ReadMe

Date:-August 2024

Abstract

This document serves as the README for the RISC-V Assembler project. It provides an overview of the project, installation instructions, usage guidelines, and details about the API and functionality of the assembler.

1 Acknowledgements

1.1 Thanks and Regards

We (our team) are grateful towards such nice and informed instructor, as well as our classmates who came up with the un-imaginable edge cases for the edge cases and complex cases handling as well as the syntax checking.

1.2 Ripes Software

We are also grateful to have the Ripes software designed by the Morten B. Petersen, used to cross check the results (or output) given by our code, as well as to know how-abouts of the syntax (or semantics).

1.3 Github repositories:

repositories designed by Nikunj Taneja were of massive help to get to know about the actual code of some standard algorithms and to get to know about how much complex these codes can be, and what our assembler should do to combat these complexities.

1.4 Universities and Institutions:

We must thank the IITH, for providing the mentor for the project which helped a lot to understand intricacies of the project in detail...

Also we would like to thank University of Tennessee, Knoxville for providing example RISC-V assembly programs...

2 API Reference

2.1 Use of inbuilt File IO functions:

Used following functions to flexibilize the reading and writing operations from or in the file.

- .tellg()
- .seekg()

2.2 In-built functions used from CPP standard library:

Functions like

- to-string
- stoll
- string::npos
- .find
- string concatenation overload

2.3 Utility functions:

2.3.1 To-Binary:

This function takes a string in decimal number base and converts it into the binary number base.

2.3.2 To-Hex:

This function takes a string which is in binary number base and converts it into the Hexadecimal number system.

Note: We could have created the To-Hex function which takes decimal number string as input and returns direct hex based string, but we wanted those binary bits to build 32 bit instruction step by step...

2.4 Is-(type) functions:

We wrote the function for identifying what kind of the instruction which we read from the file, these funtions are is-R,is-S,...is-J. Which also helps in syntax (or semantics) checking...

2.5 Hard-Coded things:

2.6 Binary-Storage Map

We hard coded the map which has *string* and *string* as its key-value pair, in which we stored the 5 bits which represent the binary number associated with the standard registers x0 to x31.

2.7 Is-(type) functions:

Hardcoded all the possible strings which are possible instructions present in the RISC V ISA (Instruction Set Architecture).

We hope you would find easier to extend or build up on our code with this much of a API reference...

3 Usage instructions:

3.1 Pre-requisites:

3.1.1 C++20 (GCC 13-64) language compiler

Ensure that you have g++ compiler which can compile source files written in C++20 (GCC 13-64)...

3.1.2 Make Utility:

Make utility should be installed on the device to smoothly ensure the build process.

3.1.3 Permissions:

Ensure you have the necessary file permissions to read from the input file and write to the output file in your working directory.

3.1.4 Operating system:

The project is built to be cross-platform, but the main tests have been performed using Linux. Make sure it works on your operating system or be ready to change some settings if needed.

4 How to run the code:

Make sure that all the program files including the makefile are present in the same directory, following are the program files:

- Assembler.hh
- main.cpp

- R.cpp
- I.cpp
- S.cpp
- B.cpp
- U.cpp
- J.cpp
- conversions.cpp
- makefile
- input.s

Now run the code using command **make** and it will create the executable file named riscv_asm , now run it by typing the command ./riscv_asm , and after the execution is complete, you will see the output.hex file is generated and has the hexadecimal numbers which expand to the 32 bit binary instruction...

5 Documentation

5.1 Project Overview:

This project is aimed to build RISC-V assembly language code decoder, which as its name suggests, converts RISC-V assembly language instructions into the machine understandable 32-bit binary instructions. The C++ code is written in the C++ 20 version of the language which uses some File I/O operations to read from the input file line by line, and accordingly convert them to 32-bit instructions and writing them on the output.hex file.

The instructions will be converted into Hexadecimal format before getting written to the output.hex file...

5.2 Instructions that our assembler supports:

Out assembler will be able to convert the assembly level language with RISC-V architecture to the machine understandable binary code, and supports R,I,S,U,B,J type instructions present in RISC-V ISA (Instruction Set Architecture).

6 Appendix:

6.1 More about File I/O:

6.1.1 *fstream*

The header fstream belongs to the C++ Standard Library. It contains facilities to perform input and output operations on files. It includes classes like ifstream,

ofstream, and fstream, which can be used to operate on files in different modes-for example, reading, writing, and both.

These classes in this header are inherited from base classes istream and ostream which can provide a full range of input and output capabilities for files, just like cin and cout do for standard input and output.

6.1.2 *seekg()*

seekg() is an inbuilt function used along with input file streams-ifstream or fstream. It moves the "get" pointer that handles the current position in the input sequence of a file. In other words, by moving this pointer, you will change the position where the next input operation will begin.

6.1.3 *tellg()*

tellg() is an inbuilt function used along with input file streams, which are ifstream or fstream. It returns the current position of the "get" pointer in the input sequence. In other words, it will return the position of the pointer currently in a file. It is pretty useful when determining the size of a file or how much of a file has been read.