**Princess Sumaya University for Technology**

King Abdullah II Faculty of Engineering

Electrical Engineering Department



**Microprocessors (22344)**

**Software Quiz App Assembly Project**

**Author:** Qais Jildeh (20210155)

**Supervisor:** Dr. Heba Abdel-Nabi

*August 6, 2024*

***Abstract***

*This project deploys a variable Multiple-Choice Quiz Application in 8086 assembly language with user login, randomly sorted questions, and scoring. Users are verified through preassigned credentials, respond dynamically requested questions on the selected subject matter, and get instant feedback. The solution focuses on modular design, memory management, and interrupt-based I/O, demonstrating deployed low-level programming to interactive software.*

**Table of Contents**

[Introduction 3](#_Toc197981456)

[Requirements 3](#_Toc197981457)

[Program 3](#_Toc197981458)

[Procedure #1: println 4](#_Toc197981459)

[Procedure #2: printLogo 5](#_Toc197981460)

[Procedure #3: userAuthentication 7](#_Toc197981461)

[Procedure #4: shuffleQuiz 12](#_Toc197981462)

[Procedure #5: quiz 15](#_Toc197981463)

[Procedure #6: choose 23](#_Toc197981464)

[Procedure #7: isValidAnswer 24](#_Toc197981465)

[Procedure #8: gradeQuiz 28](#_Toc197981466)

[Procedure #9: printScoreboard 31](#_Toc197981467)

[Procedure #10: terminateProgram 36](#_Toc197981468)

[Conclusion 37](#_Toc197981469)

[References 37](#_Toc197981470)

**Table of Figures**

[Figure 1: Quiz app flowchart 3](#_Toc197986431)

[Figure 2: Quiz app execution flow 4](#_Toc197986432)

[Figure 3: Variables used by println procedure in the data segment 4](#_Toc197986433)

[Figure 4: println Procedure 4](#_Toc197986434)

[Figure 5: printLogo procedure output 5](#_Toc197986435)

[Figure 6: Variables used by printLogo procedure in the data segment 5](#_Toc197986436)

[Figure 7: printLogo procedure 6](#_Toc197986437)

[Figure 8: Output after three wrong login attempts 7](#_Toc197986438)

[Figure 9:Output after entering correct credentials 8](#_Toc197986439)

[Figure 10: Variables used by userAuthentication procedure in the data segment 8](#_Toc197986440)

[Figure 11: userAuthentication procedure 11](#_Toc197986441)

[Figure 12: Variables used by shuffleQuiz procedure in the data segment 12](#_Toc197986442)

[Figure 13: shuffleQuiz Procedure 14](#_Toc197986443)

[Figure 14: quiz procedure output 15](#_Toc197986444)

[Figure 15: Variables used by quiz procedure in the data segment 16](#_Toc197986445)

[Figure 16: quiz Procedure 22](#_Toc197986446)

[Figure 17: choose Procedure 23](#_Toc197986447)

[Figure 18: choose & isValidAnswer procedures output 24](#_Toc197986448)

[Figure 19: isValidAnswer Procedure 28](#_Toc197986449)

[Figure 20: gradeQuiz procedure output 28](#_Toc197986450)

[Figure 21: Variables used by gradeQuiz procedure in data segment 28](#_Toc197986451)

[Figure 22: gradeQuiz procedure 30](#_Toc197986452)

[Figure 23: printScoreboard procedure output 31](#_Toc197986453)

[Figure 24: Variables used by printScoreboard procedure in the data segment 31](#_Toc197986454)

[Figure 25: printScoreboard Procedure 36](#_Toc197986455)

[Figure 26: Output of terminateProgram procedure 36](#_Toc197986456)

[Figure 27: Variables used by the terminateProgram procedure in the data segment 36](#_Toc197986457)

[Figure 28: terminateProgram Procedure 37](#_Toc197986458)

## Introduction

This project creates an 8086-assembly language customizable quiz game about the cars movie with user authentication, dynamic question shuffling, and scoring. Based on modular procedures as its basic structure, it exhibits effective memory management and input handling in real-time, demonstrating real-world uses of low-level programming for interactive systems.

### Requirements

* User Authentication: Develop a login system with predefined credentials and a 3-attempt limit.
* Dynamic Question Handling: Load, shuffle, and display 4 MCQs on a user-chosen topic.
* Scoring System: Calculate and display user scores.
* Modular Design: Organize code into reusable procedures and analyze performance efficiency.

## Program

The quiz app consists of the procedures **userAuthentication**, **shuffleQuiz**, **quiz**, **choose**, **isValidAnswer**, **gradeQuiz**, **printScoreboard**, and **terminateProgram** to provide it with the correct logic to function according to the requirements. The program will have the following the flow of execution.

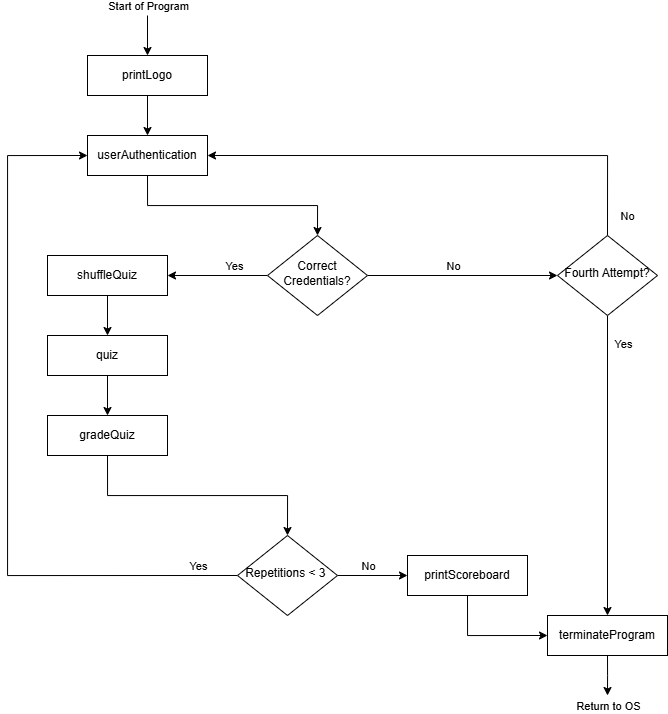


Figure 1: Quiz app flowchart

The program begins by printing the quiz logo, then prompts the user to enter their credentials. If the credentials are entered correctly, the quiz starts; otherwise, the user has two more attempts to log in. After successful authentication, the questions are shuffled using the **shuffleQuiz** procedure, and the user receives one question at a time. For each question, the user is prompted to input a choice. If the input is invalid, they will be prompted again until a valid choice is entered. Once a valid choice is provided, it is recorded and the score is updated accordingly. After the user answers 4 questions (selected from a pool of 5), the final score is displayed, and the authentication process begins again for the next user. This process repeats for a total of 3 users. Afterward, the scoreboard is updated and the program terminates.



Figure 2: Quiz app execution flow

### Procedure #1: println

**println** procedure handles printing of sequence of newline as output formatting. It uses DOS interrupt 21h subroutine 09h to print the string in memory located at address of endl containing ASCII strings 13 and 10 for carriage return and line feed, respectively, and which place cursor back on line beginning and move it one line down. This process is significant in maintaining output readable throughout the program because it keeps getting invoked following the outputting of messages, queries, or user prompts in order to prevent text overlap on the screen.



Figure 3: Variables used by println procedure in the data segment



Figure 4: println Procedure

### Procedure #2: printLogo

This routine draws an ASCII art styled logo when the program starts. It prints eight predefined strings (logoLine1 to logoLine8) sequentially with the DOS string output call (09h). Each line is topped off with a call to println to maintain vertical spacing. The logo is an opening graphical introduction of the program and is succeeded by a separator line (logoLine8) to introduce the authentication phase. The official logo emphasizes the focus of the program on user experience.

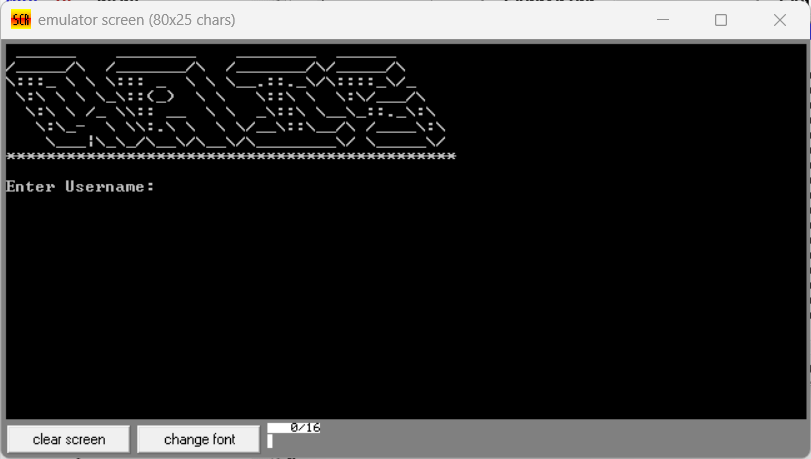


Figure 5: printLogo procedure output



Figure 6: Variables used by printLogo procedure in the data segment



Figure 7: printLogo procedure

### Procedure #3: userAuthentication

The authentication takes place by ensuring user credentials during three attempts. It requests user input of the username and password from 09h to give output prompts (userRequest, passRequest) and 0Ah to return buffered input as userIn and passIn. The operation also compares input strings with hardcoded user credentials (user1, user2, user3 and corresponding passwords) through the cmpsb instruction, performing a byte-by-byte comparison. The input size (stored in the second byte of the buffer) scales the comparison range. On successful validation, the process outputs a success message; after three failures, it terminates the program.



Figure 8: Output after three wrong login attempts



Figure 9:Output after entering correct credentials



Figure 10: Variables used by userAuthentication procedure in the data segment







Figure 11: userAuthentication procedure

To calculate the number of clock cycles, we need to analyze the code first. In the worst case, the authentication procedure will be executed three consecutive times due to login error. Here is the analysis of the procedure:

Everytime we call **println** it will cost us the time of calling, executing the procedure, and returning. Number of clock cycles is as follows

Before entering the loop, we do **mov, cx, 3** this cost us 4 clock cycles.

The number of clock cycles inside the loop, note the with each time we take user input or output to the display it will cost us (2+6+4+51 = 63 clock cycles):

The loop will be entered 3 times but the loop instruction will take branch twice only:

The procedure will call the **terminateProgram** procedure, this will cost us:

Total number of clock cycles will be 604 + 2832 = 3436 clock cycles.

The 8086 has a 5MHz clock frequency, so the time need to execute the procedure is calculated as follows:

### Procedure #4: shuffleQuiz

This process pseudo-randomizes question order of quizzes with a pseudo-random number generator. It initializes by producing a random index (0–4) based on a seed. The seed is multiplied by 4, divided by 5, and its remainder (which is stored in AH) is used to create the index. According to the index, pair-wise elements within the items array (holding question numbers 1–5) are swapped (Swap00 interchanges the second and third). The seed is then converted with the formula seed = (seed \* 7) + 3 to introduce randomness for future shuffles.



Figure 12: Variables used by shuffleQuiz procedure in the data segment





Figure 13: shuffleQuiz Procedure

### Procedure #5: quiz

The quiz process provides the shuffled questions. It cycles through the items array and proceeds to the related question label (question1) based on the shuffled index. Each question label displays the question text and multiple-choice items (q1–q5 and a1–d5) with 09h. The currentQuestion variable stores the current question to enable the choose process to retrieve the user's answer to the right question.



Figure 14: quiz procedure output



Figure 15: Variables used by quiz procedure in the data segment













Figure 16: quiz Procedure

### Procedure #6: choose

This process stores and verifies the user's response. It outputs the answerPrompt string and inputs a single character through 01h, making the character visible on screen. Incorrect inputs or non-A/B/C/D characters produce two sound beeps (ASCII 07h) through 02h, and the loop will continue until the right answer is input. Correct answers are passed to isValidAnswer to be graded and update the score.

The **choose** procedure uses the same variables as the **quiz** segment.



Figure 17: choose Procedure

### Procedure #7: isValidAnswer

The **isValidAnswer** procedure verifies answers and increments the score. It lowercase-transforms letters A–D via bitwise OR operation (or al, 0x20) and verifies if input is valid options. The answer is stored in buffers (ans1–ans5) depending on currentQuestion. Correct answers (e.g., a for Q1, c for Q2) increase the score variable, while incorrect answers activate the wrongAnswer message and beep. The score is represented as an ASCII digit (e.g., 48 for '0'), and this makes display logic simpler.

The **isValidAnswer** procedure also uses the same variables as **choose** and **quiz** procedures.

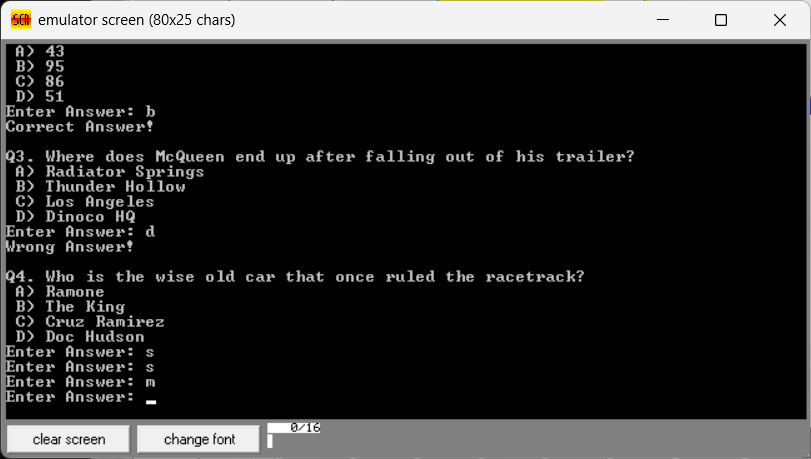


Figure 18: choose & isValidAnswer procedures output









Figure 19: isValidAnswer Procedure

### Procedure #8: gradeQuiz

Once the quiz is finished, the **gradeQuiz** procedure shows the old score and resets the score board. It prints statement1, the ASCII score (52 for '4'), and statement2 (/4!). The logged-in user is determined through string comparison, and his or her score is held in qaisScore, bassemScore, or hannaScore. The score variable is initialized to 48 ('0') in preparation for future quiz attempts. This modular solution enables multiple users to attempt the quiz one by one.

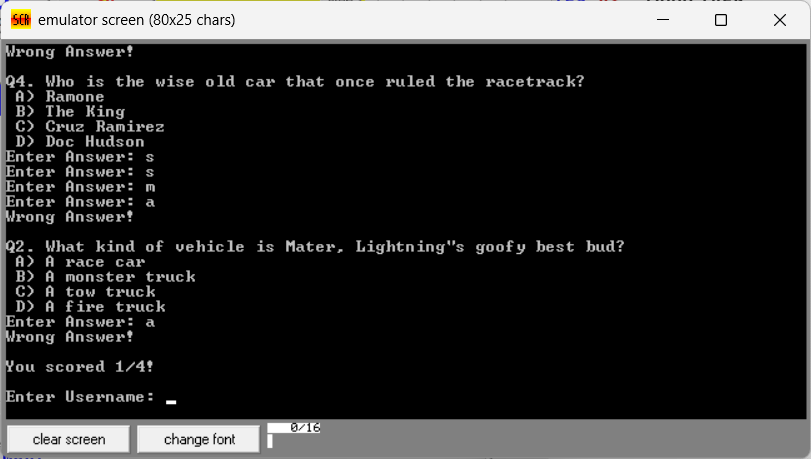


Figure 20: gradeQuiz procedure output



Figure 21: Variables used by gradeQuiz procedure in data segment





Figure 22: gradeQuiz procedure

### Procedure #9: printScoreboard

The **printScoreboard** procedure ranks and prints scores of users in descending order. ASCII score values are compared to conditional jumps (jae, jb) for ranking the users. Pointers (firstName, secondName, thirdName) are allocated to user names in relation to their scores, and the outputs are printed out using colons and spaces to enhance formatting. The scoreboard increases competition by displaying rankings, utilizing 09h for strings and 02h for a single character.



Figure 23: printScoreboard procedure output



Figure 24: Variables used by printScoreboard procedure in the data segment











Figure 25: printScoreboard Procedure

### Procedure #10: terminateProgram

The **terminateProgram** procedure allows graceful exit for the program. It displays exit messages (terminate1, terminate2, terminate3) and calls int 21h/4C00h to return control to the operating system. The structured exit provides a clean program termination with no residual memory problems.



Figure 26: Output of terminateProgram procedure



Figure 27: Variables used by the terminateProgram procedure in the data segment



Figure 28: terminateProgram Procedure

## Conclusion

This project implemented an MCQ Quiz Game in 8086 assembly with customizable functionality, with successful key features: a safe 3-attempts login mechanism based on family-based credentials, dynamic shuffling of 4 Cars-themed questions, case-insensitive answer checking, and real-time score update with feedback. Modular code structure was optimized for efficiency, with greater emphasis placed on hardware interaction and resource handling, while demonstrating real-world applications of low-level programming in designing interactive systems.

## References

<https://www.tutorialspoint.com/microprocessor/microprocessor_8086_instruction_sets.htm>

<https://yassinebridi.github.io/asm-docs/8086_bios_and_dos_interrupts.html#int21h>

<https://yassinebridi.github.io/asm-docs/8086_bios_and_dos_interrupts.html>