A · P · U
ASIA PACIFIC UNIVERSITY
OF TECHNOLOGY & INNOVATION

Individual Assignment

TECHNOLOGY PARK MALAYSIA

Computer System Low-Level Techniques

Library Management System in Assembly Language (TASM)

Name: Ahmed Bin Faisal

TP Number: TP078706

Intake Code: APU2F2411CS

Subject: CSLLT

Project Title: Library Management System

Date: 25/07/2025

Table of Contents

1- Comprehensive Study of Assembly Language

Assembly language can be described as low level programming scheme that exists with the purpose of allowing a computer's hardware features to be directly accessed using symbolic codes as well as using mnemonics. Unlike high level programming languages like Java or Python, assembly language gives programmers total control of processor operations, memory operations, as well as input/output operations (Hyde, 2003). Assembly language is architecture specific, meaning that software written for a given processor architecture (e.g., Intel x86) cannot be run on others (e.g., ARM) without modifications. Because of this very close association with hardware, assembly programs run with extreme speed and efficiency, though they are often harder to code and debug.

Turbo Assembler, developed by Borland, is an assembly language program designed specifically for the x86 microprocessor family.
The program makes it easy for programmers to write, assemble, and link assembly language programs that ultimately yield executable software products. TASM has extensive use in the academic environment for teaching students fundamental concepts of system development, including memory addressing, stack manipulation, and interrupt handling.

2. Research and Exploration: Convergence in Cybersecurity

The Need for Assembly Language in cybersecurity Fields as well as digital Forensics for computer security as a career, an understanding of an assembly language is not only handy but necessary as well (Eagle, 2011). Most low-level attacks as well as vulnerabilities are authored or discovered at an assembly level, thus a good level of comprehension of machine code is a computer security specialist's toolset for tracking, decompiling, as well as turning off malicious activity software.

Key Use Cases:
Malware Analysis
Malware often operates at a low level in an effort to evade discovery. Analysts disassemble the binaries with assistance from software like IDA Pro, Ghidra, or Radare2 to understand a binary's behavior without having to run it (Skoudis & Liston, 2006).

Reverse Engineering
When source code is unavailable (e.g., proprietary or malicious software), engineers reverse engineer executables to understand their logic and behavior (Wikipedia, 2025a). This process is almost entirely based on analyzing the disassembled assembly code.
Exploitation creation targeted towards finding vulnerabilities.
Such buffer overflow vulnerabilities require strong control flow and memory managemen t skills, based on thorough knowledge of the CPU instruction set and stack manipulation methods.

Forensics

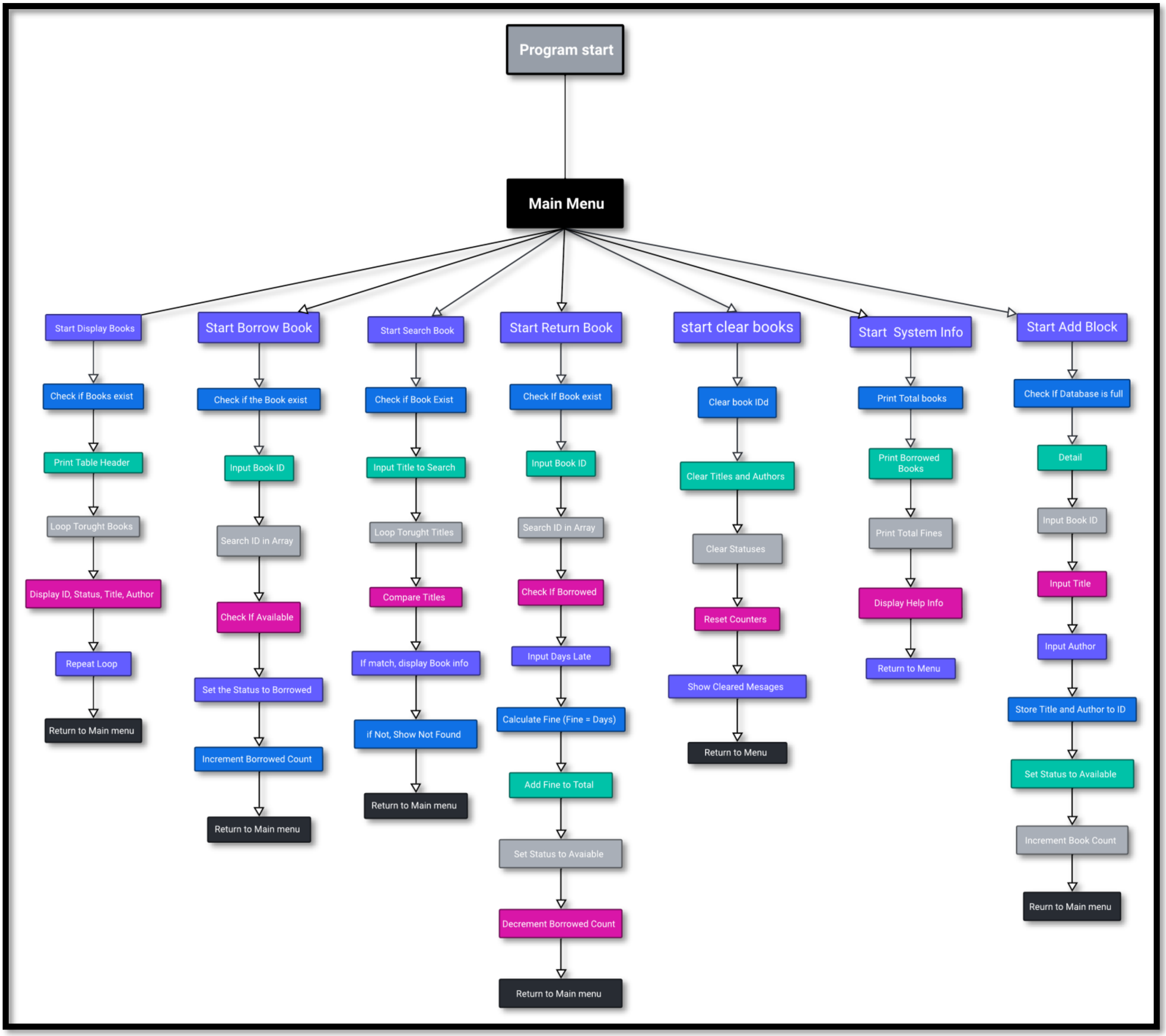
Forensic examiners perform memory dump, binary log, or residual data analysis on hard drives and RAM. These processes often require raw binary data interpretation followed by converting it into understandable assembly instructions.

Example:

Shellcode used in buffer overflow exploits is typically written in assembly because:

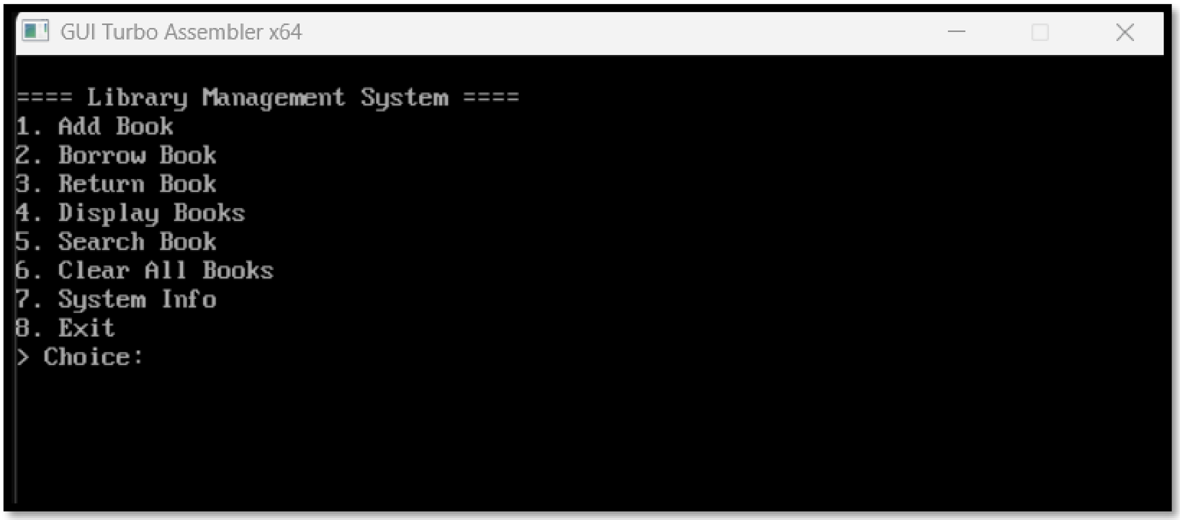
- It needs to be compact and efficient.
- It interacts directly with OS level system calls.
- It bypasses certain security mechanisms like ASLR or DEP using low level control.

3. System Design: Flowchart



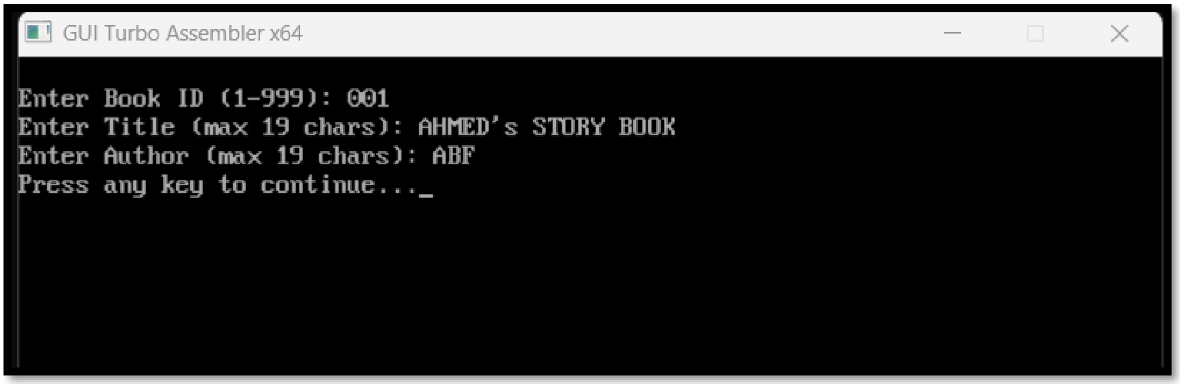
4. Screenshots of Working System

Program Main Menu



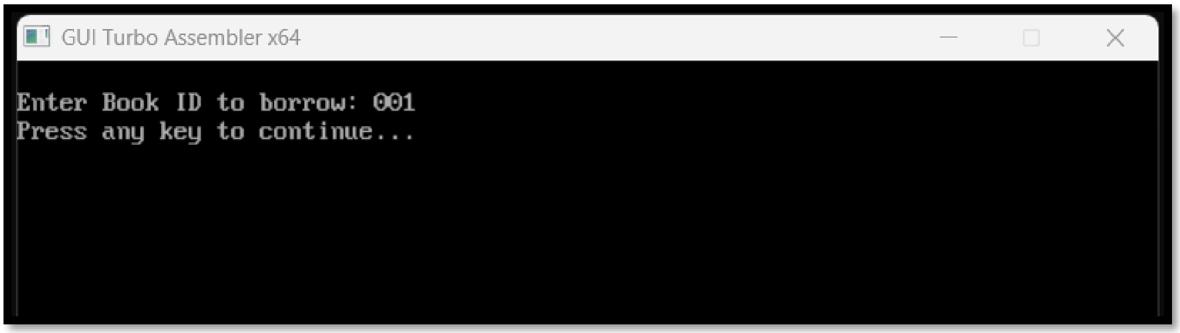
```
==== Library Management System ====
1. Add Book
2. Borrow Book
3. Return Book
4. Display Books
5. Search Book
6. Clear All Books
7. System Info
8. Exit
> Choice:
```

Adding Book Entry



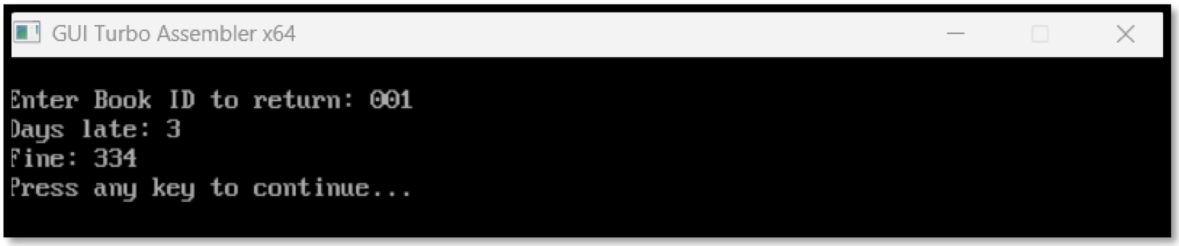
```
Enter Book ID (1-999): 001
Enter Title (max 19 chars): AHMED's STORY BOOK
Enter Author (max 19 chars): ABF
Press any key to continue..._
```

Borrowing and Returning a Book
Borrowing



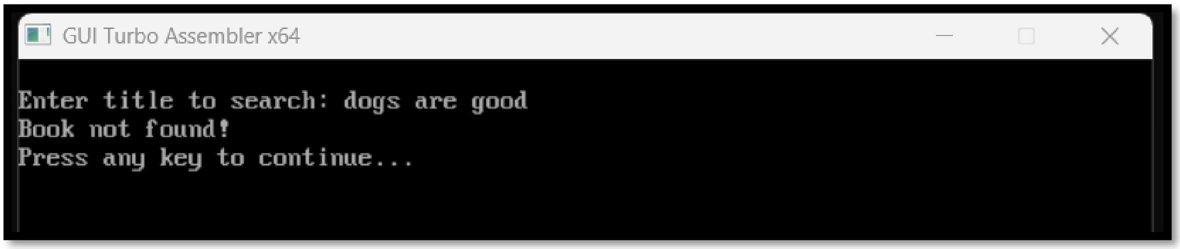
```
Enter Book ID to borrow: 001
Press any key to continue...
```

Returning



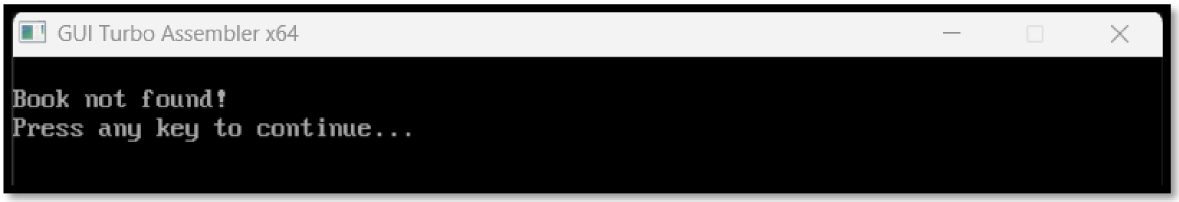
```
Enter Book ID to return: 001
Days late: 3
Fine: 334
Press any key to continue...
```

Searching a Book



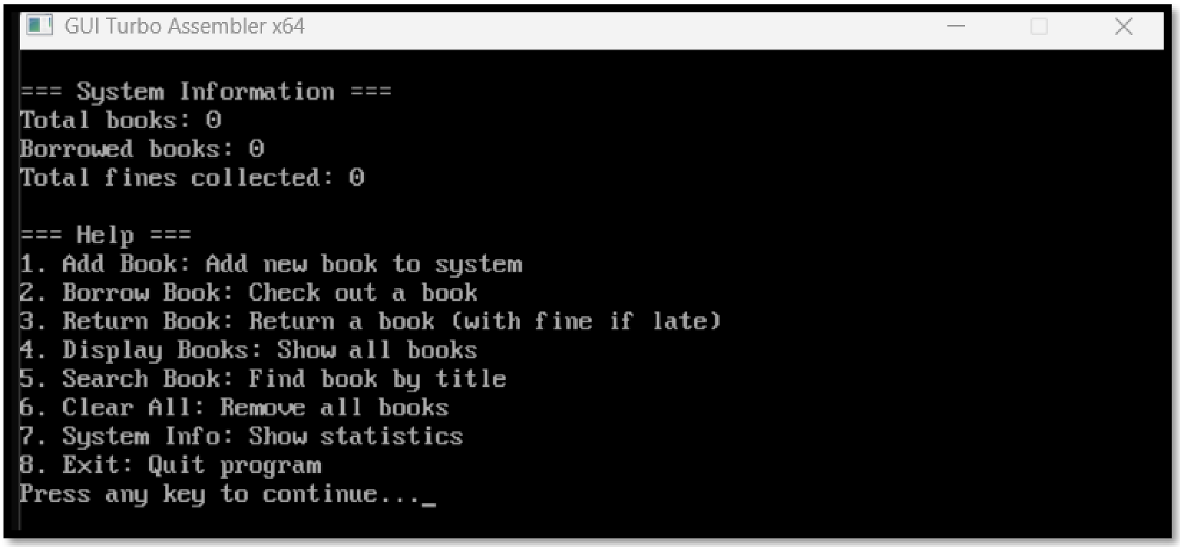
```
Enter title to search: dogs are good
Book not found!
Press any key to continue...
```

Displaying All Books



```
Book not found!
Press any key to continue...
```

System Information Output

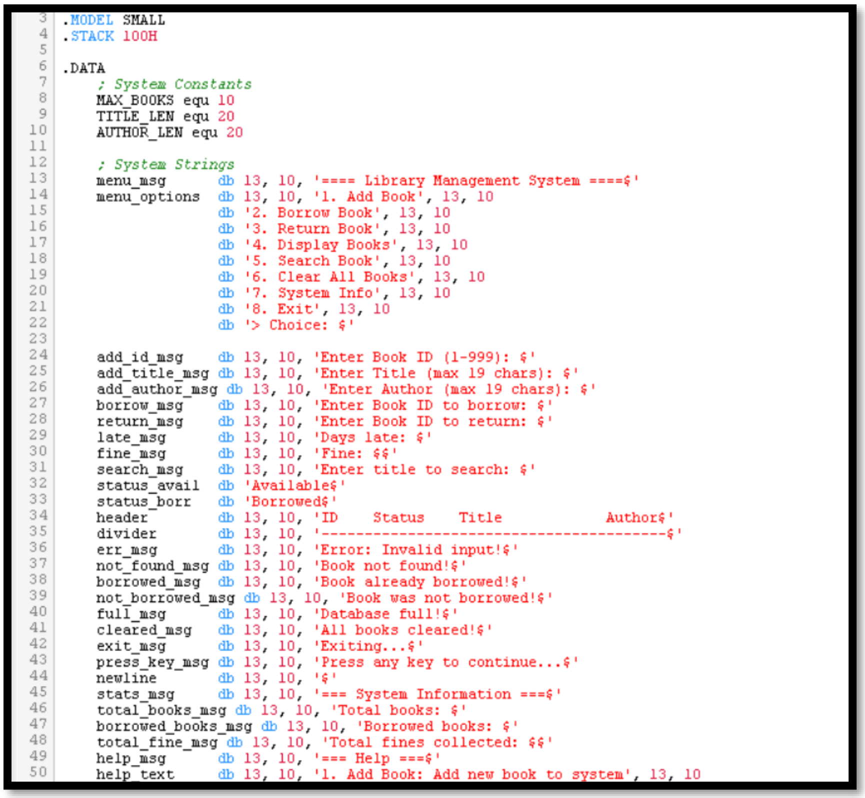


```
=== System Information ===
Total books: 0
Borrowed books: 0
Total fines collected: 0

=== Help ===
1. Add Book: Add new book to system
2. Borrow Book: Check out a book
3. Return Book: Return a book (with fine if late)
4. Display Books: Show all books
5. Search Book: Find book by title
6. Clear All: Remove all books
7. System Info: Show statistics
8. Exit: Quit program
Press any key to continue..._
```

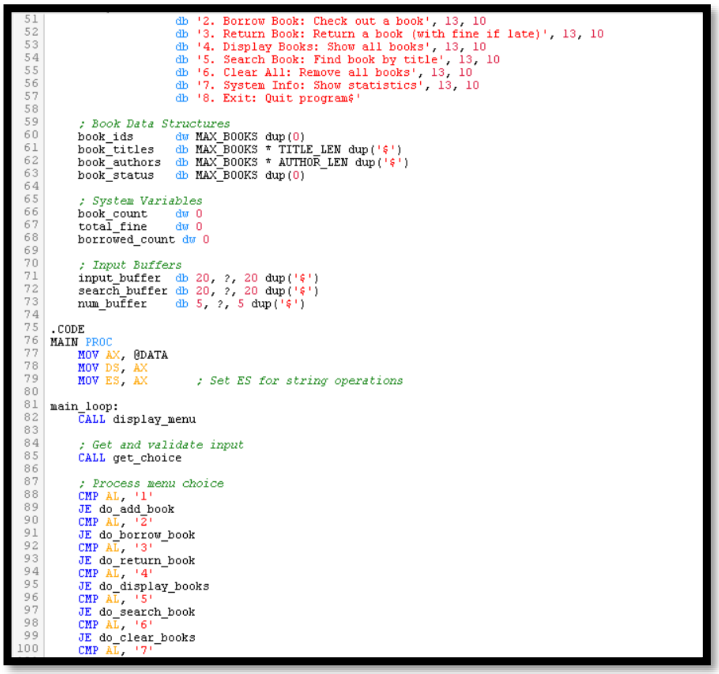
5. Source Code with Explanation

1. Program Main Menu (Control Dispatcher)

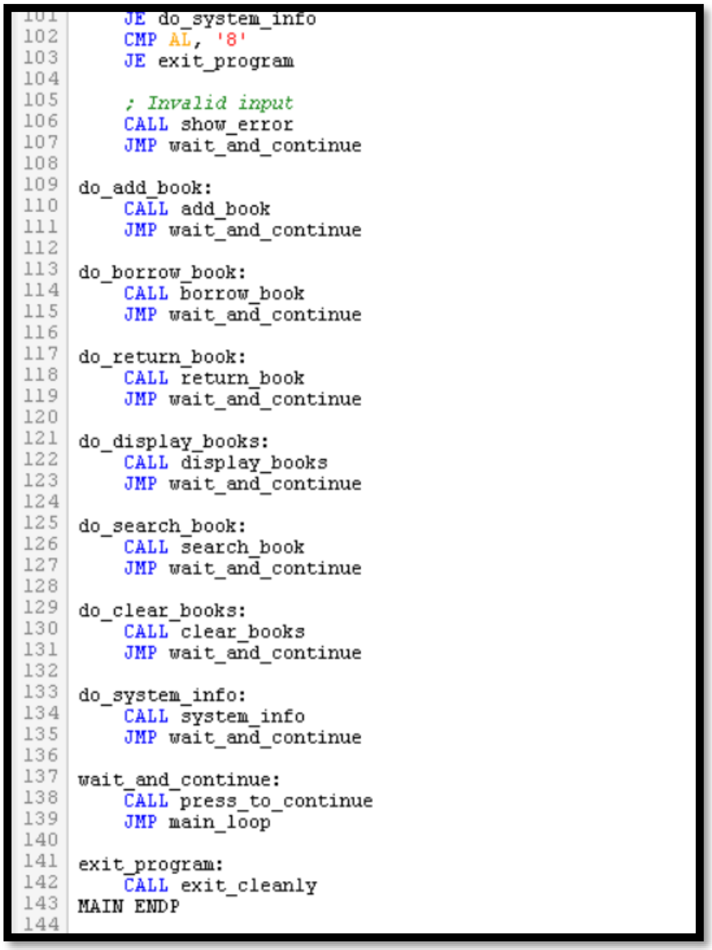
- Implements a control loop using jmp and call instructions to continuously display the menu.
- Captures user input via INT 21h with AH = 01h for single character input.
- Employs conditional branching (cmp, je, jne) to route execution flow to relevant procedures.
- Acts as the central dispatcher of the program.



```
.MODEL SMALL
.STACK 100H

.DATA
    ; System Constants
    MAX_BOOKS equ 10
    TITLE_LEN equ 20
    AUTHOR_LEN equ 20

    ; System Strings
    menu_msg       db 13, 10, '==== Library Management System ====$'
    menu_options   db 13, 10, '1. Add Book', 13, 10
                   db '2. Borrow Book', 13, 10
                   db '3. Return Book', 13, 10
                   db '4. Display Books', 13, 10
                   db '5. Search Book', 13, 10
                   db '6. Clear All Books', 13, 10
                   db '7. System Info', 13, 10
                   db '8. Exit', 13, 10
                   db '> Choice: $'

    add_id_msg     db 13, 10, 'Enter Book ID (1-999): $'
    add_title_msg db 13, 10, 'Enter Title (max 19 chars): $'
    add_author_msg db 13, 10, 'Enter Author (max 19 chars): $'
    borrow_msg     db 13, 10, 'Enter Book ID to borrow: $'
    return_msg     db 13, 10, 'Enter Book ID to return: $'
    late_msg       db 13, 10, 'Days late: $'
    fine_msg       db 13, 10, 'Fine: $$'
    search_msg     db 13, 10, 'Enter title to search: $'
    status_avail   db 'Available$'
    status_borr    db 'Borrowed$'
    header         db 13, 10, 'ID    Status    Title            Author$'
    divider        db 13, 10, '----------------------------------------$'
    err_msg        db 13, 10, 'Error: Invalid input!$'
    not_found_msg db 13, 10, 'Book not found!$'
    borrowed_msg   db 13, 10, 'Book already borrowed!$'
    not_borrowed_msg db 13, 10, 'Book was not borrowed!$'
    full_msg       db 13, 10, 'Database full!$'
    cleared_msg    db 13, 10, 'All books cleared!$'
    exit_msg       db 13, 10, 'Exiting...$'
    press_key_msg db 13, 10, 'Press any key to continue...$'
    newline        db 13, 10, '$'
    stats_msg      db 13, 10, '=== System Information ===$'
    total_books_msg db 13, 10, 'Total books: $'
    borrowed_books_msg db 13, 10, 'Borrowed books: $'
    total_fine_msg db 13, 10, 'Total fines collected: $$'
    help_msg       db 13, 10, '=== Help ===$'
    help_text      db 13, 10, '1. Add Book: Add new book to system', 13, 10
```



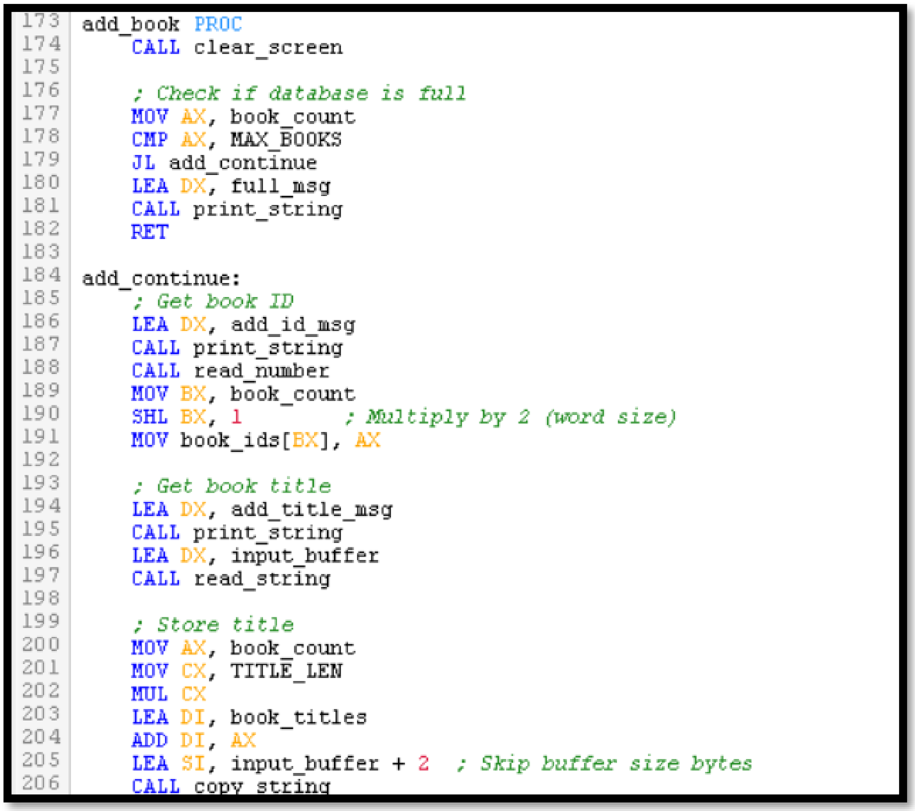
```
                    db '2. Borrow Book: Check out a book', 13, 10
                    db '3. Return Book: Return a book (with fine if late)', 13, 10
                    db '4. Display Books: Show all books', 13, 10
                    db '5. Search Book: Find book by title', 13, 10
                    db '6. Clear All: Remove all books', 13, 10
                    db '7. System Info: Show statistics', 13, 10
                    db '8. Exit: Quit program$'

    ; Book Data Structures
    book_ids      dw MAX_BOOKS dup(0)
    book_titles   db MAX_BOOKS * TITLE_LEN dup('$')
    book_authors  db MAX_BOOKS * AUTHOR_LEN dup('$')
    book_status   db MAX_BOOKS dup(0)

    ; System Variables
    book_count    dw 0
    total_fine    dw 0
    borrowed_count dw 0

    ; Input Buffers
    input_buffer  db 20, ?, 20 dup('$')
    search_buffer db 20, ?, 20 dup('$')
    num_buffer    db 5, ?, 5 dup('$')

.CODE
MAIN PROC
    MOV AX, @DATA
    MOV DS, AX
    MOV ES, AX        ; Set ES for string operations

main_loop:
    CALL display_menu

    ; Get and validate input
    CALL get_choice

    ; Process menu choice
    CMP AL, '1'
    JE do_add_book
    CMP AL, '2'
    JE do_borrow_book
    CMP AL, '3'
    JE do_return_book
    CMP AL, '4'
    JE do_display_books
    CMP AL, '5'
    JE do_search_book
    CMP AL, '6'
    JE do_clear_books
    CMP AL, '7'
```



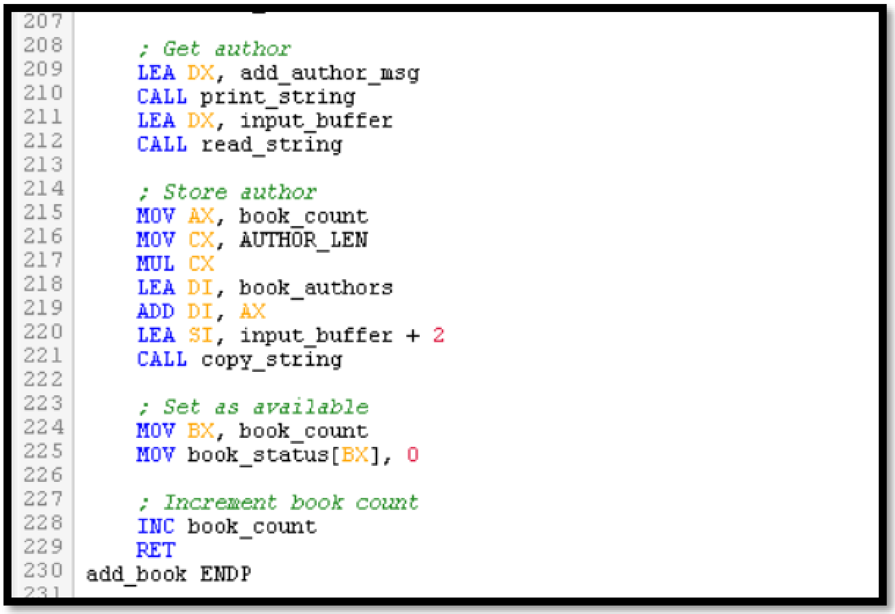
```
    JE do_system_info
    CMP AL, '8'
    JE exit_program

    ; Invalid input
    CALL show_error
    JMP wait_and_continue

do_add_book:
    CALL add_book
    JMP wait_and_continue

do_borrow_book:
    CALL borrow_book
    JMP wait_and_continue

do_return_book:
    CALL return_book
    JMP wait_and_continue

do_display_books:
    CALL display_books
    JMP wait_and_continue

do_search_book:
    CALL search_book
    JMP wait_and_continue

do_clear_books:
    CALL clear_books
    JMP wait_and_continue

do_system_info:
    CALL system_info
    JMP wait_and_continue

wait_and_continue:
    CALL press_to_continue
    JMP main_loop

exit_program:
    CALL exit_cleanly
MAIN ENDP
```

2. Adding Book Entry (Data Insertion and Validation)

- Uses indexed addressing (mov bx, index, mov si, offset array[bx]) to dynamically insert book data into memory.
- Book titles and authors are stored in a 2D byte array with manual offset calculations.
- Implements string input using INT 21h with AH = 0Ah (Buffered Input) for secure, bounded input capture.
- Uses loop counters and maximum book constraints to prevent overflow (via cmp, jl, etc.).
- Status flags for book availability stored in a dedicated byte array (book_status[]).



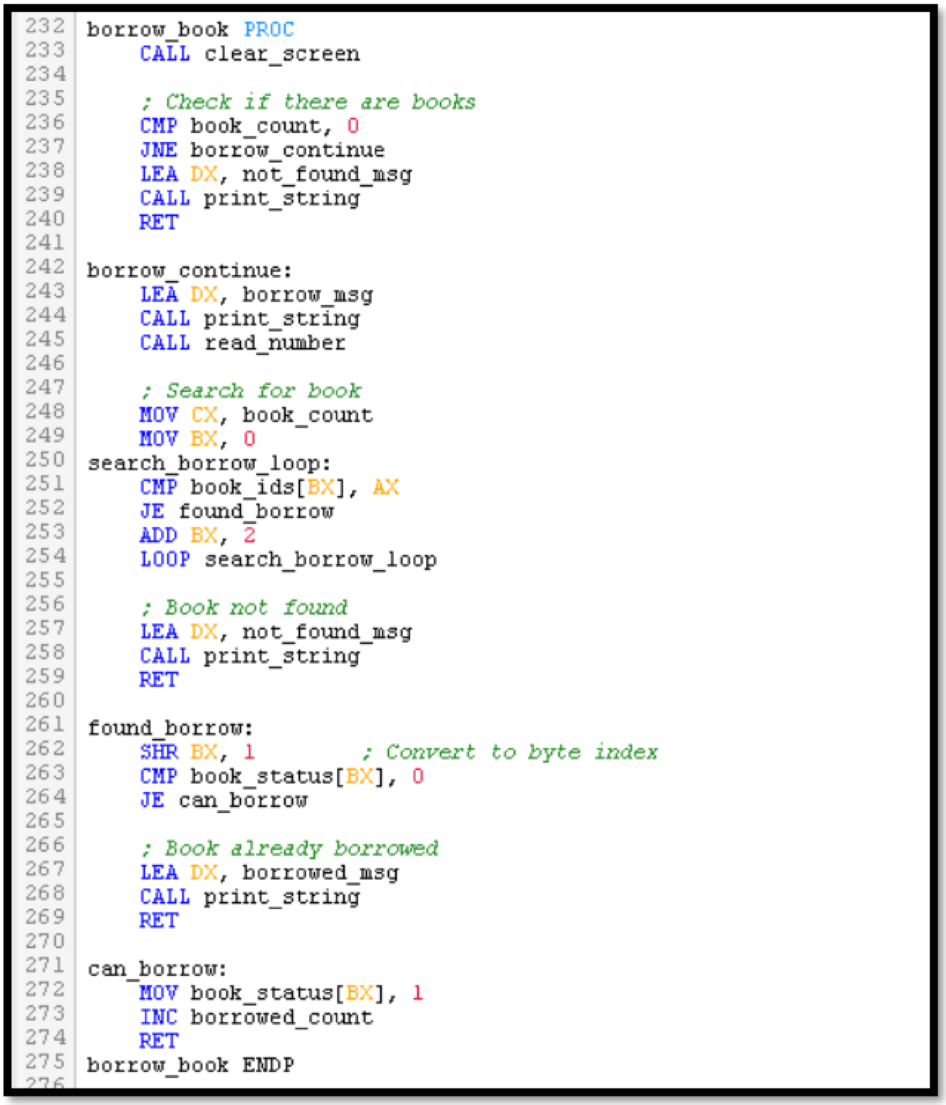
```
add_book PROC
    CALL clear_screen

    ; Check if database is full
    MOV AX, book_count
    CMP AX, MAX_BOOKS
    JL add_continue
    LEA DX, full_msg
    CALL print_string
    RET

add_continue:
    ; Get book ID
    LEA DX, add_id_msg
    CALL print_string
    CALL read_number
    MOV BX, book_count
    SHL BX, 1         ; Multiply by 2 (word size)
    MOV book_ids[BX], AX

    ; Get book title
    LEA DX, add_title_msg
    CALL print_string
    LEA DX, input_buffer
    CALL read_string

    ; Store title
    MOV AX, book_count
    MOV CX, TITLE_LEN
    MUL CX
    LEA DI, book_titles
    ADD DI, AX
    LEA SI, input_buffer + 2  ; Skip buffer size bytes
    CALL copy_string
```



```
    ; Get author
    LEA DX, add_author_msg
    CALL print_string
    LEA DX, input_buffer
    CALL read_string

    ; Store author
    MOV AX, book_count
    MOV CX, AUTHOR_LEN
    MUL CX
    LEA DI, book_authors
    ADD DI, AX
    LEA SI, input_buffer + 2
    CALL copy_string

    ; Set as available
    MOV BX, book_count
    MOV book_status[BX], 0

    ; Increment book count
    INC book_count
    RET
add_book ENDP
```

3. Borrowing a Book (Status Check and State Transition)

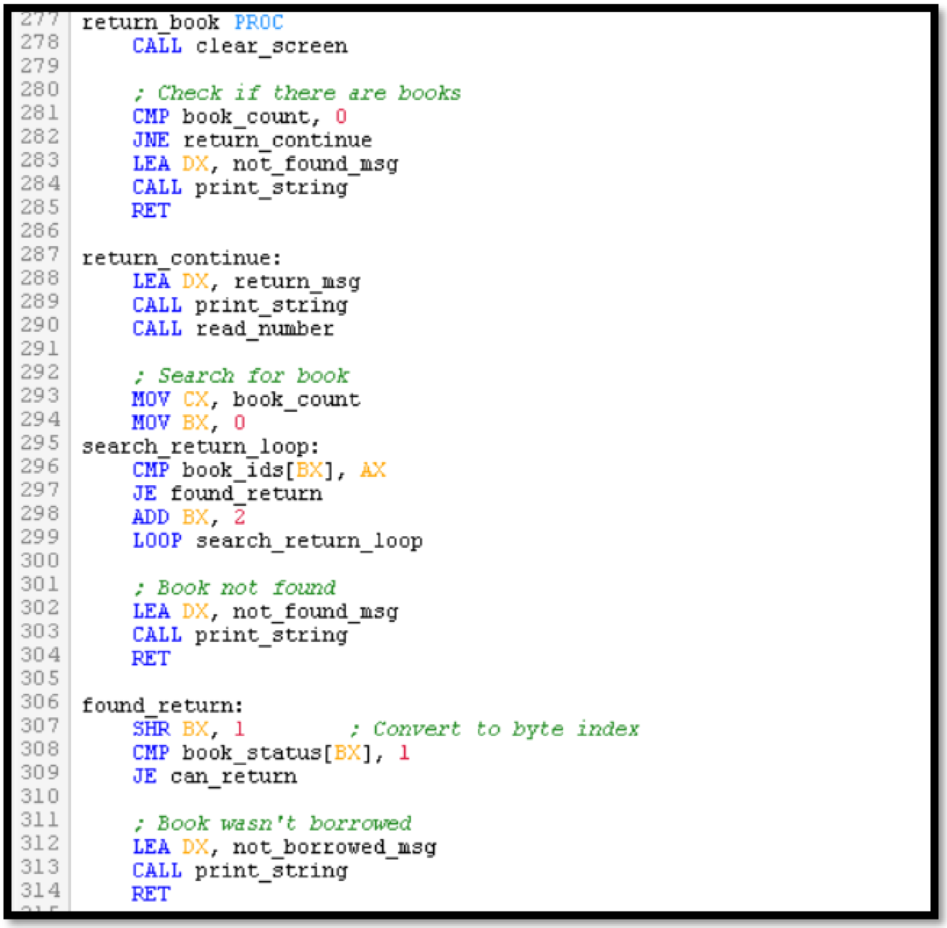
- Prompts for Book ID and converts ASCII input to numerical index using sub al, 30h.
- Verifies book existence and current state using status flags.
- If valid, the status byte is toggled from 'A' (Available) to 'B' (Borrowed).
- Maintains a borrowed counter updated using arithmetic instructions (inc, dec).



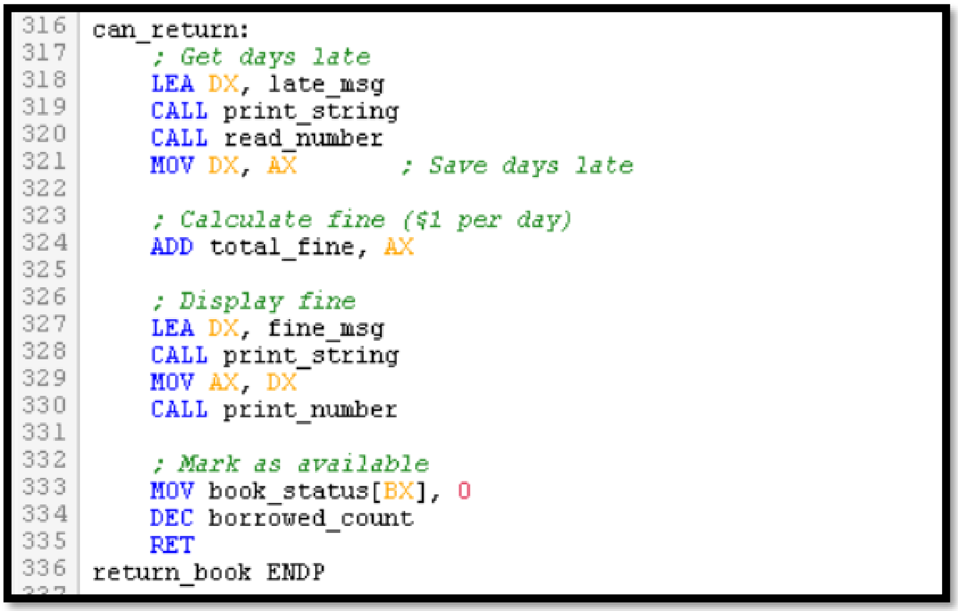
```
borrow_book PROC
    CALL clear_screen

    ; Check if there are books
    CMP book_count, 0
    JNE borrow_continue
    LEA DX, not_found_msg
    CALL print_string
    RET

borrow_continue:
    LEA DX, borrow_msg
    CALL print_string
    CALL read_number

    ; Search for book
    MOV CX, book_count
    MOV BX, 0
search_borrow_loop:
    CMP book_ids[BX], AX
    JE found_borrow
    ADD BX, 2
    LOOP search_borrow_loop

    ; Book not found
    LEA DX, not_found_msg
    CALL print_string
    RET

found_borrow:
    SHR BX, 1         ; Convert to byte index
    CMP book_status[BX], 0
    JE can_borrow

    ; Book already borrowed
    LEA DX, borrowed_msg
    CALL print_string
    RET

can_borrow:
    MOV book_status[BX], 1
    INC borrowed_count
    RET
borrow_book ENDP
```

4. Returning a Book (Arithmetic Computation and Fine Calculation)

- Prompts for Book ID and number of days late.
- Uses ASCII to integer conversion to process late days.
- Fine is calculated using multiplication via the mul instruction (e.g., mov al, fine_rate, mul late_days).
- Fine amount is accumulated in a global variable (total_fine) for system info tracking.
- Updates book status from 'B' to 'A'.



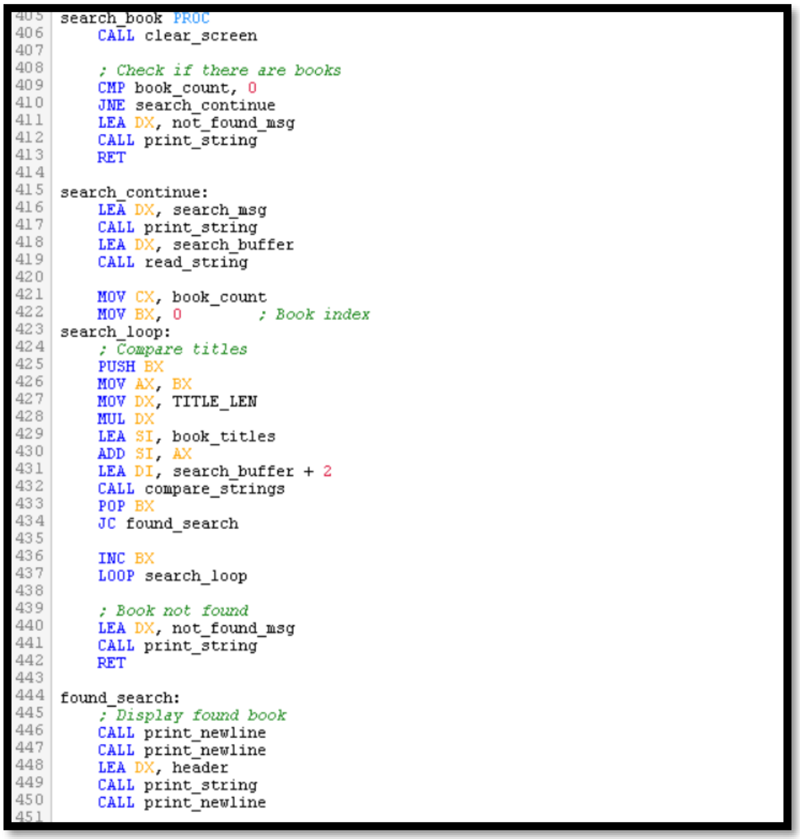
```
return_book PROC
    CALL clear_screen

    ; Check if there are books
    CMP book_count, 0
    JNE return_continue
    LEA DX, not_found_msg
    CALL print_string
    RET

return_continue:
    LEA DX, return_msg
    CALL print_string
    CALL read_number

    ; Search for book
    MOV CX, book_count
    MOV BX, 0
search_return_loop:
    CMP book_ids[BX], AX
    JE found_return
    ADD BX, 2
    LOOP search_return_loop

    ; Book not found
    LEA DX, not_found_msg
    CALL print_string
    RET

found_return:
    SHR BX, 1          ; Convert to byte index
    CMP book_status[BX], 1
    JE can_return

    ; Book wasn't borrowed
    LEA DX, not_borrowed_msg
    CALL print_string
    RET
```



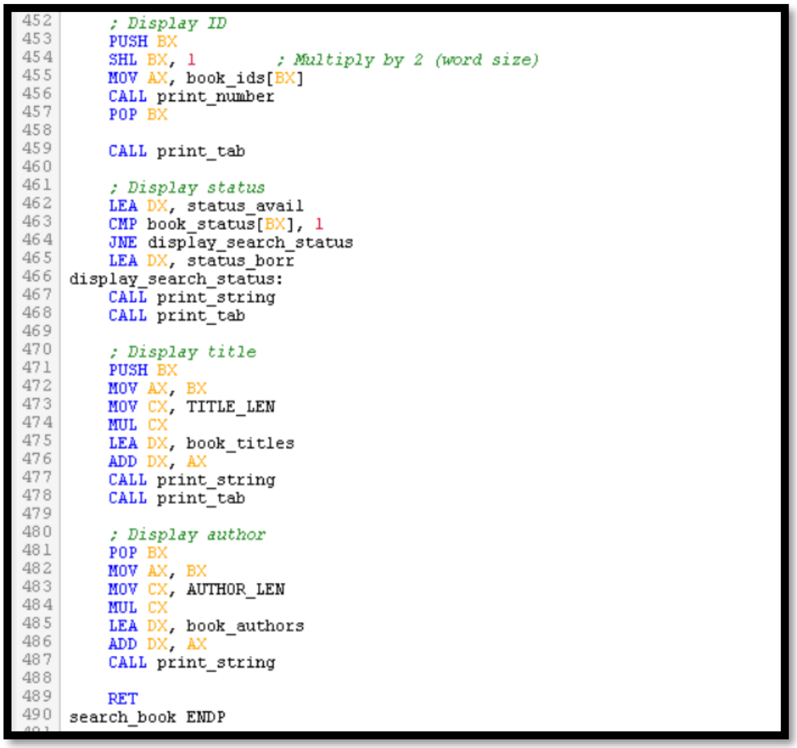
```
can_return:
    ; Get days late
    LEA DX, late_msg
    CALL print_string
    CALL read_number
    MOV DX, AX         ; Save days late

    ; Calculate fine ($1 per day)
    ADD total_fine, AX

    ; Display fine
    LEA DX, fine_msg
    CALL print_string
    MOV AX, DX
    CALL print_number

    ; Mark as available
    MOV book_status[BX], 0
    DEC borrowed_count
    RET
return_book ENDP
```

5. Searching for a Book (String Matching)

- Accepts a book title using buffered input (AH = 0Ah).
- Implements a manual string comparison routine using a loop and cmpsb instruction (or equivalent).
- Compares user input against stored titles using byte wise iteration.
- On match, calculates book's memory offset and displays corresponding metadata (ID, status, author).



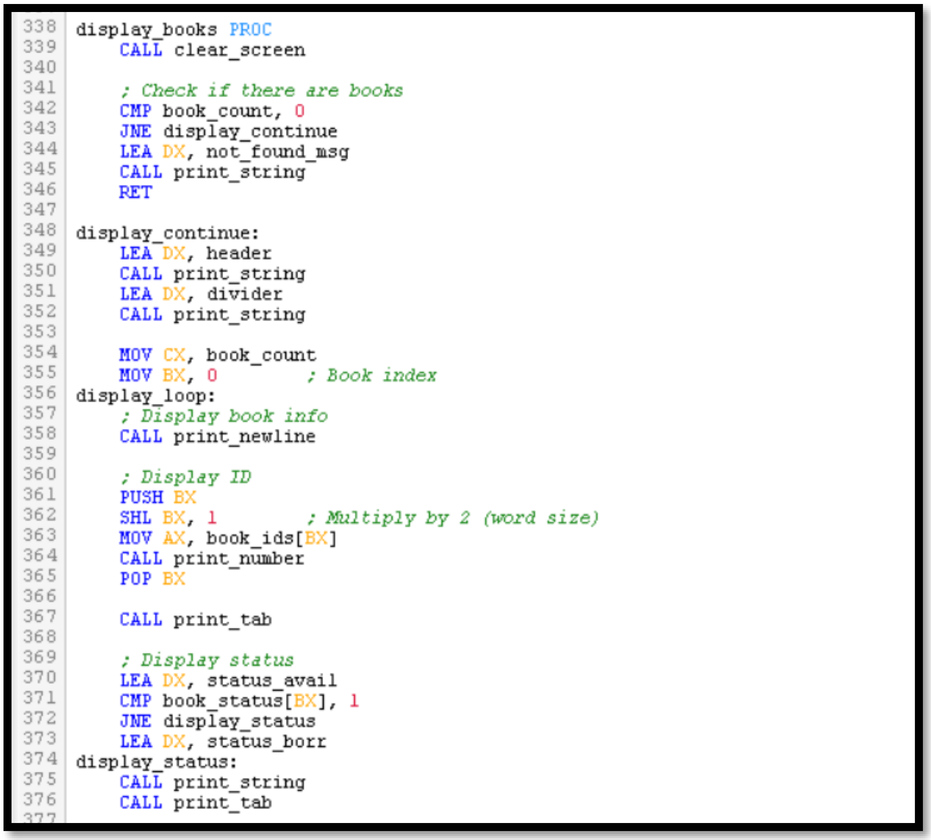
```
search_book PROC
    CALL clear_screen

    ; Check if there are books
    CMP book_count, 0
    JNE search_continue
    LEA DX, not_found_msg
    CALL print_string
    RET

search_continue:
    LEA DX, search_msg
    CALL print_string
    LEA DX, search_buffer
    CALL read_string

    MOV CX, book_count
    MOV BX, 0           ; Book index
search_loop:
    ; Compare titles
    PUSH BX
    MOV AX, BX
    MOV DX, TITLE_LEN
    MUL DX
    LEA SI, book_titles
    ADD SI, AX
    LEA DI, search_buffer + 2
    CALL compare_strings
    POP BX
    JC found_search

    INC BX
    LOOP search_loop

    ; Book not found
    LEA DX, not_found_msg
    CALL print_string
    RET

found_search:
    ; Display found book
    CALL print_newline
    CALL print_newline
    LEA DX, header
    CALL print_string
    CALL print_newline
```



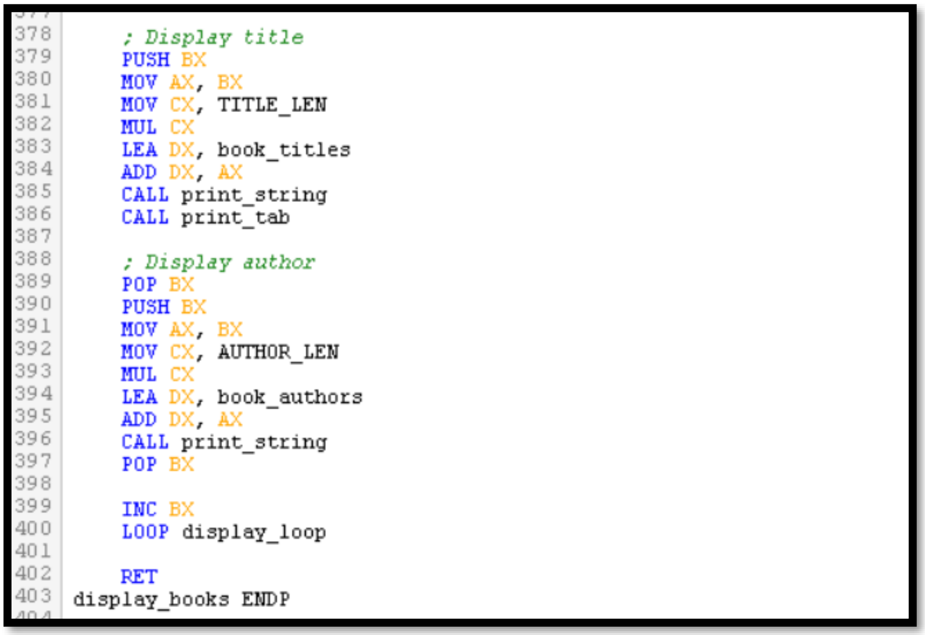
```
    ; Display ID
    PUSH BX
    SHL BX, 1         ; Multiply by 2 (word size)
    MOV AX, book_ids[BX]
    CALL print_number
    POP BX

    CALL print_tab

    ; Display status
    LEA DX, status_avail
    CMP book_status[BX], 1
    JNE display_search_status
    LEA DX, status_borr
display_search_status:
    CALL print_string
    CALL print_tab

    ; Display title
    PUSH BX
    MOV AX, BX
    MOV CX, TITLE_LEN
    MUL CX
    LEA DX, book_titles
    ADD DX, AX
    CALL print_string
    CALL print_tab

    ; Display author
    POP BX
    MOV AX, BX
    MOV CX, AUTHOR_LEN
    MUL CX
    LEA DX, book_authors
    ADD DX, AX
    CALL print_string

    RET
search_book ENDP
```

6. Displaying All Books (Memory Iteration and Output Formatting)

- Iterates over arrays of Book IDs, Titles, Authors, and Status using register based loop control (cx, bx).
- Uses INT 21h, AH = 09h for printing formatted strings to the display.
- Handles conditional output using comparison operators (cmp, jne) to check for existing entries.
- Formats book info output with appropriate spacing using hardcoded control characters (e.g., 09h for TAB, 0Dh/0Ah for newline).



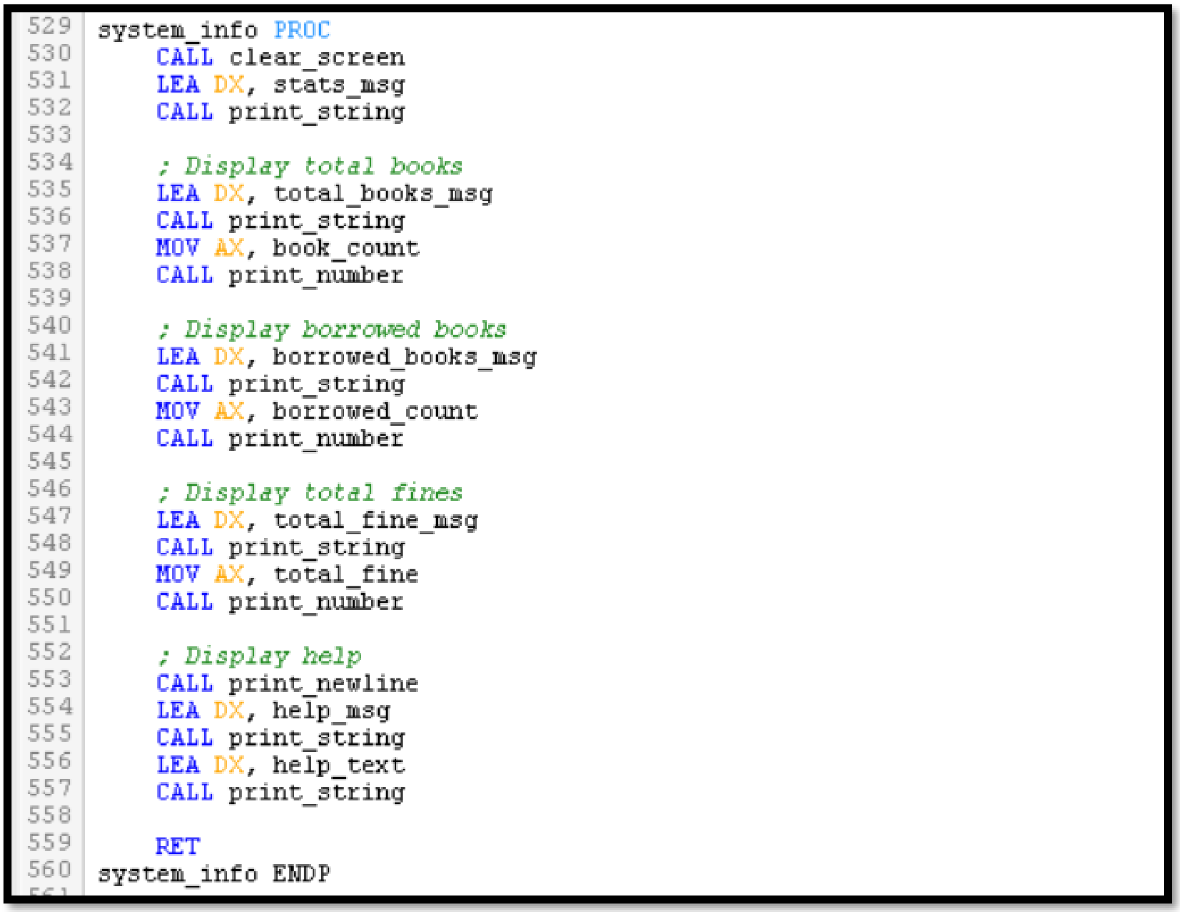
```
display_books PROC
    CALL clear_screen

    ; Check if there are books
    CMP book_count, 0
    JNE display_continue
    LEA DX, not_found_msg
    CALL print_string
    RET

display_continue:
    LEA DX, header
    CALL print_string
    LEA DX, divider
    CALL print_string

    MOV CX, book_count
    MOV BX, 0           ; Book index
display_loop:
    ; Display book info
    CALL print_newline

    ; Display ID
    PUSH BX
    SHL BX, 1           ; Multiply by 2 (word size)
    MOV AX, book_ids[BX]
    CALL print_number
    POP BX

    CALL print_tab

    ; Display status
    LEA DX, status_avail
    CMP book_status[BX], 1
    JNE display_status
    LEA DX, status_borr
display_status:
    CALL print_string
    CALL print_tab
```



```
    ; Display title
    PUSH BX
    MOV AX, BX
    MOV CX, TITLE_LEN
    MUL CX
    LEA DX, book_titles
    ADD DX, AX
    CALL print_string
    CALL print_tab

    ; Display author
    POP BX
    PUSH BX
    MOV AX, BX
    MOV CX, AUTHOR_LEN
    MUL CX
    LEA DX, book_authors
    ADD DX, AX
    CALL print_string
    POP BX

    INC BX
    LOOP display_loop

    RET
display_books ENDP
```

7. System Info Display (Global State Reporting)

- Accesses and displays values from system variables:
    - total_books (count of added books)
    - borrowed_count (currently borrowed)
    - total_fine (sum of all fines in current session)
- Displays static instruction strings from memory using data segment pointers.



```
system_info PROC
    CALL clear_screen
    LEA DX, stats_msg
    CALL print_string

    ; Display total books
    LEA DX, total_books_msg
    CALL print_string
    MOV AX, book_count
    CALL print_number

    ; Display borrowed books
    LEA DX, borrowed_books_msg
    CALL print_string
    MOV AX, borrowed_count
    CALL print_number

    ; Display total fines
    LEA DX, total_fine_msg
    CALL print_string
    MOV AX, total_fine
    CALL print_number

    ; Display help
    CALL print_newline
    LEA DX, help_msg
    CALL print_string
    LEA DX, help_text
    CALL print_string

    RET
system_info ENDP
```

6. User Manual / Guide

System Requirements:

To run the Library Management System on modern computers, a DOS environment is needed, such as:

- DOSBox (recommended) - a DOS emulator for Windows, Linux, and macOS.
- MS-DOS on legacy machines.
- TASM (Turbo Assembler) - for assembling .ASM source code.
- TLINK (Turbo Linker) - for linking object files into executable .EXE files. (Duntemann, 2009)

How to Compile and Run:

```
TASM library.asm
TLINK library.obj
library.exe
```

Features:

Feature	Description
1. Add Book	Allows user to input a new book with ID, title, and author. Prevents duplicates and enforces input validation.
2. Borrow Book	Lets the user borrow a book using its ID. Updates status and tracks borrowed count.
3. Return Book	Prompts for return with late days input. Fine is calculated and added to total fines collected.
4. Display Books	Shows all current books with their ID, availability status, title, and author.
5. Search Book	Lets the user search a book by title. Displays detailed info if found.
6. Clear All Books	Wipes all book data and resets counters. Useful for system reset or new semester.
7. System Info	Displays total books, borrowed books, and total fines collected. Also shows help commands.
8. Exit	Exits the program cleanly after screen clearing.

Basic Controls and Navigation

- **Input Numbers** only when prompted for Book ID or Days Late.
- **String input (titles/authors)** should not exceed 19 characters.
- **Do not enter alphabetic characters** when the program asks for numeric input.
- **Follow on screen prompts** and press any key when prompted to continue.

7. Conclusion

The Library Management System, developed using x86 Assembly language, illustrates the accuracy and efficiency inherent in low level programming practices. It can carry out intricate operations like:

- Structured data storage using arrays and buffers.
- Input validation and error checking.
- Conditional logic with branching and loops.
- Direct memory manipulation and screen output.

This is a classic example of incorporating real-world applications, like an actual book database system, in a constrained DOS environment using TASM. In addition, it teaches fundamental skills that can be transferably used across a wide range of fields, from systems programming and OS design to cybersecurity domains, like reverse engineering and malware analysis. Familiarity with Assembly Language facilitates understanding of systems level programming as well as a more profound understanding of software interactions with fundamental computer hardware.

8. References

Duntemann, J. (2009). *Assembly Language Step by Step: Programming with DOS and Linux* (3rd ed.)

Eagle, C. (2011). *The IDA Pro Book: The Unofficial Guide to the World's Most Popular Disassembler* (2nd ed.). No Starch Press.

Hyde, R. (2003). *The Art of Assembly Language* (2nd ed.). No Starch Press.

National Institute of Standards and Technology. (2018). *National Software Reference Library (NSRL)*.https://www.nist.gov/itl/ssd/software-quality-group/national-software-reference-library-nsrl

Skoudis, E., & Liston, T. (2006). *Counter Hack Reloaded: A Step by Step Guide to Computer Attacks and Effective Defenses* (2nd ed.). Prentice Hall.

Wikipedia contributors. (2025a, July). *Assembly language*. In Wikipedia. https://en.wikipedia.org/wiki/Assembly_language