ECE 385

Fall 2023

Experiment 5

Simple Computer SLC-3.2 in System Verilog

Alizain Bandukwala and Akshat Mehrotra

12:00 Friday

TW

Introduction: The basic functionality of the microprocessor was to be able to fetch,decode and execute instructions based on the microprocessor LC-3 ISA. The tests performed were based on 11 instructions which were created to perform a certain kind of function which was determined by its 16-bit instruction. The microprocessor created was a subset of this called S-LC3 based on two parts which involved 5.1 and 5.2. The first part was just based on the fetch phase of the processor which handled receiving the instructions from the user. Week 2 consisted of implementing the Decode and Execute phase which was reading the instruction and executing the correct function based on the opcode being the first 5 bits of the instruction. Constructing all three of these phases completed the microprocessor S-LC3.

Written Descriptions and DIagrams of S-LC3: The microprocessor's basic functionality was to first read the instruction from the user called the Fetch phase. This phase included receiving the instruction from memory based on 3 steps to achieve this.

1) Mar takes the address from the PC which is the register that stores the value of the address of the instruction to perform. This also included incrementing the PC counter by one which is the program counter for the microprocessor.
2) Store the instruction inside the MDR register which is the register that determines which instruction should be read or written
3) Then this instruction from the MDR is put in the IR register which is the register that decodes the instruction

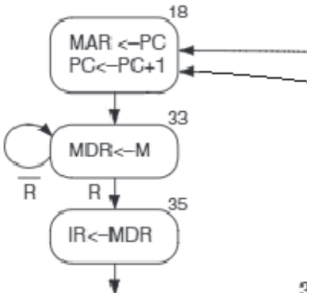


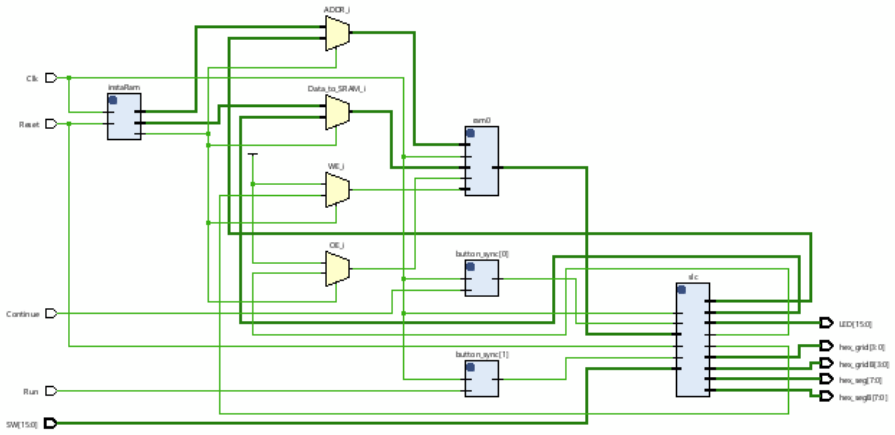
Figure 1: Fetch Phase FSM

The next phase the S-LC3 performs is the Decode step which takes the instruction and decodes the 16-bits into their specified location to perform the correct function the input requires. This requires taking the bits from the IR and decoding each section to find out key information including the type of function to perform, destination register, source registers, immediate value, etc. This involves the ISDU which is the module that stores all the control signals to be switched on or off and which components to be used such as adders, muxes, or any sign extending which needs to happen. After this comes the last step which is to compute the functions based on the control signals and the proper values are stored and delivered to the output registers. This

microprocessor can perform 8 operations and these are based on the ISDU and inputs from the user.

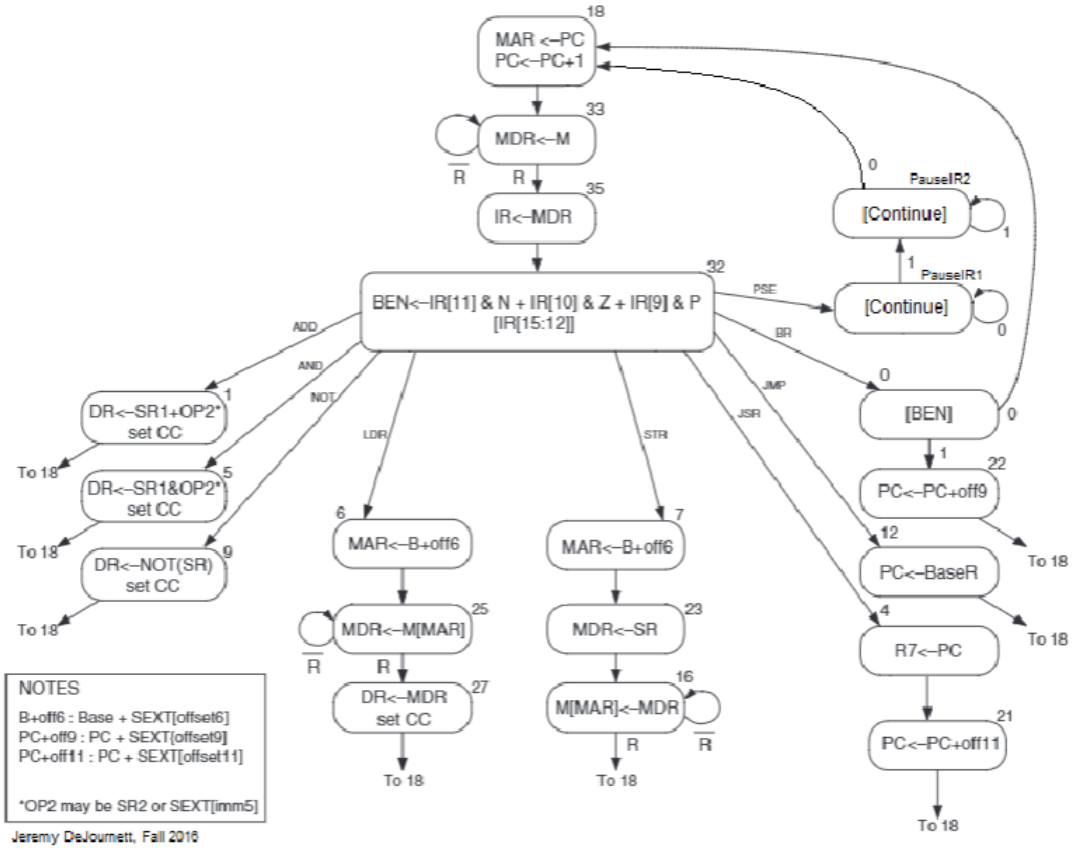
Instruction	Instruction(15 downto 0)																Operation
ADD	0001			DR			SR1		0	00		SR2					R(DR) ← R(SR1) + R(SR2)
ADDi	0001			DR			SR		1		imm5						R(DR) ← R(SR) + SEXT(imm5)
AND	0101			DR			SR1		0	00		SR2					R(DR) ← R(SR1) AND R(SR2)
ANDi	0101			DR			SR		1		imm5						R(DR) ← R(SR) AND SEXT(imm5)
NOT	1001			DR			SR				111111						R(DR) ← NOT R(SR)
BR	0000	n	z	p							PCoffset9						if ((nzp AND NZP) != 0) PC ← PC + SEXT(PCoffset9)
JMP	1100			000			BaseR				000000						PC ← R(BaseR)
JSR	0100	1									PCoffset11						R(7) ← PC; PC ← PC + SEXT(PCoffset11)
LDR	0110			DR			BaseR				offset6						R(DR) ← M[R(BaseR) + SEXT(offset6)]
STR	0111			SR			BaseR				offset6						M[R(BaseR) + SEXT(offset6)] ← R(SR)
PAUSE	1101										ledVect12						LEDs ← ledVect12; Wait on Continue

Table 1: The SLC-3.2 ISA



RTL Block Synthesis

LC3 STATE DIAGRAM FROM APPENDIX C OF PATT AND PATEL



SV Modules:

Slc3.sv: instantiates all the control signals, modules, memory to IO controls, registers, muxes, and computational units. Also loads hex driver variables to display the instructions and output on the board. Includes all BUS controls for the datapath and Gate select logic for input to the BUS.

Hex.sv: This file is a given file which acts as an IO interface to communicate with the HEX displays on board the FPGA board.

Registers.sv: Constructs the registers based on parameter N which is the size of the bits the register can hold. Controls with a clock input and based on load signal to control loaded in and out. Reset also enabled to clear registers anytime reset signal is high

sext.sv : Used to sign extend for computation depending on the size of bits needed to extend.
alu.sv : Computational unit to perform the four operations which are AND, ADD, NOT, and just pass A. This unit has four options for outputs and takes in one input and a mux input to control when register SR1 is used and checks between Sr2 and a sign extended output from the IR

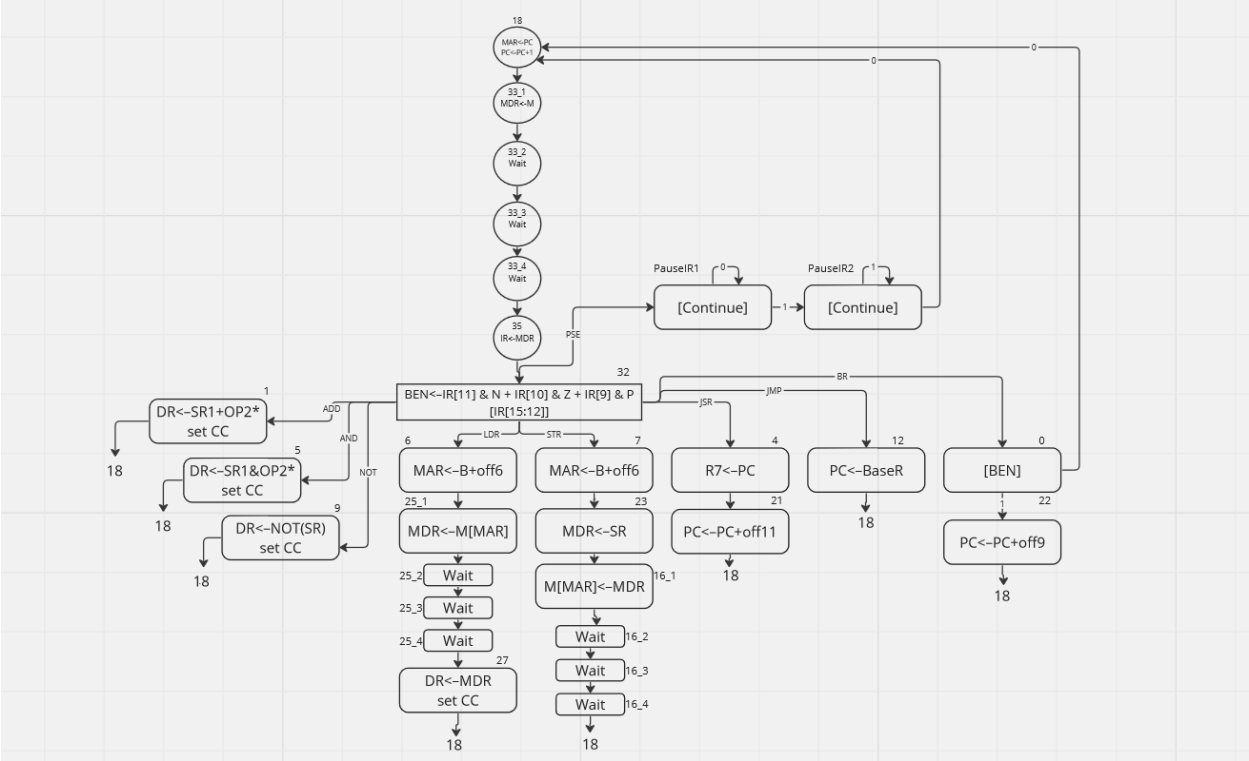
Sr1mux.sv: This mux controls the source register sr1 which can either be in IR[11:9],IR[8:6] and R6 which is bits 110. This mux outputs one of these signals into the register file to choose from source registers based on the instruction given

Drmux.sv: This mux controls the destination register and where to store based on three inputs IR[11:9,R7 which is 111, or R6 which is 110. This mux outputs one of these options into the register file

regfile.sv : Stores all the registers (R0-R7) and takes the operation from the BUS and based on the SR2 which is is IR[2:0],Sr1 Mux output and Drmux output to select the two source registers and one destination register based on the bits provided by the instruction

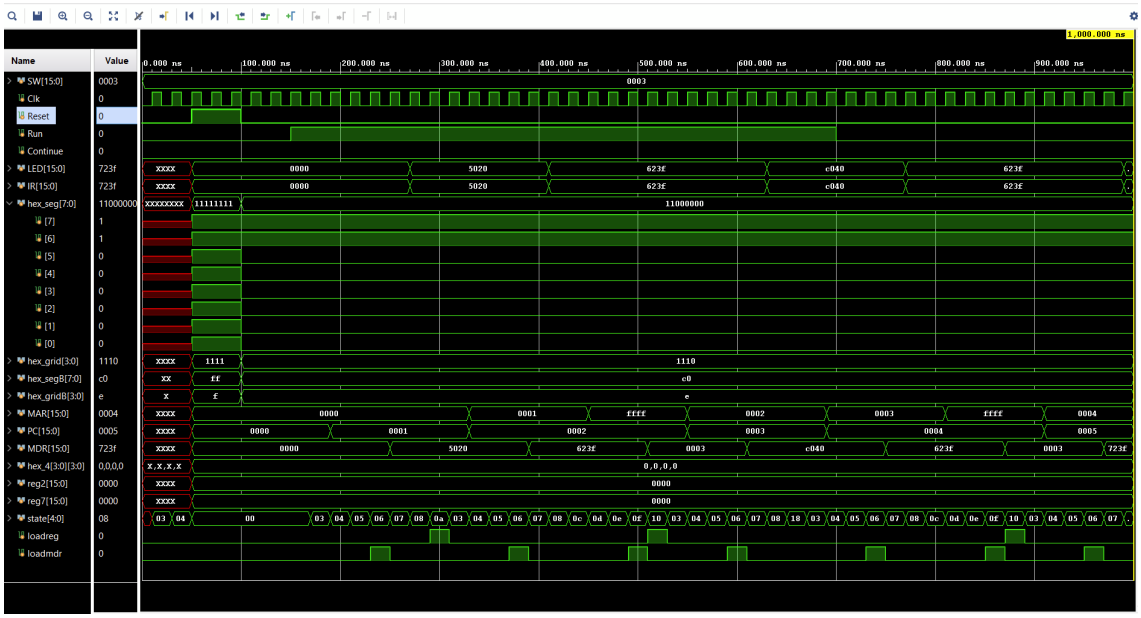
Ben_reg.sv: Contains the branch logic and controls the ISDU signals for ben and cc based on the IR[11:9] bit value. This is checked from 3 register NZP which store the negative,positive and zero value of the instruction from the bus MSB.

ISDU.sv: Controls all the signals for the components in the microprocessor and the load signals for the registers. Based on finite state machines and signals high or low determined by the states and which operations require which signals to function. The ISDU controls which state the processor is on based on the FSM and this controls all options in the diagram for when reset is pressed and the start state for each operand. Since there is no ready signal for the states, each state which requires this is extended for 3 states to account for the reading to memory for certain operations in the state. The ISDU includes this and proceeds operation as normal after these extended states are complete to account for the missing ready signal. Depending on the current state the correct control signals for those components were turned on and the gate select for the bus to make sure the operation runs smoothly and then turned off by default to account for latches and incorrect values going into the bus or muxes.



Updated FSM

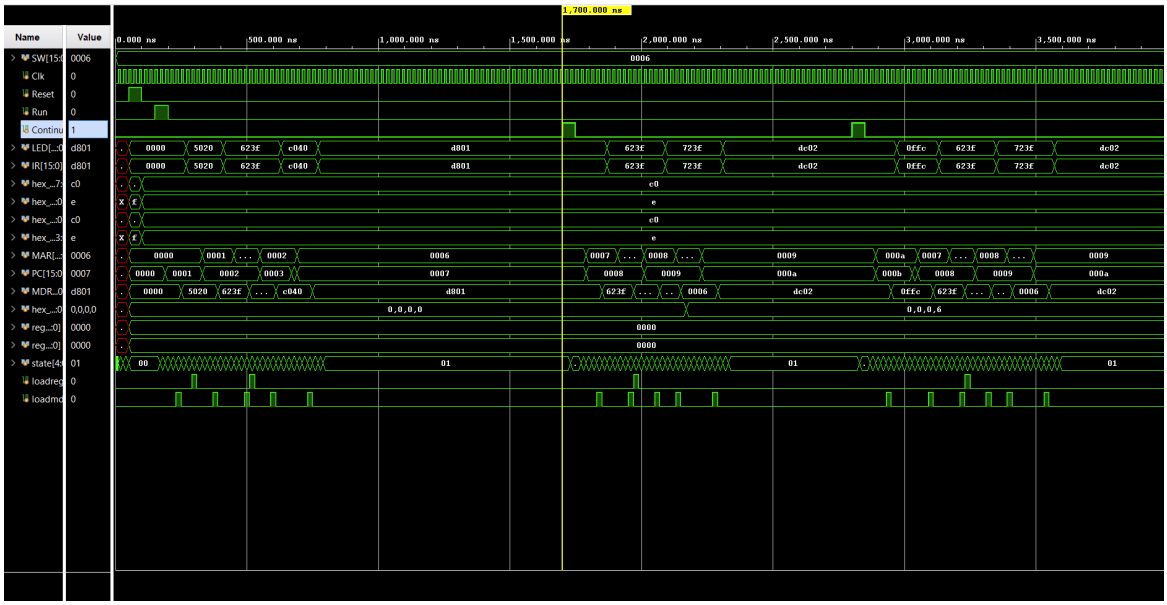
Simulations of 7 test Programs:



Basic I/O test 1

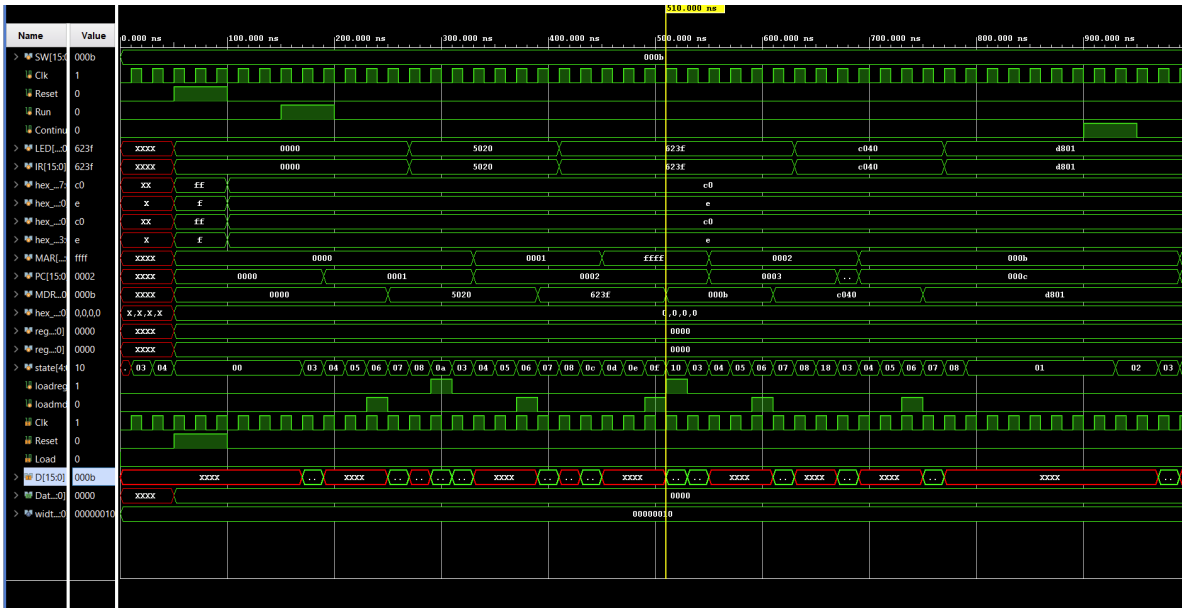
Reset if first set to high and then opcode value from switches are set and then Run is set to high
Switch value is set to x0003 and this goes through an unconditional loop with a BR statement

Hex drivers display the value of the switches



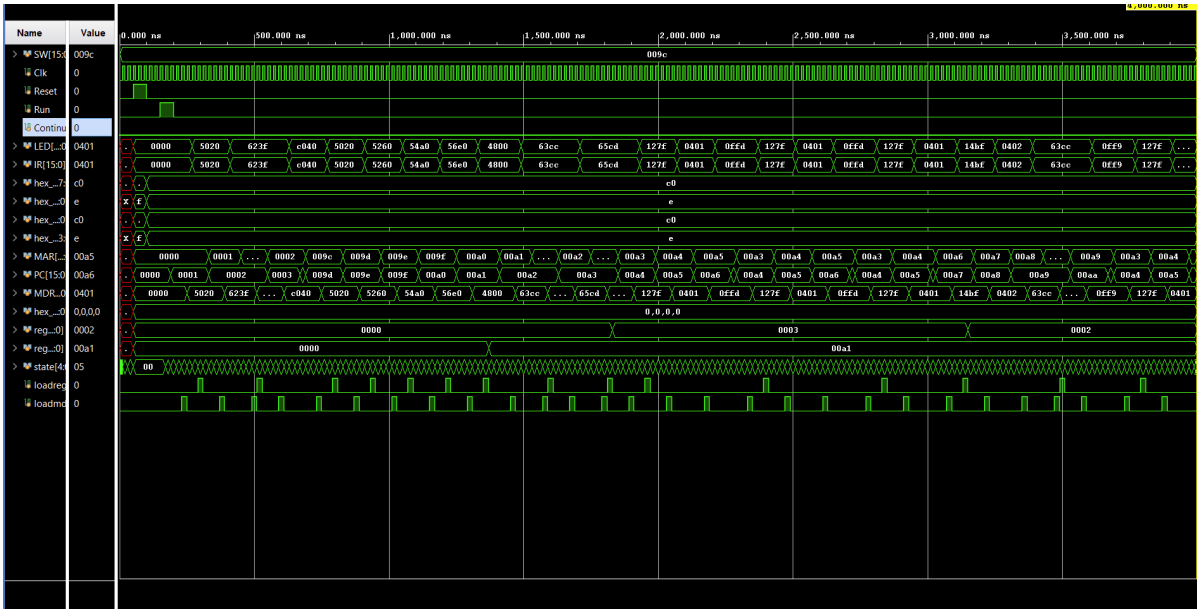
Basic I/O test 2:

Similar to test 1 but the value of the displays only changes to the switch values when Continue is pressed as shown in the sim when Continue is held at 0 the value of the LED remains the same and when set to high the value changes . There is one checkpoint for the the input and the second will display the output when COntinue is pressed



Self Modifying Code Test:

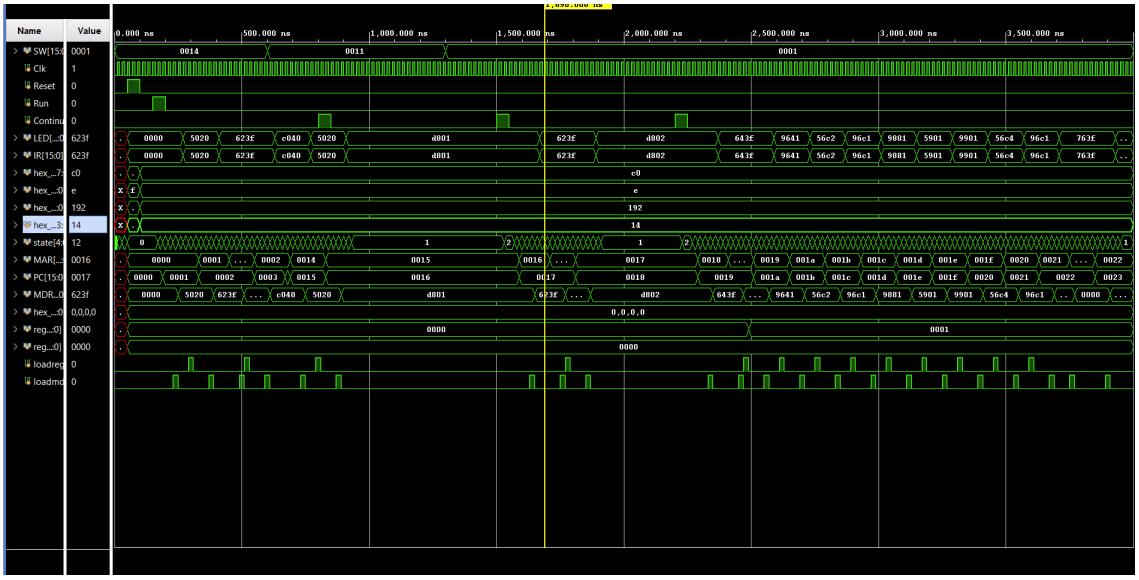
The purpose of this instruction is to set JSR to 0 and store the value of PC into R7 without changing the value of PC. THis is shown in the diagram where PC is incremented like normal while R7 is changed to the value of the switches and then each time incremented higher than the last iteration shown in R7 and the LED Display after each pause and when Continue is high.



Auto Counting Test:

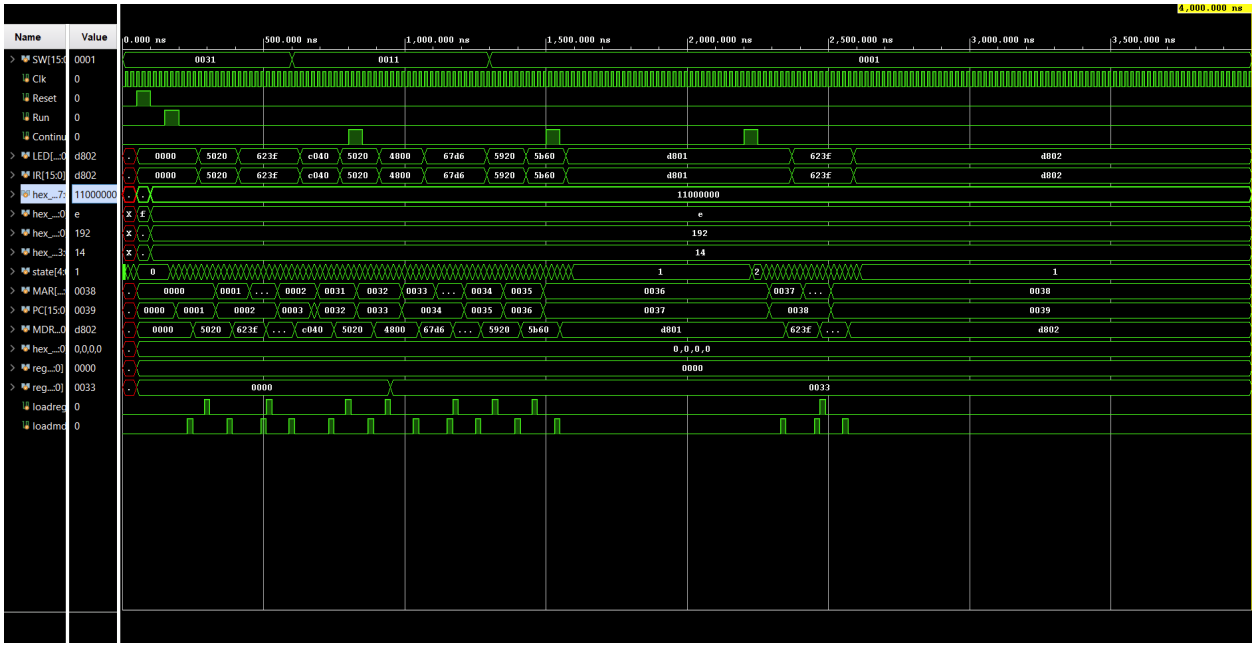
As shown in the simulation the LEDS are increasing while using R1 and R2 as a way to store counting variables and keep repeating ANDi instruction until reaching max value of display and

overflow. The delay between the counter loops is also shown in the simulation on the state controller and this delays the increments of the counter.



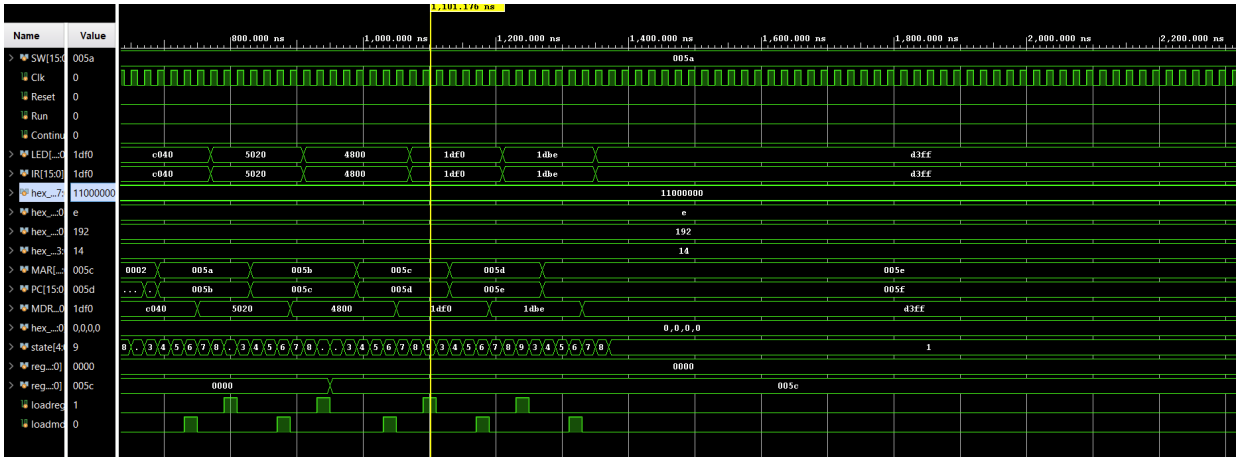
XOR Test:

First the two checkpoints take the value of the switches and XOR them together which is shown in the Simulation. With the use of 3 continues the value is stored on the hex displays. The XOR is performed by multiple AND and NOT instructions shown in the state controller and after the last continue the output is displayed.



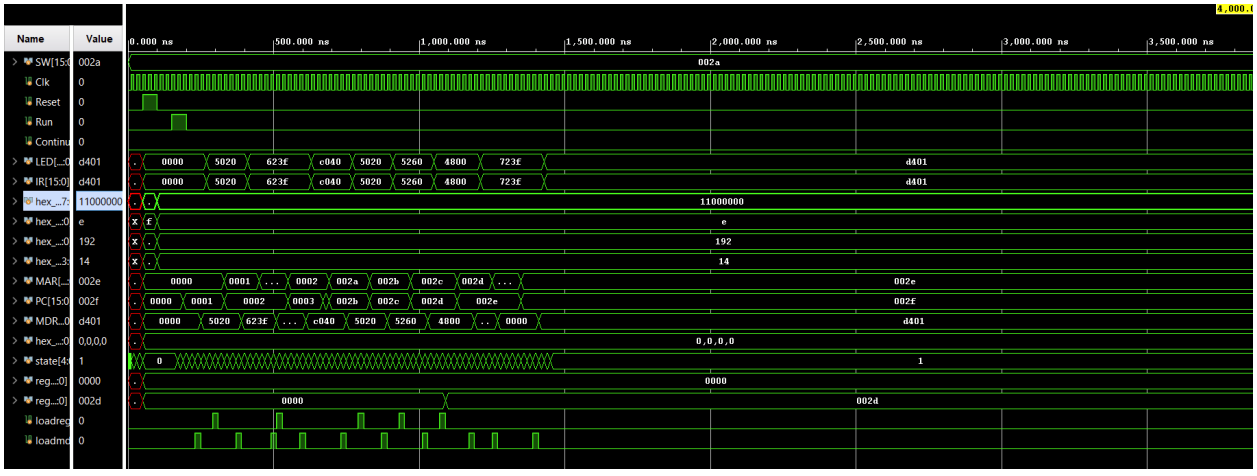
Multiplication Test:

Does the same thing as the previous test with 3 checkpoints and the first two take the values to be multiplied and the last one displays on the hex displays. This is shown in the simulation and after the operand is completed it will then repeat the process again from the first checkpoint. This is done with the add shift method.



Sort Test:

This test is divided into four parts which includes the data menu where a test is selected, which is then the input stage where the user can enter the data, then it will ask for the values on the switches at the checkpoint and at the end display the values. After each of the checkpoints the LEDS are updated and the list shown in the example is used and also shown on the simulation.



Act Once:

The purpose of this test is to act as a count every time COntinue is pressed starting from 0.

LUT	850
DSP	0

Memory BRAM	18332
Flip-Flop	275
Latches	23
Frequency	65.96 MHz
Static Power	75.54 mW
Dynamic Power	0mW
Total Power	85.63 mW

Post Lab Questions:

What is MEM2IO used for, i.e. what is its main function?

This is the interface between the FPGAs memory and the switches. It is a link between both and provides access to the memory from the board. Its main function is to write values to the hex segment displays and important for LDR and STR to have access to memory and load and write values to each.

What is the difference between BR and JMP instructions?

When both are used, the PC is changed but in different ways. When JMP is used the PC is used as a subroutine and R7 is used to store the previous address where JMP was used to return back after execution. The BR instruction changes PC is only due to NZP signal and may or may not be changed depending on the value. This instruction is used for looping and jumping to certain sections of the code.

What is the purpose of the R signal in Patt and Patel? How do we compensate for the lack of the signal in our design? What implications does this have for synchronization?

The signal is used as the Ready Signal and helps indicate when memory access has been completed between MDR and MAR or other operations. It indicated the data transfer is ready for computation and since the R signal is not available in the slc3 this is used with wait states to hold the in that specific state until memory transfer is complete and continue on. This affects synchronization because all state switches require the R signal to wait for the same time making the process asynchronous in this way which affects the synchronous method used in LC3.

Conclusion:

The overall functionality of this design was to mimic the LC3 but make a smaller version which can perform various functions using different registers and computation and memory access throughout each operation. Some struggles was bus inputs and making sure which drivers would change each, especially the register file which was implemented many ways but the last way chosen was to have all of them loaded in one file and declare functionality there instead of in the CPU. The main thing was not having a datapath which made the CPU clustered and all instantiations were there making it hard to see which registers or muxes were affected by what. This was fixed with some commenting and aligning it due to operations and tracking each of the opcodes to see what they affect. The most difficult thing was the ISDU and making sure this was correct before any operations were performed because some signals were wrong or not called correctly when they were supposed to be fixed and spent the most time debugging. This was a good lab to understand the LC3 better and familiarize ourselves with Systemverilog and a good project to understand many components of this language.