Project 07 – DLX Hazards

# Procedure

The goal of this project was to handle all of the data and control hazards. This would allow our code to run without the inclusion of no-ops. In order to handle data hazards we implemented a fast forwarding mechanism in our processor to push results from both the memory stage and the write-back stage directly back into the ALU when required by a sequence of instructions.

In order to handle control hazards, we implemented a mechanism which always assumes the branch will not be taken, then the pipeline can be flushed if that assumption proves wrong. We took this approach because the instructions following the branching instructions enter the pipeline already without any modification to our system. Assuming that branches are always taken would have required a more substantial change to our system.

We were not able to completely debug our system, and our factorial program still requires a few no-ops, but it has been greatly reduced as a result of our work. The before and after comparison of our program is shown in Figure 1.

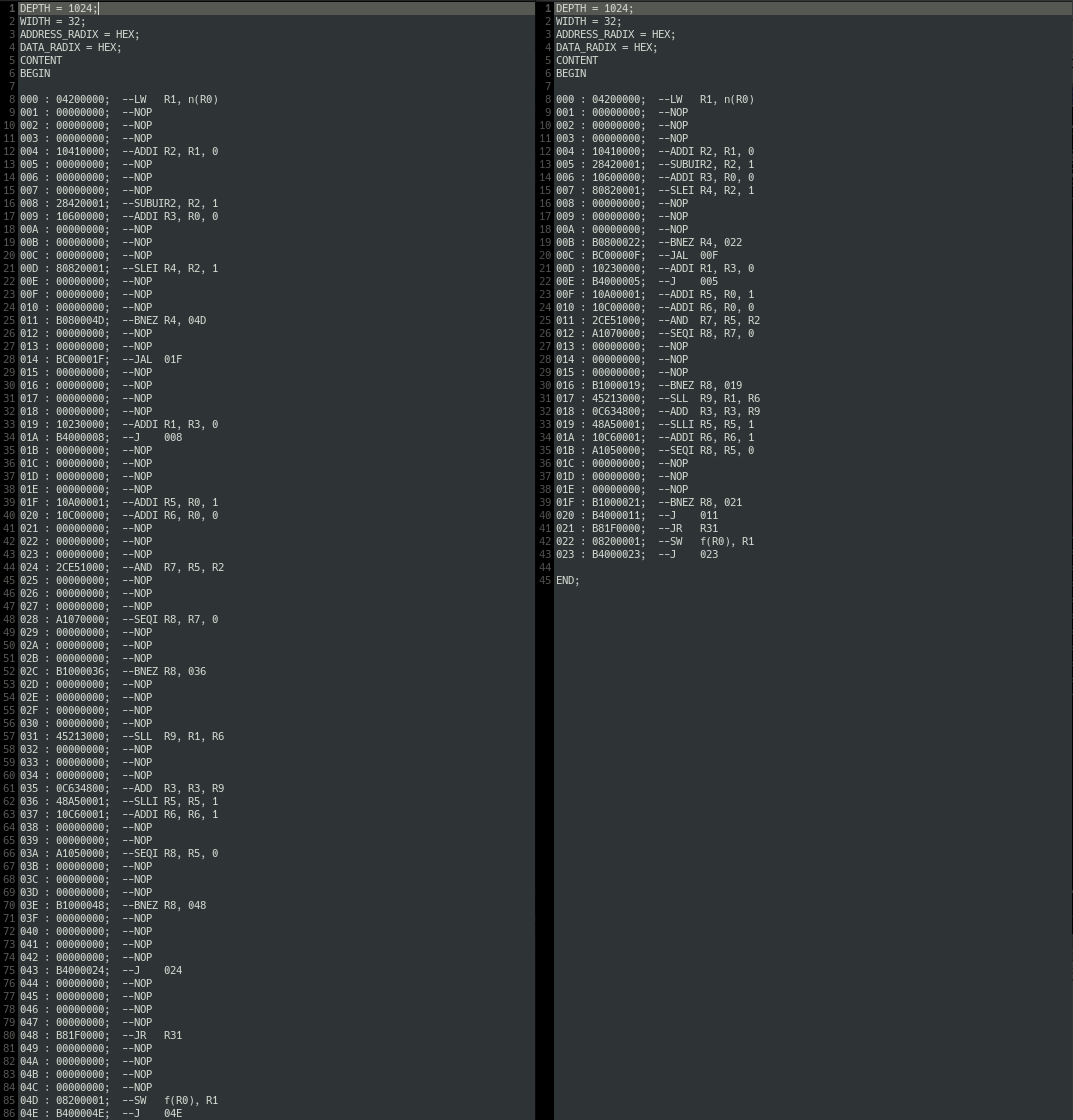


Figure 1: Final Factorial Code Reduction

As a result, our program execution greatly improved as well. We saw a 30% improvement in execution time according to ModelSim. The before and after execution results are shown in Figures 2 and 3. The flag marker in both figures shows the end of execution before entering the final “done” loop.

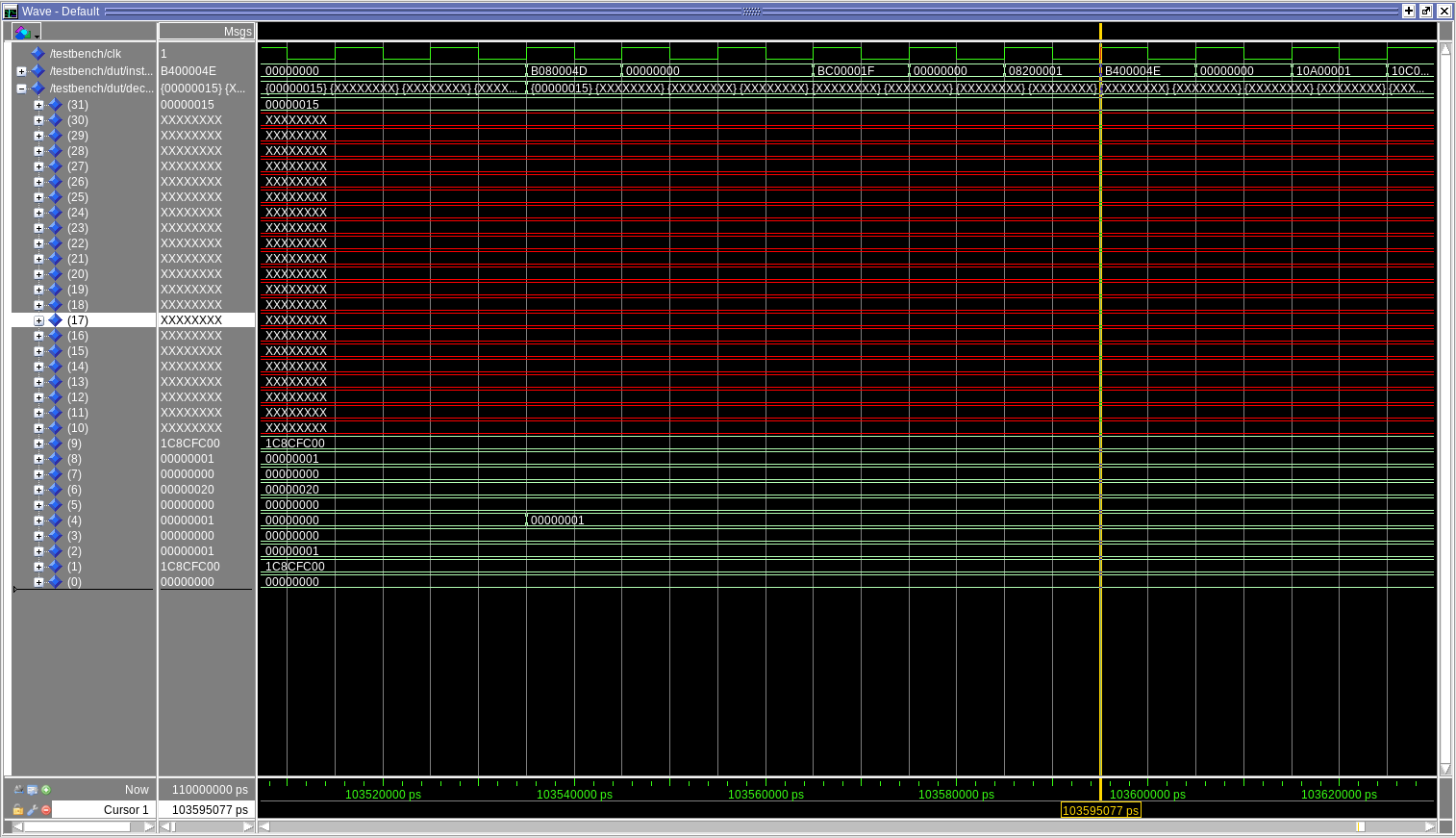


Figure 2: Initial Execution

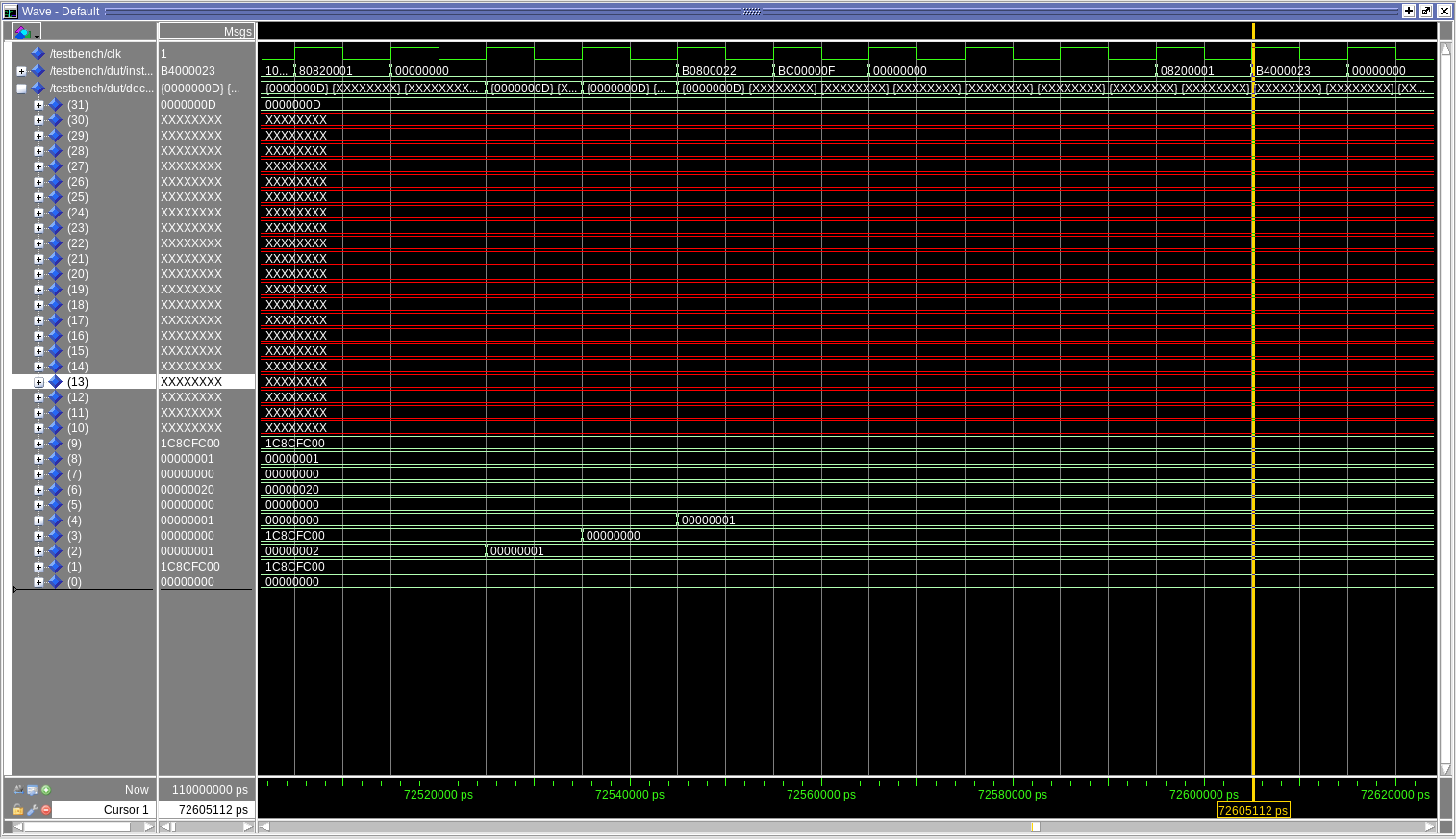


Figure 3: Final Execution

# Results

We were mostly successful in eliminating the data and control hazards in our DLX processor, improving the efficiency greatly.

# Conclusions

While this stage of design was not as successful as we had hoped, we definitely have everything in place to finish later this week. The majority of hazards are handled successfully, and the few that are not should be a matter of minor debugging. We just ran out of time.