Approach:

Our approach that we will simulate tomasulo using our execution table.The execution table will contain all the necessary details to make sure our execution runs correctly.

Each entry in this table contains 6 fields :

1-Iteration:

Which tells us this instruction belongs to which iteration

2-Id:

Which tells us in which reservation station is this instruction in (if it has not yet finished execution because if it has finished and has written its result it will be removed from the reservation station)

3-instruction:

Tells us what is the instruction

4-issue:

Clock cycle at which this instruction was issued

5-execute:

Clock cycle(s) in which this instruction is executing

6-write result:

Clock where this instruction wrote back its result on the bus

So basically at each clock cycle we can do the following:

-Issue a new instruction

Instructions can be issued as long as there is a space in the reservation station and if it is a load or store we must check on the addresses to avoid hazards.

Once an instruction is issued it will be added to the execution table.

-Execute instructions that are in the reservation stations

We will execute the instructions as long as some certain conditions are satisfied such as operands are ready otherwise we will wait

-Write the result

We can have two or more instructions that want to write their result on the bus.

We handle the writing of the result in a queue data structure if any instruction wants to write it on the bus it will be put on the queue.

The queue will poll one element from it to be published on the bus and also write the write result clock cycle in the execution table.

Code Structure:

We divided our code into 5 packages:

1-Package Data

This simulates the storing of data in registers/cache

This package contains two classes one for the cache and the other for the register file

2-Package Load_Instructions

This package is responsible for parsing the instructions from the text file.

It has 2 classes one called Main_Load that parses the text file and the other class called Instruction is just the normal instruction object with all of its properties.

3-Package Main

This package is where the main work happens.

This package has 5 classes.

Class checks is just class that contains methods that checks conditions we need.

Class Pair is just a class that helped us to return a pair from a function to ease our implementation.

Class ExecutionTABLE is a class that defines what our execution table will be composed of.

Class Execution is the class that contains our execution table and also our queue of data that will be published.This class handles the execution of the instructions and publishing and so on.

Class Main is the class that contains our main method that will initialize the GUI and then based on the user input the the execution will start and we will keep looping calling various methods from other classes we mentioned and end upon the termination of all instructions.

4-Package Print

This package is responsible for displaying the reservation stations ,execution table , and data tables .

Displaying is handled for both in the console and also a GUI.

5-Package Reservation_Station

This package is responsible for simulating our reservation stations.

It has 4 classes :Add_RS ,Mul_RS ,Load_Rs ,Store_Rs

Test Cases:

```
MUL.D F1 F2 F3
DIV.D F1 F2 F3
SUBI R1 R1 8
ADD.D F1 F2 F3
L.D F1 2
S.D F1 2
```

This test cases above checks that every instruction works correctly like issuing in first clock cycle and execute based on latency and finally write result

```
LD R1 6
LOOP SUBI R1 R1 8
BNEZ R1 LOOP
```

This test case checks looping runs correctly and branch not executing until SUBI write result in the bus and check its value with R1 either stop looping if it's equal to zero or continue otherwise note we don't use prediction means we stall until branch finish executing

```
L.D F1 6
L.D F2 6
```

This test case test that two loads works in parallel as they don't depend on each other

```
LD R10 6
SD R11 6
L.D F1 6
S.D F2 6
```

This test case test how to handle clashes between load and stores from accessing same address, means every load shouldn't issue if there is any store still executing and every store shouldn't issue if there is any load or store still executing

```
ADD.D F1 F2 F3
MUL.D F1 F2 F3
```

This test case test the final result of F1 will be the value of instruction comes from MUL.D based on their dependency

```
LD R10 6
LOOP2 L.D F1 1
L.D F5 2
DADD R1 R10 R10
MUL.D F5 F5 F1
ADD.D F3 F2 F2
S.D F3 2
SUBI R10 R10 8
BNEZ R10 LOOP2
```

This test case test the load and the branching and also test dependencies between the BNEZ and SUBI also dependencies between ADD.D and S.D and also dependency between MUL.D and L.D

It mainly tests dependencies along with the testing of branching.

```
LD R10 6
LOOP DIV.D F10 F1 F5    // DIV Latency = 6
MUL.D F9 F1 F5          // MUL Latency = 5
ADD.D F8 F1 F5          // ADD Latency = 4
SUB.D F7 F1 F5          // SUB Latency = 3
SUBI R10 R10 8
BNEZ R10 LOOP
```

This test checks if some instruction execute in the same clock cycle which one will publish result on the bus, based on our implementation we use FIFO approach note for latencies to make them finish execute in the same clock cycle