Snakes and Ladders - An Assembly Language Game

Semester Project Report

Session 2025-2026

Submitted by:

Ahmer Sultan
NUM-BSCS-2024-03

Asad Ullah Khan
NUM-BSCS-2024-17

CSC-241 Computer Organization and Assembly Language

Department of Computer Science
Namal University Mianwali

January 15, 2026

Contents

1 Introduction

The purpose of this Computer Organization and Assembly Language project is to show the basic ideas of low-level programming by making an interactive board game. By creating a game in assembly language, we connect the gap between theory of computer design and real software making. This gives hands-on experience with machine-level programming.

The scope of this project covers several important areas of computer science:

1. **Computer Architecture Understanding**: Through direct use of registers, memory, and hardware, the project shows how high-level programming ideas become machine-level operations.

2. **System-Level Programming**: The project looks at important parts of system programming: - Direct memory access and management - I/O device interaction - Program flow control at machine level

3. **Educational Value**: The game shows: - CPU architecture and instruction set usage - Memory management - I/O systems - Program flow control

2 Problem Statements

The project aims to create a fully working Snakes and Ladders game in assembly language with these requirements:

2.1 Core Gaming Requirements

2.1.1 Player Mechanics:

- Two-player turn-based gameplay
- Movement based on dice values
- Collision detection with snakes and ladders
- Win condition detection

2.1.2 Game Elements:

- 10×10 game board with 100 positions
- 7 snakes with specific start-end positions
- 6 ladders with specific start-end positions
- Visual display of board elements

2.1.3 Game Flow:

- Players must roll 6 to start the game
- Turn-based system with proper switching
- Position tracking for both players
- Game history display

2.2 Technical Requirements

2.2.1 System Integration:

- Irvine32 library for console operations
- Keyboard input handling
- Screen buffer manipulation
- Random number generation for dice

2.2.2 User Interface:

- Welcome screen with game title
- Player name input system
- Game board display with colors
- Status display showing player info
- Information box showing snakes and ladders

3 Proposed Solution

3.1 Flow Diagram

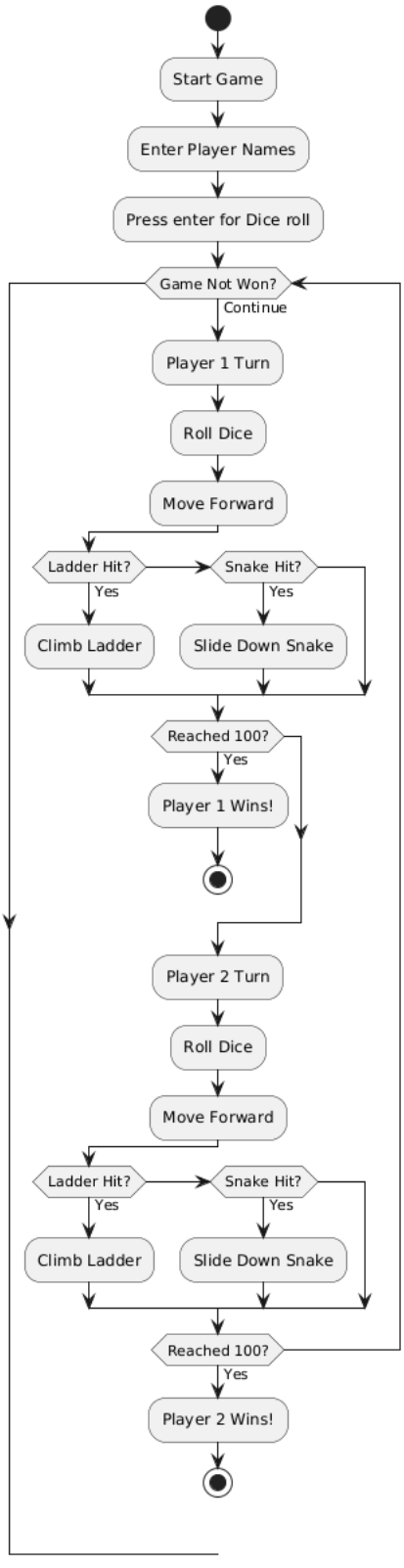

Figure 1: Game Flow Diagram

3.2 Pseudo Code

Listing 1: Pseudo Code for Main Game Loop

```
PROCEDURE Main
BEGIN
    // System Initialization
    Initialize_Display
    Show_Welcome_Screen
    Get_Player_Names

    // Game Setup
    Initialize_Board
    Initialize_Players

    // Game Loop
    REPEAT
        FOR each player DO
            Display_Current_Status
            Prompt_Dice_Roll
            Roll_Dice_With_Animation
            Process_Movement
            Check_Snakes_Ladders
            Update_Position
            Display_Board
            IF position == 100 THEN
                Declare_Winner
                EXIT
            END IF
        END FOR
    UNTIL game_ended

    Cleanup_And_Exit
END
```

3.3 Assembly Code Implementation

3.3.1 Data Section

Listing 2: Data Section Definitions

```
INCLUDE C:\irvine\Irvine32.inc
INCLUDE C:\irvine\Macros.inc

.data

; Game messages
dicePromptMsg       BYTE "ROLL THE DICE FOR",0
diceRollingMsg      BYTE "ROLLING DICE",0
diceResultMsg       BYTE "Dice roll=",0
ladderMsg           BYTE "LADDER CLIMB!!",0
snakeMsg           BYTE "X SNAKE X",0
congratsMsg         BYTE "CONGRATULATIONS!!!! ",0
winGameMsg          BYTE " WON THE GAME!!!",0
positionMsg         BYTE "Current position=",0

; Player data
```

```
player1Name          BYTE 50 DUP(0)
player2Name          BYTE 50 DUP(0)
player1Position      DWORD 0
player2Position      DWORD 0
player1Started       DWORD 0
player2Started       DWORD 0

; Dice history
player1DiceHistory   DWORD 10 DUP(0)
player2DiceHistory   DWORD 10 DUP(0)
historyCount1        DWORD 0
historyCount2        DWORD 0

; Board display elements
topBorderLeft        BYTE 201,205,205,205,205,205,205,0
topBorderMid         BYTE 203,205,205,205,205,205,205,0
topBorderRight       BYTE 187,13,10,0
midBorderLeft        BYTE 204,205,205,205,205,205,205,0
midBorderMid         BYTE 206,205,205,205,205,205,205,0
midBorderRight       BYTE 185,13,10,0
botBorderLeft        BYTE 200,205,205,205,205,205,205,0
botBorderMid         BYTE 202,205,205,205,205,205,205,0
botBorderRight       BYTE 188,13,10,0
vertChar             BYTE 186,0
boxSpaces            BYTE "      ",0

; Game element labels
snake1Label          BYTE "S1",0
snake2Label          BYTE "S2",0
snake3Label          BYTE "S3",0
snake4Label          BYTE "S4",0
snake5Label          BYTE "S5",0
snake6Label          BYTE "S6",0
snake7Label          BYTE "S7",0
ladder1Label         BYTE "L1",0
ladder2Label         BYTE "L2",0
ladder3Label         BYTE "L3",0
ladder4Label         BYTE "L4",0
ladder5Label         BYTE "L5",0
ladder6Label         BYTE "L6",0
Win                  BYTE "WIN",0

; Colors
boxColor1     BYTE 10101111b  ; White background, black text
boxColor2     BYTE 10101111b  ; Yellow background, black text
digitColor   BYTE 00001111b  ; White text, black background
```

3.3.2 Main Program Structure

Listing 3: Main Program Structure

```
.code
main PROC
    call clrscr
    ; Display welcome screen
    mov edx, OFFSET introMsg
    call WriteString
```

```
7
8       ; Get player names
9       call GetPlayerNames
10
11      ; Initialize game
12      call InitializeGame
13
14      ; Main game loop
15      call GameMainLoop
16
17      ; Exit
18      exit
19  main ENDP
```

3.3.3 Game Board Display System

Listing 4: Game Board Display

```
1   DisplayGameBoard PROC
2       ; Set cursor position
3       mov dh, 0
4       mov dl, 0
5       call Gotoxy
6
7       ; Draw top border
8       mov edx, OFFSET topBorderLeft
9       call WriteString
10      mov ecx, 9
11  top_loop:
12      mov edx, OFFSET topBorderMid
13      call WriteString
14      loop top_loop
15      mov edx, OFFSET topBorderRight
16      call WriteString
17
18      ; Draw 10x10 board with alternating colors
19      ; ... (detailed board drawing code)
20
21      ; Display numbers, snakes, ladders, and player positions
22      call DisplayBoardNumbers
23      call DisplayLadders
24      call DisplaySnakes
25      call DisplayPlayerPositions
26
27      ret
28  DisplayGameBoard ENDP
```

3.3.4 Player Movement System

Listing 5: Player Movement Processing

```
1   ProcessPlayerMove_P1 PROC
2       ; Check if player has started
3       .IF player1Started == 0
4           .IF eax == 6  ; Need 6 to start
5               mov player1Started, 1
```

```
            mov player1Position, 1
            mov edx, OFFSET ladderMsg
            call WriteString
        .ELSE
            mov edx, OFFSET shakeMsg
            call WriteString
        .ENDIF
    .ELSE
        ; Move player based on dice value
        add player1Position, eax
        .IF player1Position <= 100
            ; Check for snakes and ladders
            call CheckLadderP1
            call CheckSnakeP1
        .ELSE
            ; Move exceeds board, cancel move
            sub player1Position, eax
        .ENDIF
    .ENDIF
    ret
ProcessPlayerMove_P1 ENDP
```

3.3.5 Snake and Ladder Detection

Listing 6: Snake Detection for Player 1

```
CheckSnakeP1 PROC
    mov eax, player1Position
    .IF eax == 25
        mov player1Position, 5   ; Snake from 25 to 5
        mov edx, OFFSET snakeMsg
        call WriteString
    .ELSEIF eax == 34
        mov player1Position, 1   ; Snake from 34 to 1
        mov edx, OFFSET snakeMsg
        call WriteString
    ; ... (other snake positions)
    .ENDIF
    ret
CheckSnakeP1 ENDP
```

4 Results and Findings

4.1 Game Features Implementation

4.1.1 Menu System:

- Working welcome screen with game title
- Player name input system
- Game board with proper display

4.1.2 Game Mechanics:

- Two-player turn-based system working
- Dice roll with animation
- Proper movement based on dice values
- Snake and ladder detection working
- Win condition detection

4.1.3 Board Design:

- 10×10 board with numbers 100 to 1
- 7 snakes placed at specific positions
- 6 ladders placed at specific positions
- Color-coded board for better visibility

4.2 Technical Achievements

4.2.1 Memory Management:

- Efficient use of variables for game state
- Stack used for procedure calls
- Memory-efficient board representation

4.2.2 Display System:

- Colorful board display
- Proper positioning of game elements
- Status display updating in real-time

4.3 User Interface

4.3.1 Welcome Screen:

Figure 2: Game Welcome Screen

4.3.2 Game Board Display:

Figure 3: Game Board with Players

4.3.3 Player Win Screenshot:

Figure 4: Game Board with Players

4.3.4 In-Game UI:

- Current player turn display
- Dice roll results
- Player position tracking
- Dice history display
- Snake and ladder information

4.4 Functionality Achievements

4.4.1 Successful Implementation:

- Complete two-player game system
- Working dice roll mechanism
- Proper snake and ladder logic
- Win condition detection
- Game history tracking

4.4.2 Menu System Functionality:

- Welcome screen with title art
- Player name input
- Game board with all elements
- Status display
- Information display

4.5 Code Efficiency Analysis

4.5.1 Memory Management:

- Efficient register usage for positions
- Minimal memory for game state
- Optimized board drawing

4.5.2 Performance Analysis:

- Quick dice roll generation
- Fast board updates
- Responsive input handling

4.5.3 Optimization Achievements:

- Efficient collision detection
- Minimal screen redrawing
- Quick player position updates

4.6 Testing Observations

4.6.1 Game Mechanics:

- Dice roll works correctly every time
- Player movement accurate
- Snake detection works perfectly
- Ladder detection works perfectly
- Win condition triggers correctly

4.6.2 User Interface:

- Board displays correctly
- Player positions update properly
- Status information accurate
- Colors display correctly

4.6.3 Input Handling:

- Player names accept correctly
- Dice roll prompt works
- Game flow controls working

4.7 Limitations and Areas for Improvement

4.7.1 Current Limitations:

- Fixed board size (10×10)
- Pre-defined snake and ladder positions
- Two-player only
- No save/load game feature
- Basic graphics (text-based)

4.7.2 Performance Considerations:

- Board redrawing could be optimized
- Memory usage could be reduced further
- Input validation could be improved

5 Conclusion

Making Snakes and Ladders in assembly language has been a challenging but rewarding experience. We started with learning basic assembly instructions and ended with a complete working game. This project helped us understand how computers work at the lowest level.

We learned how to:

- Work with memory and registers directly
- Handle input and output in assembly
- Create game logic using low-level code
- Display graphics using text characters
- Manage game state efficiently

The most satisfying part was seeing our game working perfectly and watching players enjoy it. Every bug we fixed and every feature we added taught us something new about computer organization.

5.1 Future Improvements

5.1.1 Technical Enhancements:

- Add sound effects using Windows API
- Implement save/load game feature
- Add more players (3-4 players)
- Create different board designs

5.1.2 Gameplay Features:

- Add power-up items
- Implement computer player (AI)
- Add different game modes
- Create level progression

6 References

1. Visual Studio, ClickUp and V0 for code optimization and project management
2. Irvine32 Library Documentation
3. Computer Organization and Assembly Language course notes (Namal University Mianwali)
4. Online assembly language tutorials
5. PlantUML for workflow diagram generation