Green University of Bangladesh

Department of Computer Science and Engineering (CSE)
Faculty of Sciences and Engineering (FSE)
Semester: (Fall, Year: 2024), B.Sc. in CSE (Day)

Assembly Based -Text Editor

Course Title: Microprocessors & Microcontrollers Lab
Course Code: CSE-304
Section: 222-D9

Students Details

Name	ID
Md. Kaised Mollick	221902070

Submission Date: 12.12.2024
Course Teacher's Name:Sagufta Sabah Nakshi

[For teachers use only: Don't write anything inside this box]

Project Report Status	
Marks:	**Signature:**
Comments:	**Date:**

Contents

Chapter 1

Introduction

1.1 Overview

The **"Assembly Based - Text Editor"** is a lightweight and efficient text editing application created entirely in assembly language for use with the 8086 microprocessor. This project focuses on providing basic text editing functionalities, such as text entry, editing, cursor navigation, and file saving, within a character-based user interface. The editor interacts directly with hardware through low-level operations, leveraging BIOS interrupts for display control and file I/O. Designed for educational and functional purposes, this project emphasizes memory efficiency, real-time feedback, and simplicity in design, demonstrating the practical application of assembly language in system-level programming. [1] [2] [3]

Figure 1.1: Text Editor Demo

1.2 Motivation

The motivation for this project comes from the desire to learn assembly language in a practical way. Most modern text editors are made using high-level programming languages, but this project gives us the chance to work with low - level language. By creating a text editor, I can apply my knowledge of computer architecture and low-level programming to build something useful. This project also helps me show how efficient and powerful assembly language can be. Additionally, it improves my problem-solving skills by addressing challenges like memory management and user input handling. It also prepares me for future projects that require low-level programming expertise.

1.2.1 Problem Statement

My project focuses on solving two main challenges in text editing using assembly language: Understanding how hardware works directly and Making a simple, efficient tool.
Most text editors today are created using high-level programming languages, which hide the details of how the computer works. This project fills that gap by making a basic text editor with assembly language. The text editor lets users do simple tasks like typing, editing, moving the cursor, and saving files. It shows how to use memory properly, take input from the keyboard, and control the screen directly using assembly instructions. This project showcases the power and efficiency of assembly language while helping users gain hands-on experience with low-level system programming concepts.

1.2.2 Complex Engineering Problem

The complexity of this project lies in efficiently managing text within a limited memory space while ensuring smooth interaction with hardware. Table 1.1 summarizes the attributes addressed in the project:

Table 1.1: Summary of the attributes touched by the mentioned project

Name of the P Attributes	**Explain how to address**
P1: Depth of knowledge required	A deep understanding of assembly language instructions such as MOV, INT 21h for input/output operations, and control structures like CMP, JMP, and LOOP.
P2: Range of conflicting requirements	Making the text editor simple to use while adding important features like inserting, deleting, and saving text.
P3: Depth of analysis required	Managing memory efficiently for text storage and performing low-level operations like navigating text, editing, and updating the display in real-time.
P4: Familiarity of issues	Dealing with user input errors and ensuring smooth movement through the text without crashes.
P5: Extent of applicable codes	Using 8086 commands for tasks like getting input, processing text, and showing the text on the screen.
P6: Extent of stakeholder involvement and conflicting requirements	Making the editor easy to use with helpful features,while considering hardware limitations and user expectations.
P7: Interdependence	Ensuring all parts of the program (input, processing, and display) work together without problems.

1.3 Design Goals/Objectives

The goals and objectives of the project are as follows:

1. **Performance Optimization:** Ensure smooth text input, editing, and rendering through efficient memory and processor utilization.
2. **Basic Functionality:** Provide core features like text entry, editing, navigation, and file saving.
3. **User-Friendly Interface:** Design a simple character-based interface for straightforward use.
4. **Error Handling:** Implement error messages for file I/O and invalid operations.
5. **Scalability:** Use modular code to allow future enhancements, such as search or formatting features.
6. **Memory Efficiency:** Manage text storage within a predefined memory matrix, ensuring effective use of limited resources.
7. **Low-Level Interaction:** Use BIOS interrupts for hardware-level control of display and file operations.

1.4 Application

The **"Assembly Based - Text Editor"** is designed to provide a simple and efficient tool for basic text editing tasks, utilizing assembly language for its development. This project demonstrates various applications, highlighting its potential in different scenarios:

1. **Learning Tool**: Helps beginners understand assembly language concepts through practical implementation.
2. **Low-Level Programming Example**: Showcases how assembly language can be used to build functional tools.
3. **Lightweight Text Editing**: Can be used on systems with limited resources, offering a minimalistic text editing solution.
4. **Educational Projects**: Serves as an example for students and professionals learning computer architecture and programming.
5. **System Diagnostics**: For system administrators working in low-tech settings, this can be used as a lightweight text editor.
6. **Embedded Systems**: Offers a framework for creating text-based applications with hardware limitations.

7. **Research and Development**: Serves as a foundation for investigating improvements in hardware and memory interaction methods.

8. **Minimalist Software Environments**: Suitable for situations requiring a straightforward, effective, and low-resource text editing solution, such as diagnostic tools or boot environments.

By providing a practical way to learn programming and offering a useful tool for low-resource environments, the **"Assembly Based - Text Editor"**connects what we learn in education with real-world use, making it both helpful and important..

Chapter 2

Design/Development/Implementation of the Project

2.1 Introduction

This **Assembly Based -Text Editor** project is designed to help understand assembly language programming. It focuses on basic concepts like taking input from the user, editing text, and saving it to a file. The project also involves using simple commands to manage text and understand how assembly works with computer hardware directly. It is a practical way to learn how a basic text editor operates at a low level. [4] [5].

2.2 Project Details

This project involves writing simple assembly code to handle text input, moving the cursor, and saving text to a file. The goal is to create a basic text editor where users can type, edit, and save their text. The editor uses assembly language instructions to manage each character of the text and control memory. It also uses BIOS interrupts to handle things like displaying text on the screen and saving files.

2.2.1 Text Input & Editing

The text editor allows the user to input and edit text. When the program starts, the user can type their text directly into the editor. The editor supports basic editing functions, such as moving the cursor, backspacing, and adding new characters. The user can navigate the text using the arrow keys or other controls.

- **Text Input:** Users can type characters one by one. Each character is displayed on the screen as it is typed.
- **Editing:** The user can delete text, move the cursor to different positions, and add new text.

- **Cursor Control:** The editor allows users to move the cursor up, down, left, or right within the text.

2.2.2 File Saving

After the user has finished writing the text, he can save it to the form. This file is stored in the folder where the EMU8086 software is installed. The editor uses a series of commands to perform file operations and write text to the file in a specific format. This program allows users to save their work and retrieve it later. In the installation folder for users to open and edit later.

- **Saving Text:** The program saves the text to a file by writing the characters stored in memory.
- **File Location:** The saved file is located in the folder where EMU8086 is installed, making it easily accessible for the user to open and edit later.

2.2.3 Basic User Interface

The user interface is simple and text-based, making it easy to understand and use. The editor provides a basic command line interface in which users can interact with the system by typing commands. The user interface consists of text input, cursor control, and file saving operations.

- **Text Display:** Text appears on the screen as user types, and the cursor position is updated in real time.
- **File Management:** After editing, users can save the text to a file or they can exit the editor without saving.

2.3 Implementation

This section describes how the text editor project was implemented, covering the workflow, tools used, and detailed implementation steps.

2.3.1 Workflow

The workflow of the text editor is very simple.Following steps are involved:

1. **User Starts The Program:** The user open the text editor through EMU8086 software. Display Main Menu:
2. **Display Main Menu:**Show the main menu on the screen and wait for user input to either start the program or exit.

3. **Start Program:** When the user starts the program, clear the screen and initialize the text editor for input.

4. **Text Input:** The user types text, and the characters are displayed on the screen.

5. **Text Editing:** The user can move the cursor and delete or add characters.

6. **Update Text:** Each typed character is saved in memory within the matrix array, and the screen shows the character at the cursor's position.

7. **Save Text:** If the user presses Ctrl+S, the program saves the text in matrix to a file named **KaiSed.txt**; if saving fails, an error message appears.

8. **Exit Program:** If the user presses ESC, the program exits without saving any changes.

9. **Error Handling:** If any error occurs during file-saving (like failure to create or write to the file), an error message **"Error !!! Could not save file."** is displayed.

2.3.2 Tools and Libraries

Tools and Libraries used to create the **Assembly Based -Text Editor** are given below :

1. **EMU8086:** This is the software used to write and run the assembly code. It helps work with 8086 assembly language in a simple way.

2. **BIOS Interrupts:** These are used to read input from the keyboard and display text on the screen.

3. **Assembly Instructions:** Simple assembly commands are used to input characters, move the cursor, and save text to a file.

2.3.3 Implementation Details

1. **Assembly Based -Text Editor** is made using assembly language to manage system tasks, input/output, and file storage. Here are the main parts of how the editor works:

2. **Assembly Language:** The project uses 8086 assembly language to directly work with the hardware. It handles tasks like typing text, moving the cursor, and saving files.

3. **Text Input and Display:** The editor helps users type text, which is stored in memory. Every letter typed is shown on the screen.

4. **Cursor Control:** The editor allows the user to move the cursor using BIOS interrupts. This helps the user go through the text to make changes or view it.

5. **Text Saving:** The text typed by the user is saved in a file using BIOS interrupts. The file is saved in the same folder where the EMU8086 software is installed, so users can open it later and continue editing.

6. **User Interaction:** The editor communicates with the user through a simple screen interface. The user can type, edit, and save text. Assembly instructions are used to handle the input, display, and saving of the text.

7. **Error Handling :** The program checks for mistakes, like incorrect input, and makes sure everything works smoothly. If the user types something wrong, the editor stops it, ensuring the data stays correct.

8. **Memory Management:** The editor uses memory to store the text typed by the user. It makes sure the memory is used well while typing and editing the text.

9. **Documentation and Logging:** The project includes comments in the assembly code to make it easier to understand and maintain. Although there isn't much logging in this project, it handles any errors to ensure everything works properly.

This process hepls to creates a simple, efficient, and easy-to-use text editors in assembly programming, like handling input/output, memory management, and cursor control.

2.4 Algorithms

The algorithms used in this project are simple.

- **Algorithm:** Saving Text to File
- **Input:** Text entered by the user
- **Output:** Text saved to a file

Algorithm 1: Assembly Based - Text Editor Algorithm

User key press (arrow keys, characters, etc.) Updated screen with text or cursor movement **Data:** Text matrix, current cursor position, file handler

```
/* Start Program:  Initialize the data segment and display the main
   menu                                                              */
```

Initialize the data segment and set up ‘ds‘ register

Call main menu to display the menu on the screen

```
/* Main Menu:  Wait for user input to start or exit                  */
```

while *User input is not ESC* **do**

  if *User presses Enter* **then**

    Call the ‘start$_{p}rog$‘*procedure to start text editor*

  if *User presses ESC* **then**

    Exit the program

```
/* Text Editor Setup:  Clear the screen and initialize variables */
```

Clear the screen by calling ‘clear$_{s}creen$‘*procedure Set cursor position to the start of the screen Initialize the text matrix and set start in*

```
/* Listen for User Input:  Continuously check for key presses    */
```

while *User input is not ESC* **do**

  if *Arrow key pressed (Up, Down, Left, Right)* **then**

    Call the corresponding procedure (‘moveUp‘, ‘moveDown‘, ‘moveLeft‘, ‘moveRight‘) to move the cursor

  if *Enter pressed* **then**

    Move to the next line (call ‘moveNewLine‘ procedure)

  if *Backspace pressed* **then**

    Delete the previous character and move cursor left (call ‘backSpace‘ procedure)

float

  if *Character typed* **then**

    Display the typed character at the current cursor position

    Update the matrix with the character at ‘curr$_{l}ine$‘*and*‘*curr$_{c}har$*‘ *Increment*‘*curr$_{c}har$*‘*and move the cursor right*

  if *Ctrl+S pressed* **then**

    Save the text to a file (call ‘saveToFile‘ procedure)

```
/* Text Update:  Update the matrix and screen with typed characters
   */
```

When a character is typed:

- Update the ‘matrix‘ at ‘curr$_{l}ine$‘*and*‘*curr$_{c}har$*‘*with the typed character Move the cursor position to the next spot*
- Update the screen with the new character.

```
/* Saving Text:  When Ctrl+S is pressed, save the content to a file
   */
```

if *Ctrl+S pressed* **then**

  Create a file using ‘int 21h, AH=3Ch‘ instruction

  Write the matrix content to the file using ‘int 21h, AH=40h‘ instruction

  If saving fails:

    - Display the error message ‘Error !!! Could not save file.”

```
/* Exit Program:  If ESC is pressed, exit the program                */
```

if *ESC pressed* **then**

  Exit the program by calling ‘int 20h‘

```
/* Error Handling:  If file creation or writing fails                */
```

if *File creation or writing fails* **then**

2.5 Flow-Chart

- **I added a Flow-chart to show how the program works.**

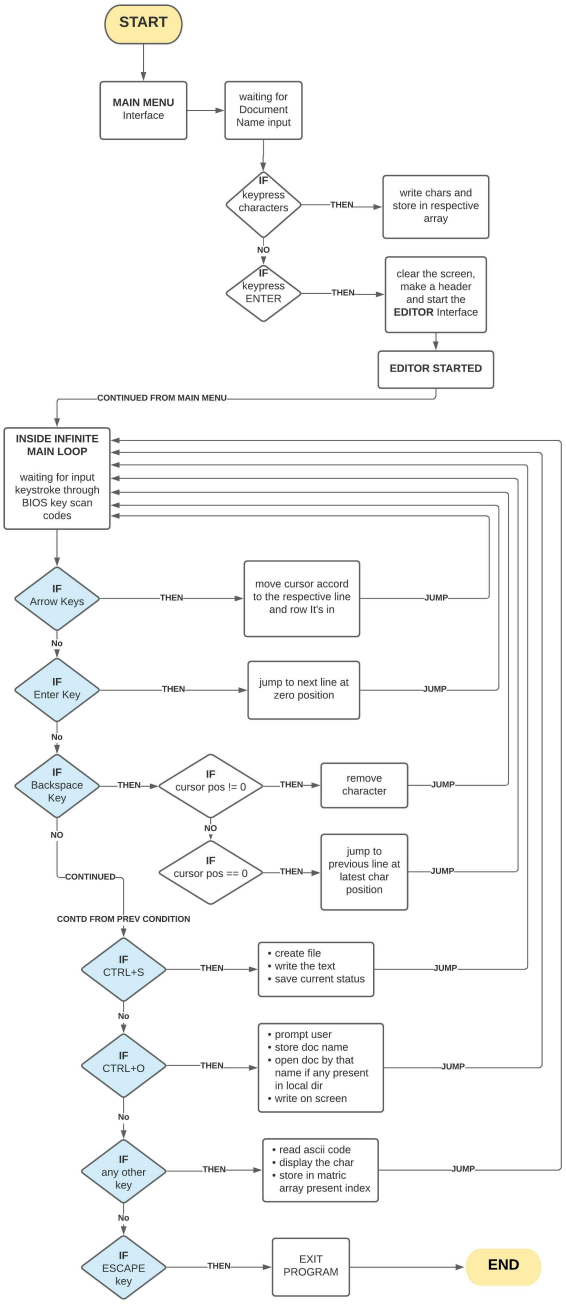

Figure 2.1: Flow-Chart For Assembly Based - Text Editor

2.6 Code Implementation

The code is implemented in Assembly Language and difference segment of our code are showing below : :

1. **The data segment contains all the variables and constant.**

```
.data
posX        db 1 dup(0)         ;posX :row
posY        db 1 dup(0)         ;posY :column
matrix      db 80*25 dup(' ')
curr_line   dw ?
curr_char   dw ?
color       db 2*16+15

filename db "KaiSed.txt",0   ;File path
handler dw ?
length dw ?

start_menu_str dw '  ',0ah,0dh

dw ' ',0ah,0dh
dw ' ',0ah,0dh
dw ' ',0ah,0dh
dw '                 ==================================================',0ah,0dh
dw '                 ||                                              ||',0ah,0dh
dw '                 ||      Welcome To Assembly Based - Text Editor ||',0ah,0dh
dw '                 ||--------------------------------------------- ||',0ah,0dh
dw '                 ||                                              ||',0ah,0dh
dw '                 ||                                              ||',0ah,0dh
dw '                 ||                                              ||',0ah,0dh
dw '                 ||          Please Type Text Here               ||',0ah,0dh
dw '                 ||       Press ESC To Exit The Program.         ||',0ah,0dh
dw '                 ||                                              ||',0ah,0dh
dw '                 ||            Press Enter To Start              ||',0ah,0dh
dw '                 ||                                              ||',0ah,0dh
dw '                 ==================================================',0ah,0dh
dw '$',0ah,0dh

error_msg db "Error !!! Could not save file.", 0Dh, 0Ah, '$'
```

Figure 2.2: Data Segment

2. **This segment initializes the data segment and calls the main menu.**

```
.code

    mov   ax,@data
    mov   ds,ax

    call main_menu

start_prog:
    call clear_screen
    jmp program
```

Figure 2.3: Code Segment Initialization

3. **The main menu displays the introduction and waits for the user's input.**

```
main_menu proc
    mov ah,09h
    mov dh,0
    mov dx, offset start_menu_str
    int 21h

    input:
        mov  ah, 0
        int  16h
        cmp  al, 27
        je   fin
        cmp  ax, 1C0Dh
        je   start_prog
        jmp input

main_menu endp
```

Figure 2.4: Main Menu Procedure

4. **Clears the screen before starting the program.**

```
clear_screen proc near
        mov ah,0
        mov al,3
        int 10h
        ret
clear_screen endp
```

Figure 2.5: Clear Screen Procedure

5. **Initializes the variables and sets up the starting cursor position.**

```
start_prog:
    call clear_screen
    jmp program

program:

    mov  curr_line, offset matrix
    mov  curr_char, 0
```

Figure 2.6: Program Initialization

6. **This section handles key presses like arrow keys, backspace, and characters**

```
any_char:
    mov   ah, 9
    mov   bh, 0
    mov   bl, color
    mov   cx, 1
    int   10h
```

Figure 2.7: Any Character Key Listener

```
129
130 backSpace:
131     cmp curr_char, 0
132     je  preventBackSpace
133
134     dec curr_char
135     mov si, curr_line
136     add si, curr_char
137     mov [ si ], ' '
138     dec length
139
140     dec posX
141     mov dl, posX
142
143     mov  ah, 2h
144     int  10h
145
146     mov  al,' '
147     mov  ah, 9
148     mov  bh, 0
149     mov  bl, 0000
150     mov  cx, 1
151     int  10h
152     jmp prntCrs
153
```

Figure 2.8: Backspace

7. **This section handles the detection of keyboard input and performs specific actions like cursor movement, character entry, or saving text based on the key pressed..**

```
219 read_char proc
220     mov   ah, 0
221     int   16h
222
223     cmp   al, 27            ; ESC
224     je    fin
225     cmp   ax, 4800h         ; Up
226     je    moveUp
227     cmp   ax, 4B00h         ; Left
228     je    moveLeft
229     cmp   ax, 4D00H         ; Rihht
230     je    moveRight
231     cmp   ax, 5000H         ; Down
232     je    moveDown
233     cmp   al, 0Dh           ; Enter
234     je    moveNewLine
235     cmp   ax, 4700H         ; Home
236     je    moveToBeginning
237     cmp   al, 08h           ; Backspace
238     je    backSpace
239     cmp   al, 13h           ; Ctrl+S
240     je    saveToFile
241
242     jmp   any_char
243 read_char endp
```

Figure 2.9: Keyboard Input Handler

8. **Creating File, writing, and saving the text data.**

```
saveToFile:

    mov   ah, 3Ch
    mov   cx, 0
    mov   dx, offset filename
    int   21h

    jc    save_error
    mov   handler, ax


    mov   ah, 40h
    mov   bx, handler
    mov   cx, length
    mov   dx, offset matrix
    int   21h
    jc    save_error

    mov   ah, 3Eh
    mov   bx, handler
    int   21h
    jmp   fin

save_error:
    mov   dx, offset error_msg
    mov   ah, 09h
    int   21h
    jmp   fin
```

Figure 2.10: File Handling

9. **Cursor Movement : Manages the cursor movement for different directions.**

```
moveRight:
    inc   curr_char
    mov   dl, posX
    mov   dh, posY
    inc   dl
    mov   posX, dl
    jmp   prntCrs
```

Figure 2.11: Move Right

```
moveLeft:
    dec  curr_char
    mov  dl, posX
    mov  dh, posY
    dec  dl
    mov  posX, dl
    jmp  prntCrs
```

Figure 2.12: Move Left

```
moveUp:
    sub  curr_line, 80
    mov  dl, posX
    mov  dh, posY
    dec  dh
    mov  posY, dh
    jmp  prntCrs
```

Figure 2.13: Move Up

```
moveDown:
    add  curr_line, 80
    mov  dl, posX
    mov  dh, posY
    inc  dh
    mov  posY, dh
    jmp  prntCrs
```

Figure 2.14: Move Down

```
moveToBeginning:
    mov curr_char, 0
    mov posX, 0
    mov dl, posX
    jmp prntCrs
```

Figure 2.15: Move To Beginning

```
moveNewLine:
    mov si, curr_line
    add si, 79
    mov [si], 0dh
    add curr_line, 80
    mov curr_char, 0
    mov posX, 0
    mov dl, posX
    mov dh, posY
    inc dh
    mov posY, dh
    add length, 80
    jmp prntCrs
```

Figure 2.16: Move Newline

Chapter 3

Performance Evaluation

3.1 Simulation Environment/ Simulation Procedure

Assembly Based -Text Editor was created using 8086 assembly language and tested with the EMU8086 emulator. This setup helped run and check how the editor works.

3.1.1 Setup and Testing Procedure

The project was implemented using 8086 assembly language and tested on the EMU8086 emulator. The EMU8086 environment provides an easy way to debug and execute assembly code by simulating the 8086 microprocessor's character.

- **Key Steps Of Setup:**

1. **Installing EMU8086 Emulator:** The software was installed on a Windows PC, providing a virtual environment for testing assembly code.
2. **Code Implementation:** The text editor code was written in EMU8086 software. This included creating functions for typing, cursor movement, and file saving.
3. **Testing Environment:** The program was tested by simulating like typing text, moving with the cursor, and saving a file.

3.2 Results Analysis/Testing

3.2.1 First Interface

- **When anyone run this project this interface will show first.**

Figure 3.1: First Interface

3.2.2 Input Text

- **Characters were entered and displayed on the screen, confirming that the input and memory storage (matrix) worked correctly.**

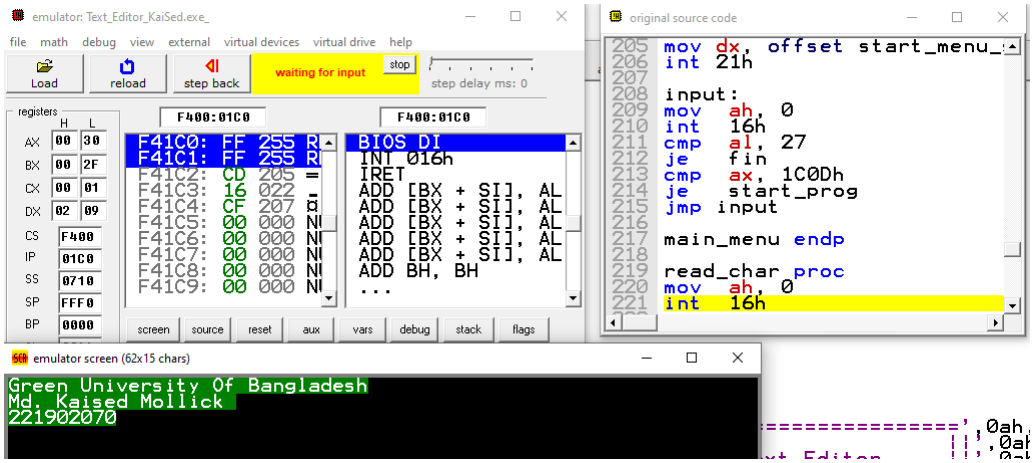

Figure 3.2: Typing Text

3.2.3 Cursor Movement

- **The arrow keys successfully moved the cursor up, down, left, and right. Entering text at new lines also worked as expected.**

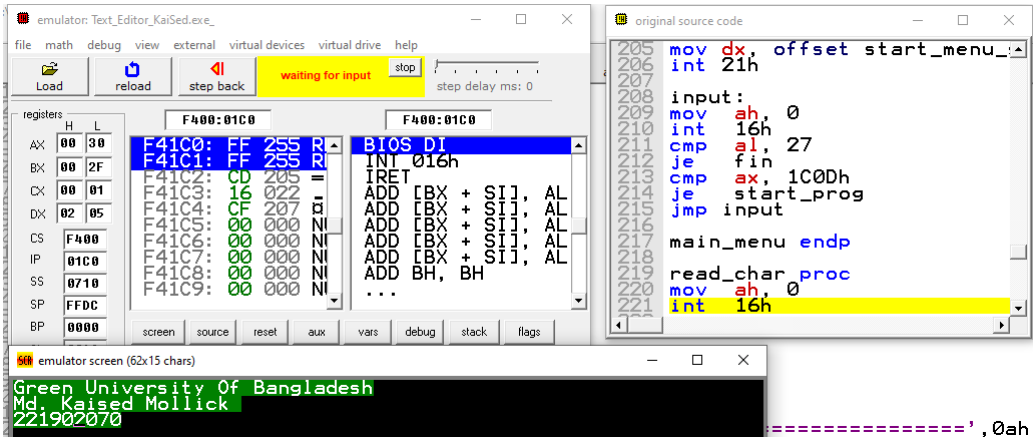

Figure 3.3: Cursor Movement

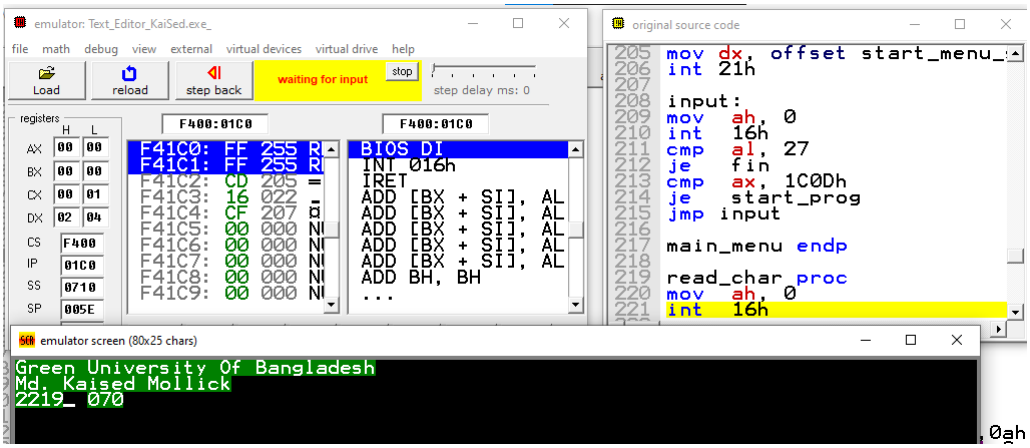

Figure 3.4: Cursor Movement & Delete Text

3.2.4 Backspace Functionality

- **The backspace key removed characters correctly and moved the cursor back one position.**

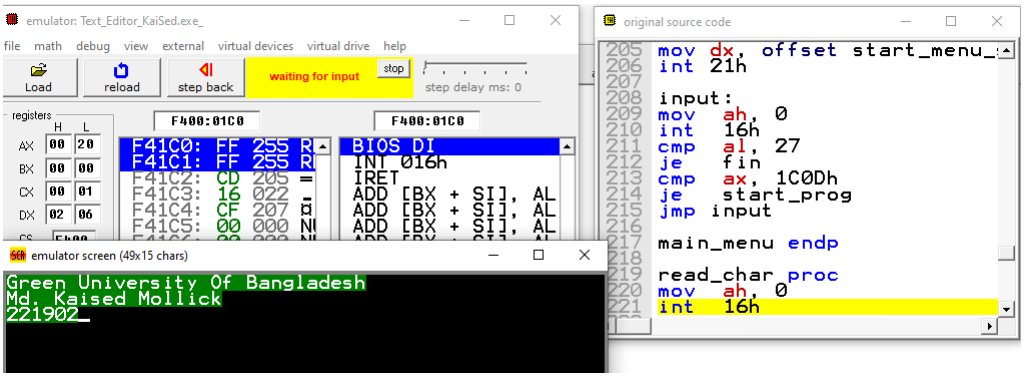

Figure 3.5: Showing Backspace Work

3.2.5 File Saving

- **If the user presses Ctrl+S, The editor successfully saved typed text to a file named *KaiSed.txt* on the computer**

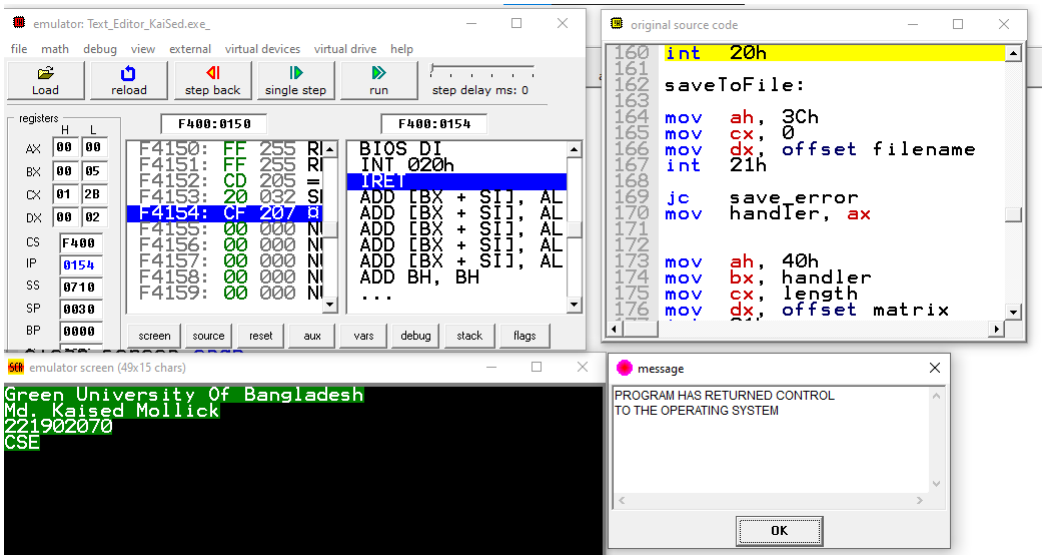

Figure 3.6: Save File

3.2.6 Path for Saving a File

- **The saved file is located in the folder where EMU8086 is installed. The saved file's name is *KaiSed*, as shown in the snapshot.**

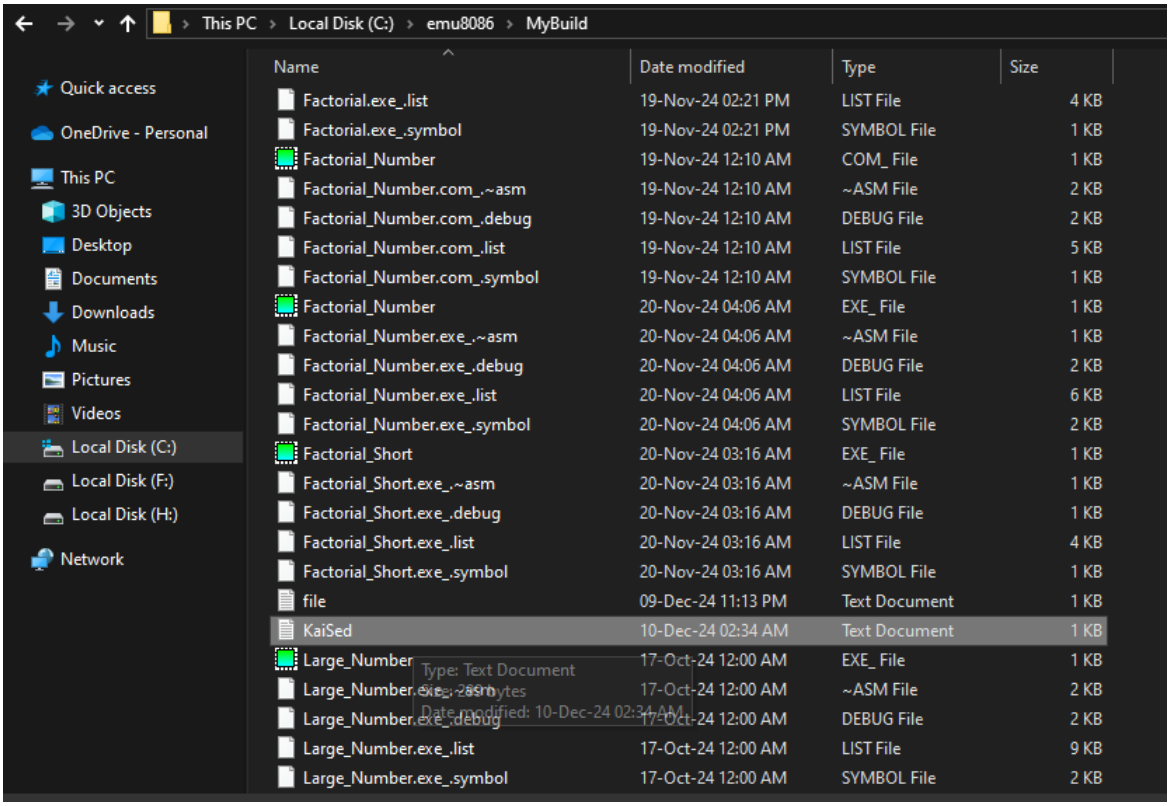

Figure 3.7: Path for Saved File

3.2.7 Viewing the Final Saved Text File

- **Here is the final text file that I saved which name is *KaiSed.txt***

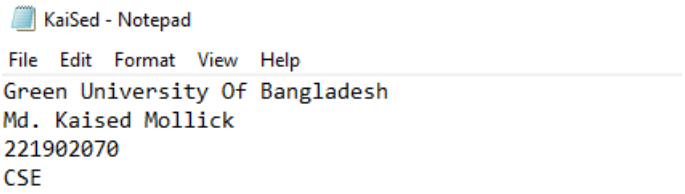

Figure 3.8: Final Saved Text File

3.3 Results Overall Discussion

This section explains how the Text Editor worked and the problems found during testing.

3.3.1 How It Worked

- **Typing:** Users typed letters that appeared on the screen and were saved in memory.
- **Cursor Movement:** The cursor moved with arrow keys and shifted to a new line when Enter was pressed.
- **Backspace:** The backspace key deleted the last character and moved the cursor back.
- **Saving:** Users could save their text to a file.

3.3.2 Problems Find

- **Backspace Problem :** The backspace didn't work at the start of a line.
- **Screen Limit:** Only 25 lines and 80 characters per line were allowed
- **Saving Errors:** If saving failed, the editor showed an error but didn't let users try again.

The **Assembly Based -Text Editor** worked well for basic tasks. It showed how assembly language can be used to directly with the computer's hardware. Although there were some problems, the project was a good way to learn about low-level programming and how to build simple tools.

3.3.3 Complex Engineering Problem Discussion

Making **Assembly Based -Text Editor** faced several challenges. Key challenges included handling cursor movement efficiently, managing memory for text storage, and implementing file-saving functionality within the limitations of the 8086 microprocessor. These challenges required careful attention to low-level programming techniques, memory management, and hardware interaction to create a functional and user-friendly text editing solution.

Chapter 4

Conclusion

4.1 Discussion

This Section explains how the **Assembly Based -Text Editor** worked, the results from testing, and what was observed. The project allowed users to type text, move the cursor, delete characters, and save their work to a file. It also showed how assembly language can be used to create simple tools by directly interacting with hardware. During testing, some issues were found, such as problems with backspace at the start of a line and limits on text size. Despite these challenges, the project successfully demonstrated basic text editing functions and provided valuable learning about low-level programming.

4.2 Limitations

The project has some limitations that became find out during testing:

- **Screen Space:** The text editor is limited to 25 lines with 80 characters per line, which is not enough for working with larger texts.
- **Backspace Issue:** The backspace key doesn't work correctly when the cursor is at the start of a line.
- **Saving Errors:** If there is an error while saving the file, like not enough disk space, the program shows an error but doesn't give an option to fix it or retry.
- **Lack of Features:** The editor doesn't have common features like copy-paste or undo, which makes it less useful for more advanced text editing.

These limitations show that while the text editor works for basic tasks, it is not suitable for more complex text editing needs. However, it is a valuable learning project that helps understand assembly programming and basic system design.

4.3 Scope of Future Work

In future, there are a few ways to improve the **Assembly Based -Text Editor** project:

1. **Increase Text Capacity:** Right now, the editor can only handle 25 lines and 80 characters per line. We can increase this limit so users can work with longer documents.
2. **Fix Backspace Issue:** The backspace key doesn't work correctly at the beginning of a line. This problem can be fixed to make the editor more reliable.
3. **Add More Features:** We can add useful tools like copy-paste, undo, and redo, which will make the editor more practical to use.
4. **Better Error Handling:** The save feature can be improved by allowing users to try saving again if there's an error, like not having enough space.
5. **User Interface:** Adding a simple interface or menu can make the editor easier to use for people who aren't familiar with text editors in assembly.

These changes would make the text editor more useful and enjoyable to work with.

References

[1] Sagufta Sabah Nakshi. Lecture on microprocessors & microcontrollers lab, 2024. Class Lecture.

[2] Emu8086 tutorial. https://www.youtube.com/watch?v=6uvO04_qitY&list=PLirRGafa75rRsvLWDJue0CkQRXKXJKxpX. YouTube Tutorial Series.

[3] Emu8086 tutorial. https://www.youtube.com/watch?v=VwN91x5i25g&list=PLBlnK6fEyqRgMCUAG0XRw78UA8qnv6jEx. YouTube Tutorial Series.

[4] Microprocessor 8086 overview. https://www.tutorialspoint.com/microprocessor/microprocessor_8086_overview.htm. Online Resource.

[5] Douglas V. Hall. *Microprocessor Programming and Applications with the 8086/8088*. Prentice Hall, New Jersey, 5th edition, 2024.