Green University of Bangladesh

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

Bank Management System

Course Title: Microprocessors Microcontrollers Lab
Course Code: CSE 304
Section: 222 D6

Students Details

Name	ID
Md Syful Islam	222002111

Submission Date: 13 December 2024
Course Teacher's Name: Mr. Md Nazmus Shakib

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

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

Contents

1 Introduction **3**

1.1 Overview 3

1.2 Motivation 3

1.3 Problem Definition 3

1.3.1 Problem Statement 3

1.3.2 Complex Engineering Problem 3

1.4 Design Goals/Objectives 4

1.5 Application 4

2 Design/Development/Implementation of the Project **6**

2.1 Introduction 6

2.2 Project Details 6

2.2.1 Banking System Functionality 7

2.2.2 System Architecture 7

2.3 Implementation 7

2.3.1 Workflow 7

2.3.2 Tools and libraries 7

2.3.3 Implementation details 8

3 Performance Evaluation **14**

3.1 Simulation Environment/ Simulation Procedure 14

3.2 Results Analysis/Testing 14

3.2.1 Create Account 14

3.2.2 Deposit money 14

3.2.3 Withdraw Money 16

3.3 Results Overall Discussion 16

4 Conclusion **17**

4.1 Discussion 17
4.2 Limitations 17
4.3 Scope of Future Work 18

Chapter 1

Introduction

1.1 Overview

This project simulates a banking system for "Green Bank Ltd" using assembly language on the 8086 emulator. The system provides key functionalities such as creating accounts, displaying account details, depositing money, and withdrawing money. It offers an efficient way to understand basic banking operations through a procedural approach in assembly language.

1.2 Motivation

The motivation for this project stems from the need to explore how low-level programming languages, like assembly, can implement real-world systems. Additionally, working with the emulator 8086 allows gaining a deeper understanding of microprocessor operations and memory management in the context of a banking system.

1.3 Problem Definition

1.3.1 Problem Statement

Banking systems are complex, requiring secure and efficient operations. Implementing such a system in assembly language demonstrates how to create a streamlined and low-resource-consuming program for essential banking tasks.

1.3.2 Complex Engineering Problem

The following Table 1.1 outlines key attributes related to addressing a complex engineering problem in the context of an assembly language-based banking system. Each attribute is linked to an explanation of how to approach the challenge:

Table 1.1: Summary of the attributes touched in this projects

Name of the P Attributess	Explain how to address
P1: Depth of knowledge required	Understanding assembly language operations and banking system logic.
P2: Range of conflicting requirements	Balancing system functionality and memory constraints.
P3: Depth of analysis required	Analysis of banking transaction workflows in assembly.
P4: Familiarity of issues	Familiarity with low-level assembly language operations is required.
P5: Extent of applicable codes	Follow 8086 assembly language syntax and rules.

1.4 Design Goals/Objectives

The design of the assembly language-based banking system focuses on creating an efficient, reliable, and user-friendly platform to simulate banking operations. The objectives for the project are as follows:

- **Efficient Memory Usage:** Design the system to use the limited memory available in the emulator 8086 efficiently.

- **Implement Core Banking Operations:** Develop and implement core banking features such as account creation, viewing account details, deposits, withdrawals, and balance management.

- **Transaction Security:** Implement basic security for transactions such as checking for sufficient balance before allowing withdrawals.

- **Scalability:** Ensure that the system can be easily extended to accommodate more features, such as loan management, interest calculations, or additional transaction types.

These design goals provide a clear direction for the development of the assembly-based banking system in emulator 8086 environment.

1.5 Application

This assembly language-based banking system can be used as a simulation tool for understanding basic banking operations like account management and transactions in low-level programming environments. It serves as an educational resource for students learning assembly language and system programming, demonstrating how complex systems can be built with limited resources. The system's principles can also be extended to security applications like PIN verification, transaction processing systems, and low-resource financial platforms, making it suitable for real-time and optimized financial applications.

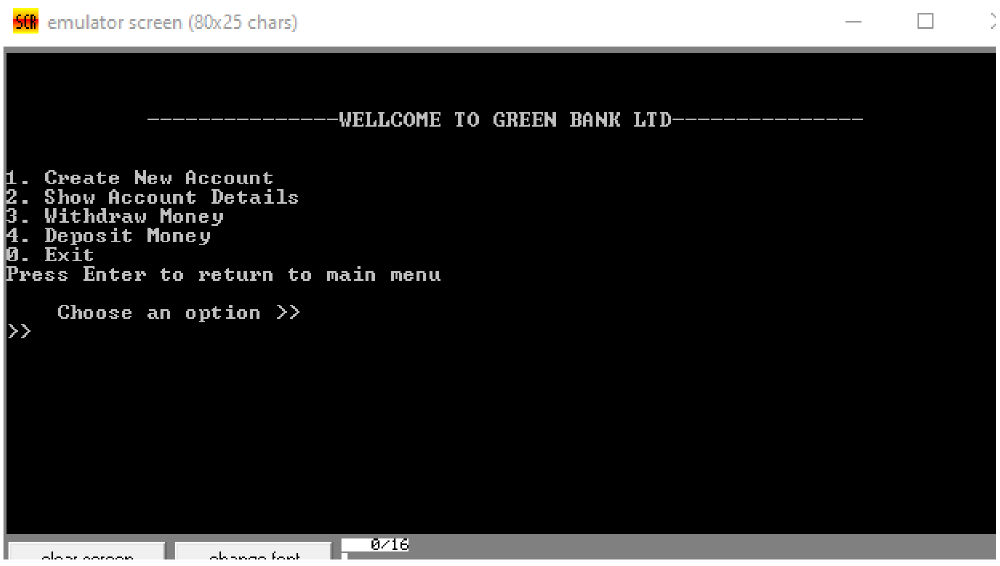
Chapter 2

Design/Development/Implementation of the Project

2.1 Introduction

This chapter provides an overview of the design, development, and implementation of the banking system project developed in assembly language for the emulator 8086. The project focuses on simulating banking operations such as account creation, deposits, withdrawals, and balance checking. By utilizing low-level programming techniques, it aims to provide a deep understanding of assembly language operations and their application in real-world systems.

2.2 Project Details



```
          ---------------WELLCOME TO GREEN BANK LTD---------------

1. Create New Account
2. Show Account Details
3. Withdraw Money
4. Deposit Money
0. Exit
Press Enter to return to main menu

    Choose an option >>
>>
```

Figure 2.1: Main menu of the project

This section elaborates on the core details of the project, including its functionality, design considerations, and system architecture.The figure includes modules for Create account, Show account Details, Withdraw money, Deposit money and Exit. Each module is designed to address specific aspects of Banking operations.

2.2.1 Banking System Functionality

The system provides basic banking operations such as creating accounts, depositing and withdrawing money, and displaying account details. It supports multiple users and manages their accounts with individual details.

2.2.2 System Architecture

The project operates in a sequential manner, using assembly instructions to manage the flow of information between different sections such as account creation, transaction processing, and balance retrieval. The memory usage is optimized to fit within the constraints of the emulator 8086.

2.3 Implementation

2.3.1 Workflow

The workflow of the banking system starts with the user selecting an operation from the menu, such as creating a new account or checking the balance. The system then processes the user's request by performing operations like storing data in memory, validating input, and updating the system state like account balances.

2.3.2 Tools and libraries

The following tools and technologies will be used to implement the project:

- **Assembly Language (8086):** For coding the bank management system.
- **emu8086 Emulator:** To write, run, and debug the assembly code.
- **BIOS Interrupts (e.g., INT 21h):** For user input/output operations and data handling.
- **Memory Management Techniques:** To handle data storage and retrieval during different banking operations.

2.3.3 Implementation details

In this sections we will show the part of some functionalities that I have used in this project to implement this system.

Main Procedure

Here is the functionalities for Main Procedure of this project.

Create Account

Here is the part of functionalities for Creating an account.

Show Account Details

Here is the part of functionalities for show the details of any created account.

Deposit Money

Here is the part of functionalities for deposit money in account.

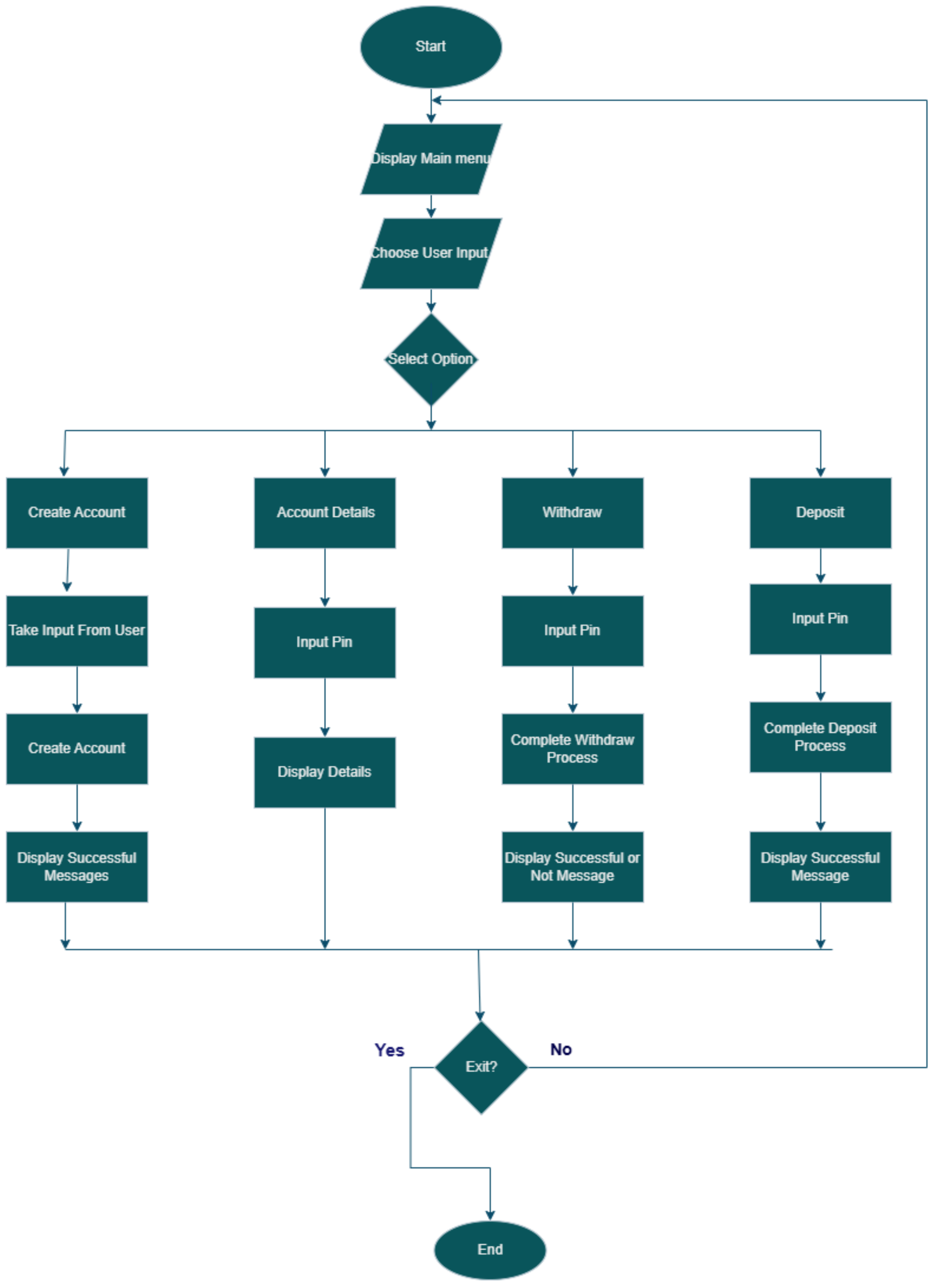
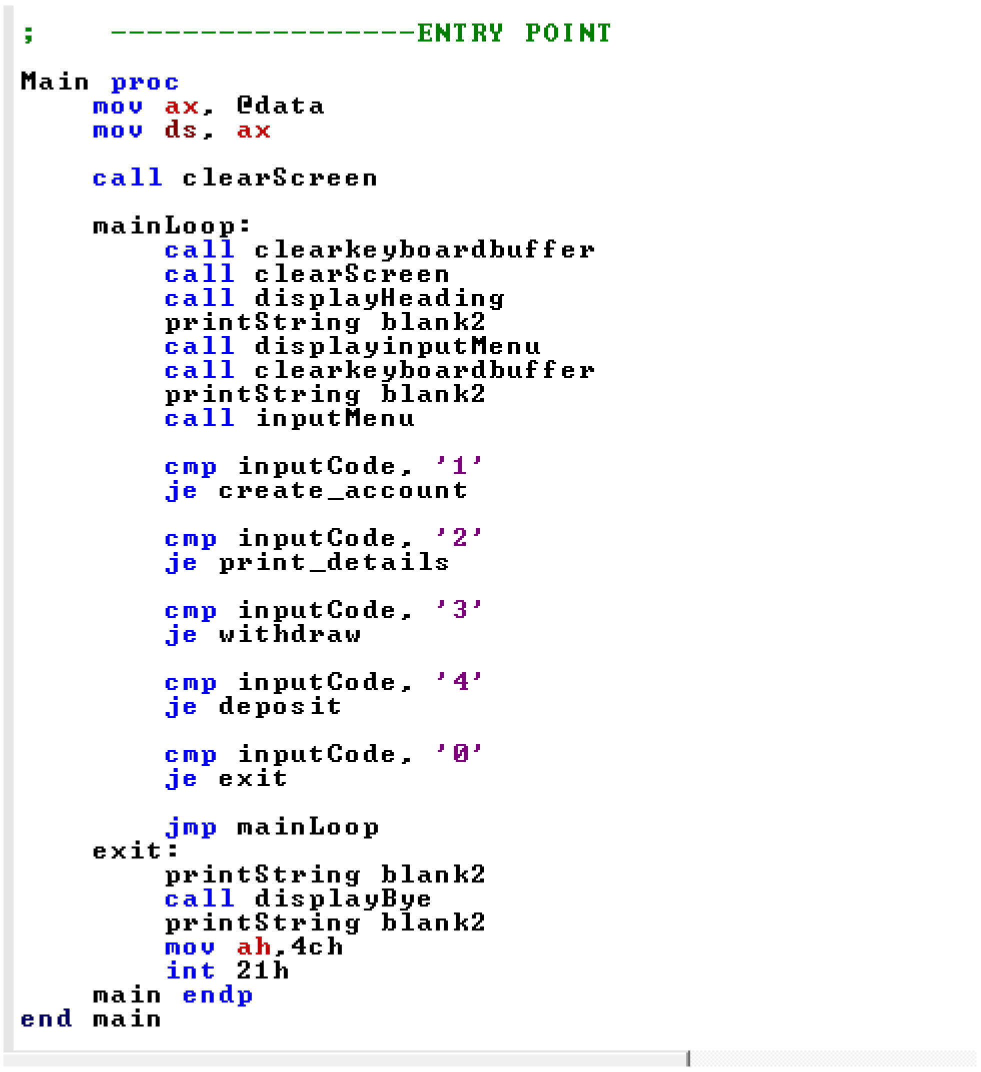
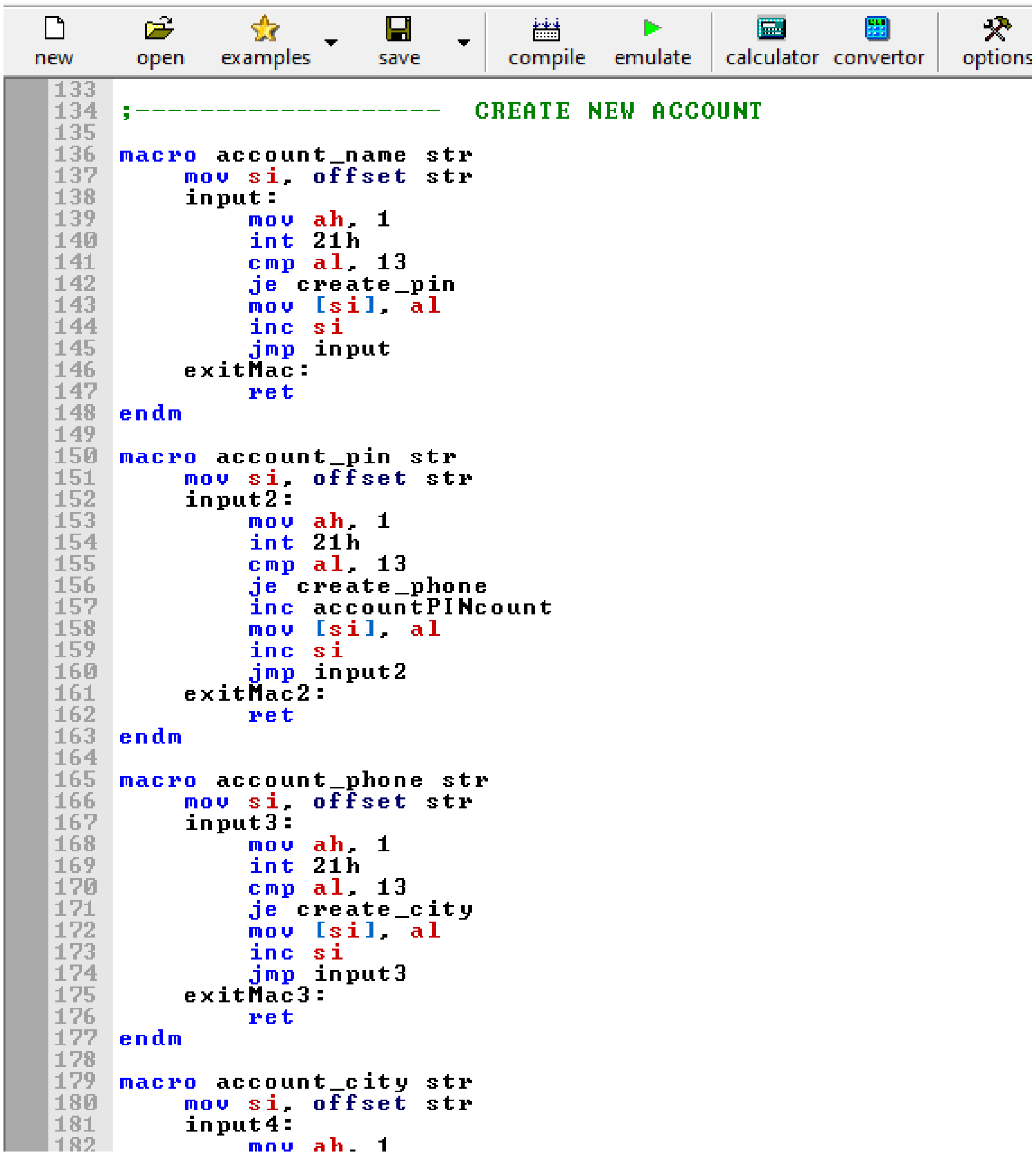


Figure 2.2: Flowchart Diagram Of this Project



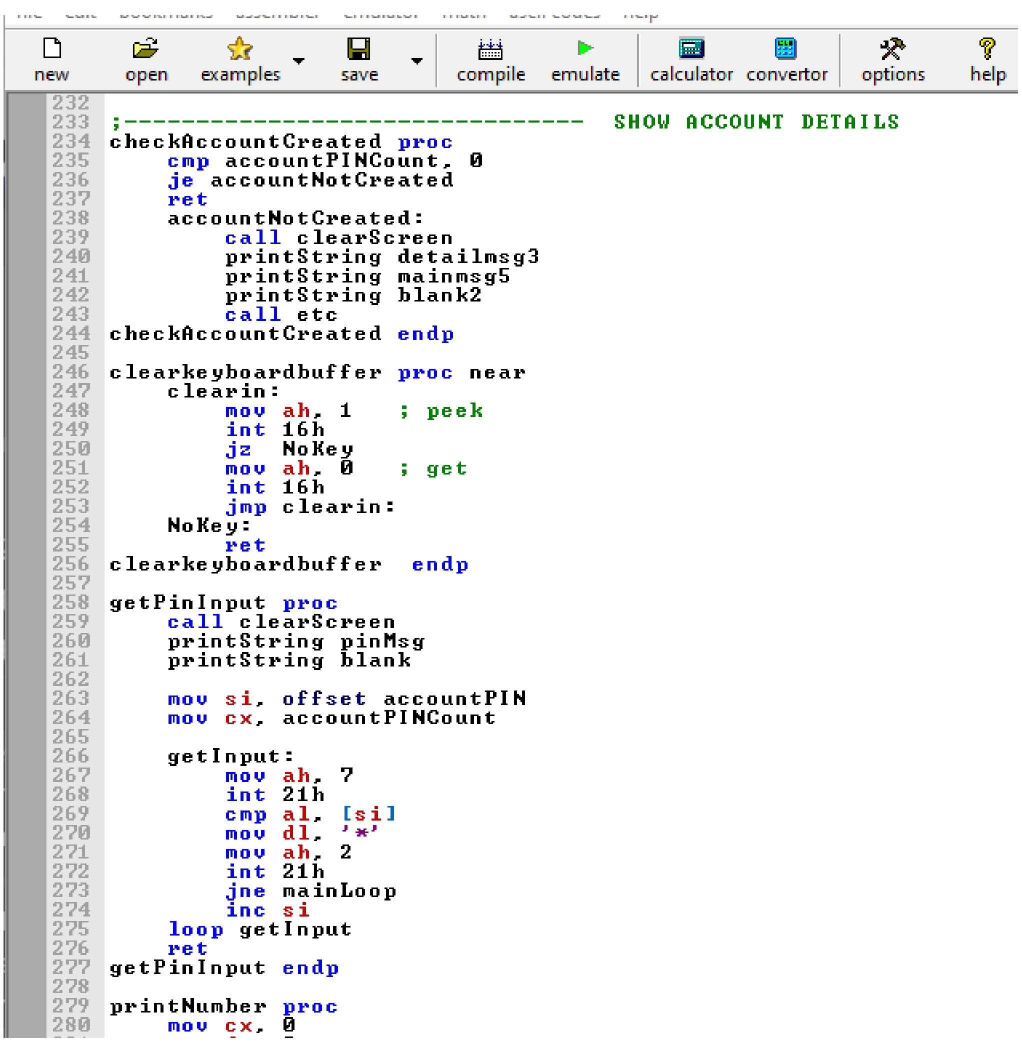
```
;    ------------------ENTRY POINT

Main proc
    mov ax, @data
    mov ds, ax

    call clearScreen

    mainLoop:
        call clearkeyboardbuffer
        call clearScreen
        call displayHeading
        printString blank2
        call displayinputMenu
        call clearkeyboardbuffer
        printString blank2
        call inputMenu

        cmp inputCode, '1'
        je create_account

        cmp inputCode, '2'
        je print_details

        cmp inputCode, '3'
        je withdraw

        cmp inputCode, '4'
        je deposit

        cmp inputCode, '0'
        je exit

        jmp mainLoop
    exit:
        printString blank2
        call displayBye
        printString blank2
        mov ah,4ch
        int 21h
    main endp
end main
```

Figure 2.3: Assembly Code for main Proce



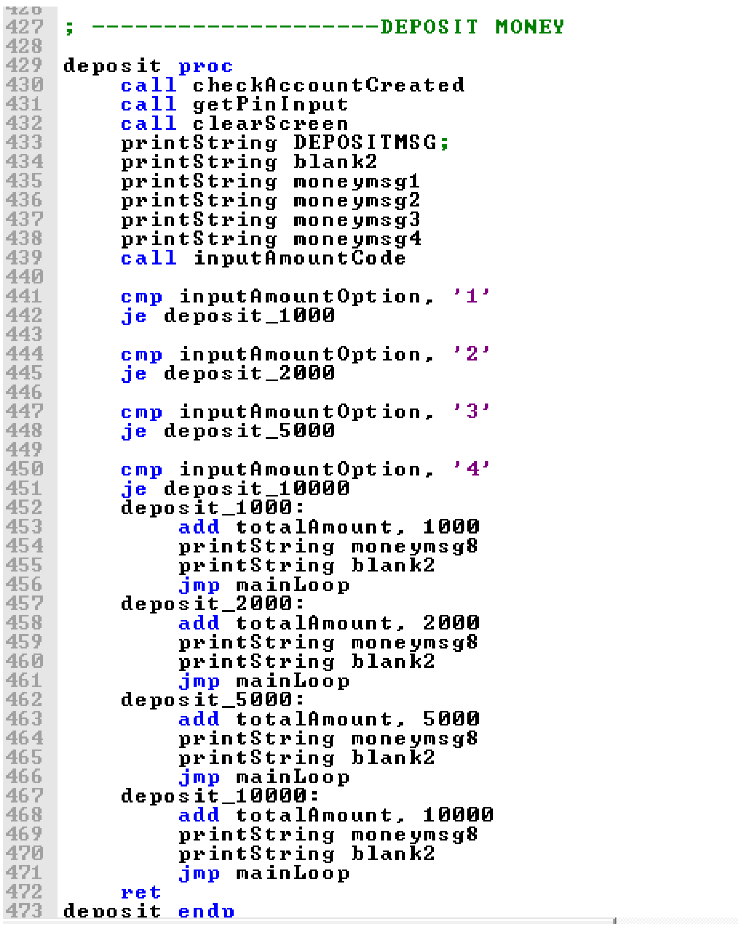
```
;--------------------  CREATE NEW ACCOUNT

macro account_name str
    mov si, offset str
    input:
        mov ah, 1
        int 21h
        cmp al, 13
        je create_pin
        mov [si], al
        inc si
        jmp input
    exitMac:
        ret
endm

macro account_pin str
    mov si, offset str
    input2:
        mov ah, 1
        int 21h
        cmp al, 13
        je create_phone
        inc accountPINcount
        mov [si], al
        inc si
        jmp input2
    exitMac2:
        ret
endm

macro account_phone str
    mov si, offset str
    input3:
        mov ah, 1
        int 21h
        cmp al, 13
        je create_city
        mov [si], al
        inc si
        jmp input3
    exitMac3:
        ret
endm

macro account_city str
    mov si, offset str
    input4:
        mov ah, 1
```

Figure 2.4: Assembly Code for Create account



```
;--------------------------------   SHOW ACCOUNT DETAILS
checkAccountCreated proc
    cmp accountPINCount, 0
    je accountNotCreated
    ret
    accountNotCreated:
        call clearScreen
        printString detailmsg3
        printString mainmsg5
        printString blank2
        call etc
checkAccountCreated endp

clearkeyboardbuffer proc near
    clearin:
        mov ah, 1    ; peek
        int 16h
        jz  NoKey
        mov ah, 0    ; get
        int 16h
        jmp clearin:
    NoKey:
        ret
clearkeyboardbuffer  endp

getPinInput proc
    call clearScreen
    printString pinMsg
    printString blank

    mov si, offset accountPIN
    mov cx, accountPINCount

    getInput:
        mov ah, 7
        int 21h
        cmp al, [si]
        mov dl, '*'
        mov ah, 2
        int 21h
        jne mainLoop
        inc si
    loop getInput
    ret
getPinInput endp

printNumber proc
    mov cx, 0
```

Figure 2.5: Assembly Code for show account Details



```
; --------------------DEPOSIT MONEY

deposit proc
    call checkAccountCreated
    call getPinInput
    call clearScreen
    printString DEPOSITMSG;
    printString blank2
    printString moneymsg1
    printString moneymsg2
    printString moneymsg3
    printString moneymsg4
    call inputAmountCode

    cmp inputAmountOption, '1'
    je deposit_1000

    cmp inputAmountOption, '2'
    je deposit_2000

    cmp inputAmountOption, '3'
    je deposit_5000

    cmp inputAmountOption, '4'
    je deposit_10000
    deposit_1000:
        add totalAmount, 1000
        printString moneymsg8
        printString blank2
        jmp mainLoop
    deposit_2000:
        add totalAmount, 2000
        printString moneymsg8
        printString blank2
        jmp mainLoop
    deposit_5000:
        add totalAmount, 5000
        printString moneymsg8
        printString blank2
        jmp mainLoop
    deposit_10000:
        add totalAmount, 10000
        printString moneymsg8
        printString blank2
        jmp mainLoop
    ret
deposit endp
```

Figure 2.6: Assembly Code for Deposit moneys

Chapter 3

Performance Evaluation

3.1 Simulation Environment/ Simulation Procedure

This section outlines the environment and setup required to implement and simulate the banking system project.

Hardware Requirements: A standard PC with a minimum of 4GB RAM, Intel/AMD processor, and sufficient storage to run emulator software.

Software Requirements:

- Emulator 8086 for coding and testing the assembly language program.
- An assembler and debugger integrated into the emulator for code execution and troubleshooting.

3.2 Results Analysis/Testing

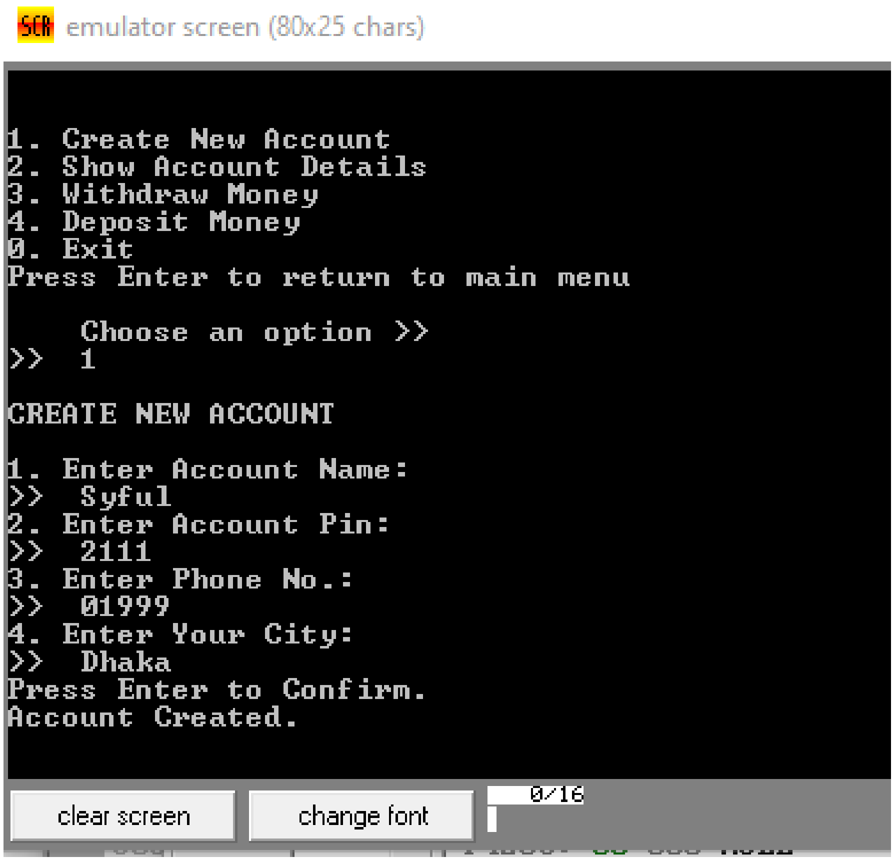
Here is the results obtained from running the project. Each functionality was tested individually and validated using multiple test cases.

3.2.1 Create Account

The program successfully captured user details, saved them, and displayed a success message.

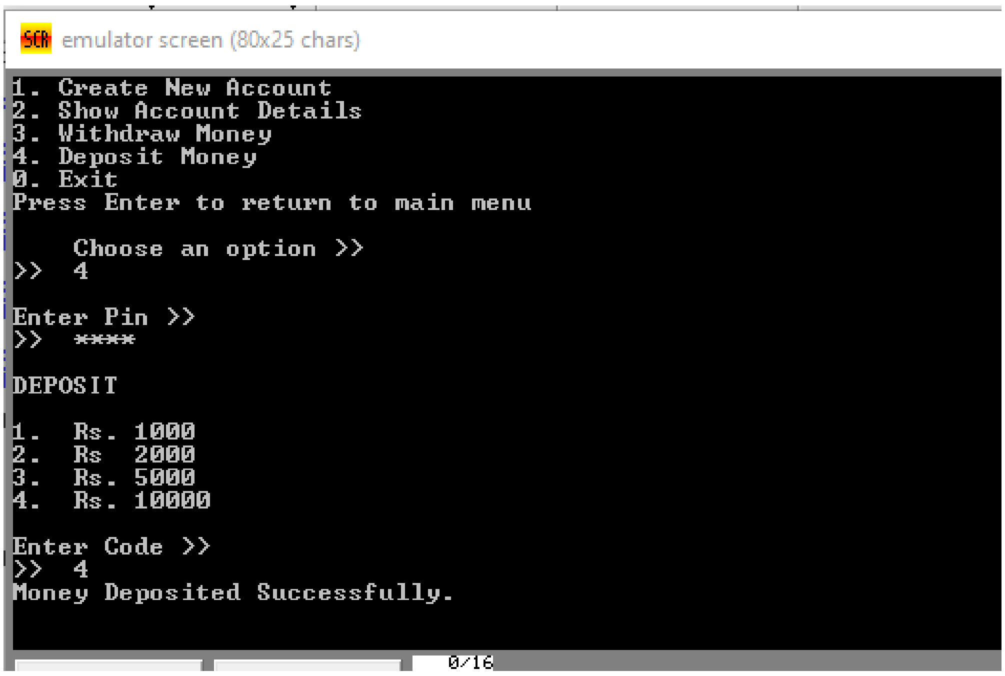
3.2.2 Deposit money

Deposits were added to the account balance without errors.



```
1. Create New Account
2. Show Account Details
3. Withdraw Money
4. Deposit Money
0. Exit
Press Enter to return to main menu

    Choose an option >>
>>  1

CREATE NEW ACCOUNT

1. Enter Account Name:
>>  Syful
2. Enter Account Pin:
>>  2111
3. Enter Phone No.:
>>  01999
4. Enter Your City:
>>  Dhaka
Press Enter to Confirm.
Account Created.
```

Figure 3.1: Successfully Created an account

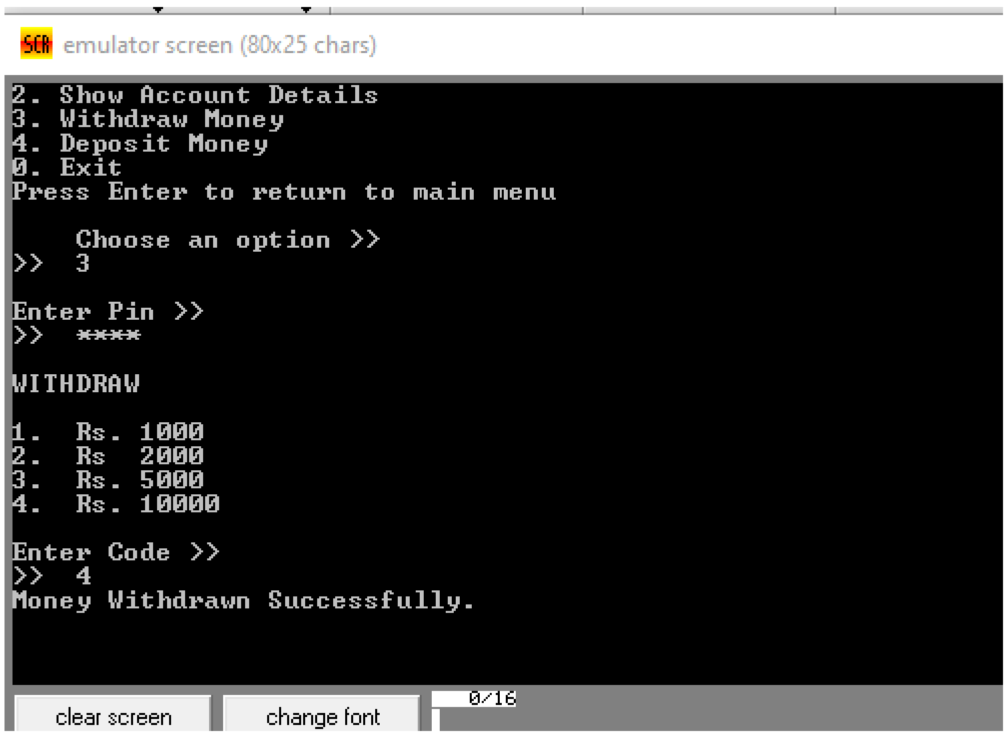


```
1. Create New Account
2. Show Account Details
3. Withdraw Money
4. Deposit Money
0. Exit
Press Enter to return to main menu

    Choose an option >>
>>  4

Enter Pin >>
>>  ****

DEPOSIT

1.  Rs. 1000
2.  Rs  2000
3.  Rs. 5000
4.  Rs. 10000

Enter Code >>
>>  4
Money Deposited Successfully.
```

Figure 3.2: Money deposited successfully

3.2.3 Withdraw Money

The withdrawal process worked accurately. If account balance is less then entered amount then will show an error messages for insufficient balance.



```
2. Show Account Details
3. Withdraw Money
4. Deposit Money
0. Exit
Press Enter to return to main menu

    Choose an option >>
>>  3

Enter Pin >>
>>  ****

WITHDRAW

1.  Rs. 1000
2.  Rs  2000
3.  Rs. 5000
4.  Rs. 10000

Enter Code >>
>>  4
Money Withdrawn Successfully.
```

Figure 3.3: Money withdraw Successfully

3.3 Results Overall Discussion

The results of the project were achieved through systematic testing of each functionality in the Emulator 8086 environment. The program successfully executed all functionalities like account creation, balance updates, and transaction processing. However, challenges such as limited memory and debugging complexities were identified. Overall, the project demonstrated reliable performance and fulfilled the primary objectives of a basic banking system.

Chapter 4

Conclusion

4.1 Discussion

This project successfully developed a banking management system using assembly language on the Emulator 8086, achieving key functionalities like account creation, balance inquiries, deposits, and withdrawals. The results validated the system's efficiency in addressing fundamental banking tasks and confirmed the project's ability to meet its objectives while addressing core banking requirements within the limitations of the assembly language environment.

4.2 Limitations

The project faced several limitations such as,

- **Limited Memory and Processing Power:** The use of assembly language imposes strict memory constraints, limiting the program's complexity and functionality.

- **No Multitasking Support:** The system does not support simultaneous multi-user interactions, making it unsuitable for real-world banking scenarios.

- **Manual Input Dependency:** All inputs are manual, which increases the likelihood of human errors in transactions.

- **Not Scalable:** The program is designed for small-scale operations and cannot handle a large number of accounts or complex banking operations.

- **No Data Persistence:** The system does not store data permanently. All data is lost when the program exits.

4.3 Scope of Future Work

The project has significant potential for future enhancements and expansions. One major area of migrating the system to a higher-level programming language could improve scalability and functionality. Introducing a graphical user interface (GUI) would also enhance usability and make the system more user-friendly for non-technical users.And following others features can be included in future,

- **Data Persistence:** Implement a mechanism for saving account data permanently, such as integrating file systems or databases.
- **Multitasking Capability:** Develop the system to handle multiple users and concurrent transactions.
- **Portability:** Adapt the system to run on modern platforms and architectures beyond Emulator 8086.
- **Support for Advanced Banking Operations:** Expand functionality to include loans, interest calculations, and account type management.

These advancements would make the system more robust, secure, and aligned with modern banking requirements.

References

1. https://www.quora.com/What-are-some-cool-assembly-language-projects-I-can-do
2. https://mlgansari.wordpress.com/wp-content/uploads/2020/04/5-procedure-and-macro-in-assembly-language-program.pdf
3. https://stackoverflow.com/questions/34375300/how-to-pass-command-line-parameters-to-emu8086
4. https://www.philadelphia.edu.jo/academics/qhamarsheh/uploads/emu8086.pdf
5. https://www.projectmanager.com/blog/sample-project-management-flow-chart
6. https://www.youtube.com/watch?v=Ucush4KDSK0