Lab 8 Report

Tanmay Goyal
AI20BTECH11021

Coding Approach:

For coding, I have used Python, and I have used multilevel hashing, present in the form of a JSON dictionary. The approach is simple, it reads machine code, converts it into binary, parses it so as to retrieve the operation and type of operation from opcode, and accordingly retrieve the other fields, and then stores the disassembled code. The reason as to why the disassembled code is stored and not printed immediately is because some loops are formed after the PC has passed.

For example, in the following snippet, there is no indication to a 'factorial' loop before PC = $(c)_{16}$. However, we only know there is a factorial loop being made at PC = $(24)_{16}$. Thus, immediately printing the commands will not allow the Loop Label to be printed correctly.

```
    0:          100001b7          lui x3 0x10000
    4:          01018213          addi x4 x3 16
    8:          0001b503          ld x10 0 x3

000000000000000c <factorial>:
    c:          ff010113          addi x2 x2 -16
    10:          00113423          sd x1 8 x2
    14:          00a13023          sd x10 0 x2
    18:          00200293          addi x5 x0 2
    1c:          02554663          blt x10 x5 44 <one>
    20:          fff50513          addi x10 x10 -1
    24:          fe9ff0ef          jal x1 -24 <factorial>
```

Because all the disassembled commands will be printed simultaneously at the end, a dictionary for all the loops has also been maintained, which stores the PC, and the loop index for the corresponding PC. Note that the loops are not indexed in the right order, because the loops get stored in its dictionary based on the order in which they are encountered in the disassembled code. Another thing taken care of is avoiding repetition of the same loops under different indices by performing a check if the loop for a particular PC is already present in the dictionary.

Apart from this, another thing taken care of is the simple conversion from binary to decimal in python using the command 'int(val , 2)' converts into unsigned decimal. We know that the immediate values for I, B and J type instructions requires signed integers. Thus, care has been taken to define a function manually for the same.

The basic code working is summarized as follows:

1. Load the opt_dictionary, and read all the machine codes from the input.txt file.
2. For every command, convert into binary, extract the optcode, and find the type of operation.
3. If the type of operation is such that only one operation exists under that type, we do not need to access the opt_dict. Else, we find the operation using the optcode, funct3 and/or funct7 fields. We also accordingly parse the rest of the binary string into its fields.
4. For B or J type commands we add an extra check to find if the loop they might be referencing to already exists in

the loop_dict, or else a new loop is added to the dictionary. Accordingly, the command is built as well.
5. Finally the command, along with the incremented PC, and loop_dict are returned.
6. After all the commands have been disassembled, the commands are printed in order, making sure, that the respective PCs are checked, for any loop that might be present.

Testing:

For testing, the very first code that I used was the one present in the instructions for this lab assignment. This was done so that I could have the basic functionality ready. After this, I brought in the code from Lab1 for an additional check.

The code from Lab2 helped me work with the loop labels and made me realize the need for conversion from binary to signed integers and not unsigned integers. This, along with the code for implementation for the calculation of factorial from Lab3 brought in the problem with immediate printing of the commands, which was resolved by storing the commands till the end and maintaining a loop dictionary.

Finally, I made sure it is matching with the factorial code I had written, because of the presence of various registers, and multiple loops being called at different points in the code. Finally, codes from the previous lab where we were supposed to experiment with different cache settings were run to test the code.
