

1/13/2024

**PROJECT PROPOSAL**

**COAL-LAB**

**Submitted to:** MAAM MARIA ALI

**Submitted by:** Musharib Rehman [032]

Muhammad Rizwan [409]

Awais Anser [181]

Umair Noor [339]

Abdullah Rashid [079]

**“BANK MANAGEMENT SYSTEM”**

# INTRODUCTION

This assembly code is designed to simulate a basic banking system where users can log in, perform various transactions like depositing money, withdrawing money, transferring funds, checking balance, and leave the program. Users are prompted to enter their ID and password for authentication. The system has a predefined set of account IDs (IDS1 and IDS2) and corresponding passwords (PASSWORDS1 and PASSWORDS2). After successful authentication, users can choose from various banking operations presented in a menu.

# PROCEDURE

* **Data Segment:**

The DATA SEGMENT section declares various data structures, constants, and messages used in the program.

It includes account IDs (IDS1 and IDS2), corresponding passwords (PASSWORDS1 and PASSWORDS2), messages for user prompts and responses, and variables like ACCOUNT\_BALANCE, IDINPUT, PASSINPUT, and TRIES\_LEFT.

* **Code Segment:**

The CODE SEGMENT contains the main logic and program flow.

The program starts at the START label and initializes the data segment (MOV AX, DATA and MOV DS, AX).

The user is prompted to choose between login and signup options. If the user selects login, they need to enter their ID and password for authentication. If they choose signup, they provide a new ID and password.

* **User Authentication (Login):**

The program enters a loop labeled AGAIN where the user is prompted to enter their ID. The entered ID is compared with predefined account IDs (IDS1 and IDS2).

If a match is found, the program proceeds to ask for the corresponding password. The entered password is then compared with the predefined passwords (PASSWORDS1 and PASSWORDS2).

If the ID and password match, the user is granted access (ALLOWED\_MESSAGE), and the program proceeds to the main menu. Otherwise, the user may have multiple attempts (TRIES\_LEFT), and if exhausted, access is denied (DENY\_ACCESS).

* **User Authentication (Signup):**

In the signup process, the user is prompted to enter a new ID and password. The entered ID and password are then stored for future authentication.

* **Main Menu:**

After successful login or signup, the program displays a menu with various options such as depositing money, withdrawing money, transferring funds, checking balance, about us, references for project and leaving the program.

The user's choice is obtained using the SCAN\_NUM function, and the program proceeds to execute the chosen operation based on the user input.

* **Banking Operations:**

The program includes operations like depositing money (ADD\_MONEY), withdrawing money (WITHDRAW\_MONEY), transferring funds (TRANSFER\_MONEY), and checking balance (CHECK\_BALANCE).

Each operation involves user input, and the program performs corresponding calculations and updates the ACCOUNT\_BALANCE accordingly.

* **Error Handling:**

The program includes basic error handling. For example, if the user enters the wrong password multiple times, they may be denied access (DENY\_ACCESS). If there are insufficient funds during a fund transfer, an error message is displayed (TRANSFER\_ERROR).

* **Exit Program:**

The program includes a LEAVE\_PROGRAM option to exit the program gracefully.

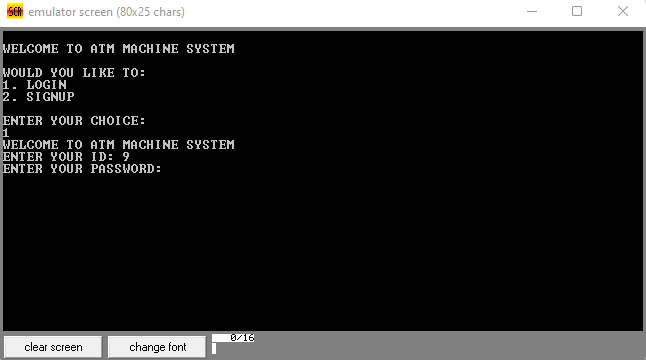
* **Code Ending:**

The program ends with the END START statement.

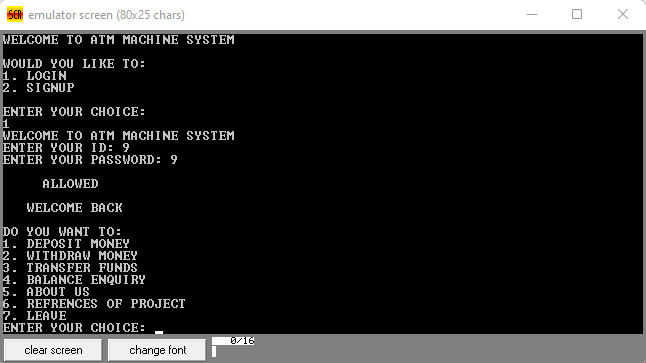
# OUPUTS

**Program Entry:**

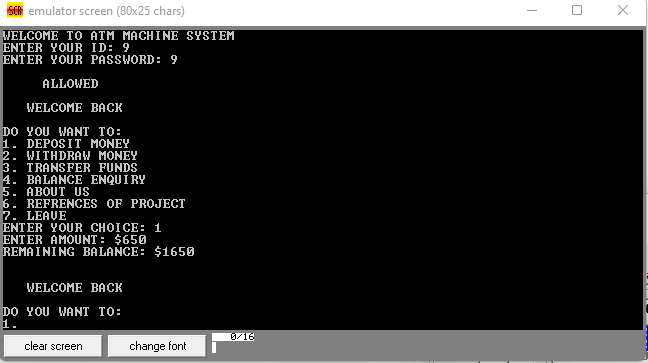
After running program it asks user for its choice to login or signup. That it then process from SCAN\_NOW function and check what user entered and perform that task.

**After Login:**

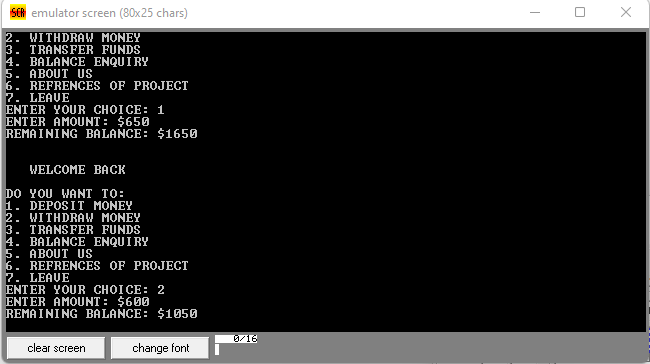
After pressing 1 it leads to user login. It then ask for user id and password that are already predefined in the program. Program checks the id and password with SCAN\_NOW function from IDINPUT and PASSINPUT defined in data segment.

**Main Menu**:  


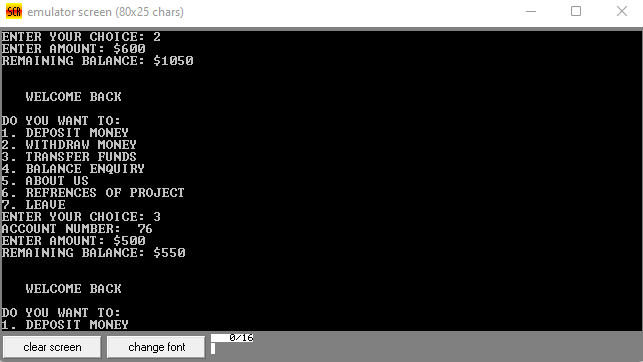
After successfully login , it shows a menu with different options. As shown in above figure. It then ask user for its choice which is processed by SCAN\_NOW. And then it is transferred to that function.

**Deposit**:  


At number 1 we have DEPOSIT option which on clicking ask user for the amount to be deposited. And then shows the remaining balance. Then MENU

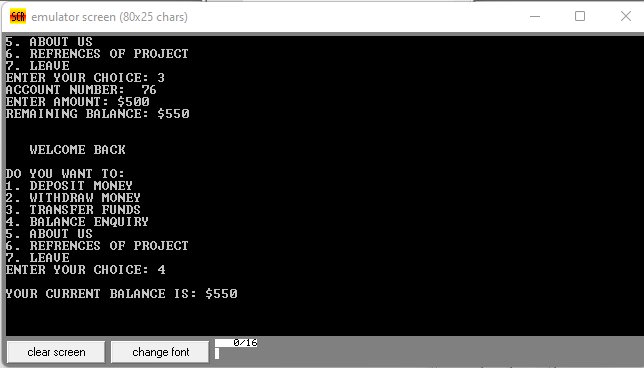
**Withdraw**:  


In option 2 WITHDRAW it ask user for the amount he wants to withdraw. And then show remaining balance. And moves to MENU.

**Transfer Funds**:  


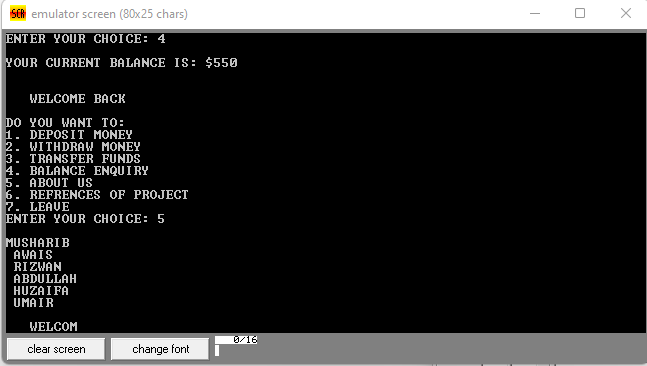
At number 3 we got TRANSFER FUNDS. If user wants to transfer funds it first ask user to enter account number to which he is trying to transfer money and then amount for transfer. Then show remaining balance and show MENU.

**Balance Enquiry:**

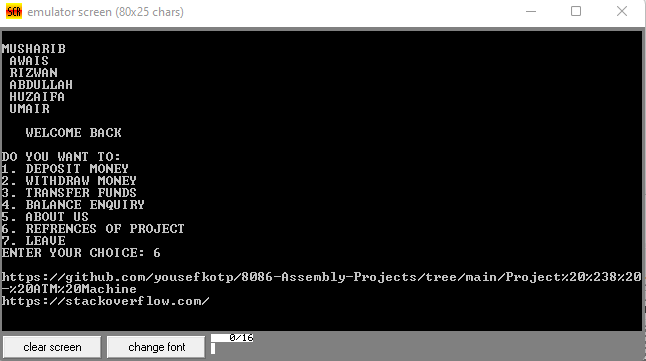


BALANCE INQUIRY is at number 4. After pressing 4 it show users current account balance and then return to MENU.

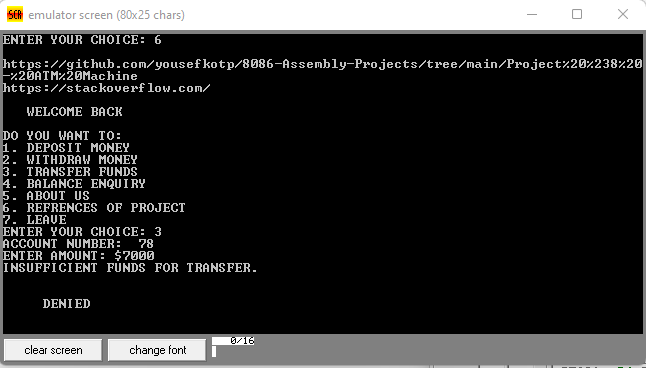
**About Us:**



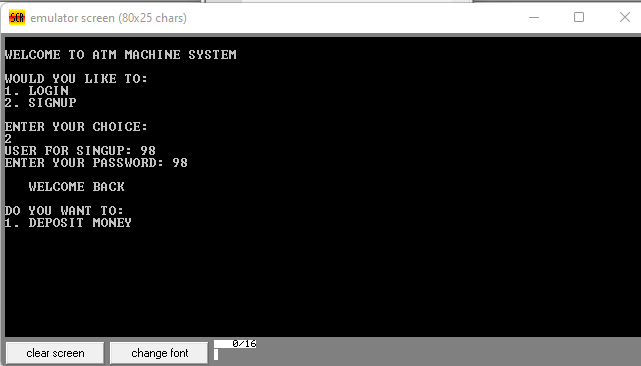
In ABOUT US which is number 5, there are names of group members for this project.

**References:**

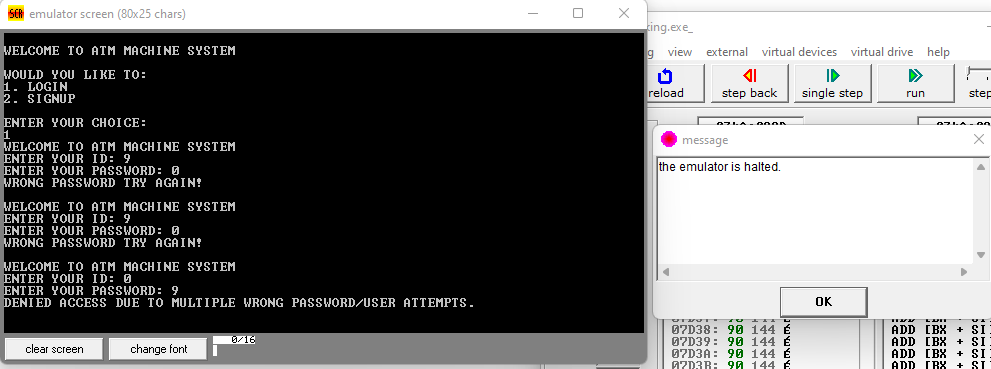
Second last is REFERENCE at 6. It show link where the program was actually given with very little working mechanism which was later on modified by group and other link which was helpful in making this project.

**In-Sufficient Funds**:  


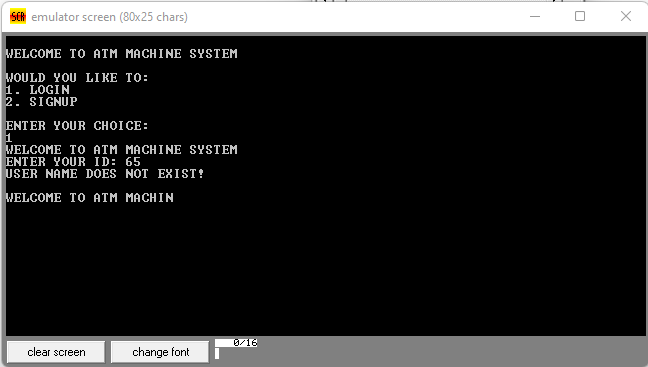
INSUFFICIENT FUNDS if the amount entered by user for transfer is greater than is accounts current balance then it denies the transaction and returns to MENU.

**Signup**:  


At entry point there are 2 option one is login second id SIGNUP. After selecting this option it ask user for its user id and its password.

**Multiple Wrong Password**:  


There is mechanism in work if a user enter wrong password or user name multiple times, here 3 times, it deny access to the user and closes the program.

**Invalid Username**:  


If user enters the wrong user id which is not in the predefined ids given in the program then it will print does not exist message and ask user again wether to login or signup.

# BENEFITS

**Educational Value:**

The code serves as an educational tool for learning assembly language programming, particularly for the x86 architecture. It provides a practical example that demonstrates essential concepts like input/output operations, control flow, and data manipulation in assembly language.

**Understanding Low-Level Operations:**

Assembly language allows programmers to interact directly with the hardware, gaining a deeper understanding of low-level operations. This can be valuable for those interested in systems programming, embedded systems, or understanding the internals of computer architecture.

**Simulation and Testing:**

The code is designed to run on the emu8086 emulator, which allows for simulation and testing of x86 assembly programs in a controlled environment. This can be useful for learners who want to experiment with assembly code without requiring physical hardware.

**Teaching Security Concepts:**

The code includes a simple user authentication mechanism, introducing the concept of secure user access. This can be a starting point for discussions on security practices, password protection, and potential vulnerabilities in real-world systems.

**Concepts of Banking Operations:**

The code demonstrates basic banking operations such as depositing money, withdrawing money, transferring funds, and checking balances. This can be useful for illustrating how assembly language can be applied to implement real-world functionality.

**Error Handling:**

The inclusion of error-handling mechanisms, such as limiting the number of password attempts and displaying appropriate messages for denied access, highlights the importance of error checking and handling in programming.

**Customizable for Learning:**

Learners can modify and expand upon the code to experiment with additional features, improve security measures, or add more sophisticated banking operations. This hands-on experience can enhance comprehension and problem-solving skills.

# LIMITATIONS

**Security Concerns:**

The code implements a basic user authentication system, but it lacks robust security measures. Real-world banking systems require sophisticated security mechanisms to protect user data, prevent unauthorized access, and handle encryption for secure communication.

**Lack of Input Validation:**

The code does not perform extensive input validation. In a production environment, input validation is crucial to prevent various types of attacks, such as buffer overflows, injection attacks, or unintended behavior due to invalid inputs.

**Limited Error Handling:**

While the code includes some error-handling mechanisms, it does not cover all potential error scenarios. Robust error handling is crucial in production systems to gracefully handle unexpected situations and provide meaningful feedback to users.

**Limited Functionality:**

The provided code covers only basic banking operations. Real-world banking systems require a more extensive set of features, such as account management, transaction logging, user profiles, and integration with other banking services.

**Platform-Specific Code:**

The code is specific to the emu8086 emulator and x86 architecture. Real-world applications need to be platform-independent and adhere to industry standards. Assembler code, in general, lacks portability across different architectures.

**Scalability Issues:**

The code may not scale well for a large number of users or transactions. Scalability is a crucial consideration for systems that need to handle a significant volume of users and transactions concurrently.

**Maintenance Challenges:**

Assembly code can be challenging to maintain and modify, especially as the complexity of the system increases. For large-scale projects, higher-level programming languages with more abstraction and maintainability are preferred.

**Limited User Interface:**

The user interface in this code is console-based and lacks the graphical interface commonly found in modern applications. Real-world banking systems often require user-friendly interfaces to accommodate a diverse range of users.

**Complexity:**

Assembly language, by its nature, is low-level and requires a deep understanding of hardware architecture. This complexity may pose challenges for developers, and modern high-level languages are often preferred for productivity and code readability.

# CONCLUSION

In conclusion, the provided assembly code for a simple banking system implemented in emu8086 serves as an educational tool, offering insights into x86 assembly language programming and foundational concepts. The code demonstrates user authentication, menu-driven interactions, and basic banking operations. However, it has notable limitations that make it unsuitable for real-world banking applications.

The key takeaways from this code are:

1. Educational Value:

The code provides a practical example for learners to understand assembly language concepts, such as low-level operations, user input/output, and control flow.

2. Limited Functionality: The code covers basic banking operations, but real-world banking systems require more sophisticated features, security measures, and scalability.

3. Security Concerns:

The code lacks robust security measures, input validation, and mechanisms for handling sensitive user data securely.

4. Platform Dependency:

It is specific to the emu8086 emulator and x86 architecture, making it non-portable and unsuitable for deployment in diverse computing environments.

5. Scalability and Maintenance Challenges:

Assembly code is difficult to scale for large user bases and complex functionalities, and it poses challenges for maintenance and modifications.

6. User Interface:

The console-based user interface is rudimentary, lacking the graphical interfaces commonly expected in modern applications.

7. Preparation for Further Learning:

The code can serve as a starting point for learners to delve deeper into assembly language or explore related topics in systems programming.

In real-world scenarios, the development of banking systems would be better approached using higher-level programming languages, industry-standard security practices, and adherence to regulatory requirements. The limitations of this code underscore the importance

# CODE

include 'emu8086.inc'

JMP START

DATA SEGMENT

TOTAL DW 20

IDS1 DW 0000H,0001H,0002H,0003H,0004H,0005H,0006H,0007H,0008H,0009H

IDS2 DW 000AH,000BH,000CH,000DH,000EH,000FH,0010H,0011H,0012H,0013H

PASSWORDS1 DB 00H, 01H, 02H, 03H, 04H, 05H, 06H, 07H, 08H, 09H

PASSWORDS2 DB 0AH, 0BH, 0CH, 0DH, 0EH, 0FH, 01H, 02H, 03H, 04H

DATA1 DB 0DH,0AH,'WELCOME TO ATM MACHINE SYSTEM',0

DATA2 DB 0DH,0AH,'ENTER YOUR ID: ',0

DATA3 DB 0DH,0AH,'ENTER YOUR PASSWORD: ',0

DATA4 DB 0DH,0AH,' DENIED ',0

DATA5 DB 0DH,0AH,' ALLOWED ',0

DATA6 DB 0DH,0AH,' WELCOME BACK ',0

DATA7 DB 0DH,0AH,'DO YOU WANT TO: ',0

DATA8 DB '1. DEPOSIT MONEY', 0

DATA9 DB '2. WITHDRAW MONEY', 0

DATA10 DB '3. TRANSFER FUNDS', 0

DATA11 DB 0DH,0AH,'ENTER YOUR CHOICE: ',0

DATA12 DB 0DH,0AH,'ENTER AMOUNT: $', 0

DATA13 DB 0DH,0AH,'REMAINING BALANCE: $', 0

DATA14 DB 0DH,0AH,'DENIED ACCESS DUE TO MULTIPLE WRONG PASSWORD/USER ATTEMPTS.', 0

DATA15 DB '7. LEAVE ', 0

DATA16 DB 0DH,0AH,'ACCOUNT NUMBER: ', 0

DATA17 DB 0DH,0AH,'INSUFFICIENT FUNDS FOR TRANSFER. ', 0

DATA18 DB 0DH,0AH,'WRONG PASSWORD TRY AGAIN! ',0

DATA19 DB '4. BALANCE ENQUIRY ', 0

DATA20 DB 0DH,0AH,'YOUR CURRENT BALANCE IS: $', 0

DATA21 DB 0DH,0AH,'USER NAME DOES NOT EXIST!', 0

DATA22 DB 0DH,0AH,'1. LOGIN', 0

DATA23 DB 0DH,0AH,'2. SIGNUP', 0

DATA24 DB 0DH,0AH,'WOULD YOU LIKE TO: ', 0

DATA25 DB 0DH,0AH,'USER FOR SINGUP: ',0

DATA26 DB 0DH,0AH,'ENTER YOUR PASSWORD: ',0

DATA27 DB '5. ABOUT US', 0

DATA28 DB '6. REFRENCES OF PROJECT', 0

DATA29 DB 0DH,0AH,'MUSHARIB ',0DH,0AH,' AWAIS ',0DH,0AH,' RIZWAN ',0DH,0AH,' ABDULLAH ',0DH,0AH,' HUZAIFA ',0DH,0AH,' UMAIR ',0

DATA30 DB 0DH,0AH,'https://github.com/yousefkotp/8086-Assembly-Projects/tree/main/Project%20%238%20-%20ATM%20Machine ',0DH,0AH, 'https://stackoverflow.com/',0

DATA31 DB 0DH,0AH,'ENTER YOUR PASSWORD: ',0

ACCOUNT\_BALANCE DW 1000 ; Initial account balanc

IDINPUT DW 1 DUP (?)

PASSINPUT DB 1 DUP (?)

TRIES\_LEFT DB 3 ; Number of password attempts allowed

DATA ENDS

CODE SEGMENT

START:

MOV AX,@DATA

MOV DS,AX

; User registration or login section

LEA SI,DATA1

CALL PRINT\_STRING

PRINT 0AH

PRINT 0DH

LEA SI,DATA24

CALL PRINT\_STRING

LEA SI,DATA22

CALL PRINT\_STRING

LEA SI,DATA23

CALL PRINT\_STRING

PRINT 0AH

PRINT 0DH

LEA SI,DATA11

CALL PRINT\_STRING

PRINT 0AH

PRINT 0DH

CALL SCAN\_NUM

CMP CX, 1

JE LOGIN

CMP CX, 2

JE SIGNUP

JMP START

DEFINE\_SCAN\_NUM

DEFINE\_PRINT\_STRING

DEFINE\_PRINT\_NUM

DEFINE\_PRINT\_NUM\_UNS

AGAIN:

LEA SI,DATA1

CALL PRINT\_STRING

LEA SI,DATA2

CALL PRINT\_STRING

MOV SI,-1

CALL SCAN\_NUM

MOV IDINPUT,CX

MOV AX,CX

MOV CX,0

L1:

INC CX

CMP CX,TOTAL

JE ERROR

INC SI

MOV DX,SI

CMP IDS1[SI],AX

JE PASS1

CMP IDS2[SI],AX

JE PASS2

JMP L1

SIGNUP:

; Perform user signup logic

LEA SI,DATA25

CALL PRINT\_STRING

CALL SCAN\_NUM

MOV IDINPUT, CX ; Store the entered account number

LEA SI,DATA26

CALL PRINT\_STRING

CALL SCAN\_NUM

MOV PASSINPUT, CL ; Store the entered password

LEA SI, DATA3

MENU:

; Display menu options

PRINT 0AH

PRINT 0DH

LEA SI,DATA6

CALL PRINT\_STRING

PRINT 0AH

PRINT 0DH

LEA SI,DATA7

CALL PRINT\_STRING

PRINT 0AH

PRINT 0DH

LEA SI,DATA8

CALL PRINT\_STRING

PRINT 0AH

PRINT 0DH

LEA SI,DATA9

CALL PRINT\_STRING

PRINT 0AH

PRINT 0DH

LEA SI,DATA10

CALL PRINT\_STRING

PRINT 0AH

PRINT 0DH

LEA SI,DATA19

CALL PRINT\_STRING

PRINT 0AH

PRINT 0DH

LEA SI,DATA27

CALL PRINT\_STRING

PRINT 0AH

PRINT 0DH

LEA SI,DATA28

CALL PRINT\_STRING

PRINT 0AH

PRINT 0DH

LEA SI,DATA15

CALL PRINT\_STRING

; Get user choice

LEA SI,DATA11

CALL PRINT\_STRING

CALL SCAN\_NUM

CMP CX, 1

JE ADD\_MONEY

CMP CX, 2

JE WITHDRAW\_MONEY

CMP CX, 3

JE TRANSFER\_MONEY

CMP CX, 4

JE CHECK\_BALANCE

CMP CX, 5

JE ABOUT\_US

CMP CX, 6

JE REFRENCES

CMP CX, 7

JE LEAVE\_PROGRAM

JMP MENU

TRANSFER\_MONEY:

LEA SI,DATA16

CALL PRINT\_STRING

CALL SCAN\_NUM

MOV AX, CX ; Transfer account number

; Perform the transfer logic, you may need to implement this logic based on your requirements

LEA SI,DATA12

CALL PRINT\_STRING

CALL SCAN\_NUM

CMP CX, ACCOUNT\_BALANCE

JG TRANSFER\_ERROR

SUB ACCOUNT\_BALANCE, CX ; Assuming the entered amount is added to the current account balance

LEA SI,DATA13

CALL PRINT\_STRING

MOV AX, ACCOUNT\_BALANCE

CALL PRINT\_NUM\_UNS

PRINT 0AH

PRINT 0DH

JMP MENU

ADD\_MONEY:

LEA SI,DATA12

CALL PRINT\_STRING

CALL SCAN\_NUM

ADD ACCOUNT\_BALANCE, CX

JMP SHOW\_REMAINING\_BALANCE

WITHDRAW\_MONEY:

LEA SI,DATA12

CALL PRINT\_STRING

CALL SCAN\_NUM

CMP CX, ACCOUNT\_BALANCE

JG WITHDRAW\_ERROR

SUB ACCOUNT\_BALANCE, CX

JMP SHOW\_REMAINING\_BALANCE

TRANSFER\_ERROR:

LEA SI,DATA17

CALL PRINT\_STRING

PRINT 0AH

PRINT 0DH

JMP DENIED\_MESSAGE

WITHDRAW\_ERROR:

LEA SI,DATA4

CALL PRINT\_STRING

PRINT 0AH

PRINT 0DH

JMP DENIED\_MESSAGE

CHECK\_BALANCE:

; Display remaining balance

PRINT 0AH

PRINT 0DH

LEA SI,DATA20

CALL PRINT\_STRING

MOV AX, ACCOUNT\_BALANCE

CALL PRINT\_NUM\_UNS

PRINT 0AH

PRINT 0DH

JMP MENU

SHOW\_REMAINING\_BALANCE:

; Display remaining balance

LEA SI,DATA13

CALL PRINT\_STRING

MOV AX, ACCOUNT\_BALANCE

CALL PRINT\_NUM\_UNS

PRINT 0AH

PRINT 0DH

JMP MENU

LOGIN:

JMP AGAIN

PASS1:

LEA SI,DATA3

CALL PRINT\_STRING

CALL SCAN\_NUM

MOV PASSINPUT,CL

MOV AX,DX

MOV DX,0002H

DIV DL

MOV SI,AX

MOV AL,CL

MOV AH,00H

CMP PASSWORDS1[SI],AL

JNE PASS2\_TRY

JMP ALLOWED\_MESSAGE

PASS2:

LEA SI,DATA3

CALL PRINT\_STRING

CALL SCAN\_NUM

MOV PASSINPUT,CL

MOV AX,DX

MOV DX,0002H

DIV DL

MOV SI,AX

MOV AL,CL

MOV AH,00H

CMP PASSWORDS2[SI],AL

JNE PASS2\_TRY

JMP ALLOWED\_MESSAGE

PASS2\_TRY:

; Display a message to try again

DEC TRIES\_LEFT

CMP TRIES\_LEFT, 0

JZ DENY\_ACCESS

LEA SI, DATA18

CALL PRINT\_STRING

PRINT 0AH

PRINT 0DH

MOV SI, 0

JMP AGAIN

DENY\_ACCESS:

; Display a message and exit or take appropriate action

LEA SI, DATA14

CALL PRINT\_STRING

PRINT 0AH

PRINT 0DH

JMP EXIT\_PROGRAM

ERROR:

LEA SI,DATA21

CALL PRINT\_STRING

PRINT 0AH

PRINT 0DH

MOV SI,0

JMP AGAIN

ALLOWED\_MESSAGE:

PRINT 0AH

PRINT 0DH

LEA SI,DATA5

CALL PRINT\_STRING

JMP MENU

ABOUT\_US:

PRINT 0AH

PRINT 0DH

LEA SI,DATA29

CALL PRINT\_STRING

JMP MENU

REFRENCES:

PRINT 0AH

PRINT 0DH

LEA SI,DATA30

CALL PRINT\_STRING

JMP MENU

DENIED\_MESSAGE:

PRINT 0AH

PRINT 0DH

LEA SI,DATA4

CALL PRINT\_STRING

PRINT 0AH

PRINT 0DH

JMP MENU

LEAVE\_PROGRAM:

; Display goodbye message or any other exit-related logic

; ...

EXIT\_PROGRAM:

; Add any additional cleanup or exit logic

; ...

CODE ENDS

END START

E:\LGU\2nd Semester\Z-Extras\Endground.jpg