# RISC-V Instruction Encoding Implementation Report


| Student Name | Roll Number |
|---|---|
| Dheeraj Agarwal | CS23BTECH11016 |
| Nenavath Kalyan | CS23BTECH11039 |


## 1 Introduction

This report details the implementation of a RISC-V assembler that converts assembly code into machine code. The implementation is spread across multiple C++ files and includes functionality for parsing assembly instructions, handling various RISC-V instruction formats, and generating the corresponding hexadecimal machine code.

## 2 Implementation Overview

### 2.1 Header File: Instruction.h

The header file `Instruction.h` declares functions and data structures essential for parsing and encoding RISC-V instructions:

- `void Parse(vector<string> program)`: Parses the assembly program and generates the corresponding machine code.
- `string trim(const string &str)`: Trims whitespace from the beginning and end of a string.
- Global Variables:
  - `unordered_map<string, int> registerMap`: Maps register names to their numbers.
  - `unordered_map<string, int> labelAddress`: Maps label names to their addresses.

### 2.2 Source Files

#### 2.2.1 main.cpp

The `main.cpp` file is responsible for:

- Reading Input:
  - Reads assembly code from `input.s`.
  - Strips comments and trims lines.
- Label Address Pass:
  - First pass through the program to record label addresses.
- Parsing and Encoding:
  - Calls `Parse(program)` to convert assembly instructions into machine code.

#### 2.2.2 Instruction.cpp

The `Instruction.cpp` file implements the following:

- `Parse()` Function:
  - Parses each line of the assembly program.
  - Identifies and processes different instruction types (R-type, I-type, B-type, etc.).
  - Handles immediate values and label addresses.
  - Writes the generated machine code to `output.hex`.
- Instruction Conversion Functions:
  - `R-type Instructions`:
    - `convertRType()`: Encodes R-type instructions by combining `funct7`, `rs2`, `rs1`, `funct3`, `rd`, and `opcode` fields.
  - `I-type Instructions`:
    - `convertIType()`: Encodes I-type instructions using `imm`, `rs1`, `funct3`, `rd`, and `opcode` fields.
  - `B-type Instructions`:
    - `convertBType()`: Encodes B-type instructions with immediate fields split into various components.
  - `S-type Instructions`:
    - `convertSType()`: Encodes S-type instructions, focusing on immediate values and register fields.
  - `U-type Instructions`:
    - `convertUType()`: Encodes U-type instructions with immediate values and `rd`.
  - `J-type Instructions`:
    - `convertJType()`: Encodes J-type instructions using split immediate values and `rd`.

# 3 Testing

## 3.1 Testing Approach

### 3.1.1 Unit Testing

- Validated individual encoding functions with known inputs and expected outputs.
- Ensured correct bitwise operations and field placements.

### 3.1.2 Integration Testing

- Tested the entire parsing and encoding process with sample assembly programs.
- Verified output hexadecimal values against expected results which we get in ripes.

### 3.1.3 Error Handling

- Tested error handling for invalid operands, out-of-range immediate values, and undefined labels, invalid register alias, wrong number of operands.
- Ensured appropriate error messages were generated and written to the output.