Thupten Dukpa
ES20BTECH11029
CS2323 Lab 8 Report

Lines 7-11: *to_binary()* is a function which converts a hexadecimal value to its binary equivalent. Every hex digit is mapped to its corresponding binary value in the *hex_to_bin* dictionary and is added to the resulting string.

Lines 15- 26: decimal_from_signextended() is a function which converts a binary value in its sign-extended form to its equivalent decimal value. If the MSB is 1, it finds its 2's complement to get the absolute value and is multiplied by -1. Else, it is converted directly to binary.

Lines 29-33: These lines are used to read the input, after stripping away the '\n' character and writing to the output text files to write the disassembled code on.

Line 35-51: hex_to bin dictionary maps every hex digit to its binary equivalent.
format_type maps the opcode of the input to the type of instruction including 'J' or 'U' type the instruction for which is explicitly stated.
R, I, L, S, B maps *funct3* or a combination of *funct3* and *funct7* to the corresponding instruction as per the RISC-V card.

Line 53: labels is a list keeping track of labels we come across in our code.

Lines 55-57: This iterates through each input instruction and finds the opcode.
It also checks for the error, i.e, in case the opcode does not match any of the values or if instruction has less than 8 hex digits.

Lines 65-69: Various fields like *funct3, funct7, rs1* and *rd* are initialized in this step.
funct7 and *rd* might be ignored depending on the type of instruction.
res is the final output we want to write to the output file.

Lines 71-75: While starting every new line, PC gets incremented by 4 so label value gets decremented by 4. Thus, by doing so, when any of the *labels* value becomes 0, we need to add that particular label in the beginning of that line.

Lines 77-147: Based on the format type, we can perform different operations to get the desired output.

Lines 77-81: Additional *rs2* field is required in R-format.

Lines 83-93: In case of slli, srli and srai, we need to consider both *funct3* and *funct7* and take only the first 6 bits of the immediate.

Lines 95-104: Load instructions are also I-type but their *funct3* range only from 0x0 to 0x6. If *funct3* <= 0x3, we consider immediate as msb-extended else zero extended.

Lines 106-111: Similarly, store instructions have their *funct3* range only from 0x0 to 0x6.

Lines 113-141: If the immediate in the branch and jump instructions are present in the *labels* list, it means that they will be pointing to the same label, so no need to include it again, else we add the immediate to the list(as the offset).

Lines 122-125: If immediate is negative we write its value, else we write it as a label.

Lines 143-145: In case of '*lui',* the immediate is taken as the hexadecimal value.

Lines 150-151: The input and output file handles are closed.

Methods used to check the code:

I've used the test instructions given in the assignment document to test my code and also tried various other instructions from each type of instruction format to verify. This custom input can be found in the 'sample_input' text file.

Furthermore, I've included the test case where two different branch/jump instructions might point to the same label. To solve this, I decremented values in *labels* by 4 in each line and checked if the offset of the current instruction is found in the *labels* list, if so there is no need to repeat it in the list.