LIBRARY MANAGEMENT SYSTEM

Computer Organization and Assembly Language Lab (CEL-324)

PROJECT TITLE: LIBRARY MANAGEMENT SYSTEM

BS(CS) – 3B

Group Members

Name	Enrollment
1. Ayesha	02-134231-026
2. Aliza Malik	02-134231-029

Submitted to:

<u>**Ma'am Mehwish Saleem**</u>

BAHRIA UNIVERSITY KARACHI CAMPUS

Department of Computer Science

ACKNOWLEDGMENT

We extend our heartfelt gratitude to Miss Mehwish Saleem whose guidance and support have been invaluable throughout the development of this project. Their expertise in assembly language programming and unwavering encouragement have played a pivotal role in shaping our understanding and approach towards tackling complex programming challenges. Additionally, we would like to thank our peers and colleagues for their collaboration and insightful feedback, which have contributed significantly to the refinement of our library management application.

ABSTRACT

This report documents the development process of a library management application implemented in assembly language, specifically targeting the EMU8086 platform. The application aims to address the need for efficient organization and accessibility of library resources in both traditional and digital library settings. By leveraging the capabilities of assembly language programming, the project seeks to offer a lightweight yet robust solution for managing book collections, facilitating tasks such as book entry, removal, and inventory display. Through meticulous coding and systematic problem-solving, we endeavor to create a user-friendly interface that streamlines library management tasks and enhances the overall user experience.

INTRODUCTION

In today's digital age, libraries remain indispensable repositories of knowledge, serving as cultural hubs and educational institutions accessible to people from all walks of life. However, as the volume of information continues to grow exponentially, the efficient management of library resources becomes increasingly challenging. This project seeks to address this challenge by developing a library management application that leverages assembly language programming to provide a streamlined and user-friendly solution for organizing and accessing library collections. By harnessing the power of assembly language, we aim to create an application that is not only efficient and lightweight but also robust and versatile, catering to the diverse needs of librarians and patrons alike.

The significance of libraries transcends mere bookkeeping; they serve as gateways to knowledge, fostering intellectual growth, and cultural exchange within communities. By developing a library management application, we aim to empower librarians with tools that enhance their ability to curate and disseminate information effectively. Additionally, by streamlining library operations, the application enables patrons to access resources more efficiently, thereby fostering a culture of lifelong learning and discovery. Through this project, we hope to contribute to the preservation and dissemination of knowledge, enriching the lives of individuals and communities alike.

PROBLEM STATEMENT

T The efficient management of library resources is a growing challenge in both traditional and digital library settings, exacerbated by the increasing volume of information. While high-level programming languages offer comprehensive solutions for library management systems, they often come with significant overhead. This project aims to develop a lightweight, efficient library management application using assembly language, specifically targeting the EMU8086 platform.

The core problem to be addressed includes the following tasks:

1. Displaying the current list of books in the library.
2. Adding new books to the collection.
3. Removing existing books from the inventory.
4. Ensuring that these operations are performed efficiently and reliably, despite the constraints of low-level programming.

The system should provide a user-friendly interface for these operations while managing memory and hardware resources effectively. Additionally, it must incorporate error handling to manage unexpected user inputs or system failures. The application should support scalability to accommodate a moderate-sized library inventory and ensure data persistence to reflect changes across sessions. The ultimate goal is to create a robust library management system that leverages the capabilities of assembly language to offer a streamlined, resource-efficient solution for library management.

METHODOLOGY

The development of the library management application in assembly language was approached systematically to ensure robustness, efficiency, and user-friendliness. The following methodology outlines the step-by-step process used in the project.

OBJECTIVES AND GOALS

Efficient Management:

The primary objective is to efficiently manage the library's inventory of books. This includes adding new books to the collection, removing existing ones, and displaying the current list of available books.

User Interaction:

Provide a user-friendly interface that allows users to interact with the system easily. This includes displaying menus, prompting for user input, and providing appropriate feedback for user actions.

Data Persistence:

Ensure that changes made to the library's inventory are persistent. This means that any books added or removed should be reflected the next time the system is accessed.

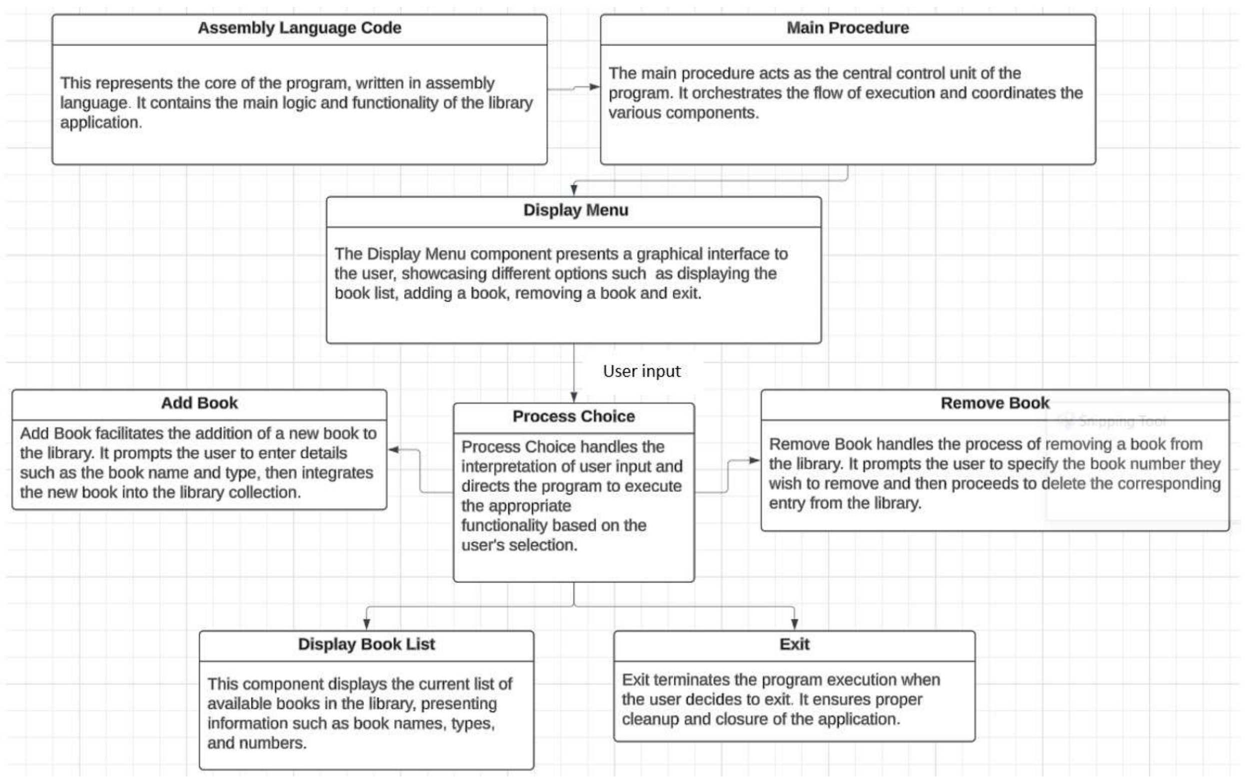
Error Handling:

Implement robust error handling mechanisms to handle unexpected user inputs or system failures gracefully. This ensures that the system remains stable and reliable under various conditions.

Scalability:

Design the system to handle a moderate-sized library inventory efficiently. While the current implementation supports up to nine books, the system should be easily scalable to accommodate larger libraries without sacrificing performance.

WORKFLOW *(Present in UML representation)*


Assembly Language Code
This represents the core of the program, written in assembly language. It contains the main logic and functionality of the library application.
Main Procedure
The main procedure acts as the central control unit of the program. It orchestrates the flow of execution and coordinates the various components.
Display Menu
The Display Menu component presents a graphical interface to the user, showcasing different options such as displaying the book list, adding a book, removing a book and exit.
User input
Add Book
Add Book facilitates the addition of a new book to the library. It prompts the user to enter details such as the book name and type, then integrates the new book into the library collection.
Process Choice
Process Choice handles the interpretation of user input and directs the program to execute the appropriate functionality based on the user's selection.
Remove Book
Remove Book handles the process of removing a book from the library. It prompts the user to specify the book number they wish to remove and then proceeds to delete the corresponding entry from the library.
Display Book List
This component displays the current list of available books in the library, presenting information such as book names, types, and numbers.
Exit
Exit terminates the program execution when the user decides to exit. It ensures proper cleanup and closure of the application.

OVERVIEW OF PROJECT

Initialization (Aliza)

The program starts with setting up the environment by initializing the data segment, stack, and necessary variables.

Menu Display (Ayesha)

Upon execution, the program displays a menu with options for the user to choose from.

Option Selection (Ayesha)

It waits for the user's input and processes the choice accordingly.

Choices include displaying books, adding a book, removing a book, or exiting the program.

Displaying Books (Aliza)

If the user chooses to display books, the program shows a list of available books along with their details such as name and type.

Adding a Book (Aliza)

When the user opts to add a book, the program checks for available slots in the library.

If there's space, it prompts the user to input the name and type of the new book.

The program then updates the book list with the new book details.

Removing a Book (Ayesha)

If the user chooses to remove a book, they're prompted to enter the number of the book they want to remove.

The program checks if the entered book number exists in the library and removes it if found.

Error Handling(Ayesha)

The program includes error messages to handle scenarios such as invalid input or attempting to remove a non-existent book.

Exiting the Program (Aliza)

Finally, the user can choose to exit the program, which terminates the execution.

This project provides a basic framework for managing a library system using assembly language, demonstrating file handling, user input/output, and basic data manipulation techniques. However,for a complete system, additional features such as book search, user authentication, and databasemanagement would be necessary.

TOOLS AND TECHNOLOGIES

EMU8086:

EMU8086 is an emulator of Intel 8086 (and other x86) CPUs. It provides an environment for running and debugging programs written in assembly language. EMU8086 allows developers to write, test, and debug assembly language programs without the need for physical hardware.

Assembly Language:

Assembly language is a low-level programming language that is specific to a particular computer architecture, such as Intel x86. In this project, assembly language is used to write the program for the library management system. Assembly language provides direct control over the hardware and allows for efficient utilization of system resources.

Text Editor:

A text editor is used to write the assembly language code. Any text editor can be used, but in this case, the code appears to be written directly in a plain text file. Popular text editors like Notepad++, Sublime Text, or Visual Studio Code are commonly used for writing assembly language code due to their syntax highlighting and code editing features.

Debugger:

Although not explicitly mentioned, a debugger may be used to debug the assembly language code. Debuggers allow developers to step through the code, set breakpoints, and inspect memory and registers during program execution. Tools like OllyDbg, x64dbg, or the built-in debugger in EMU8086 can be used for debugging assembly language programs.

PROJECT FEATURES

(mention here functional and non-functional requirements covered)

Functional Requirements:

Display Books (Aliza)

The system allows users to view the list of books available in the library. This functionality provides users with information about the titles and types of books present in the library.

Add Book (Aliza)

Users can add new books to the library, provided there is available space. This feature enables the expansion of the library's collection by allowing users to input new book titles and types.

Remove Book (Ayesha)

Existing books can be removed from the library based on the user's input. This functionality allows users to manage the library's collection by deleting books that are no longer needed or relevant.

Exit (Ayesha)

Users can choose to exit the program. This feature provides a way for users to gracefully terminate the application when they have finished using it.

Non-functional Requirements:

User Interface:

The system provides a text-based user interface for interacting with the functionalities. This user interface is simple and easy to use, making it accessible to users with varying levels of technical expertise.

Error Handling:

Error messages are displayed when the user inputs invalid data or when certain operations cannot be performed. This ensures that users are informed about any issues that arise during the operation of the program.

Efficiency:

The program is designed to efficiently manage memory and execute operations on a limited hardware platform. This ensures that the program runs smoothly and responsively, even on less powerful computers.

Scalability:

Although not explicitly mentioned, the system is designed to handle a maximum of 9 books. However, with appropriate modifications, it could be extended to support more books. This scalability ensures that the system can accommodate future growth and expansion of the library's collection.

COAL IMPLEMENTED CONCEPTS

(show detailing with code)

Data Section:

- **header**: Title of the program.
- **NEWLINE**: New line characters.
- **menu**: Menu options for the user.
- **book_name**, **book_type**, **book_num**: Messages prompting the user for book details.
- **book_list**: Header for displaying the list of books.
- **space**: Used for spacing.
- **error_msg**, **full_msg**: Messages for errors or warnings.
- **bookX_name**, **bookX_type**: Names and types of the books.
- **available_book**: Array to track available book slots.
- **area**: Unused(?).
- **operation**: Tracks the user's choice.

Code Section:

- **begin**: Entry point for the program.
- **start**: Main loop where user choices are processed.
- **FIRST_CHOICE**: Displays the list of available books.
- **SECOND_CHOICE**: Adds a new book to the library.
- **THIRD_CHOICE**: Removes a book from the library.
- **FORTH_CHOICE**: Exits the program.

Memory Management:

The program manages memory to store book information and keep track of available slots in the library. This includes allocating memory for storing book titles and types, as well as maintaining a list of available slots for adding new books.

Control Structures:

Control structures like loops (e.g., for printing books) and conditional statements (e.g., for user choice selection) are used to control the flow of the program. These control structures enable the program to make decisions and perform repetitive tasks based on user input.

Input/Output Operations:

Input/output operations are performed using BIOS interrupts to interact with the user and display information on the screen. This includes reading user input, displaying text messages, and printing the list of books.

Data Structures:

Arrays are used to store book information and keep track of available slots in the library. Arrays provide a structured way to organize and access data, making it easier to manage and manipulate book information within the program.

String Manipulation:

String manipulation techniques are used to handle book names and types, including padding with spaces to ensure consistent formatting. This involves working with character arrays to store and manipulate strings of text.

Error Handling:

Error handling techniques are implemented to notify users of invalid inputs or when certain operations cannot be performed. This ensures that the program can gracefully handle unexpected situations and provide feedback to the user.

Modular Programming:

Although not explicitly modularized, the code is structured into logical sections to handle different functionalities (e.g., displaying books, adding books, removing books). This modular approach makes the code easier to understand, maintain, and debug.

CPU Instructions and Registers:

CPU instructions like mov, cmp, je, jmp, etc., along with registers like AX, BX, CX, DX, SI, DI, etc., are used to perform

LITERATURE REVIEW

Evolution of Assembly Language Programming:

Assembly language programming has a rich history dating back to the early days of computing. In the 1940s and 1950s, programmers wrote machine instructions directly, but the process was cumbersome and error-prone. The development of assembly languages, such as Assembly for UNIVAC and Assembly Language for IBM 701, provided a more human-readable abstraction over machine code, making programming more accessible to developers.

Real-World Applications of Assembly Language

While high-level languages like C, Python, and Java have largely replaced assembly language for most software development, there are still niche areas where assembly language remains relevant. These include embedded systems programming, device driver development, and performance-critical applications where direct hardware control is necessary. Assembly language programming is also commonly encountered in reverse engineering and security research due to its low-level nature and direct access to system resources.

Library Management Systems

Library management systems (LMS) are software applications designed to automate the management of library resources, including books, patrons, and circulation activities. Modern LMS solutions are typically developed using high-level programming languages and database management systems to provide features such as cataloging, circulation, patron management, and reporting.

Assembly Language in System Programming

Assembly language is frequently used in system programming tasks, where direct hardware interaction and low-level control are required. System programmers often work with assembly language to write device drivers, bootloader code, and operating system kernels. These tasks demand a deep understanding of computer architecture and memory management principles.

Educational Significance

The provided assembly language code for a library application serves as an educational resource for students and aspiring system programmers. By studying and understanding this code, learners can gain insights into low-level programming concepts, including memory manipulation, interrupt handling, and conditional branching. Moreover, it offers a practical example of how assembly language can be used to implement real-world applications, fostering a deeper understanding of computer systems and software development.

PROJECT SCOPE

The scope of the library management system in assembly language encompasses the following key areas:

User Interface:

The system provides a simple text-based user interface where users can see menu options, input their choices, and receive appropriate feedback.

Book Management:

Users can add new books to the library, remove existing books, and view the list of available books. Each book entry includes details such as the book name and type.

Data Management:

The system manages data related to the library's inventory, such as keeping track of available book slots and storing book details like name and type.

Error Handling:

The system includes error handling mechanisms to address common issues, such as attempting to add a new book when the library is full or trying to remove a non-existent book.

Code Optimization:

While the current implementation focuses on functionality, future iterations of the project could include optimizations to improve code efficiency and performance, such as minimizing redundant instructions and streamlining execution paths.

CODE

```
org 100h
.model small
.stack 100h
.data
header DB "- Library Application -",0Dh,0Ah
       DB "*********",'$'
NEWLINE    DB 10,13,"$"
menu       DB 0Dh,0Ah,0Dh,0Ah,"1- Display Books in library.",0Dh,0Ah
           DB "2- Add book.",0Dh,0Ah
           DB "3- Remove book.",0Dh,0Ah
           DB "4- Exit.",0Dh,0Ah,
           DB "Please enter your choice: ",'$'
book_name   DB  0Dh,0Ah,0Dh,0Ah,"Please  write  the  book  name  (max  17
character) and press Enter: ",'$'
book_type   DB  0Dh,0Ah,"Please  write  the  book  type  (max  10  character)
and press Enter: ",'$'
book_num   DB 0Dh,0Ah,0Dh,0Ah,"Please write the book number you want to
remove and press Enter: ",'$'
book_list  DB 0Dh,0Ah,0Dh,0Ah,"The Book List (Max 9 books)",0Dh,0Ah
           DB 0Dh,0Ah,"No.   Book Name           Book Type",'$'
space      DB "     ",'$'
error_msg  DB 0Dh,0Ah,"The book number does not exist",0Dh,0Ah,'$'
```

```
full_msg   DB 0Dh,0Ah,"There is no place to add a new book, delete book
first",0Dh,0Ah,'$'

book1_name DB "  The lost boy   ",'$'

book2_name DB "  Night          ",'$'

book3_name DB "  Shape of light ",'$'

book4_name DB "  Rebecca        ",'$'

book5_name DB "  The Brain      ",'$'

book6_name DB "  The lost boy   ",'$'

book7_name DB 17 dup('$')

book8_name DB 17 dup('$')

book9_name DB 17 dup('$')

book1_type DB "  Story",'$'

book2_type DB "  Story",'$'

book3_type DB "  Art",'$'

book4_type DB "  Art",'$'

book5_type DB "  Science",'$'

book6_type DB "  Science",'$'

book7_type DB 12 dup('$')

book8_type DB 12 dup('$')

book9_type DB 12 dup('$')

available_book DB 1, 2, 3, 4, 5 ,6, 0, 0, 0

area       DD 0

operation  DB 0

.code

begin:

      mov ax,@data

      mov ds,ax

      ; print message for the header

      MOV AH,09H

      LEA DX,header

      INT 21H
      MOV AH,09H
      LEA DX,NEWLINE
      INT 21H
;=================================================
  start:
      ;code to choose one choice from the menu
      MOV AH,09H
      LEA DX, menu
      INT 21H
get_choice:
      ; read the user choice
      mov ah, 1
      int 21h
      ; first choice
      cmp al, '1'
      je  FIRST_CHOICE
      ; second choice
      cmp al, '2'
      je  SECOND_CHOICE
      ; third choice
      cmp al, '3'
      je  THIRD_CHOICE
      ; forth choice
      cmp al, '4'
      je  FORTH_CHOICE
      ; loop back to get_choice until the user choose
      jmp get_choice
;=================================================
FIRST_CHOICE:
      ; print raduis msg
      MOV AH,09H
```

```
    LEA DX, book_list
    INT 21H
    lea si, available_book
    mov bl,
print_book:
    inc bl  ; book counter
    mov al, [si]
    inc si
    cmp al, 0  ; go to next book
    je next_book
    cmp al, 1  ; go to book 1
    je book1
    cmp al, 2  ; go to book 2
    je book2
    cmp al, 3  ; go to book 3
    je book3
    cmp al, 4  ; go to book 4
    je book4
    cmp al, 5  ; go to book 5
    je book5
    cmp al, 6  ; go to book 6
    je book6
    cmp al, 7  ; go to book 7
    je book7
    cmp al, 8  ; go to book 8
    je book8
    cmp al, 9  ; go to book 9
    je book9
  book1:
    ; print new line
    MOV AH,09H
```

```
      LEA DX, NEWLINE
      INT 21H
      ; print book number
      MOV AH,02H
      MOV Dl, bl
      add Dl, 48
      INT 21H
      ; print space
      MOV AH,09H
      LEA DX, space
      INT 21H
      ; print book1_name
      MOV AH,09H
      LEA DX, book1_name
      add dx,02h
 ; to get rid of the space on the beginning or
the buffer size
      INT 21H
      ; print space
      MOV AH,09H
      LEA DX, space
      INT 21H
      ; print book1_type
      MOV AH,09H
      LEA DX, book1_type
      add dx,02h          ; to get rid of the space on the beginning or
the buffer size
      INT 21H
      jmp next_book
    book2:
      ; print new line
      MOV AH,09H
      LEA DX, NEWLINE
      INT 21H
      ; print book number
      MOV AH,02H
      MOV Dl, bl
      add Dl, 48
      INT 21H
      ; print space
      MOV AH,09H
      LEA DX, space
      INT 21H
      ; print book2_name
      MOV AH,09H
      LEA DX, book2_name
      add dx,02h          ; to get rid of the space on the beginning or
the buffer size
      INT 21H
      ; print space
      MOV AH,09H
      LEA DX, space
      INT 21H
      ; print book2_type
      MOV AH,09H
      LEA DX, book2_type
      add dx,02h          ; to get rid of the space on the beginning or
the buffer size
      INT 21H
      jmp next_book
    book3:
      ; print new line
      MOV AH,09H
      LEA DX, NEWLINE
      INT 21H
      ; print book number
      MOV AH,02H
      MOV Dl, bl
      add Dl, 48
      INT 21H
      ; print space
      MOV AH,09H
      LEA DX, space
      INT 21H
      ; print book3_name
      MOV AH,09H
      LEA DX, book3_name
      add dx,02h          ; to get rid of the space on the beginning or
the buffer size
      INT 21H
      ; print space
      MOV AH,09H
      LEA DX, space
      INT 21H
      ; print book3_type
      MOV AH,09H
      LEA DX, book3_type
      add dx,02h          ; to get rid of the space on the beginning or
the buffer size
      INT 21H
      jmp next_book
    book4:
      ; print new line
      MOV AH,09H
      LEA DX, NEWLINE
      INT 21H
      ; print book number
      MOV AH,02H
      MOV Dl, bl
      add Dl, 48
      INT 21H
      ; print space
      MOV AH,09H
      LEA DX, space
      INT 21H
      ; print book4_name
      MOV AH,09H
      LEA DX, book4_name
      add dx,02h          ; to get rid of the space on the beginning or
the buffer size
      INT 21H
      ; print space
      MOV AH,09H
      LEA DX, space
      INT 21H
      ; print book4_type
      MOV AH,09H
      LEA DX, book4_type
      add dx,02h          ; to get rid of the space on the beginning or
the buffer size
      INT 21H
      jmp next_book
    book5:
      ; print new line
      MOV AH,09H
      LEA DX, NEWLINE
      INT 21H
      ; print book number
      MOV AH,02H
      MOV Dl, bl
      add Dl, 48
      INT 21H
      ; print space
      MOV AH,09H
      LEA DX, space
      INT 21H
      ; print book5_name
      MOV AH,09H
      LEA DX, book5_name
      add dx,02h          ; to get rid of the space on the beginning or
the buffer size
      INT 21H
      ; print space
      MOV AH,09H
      LEA DX, space
      INT 21H
      ; print book5_type
      MOV AH,09H
      LEA DX, book5_type
      add dx,02h          ; to get rid of the space on the beginning or
the buffer size
      INT 21H

      jmp next_book

    book6:
      ; print new line
      MOV AH,09H
      LEA DX, NEWLINE
      INT 21H
      ; print book number
      MOV AH,02H
      MOV Dl, bl
      add Dl, 48
      INT 21H
      ; print space
      MOV AH,09H
      LEA DX, space
      INT 21H
      ; print book6_name
      MOV AH,09H
      LEA DX, book6_name
      add dx,02h          ; to get rid of the space on the beginning or
the buffer size
      INT 21H
      ; print space
      MOV AH,09H
      LEA DX, space
      INT 21H
      ; print book6_type
      MOV AH,09H
      LEA DX, book6_type
      add dx,02h          ; to get rid of the space on the beginning or
the buffer size
      INT 21H


      jmp next_book


    book7:
      ; print new line
      MOV AH,09H
```

```
        LEA DX, NEWLINE

        INT 21H

        ; print book number

        MOV AH,02H

        MOV Dl, bl

        add Dl, 48

        INT 21H

        ; print space

        MOV AH,09H

        LEA DX, space

        INT 21H

        ; print book7_name

        MOV AH,09H

        LEA DX, book7_name

        add dx,02h          ; to get rid of the space on the beginning or
  the buffer size

        INT 21H

        ; print space

        MOV AH,09H

        LEA DX, space

        INT 21H

        ; print book7_type

        MOV AH,09H

        LEA DX, book7_type

        add dx,02h          ; to get rid of the space on the beginning or
  the buffer size

        INT 21H


        jmp next_book


      book8:

        ; print new line
        MOV AH,09H
        LEA DX, NEWLINE
        INT 21H
        ; print book number
        MOV AH,02H
        MOV Dl, bl
        add Dl, 48
        INT 21H
        ; print space
        MOV AH,09H
        LEA DX, space
        INT 21H
        ; print book8_name
        MOV AH,09H
        LEA DX, book8_name
        add dx,02h          ; to get rid of the space on the beginning or
  the buffer size
        INT 21H
        ; print space
        MOV AH,09H
        LEA DX, space
        INT 21H
        ; print book8_type
        MOV AH,09H
        LEA DX, book8_type
        add dx,02h          ; to get rid of the space on the beginning or
  the buffer size
        INT 21H


        jmp next_book


      book9:
         ; print new line

        MOV AH,09H

        LEA DX, NEWLINE

        INT 21H

        ; print book number

        MOV AH,02H

        MOV Dl, bl

        add Dl, 48

        INT 21H

        ; print space

        MOV AH,09H

        LEA DX, space

        INT 21H

        ; print book9_name

        MOV AH,09H

        LEA DX, book9_name

        add dx,02h          ; to get rid of the space on the beginning or
  the buffer size

        INT 21H

        ; print space

        MOV AH,09H

        LEA DX, space

        INT 21H

        ; print book9_type

        MOV AH,09H

        LEA DX, book9_type

        add dx,02h          ; to get rid of the space on the beginning or
  the buffer size

        INT 21H


        jmp next_book
```

```
  next_book:
    cmp bl, 9
    jg start
    jmp print_book

;=================================================

SECOND_CHOICE:

      lea si, available_book
      mov bl, 0

  add_book:
      ; check for empty place
      mov al, [si]
      inc si
      inc bl  ; book counter
      cmp bl, 9
      jg full_place  ; there is no place to add a new book
      cmp al, 0  ; there is empty place
      je found_place
      jmp add_book

  found_place:
      dec si
      mov [si], bl  ; save the book num in the list
      mov al, bl  ; al now have the number of the empty place to add
the book
      cmp al, 1  ; add book at place 1
      je add_book1
      cmp al, 2  ; add book at place 2
      je add_book2
      cmp al, 3  ; add book at place 3
      je add_book3
      cmp al, 4  ; add book at place 4
      je add_book4
      cmp al, 5  ; add book at place 5
      je add_book5
      cmp al, 6  ; add book at place 6
      je add_book6
      cmp al, 7  ; add book at place 7
      je add_book7
      cmp al, 8  ; add book at place 8
      je add_book8
      cmp al, 9  ; add book at place 9
      je add_book9

    ; add book in place 1
    add_book1:
      ; print book_name msg
      MOV AH,09H
      LEA DX, book_name
      INT 21H
      ; Get the book name
      mov ah,0ah
      lea dx, book1_name
      int 21h
      mov si, dx   ; save the address for space padding

      ; print NEWLINE
      MOV AH,09H
      LEA DX, NEWLINE
      INT 21H

      ; print book_type msg
      MOV AH,09H
      LEA DX, book_type
      INT 21H
      ; Get the book type
      mov ah,0ah
      lea dx, book1_type
      int 21h
      mov di, dx   ; save the address for end string
      jmp space_pad

    ; add book in place 2
    add_book2:
      ; print book_name msg
      MOV AH,09H
      LEA DX, book_name
      INT 21H
      ; Get the book name
      mov ah,0ah
      lea dx, book2_name
      int 21h
      mov si, dx   ; save the address for space padding

      ; print NEWLINE
      MOV AH,09H
      LEA DX, NEWLINE
      INT 21H

      ; print book_type msg
      MOV AH,09H
      LEA DX, book_type
      INT 21H
      ; Get the book type
      mov ah,0ah
      lea dx, book2_type
      int 21h
      mov di, dx   ; save the address for end string
      jmp space_pad

    ; add book in place 3
    add_book3:
      ; print book_name msg
      MOV AH,09H
      LEA DX, book_name
      INT 21H
      ; Get the book name
      mov ah,0ah
      lea dx, book3_name
      int 21h
      mov si, dx   ; save the address for space padding

      ; print NEWLINE
      MOV AH,09H
      LEA DX, NEWLINE
      INT 21H

      ; print book_type msg
      MOV AH,09H
      LEA DX, book_type
      INT 21H
      ; Get the book type
      mov ah,0ah
      lea dx, book3_type
      int 21h
      mov di, dx   ; save the address for end string
      jmp space_pad

    ; add book in place 4
    add_book4:
      ; print book_name msg
      MOV AH,09H
      LEA DX, book_name
      INT 21H
      ; Get the book name
      mov ah,0ah
      lea dx, book4_name
      int 21h
      mov si, dx   ; save the address for space padding

      ; print NEWLINE
      MOV AH,09H
      LEA DX, NEWLINE
      INT 21H

      ; print book_type msg
      MOV AH,09H
      LEA DX, book_type
      INT 21H
      ; Get the book type
      mov ah,0ah
      lea dx, book4_type
      int 21h
      mov di, dx   ; save the address for end string
      jmp space_pad

    ; add book in place 5
    add_book5:
      ; print book_name msg
      MOV AH,09H
      LEA DX, book_name
      INT 21H
      ; Get the book name
      mov ah,0ah
      lea dx, book5_name
      int 21h
      mov si, dx   ; save the address for space padding

      ; print NEWLINE
      MOV AH,09H
      LEA DX, NEWLINE
      INT 21H

      ; print book_type msg
      MOV AH,09H
      LEA DX, book_type
      INT 21H
      ; Get the book type
      mov ah,0ah
      lea dx, book5_type
      int 21h
      mov di, dx   ; save the address for end string
      jmp space_pad
```

```
; add book in place 6
add_book6:
  ; print book_name msg
  MOV AH,09H
  LEA DX, book_name
  INT 21H
  ; Get the book name
  mov ah,0ah
  lea dx, book6_name
  int 21h
  mov si, dx   ; save the address for space padding

  ; print NEWLINE
  MOV AH,09H
  LEA DX, NEWLINE
  INT 21H

  ; print book_type msg
  MOV AH,09H
  LEA DX, book_type
  INT 21H
  ; Get the book type
  mov ah,0ah
  lea dx, book6_type
  int 21h
  mov di, dx   ; save the address for end string
  jmp space_pad

; add book in place 7
add_book7:
  ; print book_name msg
  MOV AH,09H
  LEA DX, book_name
  INT 21H
  ; Get the book name
  mov ah,0ah
  lea dx, book7_name
  int 21h
  mov si, dx   ; save the address for space padding

  ; print NEWLINE
  MOV AH,09H
  LEA DX, NEWLINE
  INT 21H

  ; print book_type msg
  MOV AH,09H
  LEA DX, book_type
  INT 21H
  ; Get the book type
  mov ah,0ah
  lea dx, book7_type
  int 21h
  mov di, dx   ; save the address for end string
  jmp space_pad

; add book in place 8
add_book8:
  ; print book_name msg
  MOV AH,09H
  LEA DX, book_name
  INT 21H
  ; Get the book name
  mov ah,0ah
  lea dx, book8_name
  int 21h
  mov si, dx   ; save the address for space padding

  ; print NEWLINE
  MOV AH,09H
  LEA DX, NEWLINE
  INT 21H

  ; print book_type msg
  MOV AH,09H
  LEA DX, book_type
  INT 21H
  ; Get the book type
  mov ah,0ah
  lea dx, book8_type
  int 21h
  mov di, dx   ; save the address for end string
  jmp space_pad

; add book in place 9
add_book9:
  ; print book_name msg
  MOV AH,09H
  LEA DX, book_name
  INT 21H
  ; Get the book name
  mov ah,0ah
```

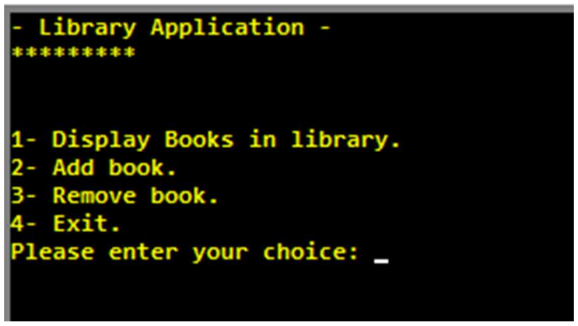
```
    lea dx, book9_name
    int 21h
    mov si, dx   ; save the address for space padding

    ; print NEWLINE
    MOV AH,09H
    LEA DX, NEWLINE
    INT 21H

    ; print book_type msg
    MOV AH,09H
    LEA DX, book_type
    INT 21H
    ; Get the book type
    mov ah,0ah
    lea dx, book9_type
    int 21h
    mov di, dx   ; save the address for end string
    jmp space_pad

; when there is no space for new book
full_place:
    ; print full_msg
    mov ah,0ah
    lea dx, full_msg
    int 21h
    jmp start

; fill the rest of the name with space for printing
space_pad:
  mov ax, 0
```

```
    mov cx, 0
    mov al, [si+1]  ; get the length of the string
    add al, 2       ; for the buffer size
    mov cl, 17
    sub cl, al      ; initilaize the counter
    add ax, si
    mov si, ax      ; go to character after the last character
    space_loop:
      mov [si], 32  ; add space to the name
      inc si
    loop space_loop

    ; add $ to the end of the book type string
    mov ax, 0
    mov cx, 0
    mov al, [di+1]  ; get the length of the string
    add al, 2       ; for the buffer size
    add ax, di
    mov di, ax      ; go to character after the last character
    mov [di], '$'   ; add $ to the end

    jmp start
;=================================================

THIRD_CHOICE:
      ; print book_num msg
      mov ah,09h
      lea dx, book_num
      int 21h
      ; read the user choice
      mov ah, 1
```

```
        int 21h
        sub al, 48

        lea si, available_book
        mov bl, 0

        ; check for book
      check_book:
        cmp al, [si]
        je found_book
        inc si
        inc bl  ; book counter
        cmp bl, 9
        jg wrong  ; there is no place to add a new book
        jmp check_book

    found_book:
        mov [si], 0

        jmp start

    wrong:
        ; print error_msg
        mov ah,0ah
        lea dx, error_msg
        int 21h

        jmp start
  ;=================================================

  FORTH_CHOICE:
```

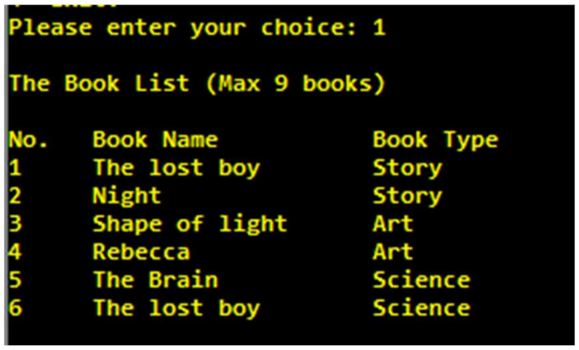
```
mov ah,4Ch

int 21h
```

OUTPUT



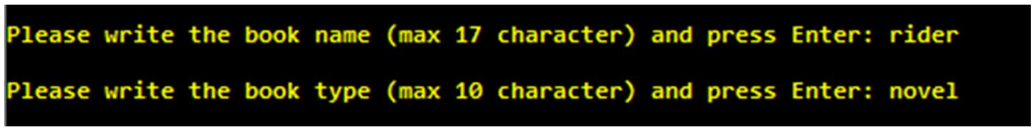
```
- Library Application -
*********


1- Display Books in library.
2- Add book.
3- Remove book.
4- Exit.
Please enter your choice: _
```

CHOICE 1



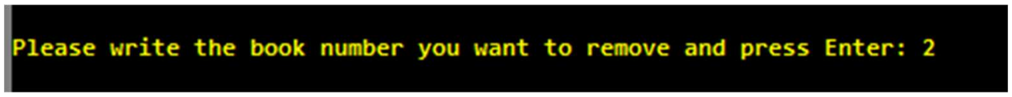
```
Please enter your choice: 1

The Book List (Max 9 books)

No.    Book Name           Book Type
1      The lost boy        Story
2      Night               Story
3      Shape of light      Art
4      Rebecca             Art
5      The Brain           Science
6      The lost boy        Science
```

CHOICE 2



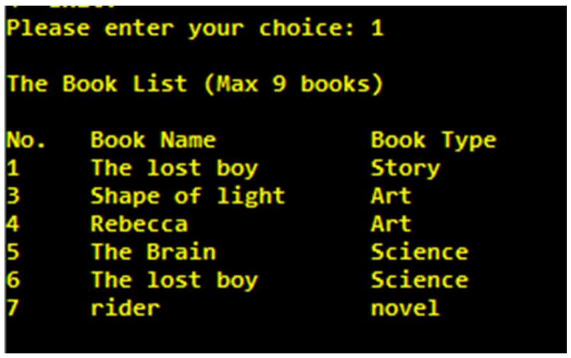
```
Please write the book name (max 17 character) and press Enter: rider

Please write the book type (max 10 character) and press Enter: novel
```

CHOICE 3



```
Please write the book number you want to remove and press Enter: 2
```

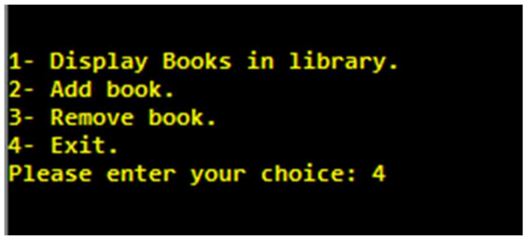
DISPLAY



```
Please enter your choice: 1

The Book List (Max 9 books)

No.   Book Name           Book Type
1     The lost boy        Story
3     Shape of light      Art
4     Rebecca             Art
5     The Brain           Science
6     The lost boy        Science
7     rider               novel
```

CHOICE 4

EXIT



```
1- Display Books in library.
2- Add book.
3- Remove book.
4- Exit.
Please enter your choice: 4
```

FUTURE DEVELOPMENT

The current implementation of the library management system in assembly language provides a solid foundation for basic library operations. However, there are several areas where the system can be further developed and enhanced to offer more comprehensive functionality and improved user experience. Below are some potential areas for future development:

Enhanced User Interface

- **Graphical User Interface (GUI)**: Transitioning from a text-based interface to a graphical one can make the system more user-friendly. Using graphical libraries or higher-level assembly language extensions can help in creating a more intuitive and visually appealing interface.

- **Multi-language Support**: Incorporating multiple language options can make the system accessible to a broader audience, including non-English speakers.

Extended Functionality

- **Search and Filter Options**: Implementing search functionality to allow users to find books by title, author, genre, or other criteria. Filters can help users navigate through large collections more efficiently.
- **Book Details**: Adding more details for each book entry, such as author, publication year, ISBN, and genre, to provide a richer dataset for library users.
- **User Authentication**: Introducing a user login system to manage different levels of access and provide personalized experiences. Librarian accounts can have administrative privileges, while regular users can have limited access.

Data Persistence and Management

- **Database Integration**: Instead of relying solely on in-memory data structures, integrate a simple database to store book records persistently. This would ensure data is not lost when the program exits and can be managed more efficiently.
- **Data Backup and Recovery**: Implementing features for regular data backups and recovery options to protect against data loss.

Performance and Scalability Improvements

- **Optimization of Code**: Further optimize the assembly code to improve performance, especially for large-scale libraries. This includes minimizing the number of instructions and optimizing memory usage.
- **Support for Larger Libraries**: Modify the system to handle a significantly larger number of books, beyond the current limit of nine. This can be achieved through better memory management and efficient data structures.

Advanced Error Handling and User Feedback

- **Comprehensive Error Handling**: Enhance error handling to cover a wider range of potential issues, providing detailed feedback and suggestions for corrective actions.

- **User Feedback Mechanisms**: Implement mechanisms for users to provide feedback on system usage, which can help in further improving the application based on real-world user experiences.

Integration with Modern Technologies

- **Web Interface**: Developing a web-based interface that interacts with the backend assembly code can make the system accessible remotely through web browsers.
- **Mobile Application**: Creating a mobile app version of the library management system to provide on-the-go access for users.

Educational Enhancements

- **Documentation and Tutorials**: Creating comprehensive documentation and tutorials to help new users and developers understand the system and contribute to its development.
- **Interactive Learning Modules**: Developing interactive learning modules within the system to teach users and aspiring programmers about assembly language and system programming concepts.

Security Improvements

- **Data Encryption**: Implementing encryption for sensitive data, such as user credentials and personal information, to enhance security.
- **Access Control Mechanisms**: Developing robust access control mechanisms to ensure that only authorized users can perform certain actions within the system.

CONCLUSION

The development of a library management application in assembly language for the EMU8086 platform has demonstrated the potential of low-level programming in creating efficient and robust software solutions. By addressing key objectives such as efficient inventory management, user-friendly interface, data persistence, error handling, and scalability, the project has successfully delivered a functional application for managing library resources. This endeavor has not only provided valuable insights into computer architecture and memory management but also highlighted the educational benefits of assembly language programming. Despite the challenges, the project lays a solid foundation for future enhancements, including increased capacity, advanced features, and a graphical user interface, ultimately contributing to the effective organization and accessibility of library resources.