Microprocessors Project Report

Team 30 (SHYNK):

1. Name: Nada Abdel Rahman Hegazy
   ID: 49-1384
   Email: nada.mohamedhegazy@student.guc.edu.eg
2. Name: Farida Tarek Kassem
   ID: 49-1122
   Email: farida.kassem@student.guc.edu.eg
3. Name: Yasmine Khaled El Serafy
   ID: 49-1224
   Email: yasmine.elserafy@student.guc.edu.eg
4. Name: Salma Ihab
   ID:49-0480
   Email: salma.ihab@student.guc.edu.eg
5. Name: Hana Mohamed Saeed
   ID: 49-2127
   Email: hana.abdelhakim@student.guc.edu.eg

Our Approach:

In the main method, the user inputs the latencies of every operation, and we call the read file method passing all the latencies provided. The flow of the whole logic is controlled in the readfile method.

First, we read all the instructions from the program.txt file and fill them in our instruction memory. Second, we start preparing our Execution table by filling all the instructions/destination/Sources in sequential order. Last but not least, we start our while loop to re check all the full reservation flags, then we start by calling the Write Back method calling followed by the Execute then lastly, the Issue method and clock/pc incrementation. We used this reversed process to allow the Execute method to automatically take the written back values in the correct cycle. The loop keeps going until the finished flag is declared true when all the instructions have written back and the reservation stations are empty

Assumptions:

- No cache misses
- No cache latency
- 3 slots in the load buffer
- 3 slots in the store buffer
- 2 slots in the multiplication reservation station
- 3 slots in the addition reservation station

Code Structure:

Components used in the code are:

1- Instruction Memory→ String array of size 2^10 (1024)

```
1 L.D F6,50
2 L.D F2,100
3 MUL.D F0,F2,F4
4 SUB.D F8,F6,F2
5 S.D F8,80
6 DIV.D F10,F0,F6
7 ADD.D F6,F8,F2
```

2- Data Memory→ Long array of size 2^20 (2048)

3- Register file (32 registers)→ String array of size 32

```
Registers content:
F0 : 400
F1 : 1
F2 : 100
F3 : 3
F4 : 4
F5 : 5
F6 : 50
F7 : 7
F8 : -50
F9 : 9
F10 : 8
```

4- Reservation stations

- There is a flag for every slot reservation station to control Write Back function)
- There is one flag for every reservation station to indicate if it is full

a) Add/Sub → 2D array of 4 rows and 8 columns

Name of Station	Busy	Operation	Vj	Vk	Qj	Qk	Time

b) Mul/Div → 2D array of 3 rows and 8 columns

Name of Station	Busy	Operation	Vj	Vk	Qj	Qk	Time

c) Load → 2D array of 4 rows and 4 columns

Name of Station	Busy	Address	Time

d) Store → 2D array of 4 rows and 6 columns

Name of Station	Busy	Address	V	Q	Time

5- Execution Table → 2D array of instruction memory length +1 rows and 9 columns

Instruction	Destination	Source1	Source2	Station	Issue	Execution Start	Execution End	Write back

6- Finished flag → Indicates whether all instructions have written back and reservation stations are empty or not
7- Program Counter (pc) → Incremented after every instruction is issued to point to the next instruction
8- Clock

- **Issue Stage:** In the Issue method, we pass an instruction in the method, then depending on the operation, we choose which reservation station to put the instruction in through looping over all reservation stations. After choosing the suitable station, we start by incrementing the busy flag to one to show that this slot is occupied, then we start to fill the operands and operation latency in the time slot. The operand can be ready in the register file/memory or may be dependent on the write back of another reservation station. As we finished updating the stations, we start updating the register file by selecting the destination register and assigning it the name of the relevant reservation station. Lastly, we update the execution table with the station and issue timing. If the instruction found a slot in a reservation station and all the data is filled correctly, then we return true as an indicator to

increment the program counter and move to issue the next instruction.

- **Execute Stage:** In our code the execute stage is a method that's called every cycle and it is called after write back and before issuing, all the latencies are passed to it. The execute method loops on all reservation stations, firstly the load buffer, since the load instruction is not concerned with anything but the memory in order to start executing, we loop on the buffer and if there is a station that's busy and has its timer equal to the load latency meaning it is ready to start executing then we loop on the execution table to find the instruction and update its executing start cycle with the cycle we are in right now, we then decrement its timer in the buffer by 1 and check if it reached 0 then this instruction finished executing so we update the execution table: execution end cycle column with the cycle we're in, otherwise, if the timer is -1 or less and it wrote back then we remove it from the buffer. For all other operations the same steps are followed however, there is an additional condition that all their operands are ready and the instruction isn't waiting on a value from another instruction so we initially check that then start checking our other conditions. We also have variables for each reservation station to carry the result executed by the instruction in the station. They are our version of the common data bus and we use those variables in the write back method

- **Write Back Stage:** the write back stage in our code is a method that gets called first thing before the execute and issue methods. Firstly, we start looping on our execution table and check for the first instruction we meet that has its execution end column with a value less than the clock cycle we are in and its write result column is empty, we then check its station column and if it's a load instruction we get the value from the data memory, if it's a store we do nothing and if it's an ALU operation we get the value from the variable of the station

(CDB). After that, we update the destination register in the register file with our value then we loop on store buffers, multiplication station, addition station to see if there is any instruction waiting on this value, if found, we update its respective V and Q columns and mark this station with a flag so that it doesn't execute later in the same cycle with the value it just got. Lastly, we update the write result column instruction that wrote back with the cycle we are in then we break from the loop to avoid 2 instructions writing at the same cycle.

Test Cases:

- Testing using the example explained in the tomasulo lecture to make sure our code is working cycle by cycle as expected
- Testing the store instruction with the previous code as it wasn't included
- Changing the latencies so that 2 instruction finish executing at the same time to test that they don't write in the same cycle and that they use the FIFO scheme
- Testing that if a station is full and an instruction writes back, that the instruction waiting doesn't get issued in the same cycle