CPU Project

Ding Qin

Dennis Your

**Overview**

In this project we are supposed to make the final CPU, which is combining the Controller and Datapath modules in one top module. The CPU will have inputs clock and reset, but no output because it’s a closed system. The controller will input the clock, reset, and opcode, which is the output from the Datapath module. The Datapath will take in the clock, reset, and control signals, which are the outputs from Controller module.

**Implementation**

We made a few changes in our final design. Initially we had a mux that selected between the [25-21] bits of the decoded instruction and [15-11] of the decoded immediate, which went into the n\_bit\_registerfile as a write register input. But we then realized that it would always write to the register in bits [25-21]. We could not think of an instance where it would write to the register specified in bits [15-11]. Since we didn’t need a mux there anymore we also took out the control signal RegDst.

Another change we made was adding a mux that selected between the bits [25-21] or [15-11] as inputs to the n\_bit\_registerfile as a Read Reg 1 input. Then we created a new control signal called reg\_A\_mux, which selects between which bits are used as the input for the Read Reg 1 input depending on the instruction.

Also to account for SLT and SLTI instructions, we added another value to the mux that selected the A input for the ALU. Then the ALUsrcA control signal becomes 2 bits, and the mux will take in an IMM now.

In the controller, we added an extra state that did sign extend for I type instructions.

We also separated the Data memory and Instruction memory. We found it easier to operate this way. The instruction memory then takes in an address input, which is the Program Counter’s current value and outputs the concurrent instruction. The Data Memory takes in an address, which is the value of the immediate that was decoded in the Instruction register. It also takes in a write data input, which is the value in register A. Then it outputs Mem Data to the Memory data register.

The final change we made to our system was taking out the ALU control. Since we decoded the ALUop in the controller already, we just directly sent the ALUop control signal to the ALU.

To test the entire system, we took the instructions given to us in the CPU\_tb file and copied it over to the instruction memory module. This is where the value of the Program Counter selects the appropriate instruction to use.

**Extra Credit**

We also incorporated extra credit instructions of Load Word and Store Word. We had to add a mux before the Data memory in order to decide between the immediate or ALUout[15:0] or ALU reg out[15:0]. We also added another control signal for this mux called Load. This was a 2 bit control signal.

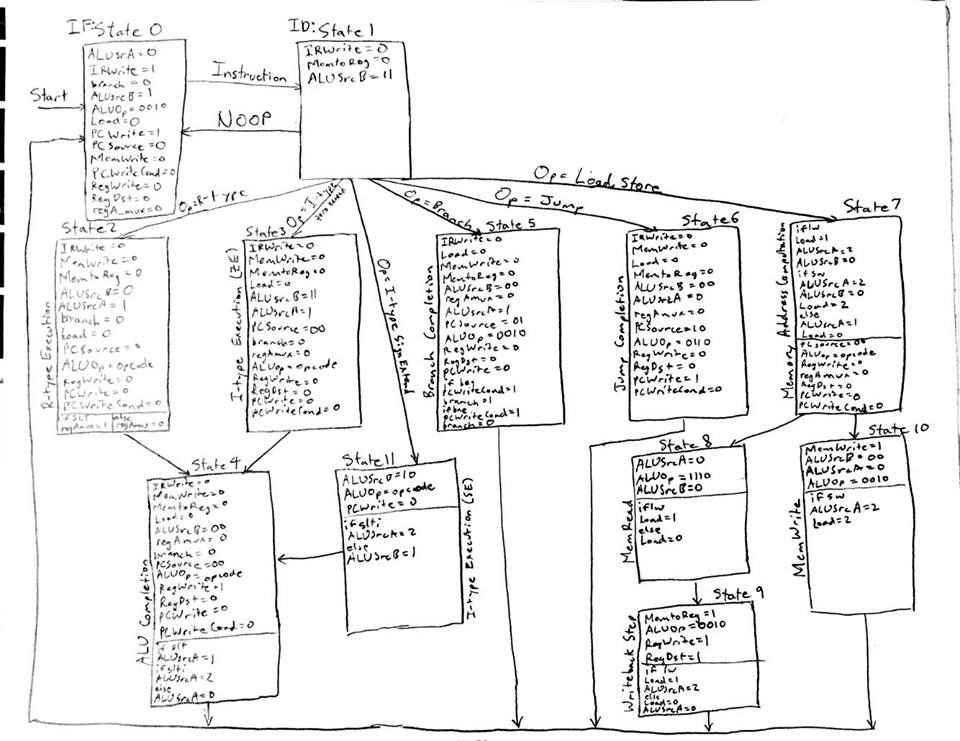
For Load word we used the instruction 111010\_10011\_10000\_0000000000000111.

This will load the value in regsiter 16+offset into register 19.

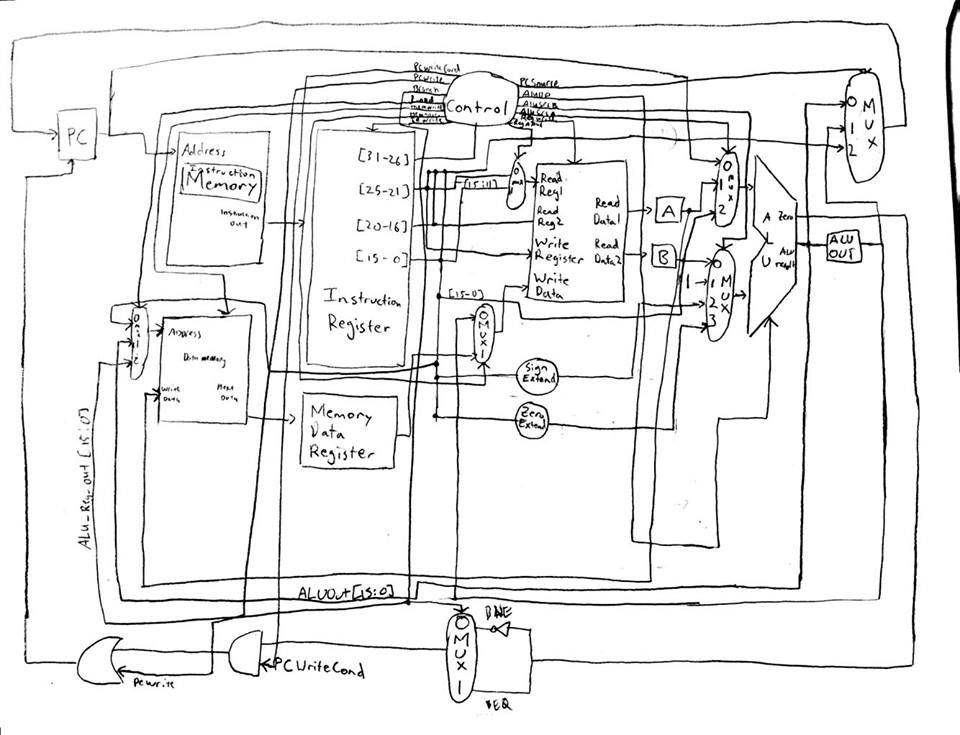
For store word we used the instruction 110001\_00100\_00001\_0000000000000010. This will store the value in register 4 into the address of the value in register 1 + offset 2.

**Diagrams and Waveforms**

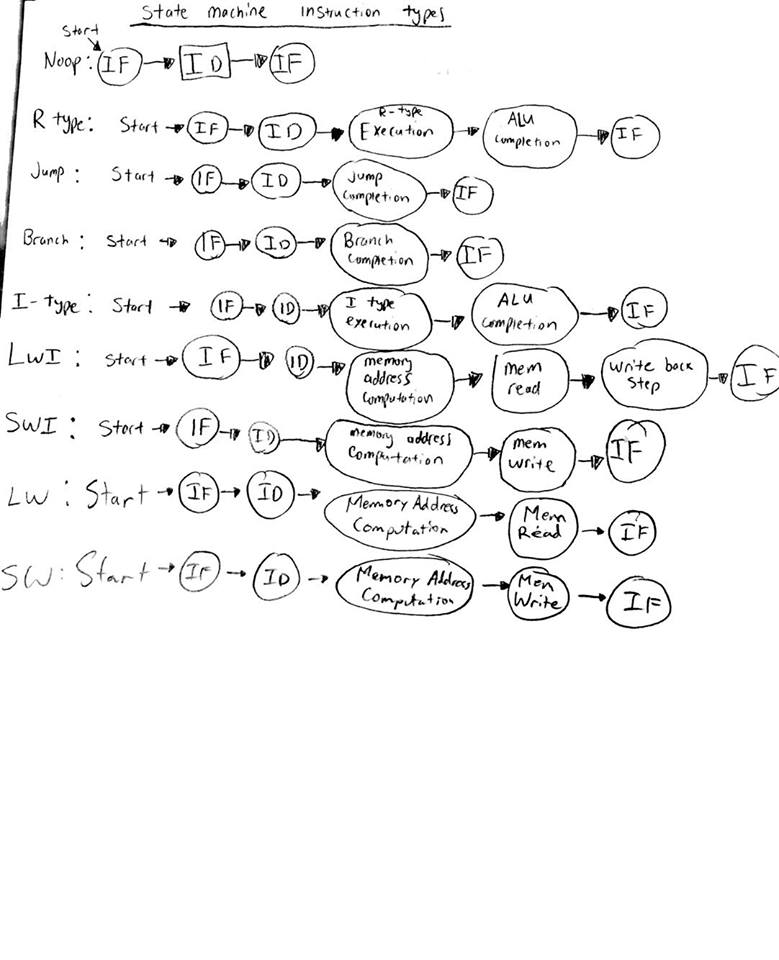
Below is the picture of our state diagram



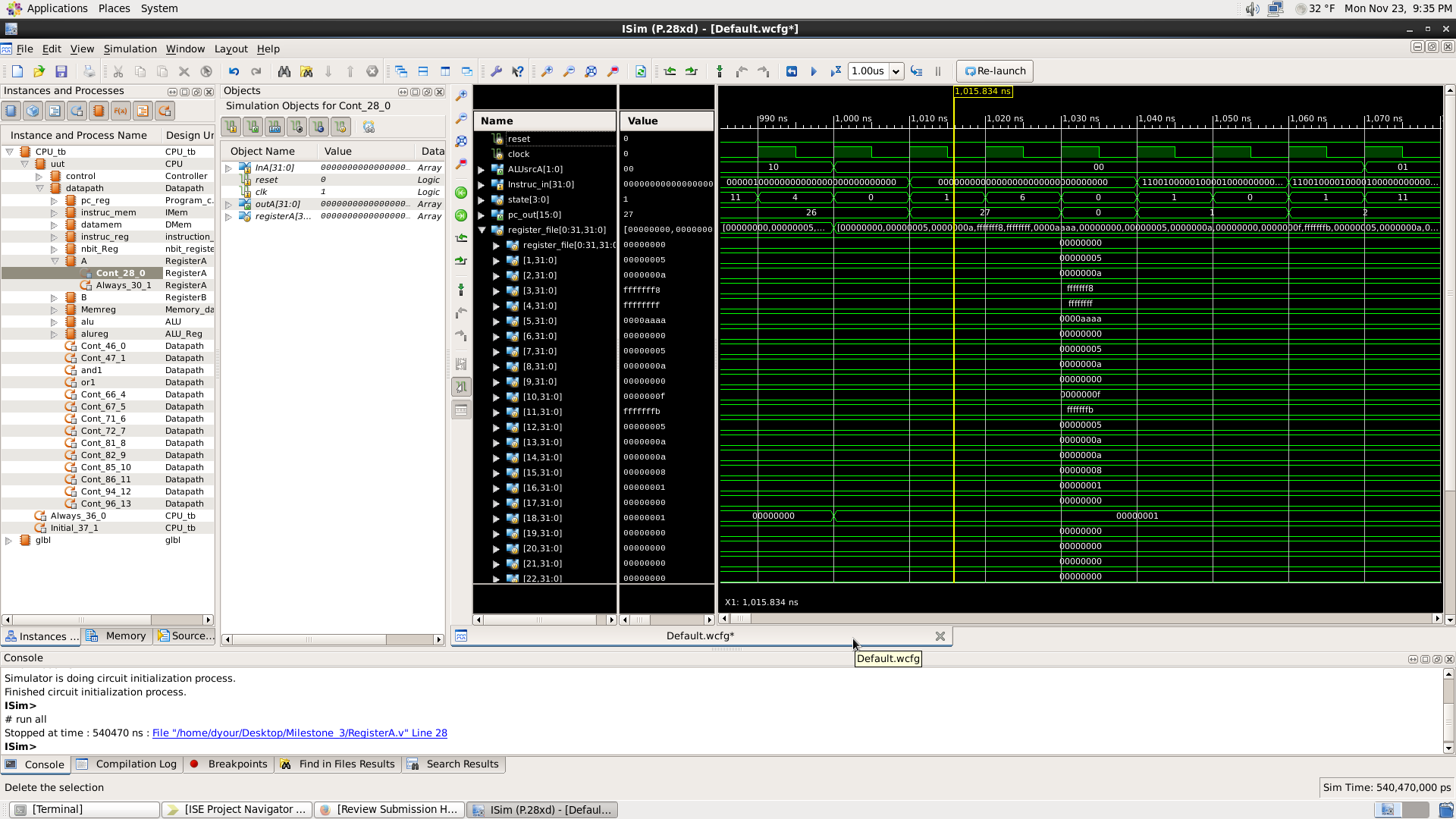
Below is a picture of the CPU datapath

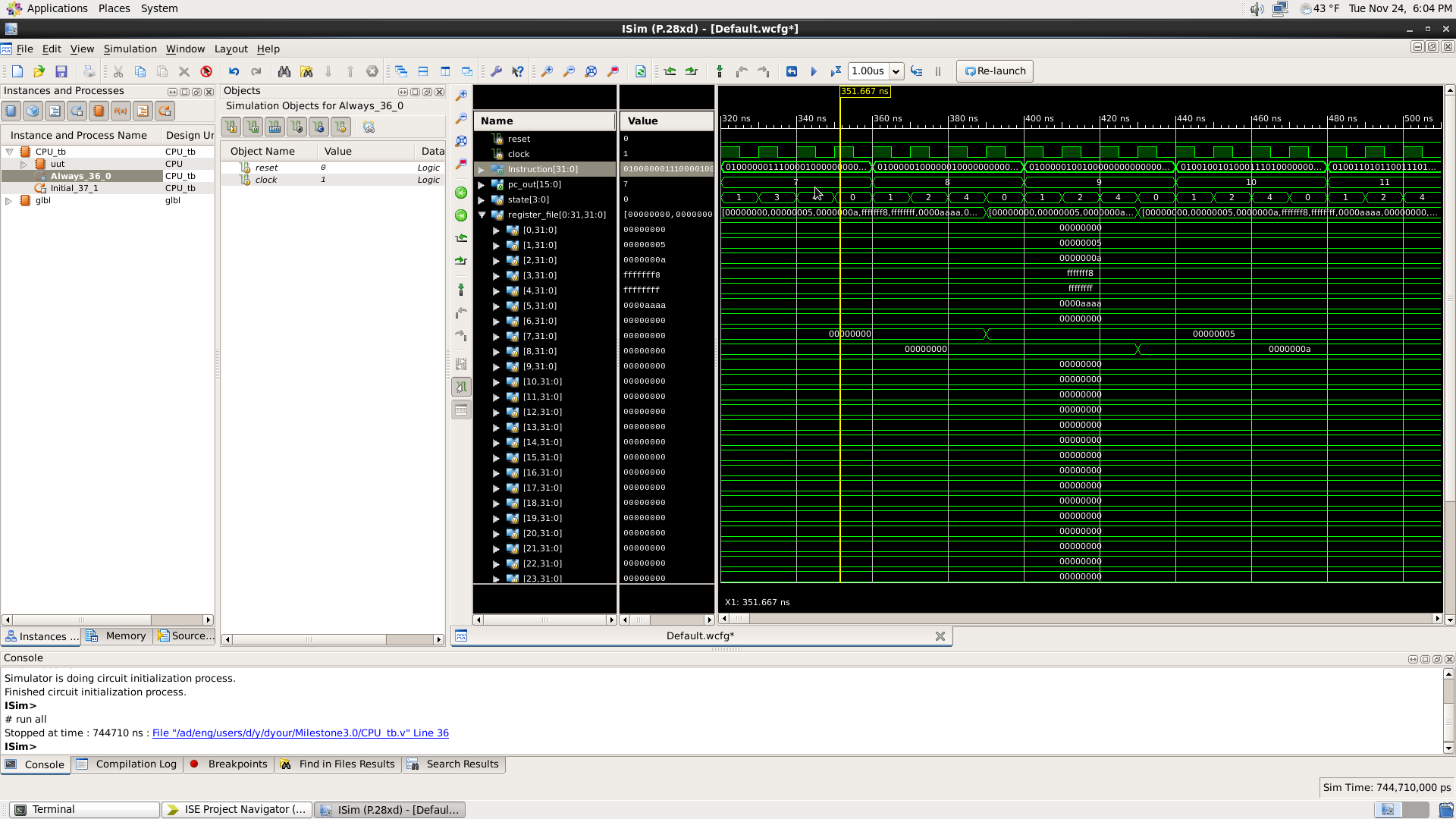


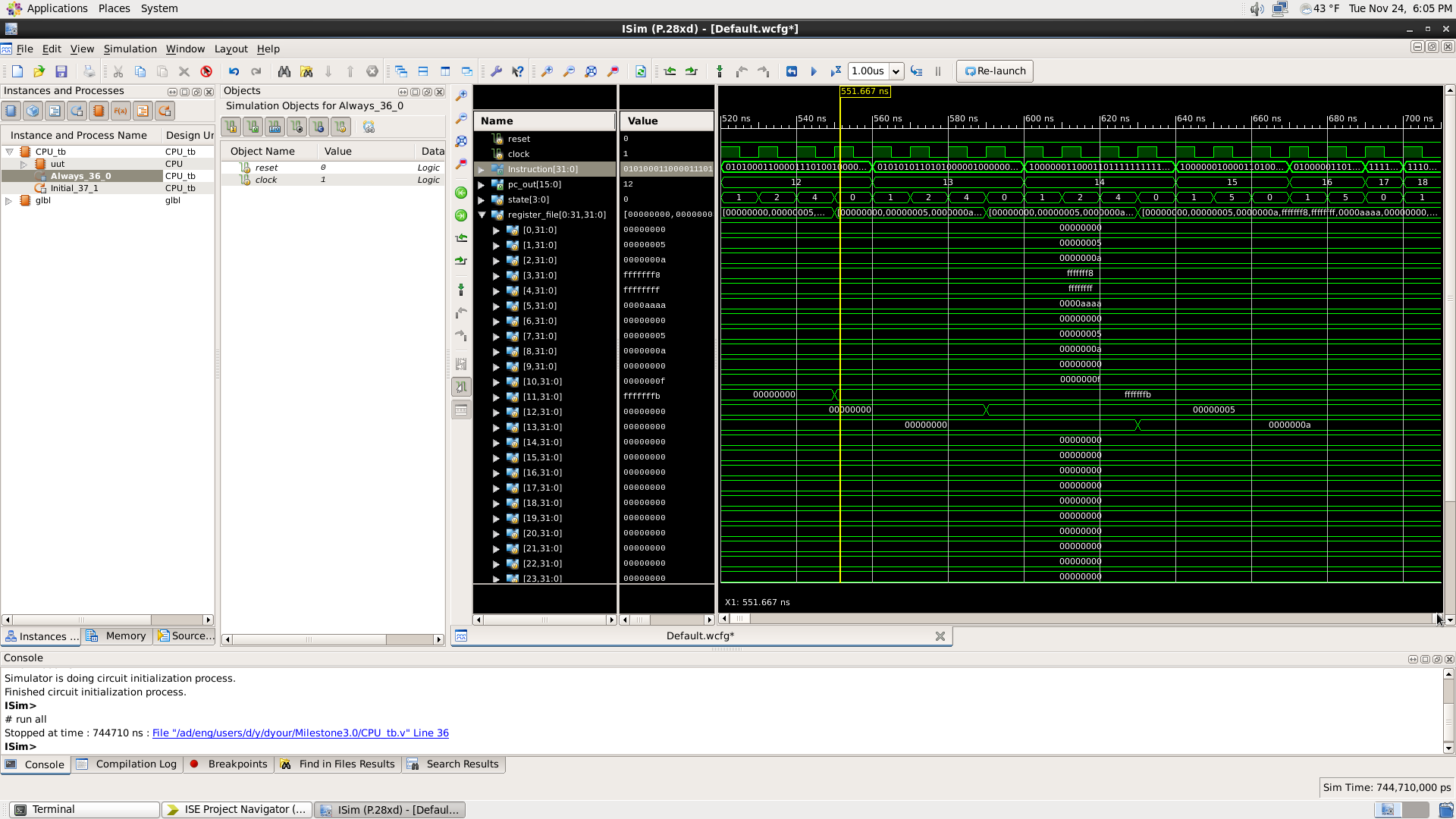
Below is a picture of our state machine instruction types

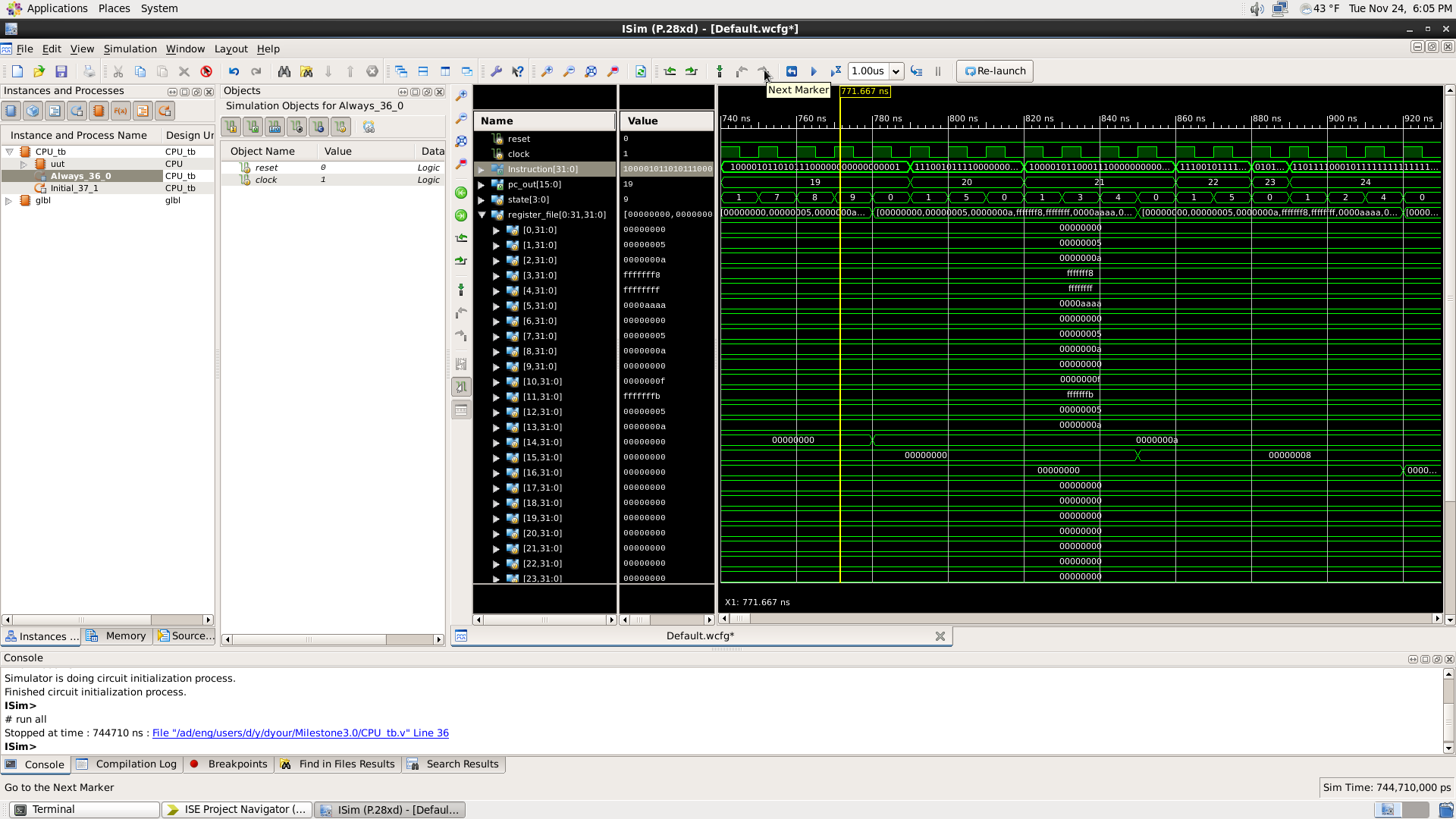


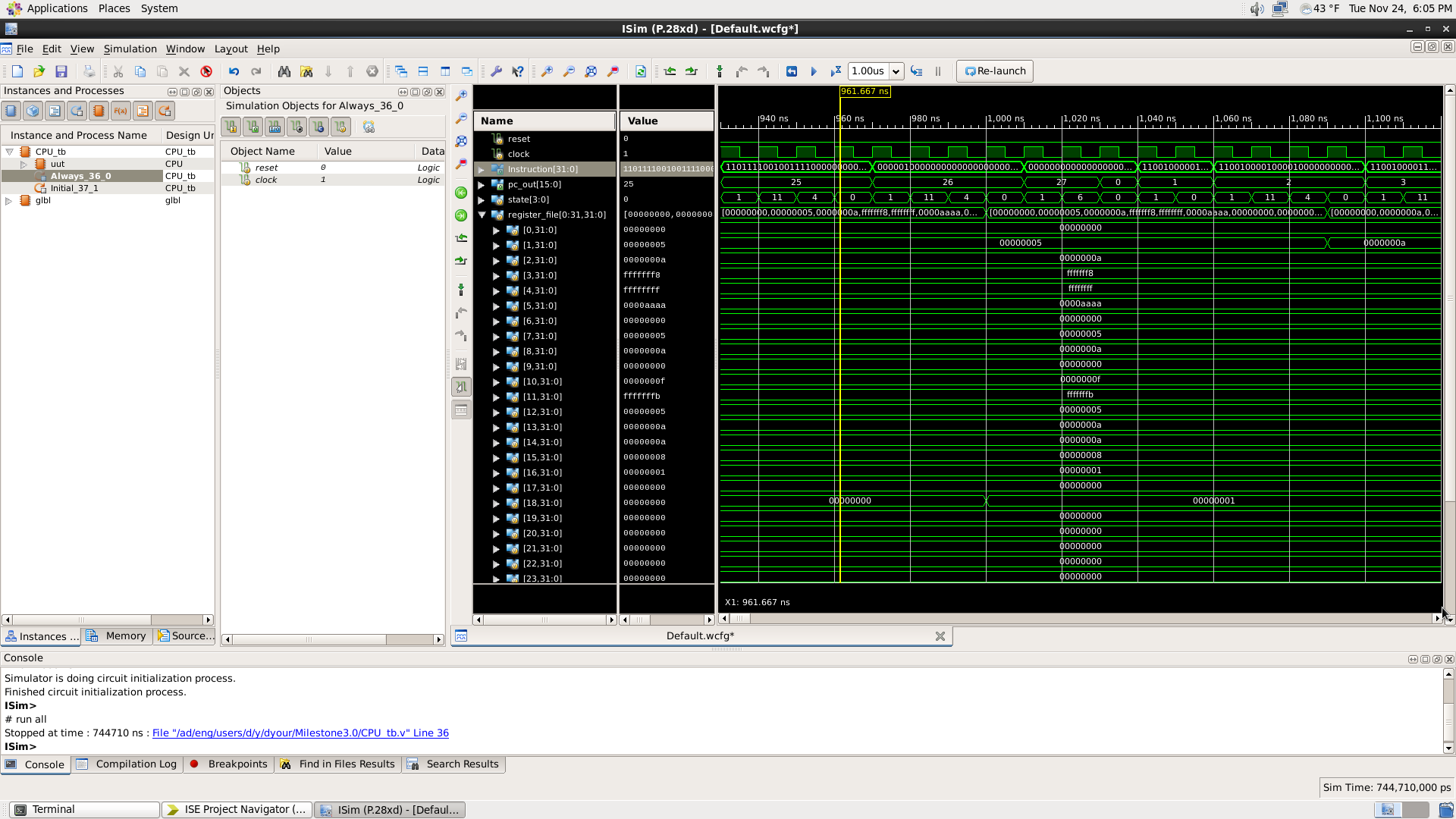
The rest of the screen shots will be the waveforms. These are based off the given CPU\_tb instructions.

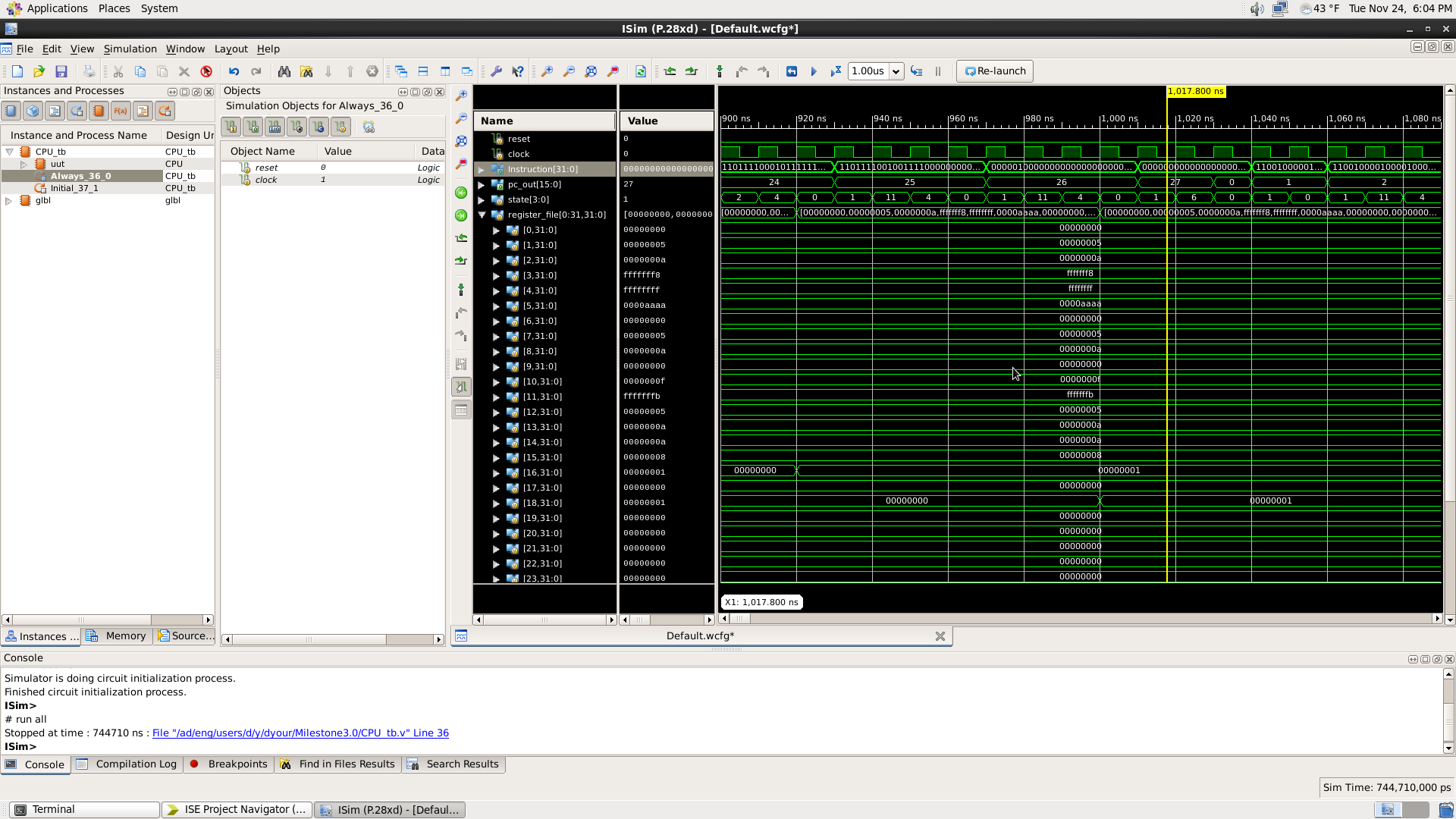




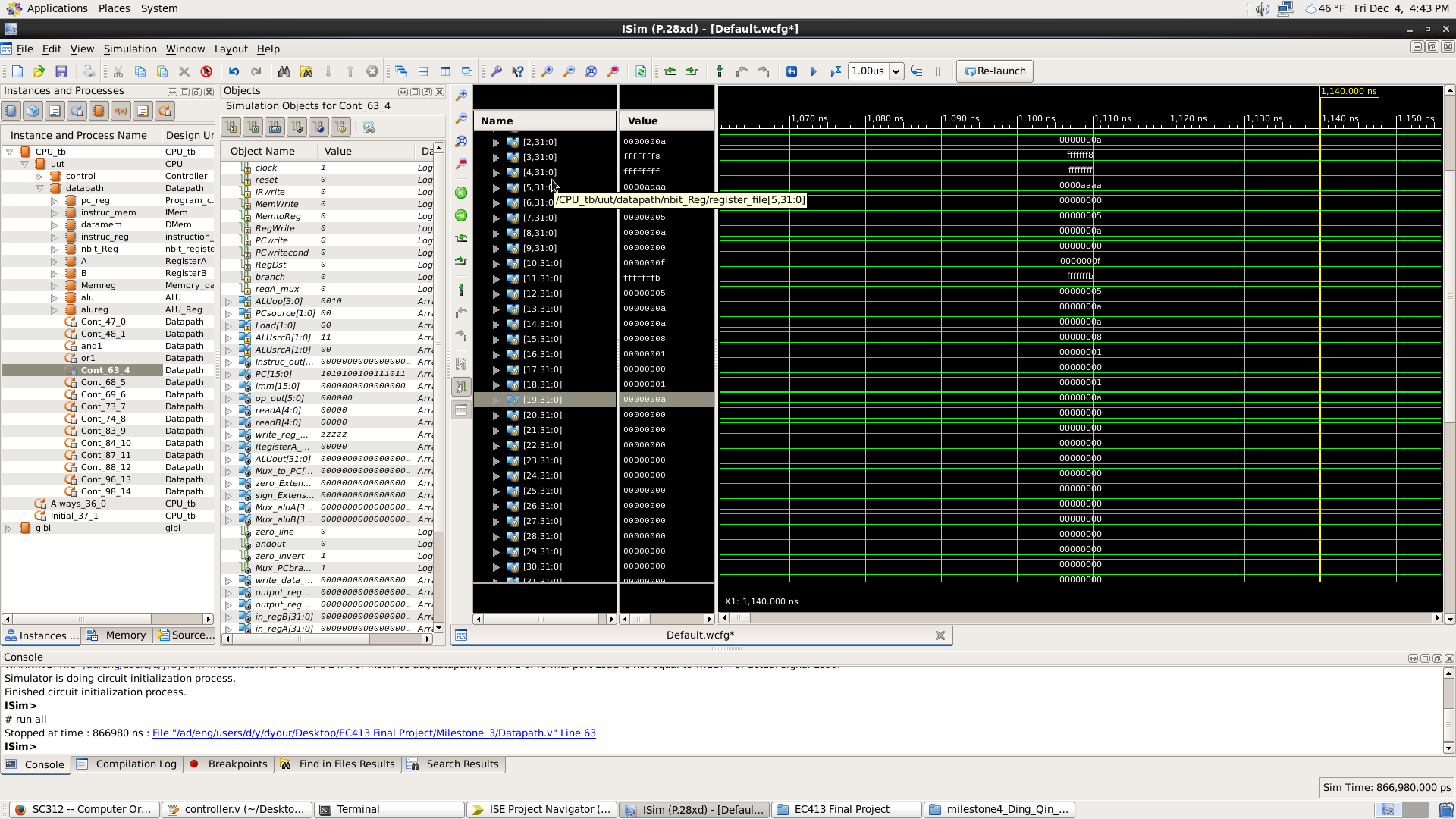








Extra Credit Load word waveform



Extra credit Store Word waveform

