University of Engineering and Technology

Muaaz Butt 2022-CS-677

Zunaira Kabeer 2022-CS-702

Amen Munir 2022-CS-703

Khadija Irfan 2022-CS-712

Hayaa Irfan 2022-CS-713

Title: ‘Procedure Library Application in x86 Assembly’

Supervisor: Lecturer Sir Hassan Imtiaz

ACKNOWLEDGMENTS

I am truly appreciative to Allah for His constant guidance during this investigation. My strength has come from his support. I am grateful to my teachers for their helpful guidance in directing the direction of this study. Their commitment to learning has served as a beacon of hope. My sincere appreciation to my staff for their attitude of cooperation. We were successful because each person contributed significantly.

I also thank the team mates who worked day and night to make this project reach its true potential. It could not have been possible without our supervisor who gave us the correct direction whenever we felt lost.

Table of Contents

List of Acronyms/Abbreviations

Abbreviation	Full Form
IA-32	Intel Architecture, 32-bit
MASM	Microsoft Macro Assembler
CPU	Central Processing Unit
VM	Virtual Machine
IOT	Internet of Things
IDE	Integrated Development Environment

List of Figures

Abstract

The procedure library application is a modified extension to the already existing Irvine32 Library in assembly. Assembly Language is a very low level language hence it demands more understanding of hardware and requires that the users interact with the computer’s hardware directly. This project focused on key areas like memory and limitation of already existing library and introduced a new set of functionalities related to manipulation of diverse data sets.

The Irvine32 Library by Kip Irvine, served as a starting point in our project. Although it is a valuable asset for the beginners coming into programming in assembly but we recognized the need for such a library that along with being simple should also be efficient and a little complex.

Our project represents an effort to extend the functionality of assembly language programming by providing a more comprehensive library of procedures. As technology continues to advance, the importance of understanding low-level programming languages remains, and our project seeks to make this understanding more accessible and practical for both developers and students alike.

Introduction

In computer programming, assembly language, often referred to simply as assembly and commonly abbreviated as ASM or asm, is any low-level programming language with a very strong correspondence between the instructions in the language and the architecture's machine code instructions. [1]. It is a low level language that is used in communicating directly with the machine's hardware. "It is a type of programming language that translates the high level language to machine language" [2]. Basically it converts the human understandable code to 1's and 0's or binary which the computer uses in order to perform all of it's processing on data.

A low level language is a programming language that provides nearly zero abstraction and is very similar to the processor or machine code and in most cases does not need much compilation for the computer's architecture to understand.

In x86 architecture, there exists a library called as Irvine32 which provides us with some basic functionalities already implemented in assembly and are ready to be used by the developer in their assembly project.

> "There is no Microsoft-sanctioned standard library for assembly language programming. When programmers first started writing assembly language for x86 processors in the early 1980s, MS DOS was the commonly used operating system. These 16-bit programs were able to call MS DOS functions (known as INT 21h services) to do simple input/output. Even at that time, if you wanted to display an integer on the console, you had to write a fairly complicated procedure that converted from the internal binary representation of integers to a sequence of ASCII characters that would display the integer on the screen". [3]

The project we developed is an extension to this already existing library. We figured that apart from some of the basic functionalities that this library provided us there were a lot of other procedures that could have been included to make the developer's life a little easier while coding in assembly. The procedure library application gives the user a wide range of functions other that the simple input/output procedures provided by Irvine32. We as developers deal with a lot of data manipulation be it in words or numbers, so we decided to add some procedures in the Irvine library regarding strings and integer numbers. These procedures are written in x86 assembly and can be used by anybody in their project to make it easier for them as they would not have to go through hassle to write extensive codes just to do simple manipulation of numbers or characters, they can simply copy or include our already implemented file in their project folder and work with them.

We used the IA-32 bit architecture, with Irvine32 standard procedures as the main setup. All of the producers are implemented in x86 assembly which produces the object code for the x86 processors class.

Apart from real-life development purposes we also aim to provide this implementation to students who are learning assembly language in their Computer Science degree. We as students struggled to find accurate implementation of a lot of procedures on the internet, so giving access to our implemented files to the students along with the source code would be our contribution to the learning development of the educational body.

Literature Review

The Irvine32 Library was introduced into the assembly language world by Kip Irvine, Florida International University. It was designed for absolute starters to write and build simple program, not for efficiency purposes. It provides library functions that read and print data onto the console and perform more of such basic functionalities for beginner developers. It is like a simplified C library. Now we do agree this is a very useful library but it just helped the users to perform basic functionalities which the user can manage. It is very beginner friendly as it converts a few lines of repetitive input/output code to one line code not it is not very efficient as there are a lot of other extensive procedures that can be converted into library functions to increase the overall productivity of a developer coding in assembly. The smaller building blocks to coding in assembly language can also be confided into a single library and that is exactly what we with project aimed to do. We built such a library that has thirty to thirty-five such procedures that can be used as basis for larger projects and increase developers productivity and efficiency to code.

Methodology

We used assembly language in 32-bit mode for this project and the built in Irvine32 library on Visual Studios 2022. The x86 assembly in 32 bit, is made to run on processors of IA-32 processors. The x86 architecture has mainly three modes of operation which are real-address mode, system management mode and protected mode. This architecture also has another mode called as virtual-8806. We used the protected mode of this architecture, in which all the features are available as well as the programs are allocated separate memory and the processers does not let the program reference memory addresses that are out of scope.

The tool we used for the compilation of our code is Visual Studios 2022. It contains MASM which is an assembly language for x86 processors. “This is what your CPU interprets when you write a piece of code in a programming language like C++, Python, Java or Assembly and is translated by the compiler and then converted to object code.” [4]

Assembly language is a low level-language that uses some general purpose registers for coding, segment registers to hold instruction codes of the program, an instruction pointer which processes the next instruction to be executed by the processors and some control and status flags.

Findings

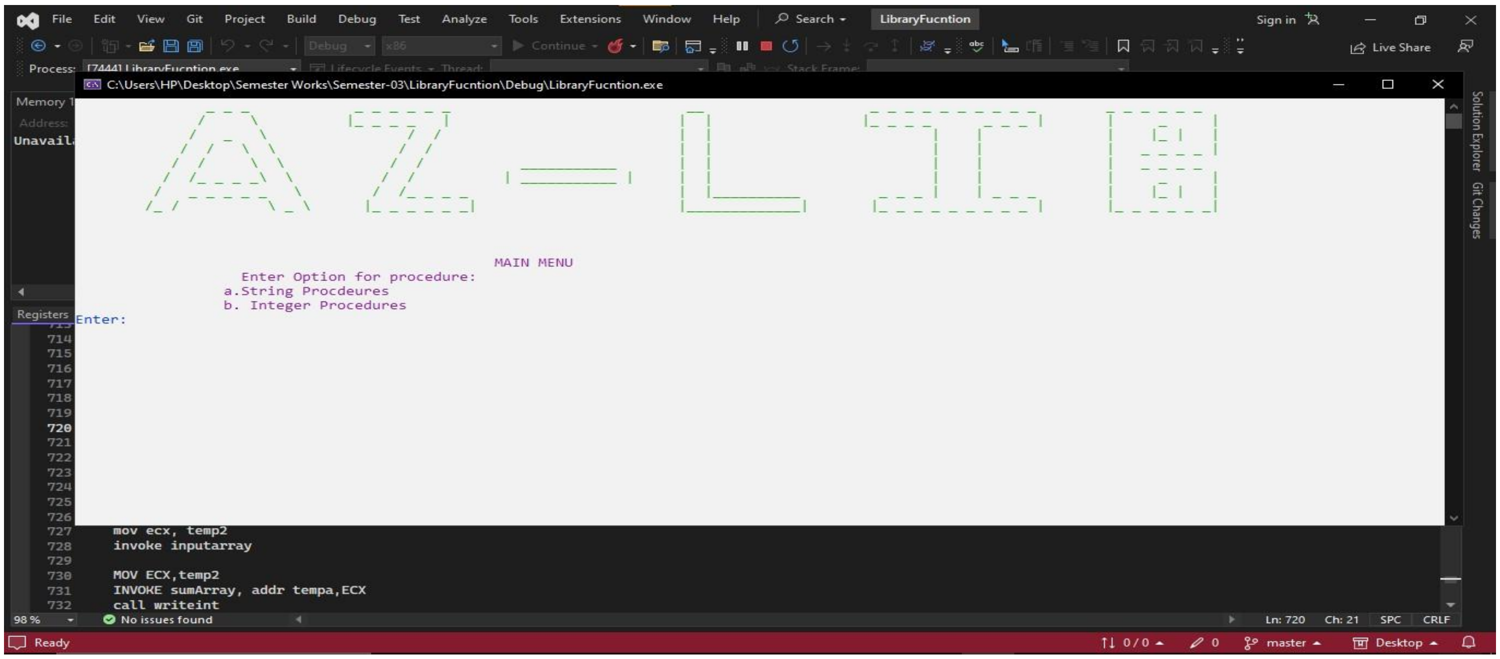
This project helped us gain a better understanding of the assembly language. We found out that using low level language to solve complex tasks was an effort and required deep understanding of the basics of assembly and the uses of registers and how the memory may or may not be accessed. Also the use of a debugger to figure out where the issue lies in the code turned out to be the best practice as you can go through your code and the memory, line by line an identify exactly where your procedure went wrong and fix it.

Discussion

Main Interface

The main interface of this project is console based. The procedures are divided into two categories:

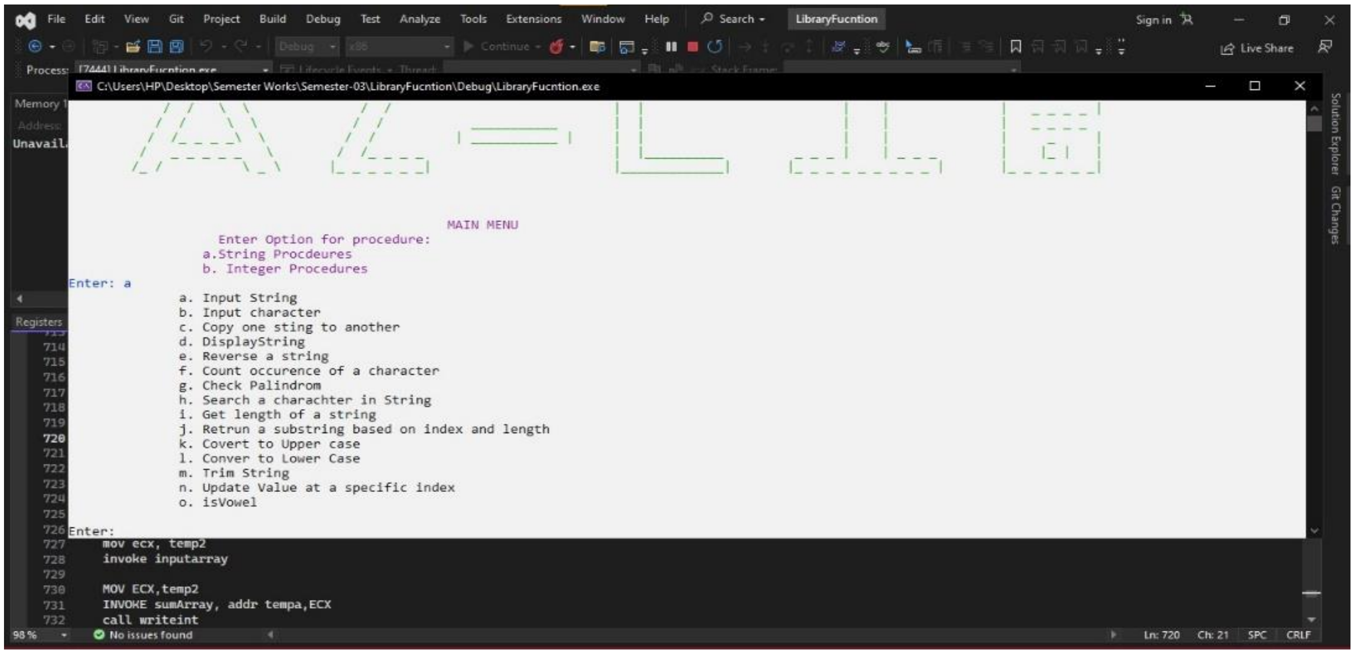
- String Procedures (Manipulation on characters)
- Integer Procedures (Manipulation on Numeric)



```
C:\Users\HP\Desktop\Semester Works\Semester-03\LibraryFucntion\Debug\LibraryFucntion.exe

                                        MAIN MENU
                 Enter Option for procedure:
               a.String Procdeures
               b. Integer Procedures
Enter:
```

```
    mov ecx, temp2
    invoke inputarray

    MOV ECX,temp2
    INVOKE sumArray, addr tempa,ECX
    call writeint
```

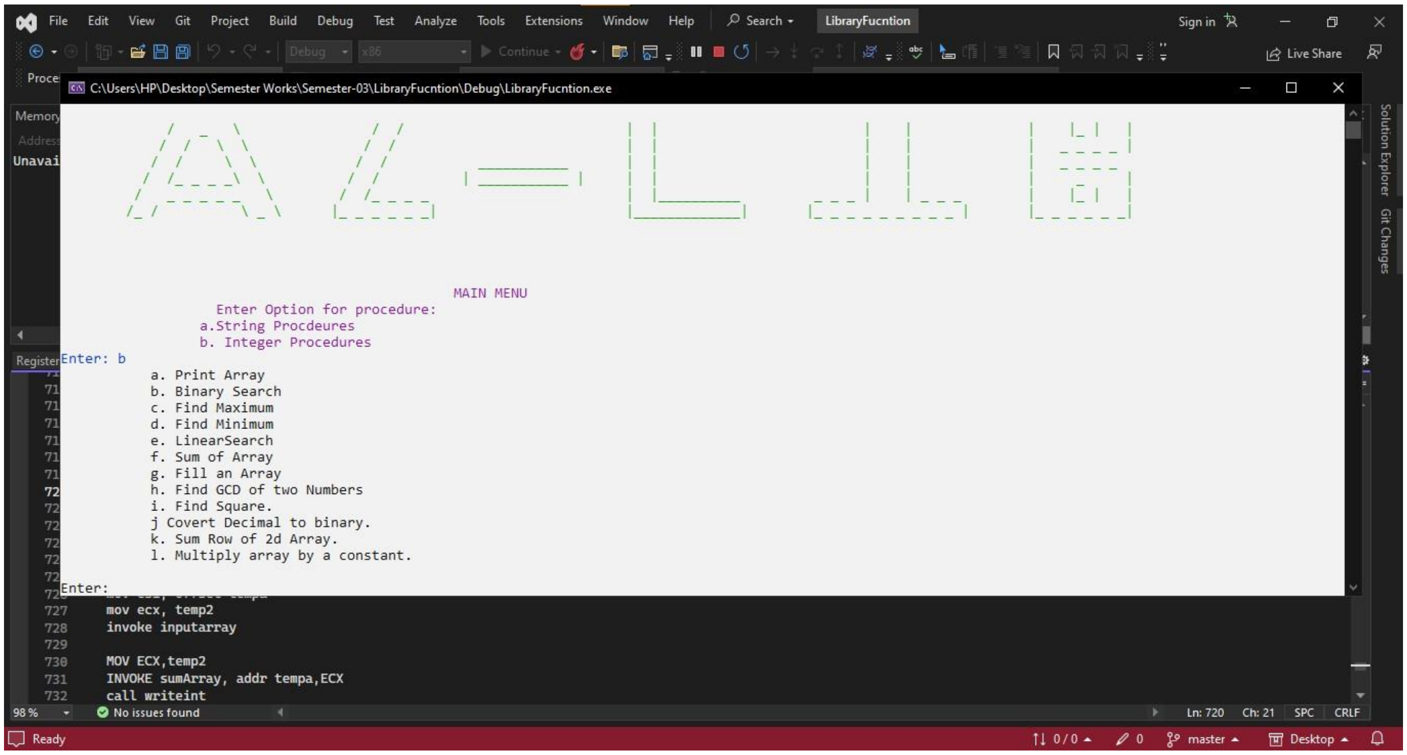
Figure 1: Main Menu



```
C:\Users\HP\Desktop\Semester Works\Semester-03\LibraryFucntion\Debug\LibraryFucntion.exe

                                        MAIN MENU
          Enter Option for procedure:
        a.String Procdeures
        b. Integer Procedures
Enter: a
               a. Input String
               b. Input character
               c. Copy one sting to another
               d. DisplayString
               e. Reverse a string
               f. Count occurence of a character
               g. Check Palindrom
               h. Search a charachter in String
               i. Get length of a string
               j. Retrun a substring based on index and length
               k. Covert to Upper case
               l. Conver to Lower Case
               m. Trim String
               n. Update Value at a specific index
               o. isVowel
Enter:
```

```
    mov ecx, temp2
    invoke inputarray

    MOV ECX,temp2
    INVOKE sumArray, addr tempa,ECX
    call writeint
```

Figure 2: String procedures



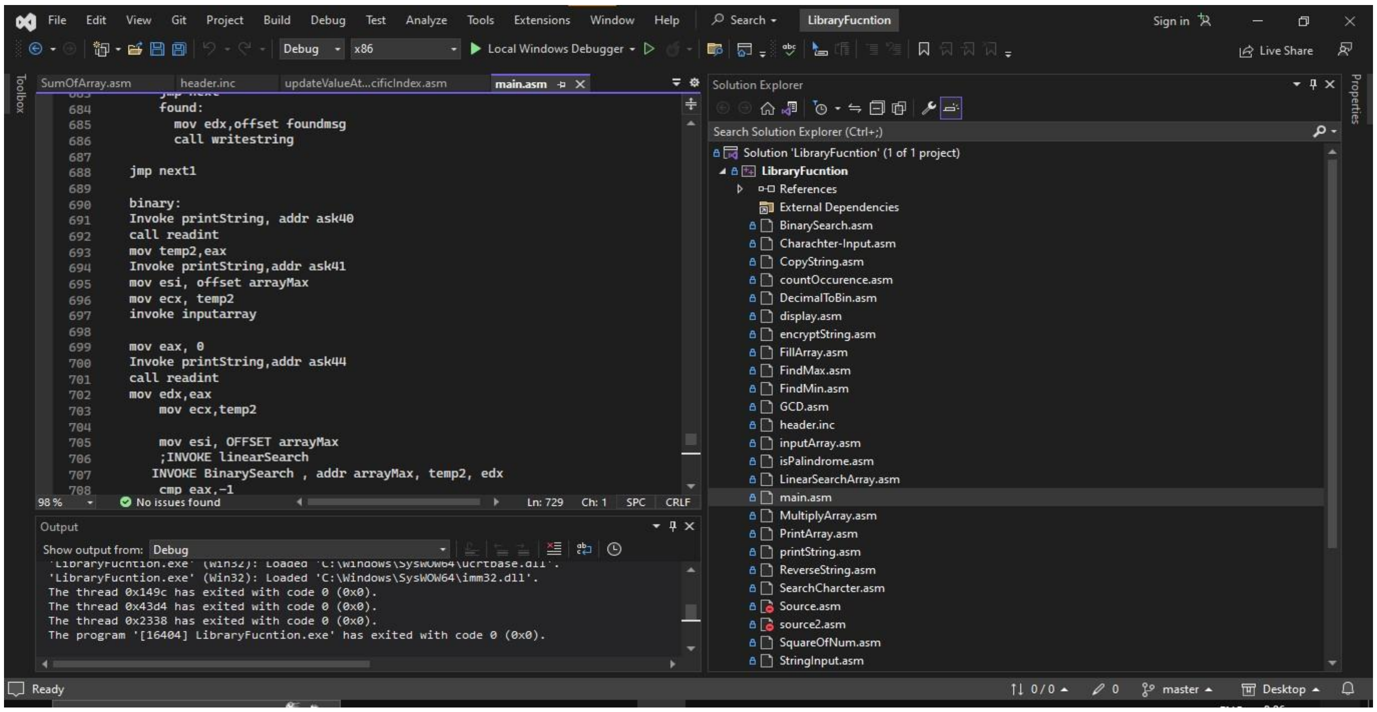
```
C:\Users\HP\Desktop\Semester Works\Semester-03\LibraryFucntion\Debug\LibraryFucntion.exe

                                        MAIN MENU
          Enter Option for procedure:
        a.String Procdeures
        b. Integer Procedures
Enter: b
            a. Print Array
            b. Binary Search
            c. Find Maximum
            d. Find Minimum
            e. LinearSearch
            f. Sum of Array
            g. Fill an Array
            h. Find GCD of two Numbers
            i. Find Square.
            j Covert Decimal to binary.
            k. Sum Row of 2d Array.
            l. Multiply array by a constant.
Enter:
```

```
    mov ecx, temp2
    invoke inputarray

    MOV ECX,temp2
    INVOKE sumArray, addr tempa,ECX
    call writeint
```

Figure 3: Integer procedures

Multi-Module

This project is created in multi module mode. The modularity of the project makes it easier for the user to actually understand the code and use it the application they are trying to build and increase their productivity. It increases the maintainability and sustainability of the code and can be aggregated into the user application files.



```
        jmp next
        found:
          mov edx,offset foundmsg
          call writestring

    jmp next1

    binary:
    Invoke printString, addr ask40
    call readint
    mov temp2,eax
    Invoke printString,addr ask41
    mov esi, offset arrayMax
    mov ecx, temp2
    invoke inputarray

    mov eax, 0
    Invoke printString,addr ask44
    call readint
    mov edx,eax
        mov ecx,temp2

        mov esi, OFFSET arrayMax
        ;INVOKE linearSearch
       INVOKE BinarySearch , addr arrayMax, temp2, edx
        cmp eax,-1
```

```
'LibraryFucntion.exe' (Win32): Loaded 'C:\Windows\SysWOW64\ucrtbase.dll'.
'LibraryFucntion.exe' (Win32): Loaded 'C:\Windows\SysWOW64\imm32.dll'.
The thread 0x149c has exited with code 0 (0x0).
The thread 0x43d4 has exited with code 0 (0x0).
The thread 0x2338 has exited with code 0 (0x0).
The program '[16404] LibraryFucntion.exe' has exited with code 0 (0x0).
```

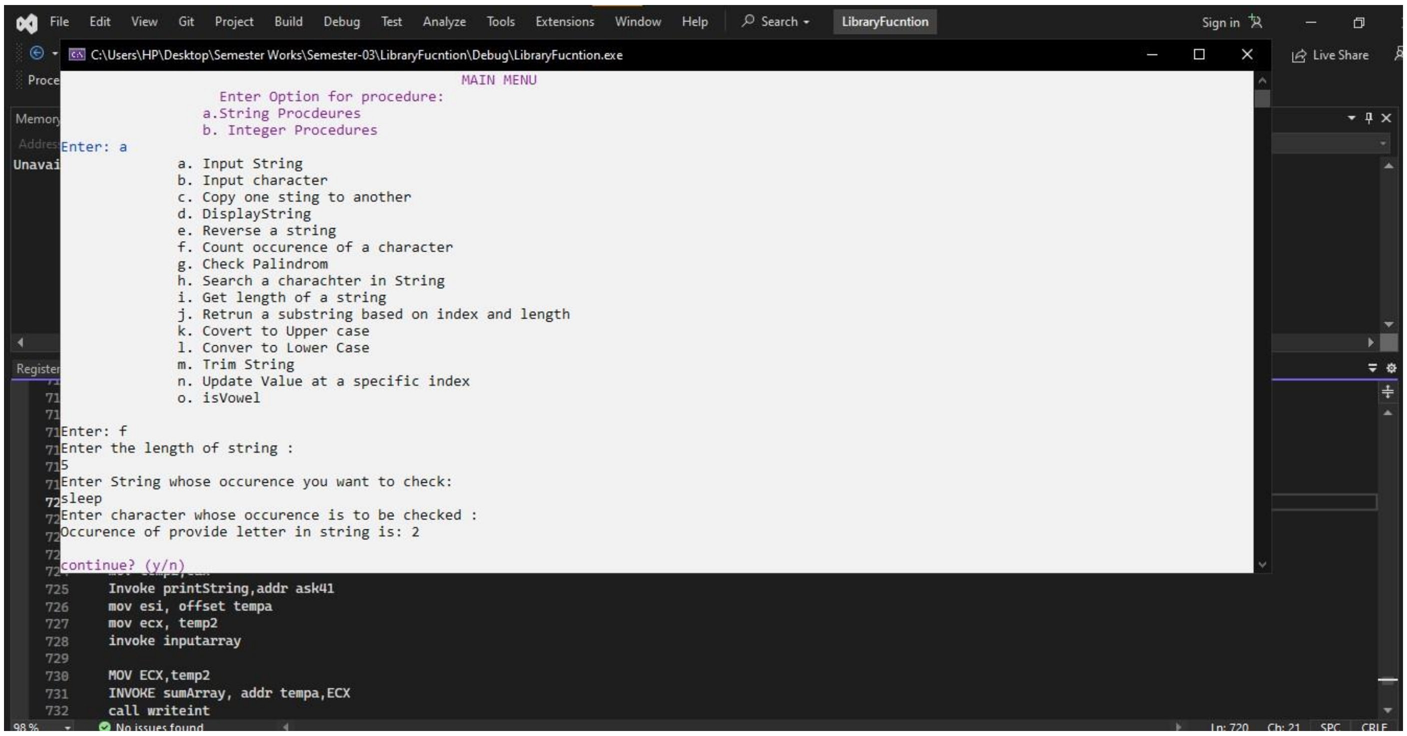
Figure 5: Multi Module

Procedures

Some of the procedure include:

- Greatest Common Divisor finder
- Checking if a string is palindrome or not
- Count the number of a specific character
- Get the length if user enter string
- Reverse a string
- Sum the integers in an array
- Find the minimum/ maximum from a given set of numbers

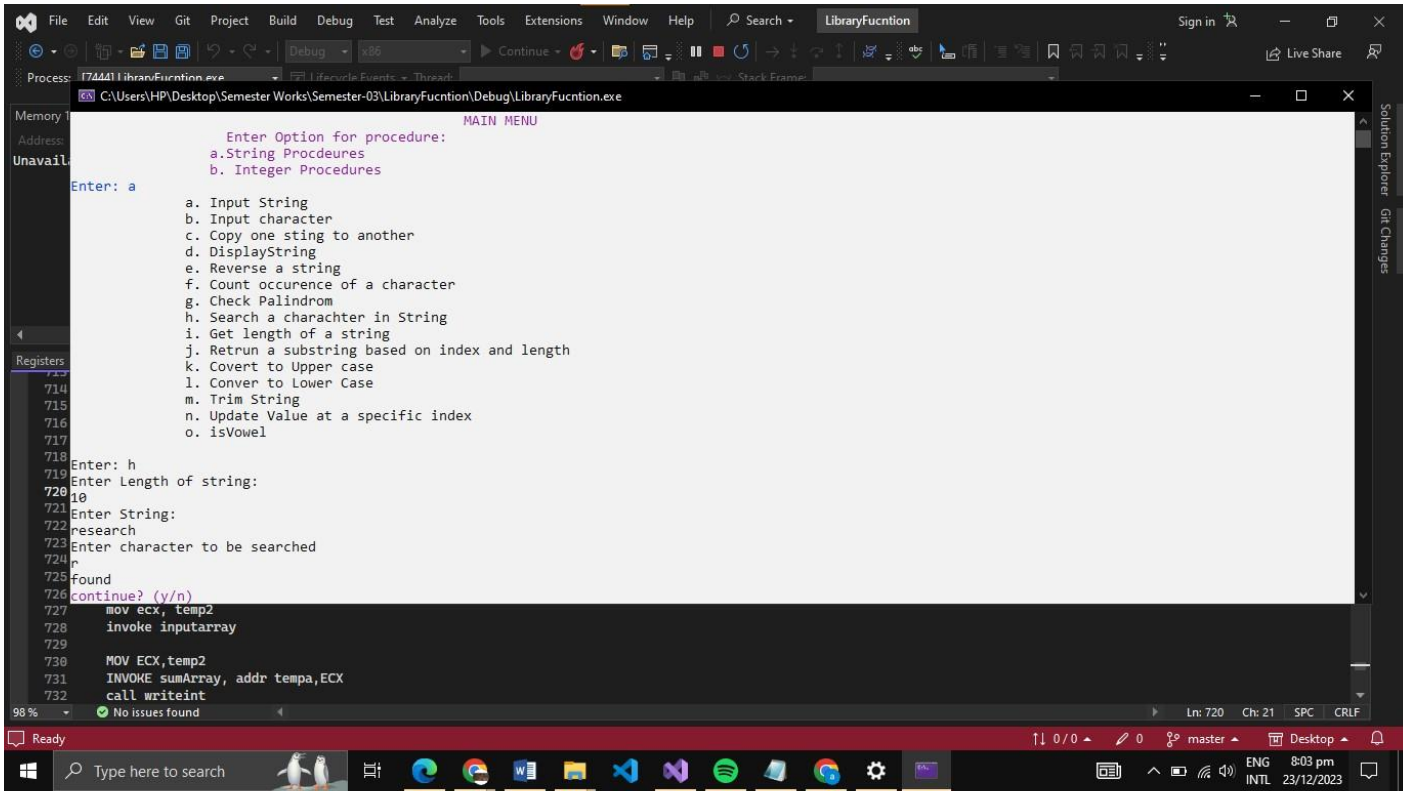
These are just some of the example of the procedures that have been written in this application in order to assist the programmers as well as the students. The separate procedure files for each function can be used by the students to study any of the functionalities that are provide in the project and benefit and have better understanding on how to write such small pieces of code into their own projects.



```
                                    MAIN MENU
          Enter Option for procedure:
        a.String Procdeures
        b. Integer Procedures
Enter: a
               a. Input String
               b. Input character
               c. Copy one sting to another
               d. DisplayString
               e. Reverse a string
               f. Count occurence of a character
               g. Check Palindrom
               h. Search a charachter in String
               i. Get length of a string
               j. Retrun a substring based on index and length
               k. Covert to Upper case
               l. Conver to Lower Case
               m. Trim String
               n. Update Value at a specific index
               o. isVowel

Enter: f
Enter the length of string :
5
Enter String whose occurence you want to check:
sleep
Enter character whose occurence is to be checked :
Occurence of provide letter in string is: 2

continue? (y/n)
```

```
    Invoke printString,addr ask41
    mov esi, offset tempa
    mov ecx, temp2
    invoke inputarray

    MOV ECX,temp2
    INVOKE sumArray, addr tempa,ECX
    call writeint
```

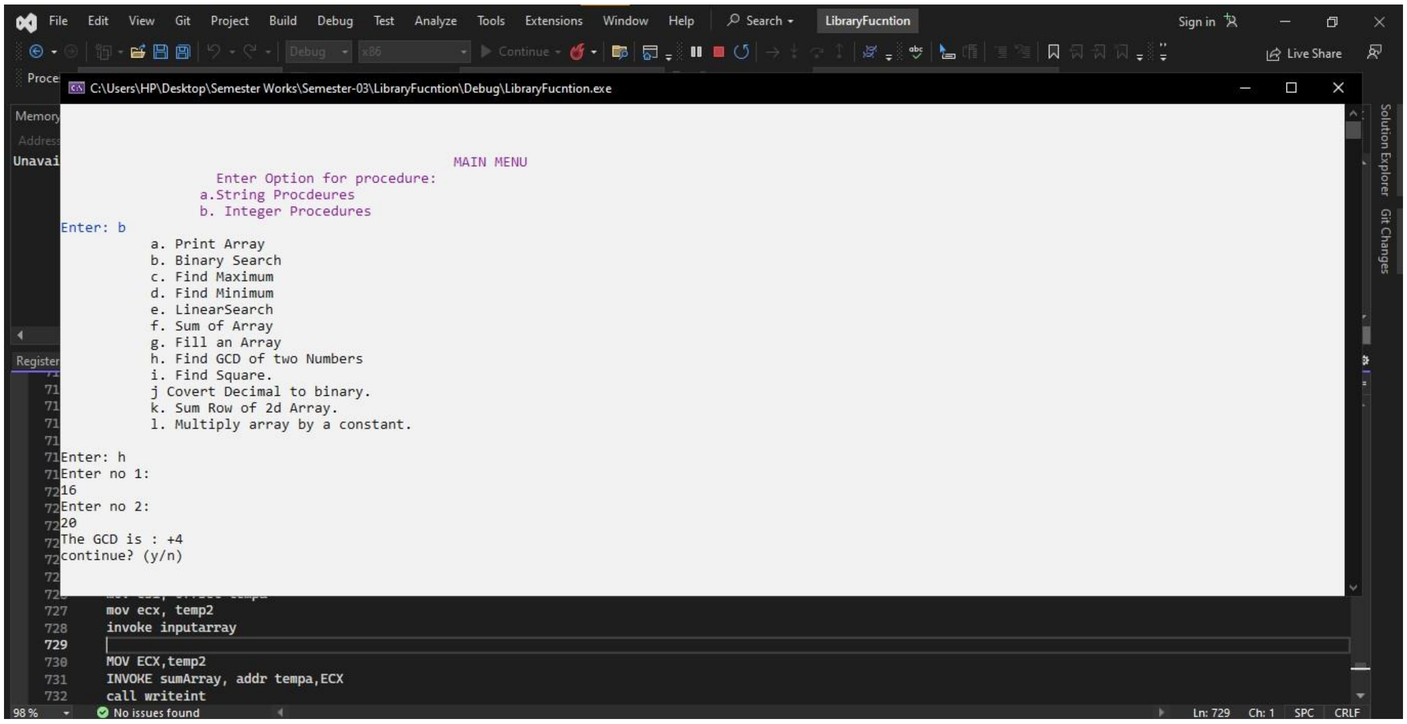
Figure 6: Occurrence of character



```
                                    MAIN MENU
          Enter Option for procedure:
        a.String Procdeures
        b. Integer Procedures
Enter: a
               a. Input String
               b. Input character
               c. Copy one sting to another
               d. DisplayString
               e. Reverse a string
               f. Count occurence of a character
               g. Check Palindrom
               h. Search a charachter in String
               i. Get length of a string
               j. Retrun a substring based on index and length
               k. Covert to Upper case
               l. Conver to Lower Case
               m. Trim String
               n. Update Value at a specific index
               o. isVowel

Enter: h
Enter Length of string:
10
Enter String:
research
Enter character to be searched
r
found
continue? (y/n)
```

```
    mov ecx, temp2
    invoke inputarray

    MOV ECX,temp2
    INVOKE sumArray, addr tempa,ECX
    call writeint
```

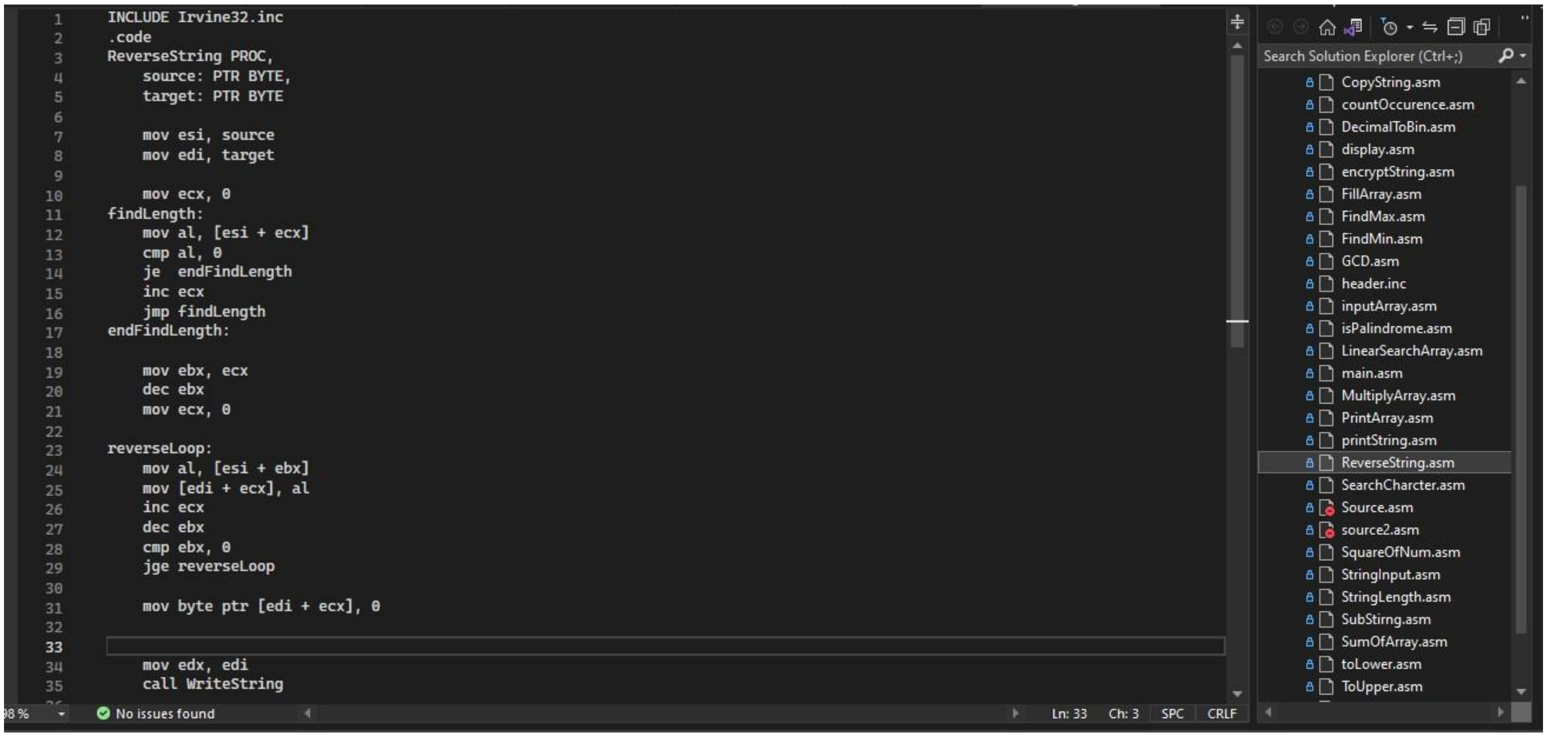
Figure 7: Searching character


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help Search LibraryFucntion
Sign in
Live Share
Continue
C:\Users\HP\Desktop\Semester Works\Semester-03\LibraryFucntion\Debug\LibraryFucntion.exe
No issues found
Ln: 729 Ch: 1 SPC CRLF

```
                                                        MAIN MENU
                 Enter Option for procedure:
               a.String Procdeures
               b. Integer Procedures
Enter: b
          a. Print Array
          b. Binary Search
          c. Find Maximum
          d. Find Minimum
          e. LinearSearch
          f. Sum of Array
          g. Fill an Array
          h. Find GCD of two Numbers
          i. Find Square.
          j Covert Decimal to binary.
          k. Sum Row of 2d Array.
          l. Multiply array by a constant.

Enter: h
Enter no 1:
16
Enter no 2:
20
The GCD is : +4
continue? (y/n)
```

```
    mov ecx, temp2
    invoke inputarray

    MOV ECX,temp2
    INVOKE sumArray, addr tempa,ECX
    call writeint
```

Figure 8: Greatest Common Divisor



```
INCLUDE Irvine32.inc
.code
ReverseString PROC,
    source: PTR BYTE,
    target: PTR BYTE

    mov esi, source
    mov edi, target

    mov ecx, 0
findLength:
    mov al, [esi + ecx]
    cmp al, 0
    je  endFindLength
    inc ecx
    jmp findLength
endFindLength:

    mov ebx, ecx
    dec ebx
    mov ecx, 0

reverseLoop:
    mov al, [esi + ebx]
    mov [edi + ecx], al
    inc ecx
    dec ebx
    cmp ebx, 0
    jge reverseLoop

    mov byte ptr [edi + ecx], 0


    mov edx, edi
    call WriteString
```

Figure 9: Source Code

Conclusion

Our project aimed to empower x86 assembly developers by enriching the Irvine32 Library with new procedures. These procedures specifically targeted challenges associated with data manipulation, especially strings and integers, making low-level programming more efficient and less error-prone. Recognizing the limitations of the Irvine32 Library for data manipulation, these additions empower developers, offering them new tools to tackle data-related tasks at the hardware level with greater flexibility and ease.

We utilized our resources like the 32-bit mode of assembly language on the specified architecture. It was implemented using Visual Studio 2022, incorporating the MASM assembler for x86 processors. As assembly language involves working with general-purpose registers, segment registers, an instruction pointer, and control/status flags, our project delved into the intricacies of these components.

This project shows our combined effort as a team to extend our skills towards increased efficiency of assembly and gain a better understanding of the low level machine language.

References

[1] Saxon, James A.; Plette, William S. (1962). Programming the IBM 1401, a self-instructional programmed manual. Englewood Cliffs, New Jersey, US: Prentice-Hall. LCCN 62-20615.

[2] https://www.investopedia.com/terms/a/assembly-language.asp.

[3] https://broman.dev/download/Assembly%20Language%20for%20x86%20Processors%207th%20Edition.pdf.

[4] https://www.wikihow.com/Use-MASM-in-Visual-Studio-2022.