



Green University of Bangladesh

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

ATM System

Course Title: Microprocessors and Microcontrollers Lab

Course Code: CSE 304

Section: 222_D9

Students Details

Name	ID
Md. Sabbir Hossain	221902126
Md. Israil Fakir	221902125

Submission Date: 12 Dec 2024

Course Teacher's Name: Sagufta Sabah Nakshi

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

<u>Project Proposal Status</u>	
Marks:	Signature:
Comments:	Date:

Contents

1	Introduction	3
1.1	Overview	3
1.2	Features	3
1.3	Motivation	3
1.4	Problem Definition	4
1.4.1	Problem Statement	4
1.4.2	Complex Engineering Problem	4
1.5	Design Goals	5
1.6	Objectives	5
1.7	Summary	5
2	Implementation	6
2.1	Procedure	6
2.1.1	Data Section	6
2.1.2	Code Section	7
3	Performance Evaluation	9
3.1	Code Structure	9
3.2	Snippets Of the Project	10
3.2.1	Login Screen	10
3.2.2	Balance Check	11
3.2.3	Withdraw Money	12
3.2.4	Deposit Money	13
3.2.5	Exit Process	14
3.3	Error Handling	15
3.3.1	Invalid Account Notification	15
3.3.2	Invalid Password	15

3.3.3	Invalid Amount	16
4	Conclusion	17
4.1	Conclusion	17
4.2	Limitations	17
4.3	Scope Of Future Work	17
4.3.1	Graphical User Interface (GUI)	17
4.3.2	Enhanced Security Features	18
4.3.3	Dynamic Data Storage	18
4.3.4	Transaction History	18
4.3.5	Multi-Account Support	18
4.3.6	Mobile Application Integration	18

Chapter 1

Introduction

1.1 Overview

This assembly language project aims to create a simplified ATM (Automated Teller Machine) that allows users to perform basic banking operations. The project focuses on user authentication, balance inquiry, cash withdrawal, and a new feature called "Transfer Cash." [1]

1.2 Features

- **User Authentication:**
 - Users must enter a Personal Identification Number (PIN) to access their account.
 - Implement secure PIN input and validation to protect user accounts.
- **Balance Inquiry:**
 - Displays the most recent account balance retrieved from the database or internal storage.
 - Provides clear and concise information on the screen for easy understanding.
- **Cash Withdrawal**
 - Users can select a predefined amount or enter a custom amount for withdrawal.
 - Automatically updates the user's account balance after a successful withdrawal.

1.3 Motivation

The motivation behind developing this assembly language project is to simulate the functionality of an Automated Teller Machine (ATM) at a low level, gaining insights

into the intricate details of hardware control and system interactions. ATMs are integral to modern banking, and understanding their operations at the assembly level is crucial for students and enthusiasts interested in computer architecture and embedded systems.

1.4 Problem Definition

The problem is to create a functional ATM system in assembly language that makes basic banking operations. The challenge involves developing a text-based user interface, implementing secure user authentication, managing account balances, and introducing a new feature called "Transact Cash." The assembly code needs to be organized, modular, and efficient, ensuring a seamless user experience and adherence to security standards.

1.4.1 Problem Statement

- **User Authentication** The project involves designing a secure user authentication mechanism. The assembly code must verify a user's entered Personal Identification Number (PIN) and grant access only upon successful authentication. Security concerns, such as limiting incorrect PIN attempts, need to be addressed.
- **Basic Banking Operations** The assembly program needs to facilitate fundamental banking operations like balance inquiry and cash withdrawal. These tasks involve data retrieval, arithmetic operations, and updating memory locations.
- **Transact Cash** The introduction of the "Transact Cash" feature poses a challenge in implementing a versatile cash transaction system. The assembly code must handle various transaction types, ensuring accurate updates to account balances.

1.4.2 Complex Engineering Problem

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

Name of the P Attributes	Explain how to address
P1: Depth of knowledge required	Need to deep knowledge about Assembly language
P2: Range of conflicting requirements	Can make different way this project
P3: Depth of analysis required	Continuous update improvement with requirement loops and deep analysis.
P4: Familiarity of issues	–
P5: Extent of applicable codes	We have to know extreme level code.
P6: Extent of stakeholder involvement and conflicting requirements	–
P7: Interdependence	Integration of conversational ATM systems.

Since, the attributes number P1, P2, P3, P6, and P7 can be touched by my project, that's why my project can be addressed as a complex engineering Problem.

1.5 Design Goals

The project is driven by the following overarching goals:

- **Personalization:** Create a platform that adapts to each user's preferences and interactions.
- **Security and User Authentication:** Implement secure PIN input and validation to protect user accounts.
- **Real-Time Performance:** Minimize response time for user interactions, ensuring smooth and real-time performance.
- **User-Friendly Interface:** Display simple and clear instructions on the screen for user guidance.
- **Transaction Integrity:** Ensure accurate calculations for withdrawals, deposits, and account balances.

1.6 Objectives

The primary objectives of the assembly language project are:

- User Authentication.
- Basic Banking Operations.
- Transact Cash Feature.
- Efficient Memory Management.
- User-Friendly Interface.
- Security Measures.

1.7 Summary

The project focuses on implementing essential banking operations with user-friendly features. It includes secure user authentication to safeguard accounts, balance inquiry for real-time updates, and efficient cash withdrawal with denomination handling. A new "Transfer Cash" feature enables account-to-account transfers with recipient validation and secure processing. The system emphasizes security, reliability, and an intuitive interface to enhance user experience. These features ensure seamless, secure, and efficient banking operations for users.

Chapter 2

Implementation

This chapter outlines the implementation of a simple AMT system using assembly language on the emu8086 emulator. The project replicates core banking features such as account verification, balance checking, withdrawals, deposits, and secure logout.

Key functionalities include:

- **Secure Access:** Verification of account number and password with masked input for privacy.
- **Menu Navigation:** User-friendly options for transactions and account management.
- **Transaction Validation:** Limits on withdrawals and deposits to ensure realistic operations.
- **Dynamic Feedback:** Real-time error handling and success messages.
- **Efficient Design:** Use of loops, string handling, and arithmetic operations in assembly.

2.1 Procedure

2.1.1 Data Section

Account Property:

- `Acnt_msg`: Prompt to enter the account number.
- `Acnt_wrong`: Message for invalid account numbers.
- `Acc_pass`: Predefined account number ("1234").
- `Acc_pass_length`: Length of the account number.

Password Property:

- pass_msg: Prompt to enter the password.
- Pass_wrong: Message for invalid passwords.
- Pass: Actual password ("test").
- Pass_len: Length of the password.

Menu Options:

- Intro_msg: Welcome message.
- Balance_menu: Option to check balance.
- Withdraw_menu: Option to withdraw money.
- dep_menu: Option to deposit money.
- Exit_menu: Option to exit the program.

Transaction Messages:

- Success_msg: Message for successful transactions.
- Limit_amount: Error for exceeding transaction limits.
- Low_balance: Error for insufficient balance.

Balance Info:

- Current_balance: Predefined account balance (20,000).
- Current_balance_msg: Message displaying the current balance.

Withdrawal and Deposit Amounts:

- above1000: Range for amounts between 1,000 and 5,000.
- above100: Range for amounts between 100 and 999.

2.1.2 Code Section**Account Verification:**

- Prompts the user to enter their account number.
- Compares the input with the predefined account number.
- Displays an error message if the account number is incorrect.

Password Check:

- Prompts the user to enter their password.
- Verifies it against the predefined password.
- Masks the input for security and exits on failure.

Menu Navigation:

- Displays the main menu with options for balance inquiry, withdrawal, deposit, or exit.
- Waits for user input and directs the program accordingly.

Balance Inquiry:

- Displays the current balance stored in memory.

Money Withdrawal:

- Prompts the user to select a withdrawal range (1,000–5,000 or 100–999).
- Verifies that the entered amount does not exceed the balance or the defined limit.
- Updates and displays the balance after a successful transaction.

Money Deposit:

- Prompts the user to select a deposit range (1,000–5,000 or 100–999).
- Updates and displays the balance after a successful deposit.

Error Handling:

- Handles invalid inputs or transactions exceeding the predefined limits.
- Displays appropriate error messages and returns to the main menu.

Exit:

- Displays a thank you message and terminates the program.

Chapter 3

Performance Evaluation

3.1 Code Structure

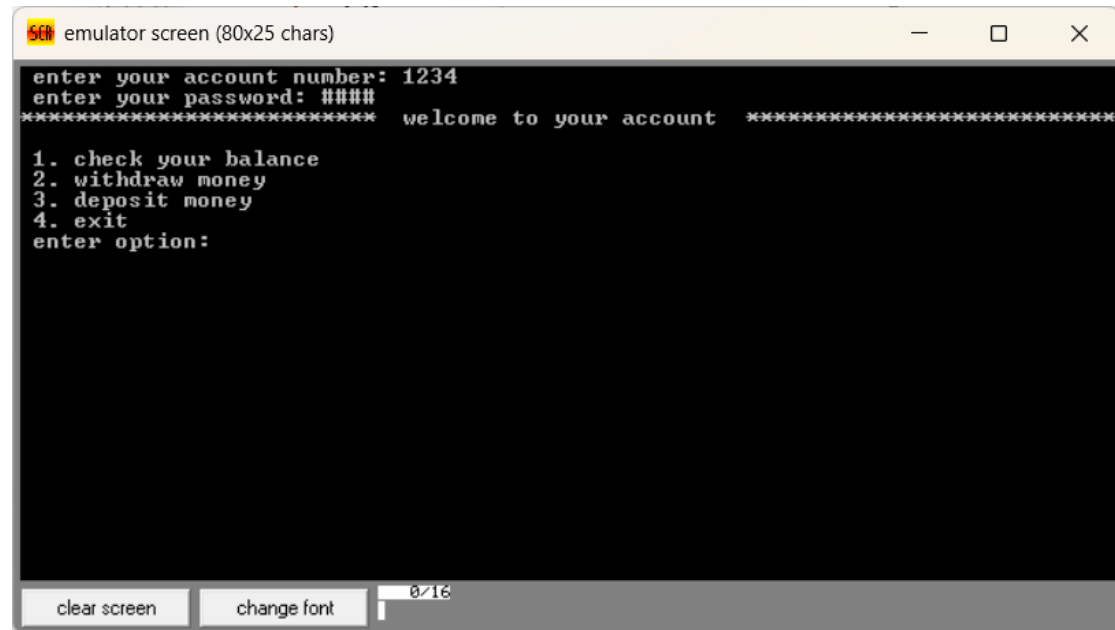
The assembly code is organized into modular functions, each responsible for specific tasks. Key functions include:

- **AuthenticateUser:** Validates the user's account number and password to grant access to the banking functionalities.
- **DisplayMenu:** Presents the user with a list of banking options (e.g., balance inquiry, withdrawal, deposit, exit).
- **CheckBalance:** Retrieves and displays the current balance from the user's account.
- **WithdrawCash:** Processes cash withdrawal requests, ensures sufficient funds, and updates the account balance accordingly.
- **DepositCash:** Handles cash deposit transactions, validates the deposit amount, and updates the account balance.
- **HandleInvalidInput:** Detects and manages invalid inputs by displaying appropriate error messages and redirecting users to the main menu.
- **TransactionSuccess:** Displays success messages for completed transactions and shows the updated account balance.
- **InputValidation:** Verifies the correctness of numeric inputs (e.g., withdrawal and deposit amounts) to ensure they fall within acceptable limits.
- **ErrorHandling:** Manages errors like insufficient balance, exceeding limits, or invalid operations and provides clear feedback to the user.
- **ExitApplication:** Displays a thank-you message and terminates the program gracefully.

3.2 Snippets Of the Project

The output demonstrates the interaction flow between the user and the ATM system. Each menu option is accompanied by relevant prompts and messages, guiding the user through the transaction process.

3.2.1 Login Screen



```
emulator screen (80x25 chars)
enter your account number: 1234
enter your password: ####
***** welcome to your account *****
1. check your balance
2. withdraw money
3. deposit money
4. exit
enter option:
```

The screenshot shows a terminal window titled "emulator screen (80x25 chars)". The text displayed is as follows: "enter your account number: 1234", "enter your password: ####", "***** welcome to your account *****", a menu with four options: "1. check your balance", "2. withdraw money", "3. deposit money", and "4. exit", and finally "enter option:". At the bottom of the window, there are two buttons labeled "clear screen" and "change font", and a small status bar showing "0/16".

Figure 3.1: Login page

- Prompts users to enter their account number and password.
- Password input is masked (####) for security.
- Displays a welcome message after successful login.

3.2.2 Balance Check

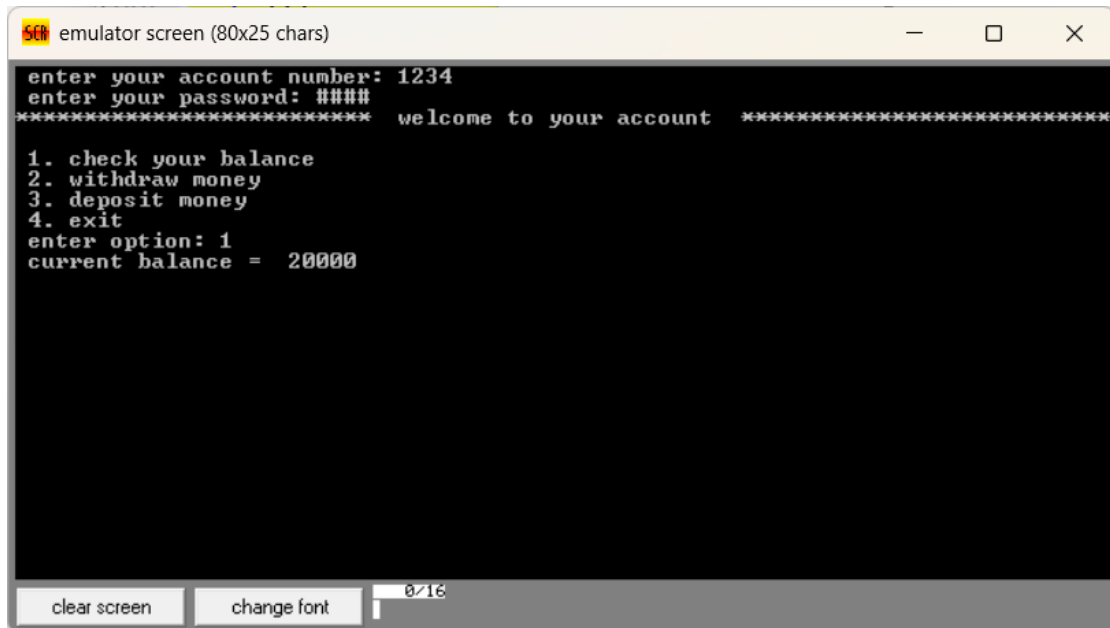
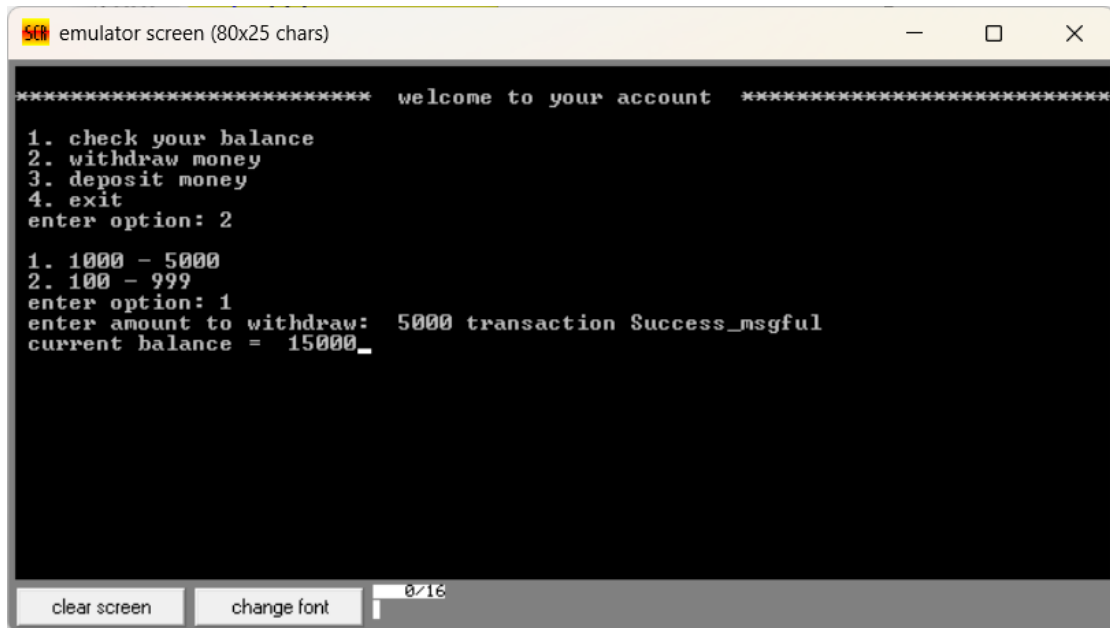


Figure 3.2: Checking Balance

- After selecting the "Check Balance" option from the main menu, the current account balance is displayed.
- The balance is shown in a simple format for easy readability (current balance = 20000).

3.2.3 Withdraw Money



```
emulator screen (80x25 chars)
***** welcome to your account *****
1. check your balance
2. withdraw money
3. deposit money
4. exit
enter option: 2
1. 1000 - 5000
2. 100 - 999
enter option: 1
enter amount to withdraw: 5000 transaction Success_msgful
current balance = 15000_
clear screen change font 0/16
```

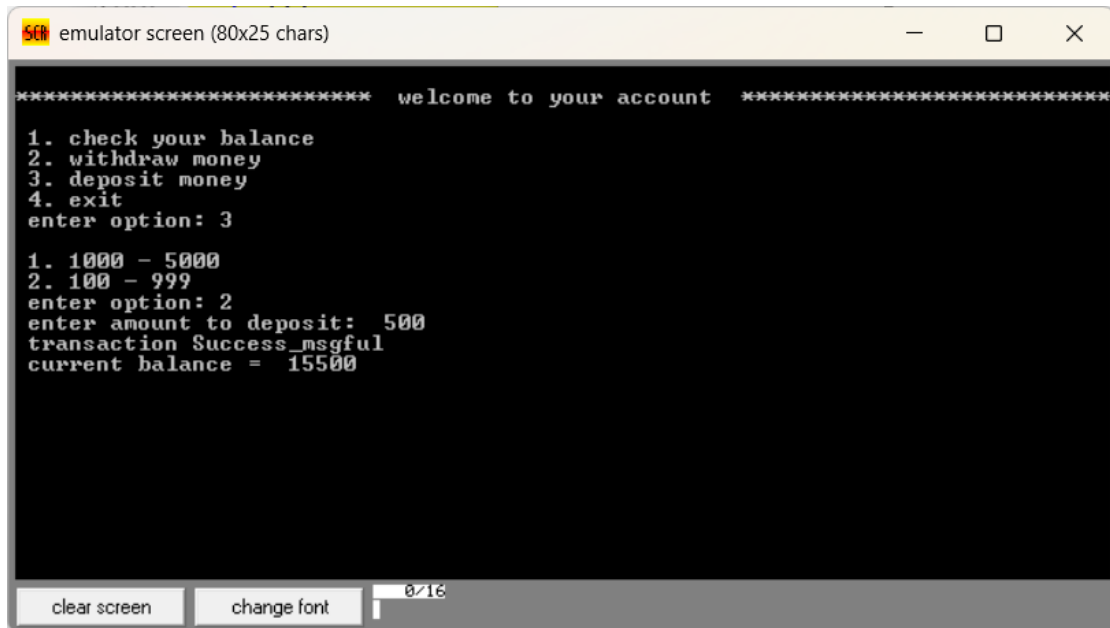
Figure 3.3: Withdrawing

- After selecting the "Withdraw Money" option from the main menu, the user is prompted to choose a withdrawal range (e.g., 1. 1000 - 5000 or 2. 100 - 999).
- The user inputs the amount they wish to withdraw, followed by a confirmation message indicating whether the transaction was successful.

Error Handling

- Ensures that the withdrawal is within the defined range.
- Invalid inputs or amounts exceeding the available balance can be handled by adding error messages in future iterations.

3.2.4 Deposit Money



```
emulator screen (80x25 chars)
***** welcome to your account *****
1. check your balance
2. withdraw money
3. deposit money
4. exit
enter option: 3
1. 1000 - 5000
2. 100 - 999
enter option: 2
enter amount to deposit: 500
transaction Success_msgful
current balance = 15500
clear screen change font 0/16
```

Figure 3.4: Deposite

- After selecting the "Deposit Money" option (Option 3) from the main menu, the user is prompted to enter an amount.
- A success message (transaction Success_msgful) confirms the deposit.
- The updated account balance (current balance = 15500) is displayed to provide immediate feedback to the user.

3.2.5 Exit Process

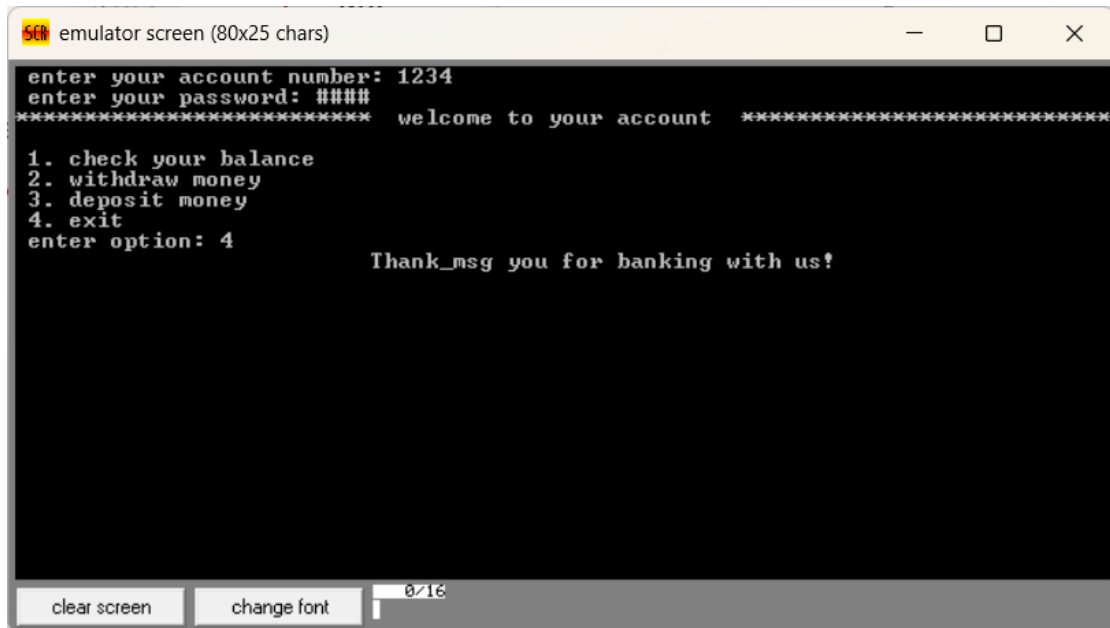


Figure 3.5: Exit Process

- If the user selects Option 4 (Exit), a friendly message is displayed: "Thank_msg you for banking with us!"

3.3 Error Handling

The program incorporates basic error handling to address potential issues with user input. Clear error messages aid users in understanding and resolving input-related issues.

3.3.1 Invalid Account Notification

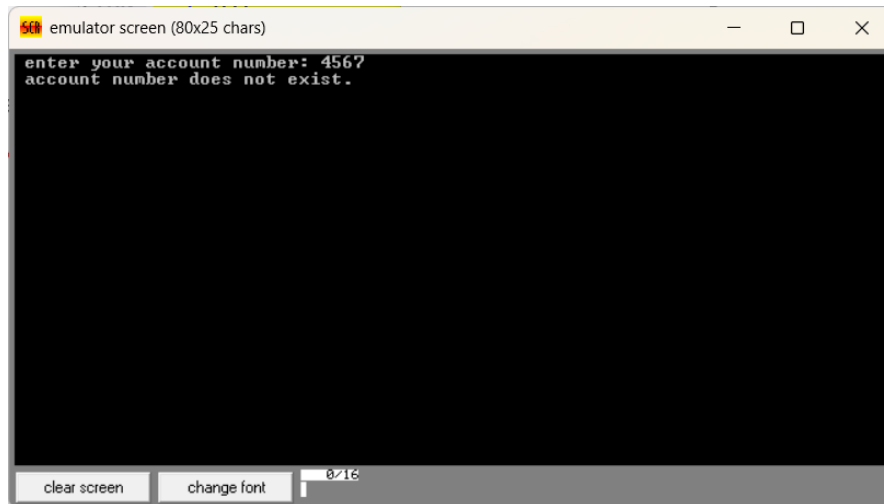


Figure 3.6: Invalid Account Number

- Input Prompt: The user is asked to enter their account number (e.g., 4567).
- Error Message: If the entered account number does not exist, the system displays the message: "**Account number does not exist.**"

3.3.2 Invalid Password

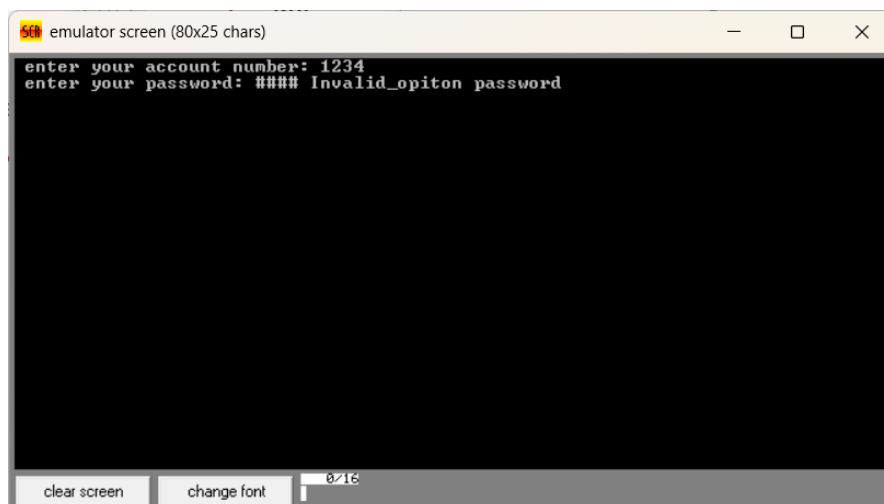
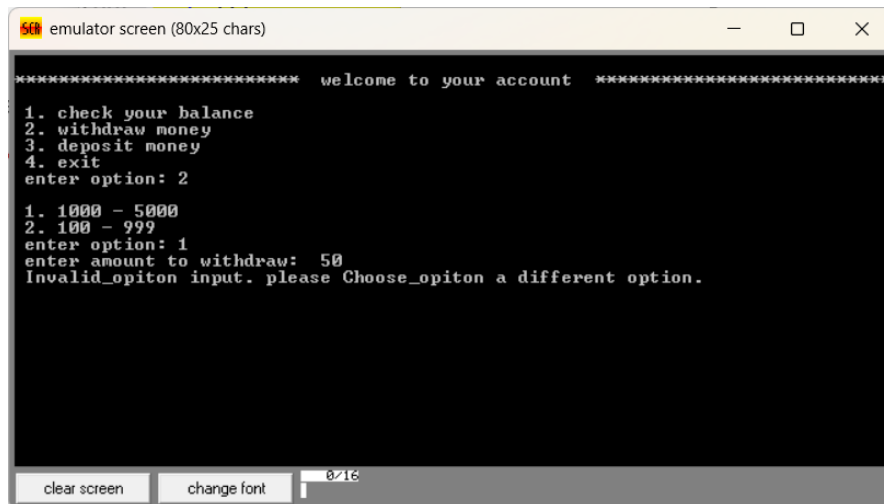


Figure 3.7: Invalid Password

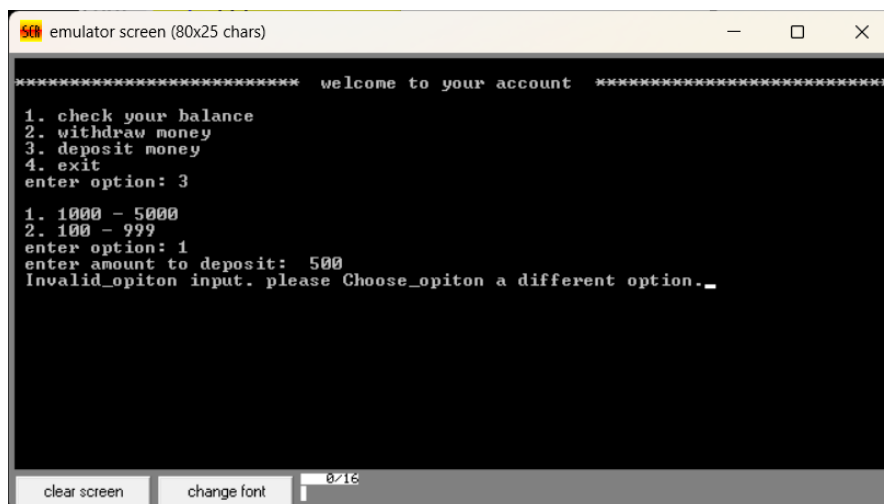
- Error Message: If the entered account number matching but the password does not exist, the system displays the message: "**Invalid Password.**"

3.3.3 Invalid Amount



```
emulator screen (80x25 chars)
***** welcome to your account *****
1. check your balance
2. withdraw money
3. deposit money
4. exit
enter option: 2
1. 1000 - 5000
2. 100 - 999
enter option: 1
enter amount to withdraw: 50
Invalid_opiton input. please Choose_opiton a different option.
```

Figure 3.8: Invalid Amount in withdraw



```
emulator screen (80x25 chars)
***** welcome to your account *****
1. check your balance
2. withdraw money
3. deposit money
4. exit
enter option: 3
1. 1000 - 5000
2. 100 - 999
enter option: 1
enter amount to deposit: 500
Invalid_opiton input. please Choose_opiton a different option._
```

Figure 3.9: Invalid Amount in Deposite

Invalid_option input. please Choose_option a different option.

This error message is triggered when a user enters an option that is not valid in the main menu or other choice-based selections.

Chapter 4

Conclusion

4.1 Conclusion

The ATM transaction simulator successfully implements essential banking operations like account verification, balance checking, withdrawal, and deposit using assembly language. It demonstrates efficient use of low-level programming, ensuring secure transactions, proper error handling, and a structured modular design. This project highlights the feasibility of creating lightweight and functional systems for resource-constrained environments.

4.2 Limitations

- Platform-dependent, relying on DOS interrupts.
- Hard-coded credentials limit scalability.
- Lacks modern user interfaces and encryption for security.
- Limited error handling and no support for dynamic multi-user operations.

4.3 Scope Of Future Work

The current ATM machine project, implemented in Assembly language, lays the foundation for further enhancements and improvements. Here are potential areas for future work and expansion:

4.3.1 Graphical User Interface (GUI)

- Develop a graphical interface to replace the text-based interface, providing a more user-friendly and visually appealing experience.
- Implement interactive buttons, menus, and graphics to enhance the overall design.

4.3.2 Enhanced Security Features

- Integrate advanced security measures, such as encryption algorithms for PINs and transaction data, to ensure secure transactions.
- Implement biometric authentication methods (fingerprint, facial recognition) for added security.

4.3.3 Dynamic Data Storage

- Replace hardcoded values with dynamic data storage using databases or file systems to handle a larger number of users and account details.
- Explore database management systems for better data organization and retrieval.

4.3.4 Transaction History

- Incorporate a transaction history feature that allows users to view their recent transactions.
- Implement mechanisms to store and retrieve transaction logs securely.

4.3.5 Multi-Account Support

- Extend the system to support multiple user accounts with unique credentials.
- Implement functionality for users to create, update, and delete accounts.

4.3.6 Mobile Application Integration

- Develop a mobile application version of the ATM system to increase accessibility for users on smartphones.
- Ensure compatibility with different mobile platforms (iOS, Android).

References

[1] Peter Knaggs and Stephen Welsh. *ARM: Assembly Language Programming*. Bournemouth University, School of Design, Engineering, and Computing, 2004.

- **Stack Overflow**
- **Youtube**
- **ChatGTP**
- **Google Gemini**