Faculty of Media Engineering and Technology
Dept. of Computer Science and Engineering
Assoc. Prof. Milad Ghantous

CSEN 702: Microprocessors
Winter 2025

Project requirements (Groups of 6)

You are required develop a GUI simulator in any language you're comfortable with, for the Tomasulo algorithm. Preferably, you can use Java FX and represent everything as tables. You're not required to develop a complex GUI, just enough to represent the status of the system and the content each cycle.

- Your simulator should accept inputs (MIPS instructions in assembly format, either by constructing them graphically using select lists or by loading a text file containing the code) and show **step by step** (cycle by cycle) how these instructions are executed as well as the content of each reservation station/buffer, the register file, the cache and the queue.)
- Your simulator should handle all types of codes, with *and* without a loop, and that might contain a combination of RAW, WAR and WAW hazards.
- Your simulator should include ALU ops (FP adds, subs, multiply, divide), (integer ADDI or SUBI needed for loops), loads and stores and branches. You can only implement LW, LD and L.S and L.D (and same for stores)
- The user should be able to enter the cache hit latency and penalty.
- Cache misses need to be incorporated by allowing the user to choose the block size and the cache size. Only consider misses for the data and not the instructions. Your addressing strategy and locations should be well documented.
- Address clashes should be addressed in Tomasulo.
- The user should be able to input the latency of each type of instruction before we start simulating. All instructions (integer and floating) enter the architecture (unlike the lecture). You can create new reservation stations types for integer instructions.
- No branch prediction is used.
- The user should be able to select the size of all stations and buffers.
- The register file can be pre-loaded with some values or allow the user to load them.

- When two instructions wish to publish their result on the bus in the same cycle, you need to handle that and explain how you did so.

Deliverables:

- A report explaining the steps you did to develop and run and test your code. The report should explain your approach, code structure, and test cases.
- A demo (while evaluating the project with your TA).

Warning:

- **Copying a project (or sharing codes) from the internet or from each other will result in a zero for both parties, the copying and the copied.**
- **It's better to submit a non-complete project than to submit a stolen one.**

Notes and deadlines:

The work should be divided equally among all group members according to the complexity of the project and its content. An equal grade for all group members is not guaranteed. In the evaluation, be prepared for any question. All group members should know how to answer.

Submission: Friday December 5, 2025

Evaluations will start Saturday December 6, 2025.

Project submission by Google form which will be shared later.

Submit ONE zipped folder named in the format "TeamXX" containing:

- The report,
- Source code and
- Text file containing your names, IDs and tutorial numbers.

Submission link: https://forms.gle/vSUpe9mUuC2V15Uu9

Notes

- The memory should be byte addressing. You can think about it as an array of bytes where each element is 8 bits/1 byte.
- A **compulsory miss** occurs when we access an address in a block and that block is not available in the cache.
- When we access an address in a block and that block is available in the cache, it is a hit.

- ● If you are loading a word then you get 4 bytes from the cache if the block is there. If the block is not there, then you should get the block from the memory and put it in the cache and then you read the 4 bytes of this block.

○ Example: LW R1, 100(R2).

i. You will first compute the cache address where address 100+R2 is mapped onto. If its block is available then it hits. Otherwise it misses.

ii. If it hits, then you get the 4 bytes with memory addresses 100+R2, 101+R2, 102+R2, 103+R2 from the cache.

iii. If it misses, then you will get its corresponding block from the memory and put it in the cache. Then get addresses 100+R2, 101+R2, 102+R2, 103+R2.

iv. Similarly the rest of the loads.

- ● For the L.S and L.D for the single and double precision floating point operation, they will be handled similarly to LW and LD for the integer operation.
- ● No need to handle the floating point numbers aka mantissa, exponent and so on. Handle it as you handle the integer values.
- ● The size of both the integer, floating point registers can be assumed to have the same size as a block of memory.
- ● For the branches BEQ or BNE, assume that they take two source registers and the address it will branch onto.
    - ○ Example 1: BEQ R1, R2, 0
    - ○ Example 2: BNE R1, R2, 0

Test Cases Test Case 1: Sequential Code

```
L. D F6, 0(R2)
L. D F2, 8(R2)
MUL. D F0, F2, F4
SUB. D F8, F2, F6
DIV. D F10, F0, F6
ADD. D F6, F8, F2
S.D F6, 8(R2)
```

Test Case 2: Sequential Code

```
L. D F6, 0(R2)
ADD. D F7, F1, F3
L. D F2, 20(R2)
MUL. D F0, F2, F4
SUB. D F8, F2, F6
DIV. D F10, F0, F6
S.D F10, 0(R2)
```

Test Case 3: Loop Code

```
DADDI R1, R1, 24
DADDI R2, R2, 0
LOOP: L.D F0, 8(R1)
```

MUL.D F4, F0, F2
S.D F4, 8(R1)
DSUBI R1, R1, 8
BNE R1, R2, LOOP
Where the word LOOP can be replaced with the address of the L.D instruction.