Computer Architecture - CS2323
Lab-3 RISC-V Disassembler

Manohar Maripe
CS22BTECH11036

October 12, 2023

Abstract

This report provides an analysis of a C program for decoding RISC-V assembly instructions. The program takes binary-encoded RISC-V instructions, converts them into human-readable assembly instructions, and handles labels for branch and jump instructions.

1 Introduction

The program under analysis is a C program that decodes binary-encoded RISC-V instructions .The code is a program designed to read hexadecimal-encoded instructions, convert them into binary, and then parse and translate these instructions into RISC-V assembly language instructions.It also manages labels for branch and jump instructions. The report covers various aspects of the code, including its purpose, structure, functions, and potential improvements.

2 Code Overview

The code serves the following main purposes:

- Reading instructions from a file.
- Converting hexadecimal instructions to binary.
- Parsing binary instructions and converting them to decimal to identify opcode, funct3, funct7, rd, rs1, rs2 and other immediate fields.

- Functions to handle different types of RISC-V instructions: R-type (register-register), I-type (immediate), S-type (store), B-type (branch), U-type (upper immediate), and J-type (jump).
- Translating binary instructions into corresponding RISC-V assembly instructions.
- Handling label resolution and control flow instructions (jumps and branches).

3 Program Structure

The program consists of several C functions, each responsible for a specific type of RISC-V instruction decoding. The main components of the program are:

- `hexDigitToBinary`: A function to convert a hexadecimal character to a binary string.
- `hexToBinary`: Converts a hexadecimal string to a binary string.
- `binaryToDecimal`: Converts a binary string to a decimal number.
- `breakCharArray`: Extracts a subarray from a character array based on start and end indices.
- Functions for identifying `opcode`, `funct3`, `funct7`, `rd`, `rs1`, `rs2`, and various `immediate values`.
- Instruction decoding functions (`rtype`, `itype`, `stype`, `btype`, `utype`, `jtype`): These functions decode specific types of RISC-V instructions based on opcode and other fields.
- `converttoinstruction`: Determines the instruction type and calls the appropriate decoding function.
- `btypelabel` and `jtypelabel`: Handle label storage and indexing for branch and jump instructions.
- `decryptinstruction`: Converts a hexadecimal instruction into a human-readable assembly instruction.
- `main`: The main function reads binary instructions from a file, processes them, and prints the resulting assembly instructions with labels.

4 Label Handling

Labels are stored in arrays and indexed for future reference. Labels by storing label addresses and associating them with instruction indices. It uses separate arrays to track label addresses and the corresponding instruction indices. Labels are used to resolve branch and jump instructions.

5 Usage

To use the program, provide a text file (e.g., "input3.txt") containing binary-encoded RISC-V instructions. The program will read the file, processes each line as a hexadecimal instruction. It prints the corresponding RISC-V assembly language instructions to the console. It also handles conditional branching and jumping instructions by labeling lines with "LX" labels, where X is a unique identifier for each branch or jump target.

6 Conclusion

The program provides a robust solution for decoding RISC-V assembly instructions from binary code and managing labels for branch and jump instructions. It is a valuable tool for developers and researchers working with RISC-V architecture.