Name: Alexander Wise **Lab #6 – Instruction Memory and Program Counter**

Instructor Name: Farzad Niknia **March 19, 2024**

Background

The instruction counter is a simple module that allows for the implementation of RISC-V instructions to be executed sequentially based on the value from the instruction counter. Similarly, the instruction memory serves as a storage location for these RISC-V instructions. The purpose of this lab is to implement both instruction memory and the program counter for more complex operations to be completed by the RISC-V processor.

Pre-Lab

There is no pre-lab for this lab.

Design Implementation

3.1.1 – Designing the Top Level

The top-level module serves as a fundamental component of the RISC-V processor and any System Verilog program as it links all components together to work concurrently. The top-level module was provided to us and it was simply uploaded to the project with some minor changes to match previous naming conventions set in prior System Verilog modules from previous labs.

3.1.2 – Instruction Memory

Next, the instruction memory was added to the processor. To do so, a Distributed Memory Generator was added from the IP catalog. This module takes in 16 bits of instruction memory and stores it within the instruction memory module. Accessing the instructions is done through the program counter which will be implemented next.

After being added, a .coe file was uploaded to the module. This file contained RISC-V instructions for the processor to process and compute.

3.1.3 – Program Counter

To iterate through each instruction stored within the Instruction Memory, a program counter was created. The program counter is an 8-bit counter that is reset with a reset input and incremented through the use of a push button. Each button press increments the value within the module by 1, thus accessing the next instruction stored within the instruction memory. The System Verilog code for this module can be found in Appendix A, labeled as Figure 1.

3.1.4 – Completing the Top Level

Changes were made to the VIO for the processor which had to be addressed. In the previous lab, the VIO had 10 input probes and 7 output probes (17 total), however in this lab, the VIO has 19 input probes as each output probe controlled in the previous lab is now controlled by the RISC-V instruction provided through the .coe file. Additionally, two more inputs were added to this lab including an input probe for the instruction and the program counter. After this implementation, the system was ready to be tested in hardware.

3.2.3 – Testing Your Design

Using the provided .coe file, each instruction was run through the processor. There was a total of 9 operations to be completed. The following table will show the converted instructions in RISC-V assembly:

Code	Operation	Opcode	Immediate	nzimm	offset	rd	rs1	rs2
05AD	addi 0x3, 0x3, 0xB	3	0	B	0	3	3	0
C14C	sw 0x3,0x4(0x2)	1	4	0	0	0	2	3
060D	addi 0x4, 0x4, 3	3	0	3	0	4	4	0
C910	sw 0x4, 0x10(0x2)	1	10	0	0	0	2	4
8605	srai 0x4, 0x4, 0x1	8	0	1	0	4	4	0
40C8	lw 0x2, 0x4(0x1)	0	4	0	0	2	1	0
8D51	or 0x2, 0x2, 0x4	6	0	0	0	2	2	4
4A80	lw 0, 0x10(0x5)	0	10	0	0	0	5	0
8C69	and 0, 0, 0x2	4	0	0	0	0	0	2

Table 1: Table of Instructions and their Associated RISC-V Assembly Code

Similarly, images of the program running on hardware can be found in Appendix B, labeled as figures 2 through 10.

Conclusion

In this lab, the instruction memory and program counter were both successfully incorporated into the RISC-V processor allowing for much more complex operations to be executed sequentially. The program counter serves as a queue executer for the instructions stored within the instruction memory to be executed in the predetermined order within the instruction memory. Similarly, the mapper file provided allowed for the decoding of the instructions to be read by the processor and executed. Together, each component works with one another to ensure the proper operation of the processor.

Appendix A – Screen Shots of System Verilog Code:

```
22
23  module program_counter(
24      input clk, rst,
25      output logic [7:0] pc
26      );
27
28      always @(posedge rst, posedge clk) begin
29          if(rst)
30              pc <= 0;
31          else if(clk)
32              pc += 1;
33       end
34
35  endmodule
36
```

Figure 1: Screen shot of System Verilog Code for Program Counter Module

Appendix B – Images of Code Working on Hardware

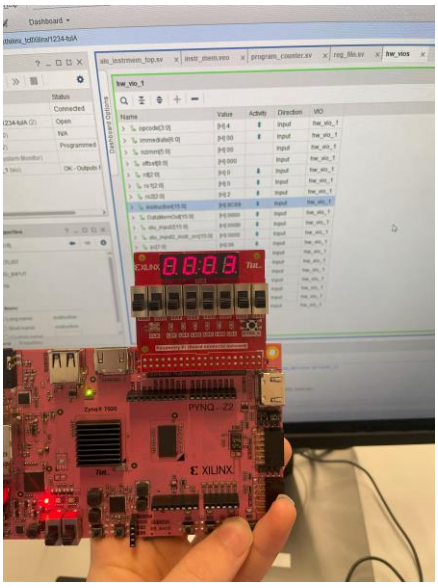

Figure 2: Instruction #1 on Hardware

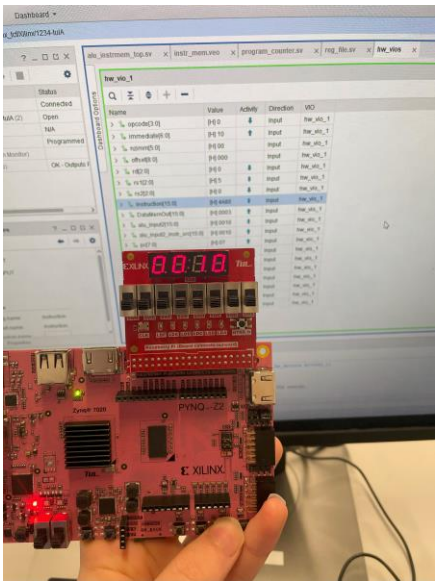

Figure 3: Instruction #2 on Hardware

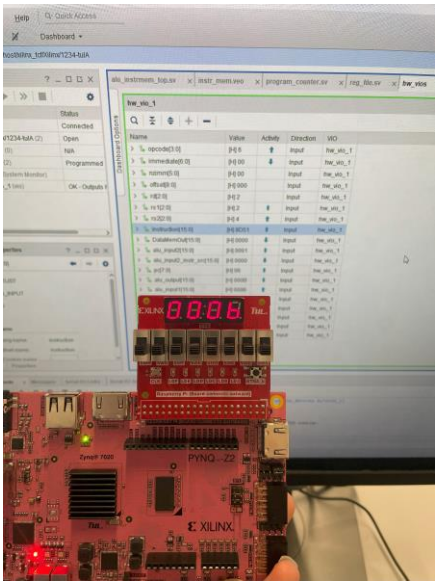

Figure 4: Instruction #3 on Hardware

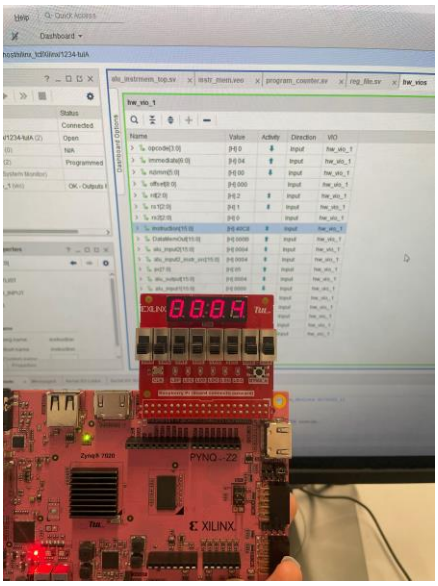

Figure 5: Instruction #4 on Hardware

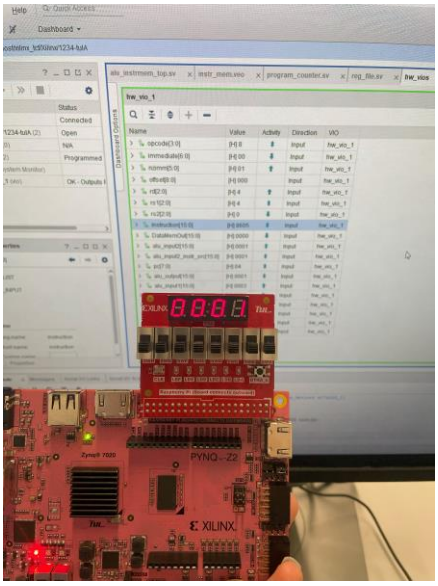

Figure 6: Instruction #5 on Hardware

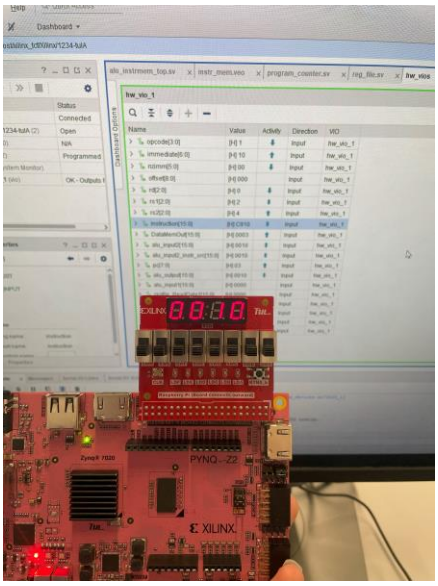

Figure 7: Instruction #6 on Hardware

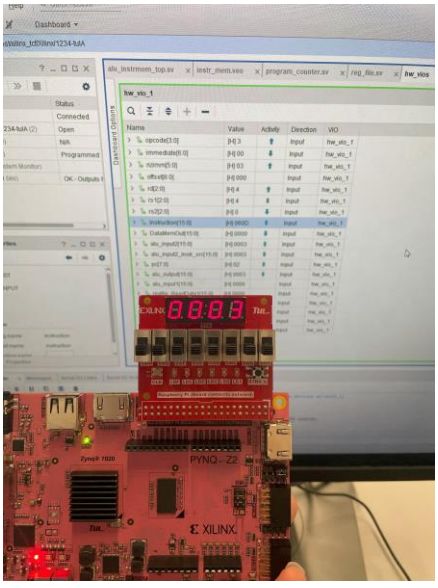

Figure 8: Instruction #7 on Hardware

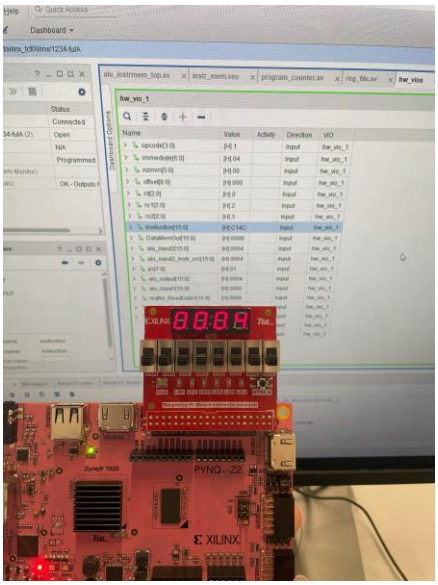

Figure 9: Instruction #8 on Hardware

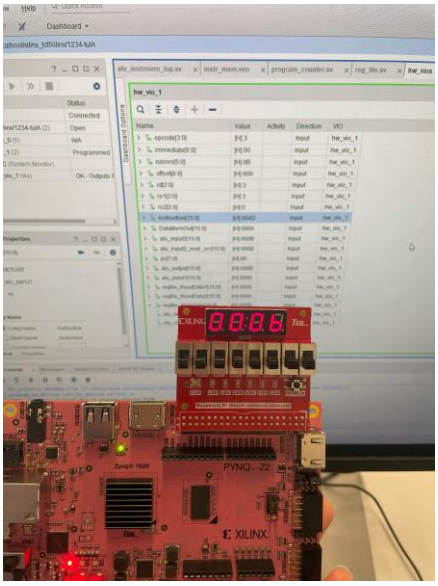

Figure 10: Instruction #9 on Hardware