Questions:

1. a. PCLd is high

b. IRLd is high

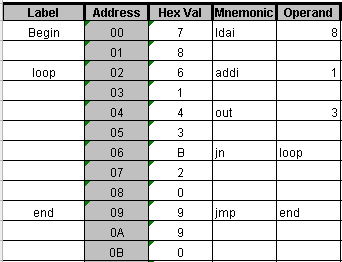
c. ACCLd is low

1. MARLoLd, PCLd,R\_W, and MemSel\_L. I/O execute should be the next state.
2. IR, ALessZero, and AEqZero are the signals sent from the datapath to the controller.

4) The ADDI function adds the next value to the current value of the accumulator meaning that during the operational cycle, the value of the accumulator will change. In order to load the correct value from the accumulator, the ACCLd has to be high.

5) There are currently 16 instructions meaning that they can be represented with four bits. Adding another instruction, the SUBI function, would require you to either make the instructions a five bit number or remove an old functionality. Making the function is not difficult because subtracting is simply adding a negative number and PRISM already has a negative function and an adding function. You would simply need to combine the two functions.

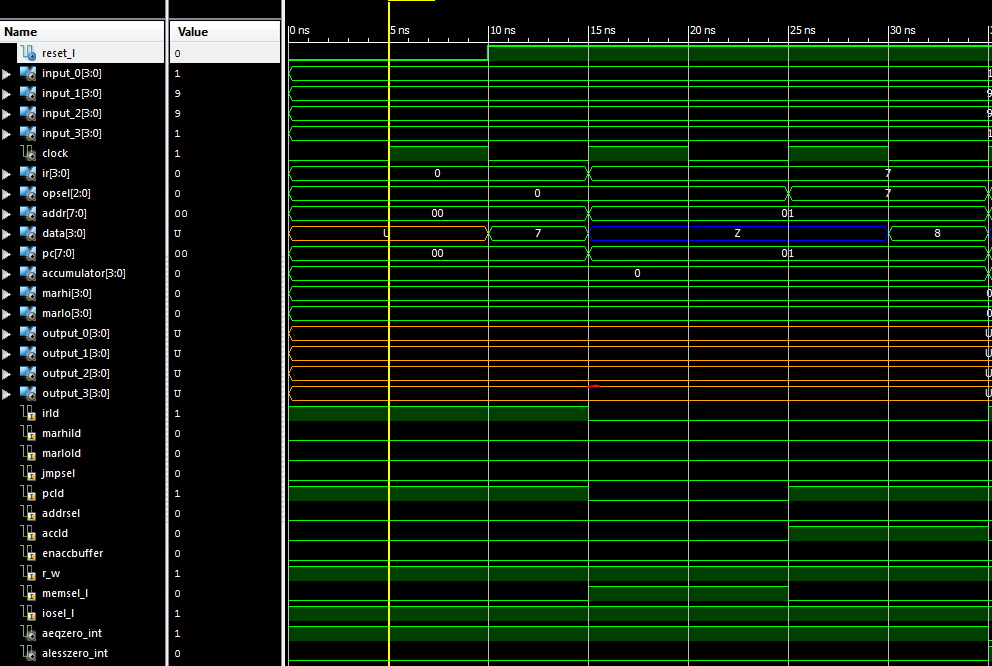
First Program Operation:



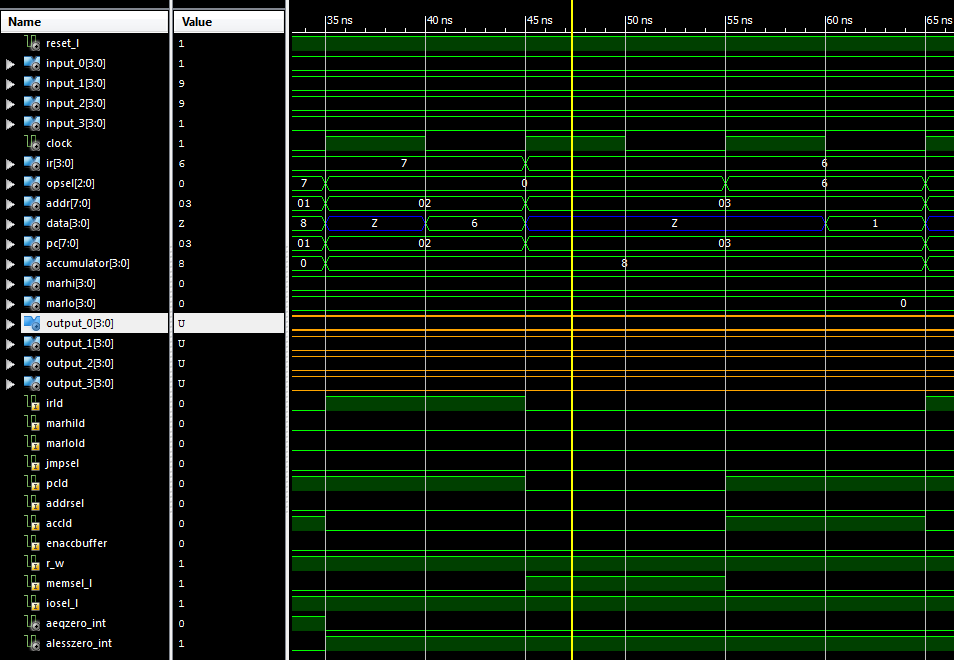
The first program we worked on counts in hex. However, the program initiates by loading an 8 into the accumulator. Once an 8 is loaded, the program outputs the current number in the accumulator to output port 3. The program then checks the value of the accumulator for a negative number. If the value is negative number it adds 1 and outputs to port 3. Once a positive number is detected the program enters an infinite loop where 0 is outputted to port 3. The program remains here until the user resets the program allowing it to start from a loaded value of 8 again.

First Program Instruction Cycle:

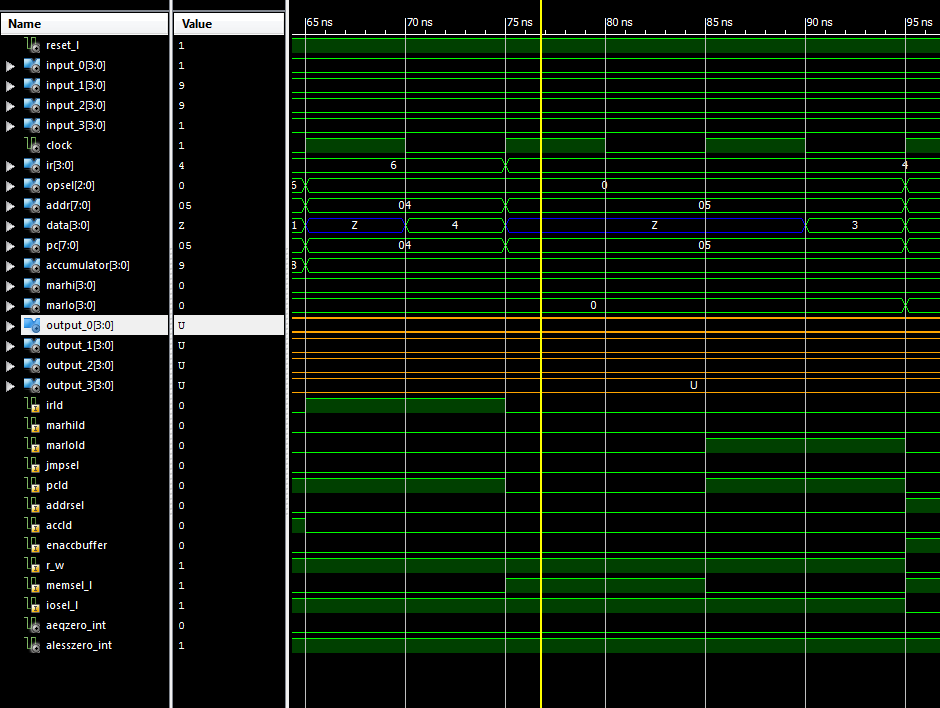
First Function – LADI: Loads a value of 8 into the accumulator. (5-15ns: Fetch), (15-25ns: Decode), (25-35ns: Execute)



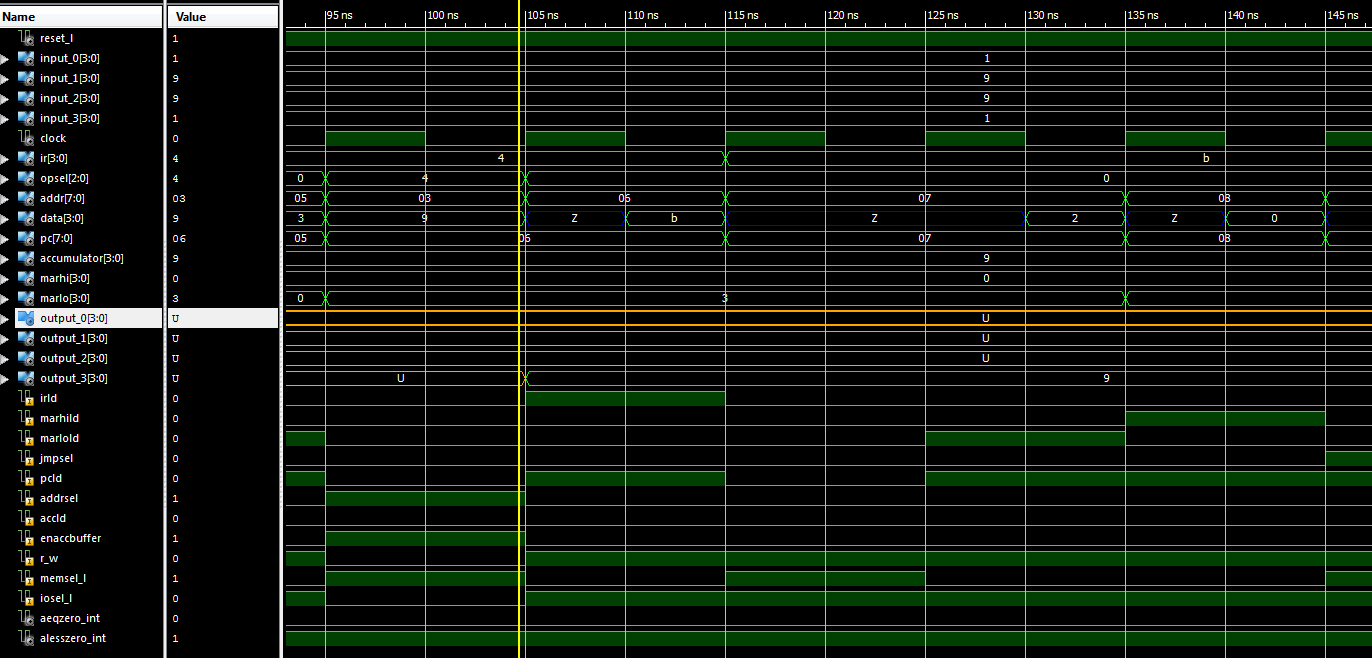
Second Function– ADDI: Adds a value of 1 to the accumulator. (35-45ns: Fetch), (45-55ns: Decode), (55-65ns: Execute)



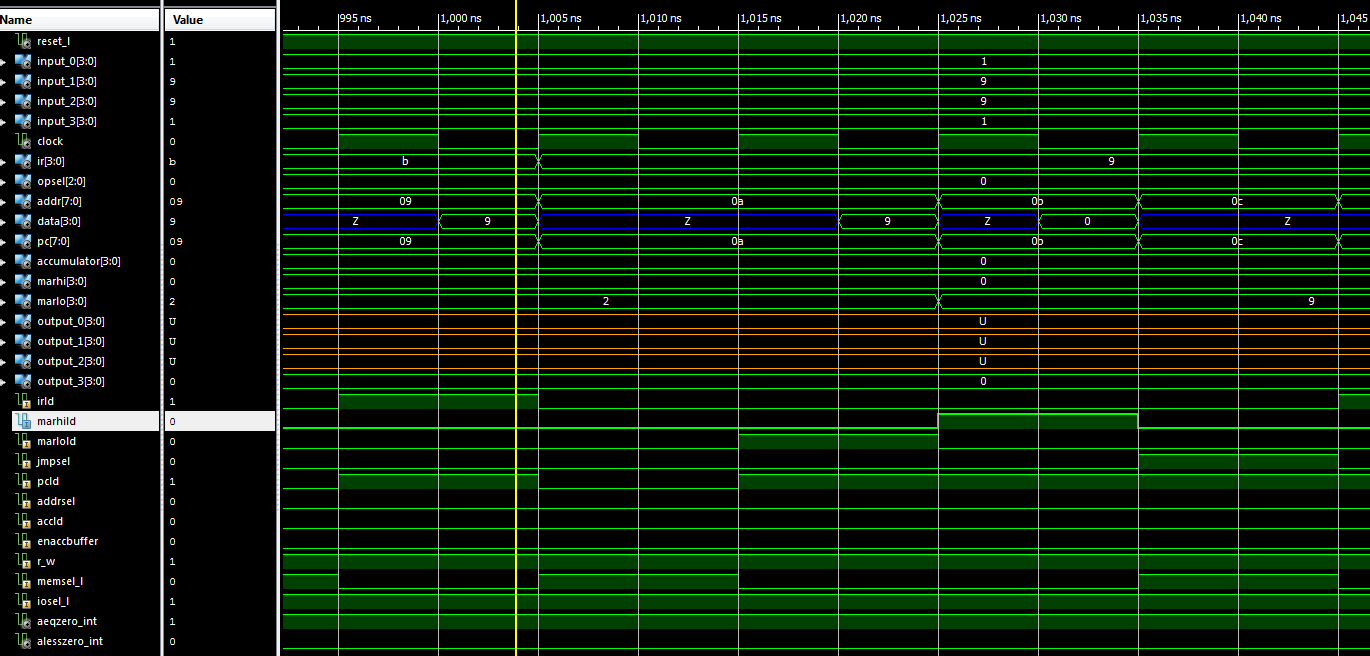
Third Function – OUT: Outputs the accumulator value to port 3. (65-75ns: Fetch), (75-85ns: Decode), (85-95ns: Execute)



Fourth Function – JN: Jumps to ADDI when the accumulator value is negative. (95-105ns: Fetch), (105-115ns: Decode), (115-125ns: Decode LoAddr), (125-135ns: Decode HiAdder), (135-145ns: Execute)



Fifth Function – JMP: Jumps to itself to create an infinite loop. (995-1005ns: Fetch), (1005-1015ns: Decode), (1015-1025ns: Decode LoAddr), (1025-1035ns: Decode HiAdder), (1035-1045ns: Execute)



First Program Errors and Debugging:

The first program went rather well. I ran into few errors and only got stuck in one part. The one error was associated with the four output to nibble lines. Once I realized where the error was, I was able to correct the error by following the instructions in the lab instructions. After correcting this error my program executed correctly and produced the correct results. I checked my results with the expectations in the lab instructions and by checking with my limited knowledge of prism. Captain Tremble checked and approved my functionality supporting my belief that my program was correct.

Second Program Errors and Debugging:

The second program was troublesome for me. I was not able to get the functionality done in class. However, after researching I became more familiar with the JZ function. This helped me execute my program correctly. At first, I could not figure out how to create a loop that only counts from 0 to 9 and from 9 to 0 where the values from A to F were not included. I tried several things but all of them failed. Once I found the jump when zero function I realized that I could check each value by adding 6 to see if it equals 0. When a value equaled 0 by adding 6, I knew that it was time to reset the output to 0. The JZ function would jump when the value + 6 equaled 0 where I would update the counter output to 0, if I was counting up, or 9, if I was counting down. Counting down was easier because I did not have to modify the data to use the JZ function but would simply reset to 9 when the counter equaled 0. To add and subtract, I stored two values in the RAM. “ADD” equaled one. I would add the “Add” value whenever I needed to increment. I stored a second value in the RAM under the name “Subtract” with the two’s complement of 1. I would add this value whenever I needed to decrement. Once I understood the JZ function, I understood how to accomplish this functionality. I did not complete the functionality on time but I did get the functionality to work correctly after additional effort and time.

Functionality:

A Functionality: Checked by Captain Tremble

B Functionality: Unchecked, but correct

B Functionality (Cool Counter): Unchecked. Not that cool but functional