**Mini Bank ATM Simulator**



**Submitted by:**

1. **Habib-ur-Rehman (NUM-BSCS-2023-12)**
2. **Rayan Badar (NUM-BSCS-2023-18)**

**CSC-241**

**Computer Organization and Assembly Language**

**Fall - 2024**

**Department of Computer Science**

**Namal University Mianwali**

**1. Introduction**

The **Bank ATM Simulation** project, developed as part of the **Computer Organization and Assembly Language** course, highlights the practical application of low-level programming in solving real-world problems. The purpose of this project is to implement a simple yet efficient banking system using assembly language, demonstrating how fundamental computer architecture concepts can be applied to create functional and interactive software systems.

Assembly language is a cornerstone of computer science, offering direct control over hardware and a deeper understanding of how software interacts with a computer’s internal mechanisms. This project emphasizes its relevance by showcasing the intricate operations involved in managing banking functionalities, such as maintaining customer account records, processing deposits and withdrawals, and generating account statements.

By simulating a banking system, this project bridges theoretical learning and practical application, reinforcing concepts such as memory management, file handling, and modular programming. The scope of the project extends beyond its academic objectives, providing a foundation for understanding low-level programming in critical systems, such as financial software, embedded systems, and operating systems.

This report outlines the design, functionality, and implementation of the Bank Management System, demonstrating the significance of assembly language in the field of computer science.

**2. Problem Statement**

In today’s rapidly advancing digital world, efficient and reliable ATM systems are essential for managing financial transactions and records. While modern high-level programming languages are often used to build such systems, it is crucial to understand the underlying low-level operations that enable their functionality. This project aims to address the following problem:

How can we design and implement a basic banking system using assembly language that provides essential functionalities such as account creation, balance inquiry, deposits, withdrawals, and transaction history management …?

The challenges in developing this project include:

1. **Limited Programming Abstractions:** Unlike high-level languages, assembly language provides minimal abstraction from the hardware, requiring precise and detailed instructions for each operation.
2. **User Interaction:** Designing a simple yet functional interface that allows users to interact with the system, ensuring usability without compromising the complexity of assembly programming.
3. **Error Handling:** Implementing robust mechanisms to handle errors, such as invalid inputs while maintaining program stability.
4. **Performance Optimization:** Managing resources like memory and CPU efficiently, which is critical in assembly language programming.

This project seeks to overcome these challenges by demonstrating how core banking operations can be implemented using assembly language. By doing so, it highlights the importance of understanding low-level programming and its role in building reliable, efficient, and secure systems.

**3. Proposed Solution**

To address the problem statement, the **Bank ATM Simulation** project is designed to simulate essential banking operations using assembly language. The solution focuses on implementing core functionalities, including account creation, balance inquiries, deposits, withdrawals, and transaction management. The program is modular, with each feature encapsulated in a dedicated procedure for clarity and reusability. The design leverages assembly language's ability to directly control hardware resources, ensuring efficient execution and resource management.

**Solution Overview**

The solution is structured into several key components:

1. **Login System:**

The system supports both administrator and account holder logins.

Credentials such as usernames and passwords are verified against pre-stored values to ensure secure access.

1. **Account Management:**

Users can view account details, including account number, name, balance, contact information, and more.

Administrators can search for user accounts and update specific information.

1. **Transaction Operations:**

Fast cash withdrawals are supported for predefined amounts.

Users can deposit money into their accounts or withdraw custom amounts within their balance limits.

The system supports transferring money between accounts, validating sufficient balances before processing.

1. **Error Handling and Validation:**

Robust error-checking mechanisms are implemented to validate user inputs, handle invalid operations, and ensure program stability.

**Key Assembly Language Concepts**

1. **Memory Management:**

The system uses fixed-size buffers for user input and ensuring controlled memory allocation.

1. **Conditional Branching and Loops:**

Decision-making structures and iterative loops are extensively used to process user input, navigate menus, and handle transactions.

1. **Procedural Design:**

The program follows a modular design with separate procedures for login, account operations, and transactions, promoting code reusability and readability.

1. **Text and Color Formatting:**

Functions are used to format text and apply colors, enhancing the user experience in the command-line interface.

**Algorithms and Frameworks**

* **Validation Algorithm:**

Ensures valid inputs by comparing user-provided data against stored values, using string comparison and conditional logic.

* **Arithmetic Operations:**

Handles calculations for deposits, withdrawals, and balance updates.

* **Data Parsing:**

Processes strings and buffers to read and write structured data.

This proposed solution demonstrates how assembly language, despite its complexity, can be used to build a functional and efficient banking system. It highlights the importance of a solid understanding of computer organization concepts in designing low-level software solutions.

### **3.1 Flow Diagram**

The flow diagram for the **Bank ATM Simulation** represents the major steps, decision points, and outputs of the program. Here’s the textual description of the flow, which you can convert into a diagram using tools like PowerPoint or any flowchart tool:

1. **Start Program**

Display the welcome screen.

Prompt the user to insert a card and press any key.

1. **Login Menu**

Display the login options:

* + - Administrator Login.
    - Account Holder Login.
    - Exit.

1. **Administrator Login**

Prompt the user for admin credentials (username and password).

Validate credentials:

* + - If correct, proceed to the admin menu.
    - If incorrect, prompt the user to retry or exit.

1. **Account Holder Login**

Prompt the user for account credentials (account number and PIN).

Validate credentials:

* + - If correct, proceed to the account holder menu.
    - If incorrect, prompt the user to retry or exit.

1. **Admin Menu**

Options:

* + - Search User Details.
    - ATM Info.
    - Start ATM System.
    - Turn Off System.

Branch based on the user's choice.

1. **Account Holder Menu**

Options:

* + - Fast Cash.
    - Cash Withdrawal.
    - Deposit.
    - Transfer Money.
    - View Account Details.
    - Change PIN.
    - Update Account Info.
    - Log Out.

Branch based on the user's choice.

1. **Perform Transaction**

Validate inputs (e.g., withdrawal amount <= balance).

Update balances accordingly.

Display success message and remaining balance.

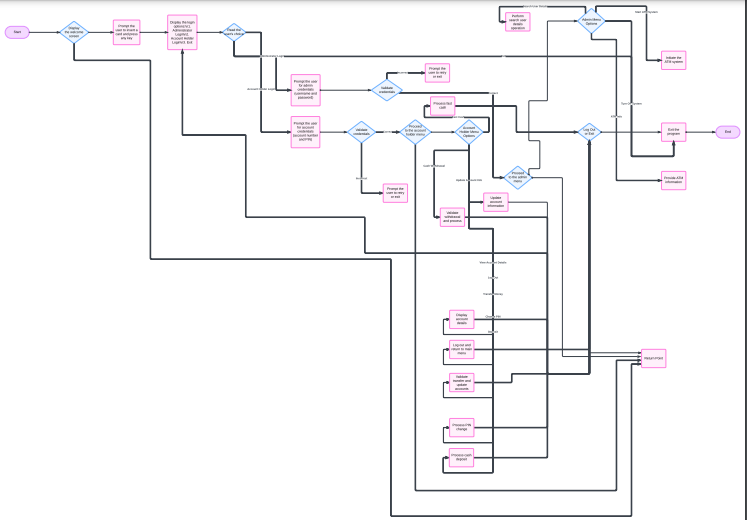
1. **Log Out or Exit**

Log out the current user and return to the main menu.

Exit the program if chosen.

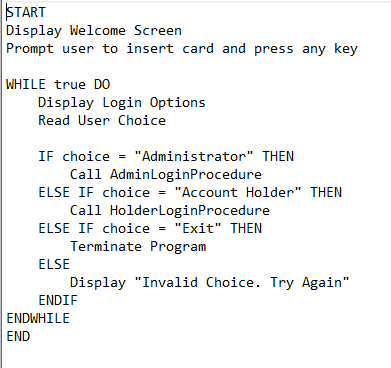
**Flow Diagram**

[..\OneDrive\Desktop\COAL Project\Blank diagram (1).pdf](file:///C:\Users\rayan\OneDrive\Desktop\COAL%20Project\Blank%20diagram%20(1).pdf)



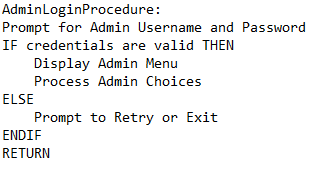
### **3.2 Pseudo Codes**

**Main Program**



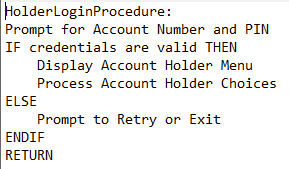
[..\OneDrive\Desktop\COAL Report Codes\Main Program.txt](C:\\Users\\rayan\\OneDrive\\Desktop\\COAL Project\\Main Program.txt)

**Administrator Login Procedure**

****

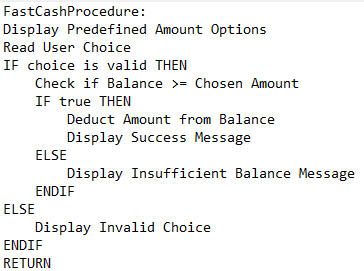
[..\OneDrive\Desktop\COAL Report Codes\Administrator Login Procedure.txt](C:\\Users\\rayan\\OneDrive\\Desktop\\COAL Project\\Administrator Login Procedure.txt)

**Account Holder Login Procedure**

****

[..\OneDrive\Desktop\COAL Report Codes\Account Holder Login Procedure.txt](C:\\Users\\rayan\\OneDrive\\Desktop\\COAL Project\\Account Holder Login Procedure.txt)

**Cash Transaction Example**

****

[..\OneDrive\Desktop\COAL Report Codes\Transaction Example.txt](C:\\Users\\rayan\\OneDrive\\Desktop\\COAL Project\\Transaction Example.txt)

### **3.3 Assembly Code**

**Complete Code**

[**C:\Users\rayan\OneDrive\Desktop\Complete code.txt**](file:///C:\Users\rayan\OneDrive\Desktop\Complete%20code.txt)

INCLUDE \irvine\Irvine32.inc

INCLUDELIB \irvine\Irvine32.lib

INCLUDE \irvine\macros.inc

BUFFER\_SIZE = 5000

;functions hain

;1.Quick cash

;2.withdrawal cash

;3.deposit cash

;4.transfer amount

.DATA

localTime SYSTEMTIME <>

WhiteTextOnBlue = black + (yellow \* 16)

choice DD ?

admin\_choice DWORD ?

UserOption DWORD ?

buffer BYTE BUFFER\_SIZE DUP(?)

filename BYTE "Designing.txt", 0

fileHandle HANDLE ?

password BYTE 20 DUP(?)

username BYTE 20 DUP(?)

AdminUser BYTE "admin", 0

AdminPassword BYTE "1234", 0

AdminLogChoice DD ?

HolderPassword BYTE 20 DUP(?)

HolderUsername BYTE 20 DUP(?)

StoredAccountID BYTE "23021519000", 0

StoredPIN BYTE "1234", 0

StoredHolderName BYTE "User", 0

StoredCNIC BYTE "35101-0000000-1", 0

StoredNO BYTE "0300-1122334", 0

StoredCity BYTE "Namal University MIANWALI"

StoredMail BYTE "cs@gmail.com", 0

gender BYTE "M", 0

StoredAmount DWORD 2000,

HolderLogChoice DD ?

fcash DWORD ?

fcashChoice DWORD ?

wdcash DWORD ?

wdcashChoice DWORD ?

depositAmount DWORD ?

TransferMoney DWORD ?

GenerateBalance DWORD 0

.CODE

AddColor PROC

MOV EAX, WhiteTextOnBlue

CALL SetTextColor

CALL CLRSCR

RET

AddColor ENDP

Time PROC

INVOKE GetLocalTime, ADDR localTime

movzx EAX, localTime.wHour

CALL WriteDec

mWrite ":"

movzx EAX, localTime.wMinute

CALL WriteDec

mWrite ":"

movzx EAX, localTime.wSecond

CALL WriteDec

mWrite " "

movzx EAX, localTime.wDay

CALL WriteDec

mWrite "/"

movzx EAX, localTime.wMonth

CALL WriteDec

mWrite "/"

movzx EAX, localTime.wYear

CALL WriteDec

CALL CRLF

CALL CRLF

RET

Time ENDP

Design PROC

; Open the file for input.

MOV EDX,OFFSET filename

CALL OpenInputFile

MOV fileHandle,EAX

; Check for errors.

CMP EAX,INVALID\_HANDLE\_VALUE ; error opening file?

jne file\_ok ; no: skip

mWrite <"Cannot open file",0dh,0ah>

JMP quit ; and quit

file\_ok:

; Read the file into a buffer.

MOV EDX,OFFSET buffer

MOV ECX,BUFFER\_SIZE

CALL ReadFromFile

jnc check\_buffer\_size ; error reading?

mWrite "Error reading file. " ; yes: show error message

CALL WriteWindowsMsg

JMP close\_file

check\_buffer\_size:

CMP EAX,BUFFER\_SIZE ; buffer large enough?

jb buf\_size\_ok ; yes

mWrite <"Error: Buffer too small for the file",0dh,0ah>

JMP quit ; and quit

buf\_size\_ok:

MOV buffer[EAX],0 ; insert null terminator

;mWrite "File size: "

;CALL WriteDec ; display file size

; Display the buffer.

mWrite <0dh,0ah,0dh,0ah>

MOV EDX,OFFSET buffer ; display the buffer

CALL WriteString

CALL CRLF

close\_file:

MOV EAX,fileHandle

CALL CloseFile

quit:

CALL CRLF

RET

Design ENDP

FrontPage PROC

CALL CRLF

CALL CRLF

mWrite " NAMAL UNIVERSITY"

CALL CRLF

mWrite " BSCS 3RD (SEMESTER)"

CALL CRLF

CALL CRLF

CALL CRLF

mWrite " COMPUTER ORGANIZATION & ASSEMBLY LANGUAGE "

CALL CRLF

CALL CRLF

CALL CRLF

mWrite " MINI-PROJECT : ATM BANK SIMULATOR"

CALL CRLF

CALL CRLF

mWrite " Batch : 2027"

CALL CRLF

CALL CRLF

CALL CRLF

CALL CRLF

CALL CRLF

mWrite " DevelopedBy : "

CALL CRLF

CALL CRLF

mWrite " HABIB UR REHMAN "

CALL CRLF

CALL CRLF

mWrite " RAYAN BADAR"

CALL CRLF

MOV EAX, 4000

CALL delay

CALL CLRSCR

RET

FrontPage ENDP

AdminLoginFUNC PROC

l1:

CALL CLRSCR

MOV EAX, 0

MOV EBX, 0

MOV ECX, 0

INVOKE Design

mWrite " [ ---------------- << LOGIN INTERFACE >> ---------------- ]"

CALL CRLF

CALL CRLF

mWrite " "

mWrite " Enter Administrator Username: "

MOV ECX, 20 ; Set the correct size of the username buffer

MOV EDX, OFFSET username

CALL readstring ; Input username

mWrite " "

mWrite " Enter Yours Password: "

MOV ECX, 20 ; Set the correct size of the password buffer

MOV EDX, OFFSET password

CALL readstring ; Input password

CALL CRLF

MOV EDI, OFFSET username

MOV ESI, OFFSET AdminUser

CMPSB

JE s1

mWrite " "

mWrite "[-] ERROR! Username or password is incorrect, please try again... "

CALL CRLF

mWrite " "

CALL WAITMSG

CALL CRLF

JMP l1

s1:

MOV ESI, OFFSET password

MOV EDI, OFFSET AdminPassword

CMPSB

JE s2

mWrite " "

mWrite " [-] ERROR! Username or password is incorrect, please try again... "

CALL CRLF

mWrite " Do you want to try again? (1 = Yes, 0 = No): "

CALL Readint

MOV AdminLogChoice, EAX

CALL CLRSCR

CMP AdminLogChoice, 1

JE l1 ; User wants to try again

JMP end\_it

s2:

mWrite " "

mWrite " [+] <<|>> Checking Admin Credentials!!! <<|>> [+]"

CALL CRLF

MOV EAX, 4000

CALL delay

CALL AdminOperation

end\_it:

RET

AdminLoginFUNC ENDP

AdminOperation PROC

CALL CRLF

CALL CRLF

mWrite " Admin Login Successful!"

CALL CRLF

CALL CRLF

CALL CRLF

mWrite " "

CALL WAITMSG

CALL CLRSCR

Back:

mWrite "\_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ Welcome Administrator \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_"

CALL CRLF

CALL CRLF

mWrite " Please select your choice : :"

CALL CRLF

CALL CRLF

mWrite " [1] Search User Detail"

CALL CRLF

CALL CRLF

mWrite " [2] ATM INFO"

CALL CRLF

CALL CRLF

mWrite " [3] Start Atm"

CALL CRLF

CALL CRLF

mWrite " [4] Turn OFF the System "

CALL CRLF

CALL CRLF

mWrite " Your choice: "

CALL ReadDec

JMP AdmOP

CALL CRLF

mWrite " "

JMP ExitAdminOperation

AdmOP:

CMP EAX, 1

JE SearchUserDetail

CMP EAX, 2

JE ATMINFO

CMP EAX, 3

JE StartAtm

CMP EAX, 4

JE TurnOFFtheSystem

JMP InvalidChoice

SearchUserDetail:

CALL CLRSCR

CALL CRLF

CALL CRLF

mWrite " Account Details"

CALL CRLF

mWrite " Enter Account Number you want to find Details :--> Single User Exist!!!"

CALL CRLF

CALL CRLF

mWrite "\_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ Account Holder Information & Detail \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_"

CALL CRLF

CALL CRLF

mWrite " Account Number: "

mov EDX, OFFSET StoredAccountID

CALL WriteString

CALL CRLF

mWrite " Holder Name: "

mov EDX, OFFSET StoredHolderName

CALL WriteString

CALL CRLF

mWrite " Current Balance: "

mov EAX, StoredAmount

CALL WriteDec

CALL CRLF

mWrite " Personal Identification Number: "

mov EDX, OFFSET StoredPIN

CALL WriteString

CALL CRLF

mWrite " ID Card Number: "

mov EDX, OFFSET StoredCNIC

CALL WriteString

CALL CRLF

mWrite " Contact Number: "

mov EDX, OFFSET StoredNO

CALL WriteString

CALL CRLF

mWrite " Address City: "

mov EDX, OFFSET StoredCity

CALL WriteString

CALL CRLF

mWrite " E-Mail Address: "

mov EDX, OFFSET StoredMail

CALL WriteString

CALL CRLF

mWrite " Gender: "

mov EDX, OFFSET gender

CALL WriteString

CALL CRLF

mWrite " "

CALL WAITMSG

CALL CLRSCR

JMP ExitAdminOperation

ATMINFO:

CALL CLRSCR

CALL CRLF

CALL CRLF

mWrite " ATM Info:"

CALL CRLF

mWrite " ----------------------"

CALL CRLF

mWrite " Bank : MCB BANK NAMAL"

CALL CRLF

mWrite " Branch Code : MAINWALI BRANCH"

CALL CRLF

mWrite " Model: ABC-1234"

CALL CRLF

mWrite " Serial No: S12345678"

CALL CRLF

mWrite " Available Cash: "

CALL CRLF

mWrite " Year: 2025"

CALL CRLF

CALL CRLF

mWrite " "

CALL WAITMSG

CALL CLRSCR

JMP ExitAdminOperation

StartAtm:

CALL CLRSCR

CALL CRLF

CALL CRLF

mWrite " Starting System..."

CALL CRLF

CALL CRLF

mWrite " HI :)"

CALL CRLF

CALL CRLF

mWrite " "

CALL WAITMSG

CALL HolderLoginFUNC

JMP ExitAdminOperation

TurnOFFtheSystem:

CALL CLRSCR

CALL CRLF

mWrite " Good Bye Admin..."

CALL CRLF

CALL CRLF

mWrite " Have a Nice Day :)"

CALL CRLF

mWrite " "

CALL WAITMSG

INVOKE EXITPROCESS, 0

InvalidChoice:

CALL CLRSCR

JMP AdminOperation

ExitAdminOperation:

JMP Back

RET

AdminOperation ENDP

UserOperations PROC

mWrite " Account Holder Login Account Found!"

CALL CRLF

CALL CRLF

CALL CRLF

mWrite " "

CALL WAITMSG

CALL CLRSCR

Back:

mWrite " Welcome User "

CALL CRLF

mWrite " Enter your choice : "

CALL CRLF

CALL CRLF

mWrite " [1] Fast Cash"

CALL CRLF

CALL CRLF

mWrite " [2] Cash Withdrawal"

CALL CRLF

CALL CRLF

mWrite " [3] Deposit"

CALL CRLF

CALL CRLF

mWrite " [4] Transactions "

CALL CRLF

CALL CRLF

mWrite " [5] Account Detail"

CALL CRLF

CALL CRLF

mWrite " [6] Change PIN "

CALL CRLF

CALL CRLF

mWrite " [7] Update Account INFO "

CALL CRLF

CALL CRLF

mWrite " [8] Generate Receipt"

CALL CRLF

CALL CRLF

mWrite " [9] Log out"

CALL CRLF

CALL CRLF

CALL CRLF

mWrite " [->] Your Choice : "

CALL ReadInt

MOV UserOption, EAX

CMP UserOption, 1

JE FastCash

CMP UserOption, 2

JE CashWithdrawal

CMP UserOption, 3

JE Deposit

CMP UserOption, 4

JE Transactions

CMP UserOption, 5

JE AccountDetail

CMP UserOption, 6

JE ChangePIN

CMP UserOption, 7

JE UpdateAccountInfo

CMP UserOption, 8

JE GenerateReceipt

CMP UserOption, 9

JE LogOut

mWrite " Invalid Choice..."

CALL WAITMSG

CALL CLRSCR

JMP Back

FastCash:

CALL CLRSCR

CALL QuickCash

CALL CRLF

mWrite " "

CALL WAITMSG

CALL CLRSCR

JMP Back

CashWithdrawal:

CALL CLRSCR

CALL WithdrawalCash

CALL CRLF

mWrite " "

CALL WAITMSG

CALL CLRSCR

JMP Back

Deposit:

CALL CLRSCR

CALL DepositCash

CALL CRLF

mWrite " "

CALL WAITMSG

CALL CLRSCR

JMP Back

Transactions:

CALL CLRSCR

CALL TransferAmount

CALL CRLF

mWrite " "

CALL WAITMSG

CALL CLRSCR

JMP Back

AccountDetail:

CALL CLRSCR

CALL AccountINFO

CALL CRLF

mWrite " "

CALL WAITMSG

CALL CLRSCR

JMP Back

ChangePIN:

CALL CLRSCR

CALL ChangeUserPIN

CALL CRLF

mWrite " "

CALL WAITMSG

CALL CLRSCR

JMP Back

UpdateAccountInfo:

CALL CLRSCR

CALL UpdateINFO

CALL CRLF

mWrite " "

CALL WAITMSG

CALL CLRSCR

JMP Back

GenerateReceipt:

CALL CLRSCR

CALL GenerateTransactionReceipt

CALL CRLF

mWrite " "

CALL WAITMSG

CALL CLRSCR

JMP Back

LogOut:

CALL CLRSCR

INVOKE Design

mWrite " THANKS FOR VISITING :)"

CALL CRLF

CALL CRLF

mWrite " Logged Out Successfully!!!"

CALL CRLF

mWrite " "

CALL WAITMSG

CALL CLRSCR

CALL LoggedOut

UserOperations ENDP

HolderLoginFUNC PROC

l1:

CALL CLRSCR

INVOKE Design

MOV EAX, 0

MOV EBX, 0

MOV ECX, 0

mWrite " [ ---------------- << LOGIN INTERFACE >> ---------------- ]"

CALL CRLF

CALL CRLF

mWrite " "

mWrite " Enter Account Number: "

MOV ECX, 20

MOV EDX, OFFSET HolderUsername

CALL readstring

mWrite " "

mWrite " Enter Your PIN: "

MOV ECX, 20

MOV EDX, OFFSET HolderPassword

CALL readstring

CALL CRLF

MOV EDI, OFFSET HolderUsername

MOV ESI, OFFSET StoredAccountID

CMPSB

JE s1

mWrite " "

mWrite "[-] ERROR! HolderUsername or HolderPassword is incorrect, please try again... "

CALL CRLF

mWrite " "

CALL WAITMSG

CALL CRLF

JMP l1

s1:

MOV ESI, OFFSET HolderPassword

MOV EDI, OFFSET StoredPIN

CMPSB

JE s2

mWrite " "

mWrite " [-] ERROR! HolderUsername or HolderPassword is incorrect, please try again... "

CALL CRLF

mWrite " Do you want to try again … ? (1 = Yes, 0 = No): "

CALL Readint

MOV HolderLogChoice, EAX

CALL CLRSCR

CMP HolderLogChoice, 1

JE l1 ; User wants to try again

JMP end\_it

s2:

mWrite " "

mWrite " [+] <<|>> Checking Entered Credentials...!!! <<|>> [+]"

CALL CRLF

MOV EAX, 4000

CALL delay

INVOKE UserOperations

end\_it:

RET

HolderLoginFUNC ENDP

QuickCash PROC

mWrite "\_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ Account Holder Information & Detail \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_"

CALL CRLF

mWrite " Account Number: "

MOV EDX, OFFSET StoredAccountID

CALL WriteString

CALL CRLF

CALL CRLF

mWrite " Current Balance: "

MOV EAX, StoredAmount

CALL WriteDec

CALL CRLF

CALL CRLF

CALL CRLF

; Check balance for Quick Cash

.if (StoredAmount > 500)

mWrite " Please select the Withdrawal amount:"

CALL CRLF

CALL CRLF

mWrite " 1. Rs. 500"

CALL CRLF

CALL CRLF

mWrite " 2. Rs. 1000 "

CALL CRLF

CALL CRLF

mWrite " 3. Rs. 1500 "

CALL CRLF

CALL CRLF

mWrite " 4. Rs. 2000 "

CALL CRLF

CALL CRLF

mWrite " 5. Rs. 2500 "

CALL CRLF

CALL CRLF

mWrite " 6. Rs. 3000 "

CALL CRLF

CALL CRLF

mWrite " Your choice: "

CALL ReadDec

MOV fcashChoice, EAX ; Store user's choice in fcashChoice

JMP FastCash

.else

mWrite " Your balance does not meet the minimum requirement for Quick Cash."

CALL CRLF

mWrite " "

JMP EndQuickCash

.endif

FastCash:

; Assuming fcashChoice is in EAX

CMP EAX, 1

JE Withdraw500

CMP EAX, 2

JE Withdraw1000

CMP EAX, 3

JE Withdraw1500

CMP EAX, 4

JE Withdraw2000

CMP EAX, 5

JE Withdraw2500

CMP EAX, 6

JE Withdraw3000

; Default case (if fcashChoice is not 1-6)

JMP InvalidChoice

Withdraw500:

CMP StoredAmount, 500

JL InsufficientBalance

SUB StoredAmount, 500

mWrite " Processing withdrawal of Rs. 500..."

MOV EAX, 4000

ADD GenerateBalance, 500

CALL delay

JMP DoneWithdrawal

Withdraw1000:

CMP StoredAmount, 1000

JL InsufficientBalance

SUB StoredAmount, 1000

mWrite " Processing withdrawal of Rs. 1000..."

MOV EAX, 4000

ADD GenerateBalance, 1000

CALL delay

JMP DoneWithdrawal

Withdraw1500:

CMP StoredAmount, 1500

JL InsufficientBalance

SUB StoredAmount, 1500

mWrite " Processing withdrawal of Rs. 1500..."

MOV EAX, 4000

ADD GenerateBalance, 1500

CALL delay

JMP DoneWithdrawal

Withdraw2000:

CMP StoredAmount, 2000

JL InsufficientBalance

SUB StoredAmount, 2000

mWrite " Processing withdrawal of Rs. 2000..."

MOV EAX, 4000

ADD GenerateBalance, 2000

CALL delay

JMP DoneWithdrawal

Withdraw2500:

CMP StoredAmount, 2500

JL InsufficientBalance

SUB StoredAmount, 2500

mWrite " Processing withdrawal of Rs. 2500..."

MOV EAX, 4000

ADD GenerateBalance, 2500

CALL delay

JMP DoneWithdrawal

Withdraw3000:

CMP StoredAmount, 3000

JL InsufficientBalance

SUB StoredAmount, 3000

mWrite " Processing withdrawal of Rs. 3000..."

MOV EAX, 4000

ADD GenerateBalance, 3000

CALL delay

JMP DoneWithdrawal

InsufficientBalance:

mWrite " Insufficient balance. Try another amount."

CALL CRLF

mWrite " "

CALL WAITMSG

CALL CLRSCR

JMP QuickCash

InvalidChoice:

CALL CLRSCR

JMP QuickCash

DoneWithdrawal:

CALL CRLF

CALL CRLF

mWrite " Remaining Balance : "

MOV EAX, StoredAmount

CALL WriteDec

JMP EndQuickCash

EndQuickCash:

RET

QuickCash ENDP

WithdrawalCash PROC

mWrite "\_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ Account Holder Information & Detail \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_"

CALL CRLF

mWrite " Account Number: "

MOV EDX, OFFSET StoredAccountID

CALL WriteString

CALL CRLF

CALL CRLF

mWrite " Current Balance: "

MOV EAX, StoredAmount

CALL WriteDec

CALL CRLF

CALL CRLF

CALL CRLF

; Check balance for Withdraw Cash

.if (StoredAmount > 500)

mWrite " Please select the Withdrawal amount:"

CALL CRLF

CALL CRLF

mWrite " 1. Rs. 200"

CALL CRLF

CALL CRLF

mWrite " 2. Rs. 500 "

CALL CRLF

CALL CRLF

mWrite " 3. Rs. 700 "

CALL CRLF

CALL CRLF

mWrite " 4. Other Amount "

CALL CRLF

CALL CRLF

mWrite " Your choice: "

CALL ReadDec

MOV wdcashChoice, EAX ; Store user's choice in wdcashChoice

JMP FastCash

.else

mWrite " Your balance does not meet the minimum requirement for Withdrawal Cash."

CALL CRLF

mWrite " "

JMP EndWithdraw

.endif

FastCash:

; Assuming wdcashChoice is in EAX

CMP EAX, 1

JE Withdraw200

CMP EAX, 2

JE Withdraw500

CMP EAX, 3

JE Withdraw700

CMP EAX, 4

JE WithdrawOtherAmount

; Default case (if wdcashChoice is not 1-4)

JMP InvalidChoice

Withdraw200:

CMP StoredAmount, 200

JL InsufficientBalance

SUB StoredAmount, 200

mWrite " Processing withdrawal of Rs. 200..."

MOV EAX, 4000

ADD GenerateBalance, 200

CALL delay

JMP DoneWithdrawal

Withdraw500:

CMP StoredAmount, 500

JL InsufficientBalance

SUB StoredAmount, 500

mWrite " Processing withdrawal of Rs. 500..."

MOV EAX, 4000

ADD GenerateBalance, 500

CALL delay

JMP DoneWithdrawal

Withdraw700:

CMP StoredAmount, 700

JL InsufficientBalance

SUB StoredAmount, 700

mWrite " Processing withdrawal of Rs. 700..."

MOV EAX, 4000

ADD GenerateBalance, 700

CALL delay

JMP DoneWithdrawal

WithdrawOtherAmount:

CALL CRLF

mWrite " Enter the Amount You want to Withdraw : "

CALL ReadInt

CMP StoredAmount, EAX

JL InsufficientBalance

SUB StoredAmount, EAX

mWrite " Processing withdrawal of Enterened Amount..."

ADD GenerateBalance, EAX

MOV EAX, 4000

CALL delay

JMP DoneWithdrawal

InsufficientBalance:

mWrite " Insufficient balance. Try another amount."

CALL CRLF

mWrite " "

CALL WAITMSG

CALL CLRSCR

JMP WithdrawalCash

InvalidChoice:

CALL CLRSCR

JMP WithdrawalCash

DoneWithdrawal:

CALL CRLF

CALL CRLF

mWrite " Remaining Balance : "

MOV EAX, StoredAmount

CALL WriteDec

JMP EndWithdraw

EndWithdraw:

RET

WithdrawalCash ENDP

DepositCash PROC

mWrite "\_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ Account Holder Information & Detail \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_"

CALL CRLF

mWrite " Account Number: "

MOV EDX, OFFSET StoredAccountID

CALL WriteString

CALL CRLF

CALL CRLF

mWrite " Current Balance: "

MOV EAX, StoredAmount

CALL WriteDec

CALL CRLF

CALL CRLF

CALL CRLF

mWrite " Please put the amount in the Deposit box : Rs. "

CALL ReadDec

MOV depositAmount, EAX ; Assuming depositAmount is a variable to store the deposit amount

; Check if deposit amount is invalid

CMP depositAmount, 0

jle InvalidDepositAmount

mWrite " Processing deposit of Rs. "

MOV EAX, depositAmount

CALL WriteDec

mWrite "..."

; Process deposit

ADD StoredAmount, EAX ; Assuming StoredAmount is a variable representing the balance

CALL CRLF

mWrite " Your New Balance is: "

MOV EAX, StoredAmount

CALL WriteDec

mWrite " Rs"

CALL CRLF

RET

InvalidDepositAmount:

mWrite " Invalid Deposit Amount. Please try again."

CALL CRLF

mWrite " "

CALL WAITMSG

CALL CLRSCR

; Jump back to get a valid deposit amount

JMP DepositCash

RET

DepositCash ENDP

TransferAmount PROC

; Display account information

mWrite "\_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ Account Holder Information & Detail \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_"

CALL CRLF

mWrite " Account Number: "

mov EDX, OFFSET StoredAccountID

CALL WriteString

CALL CRLF

CALL CRLF

mWrite " Current Balance: "

mov EAX, StoredAmount

CALL WriteDec

CALL CRLF

CALL CRLF

CALL CRLF

; Prompt for account number

mWrite " Please Enter the Account Number you want to Send Money (11 Digits): "

CALL ReadDec

; Loop for entering a valid transfer amount

EnterTransferAmount:

; Prompt for transfer amount

mWrite " Please Enter the Transaction Amount: Rs. "

CALL ReadDec

mov TransferMoney, EAX

ADD GenerateBalance, EAX

; Check for invalid transfer amount

CMP EAX, 0

jle InvalidTransferAmount

CMP EAX, StoredAmount

jg InvalidTransferAmount

; Process the transaction

mWrite " Processing Transaction of Rs. "

mov EAX, TransferMoney

CALL WriteDec

mWrite "..."

SUB StoredAmount, EAX

CALL CRLF

CALL CRLF

; Display new balance

mWrite " Your New Balance is: "

mov EAX, StoredAmount

CALL WriteDec

mWrite " Rs"

CALL CRLF

JMP EndTransferAmount

InvalidTransferAmount:

; Display error message for invalid transfer amount

mWrite " Invalid Transfer Amount. Please try again."

CALL CRLF

mWrite " "

CALL WAITMSG

CALL CLRSCR

JMP EnterTransferAmount

EndTransferAmount:

RET

TransferAmount ENDP

AccountINFO PROC

; Display account information

mWrite "\_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ Account Holder Information & Detail \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_"

CALL CRLF

CALL CRLF

mWrite " Account Number: "

mov EDX, OFFSET StoredAccountID

CALL WriteString

CALL CRLF

mWrite " Holder Name: "

mov EDX, OFFSET StoredHolderName

CALL WriteString

CALL CRLF

mWrite " Current Balance: "

mov EAX, StoredAmount

CALL WriteDec

CALL CRLF

mWrite " Personal Identification Number: "

mov EDX, OFFSET StoredPIN

CALL WriteString

CALL CRLF

mWrite " ID Card Number: "

mov EDX, OFFSET StoredCNIC

CALL WriteString

CALL CRLF

mWrite " Contact Number: "

mov EDX, OFFSET StoredNO

CALL WriteString

CALL CRLF

mWrite " Address City: "

mov EDX, OFFSET StoredCity

CALL WriteString

CALL CRLF

mWrite " E-Mail Address: "

mov EDX, OFFSET StoredMail

CALL WriteString

CALL CRLF

mWrite " Gender: "

mov EDX, OFFSET gender

CALL WriteString

RET

AccountINFO ENDP

ChangeUserPIN PROC

mWrite "\_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ Account Holder Information & Detail \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_"

CALL CRLF

CALL CRLF

mWrite " Enter New Personal Identification Number(PIN) :"

mov EDX,OFFSET StoredPIN ; point to the buffer

mov ECX,SIZEOF StoredPIN ; specify max characters

CALL ReadString ; input the string

CALL CRLF

CALL CRLF

mWrite " Personal Identification Number(PIN) Successfully changed to "

mov EDX,OFFSET StoredPIN

CALL WriteString

RET

ChangeUserPIN ENDP

UpdateINFO PROC

mWrite "\_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ Account Holder Information & Detail \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_"

CALL CRLF

CALL CRLF

mWrite " Please select the Info you want to Change :"

CALL CRLF

CALL CRLF

mWrite " 1. Your Name"

CALL CRLF

CALL CRLF

mWrite " 2. Your Phone No. "

CALL CRLF

CALL CRLF

mWrite " 3. Your Address "

CALL CRLF

CALL CRLF

mWrite " 4. Your MAIL "

CALL CRLF

CALL CRLF

mWrite " 5. Your Gender "

CALL CRLF

CALL CRLF

mWrite " Your choice: "

CALL ReadDec

JMP InfoJMP

CALL CRLF

mWrite " "

JMP ExitUpdation

InfoJMP:

CMP EAX, 1

JE YourName

CMP EAX, 2

JE YourPhoneNo

CMP EAX, 3

JE YourAddress

CMP EAX, 4

JE YourMAIL

CMP EAX, 5

JE YourGender

JMP InvalidChoice

YourName:

mWrite " Enter New User Name :"

MOV EDX,OFFSET StoredHolderName ; point to the buffer

MOV ECX,SIZEOF StoredHolderName ; specify max characters

CALL ReadString ; input the string

CALL CRLF

CALL CRLF

mWrite " User Name changed to "

MOV EDX,OFFSET StoredHolderName

CALL WriteString

JMP ExitUpdation

YourPhoneNo:

mWrite " Enter New Phone no :"

MOV EDX,OFFSET StoredNO ; point to the buffer

MOV ECX,SIZEOF StoredNO ; specify max characters

CALL ReadString ; input the string

CALL CRLF

CALL CRLF

mWrite " Phone no Successfully changed to "

MOV EDX,OFFSET StoredNO

CALL WriteString

JMP ExitUpdation

YourAddress:

mWrite " Enter New Address :"

MOV EDX,OFFSET StoredCity ; point to the buffer

MOV ECX,SIZEOF StoredCity ; specify max characters

CALL ReadString ; input the string

CALL CRLF

CALL CRLF

mWrite " Address Successfully changed to "

MOV EDX,OFFSET StoredCity

CALL WriteString

JMP ExitUpdation

YourMAIL:

mWrite " Enter Your MAIL :"

MOV EDX,OFFSET StoredMail ; point to the buffer

MOV ECX,SIZEOF StoredMail ; specify max characters

CALL ReadString ; input the string

CALL CRLF

CALL CRLF

mWrite " Your MAIL Successfully changed to "

MOV EDX,OFFSET StoredMail

CALL WriteString

JMP ExitUpdation

YourGender:

mWrite " Enter Gender (F/M/O):"

MOV EDX,OFFSET gender ; point to the buffer

MOV ECX,SIZEOF gender ; specify max characters

CALL ReadString ; input the string

CALL CRLF

CALL CRLF

mWrite " Gender Successfully changed to "

MOV EDX,OFFSET gender

CALL WriteString

JMP ExitUpdation

InvalidChoice:

CALL CLRSCR

JMP UpdateINFO

ExitUpdation:

RET

UpdateINFO ENDP

GenerateTransactionReceipt PROC

mWrite "\_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ Transaction Receipt \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_"

CALL CRLF

CALL CRLF

.if (GenerateBalance > 0)

mWrite " Account Number: "

mov EDX, OFFSET StoredAccountID

CALL WriteString

CALL CRLF

mWrite " Holder Name: "

mov EDX, OFFSET StoredHolderName

CALL WriteString

CALL CRLF

mWrite " Remaining Balance: "

mov EAX, StoredAmount

CALL WriteDec

CALL CRLF

mWrite " Transaction of Rs : "

mov EAX, GenerateBalance

CALL WriteDec

CALL CRLF

mWrite " Contact Number: "

mov EDX, OFFSET StoredNO

CALL WriteString

CALL CRLF

mWrite " Address City: "

mov EDX, OFFSET StoredCity

CALL WriteString

CALL CRLF

mWrite " E-Mail Address: "

mov EDX, OFFSET StoredMail

CALL WriteString

.else

mWrite " Please Perform any Transaction to generate Receipt!!!"

CALL CRLF

mWrite " "

CALL WAITMSG

.endif

RET

GenerateTransactionReceipt ENDP

LoggedOut PROC

CALL HolderLoginFUNC

RET

LoggedOut ENDP

main PROC

INVOKE AddColor

INVOKE FrontPage

INVOKE Time

INVOKE Design

mWrite " Please INSERT your Card [Press any Key to insert card]"

CALL CRLF

CALL CRLF

mWrite " "

CALL WAITMSG

CALL CLRSCR

try\_again:

INVOKE Design

CALL CRLF

mWrite " LOG IN:"

CALL CRLF

CALL CRLF

mWrite " [1] Administrator"

CALL CRLF

CALL CRLF

mWrite " [2] Account Holder"

CALL CRLF

CALL CRLF

mWrite " [3] EXIT"

CALL CRLF

CALL CRLF

mWrite " [->] YOUR CHOICE : "

CALL readint

MOV choice, EAX

CMP choice, 1

JE case\_1

CMP choice, 2

JE case\_2

CMP choice, 3

JE case\_3

CALL CRLF

mWrite " Invalid choice!"

CALL CRLF

mWrite " Do you want to try again? (1 = Yes, 0 = No): "

CALL Readint

MOV choice, EAX

CALL CLRSCR

CMP choice, 1

JE try\_again ; User wants to try again

JMP end\_it

case\_1:

INVOKE AdminLoginFUNC

JMP end\_it

case\_2:

CALL CLRSCR

CALL HolderLoginFUNC

JMP end\_it

case\_3:

EXIT

end\_it:

CALL CRLF

CALL CRLF

INVOKE EXITPROCESS, 0

main ENDP

**4. Results and Findings**

**Program Execution and Outputs**

The **Bank ATM Simulation** was successfully implemented and executed using MSAM 32 and Irvine 32 in Visual Studio. The following results were observed:

1. **Login System:**

The program correctly validated credentials for both administrators and account holders.

Upon providing valid credentials, users were granted access to their respective menus.

Invalid credentials resulted in appropriate error messages and retry options.

1. **Administrator Functionalities:**

The admin was able to search for user details, view ATM information, and manage the banking system effectively.

Administrative options such as starting the ATM or turning off the system worked as intended.

1. **Account Holder Functionalities:**

Users were able to perform transactions such as deposits, withdrawals, and transfers with accurate balance updates.

Account details, such as account number, balance, and personal information, were displayed correctly.

PIN changes and account information updates were implemented without errors.

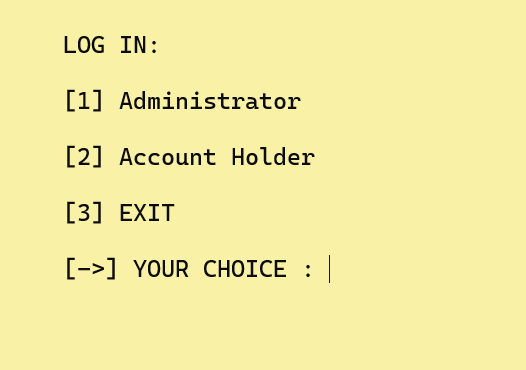
1. **Transaction Handling:**

Fast cash options provided predefined amounts for withdrawal, and all transactions were executed successfully, with the system checking for sufficient balance.

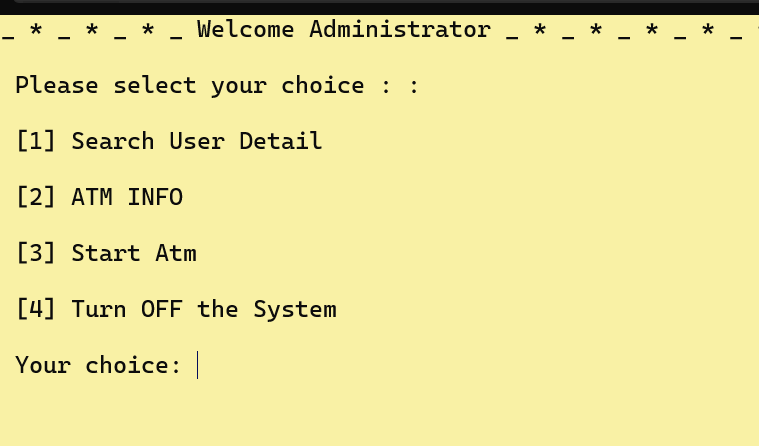
Deposits and custom withdrawals updated the account balance accurately and displayed appropriate success messages.

**Screenshots**

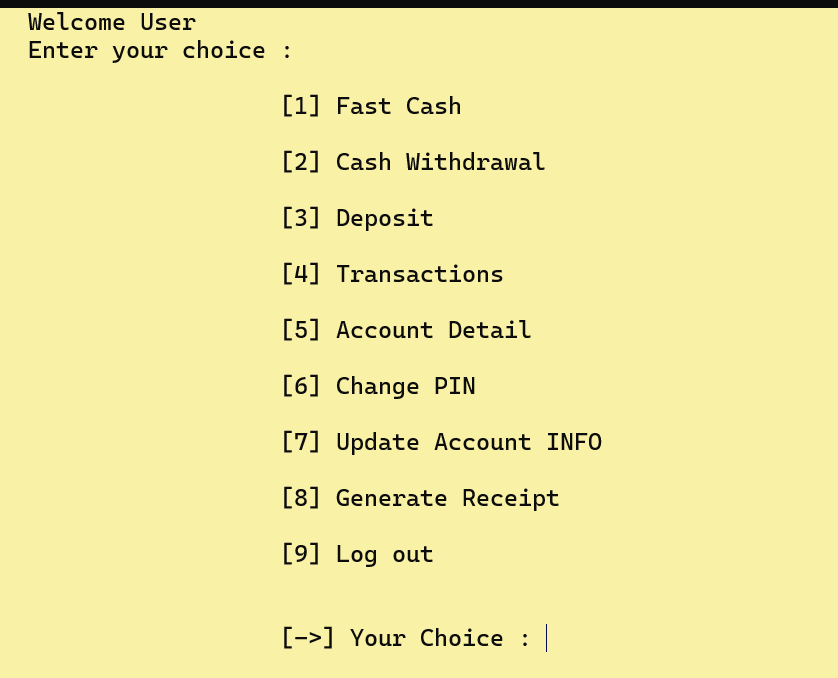
* **Login Screen:**  
  Displays options for Administrator or Account Holder login.



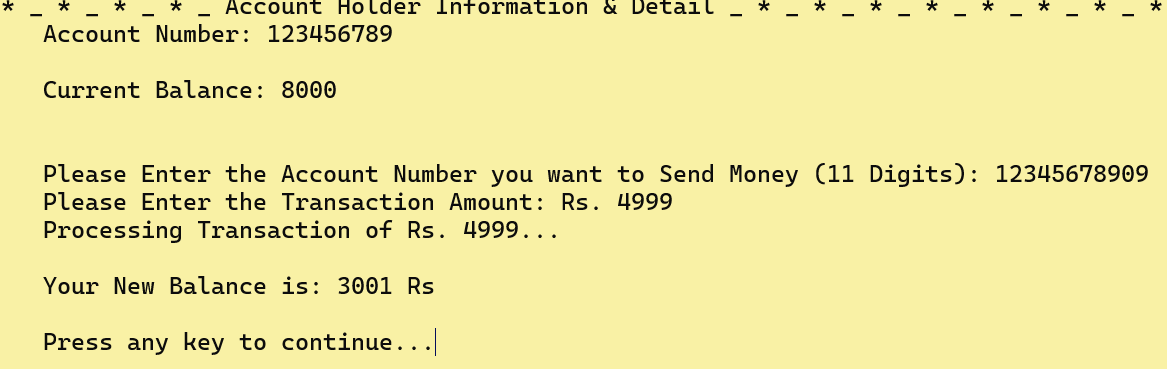
* **Administrator Menu:**  
  Displays admin-specific options, such as searching user details or viewing ATM info.



* **Account Holder Menu:**  
  Displays options for fast cash, deposits, withdrawals, transfers, and account details.



* **Transaction Success:**  
  Shows a success message after completing a withdrawal or deposit.



**Observations**

1. **Functionality:**

The program handled user inputs seamlessly, ensuring smooth navigation through menus and accurate execution of banking operations.

Errors, such as insufficient balance or invalid inputs, were managed effectively, maintaining program stability.

1. **Efficiency:**

The use of modular procedures allowed for efficient execution and minimized redundant code.

Fixed-size buffers and proper memory management ensured optimal performance, even with large data inputs.

1. **Optimization:**

The use of assembly language made the program lightweight and fast, showcasing the potential of low-level programming for critical systems.

**5. Conclusion**

The **Bank ATM Simulation** project effectively demonstrates how assembly language can be utilized to design and implement critical system functionalities. Through its modular and structured approach, the program successfully addresses the problem statement by providing essential banking operations such as deposits, withdrawals, balance inquiries, and account management. The solution highlights the importance of understanding low-level programming to build efficient and reliable systems.

**Key Findings**

* The system correctly handled user authentication for both administrators and account holders, ensuring secure access to the program’s features.
* Core banking functionalities, including transaction processing and balance updates, were implemented accurately, demonstrating the program's effectiveness.
* The use of assembly language provided a deeper understanding of hardware-level operations, file handling, and memory management.

**Effectiveness**

The proposed solution addressed the challenges outlined in the problem statement:

* Precise handling of user inputs and file operations ensured data accuracy and program stability.
* The modular design enhanced code clarity and maintainability.
* The implementation showcased assembly language's potential for creating lightweight and efficient systems.

**Limitations**

Despite its success, the project encountered the following limitations:

1. The system supports only a single user and lacks dynamic account creation.
2. The user interface is text-based, which limits its accessibility and appeal compared to graphical interfaces.
3. Data security features, such as encryption, were not implemented.

**Future Improvements**

To enhance the system further, the following improvements can be considered:

1. Adding functionality for creating, updating, and deleting user accounts dynamically.
2. Implementing file encryption to secure sensitive data such as PINs and account details.
3. Transitioning to a graphical user interface for an improved user experience.
4. Expanding the system to handle multiple users with real-time data updates.

In conclusion, the **Bank ATM Simulation** successfully fulfills its objectives, providing a strong foundation for learning and applying assembly language programming to real-world scenarios. The project demonstrates the practical relevance of low-level programming in building reliable and efficient systems, while also highlighting areas for future exploration and improvement.