68000 Assembly Game - Assembly and C Module - Year 2 Stage 2

Software Development and Cyber Security

Project Specification: Convert 68000 Assembly Code to x86_64

Project Due: Friday 3rd May 2024 (9.00am) no GitHub commits after 9.00 am
Project Demonstration and Grading: Friday 3rd May 2024 9.00am to 5.00pm
Project Quiz: Friday 3rd May 2024 9.00am

1. Objective: The objective of this project is to convert the provided 68000 assembly code to x86_64 assembly code while maintaining functionality and correctness.

2. Input: The input for this project is the provided 68000 assembly code found in the "Tutorial4 by Charles Kelly" "http://www.easy68k.com/examples/tutorial4.zip" file.

3. Output: The output of this project will be the equivalent x86_64 assembly code that performs the same operations as the original 68000 assembly code.

4. Requirements:

- The converted x86_64 assembly code should accurately replicate the functionality of the original 68000 assembly code.
- All variable names, comments, and labels should be appropriately translated to x86_64 syntax.
- The code should be well-structured and easy to understand.
- The converted code should be optimized for performance where possible.
- Proper error handling should be implemented, including checking for potential overflow or underflow conditions.

5. Tools and Resources:

- An x86_64 assembly language development environment such as NASM (Netwide Assembler).
- A debugger GDB for testing and troubleshooting the converted code.
- Documentation and references for x86_64 assembly language syntax and instructions.
- The original 68000 assembly code as a reference.

6. Deliverables:

- A complete set of x86_64 assembly files corresponding to the original 68000 assembly code.
- A README file explaining any significant changes or considerations made during the conversion process.

- A test plan and test cases to verify the correctness and functionality of the converted code.
  Test Scripts C (create a separate C file to test assembly code)_
  https://www.tutorialspoint.com/c_standard_library/assert_h.htm
  https://libcheck.github.io/check/index.html
  Assembly Macros
  http://blog.code-cop.org/2015/08/how-to-unit-test-assembly.html
- Capture a brief 10 to 20-second video of software being executed in the command line after it has been ported.

Project Rubric		
0 - 35% **(0 - 8)** **Basic**	**35% - 75%** **(8 - 18)** **Intermediate**	**75% - 100%** **(18 - 25)** **Advanced**
• Implementation will achieve minimum functionality. • Implementation may contain some syntax and/or run-time errors. • Implementation code will be poorly commented and/or formatted. • Implementation will contain basic features; application will not be tested properly. • Implementation code will not follow applicable coding conventions.	• Implementation will achieve expected functionality. • Implementation will not contain syntax and/or run-time errors. • Implementation code will be reasonably commented and/or formatted. • Implementation will contain assignment features. • Implementation will be tested to a reasonable degree. • Implementation code will follow appropriate coding conventions.	• Implementation will achieve advanced functionality. • Implementation will not contain syntax and/or run-time errors. • Implementation code will be well commented and/or formatted. • Implementation will contain assignment features. • Application will be expertly tested. • Implementation code will follow coding conventions.
Correctness • All operations and calculations produce identical results to the original 68000 assembly code. • The converted code passes all provided test cases without errors or discrepancies. Code Clarity and Readability • Variable names, comments, and labels are clear and descriptive. • The code is well-structured and easy to follow. • Proper indentation and formatting are used to enhance readability. Performance Optimization • The converted code demonstrates optimization techniques where applicable. • Redundant operations or instructions are eliminated to improve performance. • Efficient memory usage and register allocation strategies are employed. Error Handling • The converted code includes proper error handling mechanisms. • Potential overflow, underflow, or boundary conditions are checked and handled appropriately. Documentation and Testing • The README file provides clear explanations of any significant changes or considerations during the conversion process. • A comprehensive test plan with test cases is provided to verify the correctness and functionality of the converted code.		