



# **Individual Assignment**

**TECHNOLOGY PARK MALAYSIA**

**CT073-3-2-CSLLT-L-2**

**Computer System Low Level Techniques**

**APD2F2411CS(CYB)**

**HAND OUT DATE: 14th April 2025**

**HAND IN DATE: 14th July 2025**

---

## **INSTRUCTIONS TO CANDIDATES:**

Complete the assignment and submit it online through Moodle. Your assignment submission is integrated with Turnitin option for plagiarism check.

---

**NAME: THANESWARAN A/L RAJASEGARAN**

**TP NO: TP070624**

## Table of Contents

1.0 Introduction to Assembly Language .....	3
2.0 Assembly usage in the cybersecurity industry. ....	4
2.1 Reverse Engineering Write Up.....	5
First Step for solving RE.....	5
Second Step for solving RE.....	6
Third Step for solving RE .....	6
Fourth step for solving RE .....	7
Fifth step for solving RE.....	7
3.0 System Flowchart.....	8
4.0 System Screenshot .....	9
5.0 Source Code Screenshot.....	17
5.1 Data Code.....	18
5.2 Code .....	21
5.3 main menu .....	23
5.4 Check Balance .....	24
5.5 Show Savings Balance .....	25
5.6 Show Investment Balance.....	26
5.7 Withdraw Cash.....	29
5.8 Cash Deposit.....	31
5.9 Cheque Deposit .....	33
5.10 Change Pin.....	36
5.11 Transaction .....	39
5.12 Strings to Number conversation .....	42
5.13 Print Receipt .....	44
5.14 Hashing .....	45
6.0 Conclusion .....	46
7.0 References.....	47

## 1.0 Introduction to Assembly Language

Technology has had a huge, impressive revolution in the past few years, which plays a core role in various sectors of the country's economy. Hereby, the improvement of technology brings more changes to this digital world and the understanding of technology becomes more complex. In detail, the computer languages are gotten up more advanced and the assembly language in every operating system also got updated through the CPU development. What is assembly language? Assembly language work as a converter from low level language to high level language to let the machine understand the instructions of the CPU.

Assembly language has high demand in every OS, embedded systems, cybersecurity and many more. This is because, by using assembly language the developers can do high-performance coding for the machine that has low memory level and system with limited resources. In addition, cybersecurity specialists also implement assembly language into their industry to analyze the malware inside the specific machine. Therefore, the assembly language plays the main role in every industry that is beyond the technology sight. By using assembly language, operating system developers can communicate directly with the hardware efficiently.

What is Assembler? An assembler is computer software that translates assembly code into machine code and gives access to communicate with the hardware. In addition, the assembly program can change the human-readable instructions into a binary code for the Central Processing Unit (CPU) can run or execute it (What is an Assembler?, n.d.). The Assembler translates the human readable code created by programmers or developers into object code which is clarified as machine code and creates an object file there. Moreover, the linker will take one or more files and combine it into a single execution file. So that all the instructions that are generated by programmers combine in one object file and ready for the CPU execution. In the final step, the CPU will fetch the instruction from the loader or memory and decodes them to understand the instruction to perform (JaJirayu, 2024).

## 2.0 Assembly usage in the cybersecurity industry.

Nowadays, assembly language also plays a main role in the cybersecurity industry because as the report mentioned before most of the cybersecurity professionals use assembly language to do the analysis of the infected Operating System, Software, Application and many more. Furthermore, A major application of assembly language is malware analysis and reverse engineering, in which cybersecurity professionals' study malicious software to learn about its structure, behavior, and purpose without source code Assembly is vital for analyzing vulnerabilities and getting knowledge of how attackers manipulate the code.

There are several types of attacks that can help prevent and understand how the attack is conducted. For instance, Buffer overflow attack is offensive that overwrite the buffer fixed-length block of memory and to input arbitrary code to crash system for gain the unauthorized access through it. Buffer Overflow attack is one of the Common Weakness Enumeration (CWE) attacks that classified as CWE-120 in the CWE dictionary (Hanna, 2025). Hereby, to sort out these issues most of the cybersecurity professionals are using assemblers to analyze it. This is because assembler can capture the CPU instructions and can get to know the pattern that how the instruction flows. On the other hand, assemblers can get a forensic view from stack frame analysis because stack can save the local variables and return addresses. Hereby, the cybersecurity analyst can see where the overflow occurred in the buffer block and find a way to prevent it.

There is another well-known attack that can be solved by using assembly languages which Code injection and Shellcode attacks. Code injection is a attack that injects a malicious code into an application to cause a unexplained command execution. So that, the attacker can easily gain access to the system, high chances of system crash, data loss and many more consequences. For solution the IT engineers use assembler to analyze the flow of execution of the code. In addition, by using assembler the cybersecurity analyst can view the instructions step by step, so if the instructions make any malicious activity means the cyber professionals can take an action immediately to prevent the attack from starting stage. Moreover, usually shellcode stored in memory and then jumped into debuggers like Ghidra to set breakpoints. If the shellcode finds there are malicious code which means it will automatically mark as RWX. So, if the cyber professionals see this kind of symbol means they can easily identify there are some malicious codes inside the applications.

To conclude, assembly language is the highly valuable technological device in cybersecurity with the profound understanding of the system and dangerous code functioning. It enables the experts to study flaws like injection of codes and buffer overflows by monitoring CPU instructions and flows of executions. With the help of programs such as Ghidra and debuggers, an analyst will be able to notice an unusual memory activity, track shellcode, and find out what stack frame was used. It is such a level of visibility that is crucial to reverse engineering the malware and stop such attacks early. Cyber threats have been getting even more developed, and assembly is among the most relevant skills that professionals are supposed to have to analyze the advanced interventions to security breaches.

## 2.1 Reverse Engineering Write Up

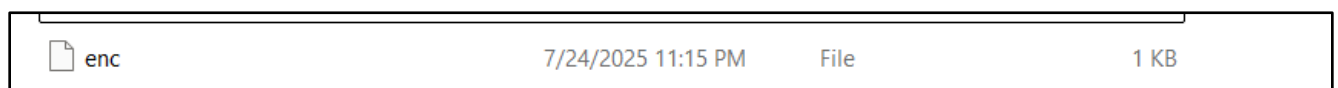


Figure 2.1.1

This is the challenge that I chose for Reverse Engineering to solve from picoCTF website.

### First Step for solving RE

We can download the “enc” file by clicking the link inside the description of the picoCTF challenge.



## Second Step for solving RE

Copy and paste the same file we downloaded inside our file explorer into our Virtual Machine. For the information, I used Kali Linux to find the flag for this reverse engineering challenge. So, I decided to save the file into my Desktop to find the file path easily. By inputting cd command, we can change the directory into Desktop from there we can access the enc file.

```
(kali㉿kali)-[~]  
$ cd Desktop  
  
(kali㉿kali)-[~/Desktop]  
$ ls -l  
total 63620  
-rwxrw-rw- 1 kali kali      533 Jul  7 03:48 BankNFT.sol  
-rwxrw-rw- 1 kali kali      998 Jul  7 03:48 Bank.sol  
-rw----- 1 kali kali 1052616 Apr  4 2022 BAT.jpg  
-rwxrw-rw- 1 kali kali 52428800 May 15 14:48 disko-1.dd  
-rwxrw-rw- 1 kali kali      57 Jul 24 11:15 enc  
-rwxrw-rw- 1 kali kali      703 Jul 19 03:37 encrypt.py  
-rwxr-xr-x 1 kali kali      269 Jun  8 2018 idle.desktop  
-rw-r--r-- 1 kali kali 11621877 Jun  9 05:17 output.txt  
drwxr-xr-x 17 kali kali    4096 May 30 04:38 Python-3.10.4  
-rwxrw-rw- 1 kali kali    7192 May 30 04:04 RE1
```

Figure 2.1.2

## Third Step for solving RE

By inputting the cat command, we can open the file called as enc. So, from there, we can see some Chinese words there. These are the flag for CTF now we must debug it to find the proper flag.

```
(kali㉿kali)-[~/Desktop]  
$ cat enc  
灑擲畧規 𐤎形梲獍楮獐? 攪潦彌彥 ㄥ一て尅
```

Figure 2.1.3

#### Fourth step for solving RE

We can use this python programming condition to find the flag. This Python script opens a file called enc and reads one character at a time through the enc file which is converted to Unicode representation of hexadecimal ord() hex() as well removing the prefix 0x in the hexadecimal column a[2:]) It decodes this hex string again to an ASCII character by: bytes.fromhex(...).decode() and prints the result. This code will decode a basic character-to-hex conversion yielding the original ASCII character text converted to a base hexadecimal format.

```
#!/usr/bin/python3

# Read form the file
flag = open('enc').read()

for i in range(len(flag)):
    # Return Unicode code from a character then convert it to Hex
    a = hex(ord(flag[i]))

    # Convert Hex to ASCII and print it
    b = bytes.fromhex(a[2:]).decode()
    print(b, end='')
```

Figure 2.1.4

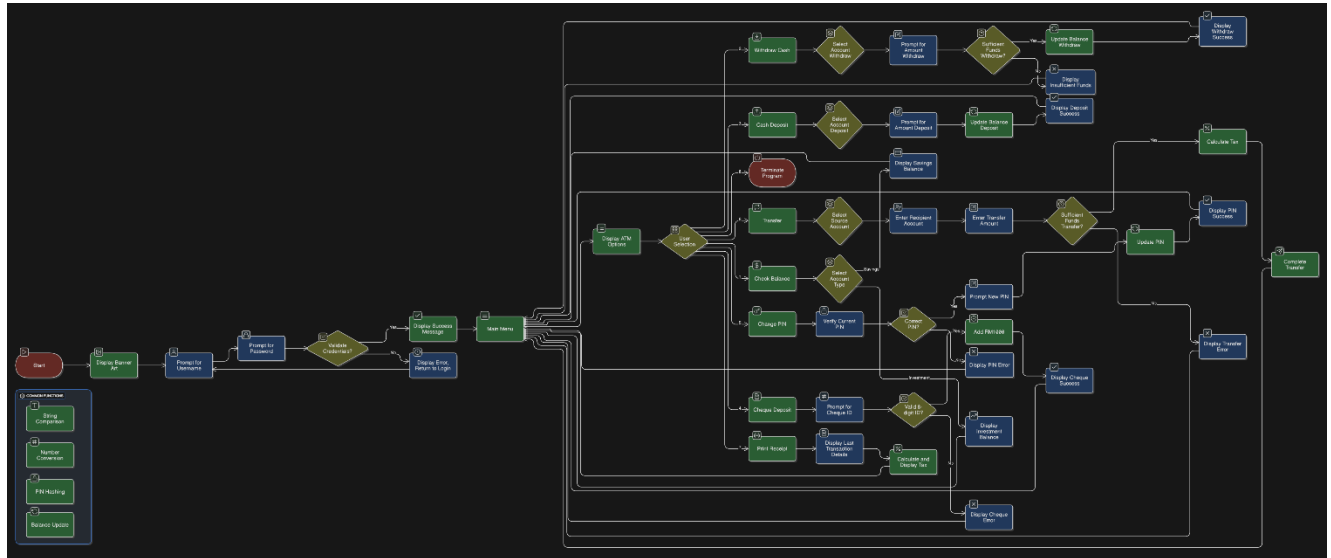
#### Fifth step for solving RE

By implemeting python3 command along with the pyhton code we manage to find the picoCTF flag.

```
(kali㉿kali)-[~/Desktop]
$ python3 re.py
picoCTF{16_bits_inst34d_of_8_e141a0f7}
```

Figure 2.1.5

### 3.0 System Flowchart



*Figure 3.1*

The flowchart depicted there shows a workflow of a whole ATM system that starts with the process of logging in and ends with the end of the program. The first screen that users see is a banner screen and they are encouraged to enter their username and masked password. There is then a decision to authenticate these credentials and in case of an error, the system returns to the login screen with an error message, and in case of success, it gives a success message and shifts to the main menu. This page, however, shows a total of eight options as shown to the users; Check Balance, Withdraw Cash, Cash Deposit, Cheque Deposit, Change PIN, Transfer, Print Receipt, and Exit. Both alternatives are either of them include a kind of sub-process and validations that represent the secure and efficient bank operation.

Every function has a logical flow. In other words, the Check Balance setting makes users choose a type of account (Savings, or Investment) and only after that the balance will appear. Withdrawal and transfer have a verification process that is conducted to understand that there is enough money on the account and that the balance of an account is updated, deposit also gets deposits adjusted in their accounts. One should use the existing PIN correctly before setting a new one. Cheque deposits check a



cheque ID of 8-digits and credits RM1000 when it is successful. Receipt printing also reads and prints out the latest transaction details in addition to tax information. The Exit button is an amicable exit button to close the session. Then there is the use of different shapes and colors diamond to indicate the decision, rectangles to indicate the process, and ovals to indicate start/end that makes it very clear and logical. Also, utility functions such as comparison of strings, hashing PIN and update of balance are handled as distinct, which encourages modularity and re-utilization in the system.

## 4.0 System Screenshot



Figure 4.1

This is the confirmation page of the login to THANES ATM SYSTEM. Once a user logs in successfully providing his username and password, a message entitled Login successful appears. The screen is then changed into a main menu with an ASCII art banner of THANES ATM SYSTEM. Some of the banking services available on the system include balance enquiry, withdrawal or deposit

of cash and cheque deposits, change of PIN, money transfer and printing of receipts. This interface is command-line ATM simulation, which is to be used simply to work with an ATM. It is easy to use and theological and contains good guidelines on how to explore the features of the system. The last option gives the user the opportunity to exit without problems.



```
1. Check Balance
2. Withdraw Cash
3. Cash Deposit
4. Cheque Deposit
5. Change PIN
6. Transfer to Another Account
7. Print Receipt
8. Exit
1
Check from:
1. Savings Account
2. Investment Account
Choice: 1
Savings Balance: 1000
```

*Figure 4.2*

This is the Check Balance entity of the THANES ATM SYSTEM, which is of a savings account. Once one has selected option 1 in the main menu the system also requests user to select which type of account they would wish to query. There are two ways in which the user can do: 1. Savings Account and 2. Investment Account. The user chooses 1 that represents the saving account. After this, the system will show the balance on the savings account as of now, that is, 1000. This functionality lets the user see easily on how much money he/ she can withdraw instantly and is useful in planning the user finances within the ATM program.

```
1. Check Balance
2. Withdraw Cash
3. Cash Deposit
4. Cheque Deposit
5. Change PIN
6. Transfer to Another Account
7. Print Receipt
8. Exit
1
Check from:
1. Savings Account
2. Investment Account
Choice: 2
Investment Balance: 1000
```

*Figure 4.3*

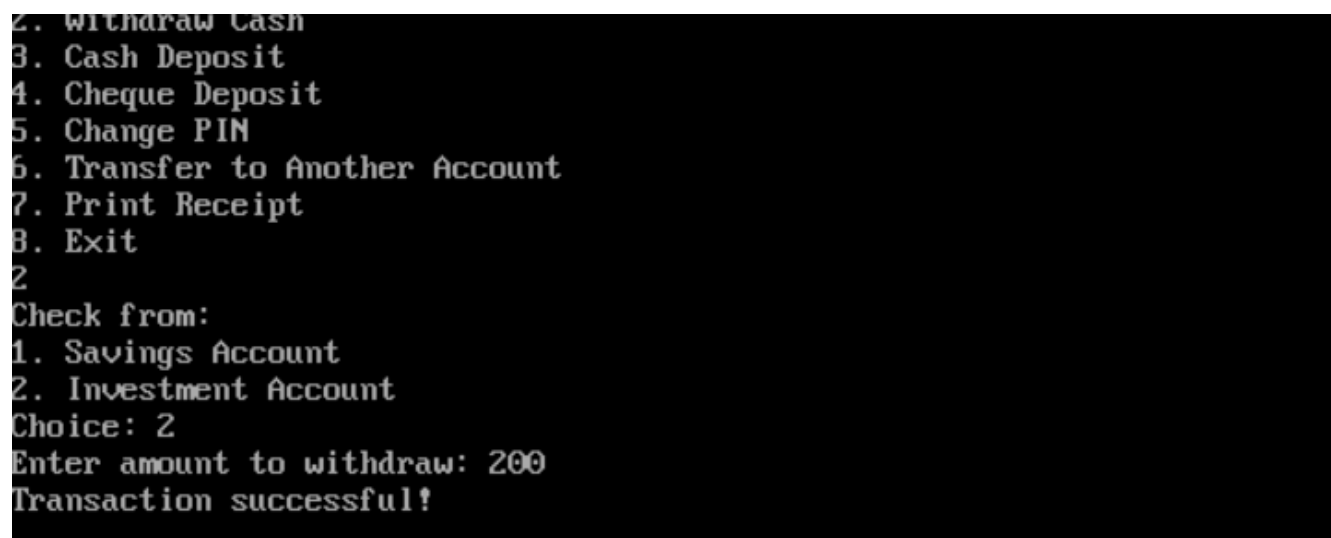
This screen displays the Check Balance application of the THANES ATM SYSTEM in this case the investment account. Once the user has settled on option 1 in the main menu, it will display another option of choosing the account type. To choose an investment account they press 2. The system will show the present balance in investment, which is 1000. This is being done to enable the user to continue tracking the amount in their investment account directly via the ATM screen to ensure that they keep tabs on their accounts and the like so that they can provide savings and investment account management. The system allows easy text-based communication so one can easily check the balance.

```
2
Check from:
1. Savings Account
2. Investment Account
Choice: 1
Enter amount to withdraw: 400
Transaction successful!
```

*Figure 4.4*

The presented screenshot is of the Withdraw Cash option in the THANES ATM SYSTEM in the saving account. The user enters 1 (Check Balance) then option 1 (Savings Account). The Savings Balance shows 600, which means that a withdrawal action took place in the past, and previously balanced amount (1000 or more) got down to 600. Although the number of withdrawals is not displayed here, the balance has changed to show that a transaction was successful. This feature enables

the user to monitor how much s/he has left in the account, once a withdrawal has been issued and gives clarity and making it easier to manage the account.



```
2. Withdraw Cash
3. Cash Deposit
4. Cheque Deposit
5. Change PIN
6. Transfer to Another Account
7. Print Receipt
8. Exit
2
Check from:
1. Savings Account
2. Investment Account
Choice: 2
Enter amount to withdraw: 200
Transaction successful!
```

*Figure 4.5*

This screenshot shows the withdrawal of cash functionality of the THANES ATM SYSTEM, of the investment account. The user presses menu no. 2 (Withdraw Cash) and option no. 2 again (Investment Account) to enter the investment account. They are assisted in putting the amount they wish to withdraw and write 200. The request is processed by the system, and it is confirmed as: "Transaction successful!". This aspect enables the user to access their investment account to retrieve funds without trouble, the prompts and confirmation of operations are clear, which makes funds withdrawal process secure and convenient.

```
2. Withdraw Cash
3. Cash Deposit
4. Cheque Deposit
5. Change PIN
6. Transfer to Another Account
7. Print Receipt
8. Exit
3
Check from:
1. Savings Account
2. Investment Account
Choice: 1
Enter amount to deposit: 100
Transaction successful!
```

*Figure 4.6*

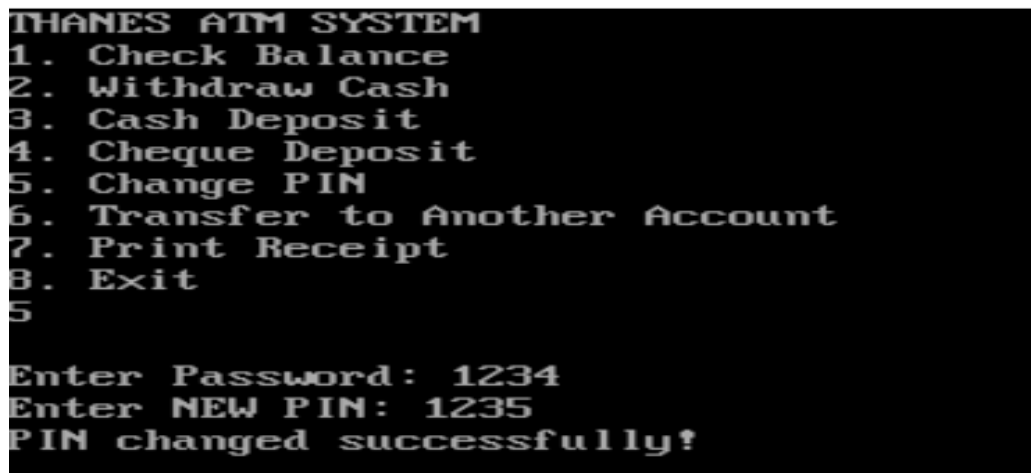
The following is a screenshot that shows how the Cash Deposit function of the THANES ATM SYSTEM looks like. When the user wants to make a deposit, she uses the main menu, by using the number 3. The system then instructs the user to select the destination account where the deposit is meant, 1-Savings Account or 2-Investment Account. The user keys 1 which means that he/she wants to deposit funds to his/her Savings Account. After that, they should put the amount of a deposit and put 100. Transaction is made by the system and gets back with the message, confirming that the cash amount has been deposited successfully: Transaction successful! The benefit of this feature is that users can easily deposit money in their account.

```
2. Withdraw Cash
3. Cash Deposit
4. Cheque Deposit
5. Change PIN
6. Transfer to Another Account
7. Print Receipt
8. Exit
3
Check from:
1. Savings Account
2. Investment Account
Choice: 2
Enter amount to deposit: 200
Transaction successful!
```

*Figure 4.7*

In this screenshot, you will see the Cash Deposit menu of the THANES ATM SYSTEM, the investment account. To deposit the user will choose 3 on the main menu. The system will then ask the

type of account to be requested and the user will type 2 to select the Investment Account. The user proceeds to deposit money and since the deposit is 200, he or she enters this in the deposit field. The deposit is transferred to the system, and the process is confirmed with a message that states the transaction to be successful. This aspect will enable customers to quickly raise their investment account balance by depositing of funds via the ATM interface.



*Figure 4.8*

In this screenshot, I have shown the Change PIN of the THANES ATM SYSTEM. To begin changing the PIN, a user follows the main menu and presses the button 5. The user is then forced to feed the currently used password or the existing PIN in this instance which is 1234. Once the verification successful the system prompts the user to key in a new PIN and the user enters the PIN 1235. After the new PIN has been selected, the system indicates that the PIN was changed correctly with the message: PIN changed successfully!. The feature offers a safe and convenient method where users can change the PIN and make their accounts more secure.

```
2. Withdraw Cash
3. Cash Deposit
4. Cheque Deposit
5. Change PIN
6. Transfer to Another Account
7. Print Receipt
8. Exit
4
Check from:
1. Savings Account
2. Investment Account
Choice: 1
Enter Cheque ID: 12345678
Cheque RM1000 deposited successfully!
```

*Figure 4.9*

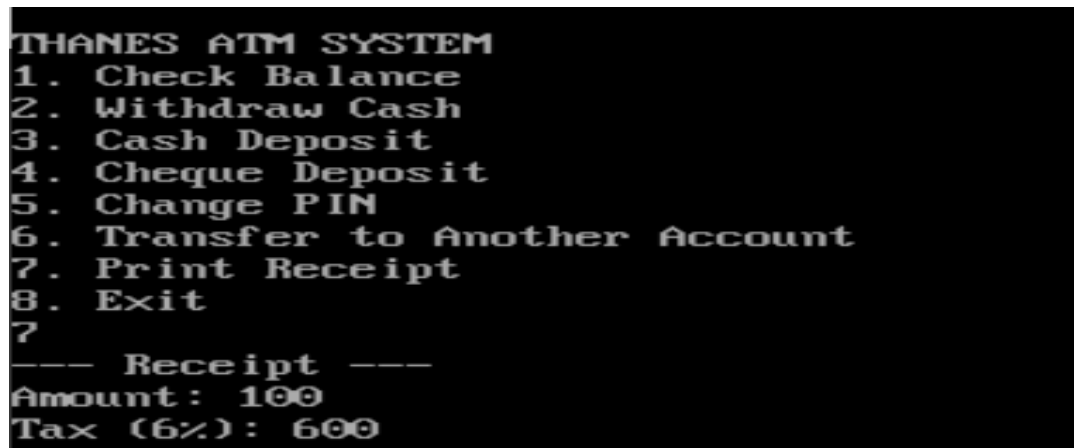
The above screen shot depicts the option of Cheque Deposit in the THANES ATM SYSTEM. The user will choose number 4 under the main menu where he gets to start the deposit procedure. They are then requested to select an account type where they select 1 as Savings account. Then the system asks to enter Cheque ID and the user types 12345678. When validated, the system will establish that a cheque of RM1000 has been successfully deposited with the message: Cheque RM1000 deposited successfully!. This aspect allows bank customers to insert their cheques safely and directly into their preferred account, which increases the versatility and convenience of the ATM system.

```
3. Cash Deposit
4. Cheque Deposit
5. Change PIN
6. Transfer to Another Account
7. Print Receipt
8. Exit
6
Transfer from:
1. Savings Account
2. Investment Account
Choice: 1
Enter recipient 12-digit account number: 123456789012
Enter amount to transfer: 100
Transfer completed!
```

*Figure 4.10*

In this screenshot, I will carry out a demonstration of the Transfer to Another Account feature of the THANES ATM SYSTEM. To transfer the funds, the user must choose option 6 of the main menu. The system then advises the user to select the source account, and the user selects 1 to represent Savings

Account. The following prompt asks an account number of the recipient, which the user types as 123456789012. Subsequently, the user is prompted to enter the sum that shall be transferred and this shall be 100. This makes the request to be processed through the system and sends confirmation of operation in the form of: "Transfer completed!". This aspect enables direct and safe transfer of funds within accounts.



*Figure 4.11*

The following is attachment of a screenshot of the Print Receipt on THANES ATM SYSTEM. Once this option 7 is selected in the main menu, the system prints out a receipt. On the receipt, the total of a transaction, 100, is indicated, and the calculation of a 6-percent tax is made but incorrectly reported as 600- probably the error in tax calculation logic. Everything goes smoothly; 6 out of 100 is ideal as 6 and not 600. Even with this flaw, the feature indicates the quality of the ATM system which shows a simple receipt summary to the recent transactions that are performed, including transparency and a history to the user. This feature assists the user to monitor their practice and fee.



## 5.0 Source Code Screenshot

[illegible]

Figure 5.1

```

savings_balance dw 1000    ; Initial savings account balance
investment_balance dw 1000 ; Initial investment account balance

msg_balance db 13,10,'Current Balance: $'
msg_withdraw db 13,10,'Enter amount to withdraw: $'
msg_success db 13,10,'Transaction successful!',13,10,'$'
msg_insufficient db 13,10,'Insufficient funds!',13,10,'$'
input_buffer db 9,0,9 dup(0)

msg_deposit db 13,10,'Enter amount to deposit: $'

msg_cheque_id db 13,10,'Enter Cheque ID: $'
msg_cheque_added db 13,10,'Cheque RM1000 deposited successfully!',13,10,'$'
invalid_cheque_msg db 'Invalid Cheque ID. Must be 8 digits.',13,10,'$'

msg_transfer_from db 13,10,'Transfer from:',13,10,'1. Savings Account',13,10,'2. Investment Account',13,10,'Choice: $'
msg_transfer_to db 13,10,'Enter recipient 12-digit account number: $'
msg_transfer_amt db 13,10,'Enter amount to transfer: $'
msg_transfer_done db 13,10,'Transfer completed!',13,10,'$'
msg_pin_updated db 13,10,'PIN changed successfully!',13,10,'$'

stored_pin_hash dw 202    ; Optional: if using hashed PINs
account_number_buffer db 14,0,14 dup(0)
msg_change_pin db 13,10,'Enter NEW PIN: $'
msg_transfer_log db 'Transfer', 0

msg_wrong_pin db 13,10, 'Incorrect current PIN. Try again.',13,10,'$'

check_balance_menu db 13,10,'Check from:',13,10,'1. Savings Account',13,10,'2. Investment Account',13,10,'Choice: $'
msg_savings_balance db 13,10,'Savings Balance: $'
msg_investment_balance db 13,10,'Investment Balance: $'

receipt_msg db 13,10, '--- Receipt ---',13,10,'$'
amount_msg db 'Amount: $'
tax_msg db 13,10, 'Tax (6%): $'
total_msg db 13,10, 'Total: $'
last_transaction_amount dw 0

```

Figure 5.2

## 5.1 Data Code

This assembly program contains data section block, where all variables, strings and constants applied within the THANES ATM SYSTEM are declared. It starts with ASCII art (banner\_art) formed in visualizing the name of ATM system. Then it determines prompts such as username\_prompt and password\_prompt, success and failure messages (login\_success, login\_fail). The static credentials are kept in stores\_user and stores\_pass. The user input is allocated in input buffers like user\_buffer and pass\_buffer. It then initializes balance, savingsbalance and investmentbalance to 1000. The main menu items will be presented using a series of menu lines (line1 to line9). Some of the message constants used are msg\_withdraw, msg\_deposit and msg\_transfer\_amt that are displayed when making transactions. It has also certain messages concerning handling of cheques, pin change, and messages due to error such as msg\_insufficient and msg\_wrong\_pin. Other utility variables such as, last\_transaction\_amount, account\_number\_buffer, and hashed PIN (stored\_pin\_hash) are defined here as well. Finally, message blocks such as receipt\_msg, amount\_msg and tax\_msg assist with the ability to print the receipt. In general, the given section gathers all the static data, and one can easily manipulate user interaction texts and vital program variables in it.

```

.code
start:
    mov ax, @data
    mov ds, ax
    jmp login_screen

login_screen:
    mov dx, offset newline
    mov ah, 09h
    int 21h

    mov dx, offset username_prompt
    mov ah, 09h
    int 21h
    lea dx, user_buffer
    mov ah, 0Ah
    int 21h

    mov dx, offset password_prompt
    mov ah, 09h
    int 21h

    ; Masked password input
    xor cx, cx
    lea di, pass_buffer + 2
.mask_loop:
    mov ah, 08h    ; Read char without echo
    int 21h
    cmp al, 13     ; Enter key?
    je .done_input
    mov [di], al
    inc di
    inc cx
    mov ah, 02h
    mov dl, '*'
    int 21h

```

*Figure 5.3*

```

.mask_loop:
    mov ah, 08h    ; Read char without echo
    int 21h
    cmp al, 13     ; Enter key?
    je .done_input
    mov [di], al
    inc di
    inc cx
    mov ah, 02h
    mov dl, '*'
    int 21h
    cmp cx, 10
    jb .mask_loop
.done_input:
    mov byte ptr pass_buffer[1], cl
    jmp continue_login_validation

continue_login_validation:
    lea si, stored_user
    lea di, user_buffer + 2
    call compare_strings
    cmp ax, 0
    jne login_jump_failed

    lea si, stored_pass
    lea di, pass_buffer + 2
    call compare_strings
    cmp ax, 0
    jne login_jump_failed

    mov dx, offset login_success
    mov ah, 09h
    int 21h

    mov dx, offset banner_art
    mov ah, 09h
    int 21h

```

*Figure 5.4*

```

; spacing after banner
mov dx, offset newline
mov ah, 09h
int 21h
int 21h
jmp main_menu

jmp main_menu

login_jump_failed:
jmp login_failed

```

*Figure 5.5*

## 5.2 Code

This is the section of the `.code` section that initializes and processes the login process of the THANES ATM SYSTEM. The label `start:` initializes data segment register (`ds`) with `@data`, necessary to access all the variables in the `.data` segment and pushes itself to `login_screen` instantly. The input of the username is prompted in `login_screen` section where the system retrieves the data entered in `user_buffer` via `0Ah` of DOS interrupt which performs a buffered input. It then requires the submission of a password which is processed in a different way. Masking the password is done instead of typing in echo characters: the program reads one character at a time using interrupt `08h`, then checks the Enter key (ASCII 13) in case the user wishes to exit the typing process, stores the character in `pass_buffer`, adds two counts, and displays a for each character (using interrupt `02h`). Once I have entered 10 characters, or hit the Enter key, it goes to `continue_login_validation`. In this case, it will load the entered username in `si` and the user input in `di`, and this will be used to invoke the `compare_strings` subroutine to ensure equality. It does the same procedure as the password. In the case that either check fails, a jump is done to `login_jump_failed` which goes to `login_failed` (presumably sending out a failure message). If both are true, then it prints the "Login successful!". The message is printed, followed by the ASCII `banner_art`, with display of the strings being based on interrupt `09h`. To keep the visual space following the banner, it reprints two lines via the identical interruption. Once the user logs in successfully, the user is taken to the `main_menu` where the entire ATM capability starts. This part provides user security through credential verification and polishes it

by having masked password and the banner display as would appear in a specific DOS based command line would have a professional login screen.

```
main_menu:
    ; Print menu header
    mov dx, offset line1
    mov ah, 09h
    int 21h

    ; Print options
    mov dx, offset line2
    int 21h
    mov dx, offset line3
    int 21h
    mov dx, offset line4
    int 21h
    mov dx, offset line5
    int 21h
    mov dx, offset line6
    int 21h
    mov dx, offset line7
    int 21h
    mov dx, offset line8
    int 21h
    mov dx, offset line9
    mov ah, 09h
    int 21h

    ; Get user input
    mov ah, 01h
    int 21h
    sub al, '0'

    cmp al, 1
    jne check2
    jmp check_balance
```

*Figure 5.6*

```

check2:
    cmp al, 2
    jne check3
    jmp withdraw_cash

check3:
    cmp al, 3
    jne check4
    jmp cash_deposit

check4:
    cmp al, 4
    jne check5
    jmp cheque_deposit

check5:
    cmp al, 5
    jne check6
    jmp change_pin

check6:
    cmp al, 6
    jne check7
    jmp transfer_account

check7:
    cmp al, 7
    jne check8
    jmp print_receipt

check8:
    cmp al, 8
    jne main_menu
    jmp exit_program

```

*Figure 5.7*

### 5.3 main menu

This main menu code displays all ATM options using predefined strings (line1 to line9) through DOS interrupt 21h with function 09h. After presenting the menu, it captures a single character from the user (int 21h, ah=01h), converts it from ASCII to a number by subtracting '0', and uses conditional jumps (cmp and jne) to navigate to the corresponding routine based on user choice. Each choice is linked to a function like check\_balance, withdraw\_cash, cash\_deposit, and so on. If option 8 is selected, the program jumps to exit\_program; otherwise, it loops back to main\_menu.

```

check_balance:
    mov dx, offset check_balance_menu
    mov ah, 09h
    int 21h

    mov ah, 01h
    int 21h
    sub al, '0'

    cmp al, 1
    je show_savings_balance
    cmp al, 2
    je show_investment_balance
    jmp main_menu

```

*Figure 5.8*

## 5.4 Check Balance

This is a block of codes that deal with the Check Balance utility of the DAEK exhibited on the THANES ATM SYSTEM. When the user chooses this option, the program will start by presenting the balance-check menu via interrupt 21h with 09h function, which will print the characters contained by check\_balance\_menu. It subsequently waits for the entry of a single data character using function 01h and after its entry, the code subtracts the ASCII 00h to bring the entry into a numeric value before checking the entry. When 1 is keyed in, it proceeds to show\_savings\_balance and when 2, it proceeds to show\_investment\_balance. Otherwise, when the input does not contain either of the two, the program skips to main\_menu. This enables the users to be able to see their current balance whether of their savings or investment account. The conditional jumps (je) cause the branching so that invalid options will not be proceeded with and there is not a disrupted flow of the user. The arrangement supports transparency, effectiveness as well as simplicity in the operation of ATM in choosing balances of accounts to be shown.



```

show_savings_balance:
    mov dx, offset msg_savings_balance
    mov ah, 09h
    int 21h

    mov ax, savings_balance
    call print_number

    mov dx, offset newline
    mov ah, 09h
    int 21h

    jmp main_menu

```

*Figure 5.9*

## 5.5 Show Savings Balance

The following block of code gives this routine `show_savings_balance` that shows the balance of a saving account of the user on the screen. It loads the message `msg_savings_balance` (that has the output message Savings Balance: in it) to the memory address and then executes the DOS interrupt function 09h through DOS interrupt 21h to show that message. Then it transfers the value to the `savings_balance` variable into the register `ax` and then calls the user-defined `print_number` procedure that transforms the numeric value into ASCII and prints it. Once it displays the balance, the program prints another line to format by displaying the newline string. It then returns to the `main_menu` label so that the user can carry out another operation at the ATM. Such routine makes the balance look cluttered and neat and it provides a smooth route back to the menu so that the system is interactive and easy to use.

```

show_investment_balance:
    mov dx, offset msg_investment_balance
    mov ah, 09h
    int 21h

    mov ax, investment_balance
    call print_number

    mov dx, offset newline
    mov ah, 09h
    int 21h

    jmp main_menu

```

*Figure 5.10*

## 5.6 Show Investment Balance

This piece of code is what creates the `show_investment_balance` a routine, this presents the balance of the current investment account of the bank to the user. It loads the address of the string `msg_investment_balance` (probably with the label of `Investment Balance:`) on the `dx` register and calls DOS interrupt `21h` with its function `09h` to display that label. Subsequently, the true balance lying in the variable `investment_balance` then is transferred to the register `ax`. `Print_number` subroutine is then called, and this prints the numeric balance in ASCII digits. Print a newline after the balance with another DOS interrupt this time with the newline string. Lastly, the routine transfers back the control to the `main_menu` by using the jump to the label. This code makes the information on balance well presentable, well-spaced, in addition the system is responsive and friendly as it automatically goes back to the larger menu once the balance is displayed.

```

withdraw_cash:
    ; Ask which account to withdraw from
    mov dx, offset check_balance_menu
    mov ah, 09h
    int 21h

    mov ah, 01h
    int 21h
    sub al, '0'
    cmp al, 1
    je withdraw_from_savings
    cmp al, 2
    je withdraw_from_investment
    jmp main_menu

withdraw_from_savings:
    mov si, offset savings_balance
    jmp do_withdraw

withdraw_from_investment:|
    mov si, offset investment_balance
    jmp do_withdraw

```

*Figure 5.11*

```

do_withdraw:
    mov dx, offset msg_withdraw
    mov ah, 09h
    int 21h

    lea dx, input_buffer
    mov ah, 0Ah
    int 21h

    ; Save SI before using string_to_number
    push si

    lea si, input_buffer + 2
    call string_to_number
    mov bx, ax          ; withdrawal amount
    mov [last_transaction_amount], bx

    pop si              ; restore SI (which holds address of account)

    mov ax, [si]        ; load selected balance
    cmp ax, bx
    jb withdraw_insufficient

    sub ax, bx
    mov [si], ax        ; store new balance

    mov al, 2           ; type = 2 (withdraw)
    mov dx, 0           ; no tax for withdraw

    mov dx, offset msg_success
    mov ah, 09h
    int 21h
    jmp main_menu

```

Figure 5.12

```

withdraw_insufficient:
    mov dx, offset msg_insufficient
    mov ah, 09h
    int 21h
    jmp main_menu

```

Figure 5.13

## 5.7 Withdraw Cash

This code portion is used in the withdrawal procedure in the THANES ATM SYSTEM. The `withdraw_cash` label will display the first message that would ask the user of what type of account he/she would like to withdraw the money-savings or investment account by providing the `check_balance_menu` and feeding the keypressed. Depending on input (1 or 2), it jumps to `withdraw_from_savings` or `withdraw_from_investment` and the `si` register is set to point at the balance to be taken (`savings_balance` or `investment_balance`). Both the paths end in the common subroutine `do_withdraw` which starts with receiving the withdraw amount the user wishes to withdraw, and placing it in the variable `input_buffer` after entering the amount the user wishes to withdraw in the command window. It first stores the `si` register (that contains the account pointer), then calls the helper routine `string_to_number`, which takes the input source as a string, converts, and stores it in a numeric form in `bx`. This value is stored in `last_transaction` amount as well. They replenish the initial `si` and compare the balance at this moment with the amount of withdrawal that is requested. When the amount is more than the available balance, the program leaves this command to a program called `withdraw_insufficient` in which an "Insufficient funds" message is printed and the control returns to the main menu. Provided there is sufficient amount of funds, the amount of withdrawal is deducted in the account and a new balance is saved. Next, metadata such as transaction type (`al = 2`), tax (`dx = 0`) are stipulated, but the withdrawals do not attract tax. Lastly, there is a success message that is posted in `msg_success` and once again we are back to `main_menu`. This properly-designed procedure effectively facilitates safe and free withdrawals where the balance will be properly updated and the user can get feedback without committing mistake such as having overdraft.

```

cash_deposit:
    mov dx, offset check_balance_menu
    mov ah, 09h
    int 21h

    mov ah, 01h
    int 21h
    sub al, '0'
    cmp al, 1
    je deposit_to_savings
    cmp al, 2
    je deposit_to_investment
    jmp main_menu

deposit_to_savings:
    mov si, offset savings_balance
    jmp do_deposit

deposit_to_investment:
    mov si, offset investment_balance
    jmp do_deposit

```

*Figure 5.14*

```

do_deposit:
    mov dx, offset msg_deposit
    mov ah, 09h
    int 21h

    lea dx, input_buffer
    mov ah, 0Ah
    int 21h

    push si                ; save SI before call
    lea si, input_buffer + 2
    call string_to_number
    mov bx, ax ; deposit amount
    mov [last_transaction_amount], bx
    pop si ; restore SI

    mov ax, [si]
    add ax, bx
    mov [si], ax           ; update selected account
    mov al, 1              ; type = 1 (deposit)
    mov dx, 0              ; no tax for deposit

    mov dx, offset msg_success
    mov ah, 09h
    int 21h
    jmp main_menu

```

*Figure 5.15*

## 5.8 Cash Deposit

This code comes in the cash deposit part of the THANES ATM SYSTEM. The cash\_deposit section starts by showing the check\_balance\_menu which requests the user to select the account to deposit to; either 1 for the savings account or 2 for investment. Depending on the selection the program then branches to one of the following deposits: deposit\_to\_savings or deposit\_to\_investment, which sets the si register to the related balance variable (savings\_balance or investment\_balance). Control is then passed on to the do\_deposit procedure where the process of the deposit takes place. At the first step, the user is asked to input an amount by prompting the user to enter the amount by showing msg\_deposit and reading the input value into input-buffer. It then stores the value of the si register and calls the string to a number function which transforms the string type into a number placed in bx. This amount is stored in the last transaction amount. Once si is restored, the code can retrieve the current balance into account and add in the amount deposited (bx) and store the new amount back in memory.

It puts the transaction type as 1 (deposit) and confirms that there is no tax. A successful message (msg\_success) is output, and the program goes back to the main menu. This makes the process of deposits safe and accurately stored.

```
cheque_deposit:
    ; Ask which account to deposit into
    mov dx, offset check_balance_menu
    mov ah, 09h
    int 21h

    mov ah, 01h
    int 21h
    sub al, '0'
    cmp al, 1
    je cheque_to_savings
    cmp al, 2
    je cheque_to_investment
    jmp main_menu

cheque_to_savings:
    mov si, offset savings_balance
    jmp cheque_enter_id

cheque_to_investment:
    mov si, offset investment_balance
    jmp cheque_enter_id
```

*Figure 5.16*



```

cheque_enter_id:
    mov dx, offset msg_cheque_id
    mov ah, 09h
    int 21h

    lea dx, input_buffer
    mov ah, 0Ah
    int 21h

    ; Validate cheque ID length = 8
    mov al, input_buffer[1]
    cmp al, 8
    jne invalid_cheque_id

    ; Deposit RM1000 into selected account (SI points to balance)
    mov ax, [si]
    add ax, 1000
    mov [si], ax

    mov dx, offset msg_cheque_added
    mov ah, 09h
    int 21h
    jmp main_menu

invalid_cheque_id:
    mov dx, offset newline
    mov ah, 09h
    int 21h
    mov dx, offset invalid_cheque_msg
    mov ah, 09h
    int 21h
    jmp main_menu

```

Figure 5.17

## 5.9 Cheque Deposit

This code segment implements the cheque deposit functionality in an ATM system, supporting both savings and investment accounts. It starts with two labeled sections: `cheque_to_savings` and `cheque_to_investment`, which load the address of the relevant balance (`savings_balance` or `investment_balance`) into the SI register, then jump to the `cheque_enter_id` routine. In `cheque_enter_id`, the program prompts the user to enter a cheque ID using the message pointed to by `msg_cheque_id`. It then uses DOS interrupt 21h functions to read and store the input into `input_buffer`. The code checks if the first byte of the buffer (representing the length of the entered string) equals 8, ensuring the cheque ID has exactly eight characters. If the condition is not met, it jumps to the `invalid_cheque_id` label, which prints a newline and an error message before returning to the main menu. If the cheque ID is valid, the code adds RM1000 to the account balance pointed to by SI. This is done by loading the existing balance into AX, adding 1000 to it, and writing the result back to

memory. A success message is displayed from msg\_cheque\_added using DOS interrupt 21h, function 09h. Finally, control returns to the main\_menu label. The invalid cheque handler ensures the program responds gracefully to incorrect input by informing the user and maintaining program stability. Overall, this code provides a secure and clear process for depositing cheques, ensuring only valid IDs proceed and that the appropriate account is updated with a fixed deposit amount. The use of registers like SI to point to memory dynamically allows for account-agnostic handling, making the logic reusable for both savings and investment operations.

```
change_pin:
    mov dx, offset newline
    mov ah, 09h
    int 21h

    ; Ask for current PIN
    mov dx, offset password_prompt
    mov ah, 09h
    int 21h
    lea dx, pass_buffer
    mov ah, 0Ah
    int 21h

    lea si, pass_buffer + 2
    mov bl, [pass_buffer + 1]
    add si, bx
    mov byte ptr [si], 0
    lea si, pass_buffer + 2
    call hash_pin_input
    cmp ax, stored_pin_hash
    jne wrong_pin_change

    ; Clear buffer
    mov cx, 10
    lea di, pass_buffer + 2
```

*Figure 5.18*

```

clear_loop1:
    mov byte ptr [di], 0
    inc di
    loop clear_loop1

    ; Ask for new PIN
    mov dx, offset msg_change_pin
    mov ah, 09h
    int 21h
    lea dx, pass_buffer
    mov ah, 0Ah
    int 21h

    lea si, pass_buffer + 2
    mov bl, [pass_buffer + 1]
    add si, bx
    mov byte ptr [si], 0
    lea si, pass_buffer + 2
    call hash_pin_input
    mov stored_pin_hash, ax

    mov dx, offset msg_pin_updated
    mov ah, 09h
    int 21h
    jmp main_menu

```

Figure 5.19

```

    jmp main_menu

wrong_pin_change:
    mov dx, offset msg_wrong_pin
    mov ah, 09h
    int 21h
    jmp main_menu

```

Figure 5.20

## 5.10 Change Pin

This segment of assembly codes acts to effect a change in PIN in an ATM system. It starts by presenting to the user to enter their current PIN with usual DOS interrupt 21h facilities to output and receive text. The input is saved in the ``pass_buffer``. The code is then able to hash this input by establishing pointers and determining the length of the PIN entered. It adds a null byte (``0``) at the end of the string and runs a hashing function on it, ``hash_pin_input``, and then compares its result with the ``stored_pin_hash``. When comparison is not true, it goes to the jump ``wrong_pin_change``, prints the error message and goes back to the main menu. When successful, the code empties the input buffer with the help of a loop that places 10 null characters into the buffer. It is followed by asking for a new PIN. The new entry is also treated likewise by being captured, measured, null-terminated, hashed and copied in the ``stored_pin_hash`` variable, which replaces the PIN. A success message is displayed confirming that the update was done and the control is returned to the main menu. This routine justifiably performs security of both verification and change of PIN through hash, buffer operations, and branching of logic to do validation. It also contains graphic feedback to the customer in every step, so it is clear and correct.

```

transfer_account:
    ; Ask from which account to transfer
    mov dx, offset msg_transfer_from
    mov ah, 09h
    int 21h

    mov ah, 01h        ; Read single character (1 or 2)
    int 21h
    sub al, '0'
    cmp al, 1
    je set_transfer_savings
    cmp al, 2
    je set_transfer_investment
    jmp main_menu      ; Invalid input

set_transfer_savings:
    mov si, offset savings_balance
    jmp continue_transfer

set_transfer_investment:
    mov si, offset investment_balance
    jmp continue_transfer

```

*Figure 5.21*

```

continue_transfer:
    ; Ask for recipient account number
    mov dx, offset msg_transfer_to
    mov ah, 09h
    int 21h
    lea dx, account_number_buffer
    mov ah, 0Ah
    int 21h

    ; Ask for amount
    mov dx, offset msg_transfer_amt
    mov ah, 09h
    int 21h
    lea dx, input_buffer
    mov ah, 0Ah
    int 21h

    ; Convert amount from input buffer
    push si                ; preserve SI which holds account address
    lea si, input_buffer + 2
    call string_to_number
    mov bx, ax              ; BX = amount to transfer
    mov [last_transaction_amount], bx
    pop si                 ; restore account pointer

    ; Calculate tax = amount * 6 / 100
    mov ax, bx              ; AX = amount
    mov cx, ax              ; CX = original amount
    mov dx, 0
    mov bx, 6
    mul bx                  ; AX = amount * 6
    mov bx, 100
    div bx                  ; AX = tax
    mov dx, ax              ; DX = tax

```

Figure 5.22

```

; Check if account has enough balance
mov bx, ax          ; BX = amount + tax
mov ax, [si]
cmp ax, bx
jae transfer_continue_logic
jmp not_enough_transfer

transfer_continue_logic:
; Deduct total (amount + tax)
sub ax, bx
mov [si], ax
mov al, 3           ; type = 3 (transfer)
; BX already contains amount + tax
; DX already contains tax

; ?? ADD THIS BLOCK HERE
mov al, 3           ; type: transfer
mov bx, cx          ; cx = original amount
; dx = tax already set

; Show success message
mov dx, offset msg_transfer_done
mov ah, 09h
int 21h
jmp main_menu

```

Figure 5.23

## 5.11 Transaction

This assembly program contains data section block, where all variables, strings and constants applied within the THANES ATM SYSTEM are declared. It starts with ASCII art (banner\_art) formed in visualizing the name of ATM system. Then it determines prompts such as username\_prompt and password\_prompt, success and failure messages (login\_success, login\_fail). The static credentials are kept in stores\_user and stores\_pass. The user input is allocated in input buffers like user\_buffer and pass\_buffer. It then initializes balance, savingsbalance and investmentbalance to 1000. The main menu items will be presented using a series of menu lines (line1 to line9). Some of the message constants used are msg\_withdraw, msg\_deposit and msg\_transfer\_amt that are displayed when making transactions. It has also certain messages concerning handling of cheques, pin change, and messages due to error such as msg\_insufficient and msg\_wrong\_pin. Other utility variables such as, last\_transaction\_amount, account\_number\_buffer, and hashed PIN (stored\_pin\_hash) are defined here

as well. Finally, message blocks such as receipt\_msg, amount\_msg and tax\_msg assist with the ability to print the receipt. In general, the given section gathers all the static data, and one can easily manipulate user interaction texts and vital program variables in it.

```
.print_digits:
    pop dx
    add dl, '0'
    mov [di], dl
    inc di
    loop .print_digits
    pop dx
    pop cx
    pop bx
    pop ax
    ret

not_enough:
    mov dx, offset msg_insufficient
    mov ah, 09h
    int 21h
    jmp main_menu

login_failed:
    mov dx, offset login_fail
    mov ah, 09h
    int 21h
    jmp login_screen

compare_strings:
    push cx
    push si
    push di
    cmp byte ptr [di], byte ptr [si]
    jne not_enough
    jmp login_screen
```

Figure 5.24



```
compare_loop:
    mov al, [si]
    cmp al, '$'
    je compare_done
    cmp al, [di]
    jne not_equal
    inc si
    inc di
    jmp compare_loop
compare_done:
    cmp byte ptr [di], '$'
    jne not_equal
    xor ax, ax
    jmp cmp_end
not_equal:
    mov ax, 1
cmp_end:
    pop di
    pop si
    pop cx
    ret

print_number:
    mov cx, 0
    mov bx, 10
```

*Figure 5.25*

```

.next_digit:
    xor dx, dx
    div bx
    push dx
    inc cx
    cmp ax, 0
    jne .next_digit

print_loop:
    pop dx
    add dl, '0'
    mov ah, 02h
    int 21h
    loop print_loop
    ret

string_to_number:
    xor ax, ax
    xor cx, cx
str_loop:
    mov cl, [si]
    cmp cl, 13
    je str_done
    cmp cl, 10
    je str_done
    sub cl, '0'
    mov bx, ax
    shl ax, 1
    mov dx, ax
    shl ax, 2
    add ax, dx
    add ax, cx
    inc si
    jmp str_loop

```

*Figure 5.26*

## 5.12 Strings to Number conversation

This code has utility procedures of handling strings and numbers, error handling and comparing two strings in an assembly-based ATM program. The `copy_str` procedure strings a null terminated string on the source index register (SI) to the destination index (DI) one-character at a time using `lodsb` and `stosb`. It stops when a character with value of null is reached. The `append_number` routine converts a number in AX into its ASCII equivalent and appends it to the string at DI. This is by repeated division by 10 into storage of digits in that order. Upon conversion, `.print_digits` removes each character in the stack, converts it to ASCII by adding a zero and puts it back in memory through DI. This makes it that the numeric value is presented correctly upon display. The `not_enough` and `login_failed` functions are

just routines dealing with error messages and containing respective error messages and allowing the reentry into main menu or to the streets of log in. `compare_strings` is string comparing routine which checks two strings (pointed on SI, DI) character by character. It comes out early on any mismatch and leaves AX set at 1 (not equal) otherwise leaving 0 (equal). The routine employs dollar sign ( `$` ) as a terminator of string. The `print_number` routine pre-loads the divisor (`BX = 10`) and prepares the count in CX so that the subsequent number to string conversions can take place. These are routines that have been necessitated by numerous ATM functions including input/output, authentication and also check balances. They present the usual x86 assembly low level manipulation based on interruptions (`int 21h`), control stack (`push/pop`) and conditional transfers of control (`jmp, je, jne`) that is common. They all work in concert to support some of the most important aspects of any ATM software including string copy, integer conversion, validation and user alert/response which is essential in allowing a working and easy to use ATM software within DOS.

```

not_enough_transfer:
    mov dx, offset msg_insufficient
    mov ah, 09h
    int 21h
    jmp main_menu

print_receipt:
    mov dx, offset receipt_msg
    mov ah, 09h
    int 21h

    ; Show Amount
    mov dx, offset amount_msg
    mov ah, 09h
    int 21h

    mov ax, [last_transaction_amount]
    call print_number

    ; Calculate Tax (Amount * 6)
    mov dx, offset tax_msg
    mov ah, 09h
    int 21h

    mov ax, [last_transaction_amount]
    mov bx, 6
    mul bx          ; AX = AX * 6 (result in AX)
    call print_number

    jmp main_menu

exit_program:
    mov ah, 4Ch
    int 21h

```

Figure 5.27

## 5.13 Print Receipt

This assembly does two essential pieces of code; it is the printing receipt and exiting program plus an error message of not enough transfer. The print\_receipt procedure prints a transaction receipt, and it starts with printing the receipt\_msg by using the DOS interrupting 21h function 09h. Then it displays the sum used in the previous transaction. This is achieved by putting the amount\_msg on the display and loading the last\_transaction\_amount to the AX register and summoning the print\_number procedure to change and present the amount to the ASCII code. Upon this, the program will compute and show the tax on the transaction. It prints the value of tax\_msg and reloads the transaction amount in AX and multiplies it by 6 using the MUL instruction (to calculate 6 percent tax) and again calls the

print\_number to print the amount of tax. The not\_enough\_transfer routine shows a message of there being no funds (msg\_insufficient) and logs the user to the main menu. It is activated when the balance of the user is insufficient to make a transfer (including tax). Finally, the exit\_program routine simply ends program by storing the AH register to 4Ch and using the interrupt 21h. It is the normal way of leaving a DOS program gracefully. These processes provide the feedback of the user and best program termination in ATM application.

```
hash_pin_input:
    xor ax, ax
    ; DS:SI must point to buffer data (pass_buffer + 2)
    ; length is at pass_buffer + 1 (1 byte)
    ; since SI = pass_buffer + 2, the length byte is at SI - 1

    mov cl, [si - 1]    ; input length from buffer
    xor ch, ch          ; clear upper byte of CX

    xor bx, bx
.hash_loop:
    mov bl, [si]
    add ax, bx
    inc si
    loop .hash_loop
    ret

hash_done:
    ret

    ; ax = number to print blinking in green

end start
```

Figure 5.28

## 5.14 Hashing

hash\_pin\_input procedure computes a simple value of a pin that is entered by the user. It also starts looping by setting the AX bit to zero to store the hash value. The SI register indicates the offset of the initial of PIN characters (off pass\_buffer+2), the length of PIN is read out of pass\_buffer+1. The loop is iterated through the length of the PIN (CL holds the length) and the ASCII of each character ([SI]) is added into AX. This constitutes the hash. It is a simple checksum to confirm input user PINs during the authentication process.

## 6.0 Conclusion

All the code screenshots will represent the design and implementation of a low-level banking application implemented in x86 Assembly spanning a complex operation of the banking application amalgamated in structured, modular subroutines. All functional blocks, e.g. checking balance, withdrawing and depositing cash, transferring money, working on cheques, PIN verification, providing receipts and exit procedures have a definite logical structure and employ DOS interrupts as a way of interacting with the user. The checking, withdrawing and depositing functions based on balance are done through registers and memory bits to load and modify the values. The possibility to branch and do conditional logic (`cmp`, `je`, `jne`), and carefully use stack operations (`push`, `pop`) so that essential data is preserved, such as buffer pointers and values, when subroutines are called and when the resulting values are returned. It also has logic to hash and check PINs in this application, so it has a minimal amount of user authentication. It implements manual looping and arithmetic to compute hash and convert digits to numbers (i.e. change ASCII codes to integers or numerical data to string to display their values). Tax calculation is a part of the transfer functions that make the financial operations more realistic. The incorporation of routines such as `print_number`, `append_number` and `compare_strings` manifests a tendency to abstract and reuse the functionality to be clear and efficient. Finally, this system is supplemented with the implementation of the basic decoding operation written in Python, which reads data stored in a file, converting this data to hex value and then decodes the information back to viewable ASCII. This is the section, which is not closely built up into the Assembly code, but shows knowledge of text encoding and decoding, which is relevant in files and data security applications. Altogether, the code snippets provide a complete simulation of a banking system that takes user input, string manipulation, numeric calculations, security checks and output receipt. It demonstrates a high level of knowledge regarding low-level programming and resource management and system-level interfacing, which are also important in generating efficient and secure applications.

## 7.0 References

- Corporation., I. (2021). *Intel® 64 and IA-32 Architectures Software Developer's Manual*. Retrieved from Intel: <https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html>
- Foundation, P. S. (2024). *Python 3 Documentation*. Retrieved from Python: <https://docs.python.org/3/>
- Hanna, K. T. (2025, March 17). *What is a buffer overflow? How do these types of attacks work?* Retrieved from informa: <https://www.techtarget.com/searchsecurity/definition/buffer-overflow>
- Irvine, K. R. (2011). *Assembly Language for x86 Processors (6th ed.)*. Pearson. Retrieved from asmirvine: <https://asmirvine.com>
- JaJirayu. (2024, Jun 3). *Assembly to Machine*. Retrieved from Medium: <https://medium.com/@jajirayu/assembly-to-machine-696d98c9a61a>
- madStacks. (2021). *Transformation*. Retrieved from picoCTF: <https://play.picoctf.org/practice/challenge/104?category=3&page=1>
- What is an Assembler?* (n.d.). Retrieved from Lenovo: <https://www.lenovo.com/in/en/glossary/assembler/?orgRef=https%253A%252F%252Fwww.bing.com%252F#>