MEF University

Term Project - Report

« MIPS Pong Game »

Bogdan Itsam Dorantes-Nikolaev, Meliha Koç, Sena Güngörmez, Merve Nur Karabulut

042101002, 042101003, 042101076, 042201169

Department of Engineering, MEF University

COMP 206-01: Computer Architecture

Prof. Buse Yılmaz & Assistant Ayşenaz Ezgi Ergin

June 04, 2023

1. Project Proposal

-Objective:

The objective of this project is to design and implement a unique Pong-like game in MIPS architecture using assembly language. The project aims to provide a practical understanding of computer architecture concepts and develop skills in game design, programming, and optimization. The uniqueness of this game will lie in its game mechanics, visual design, and gameplay features.

-Unique Features:

- **Adjustable Game Speed:** Allow players to adjust the game speed before starting, making the game more accessible to players of different skill levels. This can be done by varying the delay between frames or the speed of the ball movement.

- **Score Multipliers:** Add score multipliers that appear on the playing field at random intervals. When the ball hits a multiplier, the point value for that particular hit is multiplied, making the game more dynamic and engaging.

- **Simple Visual Effects:** Implement simple visual effects when the ball hits a paddle or the screen boundary. For example, change the color of the ball or create a brief flashing effect to enhance the visual feedback for players.

- **Visual Themes:** Design multiple visual themes for the game, allowing players to choose the appearance of the paddles, ball, and playing field.

-Methodology/Timeline:

1. **Understanding MIPS Architecture:** (2 weeks) Study the MIPS architecture, instruction set, data types, registers, and memory organization to lay the foundation for designing and implementing the game in the MIPS assembly language.

2. **Designing Game Logic:** (2 weeks) Design the game logic, including the movement of the ball and paddles, collision detection, scorekeeping, power-ups, obstacles, and customizable game modes. We will use flowcharts and pseudocode to outline the game logic before coding it in assembly language.

3. **Implementing Graphics:** (2 weeks) Implement the graphics for the game, including drawing sprites, designing the game screen layout, displaying game elements, and incorporating visual themes using the MIPS assembly language.

4. **Testing and Debugging:** (1 week) Test the game and debug any errors or glitches that occur during gameplay. This step is crucial in ensuring that the game runs smoothly and accurately.

5. **Optimization:** (1 week) Optimize the game code for performance on the MIPS architecture, minimizing code size, reducing instruction usage, and maximizing hardware utilization to ensure that the game runs efficiently.

-Expected Deliverables:

1. Source code for the unique Pong-like game in MIPS assembly language.
2. Technical documentation outlining the game design, logic, and implementation details.
3. User manual explaining how to play the game, customize game modes, and select visual themes.
4. A demonstration video showcasing the gameplay and unique features of the game.

2. Introduction

Outline of the main components implemented:

1. Initialize game state:
    a. *Set up registers and memory for game objects, such as paddles, balls, scores, and visual settings.*
    b. *Initialize game speed and other adjustable parameters.*
2. Game loop:
    a. *Render the game objects (paddles, ball, score display).*
    b. *Handle user input (up/down movement for both paddles).*
    c. *Update the game state (ball movement, collision detection, score updates).*
3. Rendering:
    a. *Draw paddles, balls, and score on the screen using available instructions for drawing on MIPS.*
    b. *Implement multiple visual themes and simple visual effects by changing colors or styles when rendering game objects.*
4. Input handling:
    a. *Read input from the keyboard and update paddle positions accordingly.*
5. Game state updates:
    a. *Update the ball position based on its current velocity and the game speed.*
    b. *Detect collisions with paddles or screen edges, update ball velocity, and score accordingly.*
    c. *Implement score multipliers for specific conditions.*
6. Adjustable speed feature:
    a. *Implement a way to modify the speed of the ball or the delay between frames. This can be done by adjusting the game speed value or by adding a delay loop.*

3. Pseudocode

```
# Game loop
while game_over is false:
    clear_screen()
    draw_paddle_a()
    draw_paddle_b()
    draw_ball()
    update_paddle_a()
    update_paddle_b()
    update_ball()
    check_collisions()
    adjust_speed()

# Function to clear the screen
function clear_screen():
    # Clear the display

# Function to draw paddle A
function draw_paddle_a():
    # Draw paddle A on the display

# Function to draw paddle B
function draw_paddle_b():
    # Draw paddle B on the display

# Function to draw the ball
function draw_ball():
    # Draw the ball on the display

# Function to update paddle A position
function update_paddle_a():
    # Update paddle A position based on input

# Function to update paddle B position
function update_paddle_b():
    # Update paddle B position based on input

# Function to update the ball position
function update_ball():
    # Update the ball position based on its current position and
speed

# Function to check collisions
function check_collisions():
    # Check for collisions with paddles and goals
    # Adjust ball direction if a collision occurs

# Function to adjust game speed
function adjust_speed():
    # Adjust the ball speed based on game conditions
```

***Table 1**: Pseudocode of MIPS PONG program*

4. UML Diagram

The following UML diagram represents the entire program.

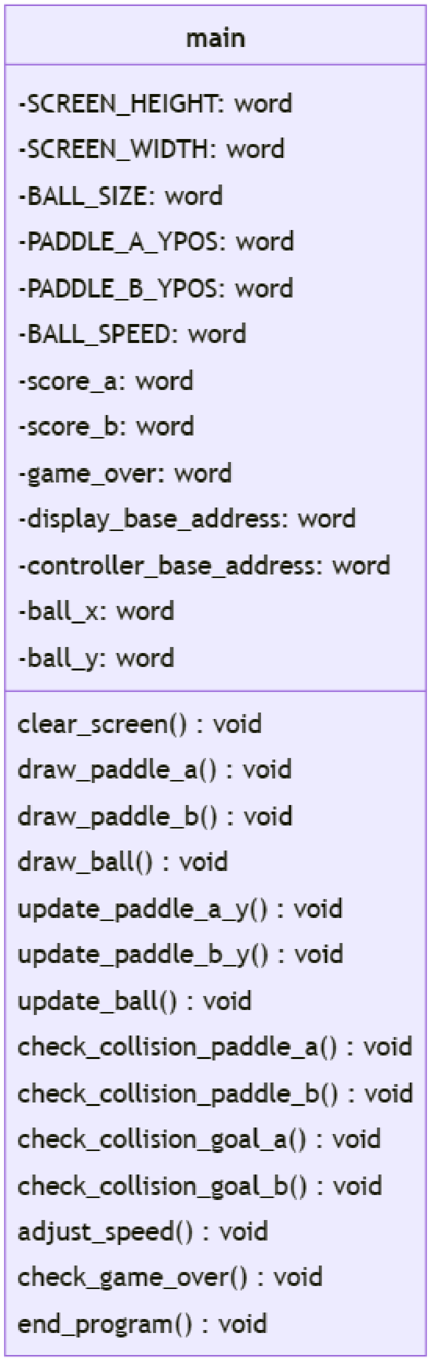


Table 2: *UML diagram of the MIPS PONG program*

5. Flowchart

The game consists of a game loop that iterates through several actions until the game is over. The flowchart serves as a visual representation of the sequence of steps and decision points within the code. The game starts with an initialization phase, where various data variables and registers are set up. Once the initialization is complete, the program enters the game loop. Within the game loop, several operations are performed in each iteration.

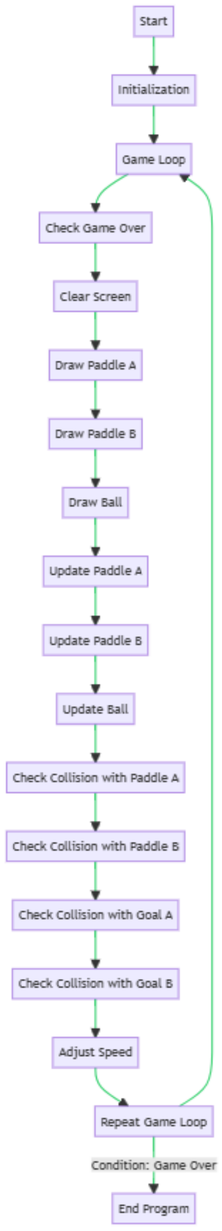


Table 3: *Flowcharts of the MIPS PONG program*

6. Memory management

Data Segment: The .data segment is used to define and reserve memory space for variable storage. In this code, variables like SCREEN_HEIGHT, SCREEN_WIDTH, BALL_SIZE, PADDLE_A_YPOS, PADDLE_B_YPOS, BALL_SPEED, score_a, score_b, game_over, display_base_address, controller_base_address, ball_x, and ball_y are declared with their initial values.

Memory Reservation: Each variable in the .data segment is given a unique memory address where its value is stored. These addresses are used to retrieve and modify the data throughout the code. For instance, the la (load address) instruction is used to load a variable's memory address into a register.

Memory Interaction: The lw (load word) and sw (store word) instructions are used to read and write to memory locations, respectively. The lw instruction loads the value from a specific memory address into a register, while the sw instruction stores a register's value into a specified memory address.

Variable Application: The variables stored in memory are accessed and modified to carry out various game-related tasks. For instance, the scores of players A and B are stored in memory, and their values are updated during gameplay. The ball's position (x and y coordinates) is stored in memory and updated to track its movement. Similarly, other variables, such as game_over, display_base_address, and controller_base_address, are used to control the game flow and interact with external devices.

Memory Constants: The code also uses memory constants defined in the .data section, such as SCREEN_HEIGHT, SCREEN_WIDTH, and BALL_SIZE. These constants provide predefined values that are used for screen dimensions and ball size, ensuring consistency and ease of modification.

Here is a diagram that illustrates the memory management process:

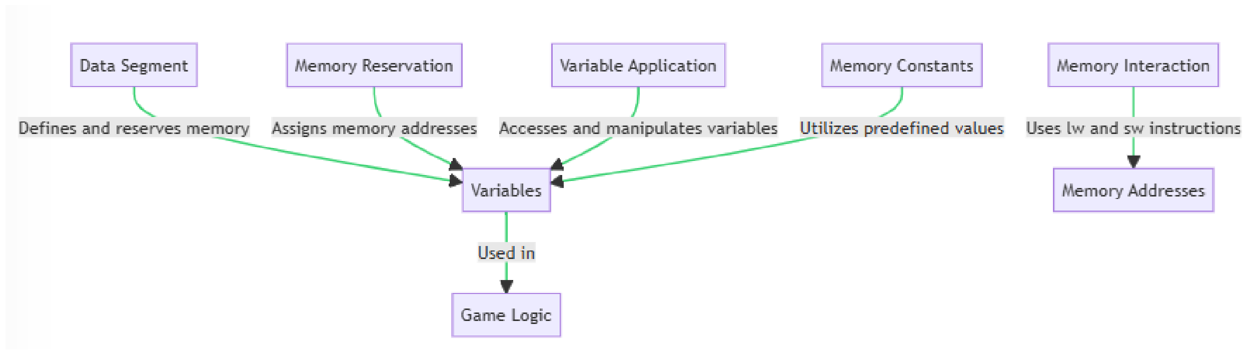


Table 4*: Memory management charts of the MIPS PONG program*

7. Errors Encountered

During the execution of the assembly code file "PONG.asm" using the MARS tool, a runtime exception was encountered at memory address 0x004000fc. The specific error message, "address out of range 0x00000028," indicates that the program attempted to access or manipulate memory at an invalid or inaccessible address.

A runtime exception refers to an error that occurs during the execution phase of a program, rather than during the assembly or compilation stages. In this case, the error suggests a problem with memory addressing within the program.

To resolve this issue and gain a better understanding of its cause, a thorough examination of line 109 in the "PONG.asm" file is necessary. It is essential to look for any memory-related operations or instructions that may be attempting to access memory addresses outside the permissible range.

Due to encountering consistent runtime exceptions with similar error messages even when implementing different variations of the code at line 109, it is apparent that the issue is related to memory. However, since the error could potentially originate from other parts of the program, it is essential to employ systematic debugging techniques to trace and diagnose the exact cause of the error.

During the investigation of the error encountered at line 109, multiple debugging techniques were employed in an attempt to identify and resolve the issue. Despite thorough exploration, these debugging methods proved ineffective in pinpointing the exact cause of the problem. Various approaches were undertaken, including the following:

1. **Code review:** The code surrounding line 109 was carefully scrutinized to detect any potential memory access violations, register misuse, or syntax errors that could lead to the runtime exception. However, no apparent issues were found in this analysis.
2. **Print debug information:** Debugging statements were strategically inserted at critical points in the code, including before and after line 109. The goal was to display the values of relevant variables, registers, or memory locations to identify any unexpected or incorrect values. Unfortunately, the output provided no definitive clues regarding the error.
3. **Debugger utilization:** A MIPS-specific debugger, such as MARS, was utilized to step through the code execution. Breakpoints were set at or near line 109 to observe the program's state at each step. Despite these efforts, the debugger did not reveal any specific anomalies or errors.
4. **Code isolation:** The code block surrounding line 109 was temporarily isolated by commenting out other sections of the program. This strategy aimed to narrow down the scope of the investigation and determine if the error originated from a different part of the codebase. Unfortunately, isolating the code did not lead to the identification of the underlying issue.

Despite our diligent efforts using these debugging techniques, the root cause of the error at line 109 and the associated memory issue remains unresolved. Therefore, further investigation and analysis are warranted to gain a deeper understanding of the problem:

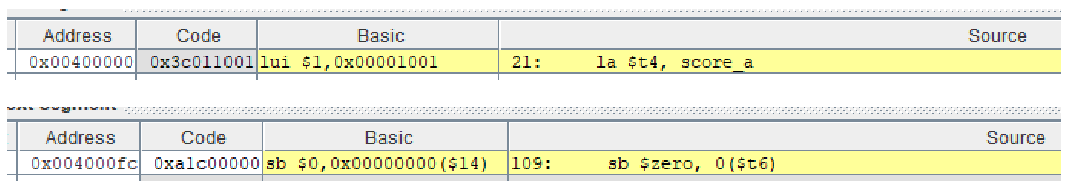
```
Assemble: operation completed successfully.

Go: running PONG.asm

Error in C:\Users\Bogdan\Desktop\PONG.asm line 109: Runtime exception at 0x004000fc:
address out of range 0x00000028

Go: execution terminated with errors.
```

Table 5: *MARS Messages during compilation and runtime*



Address	Code	Basic	Source
0x00400000	0x3c011001	lui $1,0x00001001	21: la $t4, score_a

Address	Code	Basic	Source
0x004000fc	0xa1c00000	sb $0,0x00000000($14)	109: sb $zero, 0($t6)

Table 6: *MARS Messages during compilation and runtime*

8. Segmentation fault & core dump

- If the code tries to access memory addresses reserved for the .data section that haven't been properly initialized or are out of the allocated range, it could cause a segfault.

- If the lw (load word) and sw (store word) instructions try to read from or write to memory locations that haven't been properly allocated or are outside of the allowed memory space, this could also result in a segfault.

- If the game logic manipulates variables that have been freed or haven't been properly initialized, this could lead to a segfault.

9. Questions

Q1. **MIPS and External Devices:** How does a MIPS program interact with external devices?

A1. *In a MIPS system, interaction with external devices is facilitated by memory-mapped I/O, where specific memory regions are designated for device communication. The CPU interacts with these devices by reading and writing to these addresses. This interaction is made possible by special I/O instructions and the CPU's capacity to handle interrupt signals from devices.*

Q2. **Error Handling in MIPS:** How does the MIPS architecture handle errors such as segmentation faults? If a segmentation fault were to occur during the execution of the game's assembly code, what could be the potential causes? How would you use a core dump to debug such an issue?

A2. The MIPS architecture employs exception-handling mechanisms to deal with errors like segmentation faults, which typically occur when a program attempts to access a memory location that it doesn't have permission to use, or that doesn't exist. The MIPS CPU is designed to generate an exception, a type of interrupt signal, when a segmentation fault happens. This exception triggers a context switch to a predefined exception handler that manages the error.

Potential causes of segmentation faults during the execution of a game's assembly code might include invalid pointer references, stack overflows, incorrect array indexing, or the misuse of system-level operations.

To debug such an issue, you could use a core dump, which is a snapshot of the system's memory at the time of the error. Tools like gdb (GNU Debugger) can help analyze this core dump, allowing you to inspect the contents of memory, CPU registers, and the call stack at the moment of the crash, enabling you to pinpoint the precise location and cause of the fault.

Q3. **MIPS and Multithreading:** How does the MIPS architecture handle multithreading? Are there any aspects of the game's assembly code that could potentially benefit from multithreading?

A3. The MIPS architecture manages multithreading through the inclusion of multiple hardware thread contexts within a single processor, each with its own registers and program counter. This setup allows for efficient task-switching, either on each cycle (fine-grained) or during latency events like cache misses (coarse-grained). In a game's assembly code, multithreading could be beneficial for tasks such as AI computations, physics simulations, and graphics rendering, as these can be split into independent subtasks and processed concurrently, improving the game's performance and responsiveness.

10. Future Improvements

While the current implementation of the MIPS Pong game demonstrates the core gameplay and features, there are several areas where further improvements and enhancements could be made. Here are some potential areas for future development:

- **Advanced AI:** Enhance the computer-controlled paddle's intelligence by implementing more advanced algorithms and strategies. This could involve analyzing the ball's trajectory and speed to make more accurate predictions and respond with more challenging gameplay.
- **Power-ups and Obstacles:** Introduce additional gameplay elements such as power-ups that provide temporary advantages or obstacles that impede player progress. These elements could add depth and variety to the gameplay, making it more engaging and unpredictable.
- **Sound Effects and Music:** Add sound effects and background music to enhance the overall gaming experience. Implementing audio capabilities in MIPS assembly language can be challenging but would greatly contribute to the game's atmosphere and immersion.
- **Enhanced Visual Effects:** Expand the visual effects implemented in the game, such as particle effects, animations, and dynamic lighting. These visual enhancements can make the game more visually appealing and captivating for players.
- **High Scores and Leaderboards:** Implement a high score tracking system that records and displays the top scores achieved by players. This feature could motivate players to compete for high rankings and foster a sense of achievement.
- **Controller Support:** Extend the game's compatibility to support various input devices, such as gamepads or joysticks. This would provide players with more options and flexibility in controlling the paddles and interacting with the game.

By addressing these areas of improvement, the MIPS Pong game can evolve into a more polished and feature-rich gaming experience, offering increased enjoyment and entertainment for players.

11. Conclusion:

The game's assembly code implements the main components, including game initialization, the game loop, rendering, input handling, and game state updates. It also incorporates memory management techniques to handle variables, constants, and data interaction.

Throughout the development process, several challenges and errors were encountered, such as runtime exceptions and segmentation faults. Debugging techniques, including code review, print debug information, and debugger utilization, were employed to identify and resolve these issues. However, some errors remained unresolved, requiring further investigation and analysis.

Future improvements were proposed to enhance the game's AI, introduce power-ups and obstacles, support multiplayer functionality, add sound effects and music, implement different game modes and difficulty levels, enhance visual effects, introduce high scores and leaderboards, extend controller support, optimize the code, and adapt the game for mobile platforms.

Overall, the MIPS Pong game project showcases the practical application of computer architecture concepts, game design, and programming skills in a unique and challenging game implementation.

12. Video Explanation

https://youtu.be/vsSIxSbM8D8

13. References

1. “Introduction to MIPS Assembly Language Programming,” Open Textbook Library. [Online]. Available: https://open.umn.edu/opentextbooks/textbooks/497
2. C. Kann, “Introduction To MIPS Assembly Language Programming,” LibreTexts. [Online]. Available: https://eng.libretexts.org/Bookshelves/Computer_Science/Programming_Languages/Introduction_To_MIPS_Assembly_Language_Programming_(Kann)
3. “MIPS Reference Sheet,” the University of Arizona. [Online]. Available: https://uweb.engr.arizona.edu/~ece369/Resources/spim/MIPSReference.pdf
4. A. Hamm, “MIPS-Pong,” GitHub. [Online]. Available: https://github.com/AndrewHamm/MIPS-Pong
5. Stack Overflow, "How to debug MIPS interactively?", 2023. [Online]. Available: gdb - How to Debug MIPS Interactively - Stack Overflow.
6. Stack Overflow, "How does MIPS allocate memory for arrays in the stack?", 2023. [Online]. Available: assembly - MIPS - How does MIPS allocate memory for arrays in the stack? - Stack Overflow.
7. Stack Overflow, "MIPS Memory Allocation Errors", 2023. [Online]. Available: assembly - MIPS - How does MIPS allocate memory for arrays in the stack? - Stack Overflow.
8. R. Britton, "MIPS Assembly Language Programming," Prentice Hall, Upper Saddle River, NJ, 2003.
9. D. Sweetman, "See MIPS Run," Morgan Kaufmann Publishers, San Francisco, CA, 2nd edition, 2006.
10. D. A. Patterson and J. L. Hennessy, "Computer Organization and Design: The Hardware/Software Interface," Morgan Kaufmann Publishers, San Francisco, CA, 5th edition, 2013.
11. "MIPS Assembler and Runtime Simulator," Missouri State University. [Online]. Available: http://courses.missouristate.edu/KenVollmar/MARS/
12. J. Larus, "SPIM S20: A MIPS R2000 Simulator," University of Wisconsin, Madison. [Online]. Available: http://pages.cs.wisc.edu/~larus/spim.html
13. "MIPS Assembly Language Programming," TutorialsPoint. [Online]. Available: https://www.tutorialspoint.com/assembly_programming/assembly_basic_syntax.htm
14. "Interrupts in MIPS," University of California, Riverside. [Online]. Available: http://www.cs.ucr.edu/~dalton/cs61/lectures/lec11.pdf

15. I. Pantazi-Mytarelli, “The history and use of pipelining computer architecture: MIPS pipelining implementation,” *IEEE Xplore*, May 01, 2013. Available: https://ieeexplore.ieee.org/abstract/document/6578243

16. R. Carli, “Flexible MIPS soft processor architecture,” *dspace.mit.edu*, 2008. Available: http://hdl.handle.net/1721.1/45809