Computer Architecture Lab Assignment 3

Divya Rajparia

Building a RISC V Assembler

September 8, 2024

This report has 3 main sections:

1. **Program Design:** Which explains my approach to the problem, and the procedure followed to convert the assembly language code to machine code
2. **Errors and Testing of code:** This section describes the kinds of errors my assembler is capable of handling, as well as the testing conducted to ensure correctness of code
3. **Assumptions Made:** Which explains the assumptions my assembler will make about the input assembly code

Program Design

I have explained below the procedure followed to convert the assembly instructions to machine code. Please note, in order to prevent being repetitive, I have written a general procedure which encompasses all the instruction types. I have marked in red the errors that would be thrown at each step of the conversion, and the reason for them as well.

In case of an error, the program prints the line number, as well as reason for the error into the terminal, and exits.

2 pass method to detect labels and instructions:

My assembler supports all formats: R, I, S, B, J, and U. I have structured my program to perform 2 passes of the input assembly code:

i) In the first pass, I find the location of all the labels in the input code, and I store their name as well as line number, in a map.

ii) In the second pass, the assembler detects the first non-label word of each line, which is expected to be the instruction. Based on the type of instruction, the relevant format function is called.

Error thrown if the first non-label word is not in the list of valid instructions.

Once we have detected the instruction type, the entire line, which contains the instruction name, as well as its operands, is sent to the respective function.

General function design for all formats:

1) The first step in all formats is to check the number of words in the instruction line (excluding labels, if any).
Error thrown if the number of operands is incorrect

2) Once we have checked if the number of operands are correct, we now remove leading and trailing commas from the operands(registers/immediate values), using the removeCommas function.

We now do special error checks and processing for the operands which are registers, and immediate values

3) Error checks and processing done for register operands:

i) Alias Removal: The register name passes through an alias remover, which changes the name to the standard x0 to x31 for all registers.
Error thrown if the register name is invalid, i.e. it is neither x0 to x31, nor is it an alias name
Now, the name of the register is free from aliases. We do one more check to confirm that the register is indeed from x0 to x31.
Error thrown if the alias removed register name is out of bounds, i.e. not in 0 to 31

ii) Once we have verified that the register name is valid and in bounds, we convert it to its corresponding 5 bit binary number.

At the end of this, we have converted the register to its equivalent 5 bit binary string in the case of no errors.

4) Error checks and processing done for immediate values

i) We first check if the operand value is indeed numeric
Error thrown if the immediate value is non-numeric

ii) We then check if it is in its allowed range
Error thrown if the immediate value is out of range

iii) Once we have verified that the immediate is valid and within its allowed range, we convert it to its corresponding binary string with the required number of bits(according to the instruction).

5) Instruction specific things - opcode, funct3, funct6, and funct7
We now get the values of opcode, funct3, funct6, and funct7 for the respective instruction formats which need them, and store i as a binary string.
I have written if statements for each instruction of each format, to get these values.

6) Putting it all together
We now have everything we need for the machine code, in binary string format: registers, immediate values, opcodes, and function values. We now concatenate all of these strings according to the required instruction format, and print it in the out.hex file.

Testing and Correctness of Code:

Errors Handled:

1) In the first pass, if the same label name is found twice, at different lines in the code, error is thrown.
Example:
L3: ori x1, x2, 18
blt t6, x0, L3
L3: or gp, tp, t0

2) If the instruction is not in the list of available and valid instructions, error is thrown.

3) If the **number of operands** are incorrect, error is thrown.
Example:
lui x6
blt t0, t1, L1, t2

4) If any of the operands are incorrect, **registers**, **immediate values**, or **labels**, an error is thrown. The errors are:

i) If the register name does not match either the real names or the alias names, an error is thrown
Example: xori aa, a3, 100

ii) If the register number is out of bounds, an error is thrown
Example: sd x34, 0(x38)

iii) If the immediate value contains characters which are non-numeric, an error is thrown
Example: slli x15, x16, Computer

iv) If the immediate value is not in the specified range, and error is thrown
Example: lhu x23, -2049(x24)
item In case of B and J format instructions, if the name of label operand is incorrect, i.e. it is not their in our map which we created in the first pass, an error is thrown.

v) If the jump offset for the label is out of range, an error is thrown.

Testing Conducted

I created 4 test case files which cover **all instructions**, **all register names**, **all alias names**, as well as **all edge-case immediate values**.

There are different files for different instruction types - one for I, one for R, one for S, and a combined one for B, J and U. These files are attached with my submission.

Assumptions Made:

Format of jalr instruction

I have assumed the format of the jalr instruction to be jalr rd, imm(rs). In RIPES, the format used is slightly different from what we learnt in class, it is: jalr rd, rs1(imm). However, I have stuck to the format we learnt in class.

Labels

I have assumed that lines in the input code which have labels will be of the below format:
L2: bgeu sp, x3, L1   i.e. Label name, colon, followed by instruction on the **same line**.
L2:
        bgeu sp, x3, L1   will not be accepted as valid.

Line spacing and commas

I have assumed that all the operands have a space after the comma, and not before the comma.
Example: blt t6, x0, L3 is correct. But blt t6 , x0 , L3 will not be considered to be valid.

lui instruction

i) I have assumed that the immediate values for the lui instruction will only be entered in decimal value.

ii) I have considered 20 bit immediate values, and anything out of the 20 bit range will be flagged as error.

Jalr, Load and Store instructions

I have assumed that there is no space between register name, parenthesis, and immediate value.
Example: sd x29, 0(x30)
If there is a space between 0 (x30 it will not be accepted as a valid instruction.
The same applies to load and jalr instructions as well.