RISC PROCESSOR IN VERILOG HDL

by

Felipe Rangel

Rebecca Camejo

Nicholas Alvarez

Rachel Del Paz

Jordan Cooley

Group # 1

Final Project Report

ECE 414: Computer Organization and Design

December 7th, 2021

Project directed by

Dr. Onur Tigli

Asst. Professor of Electrical and Computer Engineering

TABLE OF CONTENTS

ABSTRACT

As the final project in the class, our group designed and tested a functioning Central Processing Unit, or CPU using Verilog HDL to program sub-modules and synthesize them onto a FPGA. The processor uses a Reduced Instruction Set Architecture to perform a small set of instructions that are read from a read-only memory module and executed using a random access memory module as an external storage medium. The project serves as a sample of how modern processors are designed, tested, and verified using simulation tools and programmable logic for prototyping instead of requiring the full chip fabrication process, which is expensive and time-consuming.

INTRODUCTION

The CPU design is based on the Harvard architecture and uses a simplified 16-bit version of the MIPS architecture. The computer is multi-cycle, with instructions occurring over two cycles. There are 16 possible instructions that can be executed, and can address 16 general purpose registers. The CPU is connected to a ROM which holds the instructions, and a RAM to read and write to external memory. They both have 6 address lines and 16 data lines (there are only 64 memory locations available in RAM). There is a dummy instruction to indicate the end of execution, which controls a one bit execution complete line. The entire CPU is clocked on the FPGA using the internal 100 MHz clock, and has a reset signal to reset all internal registers in the CPU to 0 and to reset the PC. The ALU supports basic add and subtract operations, as well as logical operations for and, or, xor, not, and shift left or right by one bit. There are three memory operations to load an immediate value into a register, or to read/write to RAM at a particular address to/from a particular register. There are five branch/jump instructions which allow execution to deviate from the order that the instructions are stored in based on conditions, as well as the ability to jump and return. By using these simple instructions, it is possible to put together basic programs and run them on the synthesized hardware loaded onto an FPGA.

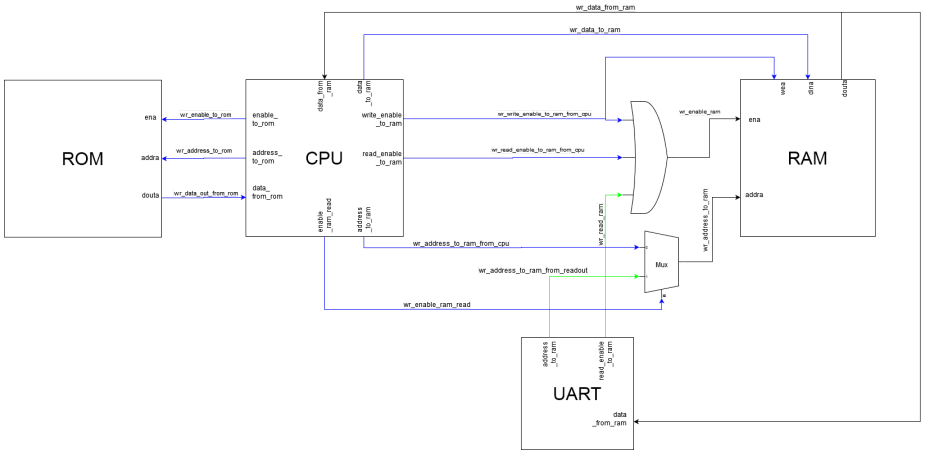
DESIGN and RESULTS

Top-level:

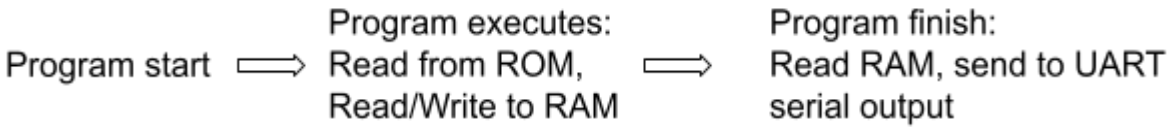
The goal of this project is to implement a fully functioning CPU that can be programmed onto an FPGA and communicate with a ROM and RAM module to run a simple assembly program. The CPU block connects to RAM and ROM and then the RAM connects to a UART module which transmits the full RAM contents on program finish to a serial connection, which is received through a PuTTY signal on a computer connected to the FPGA.

The design of the full system has many, small interconnected parts, which requires careful tracking of how individual components interact with each other. The biggest factor in success is in properly designing and testing the submodules independently of each other, and slowly connecting and testing along the way, as opposed to working from the top-level downwards. The most complicated part of connecting the CPU to the RAM and ROM modules is the final step of transmitting the final RAM dump after program finish. The ROM is simply a block of memory, and only serves to store the full program data. On the other hand, the RAM can be both read and written to during execution, and once the program is finished its contents represent the final state of the CPU since the register file is inaccessible from the top level.

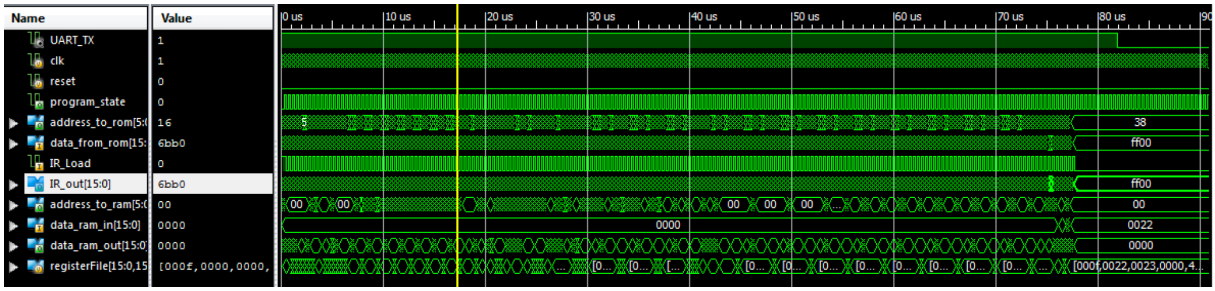
Block diagram:



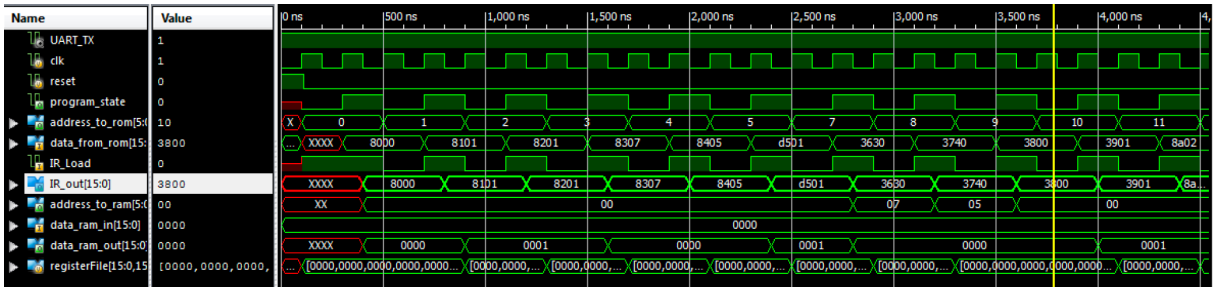
· Flowchart (of the program, the algorithm):



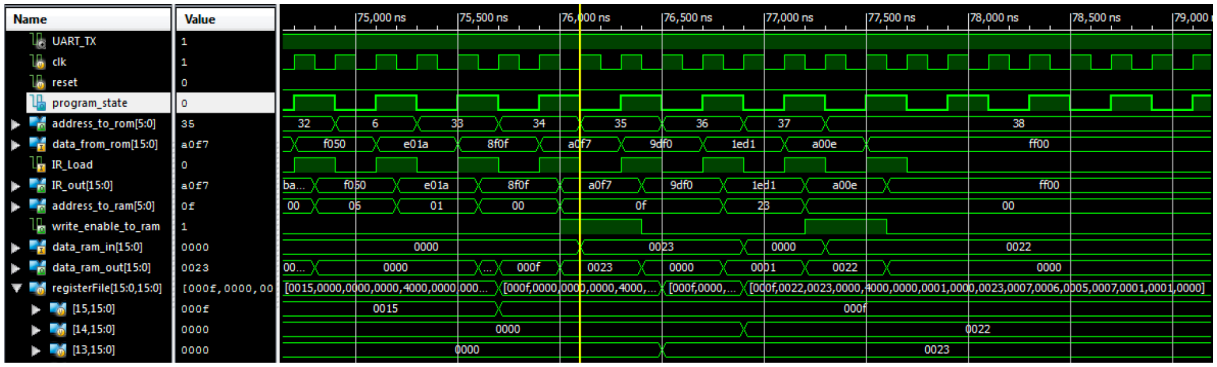
Simulation results



Overall top simulation



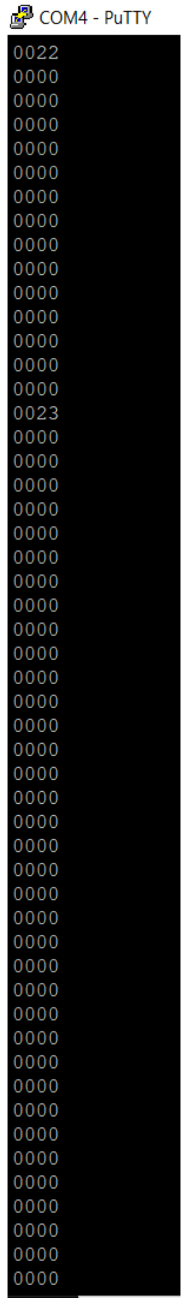
Start of program execution



End of program execution:

Most important to note that the RAM is written to address 0xF with data 0x23, then register D (13th register) is loaded from that same RAM address with data 0x23. Afterwards, it performs a subtraction operation and stores the result 0x22 into register E (14th register).

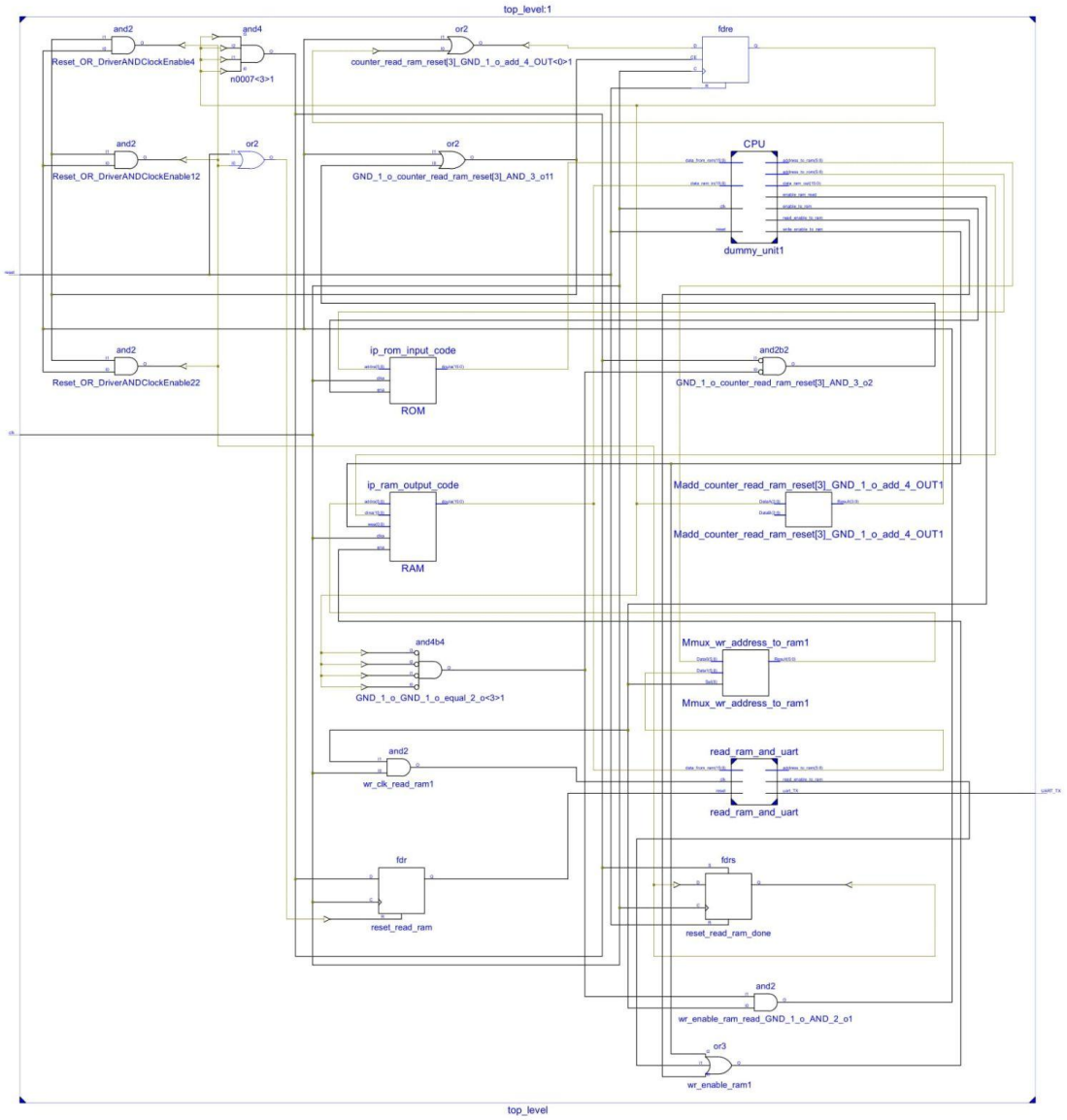
Lastly, that result 0x22 is stored at RAM address 0x0 and the program reaches the end of execution.



Putty Output:

This provides the contents of the RAM with address 0x0 at the top. Note that the data 0x22 is at address 0x0 and the data 0x23 is at address 0xF, which is stored during the final instructions in the COE program.

Synthesized circuitry:



Performance metrics:

For time, as can be seen from the simulation results, the timing requirements of two cycles per instruction was satisfied. The area usage is shown in the device utilization summary below, and the power consumption is shown in the final chart, at 0.103 W when running at 100 million transitions per second for the 100 Mhz clock.

Device Utilization Summary				[-]
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	385	126,800	1%	
Number used as Flip Flops	385			
Number used as Latches	0			
Number used as Latch-thrus	0			
Number used as AND/OR logics	0			
Number of Slice LUTs	778	63,400	1%	
Number used as logic	775	63,400	1%	
Number using O6 output only	699			
Number using O5 output only	41			
Number using O5 and O6	35			
Number used as ROM	0			
Number used as Memory	0	19,000	0%	
Number used exclusively as route-thrus	3			
Number with same-slice register load	1			
Number with same-slice carry load	2			
Number with other load	0			
Number of occupied Slices	297	15,850	1%	
Number of LUT Flip Flop pairs used	807			
Number with an unused Flip Flop	427	807	52%	

Number with an unused LUT	29	807	3%	
Number of fully used LUT-FF pairs	351	807	43%	
Number of unique control sets	14			
Number of slice register sites lost to control set restrictions	47	126,800	1%	
Number of bonded IOBs	3	210	1%	
Number of LOCed IOBs	3	3	100%	
Number of RAMB36E1/FIFO36E1s	0	135	0%	
Number of RAMB18E1/FIFO18E1s	2	270	1%	
Number using RAMB18E1 only	2			
Number using FIFO18E1 only	0			
Number of BUFG/BUFGCTRLs	2	32	6%	
Number used as BUFGs	2			
Number used as BUFGCTRLs	0			
Number of IDELAYE2/IDELAYE2_FINEDELAYs	0	300	0%	
Number of ILOGICE2/ILOGICE3/ISERDESE2s	0	300	0%	
Number of ODELAYE2/ODELAYE2_FINEDELAYs	0			
Number of OLOGICE2/OLOGICE3/OSERDESE2s	0	300	0%	
Number of PHASER_IN/PHASER_IN_PHYs	0	24	0%	
Number of PHASER_OUT/PHASER_OUT_PHYs	0	24	0%	
Number of BSCANs	0	4	0%	
Number of BUFHCEs	0	96	0%	
Number of BUFRs	0	24	0%	

Number of CAPTUREs	0	1	0%	
Number of DNA_PORTs	0	1	0%	
Number of DSP48E1s	0	240	0%	
Number of EFUSE_USRs	0	1	0%	
Number of FRAME_ECCs	0	1	0%	
Number of IBUFDS_GTE2s	0	4	0%	
Number of ICAPs	0	2	0%	
Number of IDELAYCTRLs	0	6	0%	
Number of IN_FIFOs	0	24	0%	
Number of MMCME2_ADVs	0	6	0%	
Number of OUT_FIFOs	0	24	0%	
Number of PCIE_2_1s	0	1	0%	
Number of PHASER_REFs	0	6	0%	
Number of PHY_CONTROLs	0	6	0%	
Number of PLLE2_ADVs	0	6	0%	
Number of STARTUPs	0	1	0%	
Number of XADCs	0	1	0%	
Average Fanout of Non-Clock Nets	6.68			

Device	
Family	Artix7
Part	xc7a100t
Package	csg324
Temp Grade	Commercial
Process	Typical
Speed Grade	-1

Environment	
Ambient Temp (C)	25.0
Use custom TJA?	No
Custom TJA (C/W)	NA
Airflow (LFM)	250
Heat Sink	Medium Profile
Custom TSA (C/W)	NA
Board Selection	Medium (10"x10")
# of Board Layers	12 to 15
Custom TJB (C/W)	NA
Board Temperature (C)	NA

Characterization	
Production	v1.0,2012-07-11

On-Chip	Power (W)	Used	Available	Utilization (%)
Clocks	0.002	2	---	---
Logic	0.005	778	63400	1
Signals	0.005	747	---	---
BRAMs	0.002	*	*	*
IOs	0.000	3	210	1
Leakage	0.088			
Total	0.103			

Thermal Properties	Effective TJA (C/W)	Max Ambient (C)	Junction Temp (C)
	4.6	84.5	25.5

Supply Summary		Total	Dynamic	Quiescent
Source	Voltage	Current (A)	Current (A)	Current (A)
Vccint	1.000	0.031	0.015	0.017
Vccaux	1.800	0.013	0.000	0.013
Vcco33	3.300	0.004	0.000	0.004
Vccbram	1.000	0.001	0.000	0.000
Vccadc	1.710	0.020	0.000	0.020

	Total	Dynamic	Quiescent
Supply Power (W)	0.103	0.015	0.088

ROM/RAM:

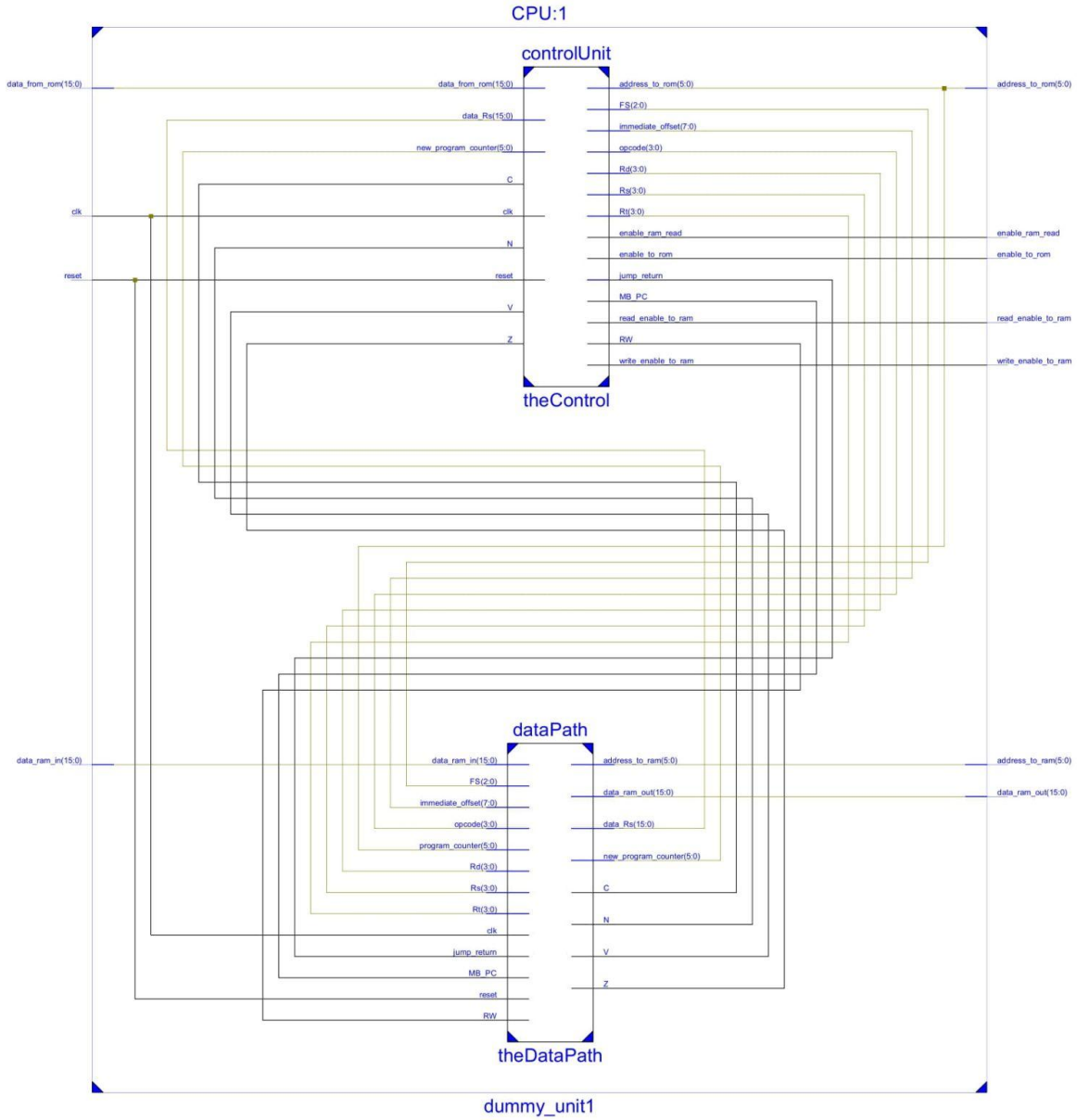
The ROM and RAM are externally sourced to the CPU. The ROM is created using the Block Memory Generator to make an ip core containing the binary code of the instructions of the program to be executed. The RAM is also made as an ip core. The two blocks can be accessed by the CPU during execution. After program completion, the RAM is read and transmitted to the UART port, which can be read on the computer connected to the FPGA through PuTTY.

CPU:

The CPU block is a simple component, with data, address, and enable lines to both the RAM and ROM, and contains the datapath and control unit which are connected to each other. The datapath is responsible for holding the register file and ALU, and the control unit holds the program counter and instruction register, as well as the logic to drive the datapath.

Block diagram:

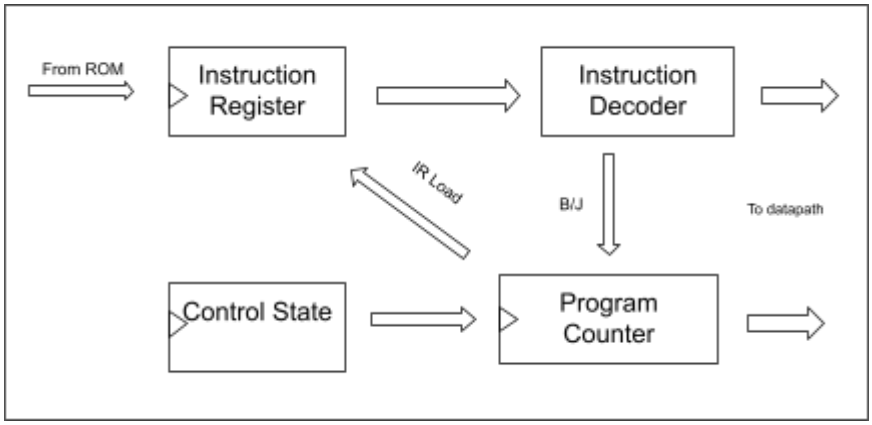
Synthesized circuitry:



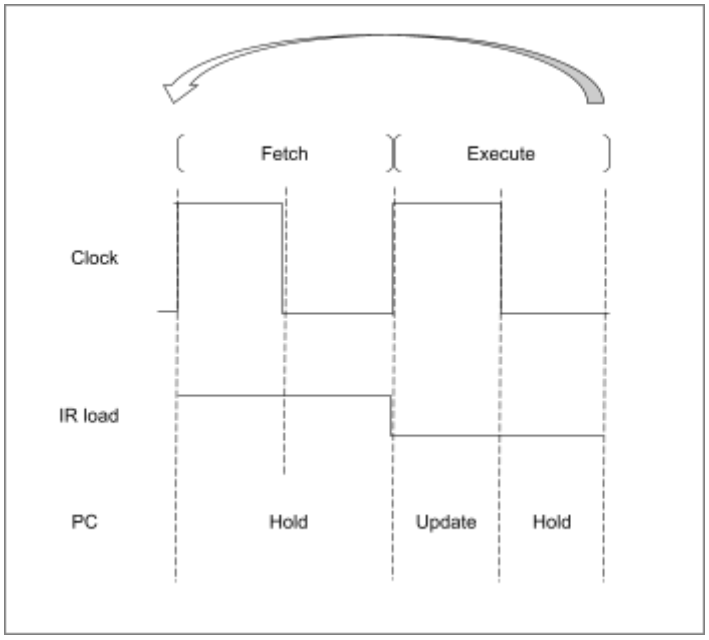
Control Unit:

The control unit has registers to track the location of the current instruction, storage for that instruction, and logic to decode the instruction (which drives the datapath) and for the current stage of execution, either instruction fetch or execution.

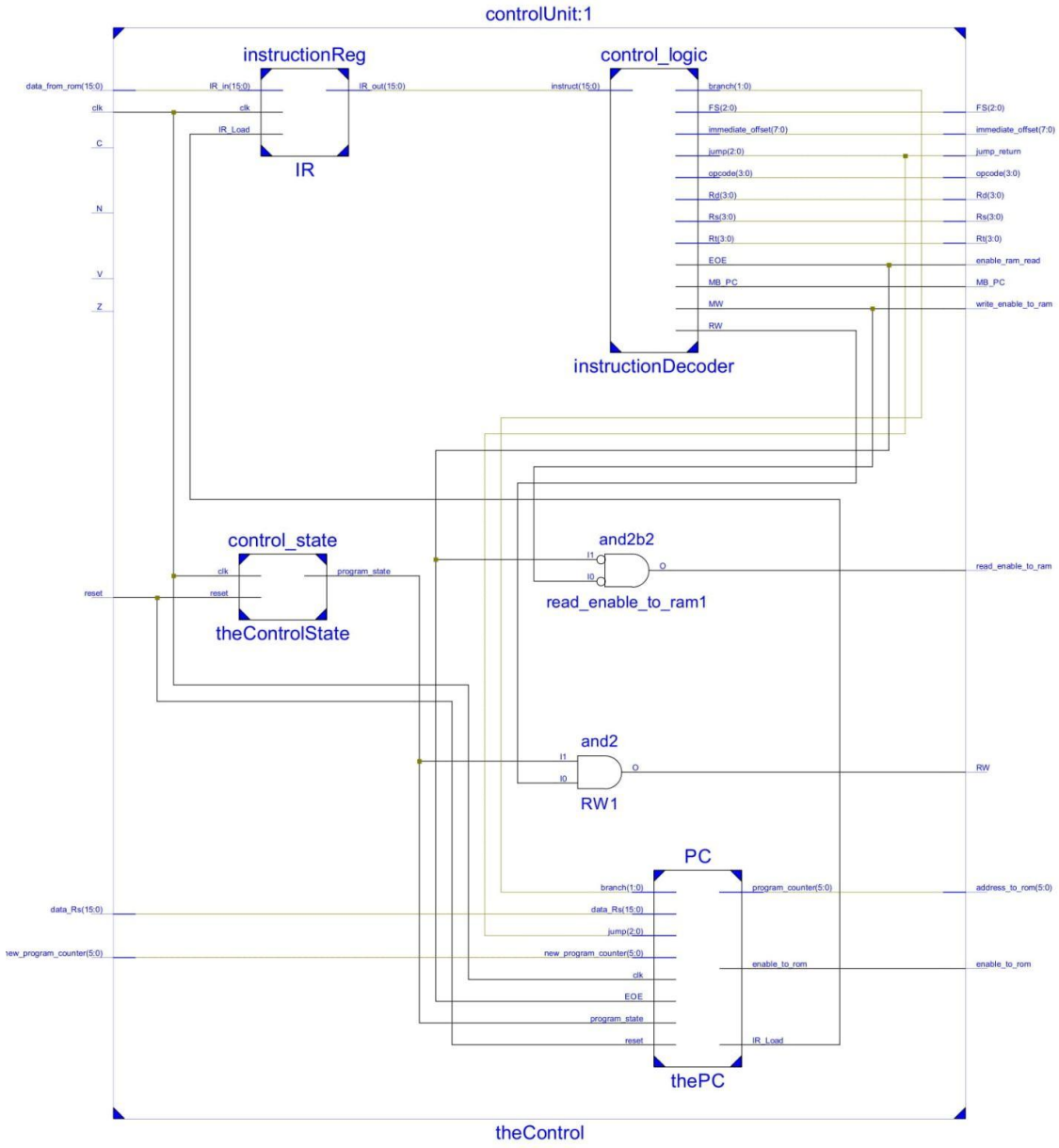
Block diagram:



Flowchart:



Synthesized circuitry:



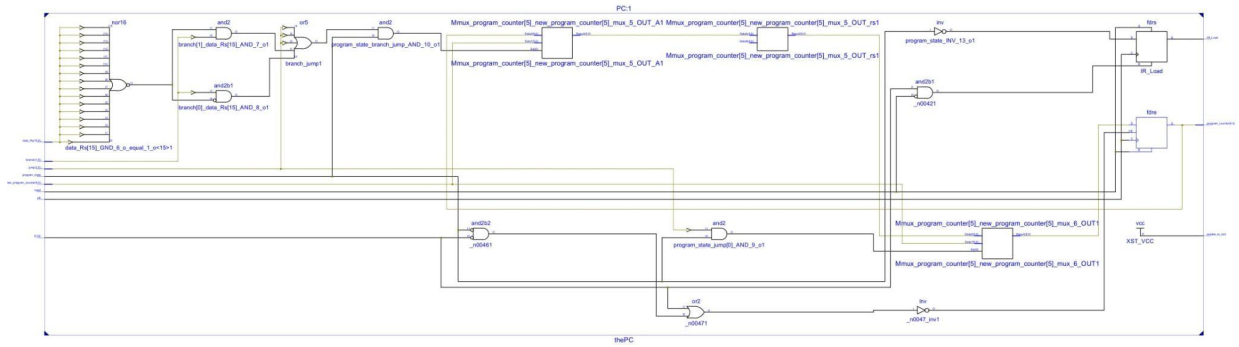
PC:

The program counter is a special register which has to track the execution of the program by storing the address to ROM as well as incrementing or jumping/branching on the start of the execute cycle. It also is responsible for telling the instruction register to load at the

start of the fetch cycle.

The special cases of reset and end of execution set the PC to zero or hold its value respectively. Otherwise, on the fetch cycle, the value is held; and on execution, the value increments by one normally or can change to a new address plus one on a branch or jump.

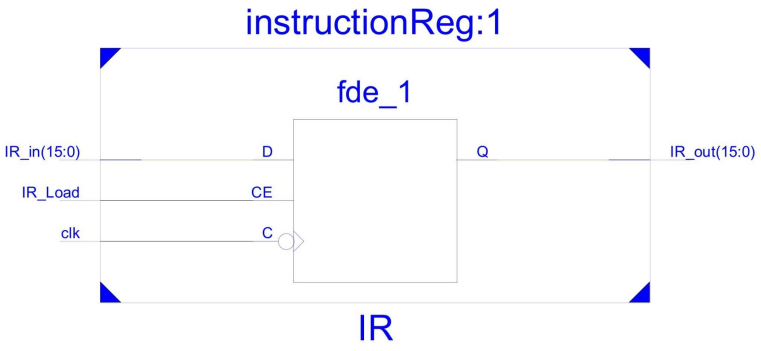
Synthesized circuitry:



IR:

The instruction register is simply a register which holds the current instruction being executed. On the fetch cycle negative edge, it is loaded from ROM with the instruction pointed to by the program counter. On the execute cycle, the instruction is fed through the instruction decoder logic, which then drives the datapath.

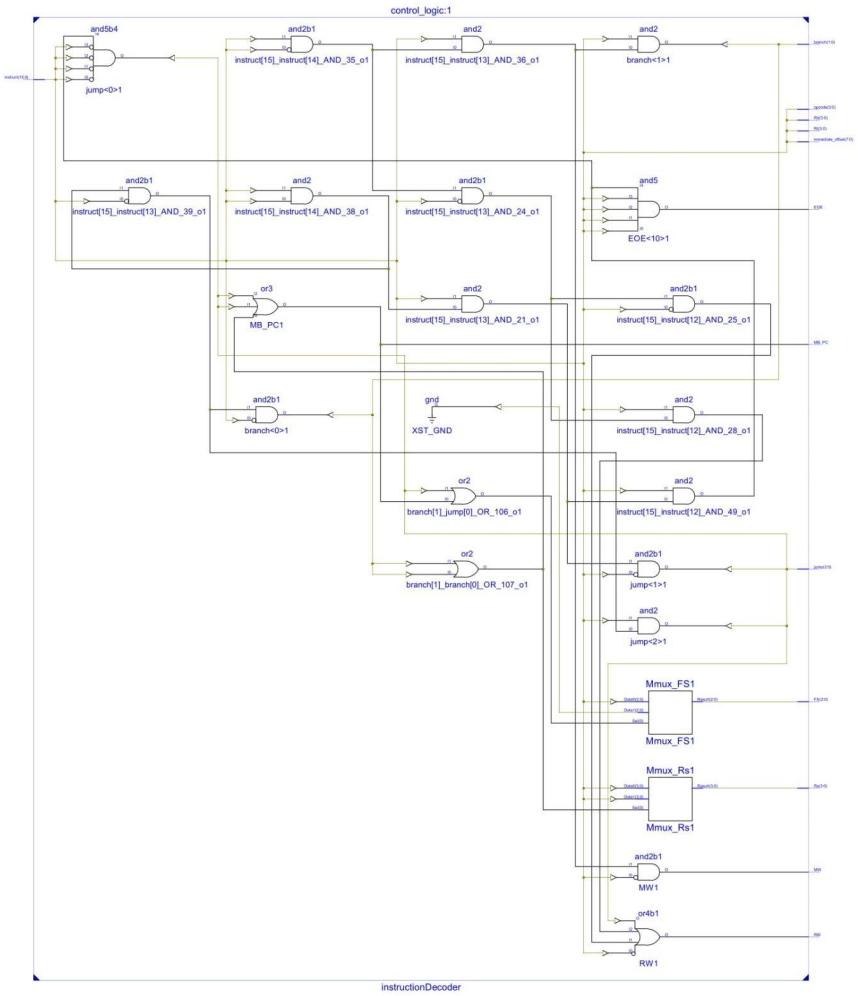
Synthesized circuitry:



Instruction Decoder:

The instruction decoder logic is a purely combinational circuit which breaks apart the 16 bit instruction stored in the instruction register and drives the datapath's input control signals based on the current instruction. Because the logic is purely combinational and not clocked, it is always decoding the contents of the instruction register, which means the datapath signals it controls are changing even during the fetch cycle. However, this has no effect on the datapath, because the datapath load signals and outputs are only used on the execute cycle.

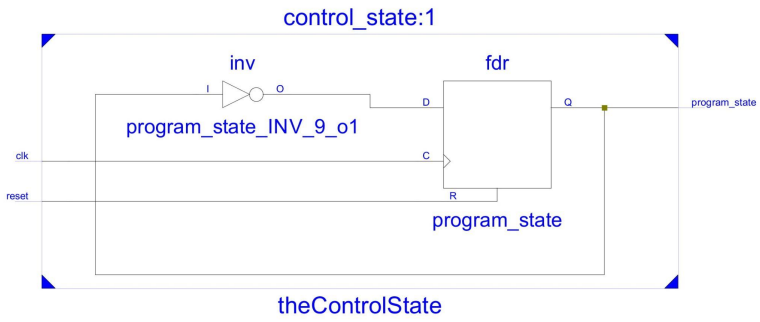
Synthesized circuitry:



Control State:

The control state is a register that stores which cycle is currently active, either fetch or execute. Since this reduced ISA only has single-execute-cycle instructions, it simply swaps between the two cycles every clock cycle, meaning it is a simple flip flop.

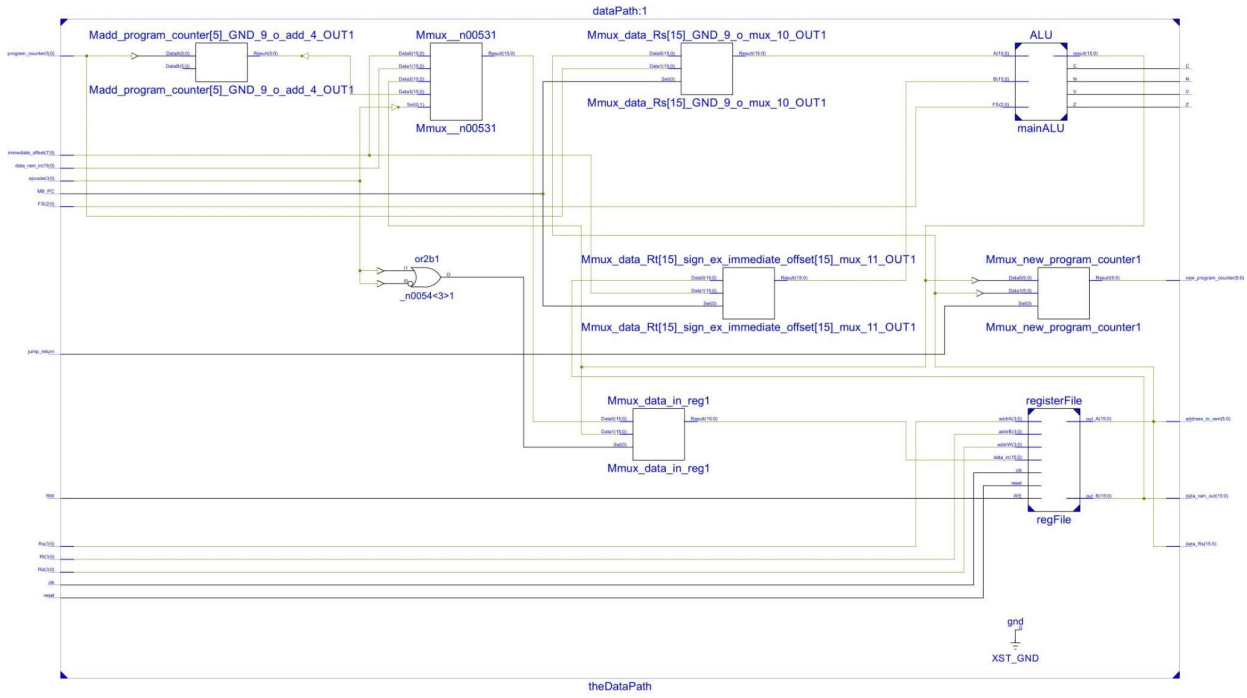
Synthesized circuitry:



Datapath:

The datapath is the section of the CPU responsible for performing the actual execution of the instructions, as well as containing the register file which contains 16 general purpose registers for use by the programmer during execution to store data between instructions. The datapath contains the ALU, which performs the actual execution of operations, and also has connections to the control unit, and the RAM for an external storage medium.

Synthesized circuitry:

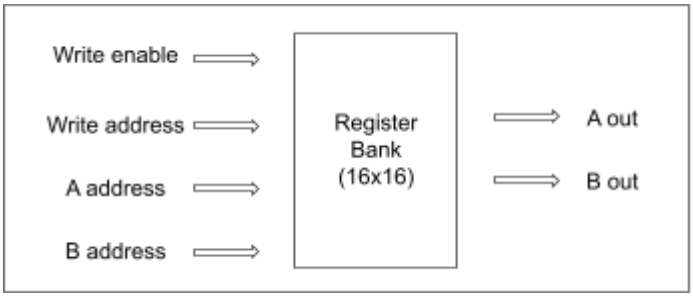


Register File:

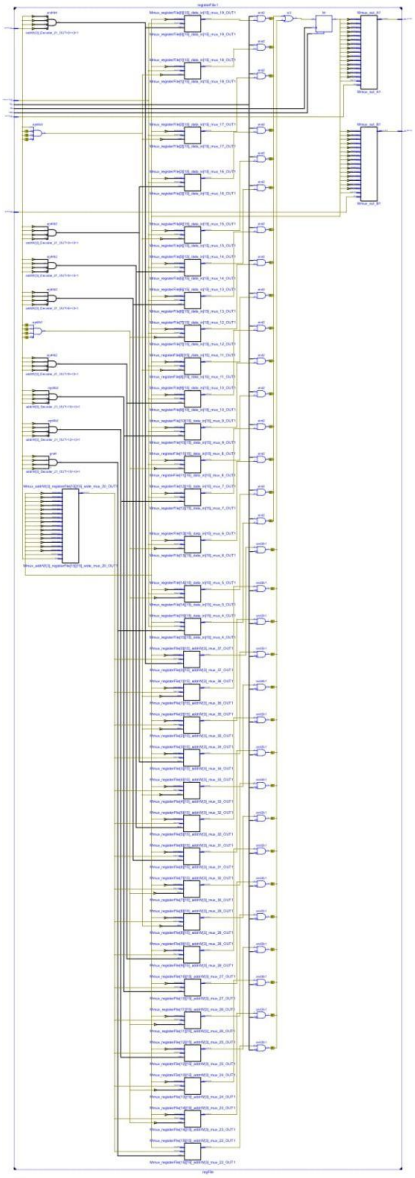
The register file is a bank of 16 bit registers which are used during program execution to store and load data between instructions. The file allows writing to one register at a time on a positive clock edge, and has two outputs for two different registers at once, all determined by input address lines. There is also a full reset to zero out all the registers.

Since there are 16 of the 16-bit registers, all three of the address lines need to be 4 bits wide to access all the registers.

Block diagram:



Synthesized circuitry:



The register file schematic is misleading, and should synthesize into a single register file. The synthesis tool seems to have decided to make two register files because the block is designed to output from two registers at once.

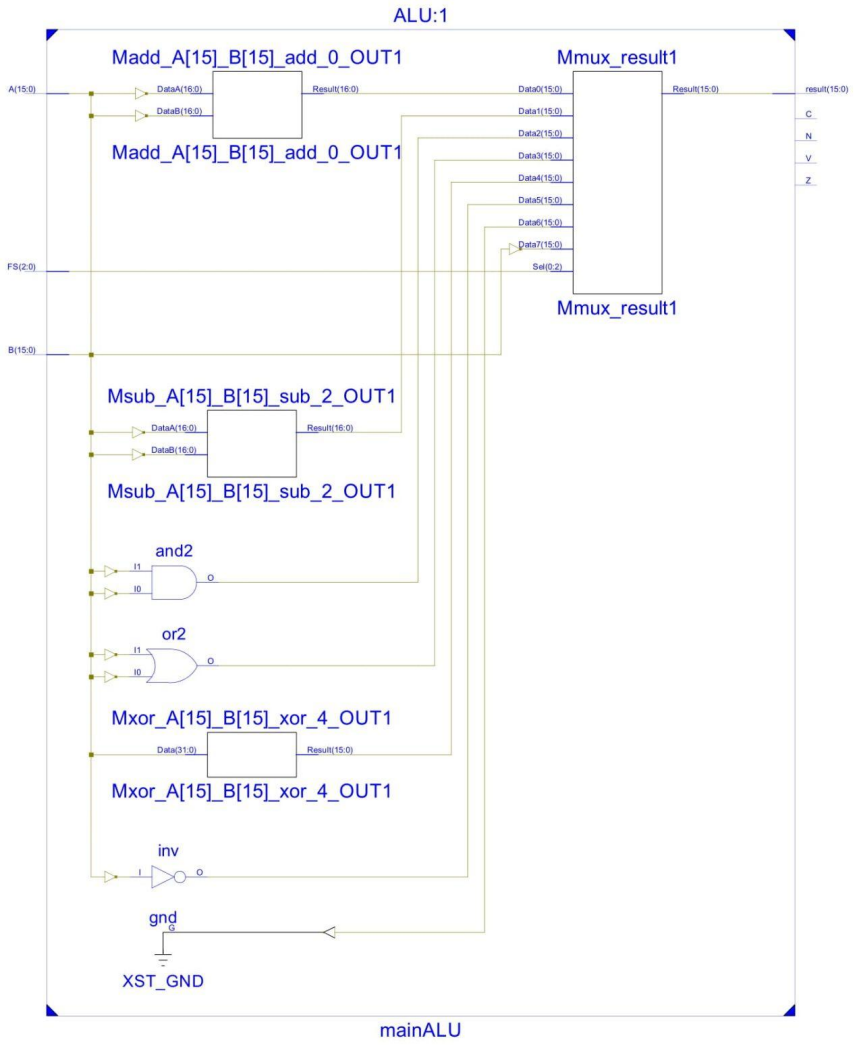
ALU:

The Arithmetic Logic Unit is the heart of the processor, which actually performs the operation defined by the instruction. The reduced ISA means the only operations it needs to perform are addition and subtraction, and basic bitwise logic operations for and, or, xor, and not.

The ALU must also detect overflow (V), zero (Z), carry (C), and negative (N), and report those four status bits to the control unit. However, those four bits are not used in the ISA, because the only instructions that depend on them actually use the values stored in registers instead of the status bits. The synthesis tool thereby decided to remove the functionality entirely because the logic would be wasted and has no effect on functionality. It left the outer connections between the datapath and control unit intact, but the wires are merely a formality and do not drive anything at all on either end.

The ALU is also used during branch and jump instructions to calculate the new PC after offsetting, as opposed to not using the ALU at all during those instructions. This saves on components, area and power slightly, with a slight increase in complexity, since the datapath has two extra multiplexers to select between either the PC and offset or the contents of the two registers A and B. In addition, the instruction decoder also makes sure to set the FS of the ALU to 000 during a branch or jump instruction to perform signed addition for the PC and offset to deliver to the program counter.

Synthesized circuitry:



CONCLUSION

The CPU design, which was split into a datapath and a controller, allows for simple programs to be executed from ROM and stored into RAM, which can be transmitted and read back to see the result. In the design process, there were many problems encountered, and plenty of debugging was required. The datapath is a simpler component, composed of mostly combinational logic along with a register file, which is a simple bank of registers. The control unit however proved to be much more difficult to design. Because the control unit needs to both track the current state of the processor and control the datapath signals, it requires many sub-modules that can decipher different parts of the process. Namely, decoding the instructions in order to properly inform the datapath, and properly manipulating the program counter register in order to account for the branching functionality. There was also difficulty in making the different pieces synchronize together over the two full clock cycles, especially when it came to the interactions between the program counter and the instruction register. In order to expand on the project, making the processor pipelined would potentially improve the performance of the processor and would serve as a better representation of how modern chips are designed. However, with the small amount of space available in ROM to store the programs, pipelining would do very little to improve the overall time the program takes to run. It should also be fairly easy to expand the processor to a 32, 64, or even 128-bit architecture, which would increase the amount of available instructions, memory, and registers greatly, and would also more closely resemble modern CPU designs.

REFERENCES

[1] M. Morris Mano, and C. Kime, *Logic and Computer Design Fundamentals, 5th Edition*, Addison-Wesley, 2015