Project Report

Team members:

1. Khaled Ayman Anwar, ID: 49-3005, Tut: T22
2. Mohamed Almarsafy, ID: 49-9147, Tut: T22
3. Hassan Sherif Haridy, ID: 49-5516, Tut: T22
4. Yousef Mohamed Hassan, ID: 49-0560, T17
5. Andrew Sameh, ID: 49-0908, T14

Steps:

1. We first the following inputs from the user and validate them:
    a. Latencies for all instructions (MUL.D, ADD.D, SUB.D, L.D, S.D, DIV.D)
    b. Instructions (ex: ADD.D F1,F2,F3) we add ; to the end of the instruction if the file is submitted from the UI.
    c. The number of reservation stations (MUL reservation stations, ADD reservation stations, LD reservation stations).
2. Our memory is initialized as an array which is of size 100 memory item.
3. We then decode the entered instructions and extract all registers (source registers, destination register, opcode, immediate).
4. We use validators for Instruction format to make sure our instructions are inserted correctly and to be able to extract the fields.
5. Our register file is an array containing the 32 floating point registers and is initialized with zero values every time we run the project.
6. We have ALU reservation stations which we initialize as an array of objects called resStation. Every resStation object contains the following attributes:
    a. id (tag) of the reservation station.
    b. busy (either 0 or 1).
    c. Op which is the opcode of the decoded instructions.
    d. Vj, value of first operand if available in memory.

    e. Vk, value of second operand if available in memory.
    f. Qj, value of first operand if it is not available in memory (ex: A1)
    g. Qk,value of second operand if it is not available in memory (ex: A1, M1)
    h. Time, which is the remaining time.
    i. Start_time, which is the time where the instruction was fetched in.

7. We have also, storeStations and loadStations. storeStations got (address, V and Q) and loadStations got (address).
8. In all stations we keep track of different timing variables.
9. We have a cycle variable that acts as our clock cycle and gets updated every iteration.
10. The bus is called “busId” which is a global value that takes the output from any instruction that finishes execution and then updates the register file and updates all reservation stations in the next cycle.
11. Every instruction gets:
    a. Issued/fetched
    b. Executed
    c. Write back on bus

10. When 2 or more instructions finish at the same time, one instruction is selected randomly to write back on the bus and so on.

11. We also handled the case when 2 or more stores are writing to the same memory address, and if that happens, we wait for the first store to finish execution and then fetch and start to execute the next store and so on.

Code structure

We used python3 to write this project.

We have the following classes

1. Instruction.py
2. loadStation.py
3. storeStation.py
4. loadStation.py
5. resStation.py
6. RegisterItem.py
7. Server.py
8. Main.py

- Instruction is the class where we store the decoded instruction.
- All of the stations are the reservation stations for different instructions. ResStation is for MUL and ADD stations.
- RegisterItem.py is the class of each register item. contains(Q and the value).
- Main.py is to run it in the cLI.
- Server.py is to run it on a server to communicate with the front-end

Approach

1. Each instruction is decoded and added to an array of decoded instructions.
2. Then, we have a function called runcycle() which calls fetch, execute and writeback in order.
3. In each function call, it does the functionality of the stage.
4. In fetch, we check first if there are empty stations to fetch the instruction into, otherwise we stall the whole remaining instructions. Also, we check for any address clashes, if so we stall as well.
5. We start executing the instruction whenever all of its operands are ready.

Running and Testing

1- First Test: In this example each register had the corresponding index value ,for example: F1 -> 1, F2 -> 2, etc..

1. Instructions:
    a. MUL.D F3, F1, F2
    b. ADD.D F5, F3, F4
    c. ADD.D F7, F2, F6
    d. ADD.D F10, F8, F9
    e. MUL.D F11, F7, F10
    f. ADD.D F5, F5, F11
2. Latencies:

    MUL.D: 6

    ADD.D: 4

    L.D: 1

    S.D: 1

    DIV.D: 1

    SUB.D: 1

2- Second Test:

1. Instructions:
    a. ADD.D F3,F3,F2
    b. MUL.D F5,F5,F2
    c. SUB.D F5,F5,F3
    d. S.D F5, 10
    e. S.D F3, 10
    f. L.D F7, 10
2. Latencies

    MUL.D -> 5

    ADD.D -> 4

    SUB.D -> 4

    DIV.D -> 1

    S.D -> 2

    L.D -> 2

3- Third Test:

1. Instructions
    a. L.D F1, 0
    b. L.D F2, 1
    c. MUL.D F2, F2, F2
    d. MUL.D F1, F2, F1
    e. MUL.D F1, F1, F1
    f. MUL.D F3, F1, F1
    g. ADD.D F3, F2, F2
2. Latencies

MUL.D -> 2

ADD.D -> 4

SUB.D -> 4

S.D ->2

L.D ->3

DIV.D -> 1

4- Fourth Test:

1.Instructions

a. MUL.D F3, F1, F2
b. L.D F3, 15
c. S.D F3, 12
d. DIV.D F3,F3,F3
e. S.D F3, 12

3. Latencies

MUL.D -> 3

ADD.D -> 4

SUB.D -> 4

S.D ->2

L.D ->2

DIV.D -> 4