**National University Computer & Emerging Sciences – Chiniot FSD Campus**

Project Documentation Manual

**Project Title:** Pong Ping Game in NASM

**Course:** EE2003 – Computer Organization and Assembly Language

**Instructor:** Dr Muhammad Usama Assoc. Prof.

**Date of Submission:** Dec, 8 2024

**Section:** BCS-3A

|  |
| --- |
| Group Members: |
| Member 1: Shawaiz Shahid 23P-0599 |
| Member 2: Muneeb Ahmad 23F-0663 |

# ****Ping Pong Game Project Report****

### ****Table of Contents****

1. **Introduction**
2. **Game Overview**
3. **Implementation Details**
   1. Core Features
   2. Additional Features
4. **Challenges and Solutions**
5. **Testing and Debugging Process**
6. **User Guide**
7. **Conclusion**
8. **Appendix**
9. **References**

### ****1. Introduction****

**Project Summary**:  
This project involves the development of a Ping Pong game using NASM (Netwide Assembler) for Intel 8086 assembly code, executed in DOSBox. The game features two player-controlled paddles, a bouncing ball, and a scoring system, replicating the classic Pong game mechanics. The game was designed to function in a DOSBox environment, simulating a retro gaming experience.

**Objectives**:

* **Develop a playable Pong-style game in NASM assembly language**: The primary goal is to build a simple but functional game using low-level assembly programming.
* **Demonstrate skills in assembly programming**: This project highlights the usage of keyboard input, screen rendering, and game logic in assembly.
* **Add optional features**: To enhance the user experience, optional features like background graphics and sound effects are included.

### ****2. Game Overview****

**Game Description**:  
The game simulates the classic Pong game, where two players control paddles and attempt to hit a ball back and forth. The goal is to prevent the ball from passing the player’s paddle while scoring points when the opponent fails to return the ball. The game ends when one player reaches a predefined winning score.

**Game Rules**:

* **Paddle Control**: Each player can control their paddle’s vertical position using the keyboard keys.
* **Scoring Mechanism**: Points are awarded when the ball crosses one player's boundary. The score is displayed on the screen.
* **Winning Criteria**: The first player to reach a predefined number of points wins the game. After winning, the player is shown a message, and the option to restart the game is presented.
* **Game Restart**: After the game ends, players are prompted to restart the game.

**Game Flow**:

1. **Initialization**: The game begins by initializing variables, such as paddle and ball positions, and clearing the screen.
2. **Input Detection**: The program detects key inputs using the INT 16H interrupt to move the paddles.
3. **Movement and Collision**: The ball moves across the screen, bouncing off paddles and walls. The ball’s direction changes when it collides with objects.
4. **Scoring**: When the ball passes the boundaries, points are awarded, and the score is updated.
5. **Game End**: Once a player reaches the winning score, a message is displayed, and players can choose to restart.

### ****3. Implementation Details****

**Development Environment**:

* **Assembler**: NASM (Netwide Assembler)
* **Emulator**: DOSBox (to run Intel 8086 assembly code)

#### ****Core Features****:

**1.Player-Controlled Paddles**:

The paddles are controlled using the keyboard. The INT 16H interrupt is employed to capture key presses for both Player 1 and Player 2. ‘W’ or ‘w’ and ‘S’ and ‘s’ for player 1. Up and down for player 2.

The paddle position is updated based on the input, and movement is restricted to the screen boundaries to prevent the paddle from going out of bound.

2.**Ball Movement**:

The ball starts at a specific position on the screen, moving in a diagonal direction. The ball’s position is updated frame by frame, using direction variables (e.g., dx, di) to control its movement.

The ball’s movement is adjusted each frame to ensure it moves continuously across the screen.

3.**Reflection Logic**:

The ball bounces off the top and bottom walls. The reflection logic checks the ball’s current position relative to the screen edges and reverses its direction if it collides with a wall or a paddle.

For paddle-ball collisions, the ball's movement direction is recalculated to ensure it continues bouncing in a realistic manner.

**4.Score Counting**:

Points are tracked by comparing the ball’s position to the player’s boundaries. If the ball crosses into the opponent’s area, the opposing player scores a point. The current score is displayed on the screen, updating after each point.

**5.Game End Condition**:

The game checks the score after each point to determine if either player has won. When a player reaches the winning score, the game ends, and a winner message is displayed. Players are prompted to restart the game if desired.

#### ****Additional Features****:

1. **Backgrounds**:

A simple scrolling background is implemented using patterns to create a dynamic visual effect. This background does not interfere with the gameplay, as it scrolls in the background while the game continues.

1. **Sound Effects**:

Sound effects are triggered for specific events, such as when the ball bounces off the paddle or when a point is scored. Simple tones are generated using the PC speaker.

### ****4. Challenges and Solutions****

**Challenge**: Handling simultaneous paddle and ball movement without performance issues.

* **Solution**: The code is optimized by separating the logic for paddle and ball movement, ensuring that the program can handle both at once without slowing down the game. The keyboard input is processed independently of the game loop to avoid delays.

**Challenge**: Managing graphics rendering in a text-based environment.

* **Solution**: DOSBox’s video memory was utilized to render basic graphics, like the paddles and ball, pixel by pixel. Efficient handling of screen clearing and updating ensures smooth gameplay.

### ****5. Testing and Debugging Process****

The game underwent extensive testing to ensure it worked as expected. Key areas tested include:

* **Paddle Movement**: Ensuring the paddles move correctly in response to keyboard inputs.
* **Ball Movement and Collision**: Verifying that the ball bounces off the walls and paddles properly and changes direction at the right times.
* **Scoring**: Ensuring points are awarded correctly and the score is updated on the screen.
* **Game Restart**: Testing the restart functionality to ensure the game correctly resets after a player wins.

The game was debugged using DOSBox’s built-in debugging tools, inspecting the registers and memory to track issues related to ball movement and collision detection.

### ****6. User Guide****

**Getting Started**:

1. Install DOSBox on your system and set it up for running 16-bit programs.
2. Assemble the NASM code into a .COM or .EXE file.
3. Run the file in DOSBox using the command: game.exe.

**Controls**:

* **Player 1**: Use the W and S keys to move the paddle up and down.
* **Player 2**: Use the Up and Down arrow keys for paddle control.
* **Pause/Resume**: Press P to pause or resume the game.
* **Restart Game**: Press R after the game ends to restart the game.

**Gameplay Tips**:

* Quick and precise paddle movements give you better control.
* Try to anticipate the ball’s direction to increase your chances of returning it.

### ****7. Conclusion****

This project was a valuable learning experience in assembly programming, particularly for creating a real-time interactive game. It improved my understanding of game mechanics, keyboard input handling, and low-level graphics programming. While the game works as expected, there are areas for improvement, such as enhancing the graphical fidelity and adding more complex features like AI opponents.

### ****8. Appendix****

**Screenshots**:

A screen shot of a video game menu

Description automatically generated

A screen shot of a computer

Description automatically generated

A black and white image of a person's face

Description automatically generated

A screen shot of a black screen

Description automatically generated

A black screen with a white line

Description automatically generated with medium confidence

A screen shot of a game

Description automatically generated

### ****9. References****

* Assembly Programming Book **"Programming from the Ground Up" by Jonathan Bartlett**
* NASM documentation: <https://www.nasm.us>
* DOSBox documentation: <https://www.dosbox.com/wiki/Main_Page>
* YouTube Channel **Programming Dimension**
* Virtual University Handouts.