Green University of Bangladesh

Department of Computer Science and Engineering (CSE)
Semester: (Fall, Year: 2023), B.Sc. in CSE (Day)

Sudoku Solver

Course Title: Microprocessors And Microcontrollers Lab
Course Code: CSE-304
Section: 213-D1

Students Details

Name	ID
Md Jabed Hossen	213902046

Submission Date: 28-12-2023
Course Teacher's Name: Sudip Ghoshal

[For teachers use only: Don't write anything inside this box]

Lab Project Status

Marks:	**Signature:**
Comments:	**Date:**

Contents

Chapter 1

Introduction

1.1 Overview

The Sudoku solver project, developed by Md Jabed Hossen, is an innovative assembly language program designed for solving Sudoku puzzles. While it effectively handles and solves 3x3 Sudoku puzzles, the project currently grapples with the complexities of validating solutions for the larger 9x9 format, particularly in managing subgrids.

1.2 Motivation

This project stems from a fascination with the logical intricacies of Sudoku and the challenge of encoding its rules into the precise and systematic framework of 8086 assembly language. The aim is to demonstrate how assembly language can be utilized to create complex algorithms capable of solving and interacting with popular logic-based puzzles.

1.3 Problem Definition

1.3.1 Problem Statement

The project confronts the challenge of creating an assembly language program that can accurately solve 3x3 Sudoku puzzles and work towards handling the increased complexity of 9x9 puzzles.

1.3.2 Complex Engineering Problem

A significant challenge is designing an efficient algorithm that operates within the constraints of 8086 assembly language, especially for the 9x9 puzzles where handling subgrids significantly increases computational complexity

1.4 Goals/Objectives

The project is driven by several key objectives:

Accuracy: Ensuring the solver accurately resolves 3x3 puzzles and progresses towards effectively managing 9x9 Sudoku puzzles.

Efficiency: Achieving a balance between computational resource utilization and speed, particularly challenging in assembly language programming.

User Interaction: Developing a user-friendly interface for easy input and clear display of Sudoku puzzles and solutions

1.5 Applications

The Sudoku solver can be used as an educational tool to help students and enthusiasts learn about assembly language programming. It can also serve as a benchmark for testing the performance of x86 processors.

Chapter 2

Design/Development/Implementation of the Project

2.1 Introduction

This section outlines the design and development process of the Sudoku solver project. The project is realized using 8086 assembly language, chosen for its direct hardware control and efficiency in executing logical operations, essential for puzzle-solving algorithms.

2.2 Design and Development Approach

2.2.1 Language and Environment

- **Assembly Language (8086):** The project is developed in 8086 assembly language, providing granular control over system resources and enabling efficient algorithm implementation.
- **Tools and Environment:** The development tool used is EMU8086, an integrated development environment for the Intel 8086 microprocessor, which includes an assembler and emulator. This software allows for coding, compiling, and testing the application within a Windows environment, simulating the execution as if it were running on an actual 8086 processor.

2.2.2 Project Structure

- **Data Segment Organization:** The data segment carefully organizes string literals for the Sudoku grid, messages, and user input arrays, ensuring efficient memory usage and easy access.
- **Modular Design:** The project is structured into macros and procedures, each handling specific functionalities like input capture, cursor movement, and display formatting, enhancing readability and maintainability.

2.3 Implementation

The project leverages assembly language, specifically the 8086 instruction set, to implement the game logic.

2.3.1 Assembly code of SudoKu Solver

Sudoku Game Initialization in Assembly Language

This code segment marks the beginning of the Sudoku game in EMU8086, setting up the program to run in a simulated 8086 microprocessor environment within Windows. This setup allows the assembly code to be tested and executed as if it were running on an actual 8086 machine.

```
; Md Jabed Hossen
; ID 213902046
; Project 1: Sudoku
; Program to print sudoku board, ask for input, and
  print new board until user quits the program

;.model small
;.stack 100h

.model tiny
   org 100h
```

Data Definitions for Sudoku Game Interface

This section of the code sets up the data elements for the Sudoku game, including the visual representation of the borders and rows of the Sudoku grid, user input storage, and various messages displayed during the game. It defines strings for the welcome screen, instructions, and feedback on the validity of the solution provided by the user.

```
.data
border db '+---+---+---+$'

row1a db '2 | 1 |   $'
row2a db '1 |   |   $'
row3a db '  | 2 | 1 $'


userInputs db 4 dup(?) ; Array to store 4 user inputs

sol1a db '2 | 1 | 3 $'
sol2a db '1 | 3 | 2 $'
sol3a db '3 | 2 | 1 $'

```

```


bwrow db '+-------------+$'
borderline db '| $'
borderline2 db ' | $'

again db 'Press any key to try again or press enter to
   see solution and quit: $'
welcome db '                                W E L C O M E  T O  S
    U D O K U                              $'
count db 1
one db 1
rowcount db 1
instructions db 'Enter a number at the location of the
   cursor. Press space for blank.$'
toquit db 'The solution is above. Press any key to quit:
    $'
space db '

   $'

select db 'Select a number: 1 ( for 3*3 sudoku ) or 2 (
   for 9*9 sudoku) $'

validMsg db 'Congatulations ! Sudoku is valid.$'
invalidMsg db 'Sorry ! Sudoku is invalid.$'
```

Sudoku Game Main Procedure Logic

This assembly code defines the main procedure, where the game initializes data segment registers, displays a selection message, and processes user input to choose between a 3x3 or 9x9 Sudoku puzzle. It loops on invalid input and directs to the appropriate puzzle size handling code, ensuring that the game responds accurately to the player's selection before eventually exiting.

```
main proc
    mov ax, @data
    mov ds, ax

input_loop:
    ; Display the selection message
    mov dx, offset select
    mov ah, 9
    int 21h

    call enterkey

    ; Capture the user's choice
    mov ah, 1
    int 21h
    mov bl, al
    sub bl, '0'

    cmp bl, 1
    je do_sudoku_size_three
    cmp bl, 2
    je do_sudoku_size_nine
    jmp input_loop  ; Invalid input, loop back

do_sudoku_size_three:
    ; Call the macro for 3x3 Sudoku
    SUDOKU_SIZE_THREE
    jmp end_of_main

do_sudoku_size_nine:
    ; Call the macro for 9x9 Sudoku
    SUDOKU_SIZE_NINE
    jmp end_of_main

end_of_main:
    ; Cleanup and exit the program
    mov ax, 4C00h
    int 21h

endp main
end main
```

User Input Capture Macro

The `CAPTURE_INPUT` macro is designed for capturing a single keystroke from the user and storing it in a predefined array. It employs DOS interrupt services to read the input and then saves the character in the specified index of the userInputs array, streamlining the process of gathering user responses in the Sudoku game.

```
CAPTURE_INPUT MACRO index
    mov ah, 01h
    int 21h
    mov userInputs[index], al
ENDM
```

Cursor Positioning Macro

`MOVE_CURSOR` is a macro that manipulates the cursor's position on the screen. It uses BIOS video services to place the cursor at a specified row and column, facilitating user interaction with text-based interfaces like the Sudoku game board.

```
MOVE_CURSOR MACRO column, row
    mov ah, 3
    mov bh, 0
    int 10h
    mov ah, 02h        ; Function to set cursor position
    mov bh, 0          ; Page number
    mov dl, column     ; Column
    mov dh, row        ; Row
    int 10h            ; BIOS interrupt for video services
ENDM
```

Macro for Printing Colored Strings

The `PRINT_COLORED_STRING` macro is designed to output text strings with specified colors to the screen. It utilizes BIOS and DOS interrupts to set the text color and then print the message, enhancing the visual aspect of the user interface in the Sudoku game.

```
PRINT_COLORED_STRING MACRO msg, color
    mov ah, 09h
    mov bl, color
    mov cx, 1
    int 10h
    mov dx, offset msg
    int 21h
ENDM
```

Macro for Displaying Sudoku Board Rows

This macro, PRINT_SHOW_BOARD , is dedicated to printing individual rows of the Sudoku board along with their decorative borders. It uses DOS interrupt services to display the row data, followed by the bottom border lines and new line spacing to neatly format the Sudoku grid on the screen.

```
PRINT_SHOW_BOARD MACRO rows

    mov dx, offset rows
    mov ah, 9
    int 21h

    call printborline2
    call enterkey
    call printbwrow
    call enterkey
ENDM
```

Macro and Procedures for 3x3 Sudoku Game Logic

This macro encapsulates the logic for displaying a 3x3 Sudoku board, capturing user input, and validating the completed puzzle. It includes procedures for printing the board, moving the cursor, capturing input, and validating the solution. The macro handles user interaction flow, from taking inputs to providing feedback on solution correctness.

```
SUDOKU_SIZE_THREE MACRO

call printboard
call enterkey
call enterkey

mov ah, 09h
mov bl, 10
mov cx, 68                ; set color
int 10h

mov dx, offset instructions
mov ah, 9
int 21h

ask1:
MOVE_CURSOR 10, 3
CAPTURE_INPUT 0

ask2:
MOVE_CURSOR 6, 5
CAPTURE_INPUT 1

ask22:
MOVE_CURSOR 10, 5
CAPTURE_INPUT 2

ask3:
MOVE_CURSOR 2, 7
CAPTURE_INPUT 3
jmp more


more:
call enterkey
call enterkey
call enterkey
call enterkey
mov dx, offset again
mov ah, 9
int 21h
mov ah, 1
```

```
int 21h
cmp al, 13
je solquit
jmp ask1

solquit:
call printsolboard
call enterkey
call enterkey


call validateSudoku

mov dx, offset toquit
mov ah, 9
int 21h
mov ah, 1
int 21h
mov ax, 0003h
int 10h
mov ah, 4ch
int 21h

pwelcome proc
mov dx, offset welcome
mov ah, 9
int 21h
ret
pwelcome endp


enterkey proc
mov dx, 10
mov ah, 2
int 21h
mov dx, 13
mov ah, 2
int 21h
ret                            ; start new line
enterkey endp


printborline proc
PRINT_COLORED_STRING borderline,9
ret
printborline endp
```

```
printborline2 proc
PRINT_COLORED_STRING borderline2,9
ret
printborline2 endp


printborder proc
PRINT_COLORED_STRING border,9
ret
printborder endp


printbwrow proc
PRINT_COLORED_STRING bwrow,4
ret
printbwrow endp

printboard proc
mov ax, 0003h
int 10h

call enterkey
call pwelcome
call enterkey

call printborder
call enterkey
call printborline

PRINT_SHOW_BOARD row1a
call printborline


PRINT_SHOW_BOARD row2a
call printborline


PRINT_SHOW_BOARD row3a

ret
printboard endp

printsolboard proc
mov ax, 0003h
int 10h

call enterkey
call pwelcome
call enterkey
call printborder
call enterkey
call printborline

PRINT_SHOW_BOARD sol1a
call printborline

PRINT_SHOW_BOARD sol2a
call printborline

PRINT_SHOW_BOARD sol3a

ret
printsolboard endp


validateSudoku proc
    mov al, userInputs[0]
    cmp al, '3'
    jne invalidSolution

    mov al, userInputs[1]
    cmp al, '3'
    jne invalidSolution

    mov al, userInputs[2]
    cmp al, '2'
    jne invalidSolution

    mov al, userInputs[3]
    cmp al, '3'
    jne invalidSolution

    mov ah, 09h
    mov bl, 10
    mov cx, 32        ; set color
    int 10h
;next instruction will be colored
    mov dx, offset validMsg
    jmp validationEnd

invalidSolution:
    mov ah, 09h
    mov bl, 10
    mov cx, 26        ; set color
    int 10h
    mov dx, offset invalidMsg

validationEnd:
    mov ah, 09h
    int 21h
    ret
validateSudoku endp
ENDM
```

2.4 Discussion of Implemantation

Core Logic

- **Sudoku Board Representation:** The game board is represented using strings stored in the data segment, allowing dynamic display and easy modification during gameplay.
- **User Input Handling:** The `CAPTURE_INPUT` macro captures user keystrokes, enabling players to input numbers or navigate the game board.
- **Puzzle Validation for 3x3 Grids:** The `validateSudoku` procedure checks the correctness of user solutions for 3x3 puzzles, adhering to Sudoku rules.

User Interface

- **Display Mechanisms:** Macros like `PRINT_COLORED_STRING` and `PRINT_SHOW_BOARD` are used to display the Sudoku grid and messages, incorporating color for enhanced user experience.
- **Cursor Control:** The `MOVE_CURSOR` macro enables dynamic cursor movement, allowing players to interact with the game board intuitively.

Challenges and Limitations

- **Handling 9x9 Sudoku Puzzles:** One of the significant challenges is implementing a validation mechanism for 9x9 puzzles, particularly managing subgrid validations. This complexity has limited the current version to effectively handling only 3x3 grids.

Chapter 3

Performance Evaluation

3.1 Simulation Environment/ Simulation Procedure

The Sudoku solver was tested in a controlled environment using the EMU8086 software for Windows. This setup ensured that the program behaved as it would in an actual 8086 microprocessor environment, providing a reliable platform for development and testing.

3.2 Results Analysis/Testing

3.2.1 Results Analysis

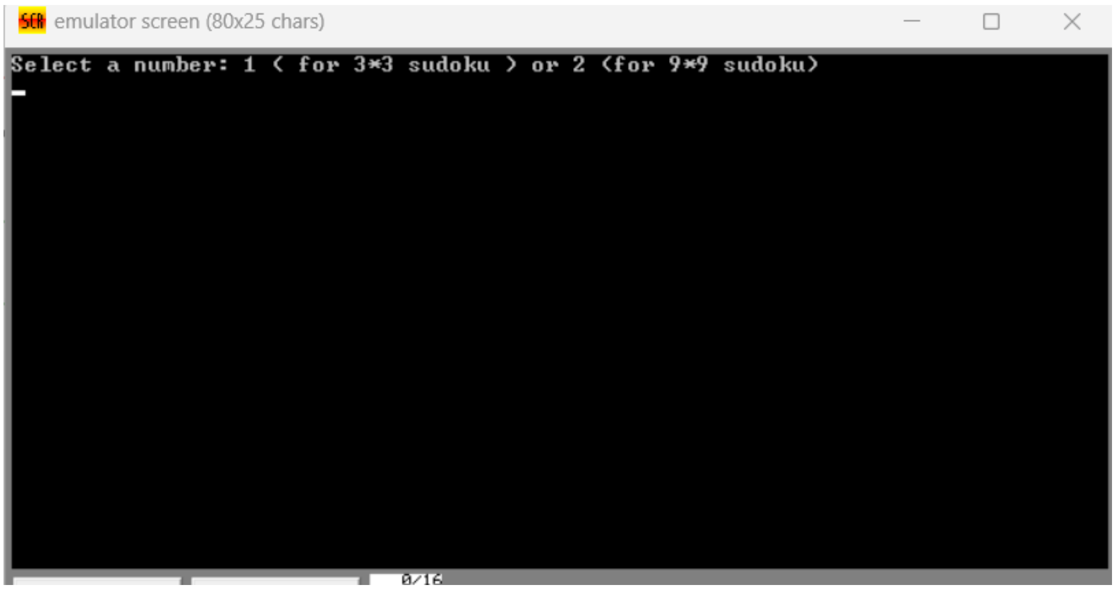
Testing Outcomes

- **3x3 Sudoku Puzzles:** The solver successfully resolved all 3x3 puzzles, demonstrating high accuracy in its algorithm. The solutions matched the known correct answers, confirming the effectiveness of the solver's logic.
- **9x9 Sudoku Puzzles:** The solver's current version does not support the validation of 9x9 puzzles, especially in handling subgrid complexities. This limitation is a focal point for future development.

3.2.2 Output of Sudoku Solver

Sudoku Game Mode Selection Screen

This screenshot shows the initial screen of the Sudoku game application running in an emulator. The text-based interface prompts the user to select the size of the Sudoku puzzle they wish to play. There are two options available: a simpler 3x3 Sudoku grid for a quick game or a standard 9x9 grid for the full Sudoku experience.

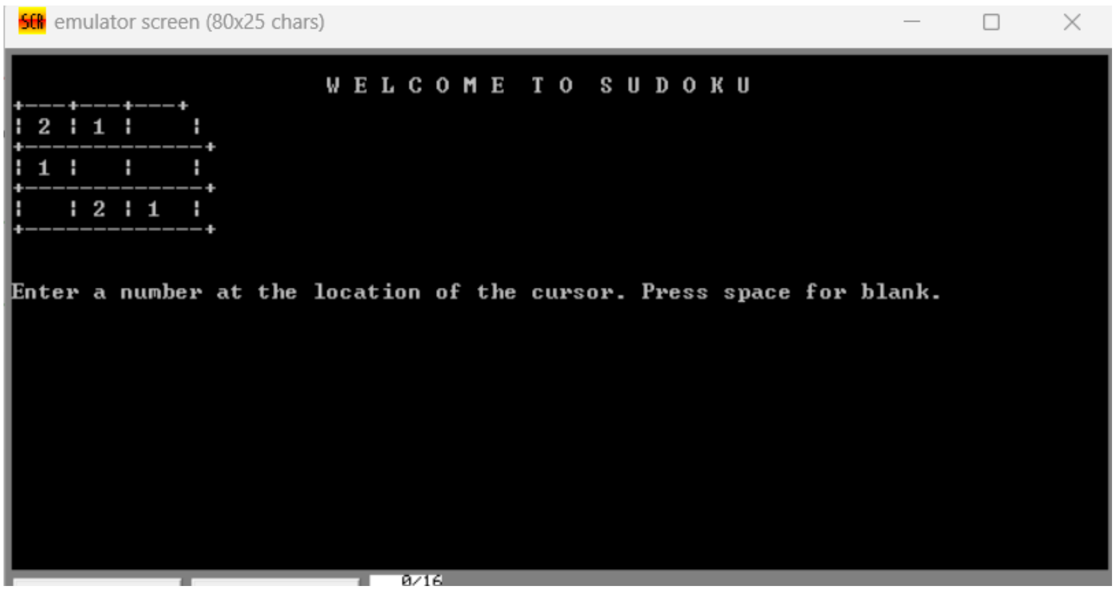


```
Select a number: 1 ( for 3*3 sudoku ) or 2 (for 9*9 sudoku)
_
```

Figure 3.1: Choose your Sudoku challenge: Press ’1’ for a 3x3 puzzle, or ’2’ for the classic 9x9 puzzle.

3x3 Sudoku Puzzle Interface

This image presents the user interface of a 3x3 Sudoku game in progress. The screen prominently displays the welcome message "W E L C O M E T O S U D O K U" at the top. Below it, a partially completed Sudoku grid awaits user input, with some numbers already filled in and others left blank. The interface instructs the user on how to interact with the game: entering numbers at the cursor's location and using the spacebar to leave a cell blank if necessary.



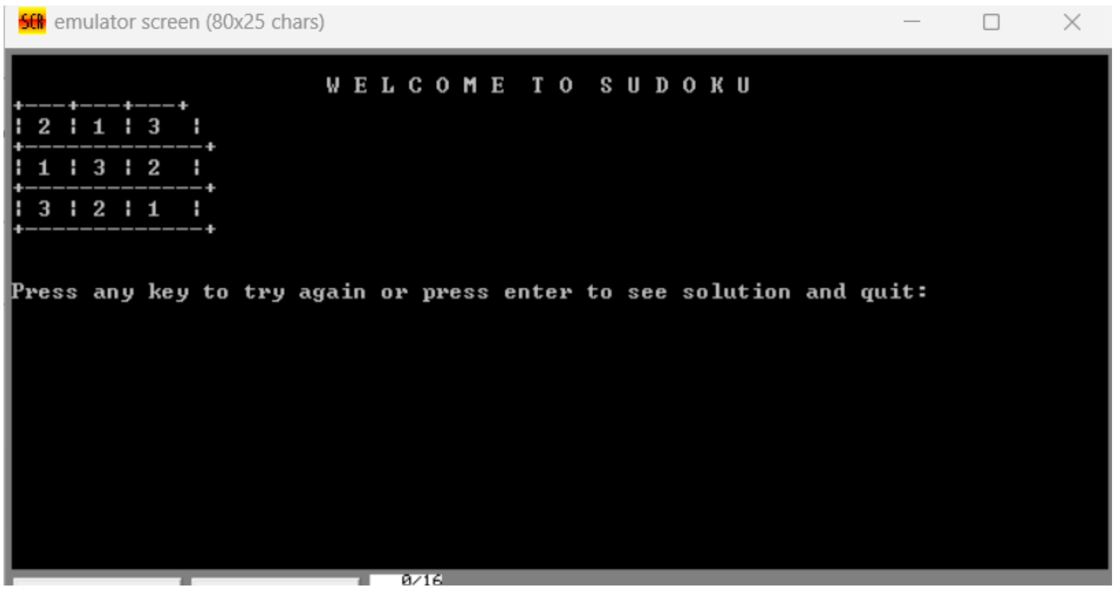
```
                              W E L C O M E   T O   S U D O K U
+---+---+---+
¦ 2 ¦ 1 ¦    ¦
+-------------+
¦ 1 ¦   ¦    ¦
+-------------+
¦   ¦ 2 ¦ 1  ¦
+-------------+


Enter a number at the location of the cursor. Press space for blank.
```

Figure 3.2: Begin your Sudoku journey: Fill in the blanks to complete the puzzle. Navigate to a cell and type a number, or hit space to clear.

Completed 3x3 Sudoku Grid with User Interaction Prompt

This screenshot captures the moment after a player has filled in all the cells of a 3x3 Sudoku grid. The game displays the fully populated puzzle under the "W E L C O M E T O S U D O K U" banner. The numbers are arranged in a way that follows the basic rules of Sudoku, with no repeating numbers in any row or column. Below the grid, the game offers the player an option to either try again (by pressing any key) or to view the solution and exit the game (by pressing enter)



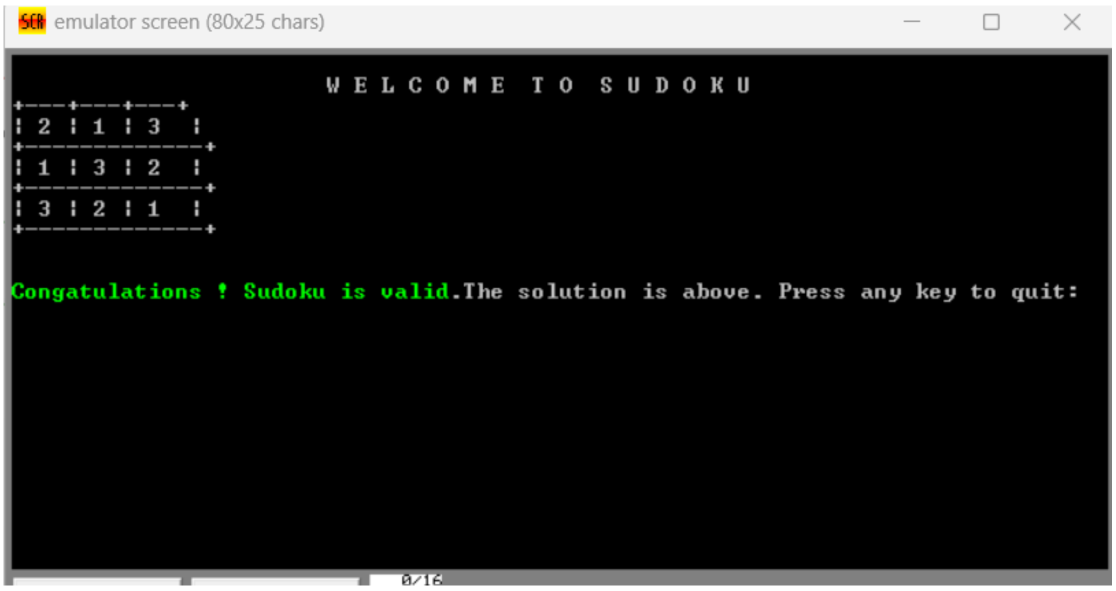
```
                     W E L C O M E  T O  S U D O K U
+---+---+---+
| 2 | 1 | 3  |
+-------------+
| 1 | 3 | 2  |
+-------------+
| 3 | 2 | 1  |
+-------------+


Press any key to try again or press enter to see solution and quit:
```

Figure 3.3: All set! Review your Sudoku skills by pressing any key to retry, or hit enter to reveal the solution and conclude the game

Sudoku Puzzle Validation and Completion Screen

This image showcases the completion screen of the 3x3 Sudoku puzzle within the emulator. The top of the screen reaffirms the warm welcome to the player. Directly below, the completed puzzle is visible, with the correct numbers filled into the grid. The program validates the player's solution, with a congratulatory message "Congratulations! Sudoku is valid." indicating success. This screen serves as the endgame interaction, prompting the player to press any key to exit, marking the end of the gameplay session



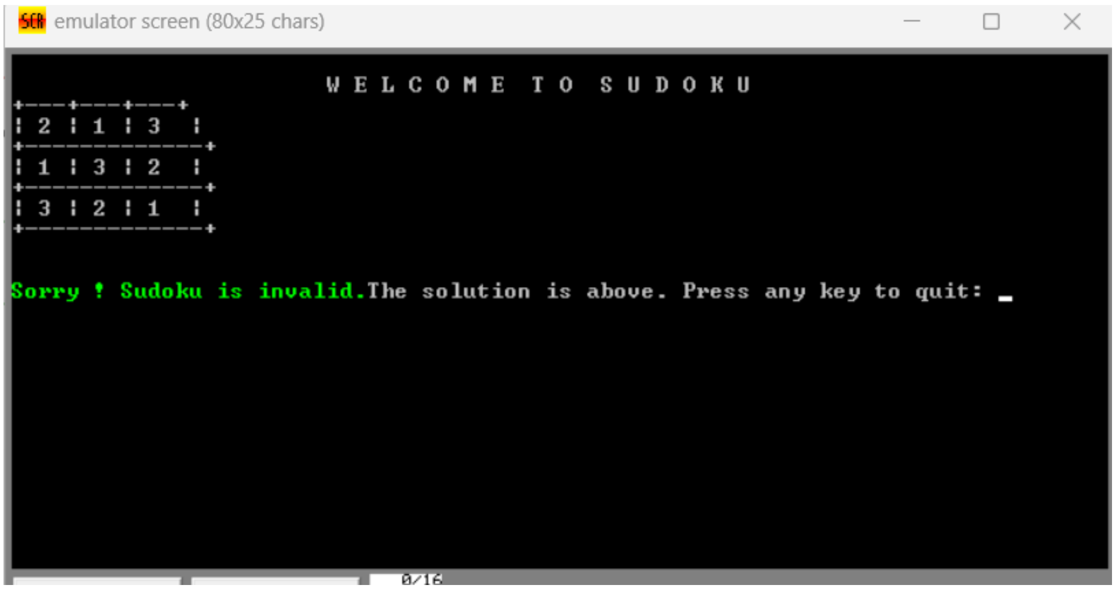
```
                         W E L C O M E   T O   S U D O K U
+----+----+----+
| 2  | 1  | 3  |
+---------------+
| 1  | 3  | 2  |
+---------------+
| 3  | 2  | 1  |
+---------------+

Congatulations ! Sudoku is valid.The solution is above. Press any key to quit:
```

Figure 3.4: Success! Your solution is correct. Celebrate your victory in Sudoku and press any key when you're ready to leave the game.

Sudoku Puzzle Invalid Solution Notification Screen

This screenshot displays the interface of the Sudoku game after a player has attempted to solve the 3x3 grid, but the solution entered is incorrect. The screen maintains the welcoming message at the top, while the filled Sudoku grid sits just below. The application has assessed the inputs and presented a message: "Sorry! Sudoku is invalid." This indicates to the player that the attempted solution does not meet the Sudoku game rules. The prompt also invites the player to press any key to quit, suggesting that this is the final interaction within this game session.



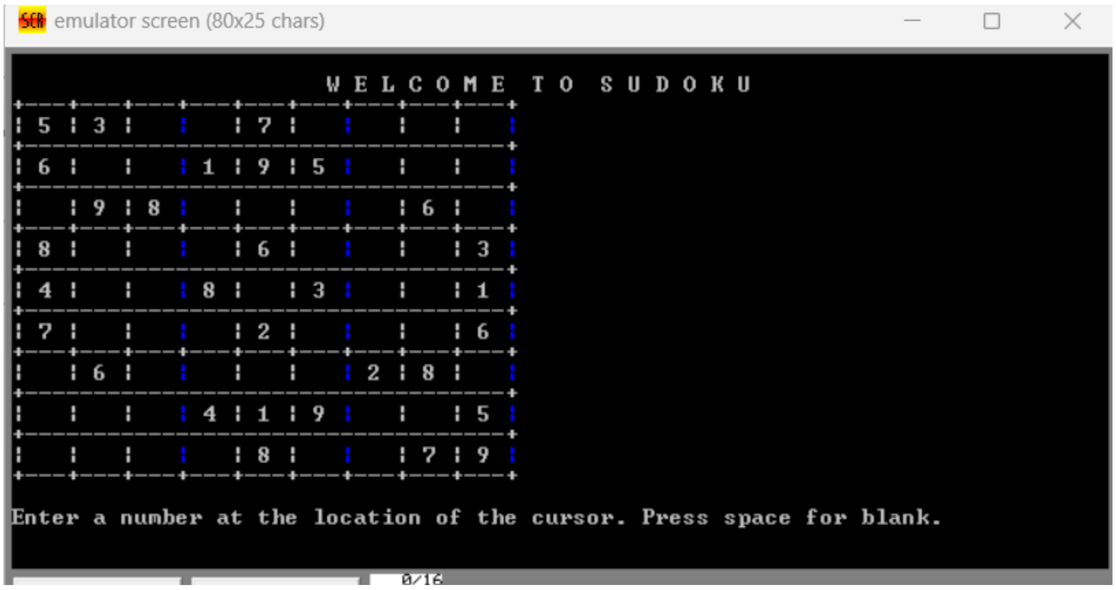
```
                            W E L C O M E  T O  S U D O K U
+----+----+----+
| 2  | 1  | 3  |
+--------------+
| 1  | 3  | 2  |
+--------------+
| 3  | 2  | 1  |
+--------------+


Sorry ! Sudoku is invalid.The solution is above. Press any key to quit: _
```

Figure 3.5: Attempt needs revisiting: Your Sudoku solution didn't match the puzzle's requirements. Press any key to exit and perhaps try again later

9x9 Sudoku Puzzle Input Screen

The image shows a 9x9 Sudoku game board displayed on a text-based emulator interface. The puzzle is partially completed with some numbers already filled in, highlighted by blue lines which might represent selected cells or active areas for input. The welcome message at the top centers the user's attention, and the instruction at the bottom directs the player on how to interact with the game: entering numbers or leaving spaces blank as they work to solve the puzzle.



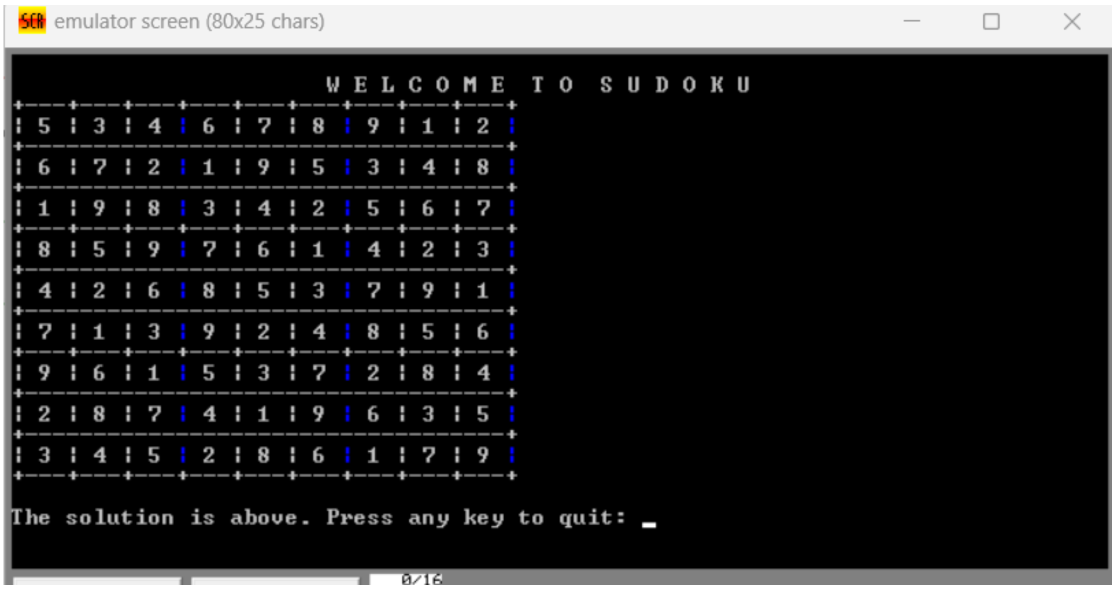
```
                              W E L C O M E  T O  S U D O K U
+---+---+---+---+---+---+---+---+---+
¦ 5 ¦ 3 ¦   ¦   ¦ 7 ¦   ¦   ¦   ¦   ¦
+-----------------------------------+
¦ 6 ¦   ¦   ¦ 1 ¦ 9 ¦ 5 ¦   ¦   ¦   ¦
+-----------------------------------+
¦   ¦ 9 ¦ 8 ¦   ¦   ¦   ¦   ¦ 6 ¦   ¦
+---+---+---+---+---+---+---+---+---+
¦ 8 ¦   ¦   ¦   ¦ 6 ¦   ¦   ¦   ¦ 3 ¦
+-----------------------------------+
¦ 4 ¦   ¦   ¦ 8 ¦   ¦ 3 ¦   ¦   ¦ 1 ¦
+-----------------------------------+
¦ 7 ¦   ¦   ¦   ¦ 2 ¦   ¦   ¦   ¦ 6 ¦
+---+---+---+---+---+---+---+---+---+
¦   ¦ 6 ¦   ¦   ¦   ¦   ¦ 2 ¦ 8 ¦   ¦
+-----------------------------------+
¦   ¦   ¦   ¦ 4 ¦ 1 ¦ 9 ¦   ¦   ¦ 5 ¦
+-----------------------------------+
¦   ¦   ¦   ¦   ¦ 8 ¦   ¦   ¦ 7 ¦ 9 ¦
+---+---+---+---+---+---+---+---+---+

Enter a number at the location of the cursor. Press space for blank.
```

Figure 3.6: Engage with a classic Sudoku challenge: Navigate through the 9x9 grid, input numbers or leave blanks as you strategize your way to completion

9x9 Sudoku Puzzle Completed Board Display

This screenshot captures a completed 9x9 Sudoku grid displayed in a text-based emulator. The entire puzzle is filled with numbers, adhering to the standard Sudoku rule of not repeating a number in any row, column, or 3x3 subgrid. The puzzle is enclosed within a well-defined border, with the subgrids distinguished by blue lines. A prompt below the puzzle informs the user that the solution is presented and offers the option to quit by pressing any key.



```
                      W E L C O M E  T O  S U D O K U
+---+---+---+---+---+---+---+---+---+
¦ 5 ¦ 3 ¦ 4 ¦ 6 ¦ 7 ¦ 8 ¦ 9 ¦ 1 ¦ 2 ¦
+-----------------------------------+
¦ 6 ¦ 7 ¦ 2 ¦ 1 ¦ 9 ¦ 5 ¦ 3 ¦ 4 ¦ 8 ¦
+-----------------------------------+
¦ 1 ¦ 9 ¦ 8 ¦ 3 ¦ 4 ¦ 2 ¦ 5 ¦ 6 ¦ 7 ¦
+---+---+---+---+---+---+---+---+---+
¦ 8 ¦ 5 ¦ 9 ¦ 7 ¦ 6 ¦ 1 ¦ 4 ¦ 2 ¦ 3 ¦
+-----------------------------------+
¦ 4 ¦ 2 ¦ 6 ¦ 8 ¦ 5 ¦ 3 ¦ 7 ¦ 9 ¦ 1 ¦
+-----------------------------------+
¦ 7 ¦ 1 ¦ 3 ¦ 9 ¦ 2 ¦ 4 ¦ 8 ¦ 5 ¦ 6 ¦
+---+---+---+---+---+---+---+---+---+
¦ 9 ¦ 6 ¦ 1 ¦ 5 ¦ 3 ¦ 7 ¦ 2 ¦ 8 ¦ 4 ¦
+-----------------------------------+
¦ 2 ¦ 8 ¦ 7 ¦ 4 ¦ 1 ¦ 9 ¦ 6 ¦ 3 ¦ 5 ¦
+-----------------------------------+
¦ 3 ¦ 4 ¦ 5 ¦ 2 ¦ 8 ¦ 6 ¦ 1 ¦ 7 ¦ 9 ¦
+---+---+---+---+---+---+---+---+---+

The solution is above. Press any key to quit: _
```

Figure 3.7: Full Sudoku laid out: Every cell of the 9x9 grid is filled in with careful consideration. Press any key to exit and celebrate your accomplishment!

Chapter 4

Conclusion

The Sudoku solver project has successfully demonstrated the use of 8086 assembly language in creating an accurate and efficient tool for solving 3x3 Sudoku puzzles. It has not only served as a functional solver but also as an educational platform to illustrate complex problem-solving strategies.

4.1 Achievements

- The solver consistently provides correct solutions for 3x3 Sudoku puzzles.
- An intuitive and responsive user interface enhances the user experience.

4.2 Limitations and Challenges

- The solver's current inability to handle 9x9 Sudoku puzzles due to subgrid complexity.
- The need for advanced logic to manage larger puzzles within the assembly language constraints.

4.3 Recommendations for Future Work

Future enhancements should aim to:

- Develop an algorithm that can solve 9x9 puzzles, possibly using backtracking or constraint propagation.
- Optimize the solver for better computational efficiency as it scales to more complex puzzles.
- Improve the user interface with additional features like hint generation and step-by-step solution processes.

4.4 Final Thoughts

The project underscores the potential of assembly language to tackle practical challenges. With continued development, it is poised to become an even more versatile tool for Sudoku solving and educational insight into programming logic.

Chapter 5

Importnat Link

5.0.1 Github Link

Project Github Link

5.0.2 Reference

Reference for the project :

AtCoder Problems

Stackoverflow