

VIETNAM NATIONAL UNIVERSITY, HANOI  
INTERNATIONAL SCHOOL



**FINAL REPORT**  
**STUDENT MANAGEMENT SYSTEM**  
**USING EMU8086**

**Team Leader: Nguyen Duc Quang Anh**

**ID: 22070306**

**Course: INS3168 - INS316802**

**Supervisor: Pham Hai Yen**

*Hanoi, .....2024*

## TABLE OF CONTENT

<b>I. Abstract theory of assembly language.....</b>	<b>2</b>
<b>II. Analysis reference documents.....</b>	<b>3</b>
1. System Name: Student Management System.....	3
2. Tasks.....	3
2.1. Login.....	3
2.2. Create account.....	3
2.3. Logout.....	3
2.4. Enter student name.....	3
2.5. Input score.....	3
2.6. Delete student.....	3
2.7. View student.....	4
3. Tools and Functions.....	4
3.1. Tools.....	4
3.1.1. Emu8086 Emulator:.....	4
3.1.2. Memory Management.....	4
3.1.3 Interrupt Handling Support.....	4
3.1.4. Debugger.....	4
3.2. Functions.....	4
3.2.1. Input and Output Operations:.....	4
3.2.2. String Handling:.....	5
3.2.3. Arithmetic Operations.....	5
3.2.4. Memory Management.....	5
3.2.5. Control Flow.....	5
<b>III. Flowchart.....</b>	<b>6</b>
<b>IV. Coding and explain code.....</b>	<b>7</b>
1. Program Structure:.....	7
2. Data Management:.....	8
3. The .CODE section:.....	9
4. Key Functions:.....	11
5. File Handling:.....	18
6. Security Mechanisms:.....	19
7. Potential Improvements:.....	20
<b>V. Run simulation with EMU8086 software.....</b>	<b>20</b>
<b>Reference.....</b>	<b>22</b>

Student Name	Student ID	Class	Contributions
Nguyễn Đức Quang Anh	22070306	AIT2022A 1	33.34%
Đỗ Xuân Minh Đức	22070047	AIT2022A 1	33.33%
Nguyễn Minh Anh	22071104	AIT2022A 1	33.33%

### **I. Abstract theory of assembly language.**

Assembly language, also referred to as ASM or symbolic machine code, is a low-level programming language that maintains a close correspondence with machine code instructions. It provides developers with precise control over hardware, offering a 1:1 mapping between its statements and machine instructions. Additionally, it supports features like constants, comments, directives, and symbolic labels for registers and memory locations, enhancing readability and flexibility [1][2].

This precision requires a thorough understanding of system architecture, including registers, memory layouts, and stack operations. Such knowledge is essential for tasks like managing student records, optimizing data storage, and performing accurate computations, such as calculating averages in a Student Management System [3]. Despite its steep learning curve and longer development times compared to high-level languages, assembly allows developers to fully leverage hardware capabilities, ensuring system responsiveness and efficiency.

The use of assembly language in systems like a Student Management System underscores its enduring relevance. By directly interacting with the processor and memory, assembly enables the creation of robust, resource-efficient applications while teaching foundational computing principles. This combination of theoretical knowledge and practical implementation highlights its value in both educational and real-world contexts [4][5].

## **II. Analysis reference documents**

### **1. System Name: Student Management System**

#### **2. Tasks**

##### **2.1. Login**

This functionality ensures that only authorized users can access the system. The program will prompt users to enter their credentials, which will then be validated against pre-stored data. If the credentials match, access is granted; otherwise, an error message is displayed, and the user is prompted to try again.

##### **2.2. Create account**

This feature allows administrators to add new user accounts. The user will provide details such as a username and password, which are stored securely in the system. This functionality should also ensure that duplicate accounts cannot be created and that passwords are stored in an encrypted or protected format.

##### **2.3. Logout**

To maintain security, this feature enables users to terminate their session safely. Upon logging out, all sensitive data related to the user session should be cleared from memory to prevent unauthorized access.

##### **2.4. Enter student name**

This feature allows users to input student names, which are stored in a predefined format in memory. Care should be taken to handle string inputs of varying lengths and ensure that the names are correctly aligned in the database structure.

##### **2.5. Input score**

Users can input students' scores for various subjects. The system should validate the scores to ensure they fall within acceptable ranges (e.g., 0-100). These scores will be stored alongside the corresponding student records for further analysis or display.

##### **2.6. Delete student**

This feature allows the user to remove a student's record completely from the system. The program should confirm the action to prevent accidental deletions and ensure that memory allocated to the deleted record is released.

## 2.7. View student

This function displays a list of all students currently stored in the system. The list should include names, scores, and any additional data. The display should be well-organized and user-friendly, making it easy for users to review and verify the records.

## 3. Tools and Functions

### 3.1. Tools

#### 3.1.1. Emu8086 Emulator:

Emu8086 provides an integrated environment for writing, assembling, and debugging assembly code. Its built-in tools allow visualization of registers, memory, and flags, enabling developers to understand the flow of data and execution in real-time [3]. It is particularly useful for beginners learning assembly language due to its intuitive interface and simulation capabilities [6].

#### 3.1.2. Memory Management

Memory segments like the **Data Segment (DS)** and **Stack Segment (SS)** in Emu8086 are utilized for storing student records and managing temporary data. This segmentation model allows efficient organization and manipulation of data, critical for a Student Management System [4].

#### 3.1.3 Interrupt Handling Support

Emu8086 supports a range of DOS and BIOS interrupts, providing a standardized way to handle input/output operations and file management [7]. For example, INT 21H enables features like reading from the keyboard, displaying messages, and file handling, which are essential for creating interactive programs

#### 3.1.4. Debugger

The step-by-step execution feature in Emu8086 helps developers track instruction execution and monitor register states, ensuring logical correctness [8]. This functionality makes debugging intuitive and efficient.

### 3.2. Functions

#### 3.2.1. Input and Output Operations:

Using **INT 21H**, the system performs essential I/O tasks:

- **Function 01H:** Input a single character from the user.

- **Function 09H:** Display a string on the screen.
- **Function 0AH:** Accept a string input from the user.

These functions handle tasks such as entering student names and displaying lists, ensuring efficient interaction between the user and the system [9].

### 3.2.2. String Handling:

String operations, such as comparing names or storing input, rely on iterative processing. Assembly instructions like MOV, CMP, and conditional jumps (JE, JNE) facilitate these operations [4]. This is critical for functions like searching for a specific student record or updating data.

### 3.2.3. Arithmetic Operations

Arithmetic instructions like ADD, SUB, MUL, and DIV are used to handle computations. For instance, calculating a student's average score involves dividing the total score by the number of subjects using the DIV instruction [10]. These low-level operations ensure precise and optimized performance.

### 3.2.4. Memory Management

Data for student records is stored in arrays using directives like DB (Define Byte) and DW (Define Word). The stack is used for temporary storage during operations, utilizing PUSH and POP instructions to preserve data integrity [11].

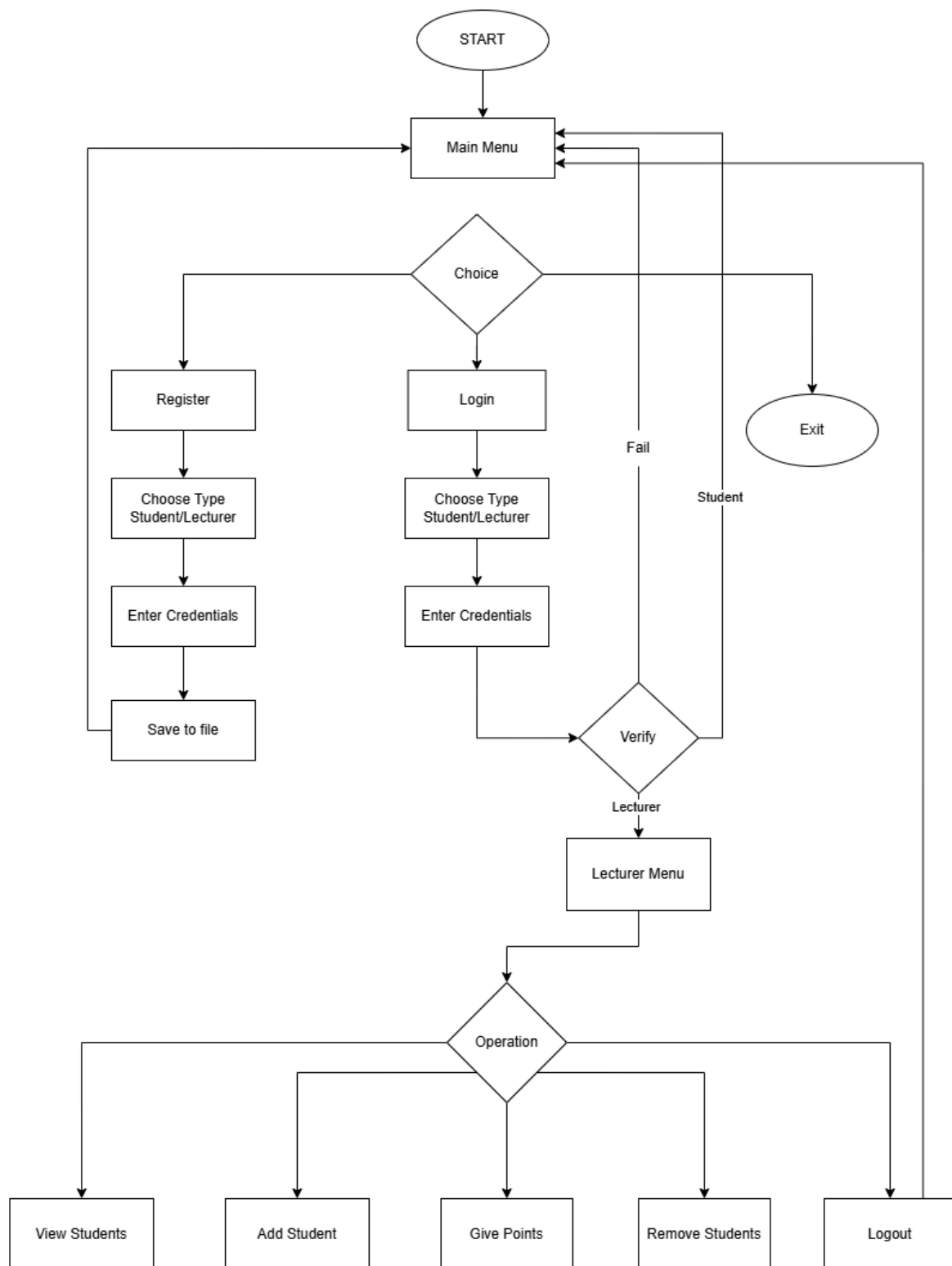
### 3.2.5. Control Flow

The system's logic relies on conditional and unconditional branching:

- **Conditional Jumps:** Instructions like JE (Jump if Equal) and JL (Jump if Less) manage decision-making, such as checking input validity.
- **Loops:** Instructions like LOOP simplify iterating through student records.

Control flow management is essential for implementing menus and navigating system functionalities [8].

### III. Flowchart



The flowchart illustrates the operation process of a user management application with two main objects: Students and Lecturers. The process is divided into the following steps:

1. **Start:** The system starts and displays the Main Menu to the user.
2. **Main Menu:** The user can choose the actions: Login, Register or Exit. If Exit is selected, the process ends.
3. **Register:** The user selects the account type (Student or Lecturer), then enters the necessary information. The information is saved in the database.
4. **Login:** The user enters the authentication information and selects the account type. The system checks the information:
  - If it is a Student, no further access.
  - If it is a Lecturer, it is transferred to the Lecturer Menu.
5. **Lecturer Menu:** Provides the main functions:
  - View student list
  - Add student
  - Give student score
  - Delete student
  - Logout
6. **Exit:** User can return to the Main Menu or exit completely.

This diagram helps to clearly visualize the system's workflow, support developers to test and improve program logic, ensure a friendly and easy-to-use interface. The system provides essential functions for lecturers in managing students, while maintaining clear authorization between user types.

#### **IV. Coding and explain code**

The program is written in Assembly (x86) and implements an academic login system with two types of users: Students and Lecturers. The program uses a small memory model (SMALL) and is organized into main sections:

##### **1. Program Structure:**

The program is organized using the **SMALL** memory model, which is structured into three primary segments. The **STACK** segment allocates 100H bytes of memory for stack operations. The **DATA** segment holds declarations for variables and data,



while the **CODE** segment contains the source code for procedures and functionalities. This organization allows for efficient memory usage and clear separation of data, instructions, and stack operations within the program.

## 2. Data Management:

```
.DATA
; Messages
WELCOME DB 'Welcome to Academic Login System', 13, 10, ' '
MENU DB '1. Login', 13, 10
DB '2. Register', 13, 10
DB '3. Exit', 13, 10
DB 'Choose option: $'
USER_TYPE DB '1. Student', 13, 10
DB '2. Lecturer', 13, 10
DB 'Choice: $'
USER_PROMPT DB 'Enter username: $'
PASS_PROMPT DB 'Enter password: $'
SUCCESS DB 'Operation successful!', 13, 10, '$'
FAIL DB 'Operation failed!', 13, 10, '$'
NEWLINE DB 13, 10, '$'

; Lecturer menu
LEC_MENU DB '== Lecturer Menu ==', 13, 10
DB '1. View Student List', 13, 10
DB '2. Add Student', 13, 10
DB '3. Give Points', 13, 10
DB '4. Remove Student', 13, 10
DB '5. Logout', 13, 10
DB 'Choose option: $'
POINT_MSG DB 'Enter points (0-10): $'
STU_ID_MSG DB 'Enter student ID: $'
LIST_HEAD DB 'ID Name Points', 13, 10, '$'
NO_STU_MSG DB 'No student found!', 13, 10, '$'

; File names
STU_FILE DB 'STUDENT.DAT', 0
LEC_FILE DB 'LECTURE.DAT', 0
RECORD_FILE DB 'STUREC.DAT', 0 ; Student records file
TEMP_FILE DB 'TEMP.DAT', 0 ; Temporary file for operations

; Buffers
USERNAME DB 20 DUP(0) ; Username buffer
PASSWORD DB 20 DUP(0) ; Password buffer
RECORD DB 40 DUP(0) ; Buffer for file operations

; Student record buffers <fixed length format>
STU_ID DB 5 DUP(0) ; Student ID
STU_NAME DB 10 DUP(0) ; Student name
STU_POINTS DB 0 ; Student points
CURR_POS DW 0 ; Current position in file

FILE_HANDLE DW ? ; File handle
TEMP_HANDLE DW ? ; Temporary file handle
USER_CHOICE DB 0 ; Store user choice
```

### a) File Structure:

- **STUDENT.DAT:** Stores login credentials for students.
- **LECTURE.DAT:** Stores login credentials for lecturers.
- **STUREC.DAT:** Stores student records in a fixed format:
  - **ID:** 5 bytes.

- **Name:** 10 bytes.
- **Points:** 1 byte.
- **Newline:** 2 bytes (CR + LF).
- **Total:** 18 bytes per record.

b) Buffers and Variables:

- **USERNAME, PASSWORD:** 20-byte buffers for storing usernames and passwords.
- **RECORD:** 40-byte buffer for file reading and writing.
- **STU\_ID, STU\_NAME, STU\_POINTS:** Buffers for student data.
- **FILE\_HANDLE, TEMP\_HANDLE:** Store file handles during processing.
- **USER\_CHOICE:** Stores user selection (1-Student, 2-Lecturer).

3. The .CODE section:

```
.CODE
MAIN PROC
    MOV AX, @DATA
    MOV DS, AX

    ; Display welcome message
    LEA DX, WELCOME
    MOV AH, 9
    INT 21H

MAIN_MENU:
    ; Display menu
    LEA DX, MENU
    MOV AH, 9
    INT 21H

    ; Get choice
    MOV AH, 1
    INT 21H

    ; Print newline after input
    PUSH AX
    LEA DX, NEWLINE
    MOV AH, 9
    INT 21H
    POP AX

    ; Save user's choice

    ; Process choice
    CMP AL, '1'
    JE LOGIN_PROCESS
    CMP AL, '2'
    JE REGISTER_PROCESS
    CMP AL, '3'
    JE EXIT_PROGRAM
    JMP MAIN_MENU

    ; Restore user's choice
```

a) Data Segment Setup:

- The **MOV AX, @DATA** loads the data segment address into the AX register.
- **MOV DS, AX** sets the data segment register (**DS**) to the value stored in **AX**, allowing access to the variables and constants defined in the data segment.

b) Display Welcome Message:

- **LEA DX, WELCOME**: Loads the address of the WELCOME string into register DX.
- **MOV AH, 9**: Prepares to call interrupt 21h with function 09h (display string).
- **INT 21H**: Triggers the interrupt to display the string pointed to by DX.

c) Main Menu Loop:

- **LEA DX, MENU**: Loads the address of the MENU string into DX.
- **MOV AH, 9**: Prepares for the interrupt call that will display the string located at DX.
- **INT 21H**: Displays the MENU string on the screen.

d) Get User Choice:

- **MOV AH, 1**: Prepares for a 21h interrupt to read a single character from the user.
- **INT 21H**: Waits for the user to input a character (pressed key), which is stored in the AL register.

e) Print Newline:

- **PUSH AX**: Saves the user's input (stored in AL) onto the stack to prevent overwriting.
- **LEA DX, NEWLINE**: Loads the address of the newline string (NEWLINE) into DX.
- **MOV AH, 9**: Prepares to display the NEWLINE string.
- **INT 21H**: Displays the newline string, effectively moving the cursor to the next line.
- **POP AX**: Restores the original value of AX (the user's choice) from the stack.

f) Process the User's Choice:

- **CMP AL, '1'**: Compare the value of AL with the ASCII value of '1'.
- **JE LOGIN\_PROCESS**: If AL is equal to '1', jump to LOGIN\_PROCESS
- **CMP AL, '2'**: Compare the value of AL with '2'.
- **JE REGISTER\_PROCESS**: If AL is equal to '2', jump to REGISTER\_PROCESS.
- **CMP AL, '3'**: Compare the value of AL with '3'.
- **JE EXIT\_PROGRAM**: If AL is equal to '3', jump to EXIT\_PROGRAM.
- **JMP MAIN\_MENU**: If none of the conditions are true, jump back to the MAIN\_MENU to display the menu again.

4. Key Functions:

```
LOGIN_PROCESS :  
    CALL GET_USER_TYPE  
    CALL GET_CREDENTIALS  
    CALL VERIFY_CREDENTIALS  
    JMP MAIN_MENU  
  
REGISTER_PROCESS :  
    CALL GET_USER_TYPE  
    CALL GET_CREDENTIALS  
    CALL STORE_CREDENTIALS  
    JMP MAIN_MENU
```

**CALL *FUNCTIONS\_NAME***: Move to other function

**JUMP *SECTION\_NAME***: Fetch the address (absolute or relative) of the target instruction is specified in the JMP instruction, update the Instruction Pointer (IP) register to point to the specified address, effectively jumping to that location.

a) Login/Register process:

- **GET\_USER\_TYPE:** Identifies the type of account (student/lecturer).

```
GET_USER_TYPE:
    ; Display user type menu
    LEA DX, USER_TYPE
    MOV AH, 9
    INT 21H

    ; Get choice
    MOV AH, 1
    INT 21H

    ; Print newline after input
    PUSH AX                ; Save user's choice
    LEA DX, NEWLINE
    MOV AH, 9
    INT 21H
    POP AX                ; Restore user's choice

    SUB AL, '0'
    MOV USER_CHOICE, AL
    RET
```

- **PUSH AX:** Temporarily saves the user's input (stored in AX) onto the stack to preserve it while printing the newline.
- **LEA DX, NEWLINE:** Loads the address of the NEWLINE string (containing a newline character) into DX.
- **MOV AH, 9:** Sets up DOS interrupt 21h, function 9, to print the newline.
- **INT 21H:** Calls the DOS interrupt to print the newline.
- **POP AX:** Restores the user's input from the stack back into the AX register.
- **SUB AL, '0':** Converts the ASCII value of the character (e.g., '1') into its numeric value (e.g., 1). This works because the ASCII codes for digits are sequential ('0' is 48, '1' is 49, etc.).
- **MOV USER\_CHOICE, AL:** Stores the numeric value of the user's choice in the USER\_CHOICE memory location.
- **GET\_CREDENTIALS:** Captures username/password

```
GET_CREDENTIALS:
    ; Get username
    LEA DX, USER_PROMPT
    MOV AH, 9
    INT 21H

    LEA SI, USERNAME
    MOV CX, 10
```

- **LEA SI, USERNAME:**

Loads the address of the USERNAME buffer into the SI register. This buffer is where the entered username will be stored. The USERNAME buffer should be defined in the data segment and have enough space to hold the user's input (likely 10 characters in this case).

- **MOV CX, 10:**

Sets the loop counter CX to 10, indicating that up to 10 characters can be stored in the USERNAME buffer. This is a safety measure to prevent buffer overflows and ensures only 10 characters (or fewer) are captured.

- **VERIFY\_CREDENTIALS:** Verifies credentials by checking the file.

```
VERIFY_CREDENTIALS:
; Choose file based on user type
CMP USER_CHOICE, 1
JE OPEN_STU_LOGIN
JMP OPEN_LEC_LOGIN
```

- **CMP USER\_CHOICE, 1:**

Compare the value in the USER\_CHOICE variable with 1. The comparison sets the processor's flags based on the result:

- If USER\_CHOICE == 1: The zero flag (ZF) is set.
- If USER\_CHOICE != 1: The zero flag is cleared.
- **STORE\_CREDENTIALS:** Saves new accounts to the corresponding file.

```
STORE_CREDENTIALS:
; Choose file based on user type
CMP USER_CHOICE, 1
JE CREATE_STU_FILE
JMP CREATE_LEC_FILE
```

b) Student Management (Lecturer Menu):

```
LECTURER_MENU:
    ; Display lecturer menu
    LEA DX, LEC_MENU
    MOV AH, 9
    INT 21H

    ; Get choice
    MOV AH, 1
    INT 21H

    ; Print newline after input
    PUSH AX                ; Save user's choice
    LEA DX, NEWLINE
    MOV AH, 9
    INT 21H
    POP AX                ; Restore user's choice

    CMP AL, '1'
    JE VIEW_STUDENTS
    CMP AL, '2'
    JE ADD_STUDENT
    CMP AL, '3'
    JE GIVE_POINTS
    CMP AL, '4'
    JE REMOVE_STUDENT
    CMP AL, '5'
    JE LOGIN_SUCCESS
    JMP LECTURER_MENU
```

**JE FUNCTIONS\_NAME:** jump if equal, jump to the function written and saved in the register.

**VIEW\_STUDENTS:** Displays a list of students from STUREC.DAT.

```
VIEW_STUDENTS:
    ; Open student records file
    LEA DX, RECORD_FILE
    MOV AH, 3DH          ; Open file
    MOV AL, 0            ; Read mode
    INT 21H
    JC NO_STUDENTS      ; If file doesn't exist

    MOV FILE_HANDLE, AX

    ; Display header
    LEA DX, LIST_HEAD
    MOV AH, 9
    INT 21H
```

The file handle, stored in AX after a successful file open operation, is moved to FILE\_HANDLE (a variable in memory).

- **LEA DX, LIST\_HEAD:** Loads the address of the LIST\_HEAD message (a header string) into DX.
- **MOV AH, 9:** Sets the function number for DOS interrupt 21h to 9, which displays a null-terminated string.

- **INT 21H:** Calls the interrupt to display the string.

**ADD\_STUDENT:** Adds a new student with an initial score of 0.

```
ADD_STUDENT:
    ; Get student ID
    LEA DX, STU_ID_MSG
    MOV AH, 9
    INT 21H

    ; Read student ID
    LEA SI, STU_ID
    MOV CX, 5                ; Read 5 characters for ID
    ---
```

- **LEA DX, STU\_ID\_MSG:** Loads the address of the message (STU\_ID\_MSG) into DX.
- **LEA SI, STU\_ID:** Loads the address of the variable (STU\_ID) where the student ID will be stored into the SI register.
- **MOV CX, 5:** Sets up the count register (CX) with a value of 5. This indicates that the student ID is expected to have 5 characters.

**GIVE\_POINTS:** Updates student scores (range: 0-10).

```
GIVE_POINTS:
    ; Clear student ID buffer first
    LEA SI, STU_ID
    MOV CX, 5
```

- **LEA SI, STU\_ID:** Loads the address of the STU\_ID buffer into the SI register.
- **STU\_ID** is likely a 5-byte memory space for storing the student ID.
- **MOV CX, 5:** Sets the CX register to 5, indicating the buffer length.



**REMOVE\_STUDENT:** Deletes a student from the list.

```
REMOVE_STUDENT:
; Clear student ID buffer first
LEA SI, STU_ID
MOV CX, 5
CALL CLEAR_BUFFER

; First check if original file exists
LEA DX, RECORD_FILE
MOV AH, 3DH          ; Open file
MOV AL, 0            ; Read only first to check existence
INT 21H
JC NO_RECORDS        ; If file doesn't exist

; Close the file and reopen in read/write mode
MOV BX, AX
MOV AH, 3EH
INT 21H

; Open file again in read/write mode
LEA DX, RECORD_FILE
MOV AH, 3DH          ; Open file
MOV AL, 2            ; Read/Write mode
INT 21H
MOV FILE_HANDLE, AX

; Get student ID to remove
LEA DX, STU_ID_MSG
MOV AH, 9
INT 21H

; Read student ID
MOV CX, 5            ; Read 5 characters for ID
LEA SI, STU_ID
```

- **LEA SI, STU\_ID:** Loads the address of the STU\_ID buffer into the SI register, which will hold the student ID.
- **MOV CX, 5:** Specifies the number of bytes (5 characters) to be cleared.
- **CALL CLEAR\_BUFFER:** Calls a subroutine (CLEAR\_BUFFER) to clear the buffer for the student ID.
- **LEA DX, RECORD\_FILE:** Loads the address of the RECORD\_FILE (the file where student records are stored) into the DX register.
- **MOV AH, 3DH:** Specifies the DOS function to open a file (function 3Dh).
- **MOV AL, 0:** Opens the file in "read-only" mode (to check if it exists).
- **MOV BX, AX:** Moves the file handle (stored in AX from the file open operation) into BX to prepare it for closing.
- **MOV AH, 3EH:** Sets the function to close a file (3Eh).
- **LEA DX, RECORD\_FILE:** Loads the address of the RECORD\_FILE again into the DX register.
- **MOV AH, 3DH:** Specifies the DOS function to open a file (3Dh).

- **MOV AL, 2:** Opens the file in "read/write" mode (function 2).
- **LEA DX, STU\_ID\_MSG:** Loads the address of a message (such as "Enter Student ID to remove") into the DX register.
- **MOV AH, 9:** Specifies the DOS function to display a null-terminated string (function 9).
- **MOV CX, 5:** Specifies that 5 characters (the length of the student ID) will be read.
- **LEA SI, STU\_ID:** Loads the address of the STU\_ID buffer into the SI register, where the input will be stored.

## 5. File Handling:

```
CREATE_USER_FILE:
    ; Try to create new file
    MOV AH, 3CH          ; Create file function
    MOV CX, 0            ; Normal attribute
    INT 21H
    JNC FILE_CREATED     ; If file created successfully

    ; If file exists, open it for append
    MOV AH, 3DH          ; Open file function
    MOV AL, 2            ; Write mode
    INT 21H
    JNC FILE_OPENED
    JMP STORE_FAIL       ; If both create and open fail

FILE_CREATED:
    MOV FILE_HANDLE, AX
    JMP WRITE_CREDS

FILE_OPENED:
    MOV FILE_HANDLE, AX

    ; Seek to end of file
    MOV AH, 42H          ; SEEK
    MOV AL, 2            ; From end of file
    MOV BX, FILE_HANDLE
    XOR CX, CX           ; High word of offset = 0
    XOR DX, DX           ; Low word of offset = 0
    INT 21H

WRITE_CREDS:
    ; Write username
    MOV AH, 40H          ; Write to file
    MOV BX, FILE_HANDLE
    LEA DX, USERNAME
    MOV CX, 10           ; Username length
    INT 21H

    ; Write password
    MOV AH, 40H
    MOV BX, FILE_HANDLE
    LEA DX, PASSWORD
    MOV CX, 10           ; Password length
    INT 21H

    ; Write newline
    MOV AH, 40H
    MOV BX, FILE_HANDLE
    LEA DX, NEWLINE
    MOV CX, 2            ; CR+LF
    INT 21H
```

```

; Close file
MOV AH, 3EH
MOV BX, FILE_HANDLE
INT 21H

LEA DX, SUCCESS
MOV AH, 9
INT 21H
RET

STORE_FAIL:
LEA DX, FAIL
MOV AH, 9
INT 21H
RET

```

The program uses **DOS interrupts (INT 21H)** for file operations:

- **MOV AH=3CH:** Create a new file.
- **MOV AH=3DH:** Open an existing file.
- **MOV AH=3EH:** Close a file.
- **MOV AH=3FH:** Read from a file.
- **MOV AH=40H:** Write to a file.
- **MOV AH=41H:** Delete a file.
- **MOV AH=42H:** Move the file pointer.
- **MOV AH=56H:** Rename a file.

## 6. Security Mechanisms:

The program incorporates several security mechanisms, but there are areas that could be strengthened. Clear Role Separation is enforced, ensuring that students and lecturers have distinct access levels, with tailored permissions for each user type. However, plain text passwords are currently stored, which presents a security risk; this could be improved by implementing encryption to securely store passwords. Strict verification is used during login, with a character-by-character comparison to prevent unauthorized access, though this method can still be vulnerable to certain attacks and could be improved with more advanced hashing techniques. Error handling is implemented to provide meaningful feedback for failed operations, helping users understand what went wrong without revealing sensitive system details. While these mechanisms offer a baseline level of security, upgrading

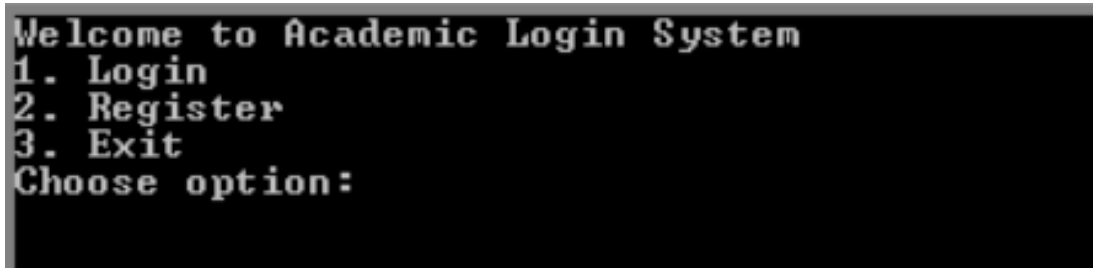
password storage and implementing additional safeguards would further enhance the system's protection.

#### 7. Potential Improvements:

To enhance our program, several key improvements can be implemented. First, adding encryption for password storage ensures secure handling of user credentials. Extending functionalities for students, such as the ability to view their scores and personal details, will provide a more comprehensive and personalized user experience. Input validation can be introduced to prevent invalid or malicious data, safeguarding the program from security vulnerabilities. Additionally, optimizing file search algorithms will improve performance, especially with large datasets, by reducing retrieval time. Finally, implementing an automatic data backup system will ensure the reliability of the data, protecting it from potential loss due to system failures. These improvements will make our program more secure, efficient, and user-friendly.

### V. Run simulation with EMU8086 software

#### 1. Main Menu



```
Welcome to Academic Login System
1. Login
2. Register
3. Exit
Choose option:
```

Fig 1: User Login Interface with Login, Register and Exit

## 2. Lecturer Menu

```
Welcome to Academic Login System
1. Login
2. Register
3. Exit
Choose option: 1
1. Student
2. Lecturer
Choice: 2
Enter username: manh
Enter password: manh
=== Lecturer Menu ===
1. View Student List
2. Add Student
3. Give Points
4. Remove Student
5. Logout
Choose option: _
```

Fig 2: Lecturer menu after login with 4 options as View Student List, Add Student, Give Point and Remove Student

## 3. Add Student

```
Choose option: 2
Enter student ID: 5
Enter username: 5
Operation successful!
```

Fig 3: Interface of the Add student after successfully add

## 4. View Student List

```
Choose option: 1
ID      Name      Points
28888   huy           0
78789   7bay          0
87997   hoa           0
98790   quo           0
78987   nam           0
13114   phuong        0
99999   hoai          0
93990   yui           0
65663   duy           0
83888   hoai          0
77129   viet          0
00109   hiep          0
00982   diep          0
89801   heop          0
79888   ruoi          0
99012   yen           0
79180   2hai          0
34718   tuy           0
77388   ti            0
99990   0             0
74727   yuti          0
```

Fig 4: Interface of the View Student with 20 students in the database.

## 5. Student menu

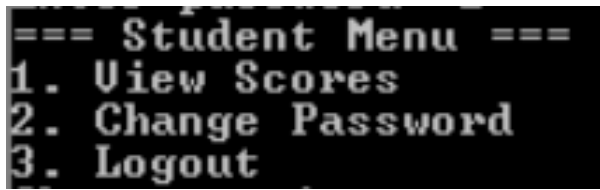


Fig 5: Interface of the Student menu after logging in with view scores that's from the lecturer given and change password option.

## Reference

- [1] "Assembler language". *High Level Assembler for z/OS & z/VM & z/VSE Language Reference Version 1 Release 6*. IBM. 2014 [1990]. SC26-4940-06.
- [2] Saxon, James A.; Plette, William S. (1962). *Programming the IBM 1401, a self-instructional programmed manual*. Englewood Cliffs, New Jersey, US: Prentice-Hall. LCCN 62-20615. (NB. Use of the term *assembly program*.)
- [3] Tanenbaum, A. S. (2006). *Structured Computer Organization*. Pearson Education.
- [4] Patterson, D. A., & Hennessy, J. L. (2014). *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufmann.
- [5] Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating System Concepts*. Wiley.
- [6] Bryant, R. E., & O'Hallaron, D. R. (2010). *Computer Systems: A Programmer's Perspective*. Pearson Education.
- [7] Mazidi, M. A., Mazidi, J. G., & Causey, R. D. (2010). *The x86 PC: Assembly Language, Design, and Interfacing*. Pearson Education.
- [8] Stallings, W. (2012). *Computer Organization and Architecture: Designing for Performance*. Pearson Education.
- [9] Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating System Concepts*. Wiley.
- [10] Tanenbaum, A. S. (2006). *Structured Computer Organization*. Pearson Education.
- [11] Bryant, R. E., & O'Hallaron, D. R. (2010). *Computer Systems: A Programmer's Perspective*. Pearson Education.

