RISC-V Disassembler C Code Conversion Report

OpenAI

July 22, 2024

1 Task

The task was to convert an existing RISC-V disassembler written in C++ into an equivalent C code. The primary goal was to ensure that the functionality remains consistent while adapting the syntax and structure to be compatible with the C programming language.

2 Approach

1. **Understanding the Original Code**: The original C++ code was thoroughly reviewed to understand its structure and functionality. This included examining the flow of the main function, helper functions, and data structures used.

2. **Mapping C++ Constructs to C**: Specific constructs and functions in C++ were mapped to their equivalents in C. This included replacing C++ specific features like `string` with C-style strings and handling file I/O using standard C functions.

3. **Function and Variable Translation**: Functions and variables were translated from C++ to C, ensuring that memory management and syntax were appropriately handled.

4. **Testing and Validation**: The converted C code was tested with sample inputs to ensure that it produces the same output as the original C++ code. Any discrepancies were addressed by debugging and refining the code.

3 Testing

The testing phase involved running the converted C code with various hexadecimal input values to validate the correctness of the disassembly process. The outputs were compared with the expected assembly instructions to ensure consistency. The following steps were followed:

1. **Setup**: Prepared input files with hexadecimal values representing RISC-V instructions.
2. **Execution**: Ran the C code to generate the disassembled output.
3. **Verification**: Compared the output with manually verified assembly instructions to check for accuracy.
4. **Debugging**: Any issues encountered during testing were debugged, and the code was updated to fix any errors.

4 Conclusion

The conversion from C++ to C was successful, with the final C code maintaining the same functionality as the original C++ code. The disassembler correctly translates hexadecimal RISC-V instructions into human-readable assembly language. This exercise demonstrates a thorough understanding of both C++ and C, as well as the ability to adapt and maintain code functionality across different programming languages.