**Practical 2: Instruction Pipeline**

Report Pipeline

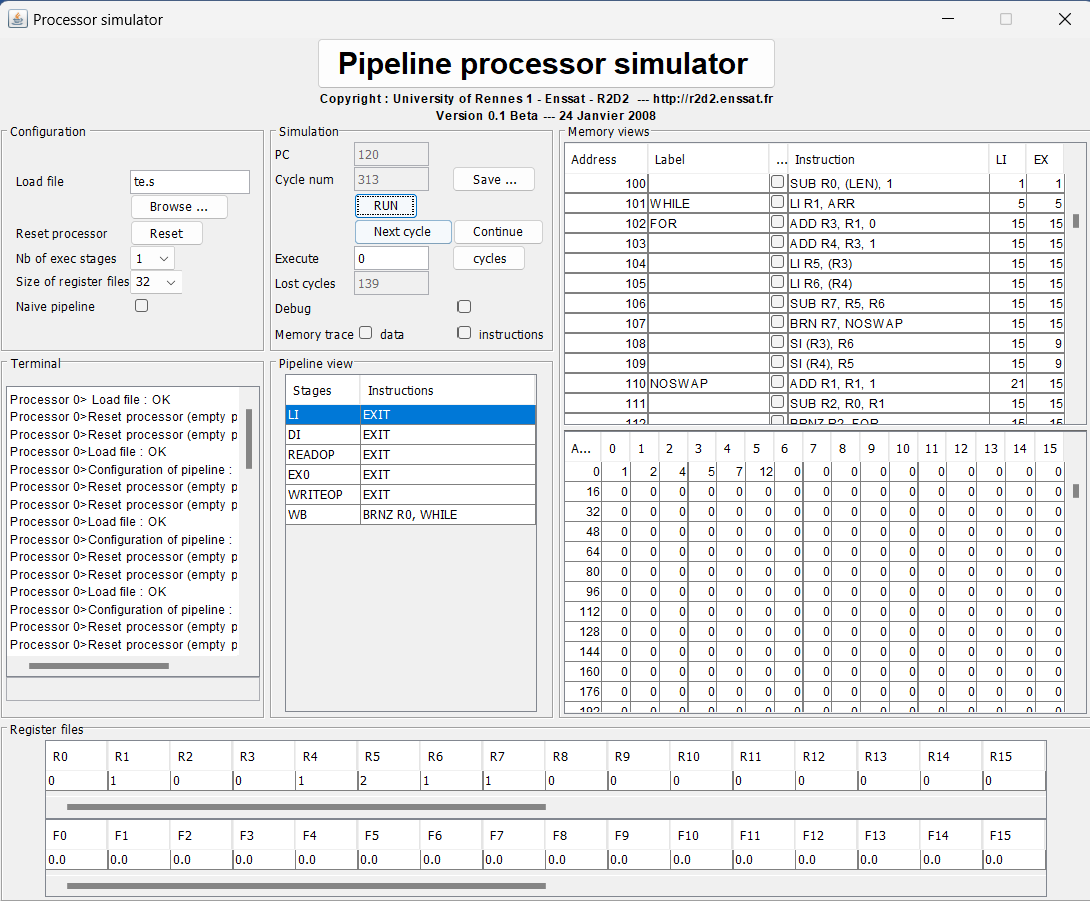
**I. Number of exec stages - Lost cycles:**



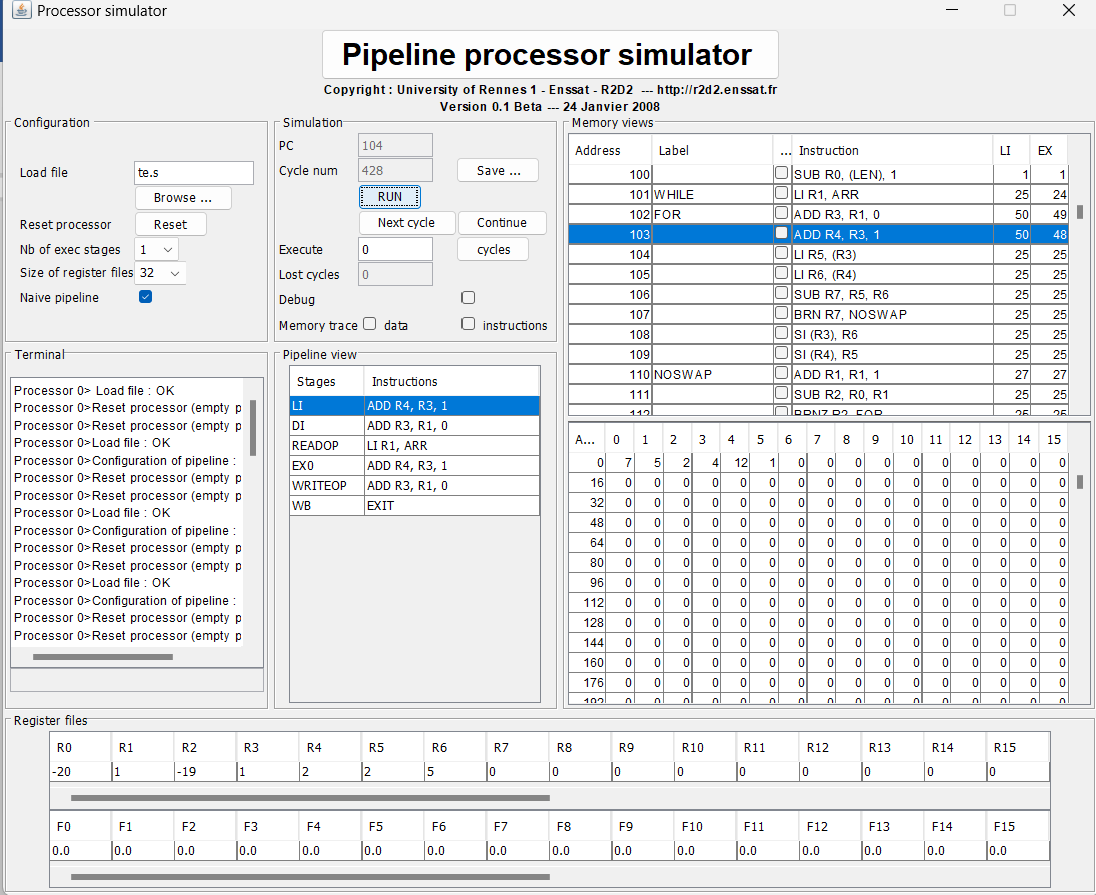
**\* Explain:** When instructions are processed in different stages, the number of instructions being executed simultaneously varies. This variance can lead to different amounts of lost cycles during execution. Generally, as the number of stages increases, so does the number of lost cycles. With more execution stages, each instruction must go through additional steps before completion, potentially increasing the number of cycles in the pipeline. However, in some cases, this number might decrease. Despite the possible rise in total cycles and lost cycles, the final output remains accurate. Adding more execution stages enhances pipeline performance by allowing multiple instructions to be processed concurrently, thus increasing parallelism.

**II. The correctness of the program with and without naive mode:**

* **Without naive mode:** The program execute smoothly without any issue and gives the result similar to the JVEM Simulator.



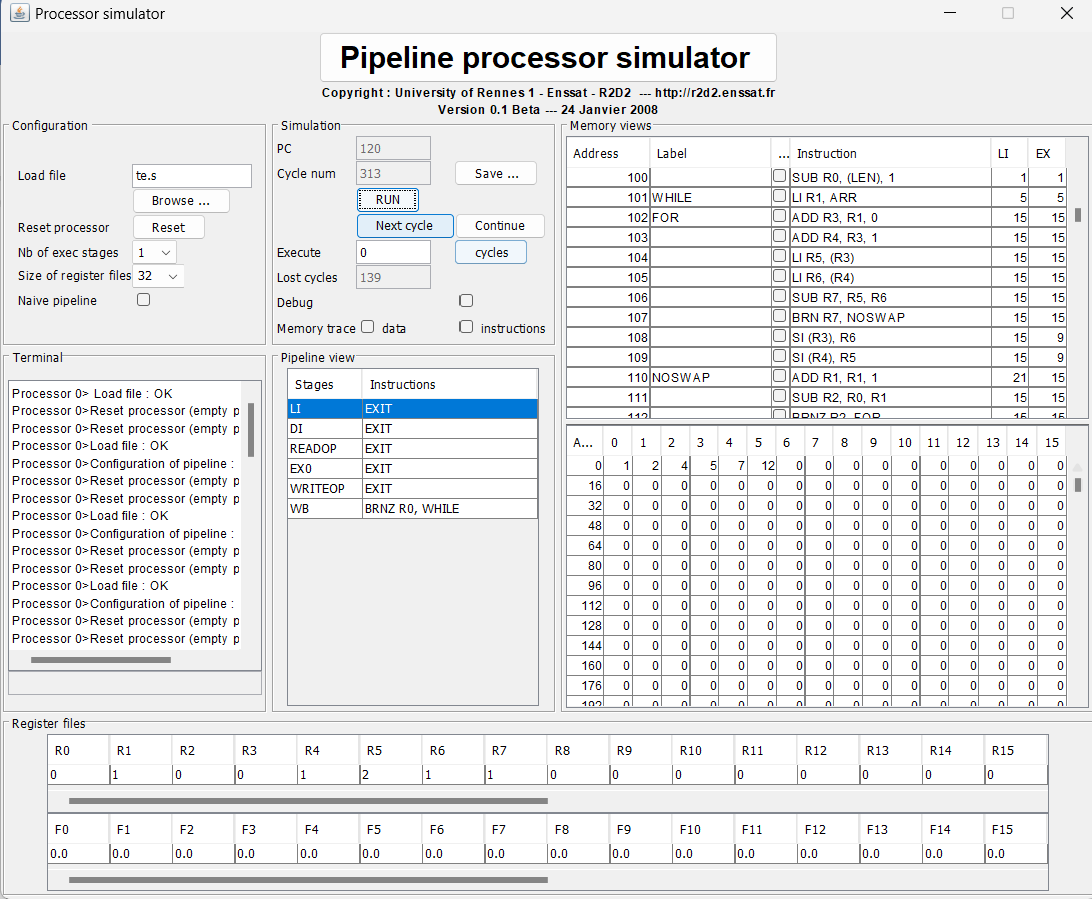
* **With naive mode:** The program is not running correctly.



- In standard pipeline mode, results are more accurate because each instruction progresses through distinct pipeline stages. This ensures each instruction's outcome is properly processed before advancing to the next stage. The previous instruction's value must reach the Write Back (WB) stage and be updated before any dependent instruction executes. For instance, in the instruction "add R5, R6, R7", the result R5 = R6 + R7 is saved first, then the next instruction, like "li R0, R5", is executed.

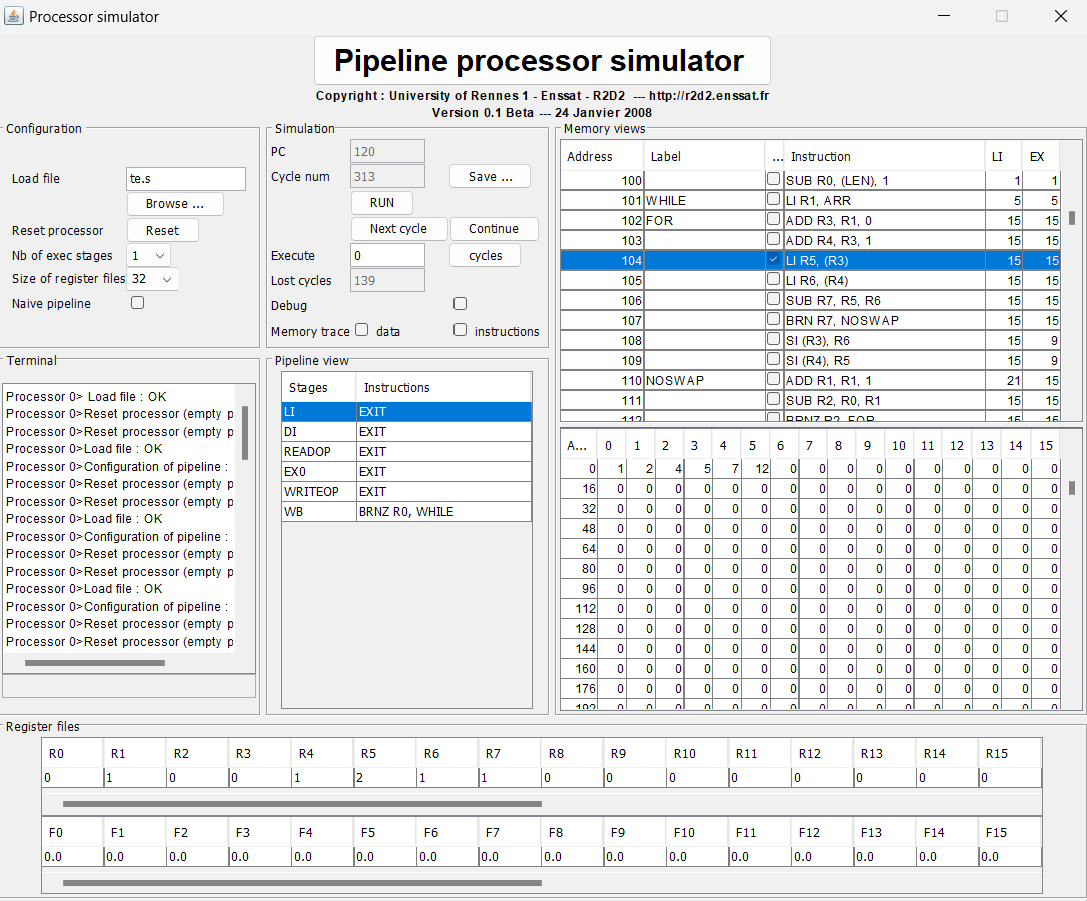
Naive pipeline mode, however, can lead to incorrect execution because instructions dependent on previous results may not wait for those results. For example, if R6 and R7 are loaded from memory and then immediately used in a comparison, swap, addition, or subtraction, the operation might execute before the load is complete or before proceeding to the next instruction involving these values. This can result in incorrect execution.

**III. Analyze the execution of bubble algorithm in pipeline:  
Note if the execution is correct or not:** The execution without naive pipeline mode is correct.

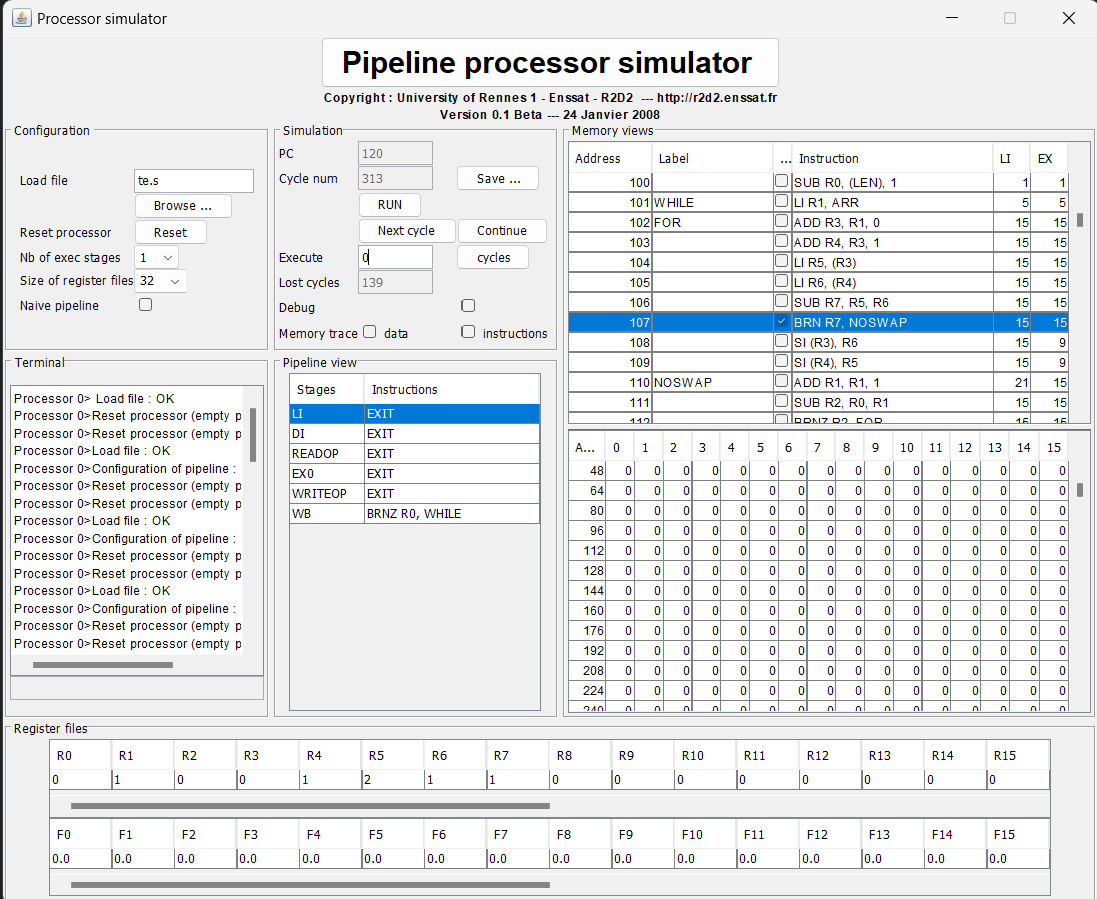


**IV. Capture case when a branch is executed and understand what happens**

When a branch is executed, the instructions within the branch are processed in the new sequence, and the instructions following the branch are not executed until the branch is resolved.



In this scenario, I first have instructions to calculate the length of the initialized array for implementing the loop. Then, I set this value to R6 and decrement it by 1 at the end of each loop iteration. When the value of R6 reaches 0, the loop will terminate, and the program will exit to prevent an infinite loop.



In this case, the value of R3 is calculated as R1 - R2, where R1 is one element in the array and R2 is the element following R1. If R3 is negative (brn R3), it means that R1 < R2, so according to the bubble sort algorithm, R1 and R2 will not be swapped.