Lab 3 Assignment: Report

Pradeep Mundlik

October 12, 2023

- **Introduction:** This report provides documentation for the code related to loading data from a file and decoding RISC-V instructions. The code is organized into separate functions to achieve these tasks. Test data is stored in `data.txt` file in same directory.

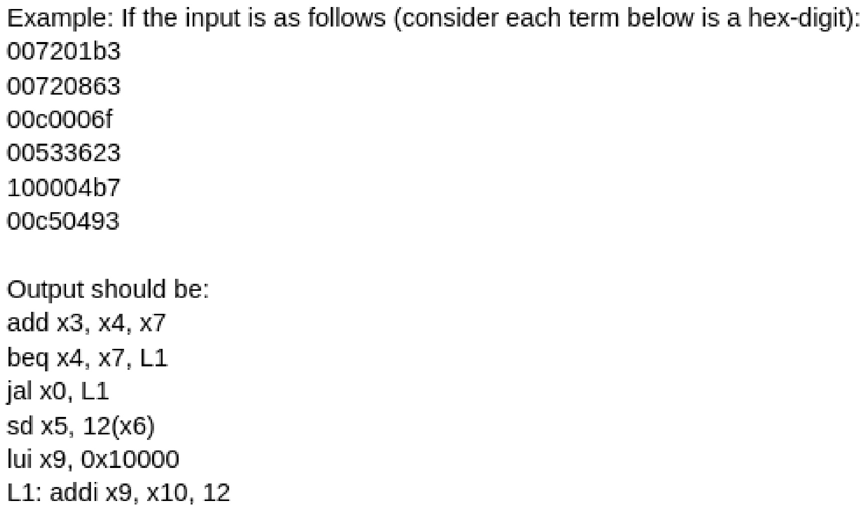
- **Loading Data from File:** A separate function named `loadData` is implemented to load machine code values from a file. This function reads the file line by line, converts the hexadecimal strings to `uint32_t`, and populates the vector.

- **Main Function:** In the main function, a vector `machineCodeList` is loaded from a text file named `data.txt`. Each line in the file contains one or more hexadecimal machine code values, which are converted into `uint32_t` and stored in the vector.

- **Instruction Decoding Functions:** The code includes separate functions to decode various RISC-V instruction types, such as R, I, S, B, U, and J types. Each function extracts relevant fields from the machine code, interprets the opcode and funct3/funct7 (for R and I type), and produces the corresponding assembly instruction. If the machine code is invalid or the opcode is not recognized, appropriate error messages are displayed.

- **Converting `uint32_t` to Hexadecimal Strings:** To convert a `uint32_t` to a hexadecimal string, the code employs a stringstream named `ss`. A stringstream is an object designed for string manipulation, providing the capability to handle data as if it were a stream. The line `ss << hex << machineCode` effectively converts the `uint32_t` value stored in `machineCode` to a hexadecimal string. This conversion is achieved by using the `<<` operator to direct the value into the stringstream `ss`. The inclusion of the `hex` manipulator ensures that the output is formatted in hexadecimal.

- **For Labels in B/J type:** All disassembled instructions are stored in vector `disassembledCode`. Every time we get B/J type, we will check its next instruction according to offset value, if next instruction is already



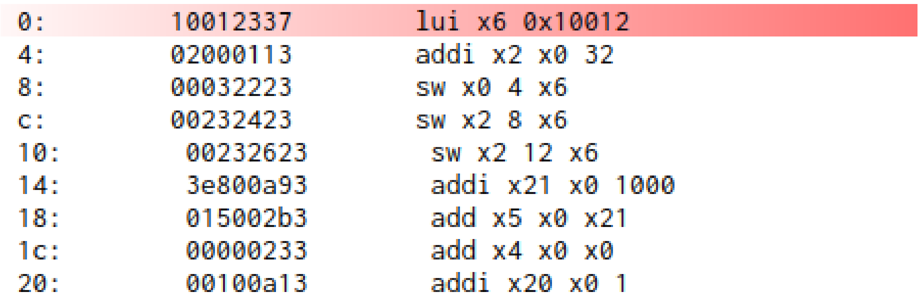
Example: If the input is as follows (consider each term below is a hex-digit):
007201b3
00720863
00c0006f
00533623
100004b7
00c50493

Output should be:
add x3, x4, x7
beq x4, x7, L1
jal x0, L1
sd x5, 12(x6)
lui x9, 0x10000
L1: addi x9, x10, 12

Figure 1: Sample text cases given with Assignment

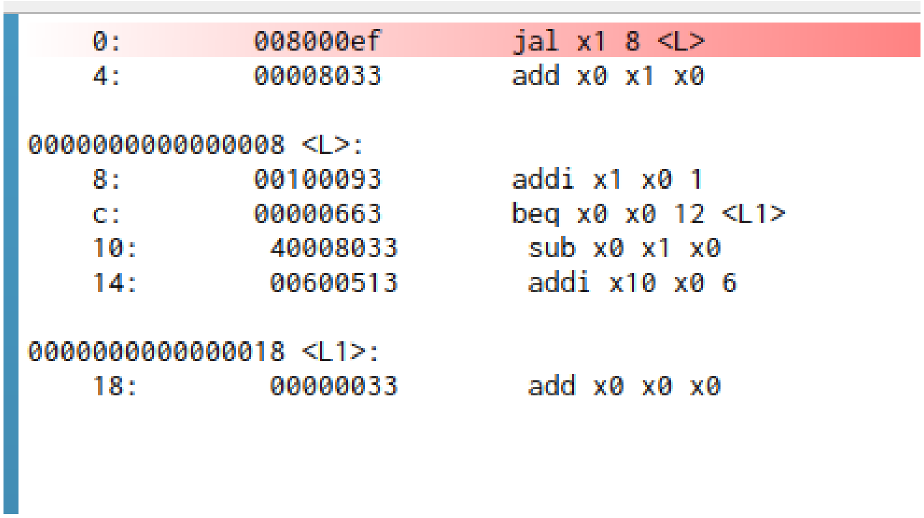
labeled then we will use that label else we will assign lable to that instruction. `num_labels` variable shows number of current labels in code and `i` variable shows number of current instruction.

- **Testing:** For testing, firstly I have used examples given along with problem statement. Then I have created some sample instructions and their Machine code is generated using Ripes Simulator.



```
0:        10012337        lui x6 0x10012
4:        02000113        addi x2 x0 32
8:        00032223        sw x0 4 x6
c:        00232423        sw x2 8 x6
10:        00232623        sw x2 12 x6
14:        3e800a93        addi x21 x0 1000
18:        015002b3        add x5 x0 x21
1c:        00000233        add x4 x0 x0
20:        00100a13        addi x20 x0 1
```

Figure 2: Sample text cases used (general)



```
    0:          008000ef          jal x1 8 <L>
    4:          00008033          add x0 x1 x0

0000000000000008 <L>:
    8:          00100093          addi x1 x0 1
    c:          00000663          beq x0 x0 12 <L1>
    10:          40008033          sub x0 x1 x0
    14:          00600513          addi x10 x0 6

0000000000000018 <L1>:
    18:          00000033          add x0 x0 x0
```

Figure 3: Sample text cases used(specific to B and J)