**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:** Sir Ali Hamza

**Date of Submission:** 7/12/2024

**Section:** 3 D

|  |
| --- |
| Group Members: |
| Member 1: [Waleed Afzal, 23P-0566] |
| Member 2: [Muhammad, 23F-0577] |

**2. Table of Contents**

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

**3. Introduction**

**Project Summary:**  
This project is a simple Pong game written in assembly language, designed to run on x86 computers. The game features two players who control paddles to hit a ball back and forth, trying to score points.

**Objectives:**

* **Graphics**: The game uses basic graphics to display the ball and paddles on the screen.
* **Ball Movement**: The ball bounces off walls and paddles. If it passes a paddle, the other player scores a point.
* **Controls**: Players move their paddles using keyboard keys.
* **Score**: Points are tracked for each player.
* **Game Over**: The game ends when a player reaches the set score, with an option to restart or exit.
* **Pause**: The game can be paused and resumed using the 'P' key.

**4. Game Overview *(Attach relevant code parts & screenshots while explaining):***

**Game Description:**  
This is simple two-player game where each player controls a paddle to hit a ball back and forth across the screen. The objective is to prevent the ball from passing your paddle while trying to get it past your opponent’s paddle. Each time a player fails to intercept the ball, the opponent scores a point. The game continues until one player reaches a predefined winning score.

**Game Rules:**  
🡪 **Control the Paddles**:

* **Player 1**: Controls the left paddle using the **'W'** key to move up and the **'S'** key to move down.
* **Player 2**: Controls the right paddle using the **'Up Arrow'** key to move up and the **'Down Arrow'** key to move down.

🡪 **Scoring Mechanism**:

* When the ball passes a player's paddle, the opponent scores one point.
* Points are displayed on the screen for both players.

🡪 **Winning Criteria**:

* The game is played until one player reaches a set score (e.g., 5 points).
* Once a player wins, the game displays the winner and offers the option to restart or exit the game.

🡪 **Game Restart**:

* After a player wins, the game can be restarted, or the player can choose to quit.

**Game Flow:**  
🡪 **Initialization**:

* Set up the game environment, including initializing the paddles, ball, and scores.
* Assign starting positions for Player 1 and Player 2 paddles.
* Set the ball to start at the center of the screen with a random direction.
* Set initial scores for both players to 0.

🡪 **Game Loop**: The game loop runs continuously and includes the following steps:

* **Detect Key Inputs**:
  + Check for key presses from both players to control the paddles.
  + Player 1 uses the **'W'** and **'S'** keys for up and down movement.
  + Player 2 uses the **'Up Arrow'** and **'Down Arrow'** keys for up and down movement.
* **Update Paddle Positions**:
  + Based on the key inputs, update the positions of the paddles to move them up or down within the screen boundaries.
* **Update Ball Position**:
  + Move the ball in its current direction (either up/down, left/right).
  + If the ball hits the top or bottom edges of the screen, reverse its vertical direction.
  + If the ball hits the left or right edges, check if a player scored a point. If so, update the score and reset the ball to the center.
* **Ball Collision Detection**:
  + Check if the ball collides with either player's paddle.
  + If the ball hits the paddle, reverse the ball's horizontal direction and give it a slight speed boost to make the game more challenging.
* **Check for Scoring**:
  + If the ball goes past either paddle, award a point to the opponent and reset the ball to the center of the screen.

🡪 **Render Screen**:

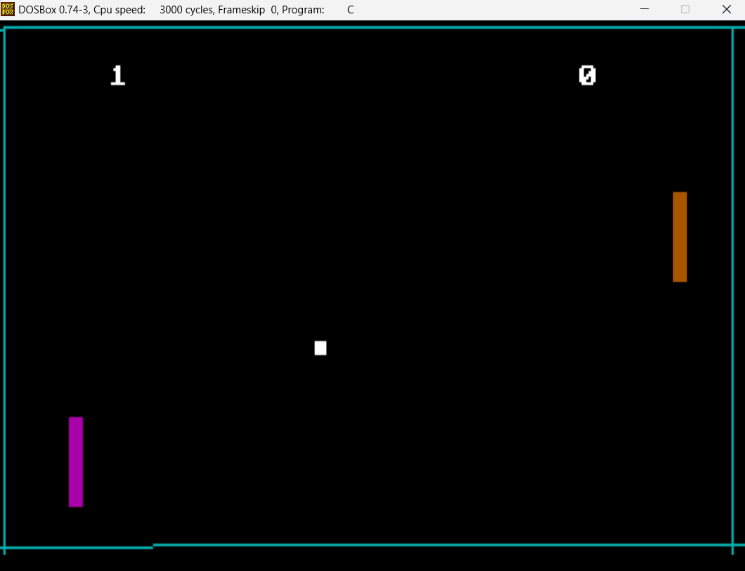
* After updating the game state, render the paddles, ball, and updated score to the screen.
* Display the scores of both players.

🡪 **Winning Condition**:

* Check if any player has reached the winning score (e.g., 5 points). If so, display the winner and offer the option to restart or quit the game.

🡪 **Restart or Quit**:

* If the player chooses to restart, reset the game to its initial state.
* If the player chooses to quit, exit the game loop and close the game.

🡪 **Output Screenshots**:

A screen shot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated



**5. Implementation Details**

**Development Environment:**  
List the tools and software used, such as:

* **Assembler:** NASM (Netwide Assembler)
* **Emulator:** DOSBox for running Intel 8086 assembly code

**Core Features:**

* 1. **Player-Controlled Paddles:**

Code Logic for Moving Paddles: The player controls the paddles using keyboard inputs. When the player presses certain keys, the paddle moves up or down.

Using INT 16H for Keyboard Input: In assembly language, INT 16H is used to capture key presses. This interrupt checks for a key being pressed and tells the program which key was pressed. Based on this input, the paddle position is updated by moving it either up or down.

* 1. **Ball Movement:**

Algorithm for Ball Movement: The ball starts in the center of the screen and moves in a random direction. Every time the game updates, the ball’s position changes by a small amount, moving either horizontally or vertically. The direction is stored as variables, so the ball continues moving in the right direction until it hits something like the wall or a paddle.

* 1. **Reflection Logic:**

Ball Bouncing: When the ball hits the top or bottom of the screen, it bounces back. The ball also bounces off the paddles. To make this happen, the program checks if the ball’s position touches the paddle or the wall, and when it does, the ball’s direction is reversed, making it go the other way.

* 1. **Score Counting:**

Tracking the Score: Every time the ball crosses the screen boundary (either past Player 1’s or Player 2’s paddle), the opponent gets a point. The score is updated and displayed after each point. The program keeps track of the score for both players.

* 1. **Game End Condition:**

Checking for a Winner: The game checks if a player reaches the winning score (e.g., 5 points). If a player wins, the program shows a message saying who won. After that, the player can choose to restart the game or quit. If they choose to restart, the game resets everything back to the starting state.

**Additional Features (Optional):**

If applicable, describe any extra features you implemented, such as:

* **Backgrounds:** Simple Black Background
* **Sound Effects:** Implemented the sound effects whenever the player gets a score.
* **Enhanced Graphics:** Added Borders and Make the paddles colorful.

**6. Challenges and Solutions**

Identify any major challenges encountered during development and the solutions you applied, such as:

* **Example Challenge:** Detecting simultaneous problems after pause and resume logic.
* **Solution:** We tried to add delay and fix it with delay loops.

**7. User Guide**

**Getting Started:**

* To Setup: Save the code in any file (e.g. pong.asm) with the extension of (.asm)
* Then run the file in DOSBOX and compile it with .com file

**Controls:**

* List all key controls for the game, such as:
  + **Paddle Control:** [W & S for Player 1, Up Arrow & Down Arrow for Player 2]
  + **Pause/Resume:** [‘P’ or ‘p’ for Pause and Unpause]
  + **Restart Game:** [‘R’ or ‘r’ to Restart]

**Gameplay Tips:**

* Tips for players, like “Keep pressing the key for better movement of paddles.”

**8. Conclusion**

This project helped develop key skills in assembly programming, particularly in handling input, screen rendering, and game mechanics. It deepened my understanding of how games work at a low level, including collision detection and score tracking.

**Issues:**

* There might be delays in key input detection and screen flickering.
* The game could feel basic without advanced graphics or sound.

**Improvements:**

* Smoother ball movement, better collision detection, and added sound effects.
* Enhanced graphics and code optimization for improved performance.

**9. Appendix**

Include any additional information here, such as:

* **Code Snippets:** Any complex or important code sections that were not fully explained in previous sections.

**10. References**

List any resources used, such as:

* **Tutorials and Guides:** References to any online guides, GitHub repositories or textbooks used for assembly programming or game development basics.
* **References Links**:
* <https://en.wikipedia.org/wiki/BIOS_color_attributes>
* <https://stanislavs.org/helppc/int_10-0.html>
* <https://programmingethicalhackerway.blogspot.com/2015/07/concept-of-interrupt-in-assembly.html>
* <https://en.wikipedia.org/wiki/BIOS_color_attributes>
* <https://faculty.kfupm.edu.sa/COE/shazli/coe205/Help/interrupts.html>