Computer Organization and Assembly Language

Project Title: Zuma Game

Section: E

Fatima Ishtiaq, 23i-0696

Table of content

1. Introduction

Zuma is a classic puzzle game where players shoot colored balls to form groups of three or more balls of the same color. The objective is to prevent the balls from reaching the end of a path by clearing them. In this implementation, the player uses the arrow keys to aim and the spacebar to shoot. Points are scored by matching and clearing balls, with increasing levels of difficulty as the game progresses.

This game comprises **three levels**, each introducing new mechanics. In Level 2, players can activate **power-ups** that temporarily slow down the balls. Level 3 introduces **bombs**, which can destroy multiple balls at once. A player starts with three lives and progresses by scoring 100 points per level. If all lives are lost, the game ends.

---

2. Game Mechanics

The mechanics of this Zuma implementation are designed to challenge the player's precision and strategy while keeping the gameplay engaging.

2.1 Controls

- **Arrow Keys**: Aim and fire the bullet in the chosen direction.

2.2 Ball Interaction

- **Matching Colors**:
  - If the bullet's color matches the color of the ball it hits, a chain reaction is triggered, clearing three balls (including the one hit).
  - **Score**: +10 points for each match.
- **Non-Matching Colors**:
  - If the bullet's color doesn't match, the ball's color changes to match the bullet.

2.3 Scoring and Level Progression

- **Points**: Each successful match adds +10 points to the player's score.
- **Level Completion**: The player progresses to the next level upon reaching 100 points.

- **Levels**: The game consists of three levels, each with increasing difficulty and additional features.

2.4 Lives and Game Over

- **Starting Lives**: Players begin with three lives per level.
- **Losing Lives**: A life is lost if a ball reaches the end of the path.
- **Game Over**: The game ends when all lives are lost.

2.5 Power-Ups (Level 2)

- Matching ball and bullet colors triggers a **power-up** that slows the movement of all balls for 5 seconds.

2.6 Bombs (Level 3)

- Matching bullet and ball colors activates a **bomb** that clears up to eight balls, offering strategic advantages in managing the increasing difficulty.

2.7 Tunnels (Level 3)

- Balls travel through tunnels, temporarily disappearing and reappearing in different sections of the path.
- Tunnels add an extra layer of difficulty by requiring players to anticipate ball movements.

These mechanics combine strategy and quick decision-making, creating a dynamic gameplay experience.

---

3. Implementation Details

The game is implemented in assembly language with the following key elements:

3.1 Level Initialization and Transition

- Predefined paths for ball movement are created at the start of each level.
- Variables for score and lives are reset to ensure replayability.

- Path lengths are dynamically managed using constants like `lengthRL` and `lengthTB`.

3.2 Ball Management and Movement

- **Position Tracking**:
  - `ballsX`, `ballsY`: Store X and Y coordinates of balls.
  - `ballsC`: Stores the color of each ball.
- **Dynamic Updates**:
  - `numOfActiveBalls`: Tracks the count of active balls.
  - New balls are assigned positions from predefined paths like `Path1XFull` and `Path2YFull`.
- **Movement Logic**:
  - Implemented using loops like `inner_level2` and `inner_level3`.

3.3 Rendering and Visual Updates

- **Ball Rendering**:
  - `Gotoxy` positions the balls on the screen.
  - The `OFFSET ball` character is displayed with color determined by `ballsC`.
- **HUD Updates**:
  - Functions like `PrintScore`, `DisplayLives`, and `PrintLevel` update real-time gameplay stats.

3.4 Game Mechanics

- **Scoring**: Incremented as the player matches balls, with a target of 100 points per level.
- **Lives**: Decremented upon collision with designated "lose points" (e.g., `lose1X`, `lose1Y`).
- **Power-Ups**: Managed via the `powerupcount` variable and triggered during gameplay.

3.5 Menu and Instructions

- The **menu screen** offers options to start the game, view instructions, or exit.
- The **instructions screen** provides detailed gameplay guidance, including control schemes and scoring information.

3.6 Level Completion and Game Flow

- Levels are structured as loops (`level2`, `level3`).
- Checkpoints (`level3fromthetop`) allow resuming after life loss.
- Level transitions occur upon completion, or the game ends if all lives are lost.

---

4. Challenges and Solutions

4.1 Path Complexity and Ball Movement

- **Challenge**: Designing dynamic paths while managing multiple balls with independent positions and colors.
- **Solution**: Predefined paths stored in arrays (`Path1XFull`, `Path1YFull`) standardized movement. A modular approach ensured scalability.

4.2 Screen Rendering Efficiency

- **Challenge**: Minimizing flickering while rendering moving balls and HUD elements.
- **Solution**: The `Gotoxy` function ensured precise updates. Loop optimizations reduced unnecessary screen redraws.

4.3 Collision Detection

- **Challenge**: Detecting collisions accurately between balls and critical points.
- **Solution**: Position arrays (`ballsX`, `ballsY`) were compared with coordinates. Conditional checks handled edge cases, ensuring consistent gameplay.

4.4 Score and Lives Management

- **Challenge**: Synchronizing scores, lives, and power-ups with level resets.
- **Solution**: Global variables (`score`, `lives`) and modular functions (`PrintScore`, `DisplayLives`) streamlined updates.

4.5 Multi-Level Transitions

- **Challenge**: Preserving player progress while transitioning between levels.
- **Solution**: Independent loops with conditional jumps (`cmp`) ensured smooth progression or game-over triggers.

---

5. Conclusion

This assembly-based implementation of Zuma successfully integrates dynamic ball movement, scoring, power-ups, and level transitions. Despite challenges in path design, rendering, and collision management, the modular approach and strategic solutions ensured robust functionality.

The project demonstrates how low-level programming techniques can create visually engaging and interactive applications. It highlights the potential of assembly programming for crafting scalable and efficient games while providing valuable insights into optimizing resource-constrained environments.