## Programming Approach:

Since the input is only 32 bits of data, I first converted the hex values(as a string) into a (unsigned) decimal value (using long long) and then I convert it into bit format and add it into a vector of ints, where each element is 0/1 represent a bit.

Then from this I extract the opcode from the first 7 bits, in decimal form.

Then the rd, rs1,rs2, funct3, funct7 values are extracted from the binary code, by running the corr. computation functions.(doing this at the start is convenient since even if an instruction doesn't have one of these in its binary form, they can be combined to get another part of the instruction(like imm)).

Then a switch case is run on the opcode, which hits a case when the opcode finds a match, after which the funct3's are checked and funct7(if needed)

The corresponding disassembled instruction is stored in 'ans'.

Most of the cases are simple if-else if conditions

B type involves some more changes, where rd and funct7 are manipulated a bit to get the immediate value

Further, B types and J types require labels(and some more care), to do this, I did the following:

1) If the offset(current+offset/4, to be specific) is greater than input size(number of instructions), I chose to simply print out the offset, which seems to be valid in RIPES
2) Else, the line at the (offset+current) value is checked for whether it has a tag or not.
    1) If it does, then you simply add the label to the instruction
    2) If it doesn't, you create a new label, assign tag[x] to that label, and then add the label to the instruction

Also, the tags are used to for printing line labels, as in, if theres is a label associated with that string, it is printed, otherwise no label is printed

## Testing

For testing the codes, I tried all the instructions at least once(a few multiple times), in batches(their format types), as you can see in the attached txt files. Furthermore, I ran the sample case in the question, and also checked special cases(like negative values, extreme values)