CSE111 Banana Emulator Final Project - Group 9 Report

Daniel Kim
datkim@ucsc.edu

Milan Moslehi
mmoslehi@ucsc.edu

Max Totten
mastotte@ucsc.edu

Omid Rezaei
orezaei@ucsc.edu

Josh Barsky
jbarsky@ucsc.edu

1 INTRODUCTION

The Banana Emulator Final Project represents a comprehensive effort to design and implement an emulator for the CSE 111 Advanced Programming course at UC Santa Cruz. This report consists of the contributions each team member made towards this project, the challenges we faced, and features that may not be fully functioning as intended.

2 TEAM CONTRIBUTIONS

2.1 Milan Moslehi

- **Tasks and Responsibilities:** Focused on developing the majority of starter code for project files, debugging throughout all files, and organizing efficiency within the project.
- **Contributions:**
  - Worked extensively on the `doInstruction()` function, which takes in an instruction, breaks it down into parts starting with the opcode, and determines if the instruction is of type R or I.
  - Successfully broke down the instruction accordingly, depending on its type, and utilized an unordered map to call functions based on the opcode or function value.
  - Added parameters to all functions (some of which remained unused) to meet the requirement that all functions in the map needed the same parameters, ensuring consistency and improving the maintainability of the codebase.
  - Played a crucial role in the development of the `setup()`, `loop()`, and `dataload()` functions once instructions could be broken down successfully. Implemented the logic for reading from the correct address to find the address to start `setup()`, calling `doInstruction()`, and breaking out when the program counter went to 0.
  - Designed the `loop()` function, using a nested while loop to return to the start of the loop address whenever the counter reset to zero, while continuing in the outer loop, a critical component of the emulator's functionality.
  - Implemented the `dataload()` function to read the information in the data section of the slug file, completing a key part of the file reading and processing functionality.
- **Challenges:**
  - Faced significant challenges after a check-in, where had to refactor the code to be separated into different classes. Required collaboration with team members, especially Omid, to decide which parts of the program belonged in which classes (memory, CPU, banana).
  - Worked closely with Max to separate code and struggled to make it function correctly with separate classes.

  - Eventually used dependency injections to find a solution to implement, ensuring the memory worked correctly between banana and CPU. Without this, changes to memory in the CPU wouldn't reflect in the emulator.
  - Applied the same ideas to the GPU, linking it to the banana class while ensuring it had access to memory.
  - Worked on `handleInput()` to take inputs from the controller and read them into memory.
  - Faced difficulty figuring out why the 'A' button wouldn't show up in `input.slug`, despite changes recommended by TAs and the professor, which resulted in overflow issues still showing as 00000000.
  - Redid the GPU class to correctly read data from memory instead of using hardcoded functions for `box.slug`.

2.2 Max Totten

- **Summary:** Focused on developing code and debugging, examples including helping develop the GPU and CPU functionalities. Implemented various GPU functions, integrated SDL, and restructured the CPU class, supporting the project's technical aspects. Ultimately enhanced the project's functionality and performance.
- **Contributions:**
  - Implemented every GPU function except for `handleInput`, which included: `clearFrameBuffer`, `decodeAndDisplay`, `setPixel`, `moveBox`, `drawBox`, `resizeBox`, `eraseBox`, `init`, and `quit`.
  - Created `gpu.h` to work with the GPU class.
  - Integrated SDL implementation.
  - Adjusted `handleInput` to apply input's effect in addition to reading input.
  - In the CPU, created functions: `add`, `shiftRightArithmetic`, `bitwise_and`, and `jumpRegister`. These functions were based on the document provided.
  - Created enumerations for `Opcode` and `function_codes` in `cpu.h`.
  - Worked with Milan to restructure the large CPU class into OS, CPU, GPU, and memory.
  - Debugged problems with CPU functions by adjusting the `programCounter`.
  - Implemented `clearFrameBuffer` to clear the frame buffer by setting all pixel values to 0.
  - Developed `decodeAndDisplay` to render the SDL window and read from memory for the color value of each pixel, using `getPixelAddress()`, `memory.read8()`, and `setPixel()`.
  - Created `setPixel` to take in coordinates and a color value, setting the pixel at the point to white (color = 1) or black (color = 0).
  - Implemented `moveBox` to first erase the current box, adjust `box_X` and `box_Y` (class private variables), then draw a box in the new location.
  - Developed `resizeBox` to first erase the current box, adjust `box_Size` (a class private variable), and draw a new box.
  - Implemented `eraseBox` to erase the box.
  - Created `init` to initialize the frame buffer, the window, and the renderer.
  - Developed `quit` to destroy the renderer and the window, then call `SDL_Quit()`.

2.3 Omid Rezaei

- **Summary:** Focused on debugging and contributing both with coding and conceptual topics, examples being identifying and rectifying errors in the initial R and I type instruction implementations, suggesting refactoring strategies to enhance code organization, and actively participating in debugging sessions, including resolving issues with the 'A' controller value in `input.slug`.

- **Contributions:**
  - Helped discuss and redo many of the initial R and I type instruction implementations as errors were discovered while running the emulator on the `hello_world.slug` files for check-in 1.
  - Discussed code refactoring with Milan after Check-in 1 to separate out functionality and variables from one giant CPU class.
  - Suggested using Dependency Injection of CPU, Memory, and GPU classes to get the OS class properly working after refactoring based on Check-in 1 feedback.
  - Converted magic numbers to defined constants to improve readability, particularly for address space such as STDOUT and STDERR prints to the terminal.
  - Worked with Josh on homework 5 to set up and install SDL on the GitHub repository and personal machines, which helped get SDL working on both Mac and Windows computers (Max and I were on Windows while the rest of the group used Macs).
  - Worked on `handleInput()` and used SDL to receive controller inputs.
  - Helped debug the 'A' controller value in `input.slug`, tried different masks, looked into the most likely issue of overflowing, considered other issues such as signedness, and consulted a TA and Professor Sifferman about potential other issues.
  - Deleted extraneous files, such as build and temporary testing code files, from the repository, and moved all source code to an appropriately named directory.
  - Created a Makefile that automatically formats and verifies code format (`make format` and `make format-verify` respectively) from the root directory, which was more efficient than setting it up within `cmakelists.txt` directly.

2.4 Daniel Kim

- **Tasks and Responsibilities:** Focused on organizing project files and increasing efficiency, helping develop code for initial files within the project, helping debug throughout the project. in developing the initial code for main.cpp, collaborated with team members to clean and organize the code into different files (Banana.cpp, cpu.cpp, memory.cpp).
- **Contributions:**
  - Helped create and configure YAML files to function correctly and pass requirements.
  - Assisted in using a hex editor to pass check-in 1.
  - Aided in understanding and implementing R and I Type instructions based on the assignment pdf.
  - Implemented specific functions (storeWord, storeByte, orImmediate, and branchOnNotEqual) for the cpu.cpp file.
  - Participated in attempting to debug a loop within Banana.cpp where the programCounter was not functioning correctly, causing unintended looping.
  - Contributed to cleaning the repository by managing `.DS_Store files`, removing unnecessary comments, renaming and restructuring files for better readability, and removing build files when necessary.
  - Implemented enum's throughout project files to increase code readability and efficiency.
  - Helped debug issues in Banana.cpp during the first check-in, where the counter would not stop correctly, leading to incorrect output in slug files.

  - Assisted with GitHub pull request and YAML file issues related to build and release processes.
  - Utilized git pull requests with a branch "Daniel" to implement changes and edits to main without causing errors in the main branch.
  - Organized and wrote the report in LaTeX, ensuring clarity and coherence.

2.5 Josh Barsky

- **Summary:** Focused on helping with setting up the project structure on GitHub, providing skeletal structures for the main loop and ‘cpu.h‘ class, debugging initial code issues, installing SDL, and maintaining the repository by removing unnecessary files and updating formatting and build files.
- **Contributions:**
  - Set up the initial project structure on GitHub and helped establish the skeletal structure for the main execution loop.
  - Assisted in debugging the "hello world" issue by suggesting that the iterations might be going out of bounds, leading to a solution.
  - Worked on installing the SDL import to use a GUI interface and attempted to get the GUI working, although faced issues with the GUI on his system.
  - Provided a skeletal structure for the ‘cpu.h‘ class and replaced many magic numbers to improve code readability.
  - Conducted organizational work such as deleting unnecessary code and unused files.
  - Updated the formatting and build YAML files, ensuring the format command passed and the build commands were correct.
- **Challenges:**
  - Struggled with the GUI not working on his system, despite numerous attempts to sync with the team’s progress through GitHub.
  - Felt less confident in his coding abilities compared to his team members and found it challenging to contribute effectively to the codebase.
  - Experienced difficulty in motivating the team due to his limited programming experience, which sometimes led to feeling demotivated.
  - Faced challenges in understanding what was expected of him in the project, which led to seeking external resources like Cherno videos to relearn coding and understand C++ better.
  - Had difficulty getting the program to work on his system, particularly with the GUI, which affected his ability to contribute to that part of the project.

3 DEPENDENCY INJECTION AND GPU IMPLEMENTATION

- The GPU class was the hardest for us to implement. As a group we struggled to get our heads around how the GPU was set up and that it should be reading into the VRAM every iteration of loop(). We initially developed a system that correctly read into the VRAM at first, such that image.slug would load, but box and input didn’t. We then started to work on accepting inputs and writing them into the correct place in memory, however as of this report we are still struggling to implement A because of an overflow error. As for box.slug our initial idea was to just implement a box in SDL2 and give it the needed functionality. However, we were later informed that the code needed to run by updating the VRAM and loading in the new states of the screen frame by frame, using instructions from the .slug file. After this we reorganized the GPU file and wrote a new version of decodeAndDisplay() which successfully would read the

VRAM addresses, and load in the correctly colored pixels on screen.

- In our project, we utilized dependency injection to seamlessly integrate the GPU, CPU, Memory, and filename handling into our OS class, which we named Banana. This approach allowed us to efficiently manage dependencies and enhance the modularity of our codebase. By structuring our classes in this way, we were able to improve the overall organization and maintainability of our emulator project.
- The CPU class implements the registers and the instructions that use them. There are two instruction types, R and I type respectively. The program counter is used to find the next instruction to execute and the stack pointer register respectively keeps track of the call stack. Some instructions manipulate these values. Group members split up implementation of these instructions. The Memory class enables reading and writing from memory. Most crucially it's filereader function enables reading SLUG file contents. There are also functions that read and write to 8 and 16 bits.
  This was espeically helpful in implementing CPU (load and store instructions) as well as GPU (properly rendering pixels).

4 UNSOLVED FEATURE

In our project, there's a feature where pressing the "A" input should shrink a box, but it's not functioning as expected. We attempted to resolve this by adjusting the signedness of the registers and modifying certain instructions, suspecting an overflow issue that may be causing the value to be set to 0. Despite our efforts, the problem persists, suggesting a more complex underlying issue that requires further investigation.

5 CONCLUSION

In conclusion, this project has been an intriguing journey into the realm of emulator development, offering a hands-on experience in tackling complex technical challenges. The process of implementing and debugging various functionalities, such as handling inputs and ensuring proper execution of instructions, has been both rewarding and enlightening. Despite encountering some obstacles, such as the 'A' button input issue, the team's perseverance and collaborative efforts have been instrumental in making significant progress. This project has not only deepened our understanding of computer architecture and emulation but also honed other skills such as within C++, GitHub, Makefiles, and overall problem-solving and teamwork skills.

ACKNOWLEDGMENTS

We appreciate Professor Sifferman and TA's Benjamin Princen and Gurpreet Dhillon for clarifications and support throughout this project during section and office hours.