATION'*/
          }
        }
      }
  }                                       /*Repeats arithmetic calculator
                                          cycle*/
}

int col_scan (void) {                     /*The 'col_scan' function scans the columns
  if (COL1 == 0){                           of the keypad to determine which key is
    return 0;                               pressed*/
  }
  else if (COL2 == 0){
    return 1;
  }
  else if (COL3 == 0){
```

```
      return 2;
   }
   else if (COL4 == 0){
      return 3;
   }
   else {
   return 5;
   }
}
char keypad_scan (void) {
char key_map[4][4] = {
{'1','2','3','A'},
{'4','5','6','B'},
{'7','8','9','C'},
{'*','0','#','D'}
};

   for (int i = 1; i <= 4; ++i) {
      if (i == 1) {
         ROW1 = 0;
         ROW2 = ROW3 = ROW4 = 1;
      }
      if (i == 2) {
         ROW1 = ROW3 = ROW4 = 1;
         ROW2 = 0;
```

/*The ‘keypad_scan’ function scans the entire keypad to get the pressed key*/

```
}
if (i == 3) {
   ROW3 = 0;
   ROW1 = ROW2 = ROW4 = 1;
}
if (i == 4) {
   ROW4 = 0;
   ROW1 = ROW2 = ROW3 = 1;
}

int row = i - 1;
int col = col_scan();
if (col == 0) {
   return key_map[row][col];
}
if (col == 1) {
   return key_map[row][col];
}
if (col == 2) {
   return key_map[row][col];
}
if (col == 3) {
   return key_map[row][col];
}
```

```
    }
    return 'n';
}
```

Add_asm.s

```
        AREA |.text|, CODE, READONLY    /*Defines a new code section named
                                        '|.text|' that is read-only*/
add_asm     PROC        /* Declares the start of a function) 'add_asm'.*/
        EXPORT add_asm      /*This line exports 'add_asm'*/
        ADD R0, R1          /*R0 = R0 + R1*/
        BX LR
        ENDP                /*Ends 'add_asm' function*/
        ALIGN               /*Aligns next instruction on a word boundary*/
        END                 /*End of code section*/
```

Sub_asm.s

```
        AREA |.text|, CODE, READONLY    /*Defines a new code section named
                                        '|.text|' that is read-only*/
sub_asm     PROC        /* Declares the start of a function) 'sub_asm'.*/
        EXPORT sub_asm      /*This line exports 'sub_asm'*/
        SUB R0, R1          /*R0 = R0 - R1*/
        BX LR
        ENDP                /*Ends 'sub_asm' function*/
        ALIGN               /*Aligns next instruction on a word boundary*/
        END                 /*End of code section*/
```

Mul_asm.s

```
        AREA |.text|, CODE, READONLY   /*Defines a new code section named
                                        '|.text|' that is read-only*/
mul_asm     PROC                /* Declares the start of a function) 'mul_asm'.*/
       EXPORT mul_asm           /*This line exports 'mul_asm'*/
       MUL R0, R1               /*R0 = R0 * R1*/
       BX LR
       ENDP                     /*Ends 'mul_asm' function*/
      ALIGN                     /*Aligns next instruction on a word boundary*/
      END                       /*End of code section*/
```

Div_asm.s

```
        AREA |.text|, CODE, READONLY   /*Defines a new code section named
                                        '|.text|' that is read-only*/
div_asm     PROC          /* Declares the start of a function) 'div_asm'.*/
          EXPORT div_asm    /*This line exports 'div_asm'*/
          SDIV R0, R1       /*R0 = R0 / R1*/
          BX LR
          ENDP              /*Ends 'div_asm' function*/
          ALIGN             /*Aligns next instruction on a word boundary*/
          END               /*End of code section*/
```

Conclusion

The designed arithmetic calculator provides a robust and user-friendly solution for basic arithmetic operations. The modular approach ensures maintainability and ease of testing. It is simple, efficient, and easy to use. The calculator can perform all four arithmetic operations (addition, subtraction, multiplication, and division)

on two user-supplied operands and display the result on the LCD screen. The user can utilize the keypad to enter the operands and select the operation. The design approach meets the project design requirements by implementing each operation using an assembly language-based function. The calculator has been designed to add, subtract, multiply, and divide integer numbers and display the result on the LCD screen. The user input of the operands and operation selected will be mirrored on the screen. If a non-integer value is entered, the calculator will signal an error to the user by displaying the text “ERROR” and then restart the corresponding step. The test plan ensures that each module in the design hierarchy will be tested to ensure that it is functioning correctly. Overall, the solution meets the project requirements effectively, offering a reliable calculator for integer arithmetic operations. The only shortcoming of this design is the ease of portability.