

MIPS Project

California Polytechnic University – Pomona

College of Engineering

ECE 425 – Computer Architecture



December 5, 2016

To: Dr. Halima El-Naga

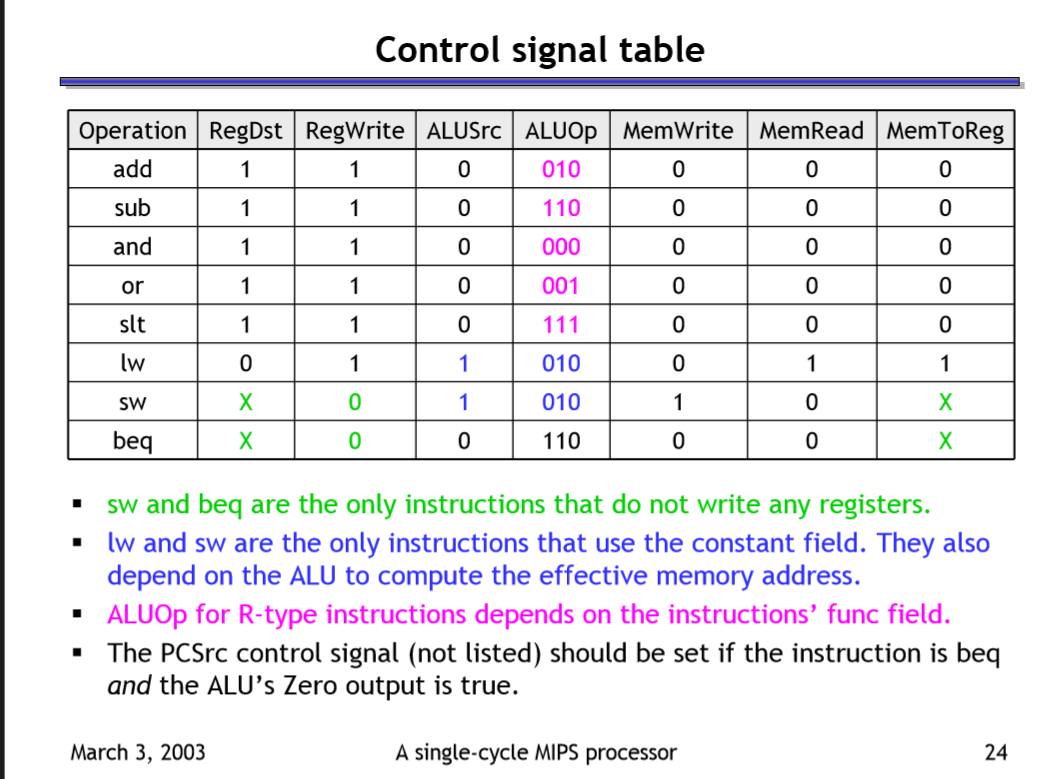
By: Tony Le, Josue Llamas

**MIPS Processor**

In the final project for ECE 425 Computer Architecture, we are tasked in designing and implementing a MIPS processor. Our main goal is to program the MIPS processor to simulate the following instruction for us: R3 = R1\*R2. Using what we learned in ECE425 and along with ECE205 and ECE424, we attempted to complete this task in a short amount of time. First we will demonstrate the essential components that are necessary for our processor via the code and its simulation.

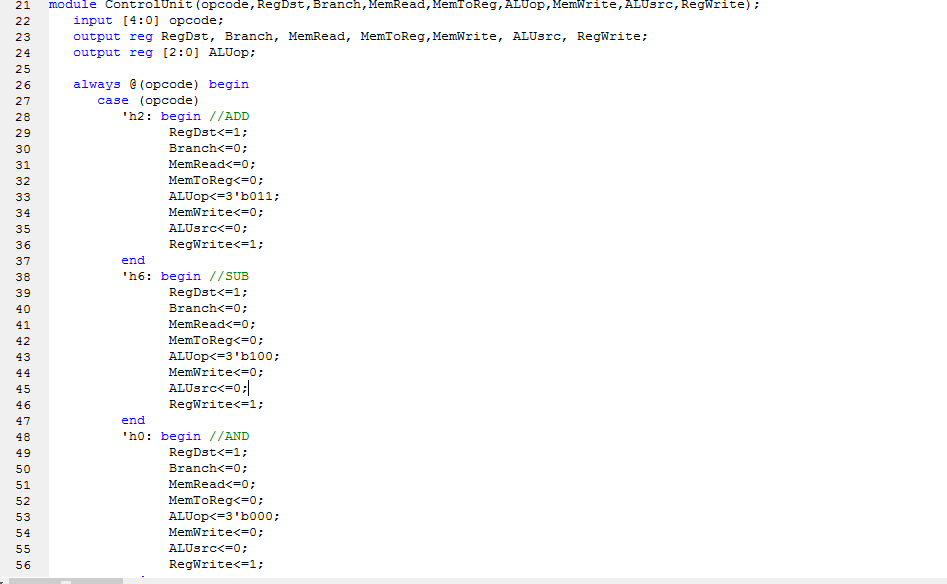
1. **Control Unit**

For the control unit we didn’t really understand how we were supposed to implement this part but we found this slide of a PowerPoint. This slide gave us a good breakdown of what controls are being affected depending on the operation.



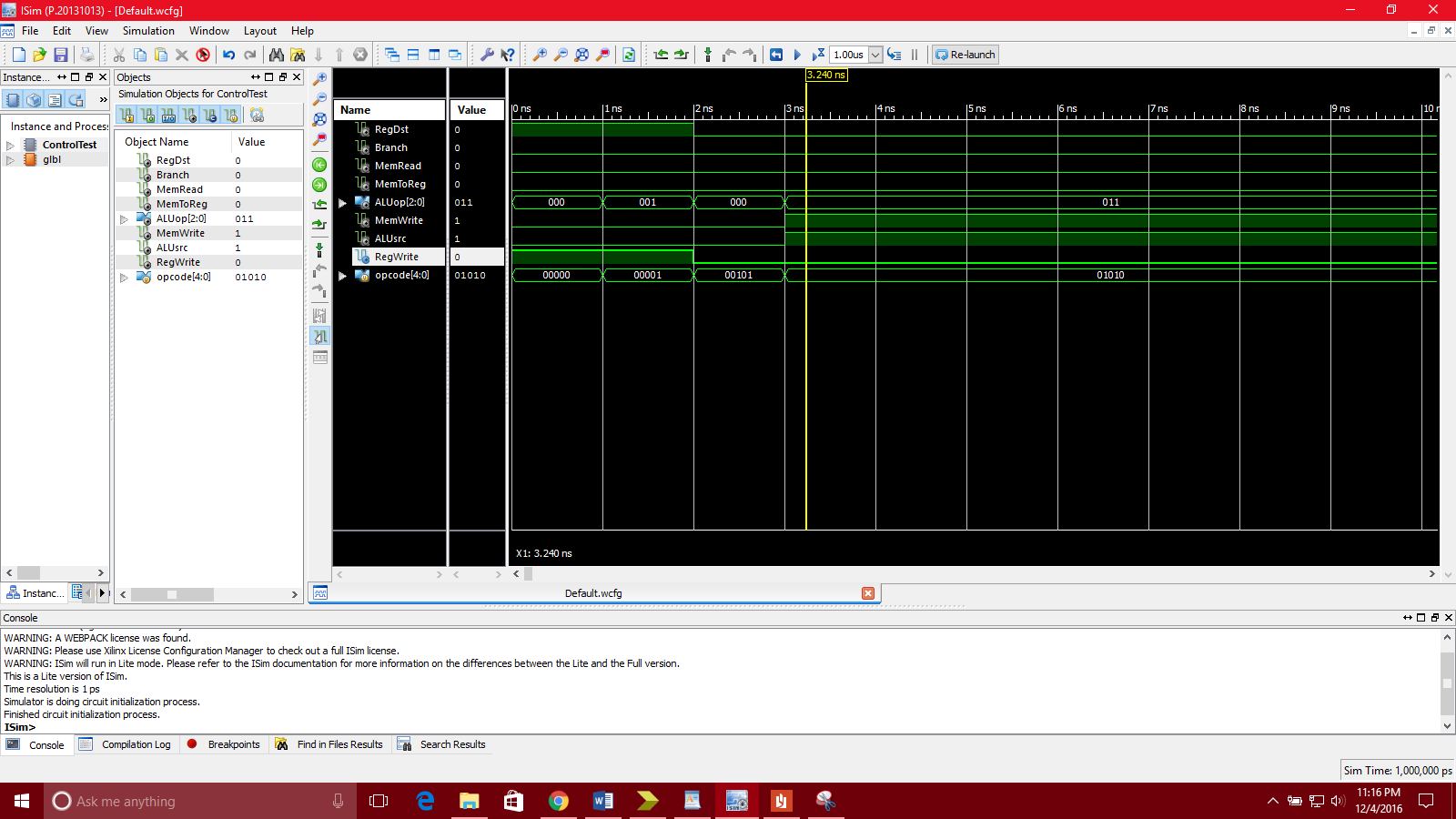
1. Controlunit**.**v

The inputs and outputs can be seen below.



For this code it was a simple case statement depending on the Opcode.

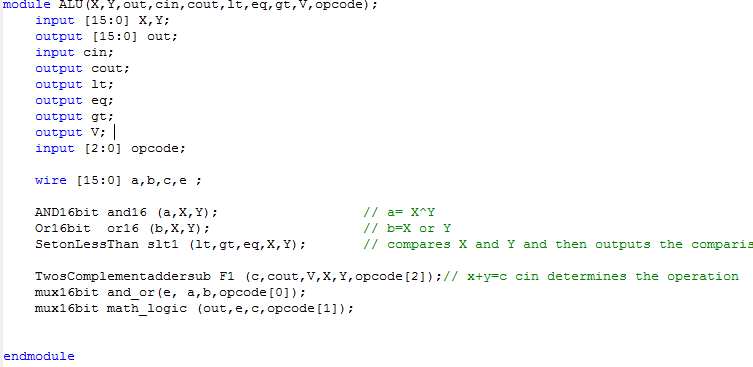
1. Control Test Simulation



1. **Arithmetic Logic Unit (ALU)**

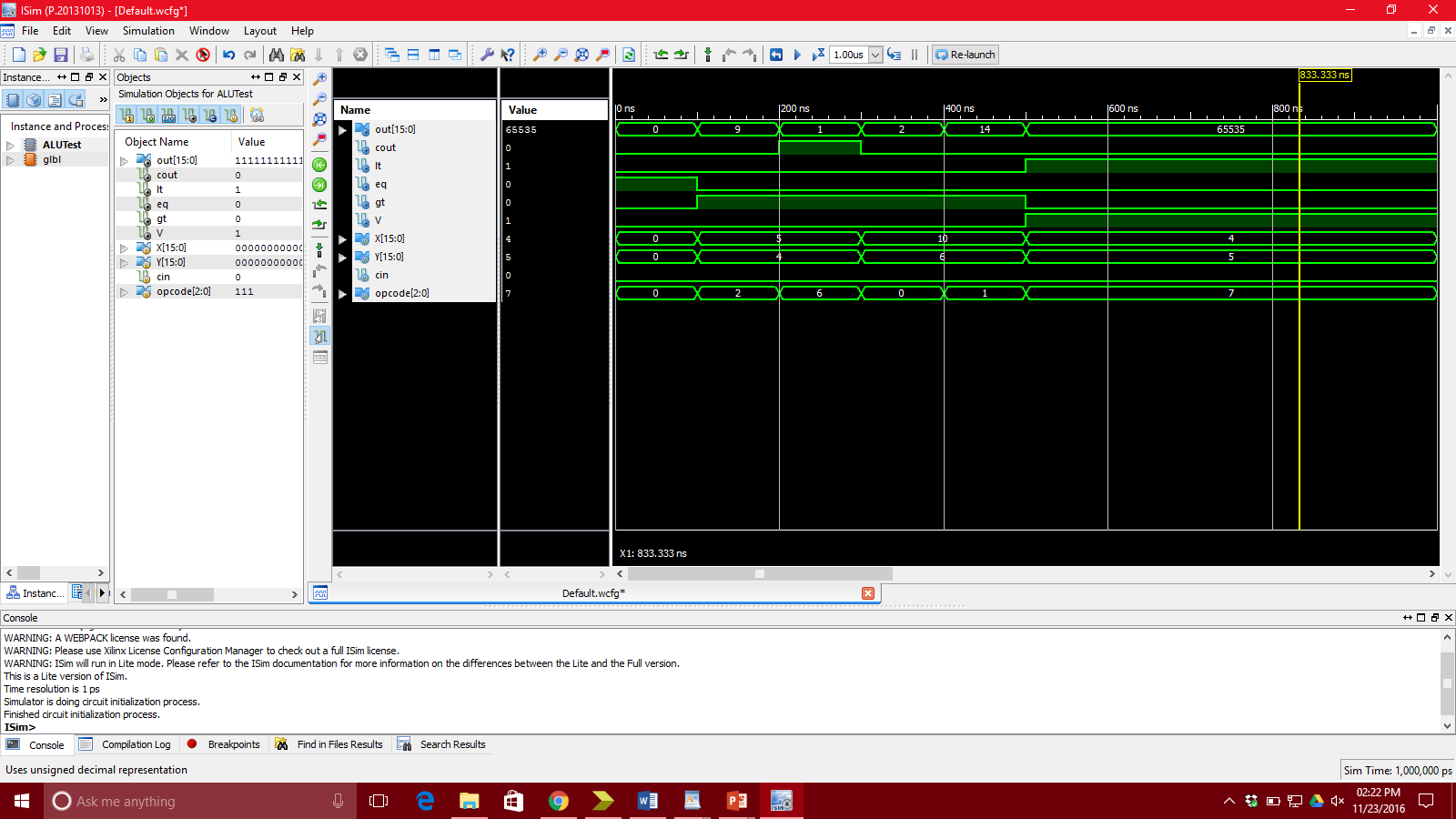
The code works, but I didn’t see a need to use the Cin value, if we already had the opcode to choose addition or subtraction for us. Therefore, Cin is not used. Lt, gt, cout, and V were not used in the MIPS module

* 1. **ALU Code**



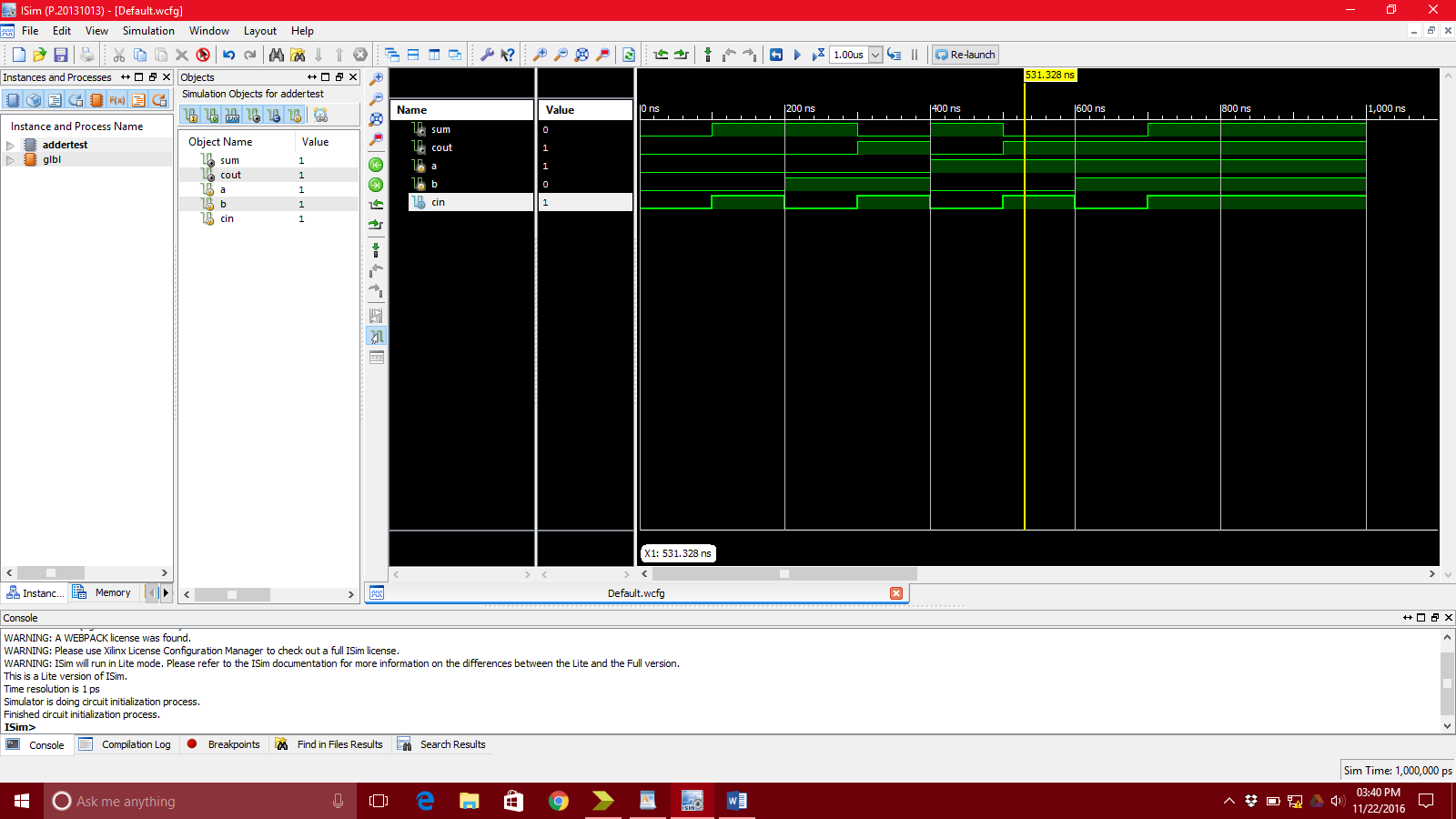
* 1. **ALU Simulation**

4+5, 10 and/or 6 and 4<5 were tested correctly

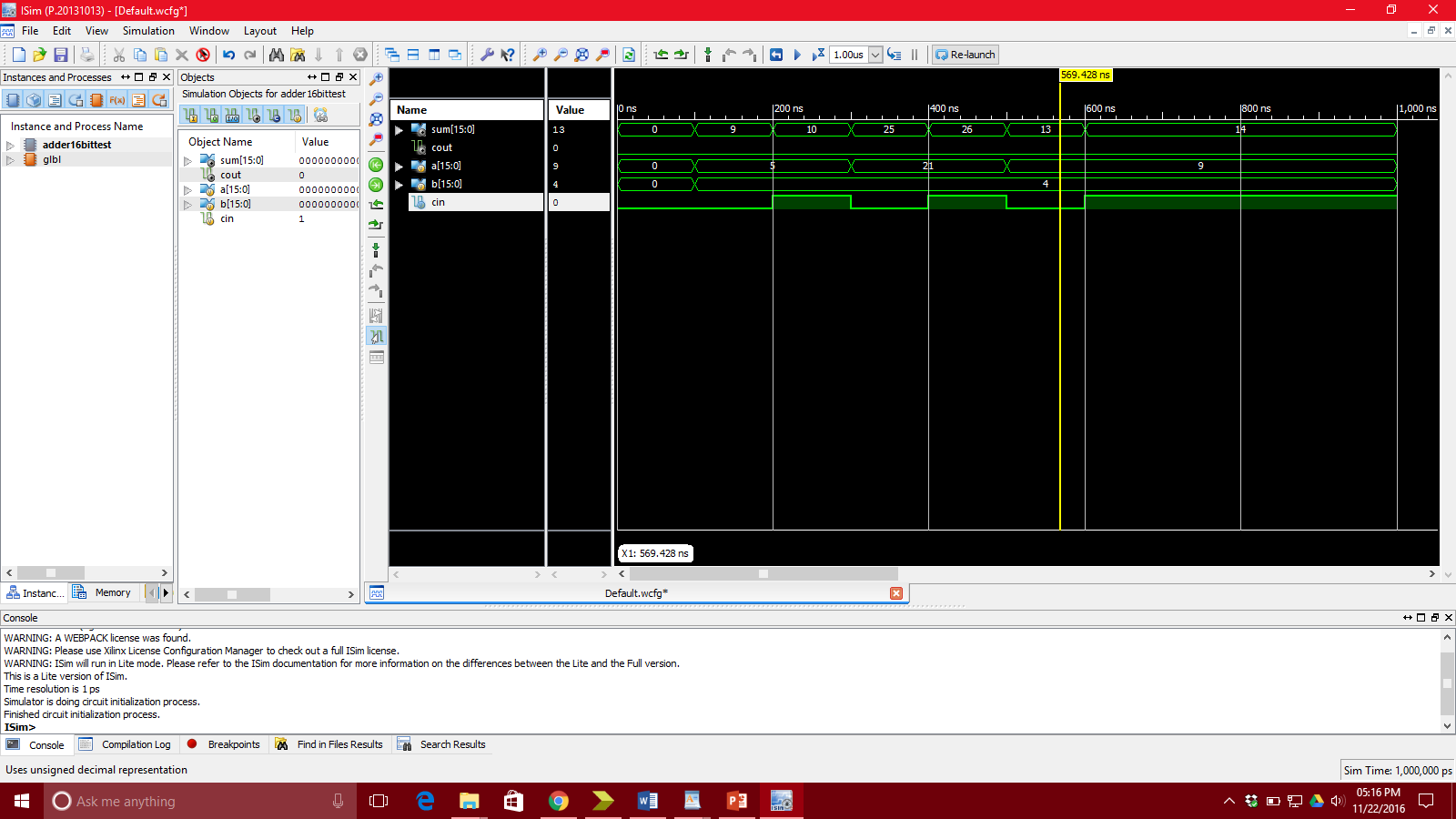


In the simulation, you can see that the ALU works perfectly depending on the opcode given to it.

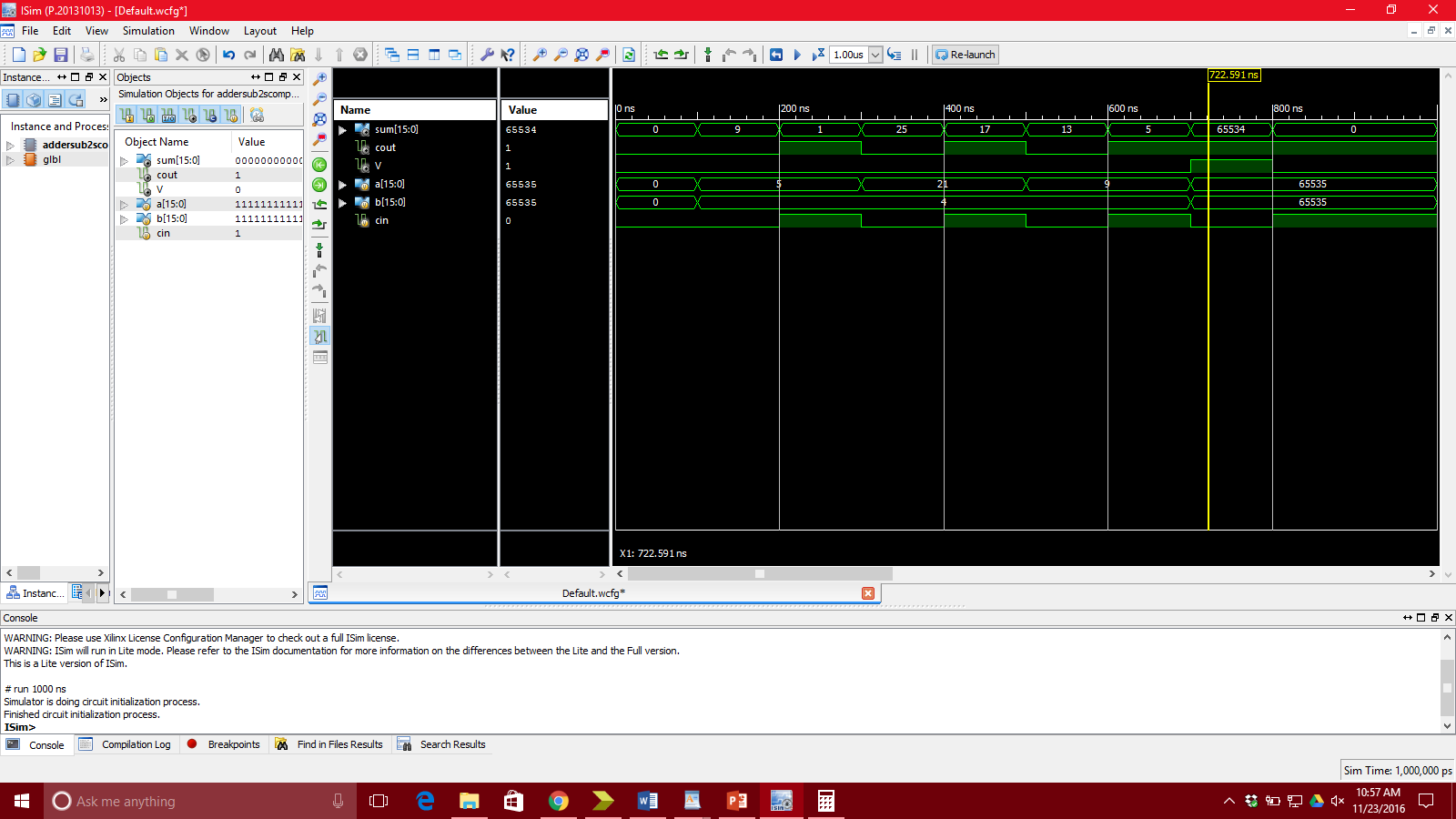
* 1. **Additional Modules**
     1. Adder Test Simulation



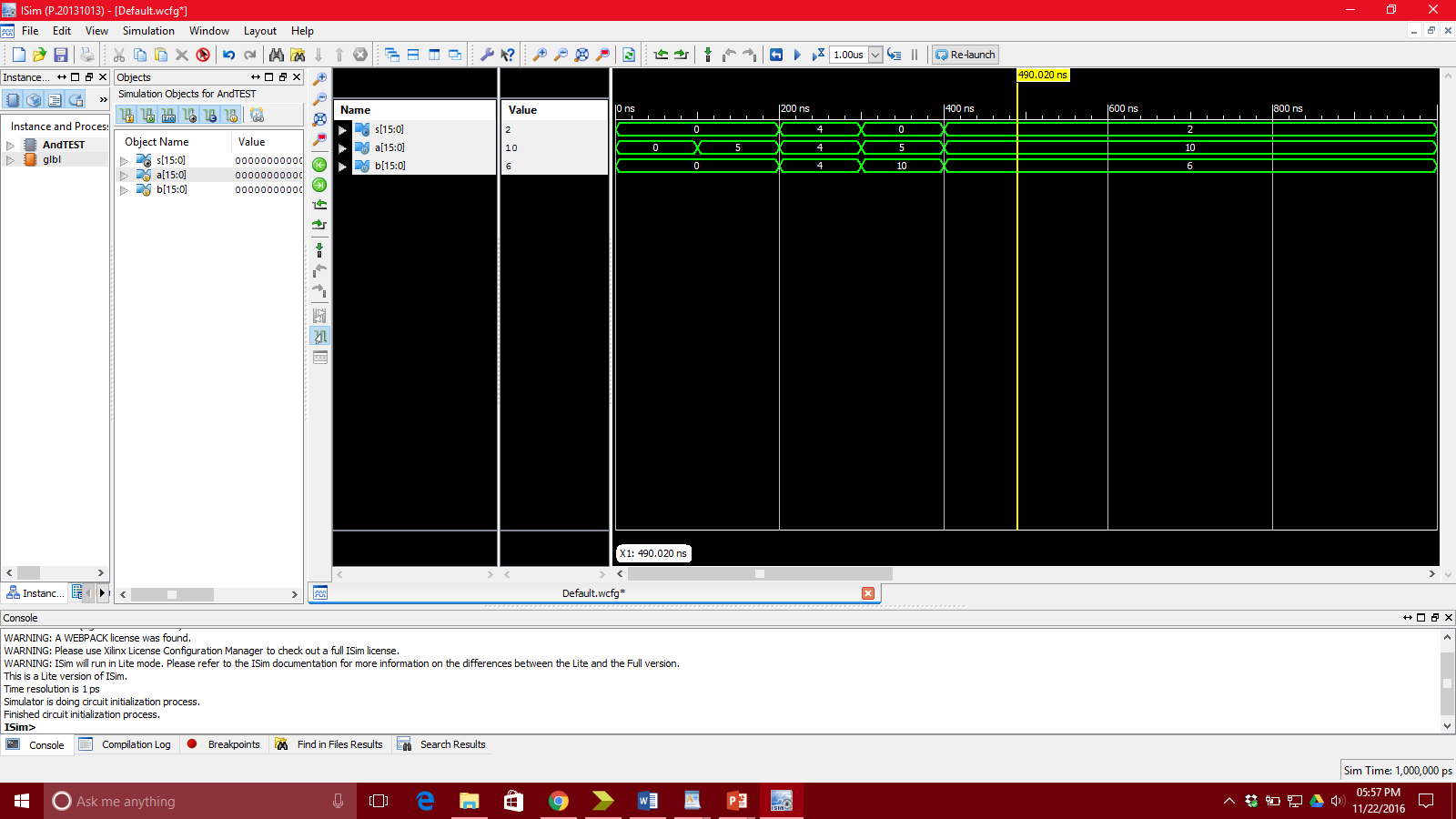
* + 1. 16-bit Adder Test Simulation



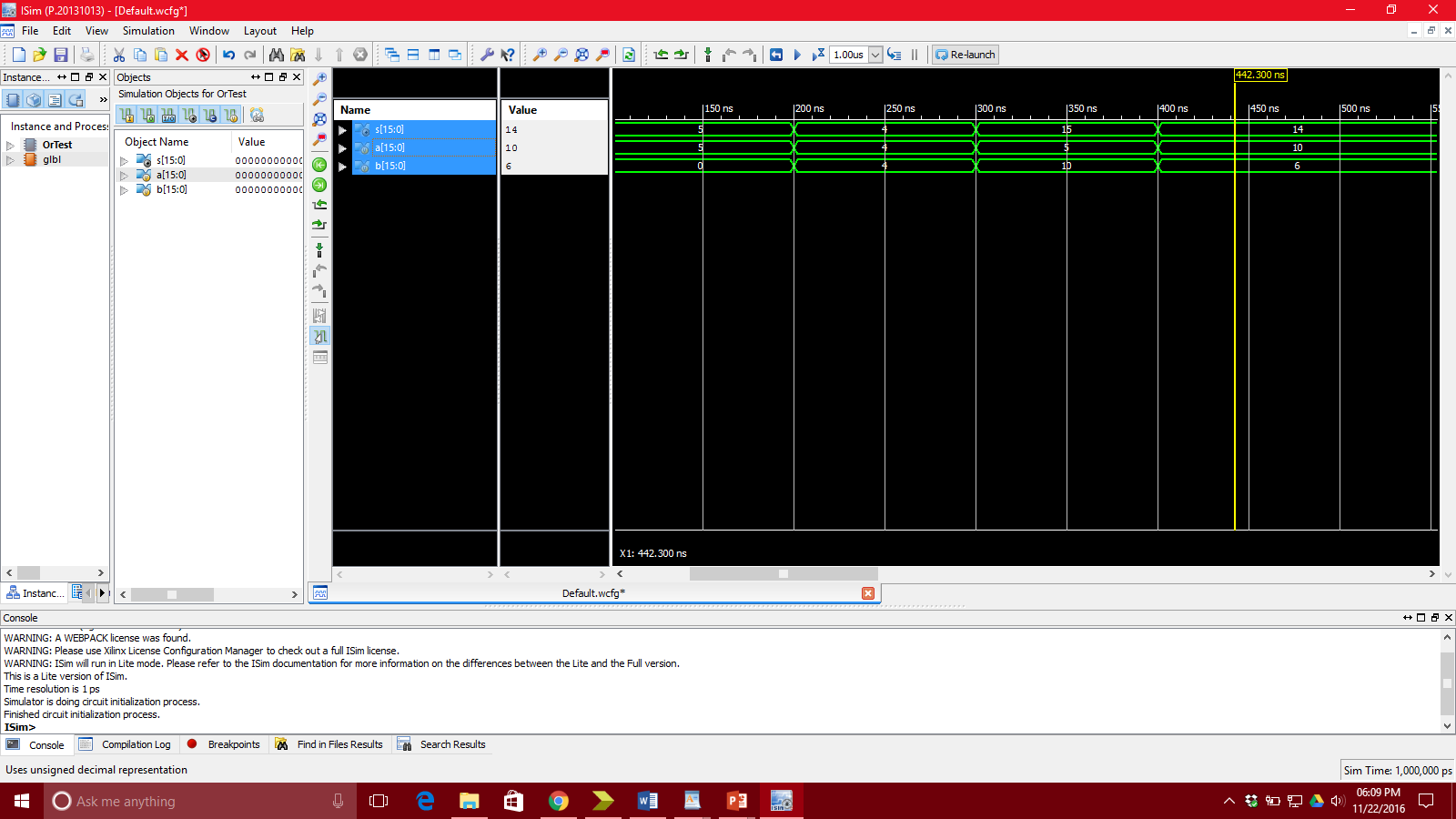
* + 1. Two’s Complement Adder/Subtractor Test Simulation (when Cin is 1, subtract, otherwise add)



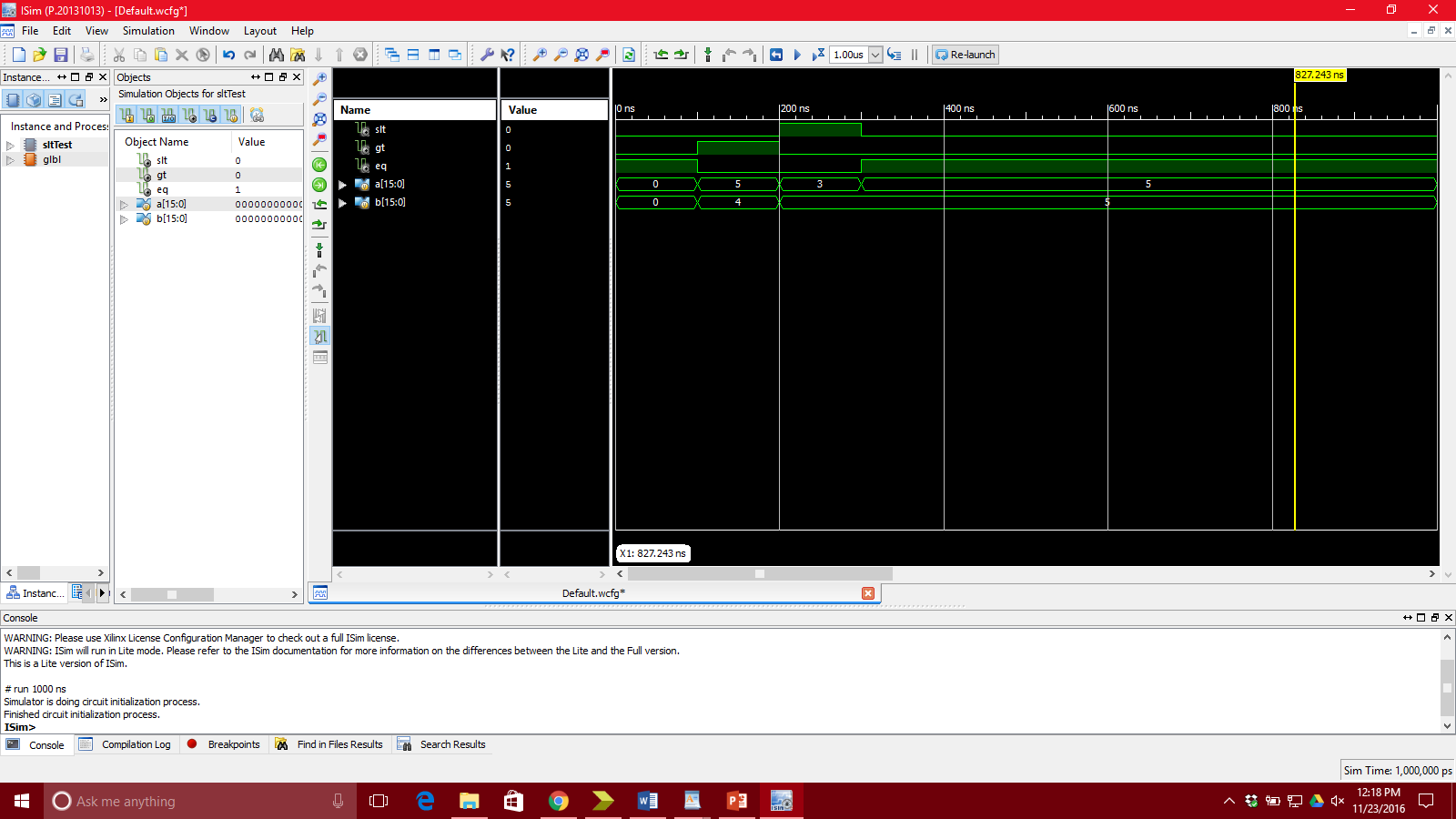
* + 1. AND 16-bit Test Simulation



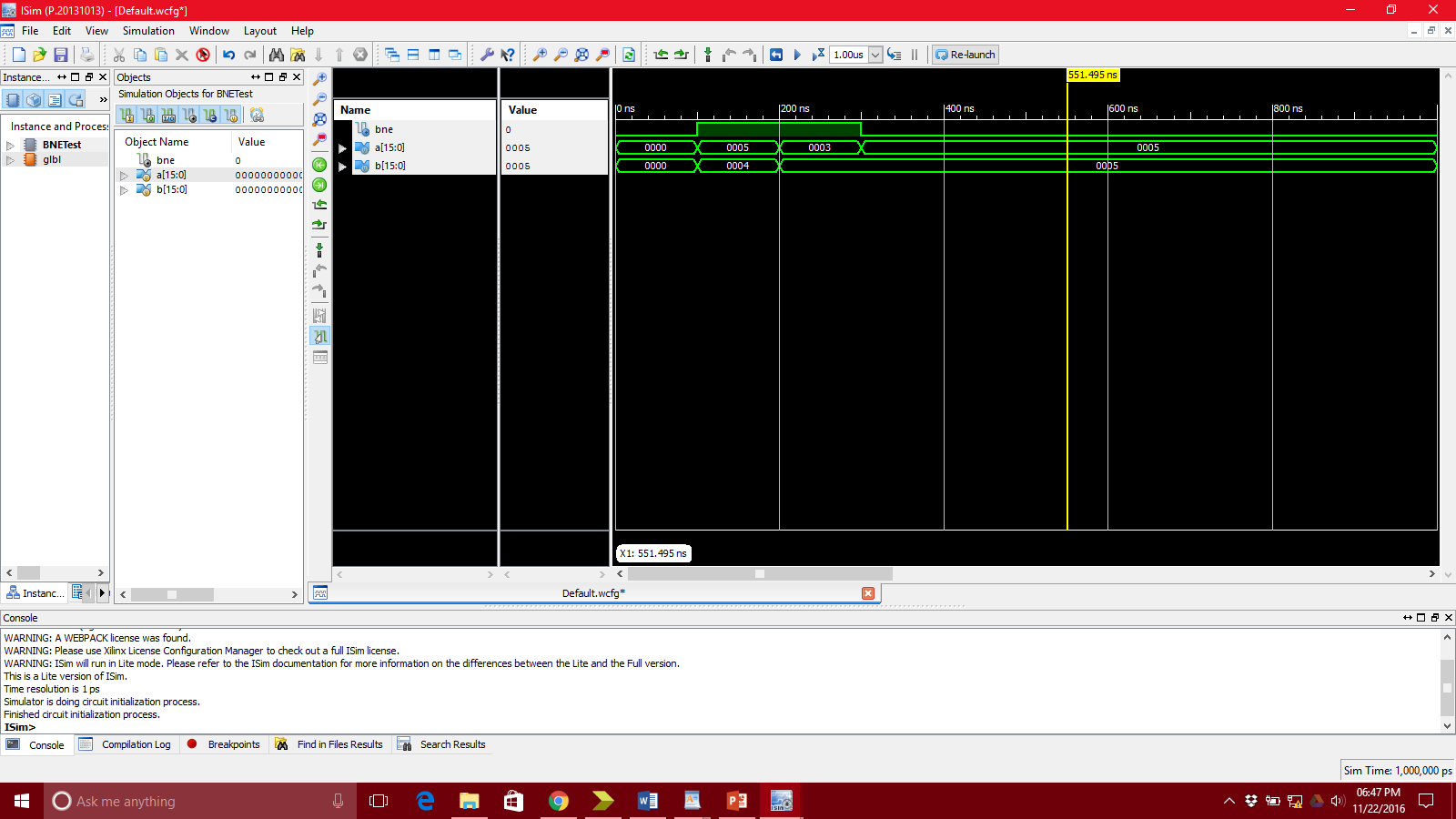
* + 1. OR 16-Bit Test Simulation



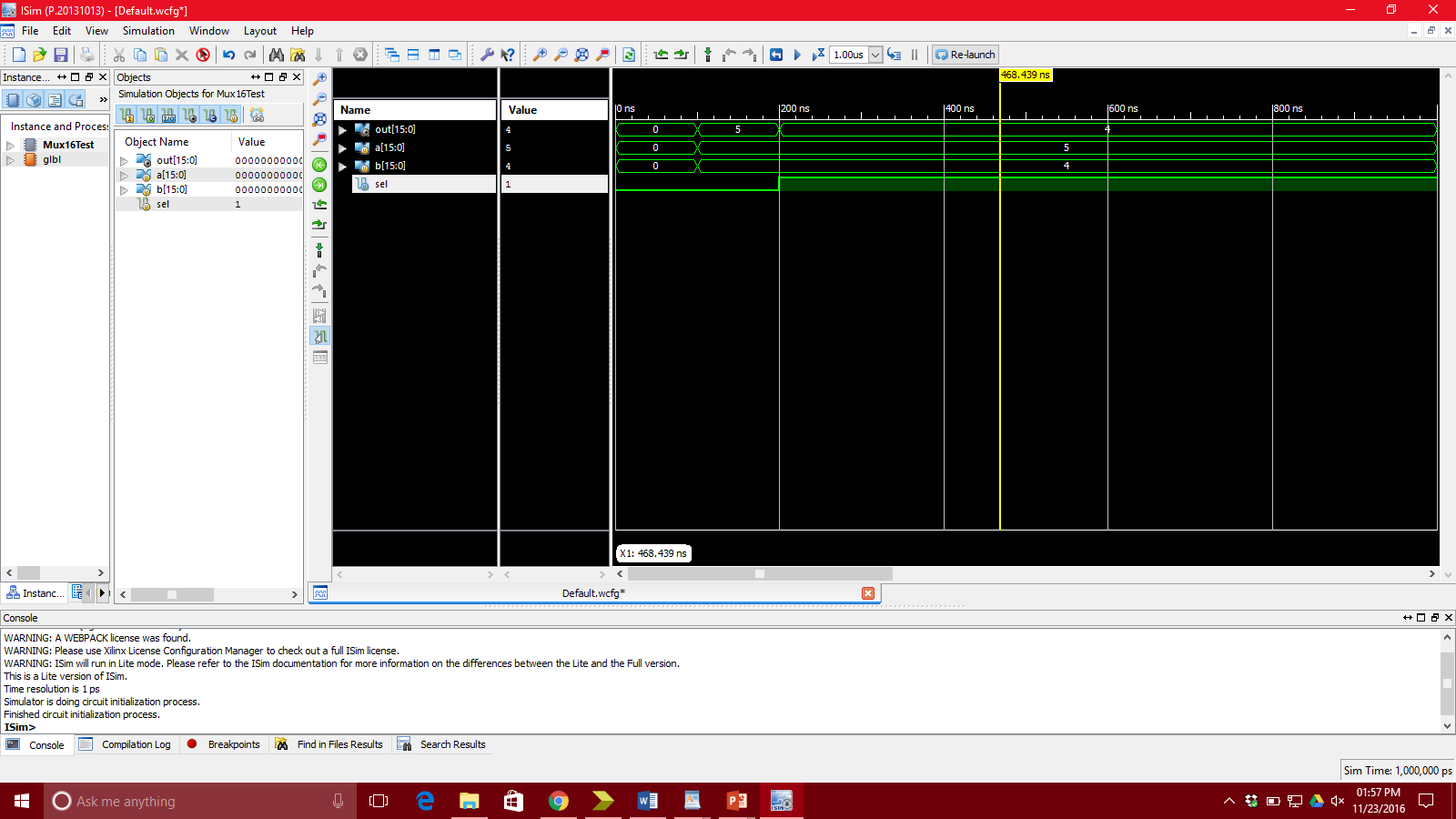
* + 1. Set On Less Than Test Simulation



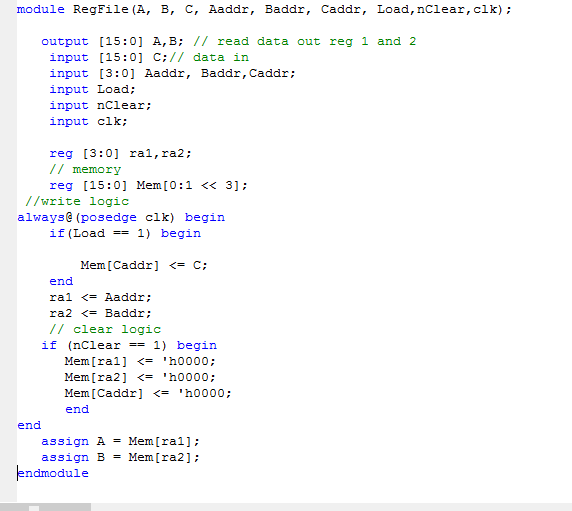
* + 1. Branch not equal Test Simulation (not used)



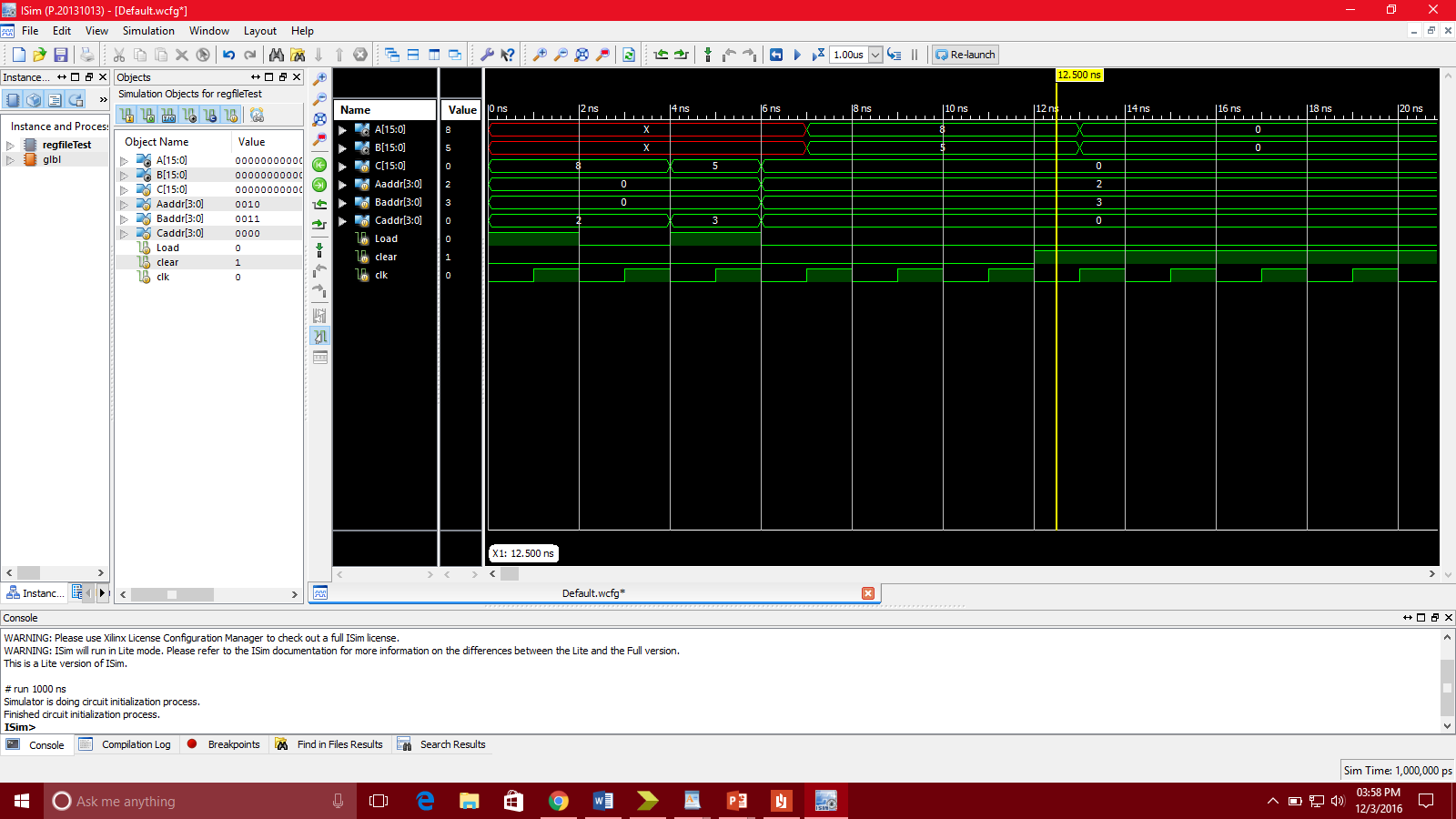
* + 1. 16-bit 2-to-1 MUX Test Simulation



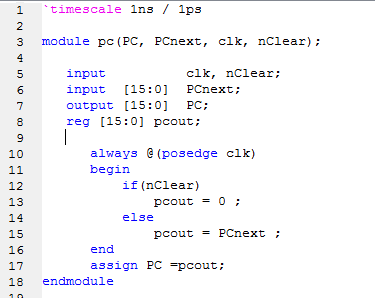
1. **Register File**
   1. RegFile.v



* 1. RegFile Test Simulation. C=8 to caddr 2 which is given to aaddr and output as A.

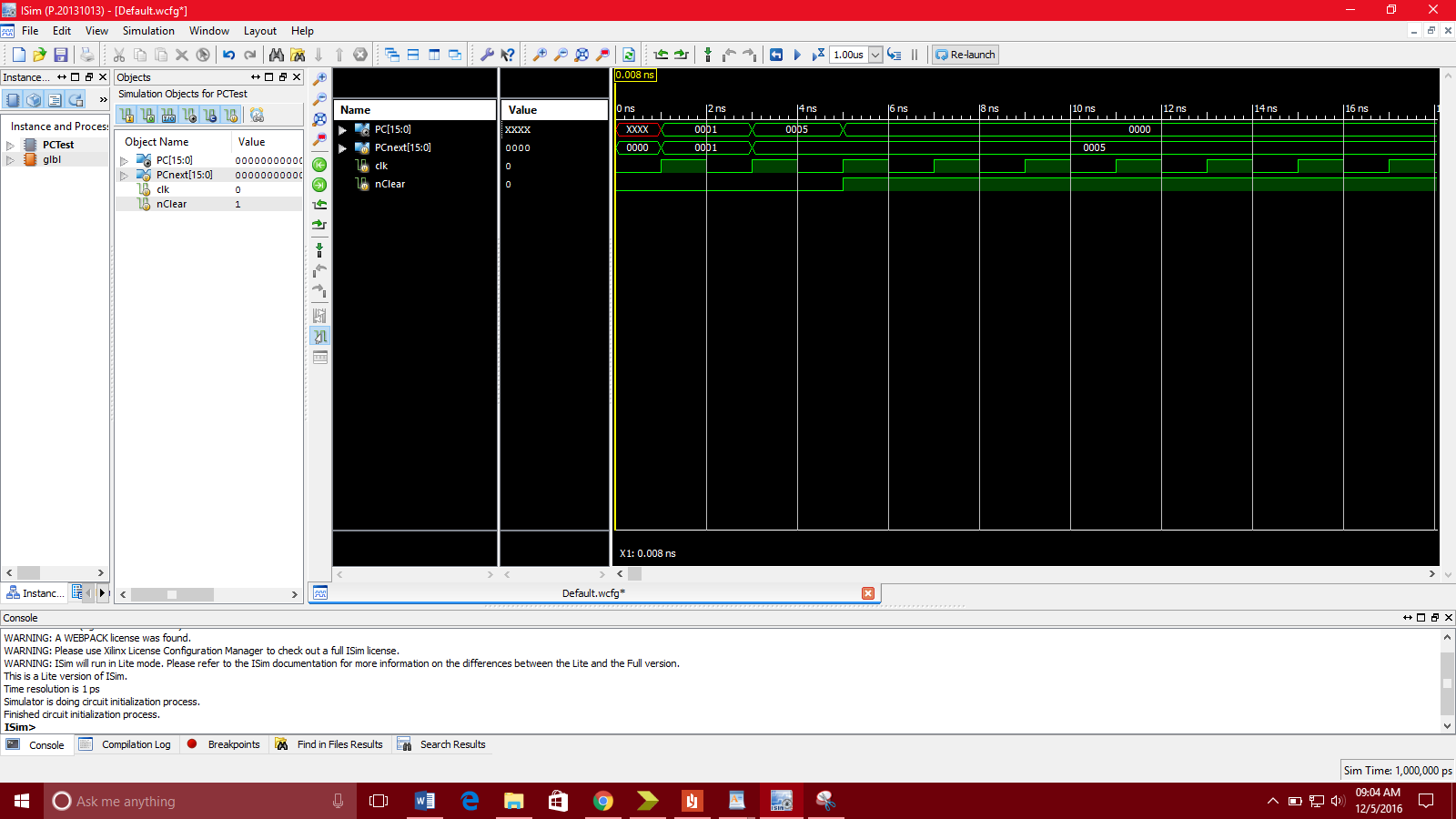


1. Program Counter (PC)
   1. PC code

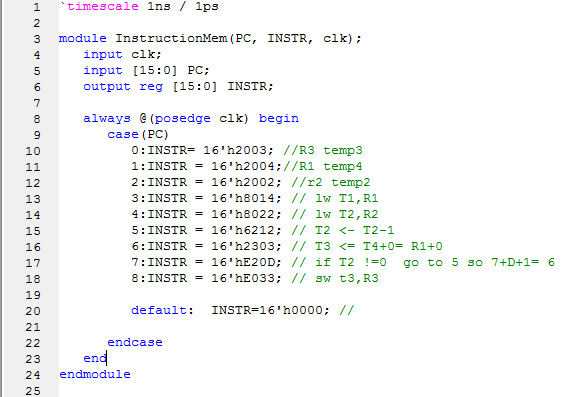


* 1. PC Test Simulation

Simple PC as PC turns into the Next PC

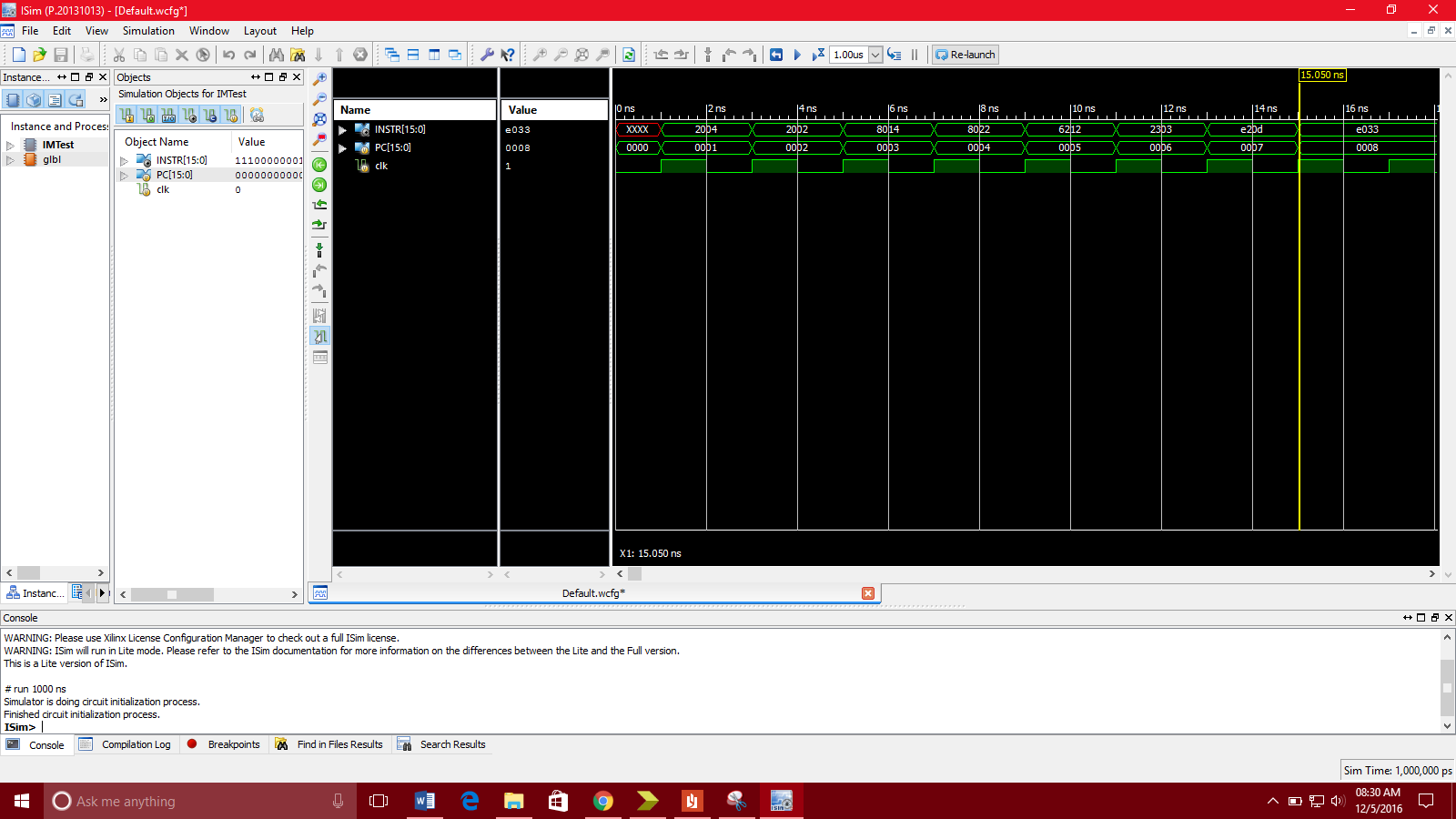


1. **Instruction Memory**
   1. Code

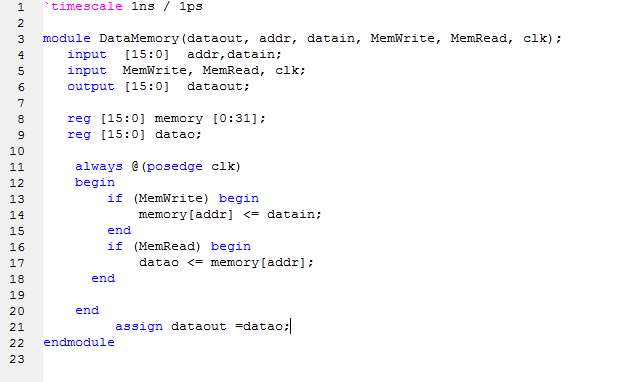


* 1. Instruction Memory Test Simulation

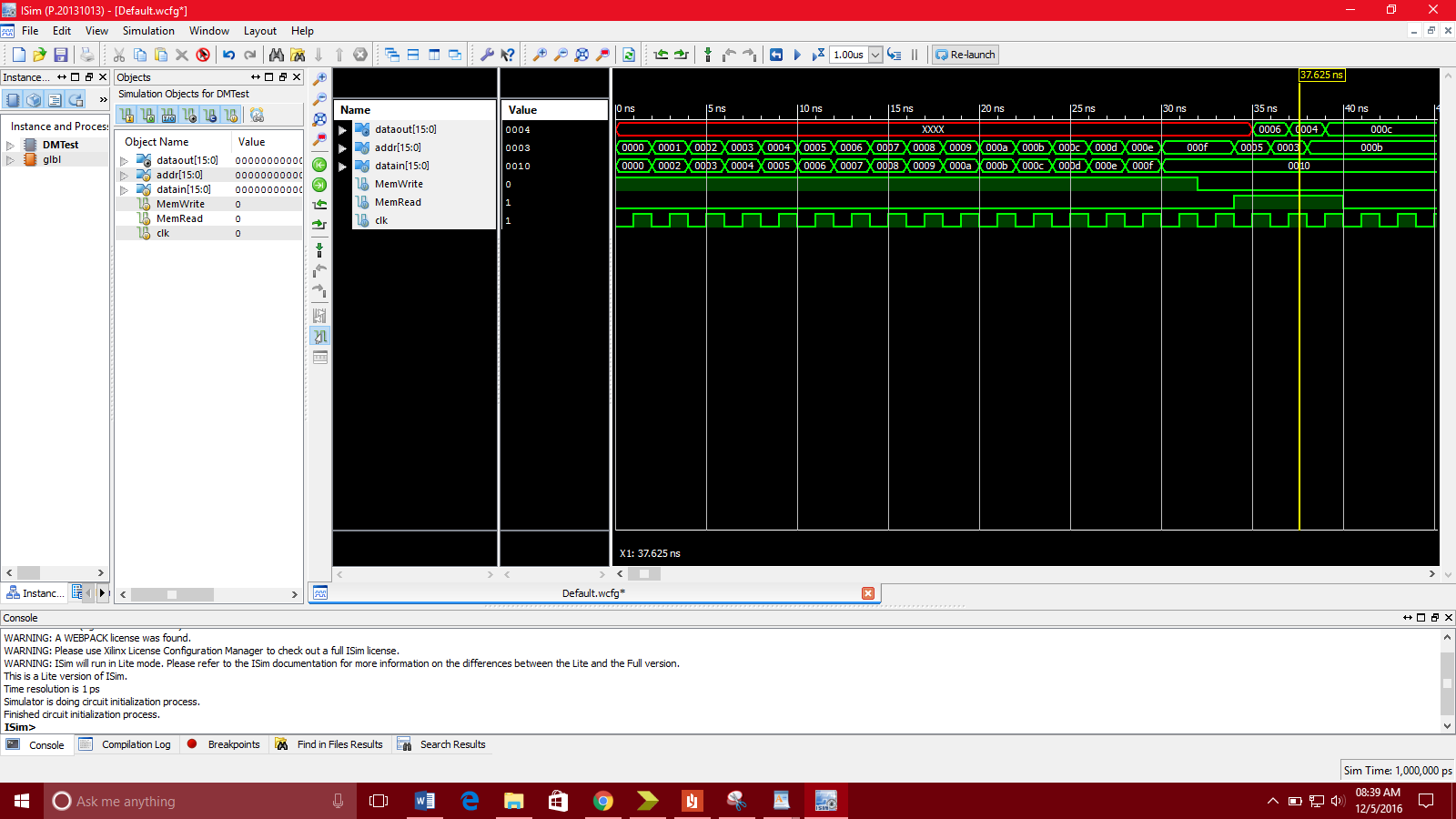
This is the program we designed, I don’t know if it runs correctly in the MIPS but it works here



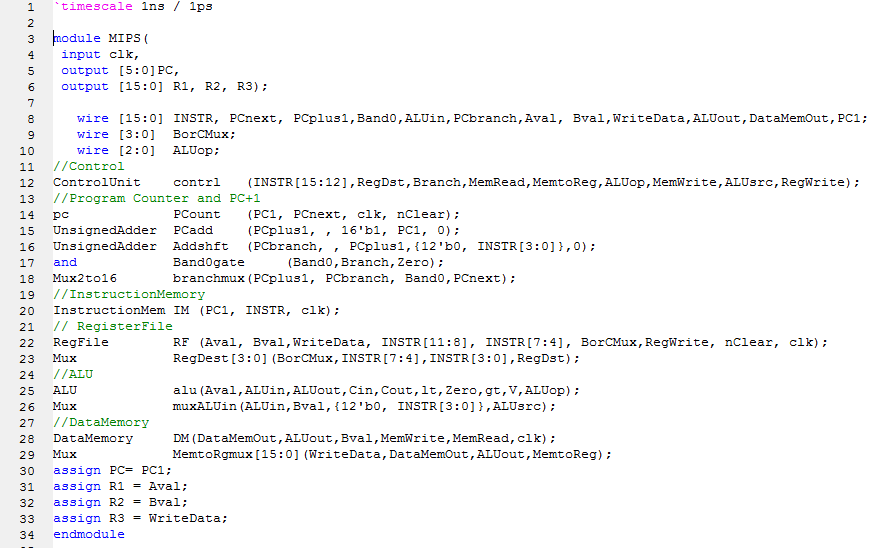
1. **Data Memory**
   1. Code



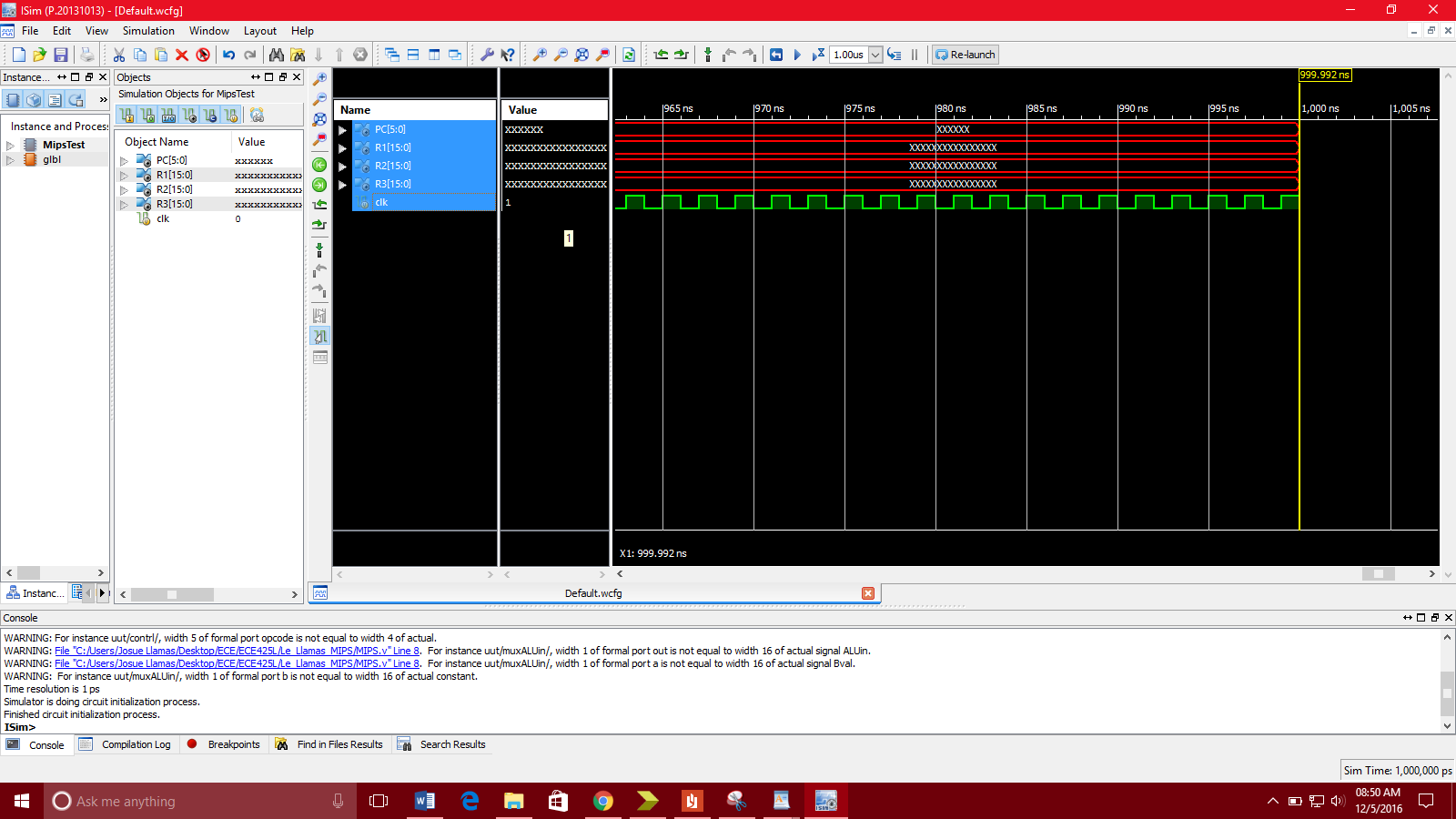
* 1. Data Memory Test Simulation



1. MIPS
   1. Code

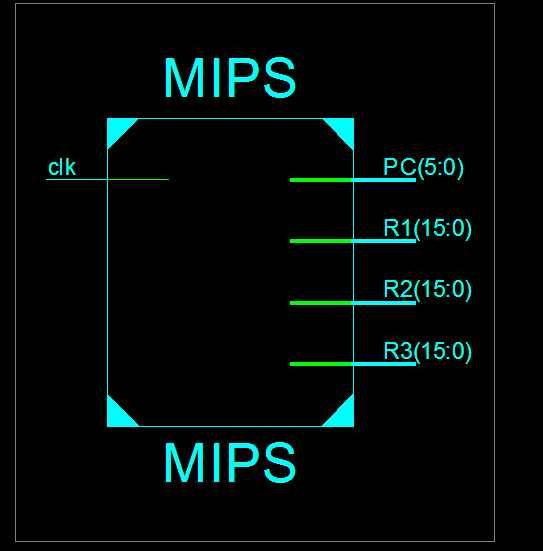


* 1. MIPS Test Simulation

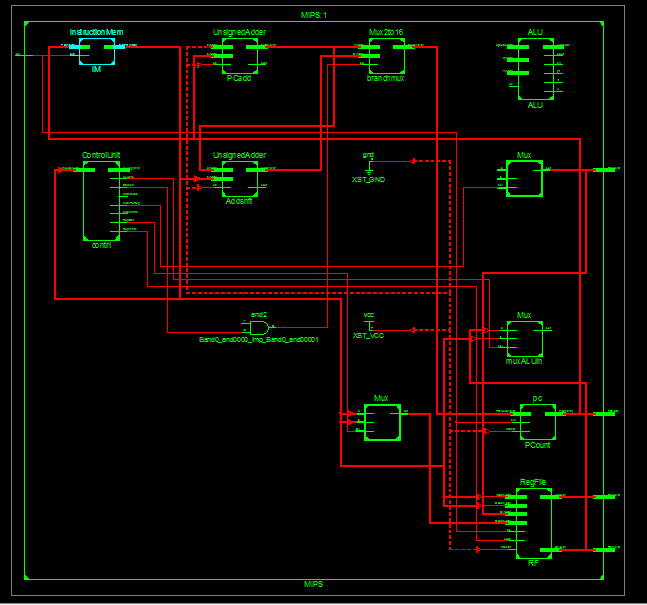


* 1. **RTL Schematic**

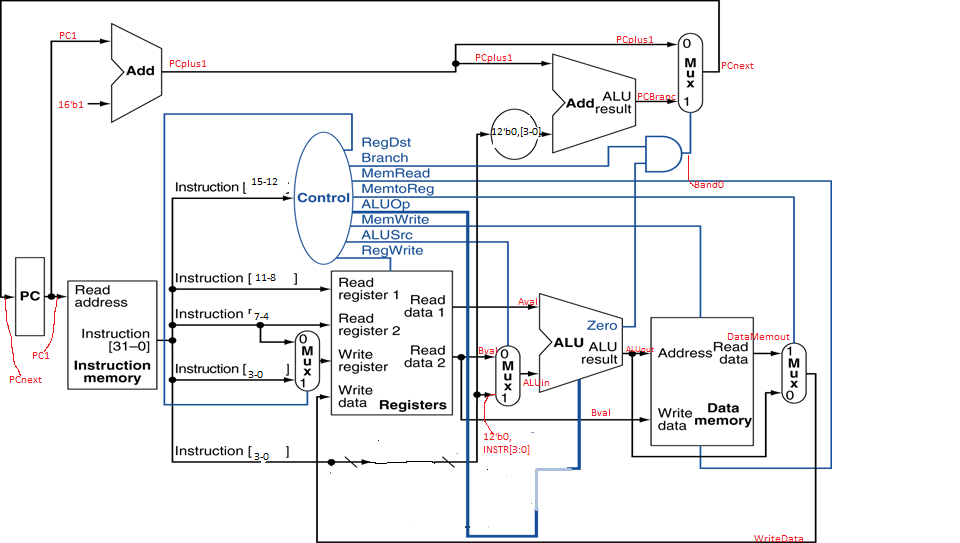
**Top Module**



**Main**



Hand Drawn Design



1. **Conclusion**

We believe most of the components in the processor are working individually, however we could not get the processor itself to output any results. From the RTL schematic generated by Xilinx, we observed that some components were not properly connected. We tried to determine the issue regarding this problem, but we’re unable to get full component integration. The ALU module, for example, is clearly not connected within the schematic. Without the ALU module, the processor cannot calculate any inputs we provide. Based on test simulations, we concluded that the ALU module itself works without any issues, but we were not able to test the desired task of solving for R3. This hindrance prevented us from completing the project. We think that if we had more time, we would have been able to solve this problem.

We are confident that most of the other modules are designed correctly based on simulation testing. For the MIPS processor, we decided to not implement a “Jump” function because it was simply not needed. We also decided to get rid of the sign extend and the shift modules as well. By doing so, it might have been caused additional problems, but we were not able to see if it would affect our processor since our priority was fixing the ALU module first. However, it was good to review Verilog for the first time in 3 years since taking ECE 205. Having used both VHDL and Verilog, we determined that the two programming languages were very similar in terms of structure, but ultimately we prefer the simplicity of Verilog in comparison to VHDL. Regardless, both programming languages are worthy to know.