Project 2 Report

Tomasulo Algorithm Simulator

Fall 2026

Doha Nour El- Din

Habeba Saad

1. Implementation Description

This project implements a cycle-accurate simulator for a simplified 16-bit RISC processor using Tomasulo's algorithm with speculation. The full simulator is implemented in C++17 in a single file (main.cpp) for convenience and clarity.

The simulator models:

1.1 Core Components Implemented

- Register File (8 registers, R0 always 0)
- Register Status Table for tracking pending writes
- Reservation Stations (RS)
  - Load RS (2 entries)
  - Store RS (1 entry)
  - Branch RS (2 entries)
  - ADD/SUB RS (4 entries)
  - NAND RS (2 entries)
  - MUL RS (1 entry)
  - CALL/RET RS (1 entry)
- Reorder Buffer (ROB)
  - 8 entries
  - Holds instruction metadata, destination, values, and commit status
- CDB (Common Data Bus)
  - Only one write-back per cycle

1.2 Instruction Set Implemented

All instructions required by the project specification were implemented, which are LOAD, STORE, BEQ, CALL, RET, ADD, SUB, NAND, and MUL.

1.3 Pipeline Stages

The simulator implements the backend pipeline stages:

1. Issue (1 cycle)
   - In-order
   - Requires free ROB entry and free RS entry
2. Execute (multi-cycle)
   - Begins once operands are available
   - Latency depends on instruction type
3. Write-back (1 per cycle)
   - Writes result to ROB
   - Frees RS entry
4. Commit (in order)
   - Updates register, memory, or control flow

    - One cycle for all instructions except STORE
    - Handles speculation recovery.

1.4 Speculation and Branch Handling

- Always-not-taken predictor
- On misprediction:
    - Flush all RS entries
    - Flush ROB
    - Clear register status table
    - Reset PC
    - Rebuild fetch queue

1.5 Memory Model

- 16-bit word-addressable memory
- 32000 memory entries
- Loaded from external file (memory.txt)

1.6 Bonus Features

No bonus features were implemented to focus on the correctness of the core Tomasulo algorithm.

2. User Guide

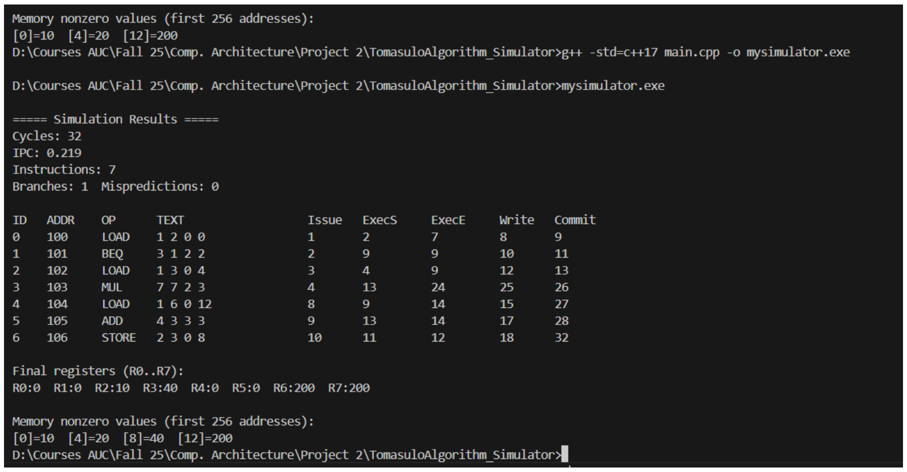
Here is a full guidance for how to compile and run the Tomasulo simulator, as well as how to prepare input files.
To generate and run the code, you need to:

1. Download the project from GitHub link here.
2. Open the project on VS code (note that it's a c++ code).
3. Open the terminal → command prompt.
4. Run: g++ -std=c++17 main.cpp -o mysimulator.exe
5. Run: mysimulator.exe

3. Test cases

For the test cases, there are different tests that were used in running the simulator.

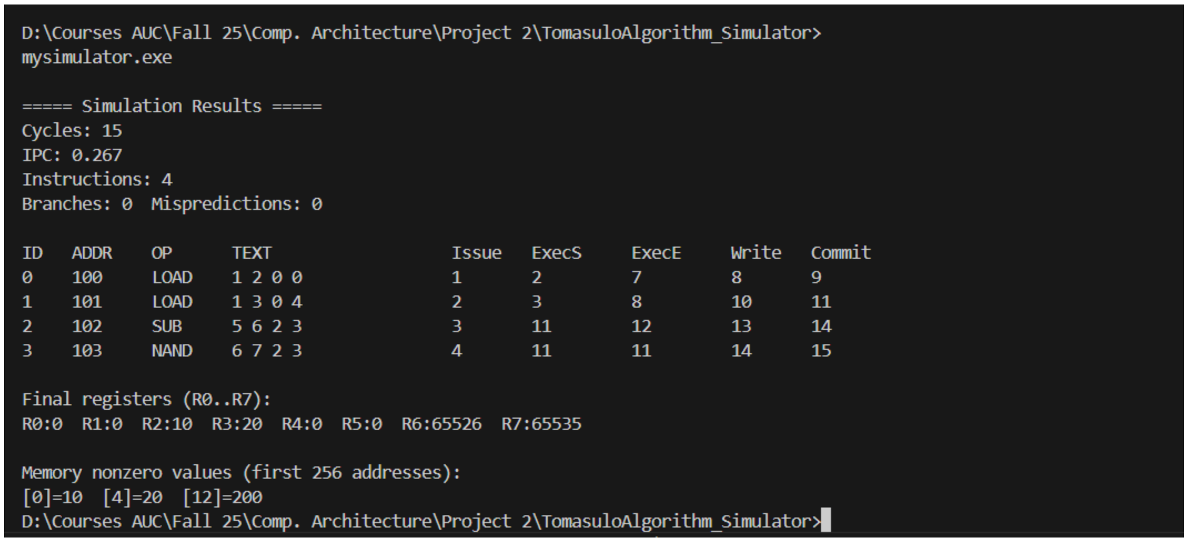
1. Testing the load, beq, mul, store, add together, which shows a correct result of the process. The branch resulted in the output correctly and didn’t affect any delay in the cycles or stalling cycles by mistake, representing how good the simulator handles the tests. The test case is in the folder “test cases in branch not taken.txt” in Github. The screenshot for the result in the simulator is:



```
Memory nonzero values (first 256 addresses):
[0]=10  [4]=20  [12]=200
D:\Courses AUC\Fall 25\Comp. Architecture\Project 2\TomasuloAlgorithm_Simulator>g++ -std=c++17 main.cpp -o mysimulator.exe

D:\Courses AUC\Fall 25\Comp. Architecture\Project 2\TomasuloAlgorithm_Simulator>mysimulator.exe

===== Simulation Results =====
Cycles: 32
IPC: 0.219
Instructions: 7
Branches: 1  Mispredictions: 0

ID   ADDR    OP      TEXT                 Issue   ExecS    ExecE    Write   Commit
0    100     LOAD    1 2 0 0              1       2        7        8       9
1    101     BEQ     3 1 2 2              2       9        9        10      11
2    102     LOAD    1 3 0 4              3       4        9        12      13
3    103     MUL     7 7 2 3              4       13       24       25      26
4    104     LOAD    1 6 0 12             8       9        14       15      27
5    105     ADD     4 3 3 3              9       13       14       17      28
6    106     STORE   2 3 0 8              10      11       12       18      32

Final registers (R0..R7):
R0:0  R1:0  R2:10  R3:40  R4:0  R5:0  R6:200  R7:200

Memory nonzero values (first 256 addresses):
[0]=10  [4]=20  [8]=40  [12]=200
D:\Courses AUC\Fall 25\Comp. Architecture\Project 2\TomasuloAlgorithm_Simulator>
```

Fig 1: test1

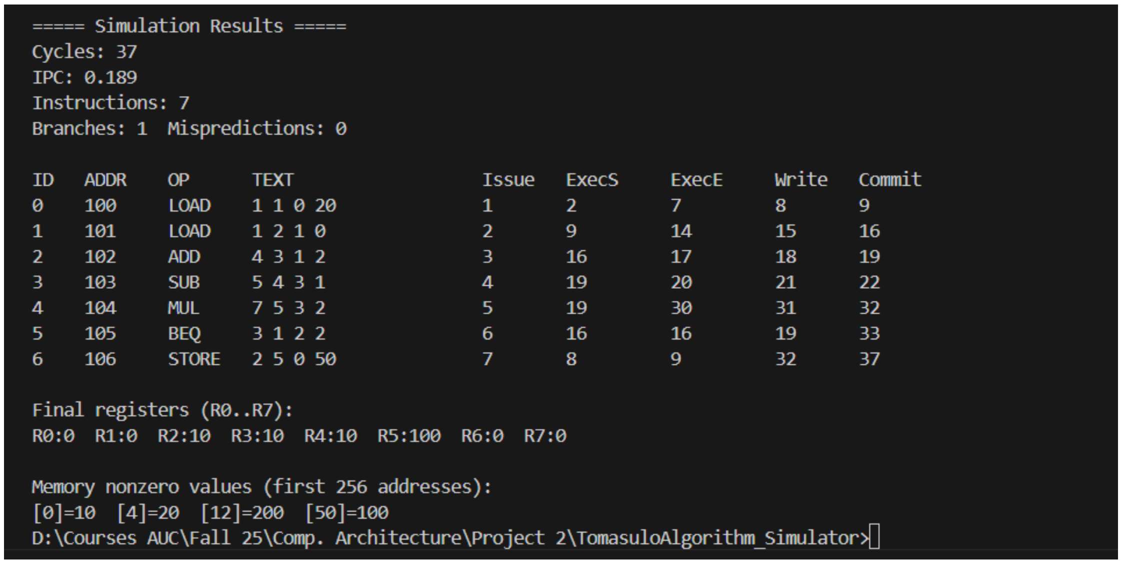
2. Testing the LOAD, NAND, SUB together, which shows a correct result of the process. The 2 loaded after each other resulted in an output correctly and didn’t affect any delay in the cycles, representing how good the simulator handles the tests. Both executed, written, and committed without being affected together. The test case is in the folder “test cases in sub, nand.txt” in Github. The screenshot for the result in the simulator is:



```
D:\Courses AUC\Fall 25\Comp. Architecture\Project 2\TomasuloAlgorithm_Simulator>
mysimulator.exe

===== Simulation Results =====
Cycles: 15
IPC: 0.267
Instructions: 4
Branches: 0  Mispredictions: 0

ID   ADDR    OP      TEXT                 Issue   ExecS    ExecE    Write   Commit
0    100     LOAD    1 2 0 0              1       2        7        8       9
1    101     LOAD    1 3 0 4              2       3        8        10      11
2    102     SUB     5 6 2 3              3       11       12       13      14
3    103     NAND    6 7 2 3              4       11       11       14      15

Final registers (R0..R7):
R0:0  R1:0  R2:10  R3:20  R4:0  R5:0  R6:65526  R7:65535

Memory nonzero values (first 256 addresses):
[0]=10  [4]=20  [12]=200
D:\Courses AUC\Fall 25\Comp. Architecture\Project 2\TomasuloAlgorithm_Simulator>
```

Fig 2: test2

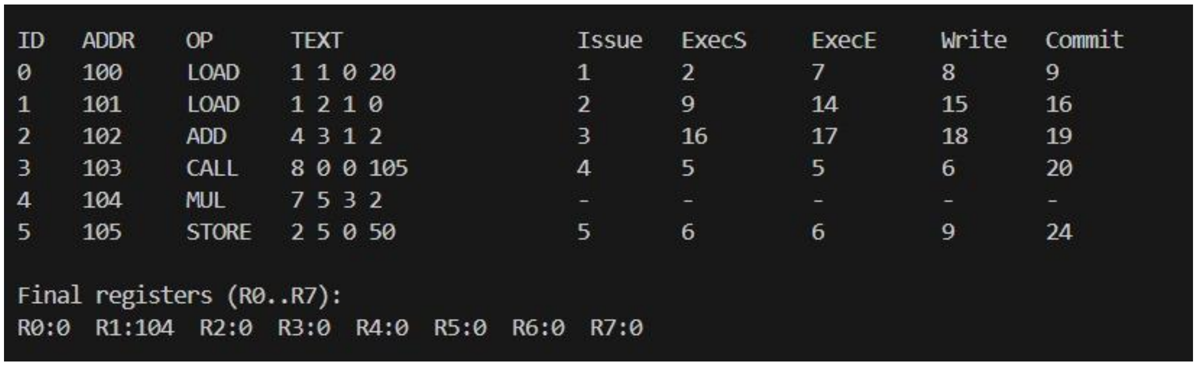
3. This test the read after write dependency between add and sub, which shows a correct result of the process. The 2 loaded after each other resulted in an output correctly, and the cycle count and delay is correctly handled.



```
===== Simulation Results =====
Cycles: 37
IPC: 0.189
Instructions: 7
Branches: 1  Mispredictions: 0

ID   ADDR   OP     TEXT               Issue   ExecS    ExecE    Write   Commit
0    100    LOAD   1 1 0 20           1       2        7        8       9
1    101    LOAD   1 2 1 0            2       9        14       15      16
2    102    ADD    4 3 1 2            3       16       17       18      19
3    103    SUB    5 4 3 1            4       19       20       21      22
4    104    MUL    7 5 3 2            5       19       30       31      32
5    105    BEQ    3 1 2 2            6       16       16       19      33
6    106    STORE  2 5 0 50           7       8        9        32      37

Final registers (R0..R7):
R0:0  R1:0  R2:10  R3:10  R4:10  R5:100  R6:0  R7:0

Memory nonzero values (first 256 addresses):
[0]=10  [4]=20  [12]=200  [50]=100
D:\Courses AUC\Fall 25\Comp. Architecture\Project 2\TomasuloAlgorithm_Simulator>
```

Fig 3: test3

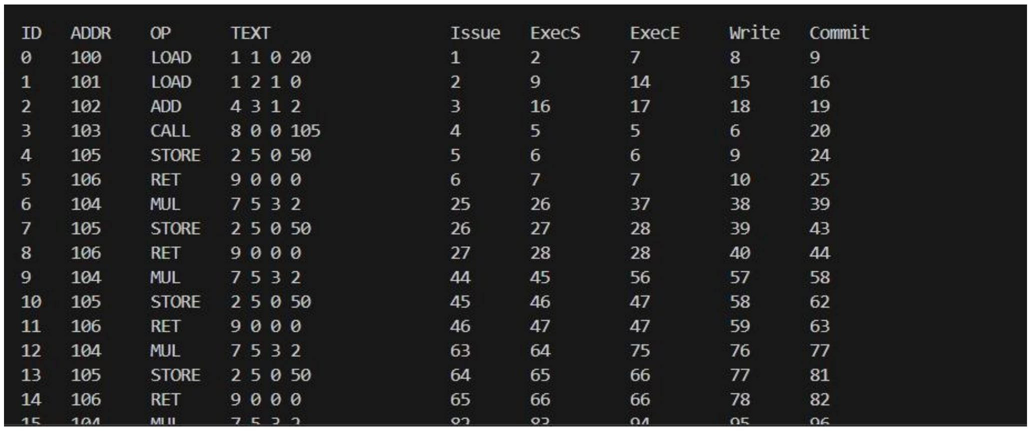
4. For the call instruction, it works well which shows a correct result of the process as shown in the test below:



```
ID   ADDR   OP     TEXT               Issue   ExecS    ExecE    Write   Commit
0    100    LOAD   1 1 0 20           1       2        7        8       9
1    101    LOAD   1 2 1 0            2       9        14       15      16
2    102    ADD    4 3 1 2            3       16       17       18      19
3    103    CALL   8 0 0 105          4       5        5        6       20
4    104    MUL    7 5 3 2            -       -        -        -       -
5    105    STORE  2 5 0 50           5       6        6        9       24

Final registers (R0..R7):
R0:0  R1:104  R2:0  R3:0  R4:0  R5:0  R6:0  R7:0
```

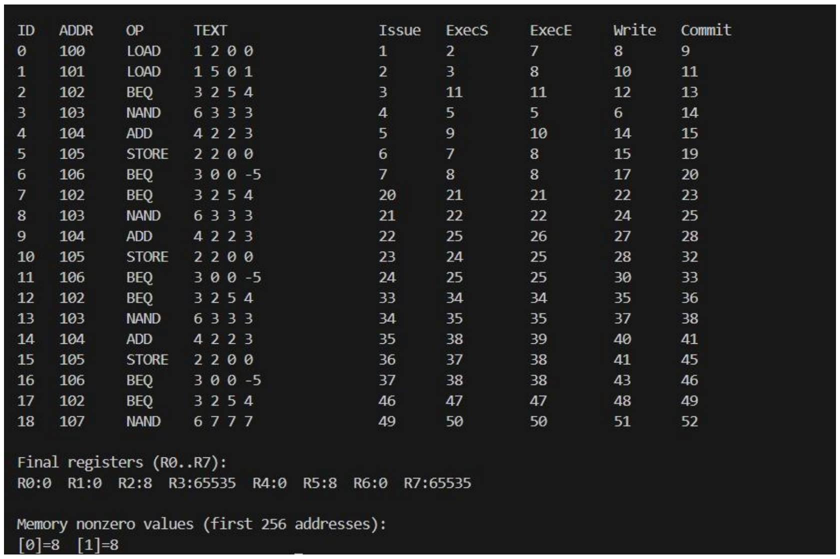
Fig4: test4

5. For the ret instruction, it works well, too by looping correctly, as shown below:



```
ID   ADDR   OP     TEXT               Issue   ExecS    ExecE    Write   Commit
0    100    LOAD   1 1 0 20           1       2        7        8       9
1    101    LOAD   1 2 1 0            2       9        14       15      16
2    102    ADD    4 3 1 2            3       16       17       18      19
3    103    CALL   8 0 0 105          4       5        5        6       20
4    105    STORE  2 5 0 50           5       6        6        9       24
5    106    RET    9 0 0 0            6       7        7        10      25
6    104    MUL    7 5 3 2            25      26       37       38      39
7    105    STORE  2 5 0 50           26      27       28       39      43
8    106    RET    9 0 0 0            27      28       28       40      44
9    104    MUL    7 5 3 2            44      45       56       57      58
10   105    STORE  2 5 0 50           45      46       47       58      62
11   106    RET    9 0 0 0            46      47       47       59      63
12   104    MUL    7 5 3 2            63      64       75       76      77
13   105    STORE  2 5 0 50           64      65       66       77      81
14   106    RET    9 0 0 0            65      66       66       78      82
```

6. For the loop, the test case are working well, as well. The loop function generates



```
ID   ADDR   OP     TEXT               Issue  ExecS    ExecE    Write   Commit
0    100    LOAD   1 2 0 0            1      2        7        8       9
1    101    LOAD   1 5 0 1            2      3        8        10      11
2    102    BEQ    3 2 5 4            3      11       11       12      13
3    103    NAND   6 3 3 3            4      5        5        6       14
4    104    ADD    4 2 2 3            5      9        10       14      15
5    105    STORE  2 2 0 0            6      7        8        15      19
6    106    BEQ    3 0 0 -5           7      8        8        17      20
7    102    BEQ    3 2 5 4            20     21       21       22      23
8    103    NAND   6 3 3 3            21     22       22       24      25
9    104    ADD    4 2 2 3            22     25       26       27      28
10   105    STORE  2 2 0 0            23     24       25       28      32
11   106    BEQ    3 0 0 -5           24     25       25       30      33
12   102    BEQ    3 2 5 4            33     34       34       35      36
13   103    NAND   6 3 3 3            34     35       35       37      38
14   104    ADD    4 2 2 3            35     38       39       40      41
15   105    STORE  2 2 0 0            36     37       38       41      45
16   106    BEQ    3 0 0 -5           37     38       38       43      46
17   102    BEQ    3 2 5 4            46     47       47       48      49
18   107    NAND   6 7 7 7            49     50       50       51      52

Final registers (R0..R7):
R0:0  R1:0  R2:8  R3:65535  R4:0  R5:8  R6:0  R7:65535

Memory nonzero values (first 256 addresses):
[0]=8  [1]=8
```

Those are the main test cases for the simulator.

4. AI Usage

AI tools were used selectively during this project to help resolve certain logical issues in the code, particularly related to Tomasulo's algorithm behavior, speculation handling, and correct synchronization between the simulator components. The AI also provided suggestions to make some parts of the implementation simpler and clearer—such as structuring the ROB and reservation stations, improving condition checks, and reducing redundant code. All AI-assisted outputs were fully understood, rewritten, and tested by the team to ensure correctness.