CS2323 - Lab 3 - RiscV Assembler Report

Shiven Bajpai - AI24BTECH11030

Contents

I. Preface

This Report outlines the process of creation of this Assembler. The aim was to have it work with all base class instructions, however pseudo instructions were left out. There is however a section at the end briefly detailing how one may add pseudo instructions to this assembler.

II. Breaking down the problem

At its core, parsing assembly instructions involves 3 simple steps:

1) Read the instruction
2) Convert it to binary
3) Write the binary code to the output file/buffer

Since every instruction is processed sequentially, The entire process looks as thought it can be implemented in a single loop. However it is not so. Say there is a jump or branch instruction with a Label that is declared on a later line, if we are reading and converting line by line, we wouldn't have reached that label yet and thus wont be able to convert the current instruction to binary.
To solve this issue we use two passes. The first pass looks for Labels and notes down the address to which each label points. Then in the second pass we actually convert the instructions, which can be now be down without any issues since we already know the position of all labels in the code.
For convenience, we also remove excess whitespace in the first pass.

Theoretically this could be also be done in a single pass by deferring the conversion of all jump/branch statements to the end of the pass. However my assembler goes with two passes for simplicity
The rough control flow can be written as follows:

1) Load the file
2) Read every line, for every line
   a) If there is a flag declared, note down the flag's position
   b) Remove whitespace/flags and write it to a temporary buffer
3) Read every line from the temporary buffer, for every line
   a) Read the name of the instruction
   b) If the instruction is invalid, exit gracefully
   c) Based on the instruction, try to read the arguments and check for validity
   d) If the arguments are invalid, exit gracefully
   e) Convert to binary and write to another temporary buffer
4) Write the full binary code from the second temporary buffer to the output file

III. Implementation details

IV. Testing

V. A Note of Thanks