CS211 LAB0 REPORT

Yuhao Zhang

2021533141

School of Information Science and Technology

October 12, 2023

1 Analysis on the original code

This program begins from MainCPU.cpp. At the very first, this program parses the parameters to get the input file and determine the mode selected by the simulator. Then initialize L1, L2 and L3 cache and load ELF file. Finally, initialize the simulator and simulate.

In the Simulator.cpp, there is a five-stage CPU pipeline, each stage was handled by a function. Two adjacent stages are connected by two regs, i.e., the function fetch will store the new reg information into fRegNew, if there is not stall in the next stage decode after executing all the stage, fReg will be assigned as fRegNew, and so on.

For the stage fetch, simulator get the instruction from PC and put it into fRegNew. For the stage decode, if it is not in stall stage, we translate the machine code into real instruction and get the corresponding operate number, then store them into dRefNew. For the stage execute, we execute the instruction and write the result to rd in eRegNew. For the stage memory, we write or read data from a simulated memory implemented in MemoryManager.cpp if needed. For the stage writeback, after checking for data hazard and forward data, write back the data.

2 Description of my design

All of my changes has been marked out by "MY CODE HERE". All data in reg is saved as type int, and for float reg, I use *(float*)& to convert it to type float.

2.1 Simulator.h

In Simulator.h, I added the instname into enum list 'Inst', and add some new opcode in the opcode field. Then, add op3 in each reg struct for bonus part. Complete the latency list and the function 'getComponentUsed'.

2.2 Simulator.cpp

Add the instname into the list 'INSTNAME'. Add new instructions into decode part. Depending on the different opcode and funct3 or funct7, do different operation to them, and add op3 and rs3 for bonus part. Part execute is the same, calculate the result for each new instruction. Finally add the hazard checking part for rs3 in memory and writeback. Then we are done.

In detail:
1). Because all the data in the reg is saved as type int, so I just move op1 to out under the case fmv.w.x, fmv.s.x, fmv.x.w and fmv.x.s.
2). For fcvt, I get the address and change their type using a temp variable. For the basic computing, we convert the data into type float, and then we can compute them easily.
3). There are only one thing needed to take attention: do not forget to convert it back to type int.
4). As for fmadd and fmsub, they are nearly no different with fadd.s, fsub.s and fmul.s, except that they have an op3.

3 Evaluation and analysis

3.1 Test on the given test case

Most of given test cases can be executed successfully once we compile the simulator, what we are talking about is the case test_float.riscv. After running the elf file, the output should be:

5.794590

0.488590

370.349976

41.149998

If the code was compiled correctly, the output of the simulator should look like the above.

3.2 Test on my own test case

I create test cases for each new instruction, you can view them in the test folder, also you can test them all using the test_all case. The command to test these cases are wrote in README.md. And the output is a little bit long so I won't show it here, you can also check it in README.md.

It seems that all the test cases are running correctly, so I think these intructions were successfully added.