Computer Architecture

Lab-3 (RISC-V Disassembler)

Introduction :

This report contains a short description of the coding approach I have used in the development of a RISC-V disassembler. The disassembler takes machine code as input and generates RISC-V assembly instructions as output.

Coding Approach :

- **Input :** Reading the machine code from a file using file stream. Reading each machine code, which is in hex , I am converting that into binary 32 bits.

- **Instruction Format :** I am separating the instructions based on their format (i.e, R-type, I-type, B-type,.....) using the opcode (the last 7 bits of the instruction.)

- For each instruction format, I am writing a function which takes the 32 bit string as an argument. Now, dividing that string into substrings (for example : R-type : funct7, rs1, rs2, funct3, rd, opcode) and differentiating the instructions of that format. Now this instruction is printed into the output file using file stream.

- **Label Handling :** I used maps from stl to handle these. map<int , vector<int>>. Key of the map is an integer which has the line number where the label has to be printed. Whenever I encounter a B-type or J-type instruction I am storing this line number into the vector<int> of the map. So, all these elements of the vector<int> will point to the same line number given by key.

- **Some helper functions :** I have used some helper functions like
    1) hex_to _binary : this takes hexadecimal as argument and returns equivalent binary.
    2) Binary_to_int : this takes binary string as argument and returns equivalent decimal value.

Testing :

I have used some machine codes to test if my implementation is correct. I will be attaching the input files that I have used to test my code along with the assignment.

Error Handling :

In label handling, if the machine code tells it to print a label at some particular line number which exceeds the number of lines of the output or is less than zero , then an error message will be printed to the terminal.