Lab 3 : RISC-V Disassembler

Name: Yasir Usmani
Roll No. : AI22BTECH11031

- ***Please refer to the C++ program provided in the source code file.***

- **EXPLANATION**

1. **Functions:**

   a. **Hex to Binary Conversion:** The function named "**hexToBinary**" takes individual characters of the hexadecimal string as an input and returns a 4 digit binary string corresponding to the character. The function uses if-else conditions to return the binary string as each character in hexadecimal has a fixed binary corresponding to it.

   b. **Binary to decimal Conversion:** The function named "**binaryToDecimal**" takes a binary string as an input and converts it into the corresponding decimal value. The function uses a for loop to read each character and depending on its value(0 or 1) the loop operates, if the value is **1** the loop considers the power of 2 corresponding to the character else it ignores the particular power of 2.

   c. **Two's complement to Decimal:** The function "**twosComplimentToDecimal "** takes a string as an input and returns its decimal value.The logic I used is if the MSB is 1 then I am taking the power of 2 corresponding to the index to be negative and subsequently adding it to the other powers of 2 corresponding to the indexes where the value of character is 1.

   d. **Functions corresponding to Instruction Types:**

      1. **J_type :** The function takes a hexadecimal string as an input and then converts it into a Binary string. Then it filters out the **rd** register value and the **immediate** value from it. Also, I am declaring a string which contains the name of the function to jump to and then returning a string containing the immediate as well as the function name.

      2. **U_type :** The function takes a hexadecimal string input and after converting it to a binary string filters out the value of the **rd** register, **immediate** and then outputs the instruction corresponding to the opcode.

      3. **B_type :** The function takes a hexadecimal string input and after converting it to a binary string filters out the value of the **rs1** register, **rs2** register, **funct3** register, and the **immediate.** Again, a string is declared which contains the name of the function to jump to and then the name of the instruction is derived accordingly from the value of ***funct3.*** And finally after outputting the corresponding instruction, the value of immediate and the function to jump to is returned.

         4. **S_type :** The function takes a hexadecimal string input and after converting it to a binary string filters out the value of the **rs1** register, **rs2** register,

funct3 register, and the **immediate.** Then the name of the instruction is derived accordingly from the value of ***funct3.*** And finally the corresponding instruction is outputted.

5. **I_type :** The function takes a hexadecimal string input and after converting it to a binary string filters out the value of the **rs1** register, **rs2** register, **funct3** register, and the **immediate.** Then the name of the instruction is derived accordingly from the value of ***funct3*** and ***opcode.*** And finally the corresponding instruction is outputted.

6. **R_type :** The function takes a hexadecimal string input and after converting it to a binary string filters out the value of the **rs1** register, **rs2** register, **funct3** register, **rd** register, and **funct7.** Then the name of the instruction is derived accordingly from the value of ***funct3*** and ***fuct7.*** And finally the corresponding instruction is outputted.

e. **Main :** In the main function I am initiating a string type array named as **input** with its values as hexadecimal string. Then again, I am creating a string array named **opcode** with opcode as its values. Then I declared a few variables for jump functions in **B** and **J** type instructions (The offset of B and J is initialised to be -1 because when it becomes 0 i am outputting the value of the function to jump to). After that I ran a for loop to go through each of the elements of the input array and take its last two hex digits as it contains 8 bits to get the last 7 bits of the hexadecimal string. Then I again ran a for loop to check if the opcode matches with any of the instruction type opcodes. When a value is matched a function is called through the switch which then runs the function for a particular instruction type and prints the corresponding instruction for each hexadecimal string input.

   NOTE: For the cases of **B/J** instruction type, I am storing the value of immediate and jump function in a string named **temp.** Then I am filtering out the value of immediate and jump function from it to get the value of offset and the name of jump function from it respectively.

- **TESTING**

1. **Conversion Functions:** I checked each of the conversion functions by inputting my own values and then outputting the converted values. I received wrong answers sometimes because of running the loop for the wrong values or in the wrong direction which I corrected subsequently.

2. **Instruction type Functions:** I again checked each of the Instruction type functions by inputting my own values and then outputting the instructions. I also checked for individual values of **immediate, registers, funct3 and funct7.**

3. **Arrays and Loops in Main:** I inputted a few different sets of arrays in the main function and then ran the loop for each of them. I solved a few of the minor mistakes I made due to which the loop was running infinitely.