Tomasulo Simulator

CS203 FINAL PROJECT

Nikhil Kamthe
861245635

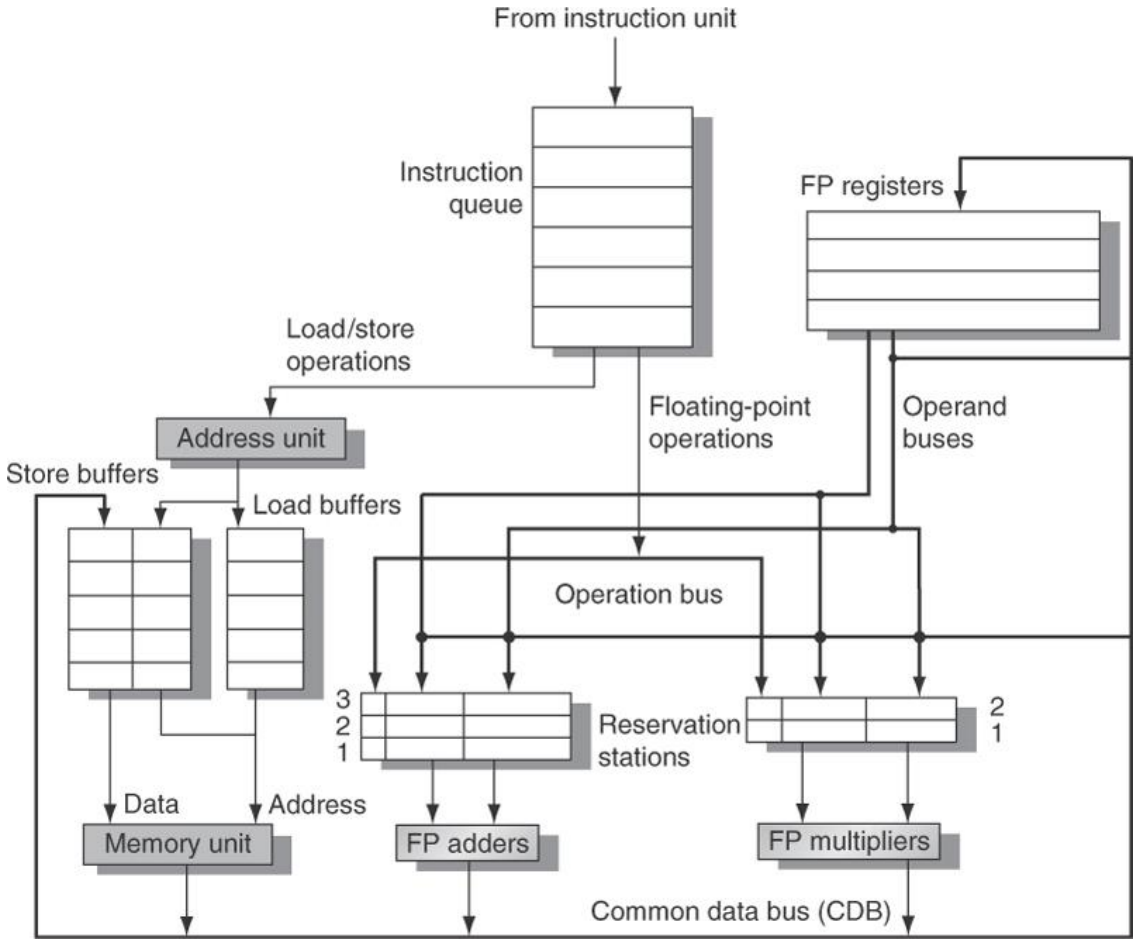
Introduction:

Tomasulo's algorithm was developed by Robert Tomasulo at IBM in 1967, and first implemented in the IBM System/360 Model 91's floating point unit. It is a computer architecture hardware algorithm for dynamic scheduling of instructions that allows out-of-order execution. Dynamic scheduling rearranges instruction execution to reduce stalls while maintaining data flow and exception behavior. Tomasulo was designed to efficiently utilize multiple execution units.

The major innovations of Tomasulo's algorithm include register renaming in hardware, reservation stations for all execution units, and a common data bus (CDB) on which computed values broadcast to all reservation stations that may need them. These developments allow for improved parallel execution of instructions that would otherwise stall under the use of score-boarding or other earlier algorithms [1].

In this project the non-speculative version of the algorithm has been implemented.

Design:



1. Instruction Queue:

Instruction queue is used to fetch the instructions from the instruction unit. The instructions are fetched in order and forwarded to reservation stations.

2. Load/Store Buffers:

This is where the memory instructions are stored when they are fetched from the instruction queue. It makes sure that the memory instructions are executed in order, to avoid data hazards.

3. Reservation Stations:

This is where all the arithmetic and branch instructions are stored before their execution. The reservation station stores details like opcode, values of the source registers, pointers to instructions on the source registers are dependent and the status of the instruction etc. This information is used throughout the execution period of an instruction. An instruction is removed from reservation station only when it completes its execution i.e. after common data bus stage for arithmetic and load, after write back stage for store and after execution stage for branch.

4. Common Data Bus:

Common Data Bus is used to publish the results once the instruction completes its execution. The value published by the common data bus is used by the reservation stations and register result station to update the corresponding entries. This enables instructions waiting on this instruction to move to the execution stage.

5. Register Result Station:

This component is used to keep track of the registers which are to be modified by an instruction. Every time an instruction is inserted into the reservation station, the pointer to the instruction is stored against the destination register in Register Result Station. This information is used to update the reservation tables.

Stages:

1. Issue:

This is the stage where the instruction is fetched from the instruction queue and added to the reservation table. Depending on the opcode of instruction the corresponding reservation station is checked for available space. The instruction is fetched only when there is space in the reservation station. As the instruction is fetched, we check the destination register and update the corresponding entry in the Register Result Status array to store the identifier of the current instruction. Then the values in the source registers are read if they are ready to be read. In case value of the source register is to be produced by some instruction in the reservation station, the pointer to the corresponding instruction is stored in the reservation station. This is how register renaming is implemented in Tomasulo's Algorithm.

2. Execution:

This is the stage the instructions are fetched from the reservation station and executed. An instruction is picked for execution only when values of all its source registers are available. It is the stage where different types of instructions can be executed parallelly by corresponding executors. Whenever an executor is idle it checks the corresponding reservation station for instruction which is ready for execution. In case more than one instruction is ready for execution, an instruction is fetched in the order in which it was fetched from instruction queue. Note that we could pick any instruction at this point except for memory instructions. In case of memory instructions, we need to execute the instructions in proper order to avoid data hazards.

3. Memory Stage:

This stage is applicable only for memory instructions, where they read or write to some address in memory. The memory stage fetches those instructions from memory execution unit which have completed their execution.

4. Common Data Bus:

This is the stage where we publish the computed results on the common data bus. The published results are used by the Register Result Station array to free the corresponding destination register. At the same time, reservation stations read value from the common data bus to update source values for appropriate instructions. This is the stage where an instruction completes its execution and can be removed from the reservation station.

Implementation:

In this project, the non-speculative version of the Tomasulo's Algorithm has been implemented. The implementation has been done entirely in JAVA. Important components (classes) of the code have been explained in detail below:

1. InstructionUnit.java:

This class simulates an instruction unit. It stores the instructions read from the input file. Along with storing the instructions it also stores loop identifiers in a map along with their start and end indices. The instructions are fetched using a PC counter which points to the current instruction. Every time an instruction is fetched this counter is incremented to point to the next instruction. In case of branch statements, we use the loop map to find the value of pc.

2. ReservationStation.java:

This class simulates reservation station. It contains an array of type ReservationStationEntry.java which stores information corresponding to the fetched instruction.

3. ReservationStationEntry.java:

This class is used to store information corresponding to every entry in the reservation table. It stores the id of the instruction, pointers to the instructions that produce the values of the source registers for current instruction and a flag to denote if the instruction is busy.

4. StageProcessor.java:

This class is used to simulate stage processors and executors. It stores id of the instruction being processed, time taken for execution, time elapsed since the execution started, flag to denote if the processor is occupied and a flag to denote if the processor finished execution of current command.

5. Instruction.java:

This class is used to store information corresponding to an instruction.

Note: I have only listed the classes which important in the context of Tomasulo Algorithm.

Instructions to run and input file format:

The file containing the instructions should to be in a specific format. Following are the points to consider while writing an instruction file:

- Register names should start with small case r followed by a number e.g. r2
- Use following opcodes for instruction:
    - LW for load
    - SW for store
    - ADD for addition
    - SUB for subtraction
    - MULT for multiplication
    - DIV for division
    - BNEZ for branch
- To represent a loop, the identifier for the loop with a colon should be included above the first instruction in the loop and the keyword END should be included after the last instruction in the loop. Sample input can be as follows:

```
LOOP:
ADD r1 r2 r3
SUB r2 r3 r4
MULT r3 r1 r5
DIV r4 r3 r6
LW r5 r4
SW r5 r7
BNEZ r7 r8 LOOP
END
ADD r1 r2 r3
```

- I have provided 3 sample instruction files along with the code.

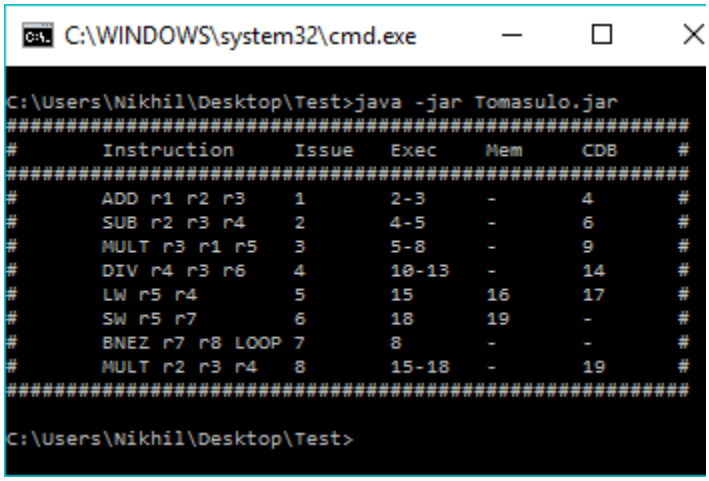
I have provided the executable jar for the Tomasulo Simulator which can be executed with the following command:

Java -jar <Path to Tomasulo.jar>

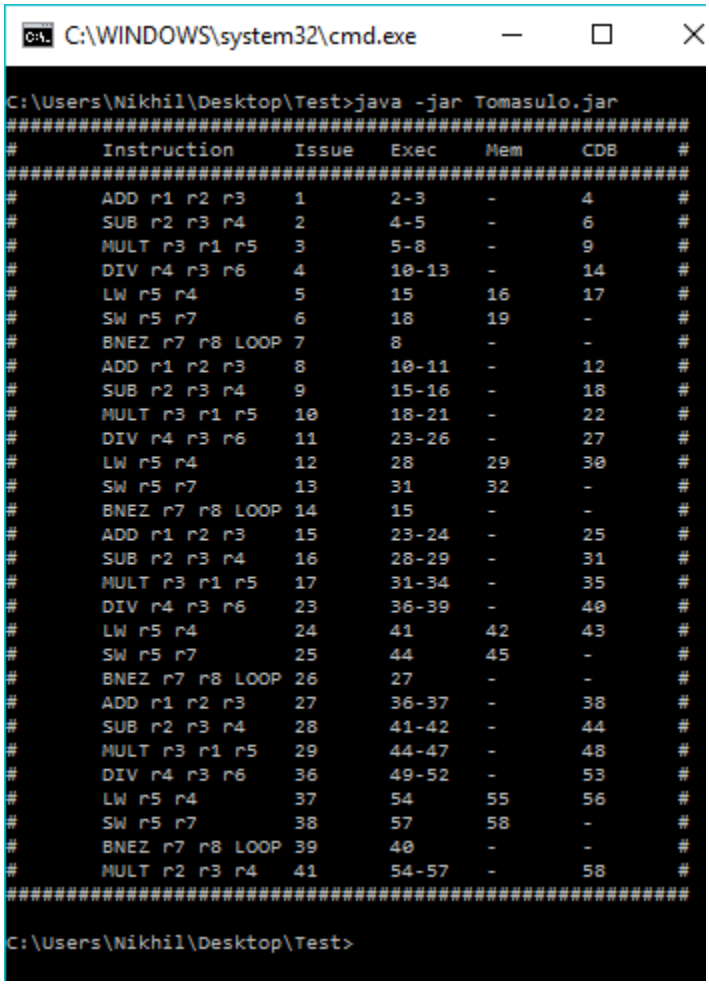
The program will load the properties from the config file provided along with the Tomasulo.jar. Properties like time taken by execution units, size of reservation tables, branch prediction output and the number of times branch is taken (for branch always T case) can be configured using this file. Make sure the config file is placed in the same folder as the Tomasulo.jar. Appropriate comments have been added above every property in the config file.

Screenshots:

Branch Not Taken:



Branch Taken:



References:

[1] https://en.wikipedia.org/wiki/Tomasulo_algorithm

[2] Course material