Phase 1 Summary

Names: Abbas Paryavi & Allen Onyegbado

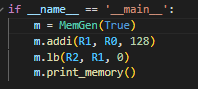
Date: 22 Nov 2022

For this project, we both worked on it together so our individual contributions are pretty much the same. We had one person, in this case, Allen since he had Vivado on his computer, to share his screen. We then worked on figuring out the Verilog codes together. Initially, however, we split the Verilog codes amongst ourselves, and each person copied some of the code available to us on the lecture slides. This took away a lot of our time but we were able to finish this during lab hours. We put in the most hours the Monday before it was due. We each put our entire day working on it on zoom. We also put a fair amount of the Tuesday that this phase of the project is due to get the rest done. We faced a lot of challenges along the way. There were a lot of confusion and a bit lack of guidance in the instruction. However, we were able to get everything done for the first phase after a lot of work.

# Instructions:

We implemented allof the instructions successfully. To test that they worked as intended, we generated some instructions with a script and then put them into the memory file. Next we ran simulations to see that the instructions worked as intended.

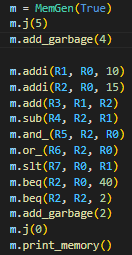
The script for creating instructions is a file named memgen.py

 -> 

## Simulation Proof:

**addi, add, sub, slt, and, or, beq, and j are shown to work as intended below**

Here is the assembly code and the corresponding memory file for this set of instructions:

08000005

FFFFFFFF

FFFFFFFF

FFFFFFFF

FFFFFFFF

2001000a

2002000f

00221820

00412022

00402824

00403025

0001382a

10020028

10420002

FFFFFFFF

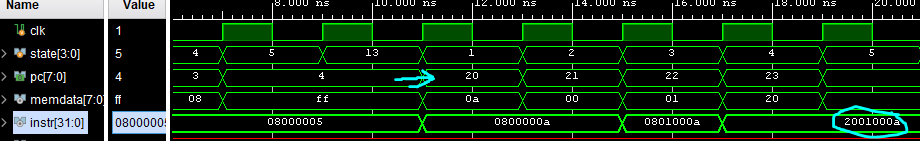
FFFFFFFF

08000000

These instructions simply do all of the rtype instructions and the beq/j instructions

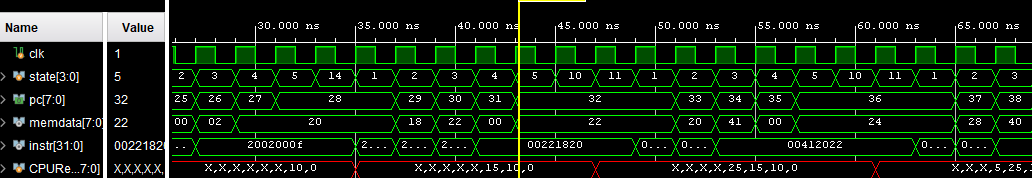
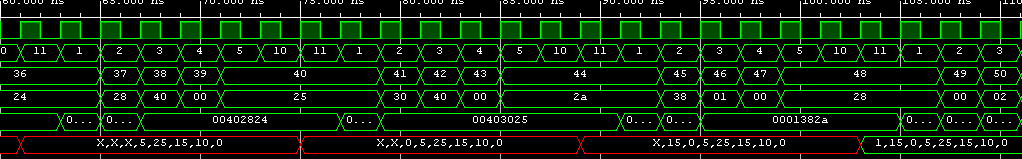
Here is the simulation:

First the jump instruction skips the garbage instructions and starts executing 2001000a

****

The first jump instruction skips the four garbage instructions and arrives at address 20 where it then starts the computations.

Then the values of the registers start to reflect the instructions given: (CPURegs)

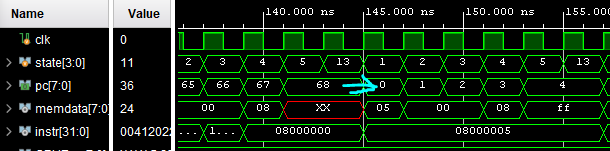


R2 is loaded with 15 and R1 is loaded with 10

R3 is R2 + R1 = 25

R4 = R2 - R1 = 5 and so on

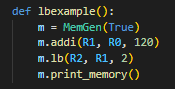
And at the end, the beq skips to the jump instruction which resets everything in a loop:



**Lb:**

Here is the code and memory file:

Note: the memory is filled with garbage so that we could place the data needed at address 128



20010078

80220002

FFFFFFFF

FFFFFFFF

FFFFFFFF

FFFFFFFF

FFFFFFFF

FFFFFFFF

FFFFFFFF

FFFFFFFF

FFFFFFFF

FFFFFFFF

FFFFFFFF

FFFFFFFF

FFFFFFFF

FFFFFFFF

FFFFFFFF

FFFFFFFF

FFFFFFFF

FFFFFFFF

FFFFFFFF

FFFFFFFF

FFFFFFFF

FFFFFFFF

FFFFFFFF

FFFFFFFF

FFFFFFFF

FFFFFFFF

FFFFFFFF

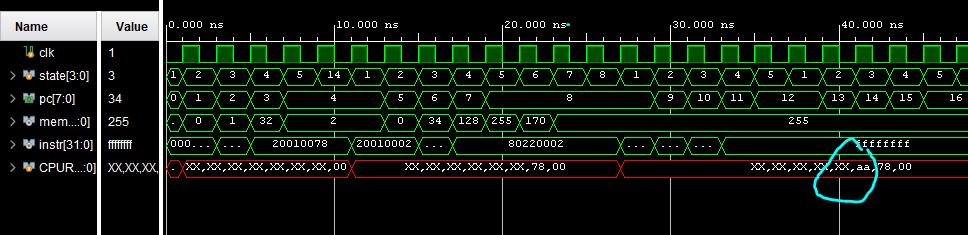
FFFFFFFF

FFFFFFFF

FFFFFFFF

CCCCCCAA

Simulation:



We did not have enough time to finish sb, but it can be done within time for the final presentation

# Synthesis

