Computer Architecture - CS2323
Lab 3
Report : RISC-V Assembler Implementation

Devraj ES22BTECH11011

Introduction

This project focuses on the implementation of a RISC-V assembler using C++. The assembler converts assembly instructions into their corresponding machine code in hexadecimal format.

Instruction Encoding

The implementation uses structures and unordered maps to store instruction encodings and register mappings, which allow for efficient conversion and error handling.
Structures are used to hold encoding details for various RISC-V instructions, namely the encoding format, opcode, funct3, and funct7 fields.
An unordered map is employed to store the mappings of instruction names to their encoding fields. Additionally, register names are mapped to their 5-bit binary equivalents.

Input Handling

Input is taken from a file called `input.s`, which contains the assembly instructions. These instructions are stored in a 2D vector, where each row corresponds to one line from the input file, split into individual words.
The parsing process is done using `getline()` and `stringstream` functions to handle spaces, commas, and parentheses.

Number System Conversions

Several helper functions are used for number system conversions:

- Conversion of decimal values to binary strings.
- Conversion of binary strings to hexadecimal.

- Handling negative numbers using two's complement.

These functions ensure that immediate values and register addresses are correctly converted to their respective formats.

Error Handling

The assembler includes comprehensive error handling, covering the following scenarios:

- Invalid instruction or register names.
- Incorrect number of operands for instructions.
- Immediate values exceeding valid ranges.
- Duplicate or missing labels.

Errors are reported with the corresponding line number for easy debugging. If an error is encountered, the assembler halts further processing to avoid producing incorrect output.
Errors are printed onto the terminal.

Instruction Encoding Functions

The assembler implements different encoding functions for each instruction format (R, I, S, B, U, J).
These functions build the binary representation of each instruction based on its format and then convert it to hexadecimal. The results are written line by line to an output file (`output.hex`).
The conversions and error handling are taken care of by these functions.

Testing

The program was tested with assembly code given in the Lab problem statement, as well as other instructions, and the results were verified with the RIPES simulator.
Many erroneous instructions were also fed into the program to test the error handling capabilities of the implementation.

Conclusion

This C++ implementation of a RISC-V assembler successfully translates RISC-V assembly instructions into machine code. By using unordered maps, efficient number conversions, and extensive error checking, it ensures reliability and correctness. Testing was conducted on a variety of assembly programs, demonstrating both functionality and robustness in handling errors.