RISC V Assembler

September 8, 2024

1 Implementation

1.1 Initialising

- A map(reg) was used to store the mapping between register names and corresponding value(0-31). This was simplified using store function.
- Similarly a map(I) was used to store names of various instructions of various formats seperately mapping them with combination of funct3 and funct7 values.
- Data storage is done using vector of strings in which each line is read through ifstream and pushed into line(storage) until EOF. Empty lines are skipped.
- The input is parsed once to store all labels in the code mapped with their corresponding program line count, using a map(label).

1.2 Helper functions

- The function hex takes a 32 bit number as input and outputs a string in hexadecimal format.
- The function option is used to choose between various formats of instructions to which the operands are passed.
- The function **check** is used to check whether the input string can be converted to a valid integer within the range allowed and stores the value in immediate, accordingly returns a boolean value which is used for error handling.

1.3 I/O and variables

- The input is received from **input.s** and output is printed in **output.hex** in hexadecimal format. More about this in README file.
- The unsigned int mcode is used to store a 32 bit machine code which is generated from the instruction and converted to hex.

- The array **o** is used to store the words the in each line which is later processed to decode the format, registers and offset/immediate value.
- the variables line_ct and p_ct correspond to the line counter and program line counter.(Both can vary due to labels)
- the variable immediate is used to store the constants in instructions or offset calculated from labels.

1.4 Various Formats

- Each format instructions are processed as seperate functions, I and S format are clubbed together as majority of their code overlap. The operation code is hardcoded for each instruction format.

1.4.1 R - format

The words corresponding to destination and source registers are checked for correctness, Then the machine code is generated using the addition, left shift operation and mapped values.

1.4.2 IS - format

The load and store operations are processed together as they have same structure, parantheses and offset values are processed and verified for correctness.
The jalr instruction is processed seperately and label is converted to corresponding address.
The mcode is created similar to R - format seperately for S and I format.

1.4.3 B - format

The branch instructions are checked similarly word by word for correct operands and labels are converted to corresponding offsets.
The mcode is obtained by splitting the immediate value and inserting the corresponding bits in respective locations.

1.5 J - format

The J format including only jal instruction has only 2 operands and allows 20 bit immediate which is jumbled in the machine code. The immediate is obtained similar to B - format, the last bit in case of constants is made 0.

1.6 U - format

The U format which supports lui is used to load 20 bits and shifts it left by 12 bits.
The immediate value provided which is positive is taken modulo 2^{20}.

1.7 Error Handling

- Various functions are used to print various error messages along with line number in the terminal and exit the program.
- The function rd is used for checking destination register, rs for source register, lb for label/offset values, op for number of operands, op1 for checking existence of operation.
- label_check is used to check for valid label name and duplicate labels.(valid C variable names are applied to labels).

1.8 Main

- The main function reads from input.s and writes through file pointer out to output.hex , calling various functions mentioned above while parsing through the file. Labels at the end of the input are neglected.

1.9 TESTING

- The testing of the correctness of the code was done in various time instants during development of this program.
- The parts which were remodified were retested for correctness with old testcases.
- Few important testcases are provided in file Sample_input.s
- Several wrong instructions were also tested to check working of error handling and comparison with Ripes Simulator was done for each test case tested.
- The constraints on the instructions are provided in the README file, the program is intended to work correctly with those instructions.
- Debugging was done mostly by printing parts corresponding to variation in machine code (ex. if variation in beginning then issue with immediate values)