RISC-V Disassembler

CS2323 Computer Architecture

Lab Report

Edward Nathan Varghese - CS22BTECH11020

October 12, 2023

1 The Task

For this lab assignment, we were tasked with converting given assembly instructions into their corresponding RISC-V syntax using C/C++.

2 Approach

I selected C++ as my programming language for this assignment and followed these steps:

1) **Reading from the file:** I utilized the `<fstream>` header and the `freopen()` function to read the content from the `input.txt` file. Each line was processed individually.
2) **Converting Hex to Binary:** To convert hexadecimal values into binary strings, I initialized a hashmap with hex-binary conversions using the `initializeMap()` function. The `HexToBinary()` function was implemented.
3) **Writing peripheral functions:** I improved code modularity by creating functions such as `BinToReg()` and `BinToImm()`. These functions transformed binary substrings into the desired string format (e.g., "x5" or "190").
4) **Bringing it all together:** The core of the code was constructed through the `BinToAssembly()` function. This function transformed binary strings into assembly language, utilizing peripheral functions. The function operated as follows:
    1) I retrieved the last 7 characters and stored them as the opcode.
    2) The opcode was employed to categorize functions into R, I, S, J, U, and B types.
    3) Using the known bit mappings of these types, I extracted vital information such as binary values for immediates, funct3, funct7, rd, rs1, and rs2 when applicable.
    4) I combined all peripheral functions, accounting for both signed and unsigned values, to construct the resulting assembly syntax string.
    5) I maintained a global map that recorded all program counter (PC) points where labels may be inserted if offsets to specific instructions were required.

5) **Fixing the labels:** I created another map to assign textual labels, starting from L1, to all instructions that could be accessed via branches or jumps. I then iterated through the assembly instruction strings, replacing offsets with these labels where applicable.

6) **Validation and error handling:** After implementing the program's logic, I handled certain errors and further modularized the code. Comments were added to enhance code readability.

3 Testing

I wrote assembly instructions in the Ripes Simulator and used the generated hexadecimal instructions to validate the correctness of my code. I further wrote my test cases and used peers' test cases to ensure that my code worked correctly for all the valid inputs it was presented.

4 Learning Outcome

Through this assignment, I have learned how to convert hexadecimal instructions into RISC-V syntax using C/C++, enhancing my programming skills and deepening my understanding of computer architecture and organization. I have gained hands-on experience in reading files, processing data line by line, and using various functions to perform hex-to-assembly conversion, creating a modular and efficient code structure.