Vanshika Gupta

CODE REPORT

1 Input-Output

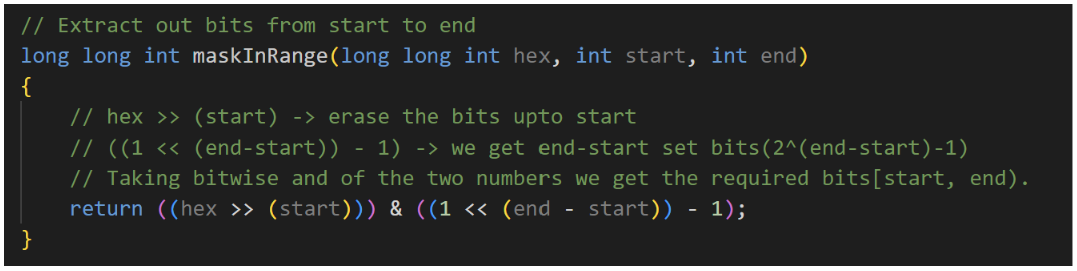
Input is taken from a file named Input.txt, one hex number at a time, directly using cin >> hex >> inp. Disassembled code is output into Output.txt line by line.

2 Structure of the Code

The code has a class for each of the assembly formats. Each class has a constructor. The constructor utilizes bit masking to calculate rs1, rs2, rd, etc. Each class also has a convert method, which contains all the if-else conditions based on the values of each field to convert into equivalent assembly code.

3 Flow of the Code

For every hex number, a function is called, which converts hex numbers into equivalent assembly code, which computes the opcode of the given instruction, and based on the opcode, an object of the relevant class is created, and their convert method is called. A very important function used is maskInRange.

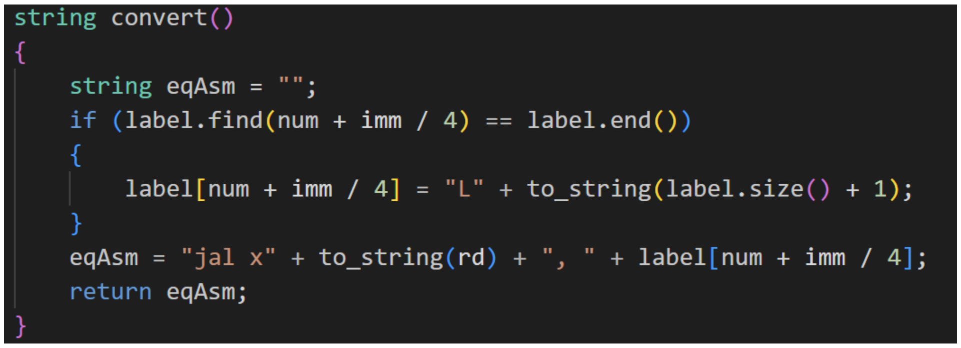


```
// Extract out bits from start to end
long long int maskInRange(long long int hex, int start, int end)
{
    // hex >> (start) -> erase the bits upto start
    // ((1 << (end-start)) - 1) -> we get end-start set bits(2^(end-start)-1)
    // Taking bitwise and of the two numbers we get the required bits[start, end).
    return ((hex >> (start))) & ((1 << (end - start)) - 1);
}
```

Figure 1: Bit Masking

4 Problem of Labels

To handle the problem of assigning labels, I created a global vector¡strings¿, variable num for line number and a map named label. The vector stores the equivalent assembly returned by the convert method. Each time a branch statement is used, the target line is checked for pre-existing labels. If it has a particular label, we use it. If not, we give it a new label. In the end, the equivalent assembly code is printed with the labels.



```
string convert()
{
    string eqAsm = "";
    if (label.find(num + imm / 4) == label.end())
    {
        label[num + imm / 4] = "L" + to_string(label.size() + 1);
    }
    eqAsm = "jal x" + to_string(rd) + ", " + label[num + imm / 4];
    return eqAsm;
}
```

Figure 2: Label Handling

5 Testing

For testing purposes, I took the code for Lab-2 and appended one by one all the instructions I was asked to implement. For immediate values, I focussed more on edge values and negative values. After making sure each of the statements worked with edge and negative values, I concluded the testing of the code.