Figure 1: BNU

Software Engineering Project

Authors:
Toni Ihab Youssef
Omar Tamer

Supervisor: Dr. Fatma Sakr
Faculty of Computer Science at Benha National University

2024-2025

Contents

1 Abstract

This project focuses on developing a Pong Game simulation using a microprocessor and assembly language. The game involves two players who control paddles to prevent a ball from passing them, with the ball bouncing off the top, bottom, and side walls. The project emphasizes the integration of hardware with software to create an interactive gaming experience. The assembly code controls the ball movement, paddle control, and score management, while the microprocessor handles the input/output operations and visual display.

2 Main Thesis

The Pong Game is implemented on a microprocessor using assembly language to provide a low-level interaction between hardware components and software logic. The game's primary goal is to simulate a two-player environment where players can control paddles on the screen to keep the ball in play. The ball's position is tracked, and it bounces off the walls and paddles according to the game's rules.

2.1 Key Components

1. **Ball Movement and Collision Detection:** The ball's position is continuously updated based on its velocity. When the ball reaches the left or right boundary of the screen, it triggers scoring for the opponent and resets the ball's position. The same logic applies to the top and bottom boundaries to keep the ball within the playing area.

2. **Paddle Movement:** Players control paddles using keyboard input or pre-programmed keys. The paddles move up or down within the screen boundaries. The paddle's position is constantly updated, and if the ball collides with the paddle, it changes direction accordingly.

3. **Scorekeeping:** The score for each player is displayed on the screen, and a player earns a point when the opponent fails to intercept the ball. The game continues until one player reaches a set score, triggering the game-over sequence.

4. **Input Handling:** The project utilizes keyboard input to control the movement of the paddles. The assembly language code interfaces with the microprocessor to detect key presses and map them to paddle movement (up and down).

5. **Visual Display:** The game graphics are displayed on a simple text or graphical screen, depending on the microprocessor's capabilities. The paddles and ball are represented as characters or pixels on the screen, and the game status (score, winner) is updated in real-time.

6. **Game Over and Reset:** When a player reaches the target score, the game ends, and a message is displayed indicating the winner. The player can press a key to reset the game and start a new round.

2.2 Development Process

1. **Microprocessor Selection**: The project uses a microprocessor such as the Intel 8086, 8051, or a similar processor suitable for assembly language programming and handling input/output operations.

2. **Assembly Code Development**: The game logic is written in assembly language, which directly interacts with the microprocessor. The code handles ball movement, collision detection, paddle movement, and score updating. Interrupts are used to manage timing and input handling effectively.

3. **Testing and Debugging**: The system is tested to ensure accurate detection of ball and paddle collisions, as well as proper scorekeeping. Debugging tools and emulators are used to simulate the game and identify any issues in the assembly code.

This project demonstrates the capabilities of microprocessors in handling real-time games, focusing on low-level programming and hardware interaction. It also serves as an introduction to the use of assembly language for developing interactive applications.

3 Comprehensive Code Insight and Architectural Overview

3.1 Stack Segment

The STACK SEGMENT section defines the memory space reserved for the program's stack. The stack is crucial for storing local variables, return addresses, and other temporary data during the execution of the program. In this case, the stack is allocated with 64 bytes, initialized with blank spaces. This ensures that the stack can grow as needed during program execution.

3.2 Data Segment

The DATA SEGMENT is where all the game-related data and variables are stored. This segment includes the initialization of game state variables, graphical settings, player statistics, and text prompts for the user interface. The values in this segment define key aspects of the game, such as the positions of the ball and paddles, scoring system, and various UI messages.

In assembly language programming is a section of the program's memory that is specifically reserved for storing variables and constants. This segment is vital for managing the data the program will work with throughout its execution, and it plays a significant role in controlling game logic and interface elements.

In the provided code, the Data Segment is declared with the DATA SEGMENT directive and contains various variables that are used to store information regarding the game's state, graphical elements, and textual information.

3.3 Code Segment

3.3.1 Main Procedure

The MAIN PROC procedure in this project serves as the core control flow for the Pong game, managing the game's various states and updates. It ensures that the game runs smoothly through a continuous loop, updating the game elements, handling user input, and switching between different scenes such as the main menu, game over screen, and active gameplay.

1. **Segment Initialization**: The procedure sets up the necessary segments (code, data, and stack) and prepares the environment by pushing and popping the segment registers.

2. **Time Check**: A time-checking mechanism is implemented to ensure that the game updates at regular intervals, preventing unnecessary CPU usage by comparing system time with the previously stored time.

3. **State Management**: The procedure handles different game states (e.g., main menu, gameplay, game over) by checking flags and calling appropriate subroutines to display the relevant UI or perform game logic.

4. **Game Mechanics**: The procedure integrates game logic for moving and drawing the ball and paddles, updating the user interface, and maintaining the overall game flow, including detecting game-over conditions.

5. **Exit Handling**: A clean exit mechanism is included to stop the game and exit gracefully when required.

3.3.2 Ball Mechanics: Reset, Movement, and Collision Handling

The ball's behavior in the game is controlled through two primary procedures: `RESET_BALL` and `MOVE_BALL`. These procedures work together to manage the ball's positioning, movement, collisions with game boundaries and paddles, as well as scoring and game state transitions. The ball's reset mechanism ensures consistency at the start of each round, while the movement and collision handling logic governs its interactions within the game environment.

1. **Ball Reset Mechanism**: The `RESET_BALL` procedure is executed to reposition the ball at the initial starting position and reverse its velocity when a new round begins or the game is reset.

   - The `BALL_X` and `BALL_Y` registers are set to their initial values (`BALL_SX` and `BALL_SY`).
   - The ball's velocities `BALL_VX` and `BALL_VY` are negated to reverse its direction, ensuring unpredictability.

2. **Ball Movement Updates**: The `MOVE_BALL` procedure governs the ball's continuous movement by updating its position based on its velocity components `BALL_VX` and `BALL_VY`.

3. **Boundary Collisions**: The procedure detects when the ball hits the edges of the game area and reverses its velocity to simulate a bounce off the boundaries.

4. **Paddle Collisions**: When the ball intersects with the paddles, its direction is reversed to reflect the bounce interaction between the ball and the paddles.

5. **Scoring Logic**: The procedure checks when the ball exits the play area to update player scores. It resets the ball and prepares for the next round. If a score limit is reached, the game transitions to the "Game Over" state.

6. **Game Over Handling**: Tracks when a player wins, updates the winner index, and resets the game state for either continuation or termination.

7. **Collision Dynamics**: Validates the ball's position against the paddles and boundaries, ensuring realistic behavior during interactions.

3.3.3 Paddle Movement and Control Logic

The MOVE_PADDLES procedure handles the movement of the paddles in the Pong game, using keyboard inputs for the left paddle and either keyboard inputs or AI logic for the right paddle. It ensures that paddle positions stay within the game boundaries and adjusts their behavior dynamically based on inputs and the ball's position.

1. **Keyboard Input Handling**: Reads keyboard inputs to detect player commands for moving the left paddle (`LPADDLE_Y`) up or down using keys `W/S` or `w/s`.

2. **AI Control for Right Paddle**: Implements simple AI logic to move the right paddle (`RPADDLE_Y`) based on the ball's position, ensuring it tracks the ball within its vertical range.

3. **Boundary Enforcement**: Prevents paddles from moving beyond the top or bottom bounds of the game window by clamping their positions within allowable limits.

4. **Player-Specific Logic**: Differentiates between manual controls for the left paddle and AI or keyboard-based controls for the right paddle.

5. **Collision Dynamics**: Ensures paddle movement aligns with the game's design rules, including smooth vertical adjustments and responsive handling of edge cases.

3.3.4 Rendering Logic for Ball and Paddles

The rendering of the ball and paddles is handled by the `DRAW_BALL` and `DRAW_PADDLES` procedures. Both procedures share similar logic for drawing the graphical elements on the screen, using pixel plotting within a defined area to create the visual representation of the ball and paddles at their respective positions.

1. The initial coordinates for the ball or paddles are loaded into the registers `CX` (x-axis) and `DX` (y-axis).

2. A nested loop is used to draw pixels column by column and row by row within the defined size for each element.

- **Horizontal Rendering**: Pixels are drawn along the x-axis until the element's width is complete.
- **Vertical Rendering**: After completing a row, the process moves to the next line (y-axis) and repeats until the element's height is rendered.

3. The procedure exits when the entire area is filled, and control is returned with `RET`.

3.3.5 Game Logic and User Interface Procedures

The following section outlines the procedures used to manage the user interface, game flow, scoring, and main menu logic in the Pong-like game. These procedures control drawing elements on the screen, incrementing player scores, handling the game-over menu, and responding to user inputs for gameplay actions.

DRAW_UI procedure is responsible for displaying the user interface, including the player scores, at various stages of the game.

1. The procedure sets the cursor position to display Player 1's score at coordinates (04h, 06h).
2. It then prints the label for Player 1's score using TXT_PI_PTS.
3. The cursor is moved to a new position to display Player 2's score at coordinates (04h, 1Fh).
4. It prints the label for Player 2's score using TXT_PII_PTS.

INC_PI_PTS procedure is used to increment Player 1's points. It performs the following steps:

1. Clears the AX register and loads LPAD_PTS into AL.
2. Converts the points into ASCII by adding 30h.
3. Updates the TXT_PI_PTS string with the updated score.

INC_PII_PTS procedure is used to increment Player 2's points. It performs the following steps:

1. Clears the AX register and loads RPAD_PTS into AL.
2. Converts the points into ASCII by adding 30h.
3. Updates the TXT_PII_PTS string with the updated score.

KO_MENU procedure handles the game-over menu, displaying options for replay or returning to the main menu. It follows these steps:

1. Clears the screen and sets the cursor position.
2. Displays the *"Game Over"* message and updates the screen with the winner's name.

3. Prompts the user with options to either replay or exit to the main menu.
4. Waits for user input, with responses for replay (R or r) or exiting to the main menu (E or e).

MAIN_MENU_UI procedure displays the main menu where users can choose to start a single-player game, a multi-player game, or exit the game. The process involves:

1. Clearing the screen and displaying the title of the game.
2. Presenting options for single-player, multi-player, and exit.
3. Waiting for user input to select the desired action.
4. If S or s is pressed, the single-player game begins; if M or m is pressed, the multi-player game begins; if E or e is pressed, the game exits.

UPDATE_KO procedure updates the winner's text in the game-over screen based on the value of WINNER_INDEX, which represents the winning player. It performs the following:

1. Loads WINNER_INDEX into AL.
2. Converts the index into ASCII format by adding 30h.
3. Updates the winner's text display.

CLEAR_SCRN procedure clears the screen by setting the video mode and background color:

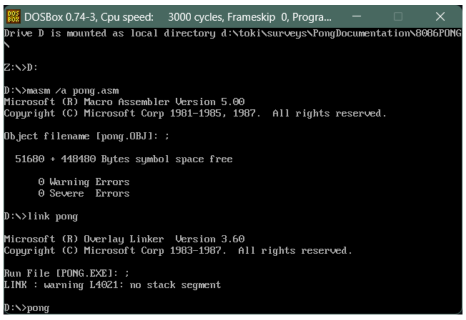
1. It sets the video mode to 0Dh.
2. It sets the background color to black (00h) using BIOS interrupt 10h.

EXIT_GAME procedure exits the game by setting the video mode to 02h and then invoking DOS interrupt 21h to terminate the program.

1. It sets the video mode to 02h.
2. It invokes DOS interrupt 21h to terminate the program and return control to the operating system.

4 Run Time Overview Snapshots

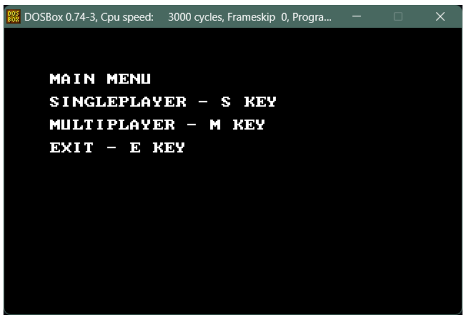
4.1 DOSbox Terminal


DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Progra...

```
Drive D is mounted as local directory d:\toki\surveys\PongDocumentation\8086PONG
\

Z:\>D:

D:\>masm /a pong.asm
Microsoft (R) Macro Assembler Version 5.00
Copyright (C) Microsoft Corp 1981-1985, 1987.  All rights reserved.

Object filename [pong.OBJ]: ;

  51680 + 448480 Bytes symbol space free

      0 Warning Errors
      0 Severe  Errors

D:\>link pong

Microsoft (R) Overlay Linker  Version 3.60
Copyright (C) Microsoft Corp 1983-1987.  All rights reserved.

Run File [PONG.EXE]: ;
LINK : warning L4021: no stack segment

D:\>pong
```

Figure 2: DOSbox Process

4.2 Main Menu


DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Progra...

```
MAIN MENU
SINGLEPLAYER - S KEY
MULTIPLAYER - M KEY
EXIT - E KEY
```

Figure 3: Main Menu

4.3 Gameplay

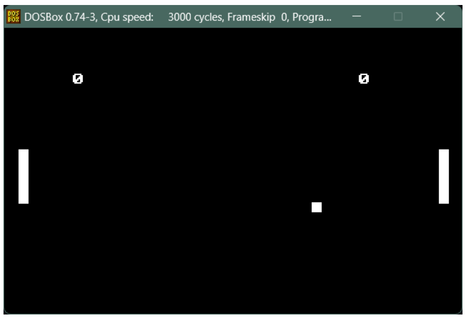
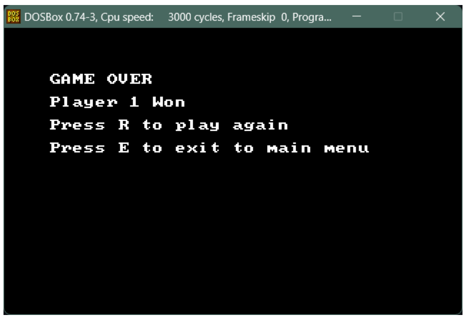

DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Progra...
0
0

Figure 4: Gameplay

4.4 Game Over


DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Progra...

```
GAME OVER
Player 1 Won
Press R to play again
Press E to exit to main menu
```

Figure 5: Game Over

5 Summary and Insights

The Pong Game simulation using a microprocessor and assembly language showcases the use of low-level programming for interactive applications. It addressed challenges like real-time control, collision detection, and score tracking while optimizing system resources.

This project enhanced our skills in embedded systems programming, real-time resource management, and low-level game development, providing practical experience in building efficient interactive systems with minimal resources.