National University of Computer and Emerging

Sciences



Tomasulo Algorithm Simulator

By: Saad Farooq (17l-6105)

**Table of Contents**

Class Diagram…………………………………………………………………….……2

Pseudocode……………………………………………………………………….…….2

Features……………………………………………………………………………..…..5

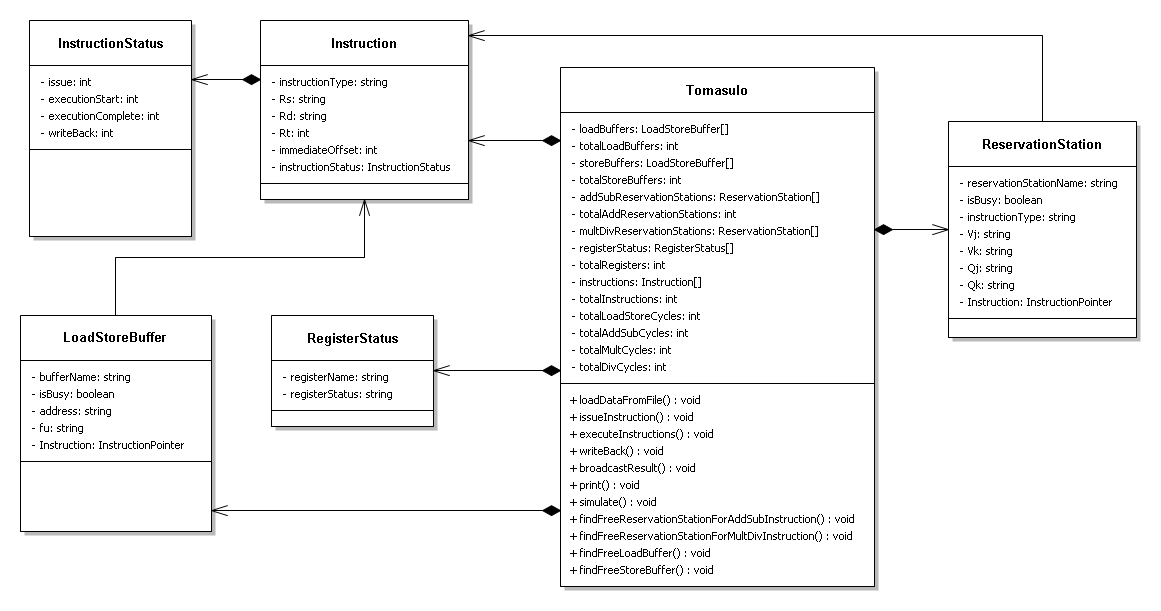
What can be further added……………………………………………………….….….5

Limitations………………………………………………………………………,….….5

Tools Used………………………………………………………………………...........6

Screenshot of Simulation…………………………………………………………….…6

1. **Class Diagram:**



1. **Pseudocode:**

**2.1 Simulate() Function:**

short currentCycleNumber = 0;

while (there is an instruction remaining)

{

printInformationOnScreen();

waitForUserToPressAKey();

currentCycleNumber++;

issueInstruction();

executeInstructions();

writeBackInstructions();

}

**2.2 The Required Functions Used by Simulate() Function:**

issueInstruction()

{

/\*only one instruction can be issued at a time.

The instruction issue will be in-order\*/

if (Required ReservationStations / Buffers are Busy)

{

return; //do not issue the current instruction because a structural hazard exists

}

else

{

1. Read Values of source registers Rs and Rt into Vj and Vk fields respectively.
2. If Any source registers(s) is not available. Write the name of the reservation Station / Buffer, which will produce the result for that register, into Qj/Qk. //RAW hazard is handled.
3. In Register Status table record the information that this reservation station / buffer will write the result to destination register Rd.
4. Mark the reservation station / buffer as busy.
5. Set IssuedInstruction.issued = currentCycleNumber.
6. currentInstructionToBeIssued++; (This integer variable is instruction counter)

}

}//end of IssueInstruction() function

executeInstructions()

{

//Take care to not execute the instruction issued in the current Cycle

for-each issued Instruction

{

1. if all source register values are available, then execute instruction.(remainingExecutionCycles--); else continue the next iteration of loop.
2. If this is the first cycle of execution of the instruction, then set Instruction.ExecutionStartedOn = currentCycleNumber.
3. If this is the last execution cycle of the instruction, then set Instructin.ExecutionFinishedOn = currentCycleNumber.

}//end of loop

}//end of executeInstructions() function

writeBackInstruction()

{

//Take care to not write back the instruction which completed the execution in the current cycle

for-each issued Instruction

{

1. if the instruction has completed its execution cycles (Instruction.remainingExecutionCycles = 0).then goto line 2, else continue the next iteration of loop.
2. Set Instruction.writeBack = currentCycleNumber.
3. Broadcast the results to dependent reservation stations / Buffers
4. Mark the reservation station / buffer as free.

}//end of loop

}//end of writeBackInstructions() function

1. **Features:**
2. Although the current instruction set is limited, yet the instructions can occur in any order, and there is no limitation on the number of instructions that can be executed by the simulator.
3. You can increase/decrease the number of buffers, as well as reservation stations.
4. You can specify the number of cycles that each type of instruction takes in its execution stage.
5. The number of registers can be increased or decreased.
6. **What can be further added:**
7. The program can be modified to simulate score boarding algorithm.
8. Re-order buffer can be added.
9. Instruction set can be enhanced to accumulate more instructions
10. Branch prediction may be added (may require a lot of changes)
11. **Limitations:**
12. The output may not appear good below 1366\*768 resolution; output definitely does not appear intact on resolution below 1024\*768.
13. The program can only run on windows platform due to its dependence on Windows.h library.
14. The program does not work efficiently. (For example, while executing and writing back instructions, the program is visiting each and every reservation station/buffer). With some modifications, such inefficiency can be removed.
15. Current Instruction set is limited to:

ADDD, STORE, LOAD, SUB, and MULT instructions.

1. **Tools Used:**
2. For class diagram, NClass was used.
3. The simulation program was written in C++.
4. The IDE used for programming was Visual Studio 2017.
5. Microsoft Word was used for developing this documentation

**8.\_\_Screenshot of Simulation**

