MIPS Binary Game

a) Program Description

This project is a single-player Binary Game built entirely in MIPS assembly to run in the MARS simulator. Its goal is to test a player's ability to convert between 8-bit binary numbers and their decimal equivalents (0-255). The game randomly switches between Binary and decimal and respectively prompts te user to enter it's equivalent input. Level L has L number of problems. Eg Level 7 has 7 problems. A correct answer gives +1 point, for a maximum possible score of 55. To keep the code clean, it's organized into 10 separate .asm files. I also implemented extra features like sound effects for correct/wrong answers and a configurable timeout timer .

b) Challenges Faced and How They Were Overcome

1) Challenge: The most critical bug was that the game would mark perfectly correct answers as wrong.
   - Solution: I tracked this down to a MIPS calling convention error. I was storing the correct answer in a temporary register instead of a saved register and hence the function call would override the value in the temporary register which caused unusual output. Hence I fixed it using a saved register to store the correct answer, which is preserved across function calls.

2) Challenge: MIPS doesn't have a built-in rand() function, so the game would be the same every time.
   - Solution: I wrote a simple Linear Congruential Generator in randgen.asm. This is a small math formula that creates a sequence of pseudo-random numbers. I seed it using the system clock (syscall 30) when the game starts, which ensures the problems are different every time you play.

3) Challenge: The MARS read_string (syscall 8) pauses the whole program until you hit Enter and hence woldn't run a timer in the background to interrupt you.
   - Solution: I used a post-input check. I record the system time (syscall 30) before asking for input, and then I record it again after the user hits Enter. I then subtract the two to find the elapsed time. If it's over the limit, the answer is marked as wrong.

c) What I Learned From the Project

As I have only learnt and practiced to program assignments and homeworks in assembly, answering an opened question like building a game was a very challenging yet fun project that I have encountered. I have learnt the following from the project

1. MIPS Register Discipline and Calling Conventions

   I learned how crucial it is to use registers correctly when functions call other functions. Temporary registers ($t0–$t9) can be overwritten by any subroutine, while saved registers ($s0–$s7) must be preserved across calls using stack frame prologue/epilogue. Getting this wrong caused correct answers to become incorrect until I enforced proper calling conventions.

2. Low-Level Parsing and Validation Without High-Level Language Tools

   Since there are no built-in parsing functions or exceptions in MIPS, I had to manually validate every character for both decimal and binary input, track buffer bounds, check ASCII values, detect invalid formats, and enforce numerical ranges (0–255). Every edge case like trailing spaces, wrong length and non-digit characters required explicit logic.

3. Implementing Algorithms (Conversion + RNG) Using Pure Assembly

   Binary to decimal and vice versa conversion and random number generation had to be constructed from basic CPU operations like bit shifts, arithmetic, and masking.

   → Binary to decimal: shift-and-add accumulation
   → Decimal to binary: bit masking and iterative division by 2
   → Random numbers: full Linear Congruential Generator using system time as seed

d) Discussion of Algorithms and Techniques

- How the Program Works (Game Loop):
  The main function has two nested loops. The outer loop counts from level = 1 to 10. The inner loop counts from i = 1 to level. Inside this inner loop, the game:
  → Calls generate_problem to get a random mode and number.
  → Calls draw_binary_line or draw_decimal_line to show the problem.
  → Calls read_string (syscall 8) to get the user's answer.
  → Calls a validation function (read_int_str or binstr_to_int) to parse the answer.
  → Compares the parsed answer to the correct answer.
  → Updates the score and plays a sound (play_correct or play_wrong).

After all loops finish, it prints the final score and exits.

- How to Display the Board (drawboard.asm):
  The code checks the ENABLE_GRAPHICS flag.
  → Fancy Mode: It uses syscall 11 (print_char) in a loop 8 times to print | and the _ to create the | _ | _ | look
  → Minimal Mode: It just uses syscall 4 (print_string) to print the plain text, like 10101010
- How to Validate a Move (input.asm):
  I wrote two different parsers:

→ Decimal (read_int_str): This loops through the input string, character by character. For each digit, it multiplies the current total by 10 and adds the new digit's value. It stops at a newline and checks if the final value is over 255. If it sees a letter or the value is out of range, it returns -1

→ Binary (binstr_to_int): This also loops through the string, but it uses bit-shifting. For each character, it shifts the current total left by 1 (sll) and adds 1 if the character was a 1. It also counts the digits. If it sees a non-binary character or the final count isn't exactly 8, it returns -1

e) Interesting ideas which could be implemented

- High Score System: I could use MARS's file I/O syscalls to save the player's high score to a file (highscore.txt). When the game ends, it would read the file, compare the scores, and write the new high score if it's beaten.
- Difficulty Modes: I could add a menu at the start for Easy (4-bit numbers, 0-15) or Hard(16-bit numbers, 0-65535). This would require making the validation and drawing loops use a variable instead of being hard-coded to 8.
- Practice Mode: Add a simple Practice Mode that skips the levels and scoring. It would let the user just endlessly practice one type of conversion (like Decimal → Binary) to help them study.