**EENG 2910 Project III – Digital System Design**

Design Project 4

Nathan A Ruprecht

Ayodele Ojo

Maher Alsanwi

Due 26 October 2016

# Introduction:

For our fifth/sixth project, we were required to design, implement, and test a 4-bit program counter as well as a 1 word, 4-bit memory.

A program counter (PC) is a pointer that points to a location in the memory. The value of the pointer is the address in memory, not the data in that memory slot.

Memory is where data is stored. 2 raised to the power of address bits is how many addresses there are for data to be stored in. So 3 address bits means 8 memory addresses. 1 word is 8 bits (or 2 bytes). So for this assignment, we need 16 address locations, able to store 8 bits of data in each location.

# Theory of Operation and Explanation of the Design:

We were given the code for both the PC and RAM. We just had to change the logic vectors to reflect the correct number of bits needed. The I\_nPC, O\_PC, and I\_addr went down to 4-bit vectors since they dealt with the address locations and were linked to each other. The I\_data, O\_data, and O\_OP went to 8-bit vectors since they deal with the actual data located at I\_addr and were linked to each other.

Since we were given the code, the main point of this assignment is to recognize the PC doing the opcodes and see it increment down the line of address locations (when the opcode was to increment). As the PC changed, we needed to see the change of output data as PC changed what data in RAM it was pointing at.

# Experimental Results:

Since we were given the code, the first thing we did before running it was sit down, read through it, and understand what was supposed to happen.

The clock signal everything runs off of is I\_clk. On the rising edge we enter a case statement that depends on I\_nPCop which looks at the predefined values of PCU\_OP\_NOP (the opcode of the PC). When PCU\_OP\_NOP is 00, the PC does nothing (no operation). At 01, the PC increments to go to the next address location. At 10, the PC is assigned a value, I\_nPC. And at 11, the PC is resest to 0. The variable that holds the PC during this process is the signal, current\_pc. After the case statement, current\_pc is assigned to O\_PC which is the output to show the value of PC.

RAM is a component of the PC code. It uses the same clock, I\_clk. It takes in current\_pc as the address location (I\_addr) to know where in memory the PC is pointing. In the RAM code, it starts with checking for the rising edge. On the rising edge, we have the option to read or write. When it writes, I\_data is written to that memory location. Otherwise it checks if the PC (here assigned I\_addr) is greater than 16. If it is then O\_data (the return value of the component) is 0xFF. Otherwise, the output data is the value at that memory location. Since it’s a 4-bit memory, there are 16 memory locations. So as the PC (and therefor I\_addr) increments, the output is the value saved in the specified memory location.

What we immediately noticed about the testbench is the delay between changing the PC, and the output reflecting the correct value at the new memory location. There was a delay of 1 clock cycle. So the PC was assigned on a rising edge, and we would see the appropriate output of memory on the next rising edge. When we asked about it, it was emphasized to pay attention to this delay when we made our processor (Final Project for the course) so that the incorrect value wasn’t passed to the processor from the memory.

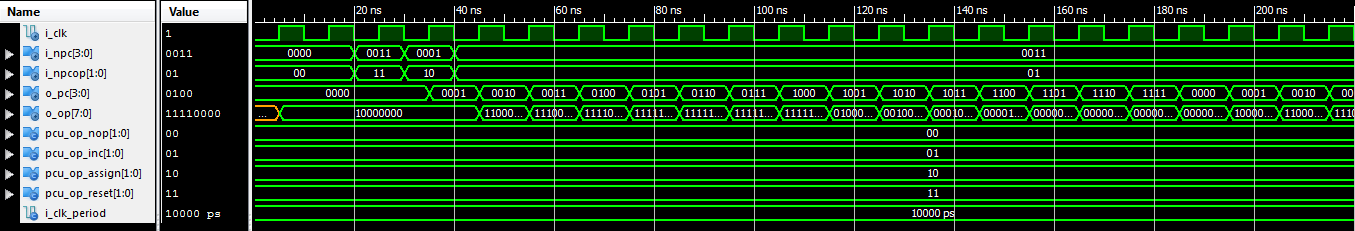
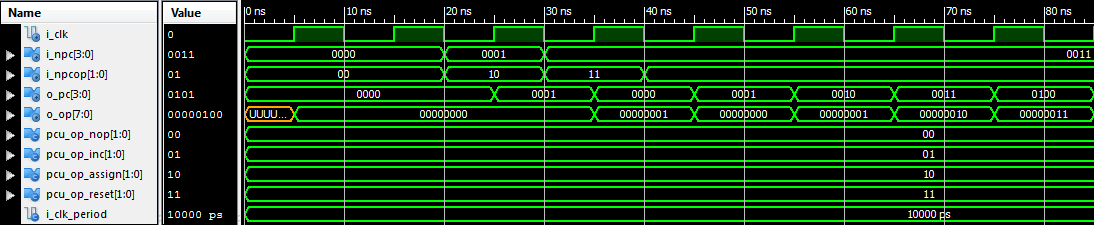
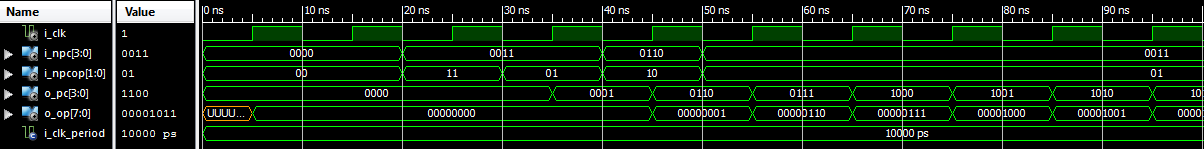


Figure : Overarching Test Bench

With the tb given (and modified to the correct vector size), we see everything at once. We zoomed in and modified the tesbench to verify that the change in opcode correctly changed the PC, which in turn successfully showed the correct output of the value at the memory location.



In the above test, we wanted to verify resetting the PC and incrementing it. When I\_nPCop was 00, the PC did not change. At 20 ns (falling edge), I\_nPCop was set to 10 which would make the PC increment. We saw that happen at 25 ns (rising edge) as it was the PC went from 0000 to 0001. We saw the appropriate data at the address location the next rising edge (35 ns) which is 0x01. At 35 ns is when the next opcode changed the PC to 0x0 which is resest. So at the next rising edge (45 ns) is where we see the data reflect the reset in address (back to 0x00). Then the opcode is to increment again, and we continue to see the pattern of changing the PC on the rising edge, and seeing the correct data from memory show to the output on the next rising edge.



# Conclusion:

The goal of the lab was to design, implement, and test an 4-bit PC as well as a 1 word, 4-bit memory. In the design process, I understood how a MUX/DEMUX was made using logic gates. How to make a simple MUX using a case statement. I expanded on given code to make an 8-bit wide, 8-to-1 MUX/DEMUX. I understood and implement the MUX/DEMUX to show an output the correctly selected the bit value of an input and successfully makes the rest of the 8-bit number 0 while holding that input value the same.

# References:

Figure 2 and Table 1 came from the lecture slides.

Figure 1 was made in paint.

Figures 3-5 are screenshots.

# Source Code: