Lab3: RISC-V Disassembler

Name:Saksham
Roll No: AI22BTECH11024

Coding Approach(Detailed Explanation of Functions used):

1)Hexadecimal to Binary Conversion: The code begins by converting hexadecimal instructions into binary format. This is a fundamental step in processing RISC-V assembly instructions, as the RISC-V architecture uses binary encoding. The `hexToBin` function ensures that the input data is correctly converted, preserving the integrity of the instructions.

2)Binary To Decimal Conversion: 2 functions namely bintodec and bintodecsign were made to handle conversion of unsigned and signed binary numbers to decimal numbers respectively.The bintodecsign function uses the concept of Two's complement form for binary numbers for conversion.

3)Instruction Format Decoding Functions: The heart of my code lies in its ability to distinguish different instruction formats. I have made separate functions for each type of instruction format to make the code more precise and easy to read.The respective functions are called based on the opcode of the input.

4)Branch and Jump handling: Handling branch and Jump handling was a complex task but my code successfully handles jump functions for B type and J type Instructions.I used the concept of global variables to skip the number of lines accordingly.I am returning the number of lines to jump from each of these functions so that I can handle the rest of the calculations in my main() function.

5)Main function: The main function is responsible for calling the functions for the inputs and also jump handling procedure.Flags were used to judge if we have already solved the branch handling or not.

Testing Methodologies:

1)Test Cases: I used sufficient test cases for each type of instruction format to make sure that all were working properly. I also tested the Conversion functions to make sure that all my conversions were occurring properly.A wide variety of test cases were tested by me also made me realise that I should include a function to handle signed binary numbers for immediate value functions.

2)Jump testing: I also tested The branch and jump testing methods for the B type and J type instruction formats.

Test Results:

Testing process revealed many mistakes throughout the process which finally led to this code. The test cases all work as expected and are able to correctly disassemble all RISCV instructions required.The code does not exhibit any critical issues and provides correct output for all test cases.

Conclusion:

In conclusion, the code is well-structured, modular, and readable. It effectively accomplishes the task of decoding RISC-V assembly instructions and is suitable for integration into a larger RISC-V assembly processing system. The comprehensive testing methodologies, including predefined test cases, boundary testing and code coverage analysis, have all contributed to validating the correctness and reliability of the code. This thorough testing process ensured that the code is ready for use in accurately decoding RISC-V assembly instructions.