ECE 385, Spring 2023

Lab Report 5: Simplified LC3

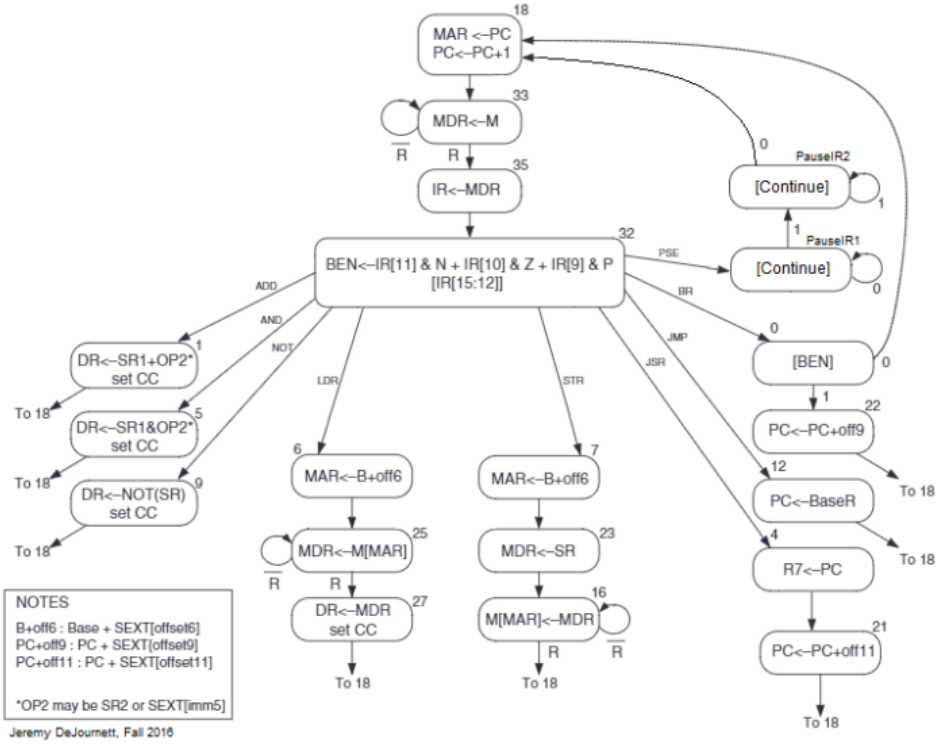
Partners: adnanmc2, akshat4
TA: Shitao Liu
Section: SL

1. Introduction:

Lab 5 consisted of two parts which culminated in the creation of a simplified microprocessor in the form of LC3. Designing this lab called upon all prior concepts in logic circuits and SystemVerilog. It was also the first lab that made use of memory, using a simulated version for ModelSim and the FPGA's onboard memory for the actual demonstrations.

The subset of the LC3 instruction set architecture implemented in this lab will be referred to as SLC3. Lab 5.1 simply involved the creation of the fetch-execute cycle, which uses states 18, 33 and 35. PC is a register that stores the address of the line being executed, MAR stores the value of PC in state 18, the memory content of the address is stored to MDR in state 33 and the instruction register (IR) receives this memory content in state 35. The value in IR is usually the opcode, which is a 16-bit binary number that tells the system which instruction to execute.

LC3 STATE DIAGRAM FROM APPENDIX C OF PATT AND PATEL



As we can see in the shown state diagram, 16 other states have to be implemented for the full functionality of the LC3 microprocessor. All the instructions made possible by these states alongside their opcodes and parameters are listed in the following table.

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

Each 16-bit [15:0] opcode can be broken up into segments that signify important information for each instruction. For example, for ADD, [15:12] holds the opcode, [11:9] indicates the destination register, [8:6] indicates source register 1, [5] indicates whether addition is occurring between two registers or a register and a sign extension of IR[4:0], [4:3] is redundant and [2:0] indicates source register 2. These terms and their explanations will be emphasized upon later, but this example serves to explain how an instruction works. An entire datapath of logic components was created in order to give life to the abstract ideas of the LC3 microprocessor (figure 1). This datapath mainly includes registers and MUXes alongside a BUS wire and larger modules with many internal components such as the control unit, MEM2IO, ALU and SRAM.

Tri-state buffers cannot be implemented in SystemVerilog, so we used conditional statements in our datapath file. Tri-state buffers are crucial as only 1 of 4 can release the internally incoming data to the BUS at a given time: GateALU, GatePC, GateMAR or GateMDR. This is known as **one hot encoding**. Sign extensions and minor incrementing was done within the arguments of SystemVerilog modules themselves, eliminating the need for more logical units and layers of abstraction.

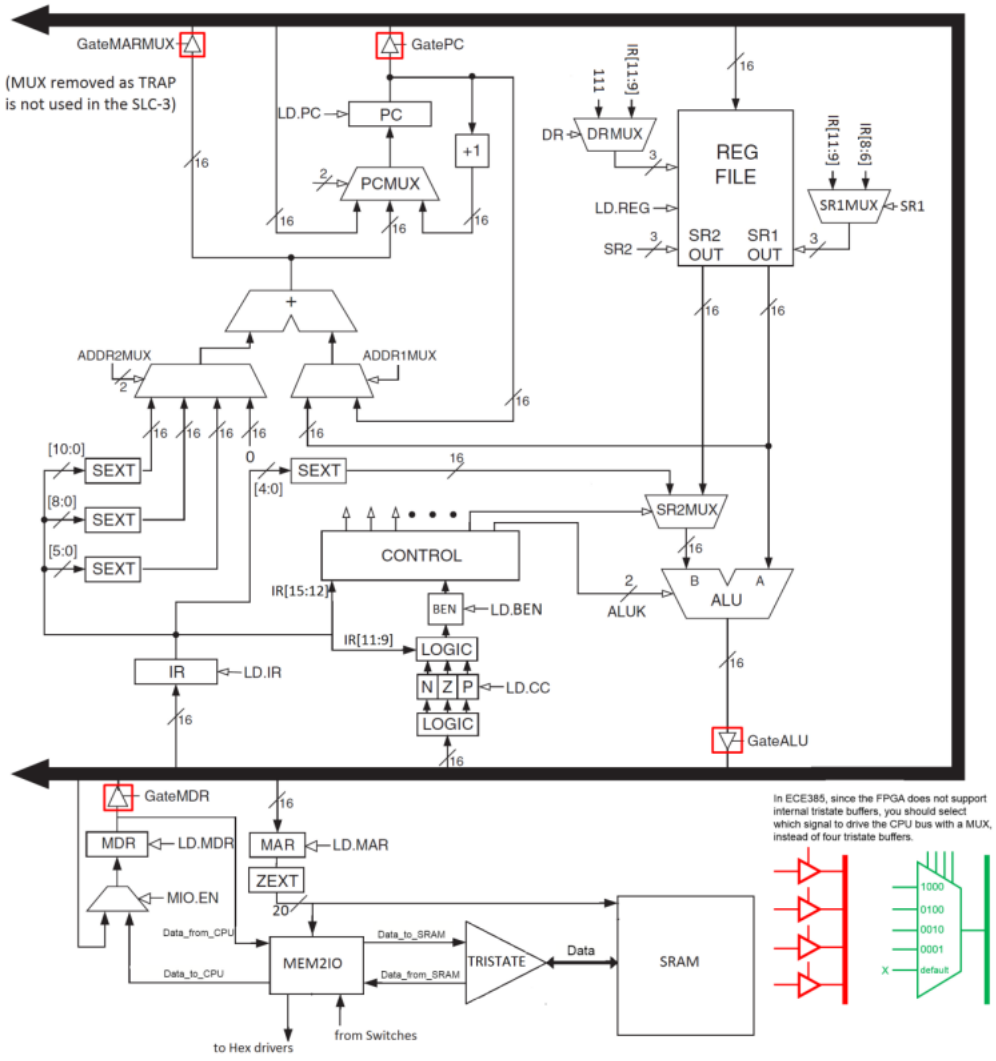
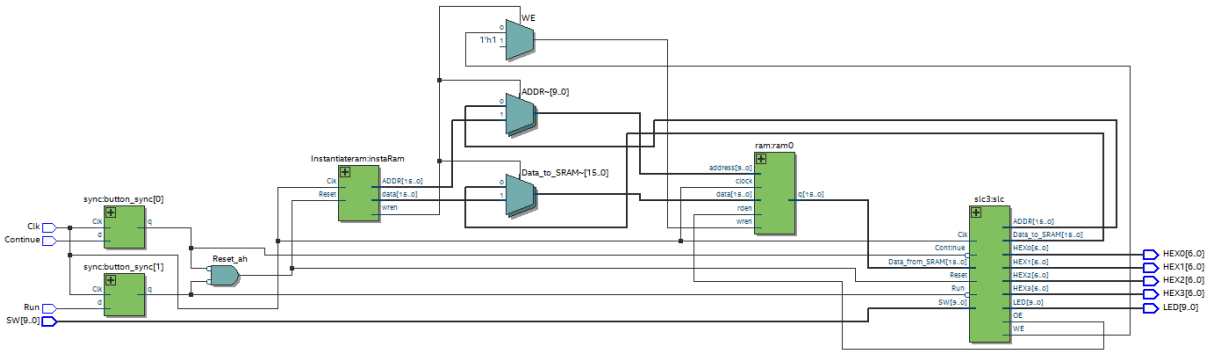


Figure 1: The physical SLC3 datapath

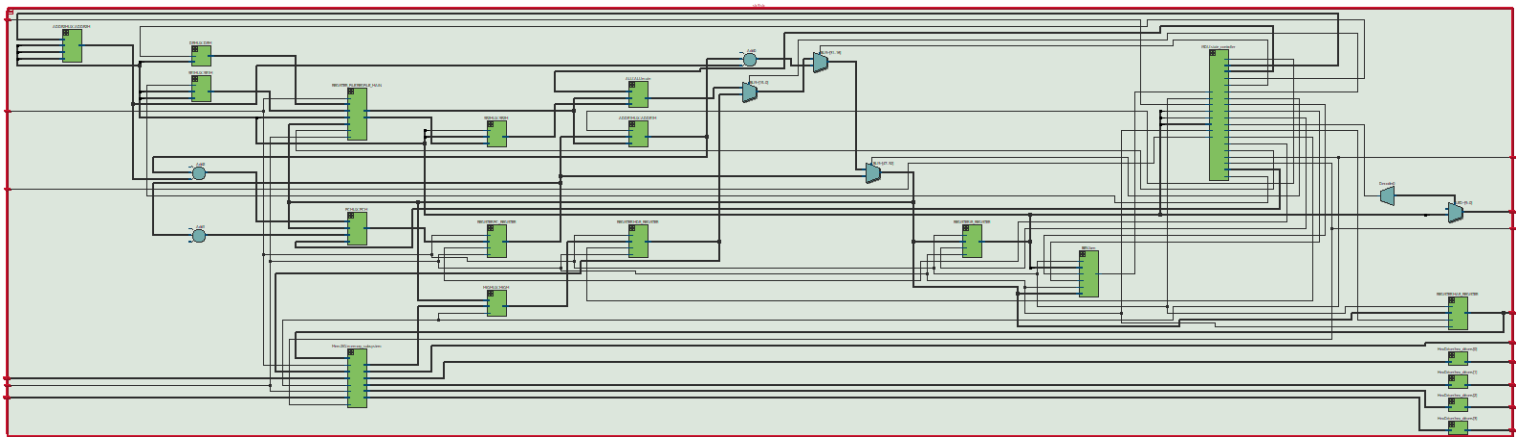
The MEM2IO unit is another crucial piece of Lab 5. It acts as the memory mapped I/O interface to enable communication between FPGA switches, SRAM, hex drivers and internal addresses/memory content. It is what allows for the results of instructions to appear on the FPGA and for opcodes to be loaded into IR through the fetch-execute cycle. The main parameters for MEM2IO which we had to control in our state machine were OE (read enable) and WE (write enable). Either of these signals could be set to a 1 at a time, performing the memory operations indicated by their names. The working of this unit is described in detail in the post-lab section of the report.

c. Block diagrams from Quartus Prime:

a. Top-level diagram for entire system:



b. Block diagram of slc3.sv:



d. Written description of every .sv module:

slc3_testtop.sv

Module: slc3_testtop
Inputs: [9:0] SW, Clk, Run, Continue
Outputs: [9:0] LED, [6:0] (HEX0, HEX1, HEX2, HEX3)
Purpose: Reading through the section for RTL simulations and modelsim traces paints a detailed picture of how this module's parameters work but it basically acts as the **top level entity** for calls upon the sync and slc3 module

SLC3_2.sv

slc3.sv

Module: slc3
Purpose: Acts as the primary module within which other modules such as MEM2IO, registers, MUXes, BEN and more are instantiated
Inputs: [9:0] SW, Clk, Reset, Run, Continue, [15:0] Data_to_SRAM
Outputs: [9:0] LED, OE, WE, [6:0] (HEX0, HEX1, HEX2, HEX3), [15:0] ADDR
Description: Sets all signals and controls MEM2IO via the switch input SW alongside handling internal connections such as Data_to_SRAM and containing the **one hot** logic for GateALU, GatePC, GateMARMUX and GateMDR.

REGs.sv

Module: REGISTER
Purpose: Just a template for a 16-bit register
Inputs: Clk, Reset, Load, [15:0] D
Output: [15:0] Data_Out;
Description: Loads values within D to Data_Out at every rising edge of Clk and clears register if Reseet goes high

MUXes.sv

Module: PCMUX
Inputs: [1:0] select_bits, [15:0] (incremented_PC, BUS_input, ADDED_addr_input)
Output: [15:0] output_PCMUX
Purpose: Serves as the PCMUX in the SLC3 datapath
Description: Depending on the select_bits case, it sets either incremented_PC, BUS_input or ADDED_addr_input as output_PCMUX

Module: ADDR1MUX
Inputs: ADDR1MUXselect, [15:0] (input_PC, input_SR1)
Output: [15:0] output_ADDR1MUX
Purpose: Serves as the ADDR1MUX in the SLC3 datapath
Description: Depending on the ADDR1MUXselect case, it sets either input_PC or input_SR1 as output_ADDR1MUX

Module: ADDR2MUX
Inputs: [15:0] (input_IRSEXT6, input_IRSEXT9, input_IRSEXT11), [1:0] ADDR2MUXselect
Output: [15:0] output_ADDR2MUX
Purpose: Serves as the ADDR2MUX in the SLC3 datapath
Description: Depending on the ADDR2MUXselect case, it sets either 16'h0000, input_IRSEXT6, input_IRSEXT9, input_IRSEXT11 as output_ADDR2MUX

Module: MIOMUX
Inputs: [15:0] (BUS_input, Data_CPU_In), MIO_EN
Output: [15:0] output_MIOMUX
Purpose: Serves as the mux for the MDR register in the SLC3 datapath
Description: Depending on the MIO_EN case (which is set by the MEM2IO unit in the slc3 module), it sets either BUS_input or Data_CPU_In as output_MIOMUX

Module: DRMUX
Inputs: [2:0] input_IR11to9, DRMUXselect
Outputs: [2:0] output_DRMUX
Purpose: Serves as the mux for the destination register on the SLC3 datapath
Description: Depending on the DRMUXselect case, it sets output_DRMUX as input_IR11to9 ot 3'b111

Module: SR1MUX
Inputs: [2:0] (input_IR11to9, input_IR8to6), SR1MUXselect
Outputs: [2:0] output_SR1MUX
Purpose: Serves as the mux for source register 1 on the SLC3 datapath
Description: Depending on the SR1MUXselect case, it sets output_SR1MUX as input_IR11to9 or input_IR8to6

Module: SR2MUX
Inputs: [15:0] (input_SR2, input_IRSEXT5), SR2MUXselect
Outputs: [2:0] output_SR2MUX
Purpose: Serves as the mux for source register 2 on the SLC3 datapath
Description: Depending on the SR2MUXselect case (which is instantiated as IR[5] in the slc3 module, it sets output_SR2MUX as input_SR2 or input_IRSEXT5

Memory_contents.sv

Module:

Mem2IO.sv

ISDU.sv

Module: ISDU
Inputs: Clk, Reset, Run, Continue, IR_5, IR_11, BEN, [3:0] Opcode
Outputs: LD_MAR, LD_MDR, LD_IR, LD_BEN, LD_CC, LD_REG, LD_PC, LD_LED, GatePC, GateMDR, GateALU, GateMARMUX, [1:0] (PCMUX, ADDR2MUX, ALU), DRMUX, SR1MUX, SR2MUX, ADDR1MUX, Mem_OE, MEM_WE
Purpose: Serves as the control unit for the SLC3 microprocessor, specifying all state transitions and signals for the system to “understand” how and when to carry out each instruction
Description: The state machine starts when Run goes high and other transitions are either unconditional, depend on Continue/Reset or Opcode for “decode” - in the states for each instruction, the LD and MUX signals are specified (these include MEM_OE and MEM_WE, which will tell MEM2IO whether to **read** or **write** from memory in the slc3 module

Instantiateram.sv

testbench.sv

test_memory.sv

Module: test_memory
Inputs: Reset, Clk, [15:0] data, [9:0] address, rden, wren
Outputs: [15:0] readout
Purpose: This module acts as the SRAM IC of the DE-10 FPGA board for RTL simulations. A connection is established between this memory module and the computer using a specific top-level entity.

synchronizers.sv

ALU.sv

Module: ALU
Inputs: [1:0] ALUcontrolpins, [15:0] (A, B)
Outputs: [15:0] ALU_output
Purpose: Acts as the ALU unit on the SLC3 datapath
Description: Sets ALU_output depending on the ALUcontrolpins case: 2'b00 for A + B, 2'b01 for A AND B, 2'b10 for NOT A, 2'b11 for passing A

REG_FILE.sv

Module: REGISTER_FILE
Inputs: [15:0] input_fromBUS, [2:0] (DR_info, SR1_info, SR2_info), reset_registerfile, Clk, load_register_file
Outputs: [15:0] (SR1OUT, SR2OUT)
Purpose: Holds 16-bit registers R0 through R7 as temporary storage for fundamental operations in the SLC3 microprocessor
Description: Clears all registers if reset_registerfile is 1 - otherwise, if load_register_file is 1, it loads the relevant destination register with input_fromBUS depending on the case for DR_info and accesses data from the relevant source register depending on the case for SR1_info or SR2_info

BEN.sv

Module: BEN
Inputs: LD_CC, LD_BEN, Clk, Reset, [2:0] IR, [15:0] bus_in,
Output: Dout

Purpose: Sets NZP on the SLC3 datapath for use in the BR instruction to compare register values and set the sign for register contents or arithmetic operations
Description: Depending on whether LD_CC or LD_BEN are high, it updates Dout with internal logic concerning the n (negative), z (zero), p (positive) indicator bits and IR [2:0] at the rising edge of the Clk signal

e. Description of Instruction Sequence Decoder Unit:

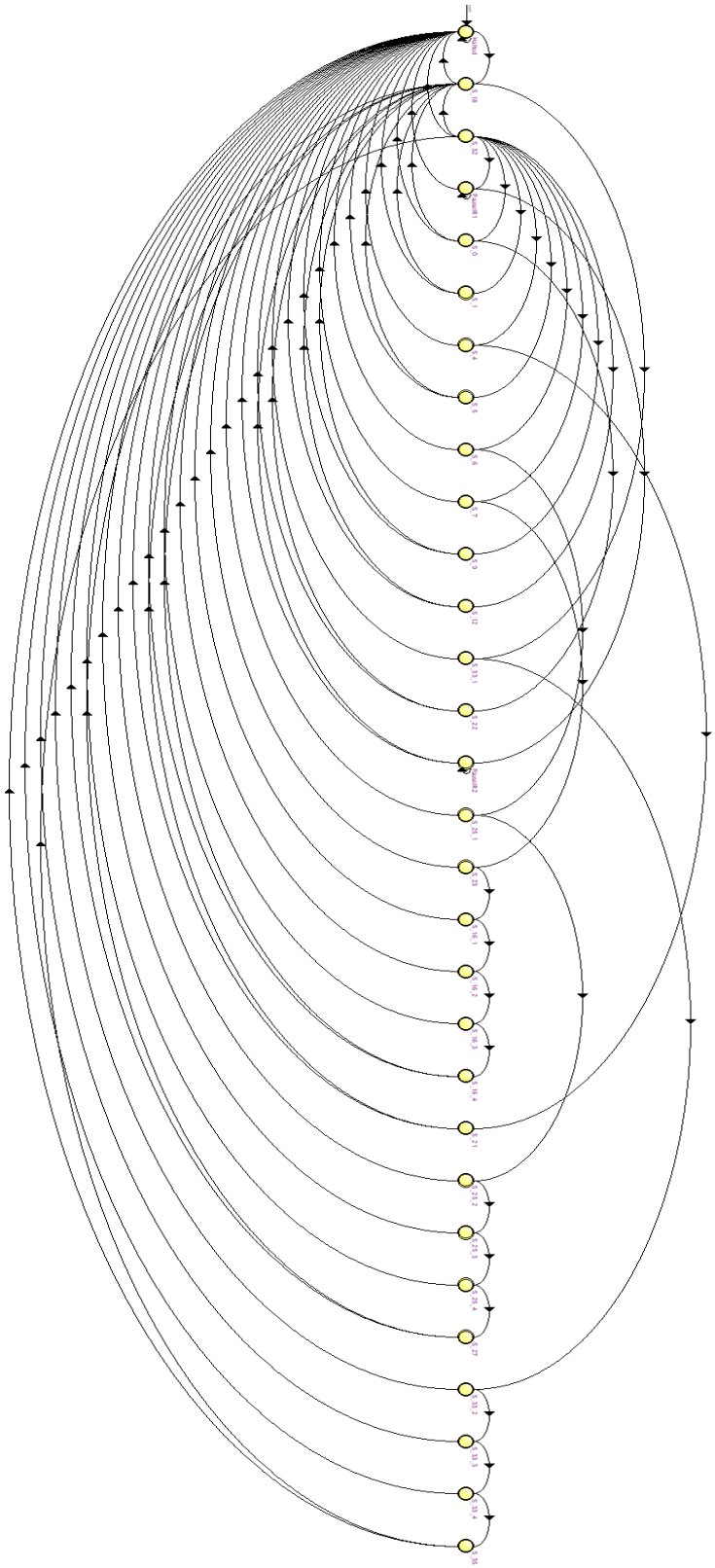
The ISDU is the driving factor of the LC3 microprocessor. All state transitions and signals are specified in this module, enabling the system to “understand” how and when to carry out each instruction. The traditional LC3 implementation makes use of an “R” signal to indicate whether memory is ready to be accessed, which is seen in states 33, 25 and 16 where as long as the memory is not ready, no state transition can occur. Since we were not able to generate an “R” signal from the SRAM unit, we created 4 wait states for each of these states: 33_1, 33_2, 33_3 and 33_4 for example, after which we transition unconditionally to 33. This ensures that there is enough time for the memory to be ready and **4** wait states give the FPGA’s onboard memory 4 clock cycles to get ready.

Going over each and every state would be tedious, but explaining the function of states 18, 33 (1 to 4), 35 and 32 in that sequence paints a clear picture of the ISDU’s working. Starting from the “halted” state, we move to state 18 when Run = 1. In 18, GatePC = 1 to release the value of PC onto the BUS, LD_MAR = 1 to load MAR with PC, PCMUX = 00 to select the incremented PCMUX input and LD_PC = 1 to load the PC register. We then move through 4 cycles of state 33, where MEM_OE = 1 throughout in order to access memory and MDR = 1 in 33_4 to load the MDR register with M[MAR]. Next, we move to state 35 where GateMDR = 1 to release MDR’s contents onto the BUS and LD_IR = 1 to load the IR register with the instruction from the memory content. This is the “fetch” cycle. Next, we move to state 32 which is “decode”: the transition from here depends on the opcode (through case statements). LD_BEN = 1 in “decode”, setting NZP in the BEN module on the datapath.

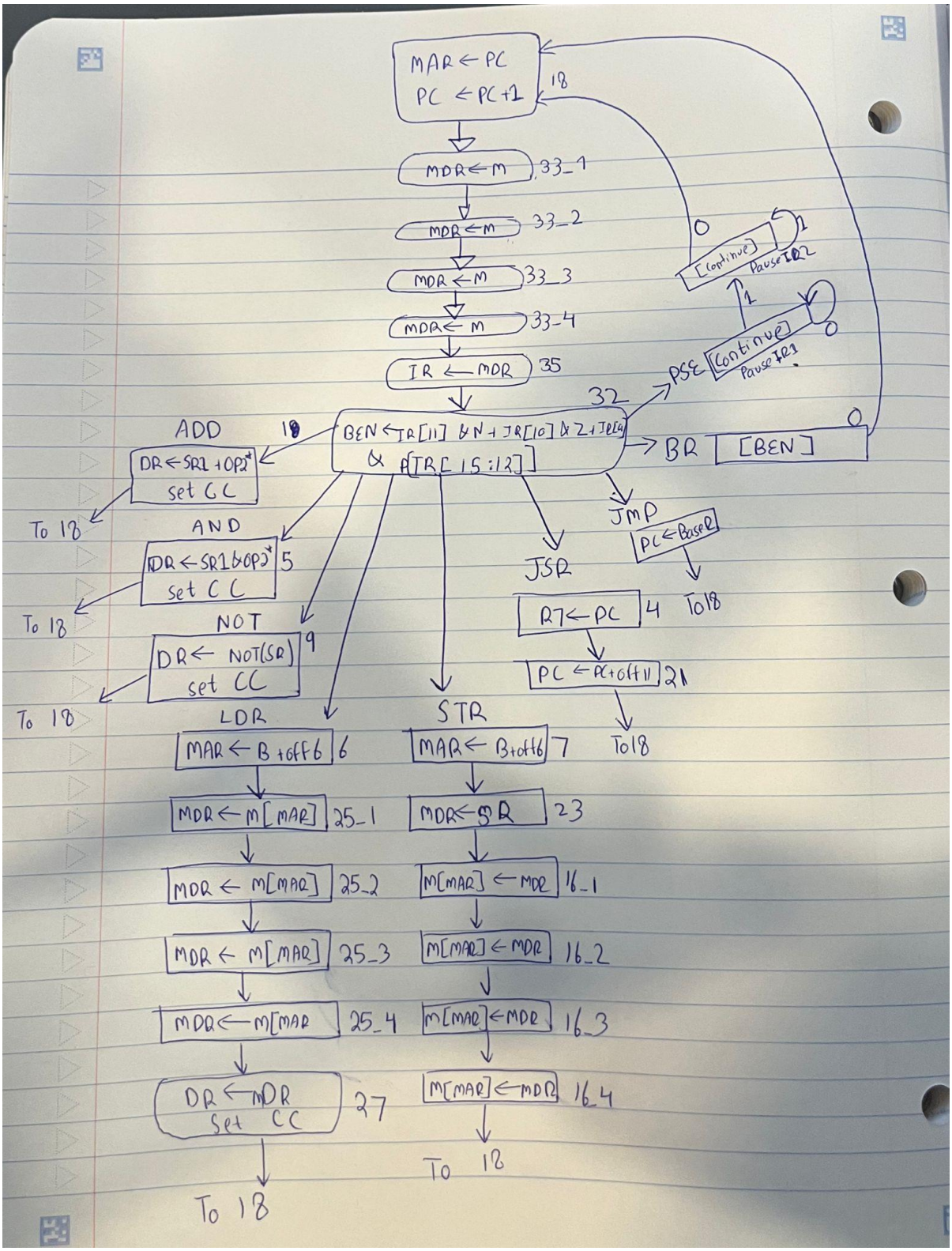
As demonstrated in the explanation, each state deals with the select bits for various MUXes alongside load bits for different registers among other information.
The PSE instruction takes us to PauseIR1, where we remain until Continue goes high again. This causes an immediate transition to PauseIR2, which prevents us from moving back to state 18 until the Continue signal goes back down. This simple precaution prevents **multiple runs of the instruction** in the tiny interval of time over which the FPGA input button for “Continue” is pressed and released.

f. State diagram from ISDU:

1. **Diagram produced by Quartus Prime (without annotations):**



2.Quartus Prime diagram supplemented with handwritten annotations (in case the states aren't clear enough on the computer generated version):



3. **Simulations of SLC-3 instructions:**

```
module HexDriver (input  logic [3:0]   In0,
                  output logic [6:0]   Out0);
   always_comb
   begin
      unique case (In0)
         4'b0000   : Out0 = 7'b1000000; // '0' 40
         4'b0001   : Out0 = 7'b1111001; // '1' 79
         4'b0010   : Out0 = 7'b0100100; // '2' 24
         4'b0011   : Out0 = 7'b0110000; // '3' 30
         4'b0100   : Out0 = 7'b0011001; // '4' 19
         4'b0101   : Out0 = 7'b0010010; // '5' 12
         4'b0110   : Out0 = 7'b0000010; // '6' 2
         4'b0111   : Out0 = 7'b1111000; // '7' 78
         4'b1000   : Out0 = 7'b0000000; // '8' 0
         4'b1001   : Out0 = 7'b0010000; // '9' 10
         4'b1010   : Out0 = 7'b0001000; // 'A' 8
         4'b1011   : Out0 = 7'b0000011; // 'b' 3
         4'b1100   : Out0 = 7'b1000110; // 'C' 46
         4'b1101   : Out0 = 7'b0100001; // 'd' 21
         4'b1110   : Out0 = 7'b0000110; // 'E' 6
         4'b1111   : Out0 = 7'b0001110; // 'F' E
         default   : Out0 = 7'bX;
      endcase
   end

endmodule
```

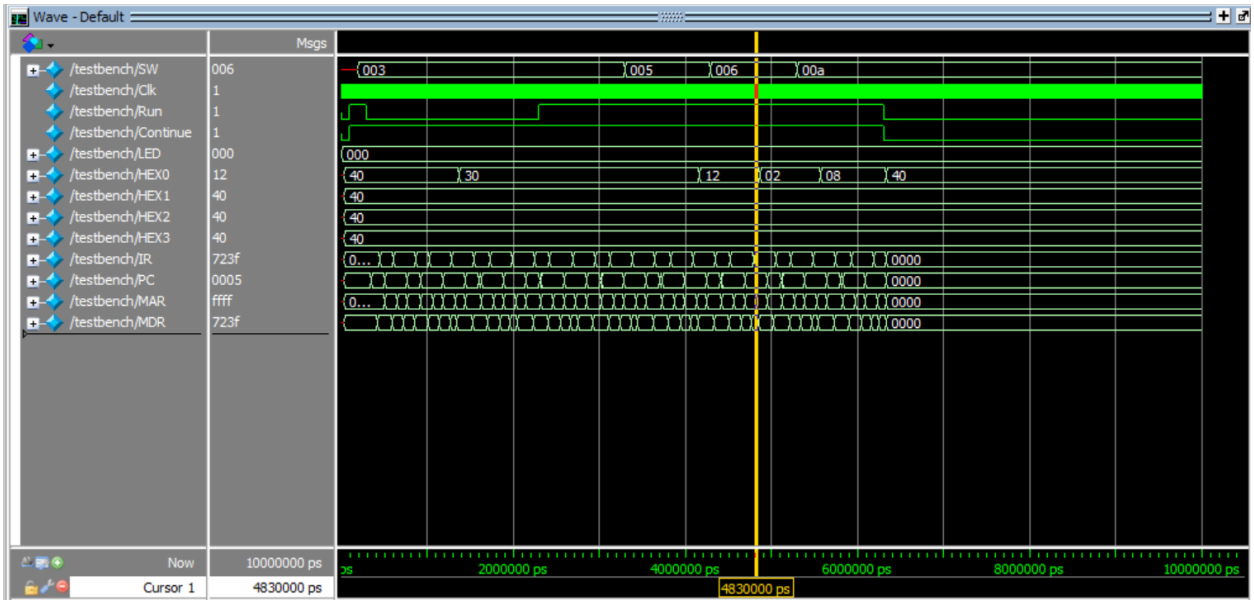
It is important to note that to understand the modelsim testbench simulations, one must know what the simulated hex values represent. In the hexdriver module, these initiations were made that show how each hex value in this module corresponds to the hex value which would be seen on the DE-10 board. For all test cases the first step of execution is as follows:

- Load the Program Start address into SW (switches) and set Run high momentarily. Then, for all test cases except Basic I/O 1, the remaining executions depend on pulses of the Continue signal.
- Once done, in order to ready system for the next Program Start address, reset the system by briefly setting Run and Continue both to high

Clarifications to improve understanding:

- All “high” signals appear as “low” on simulations since they are active low.
- The Program Start addresses are specified in the test_memory module and cause the execution of a series of opcodes in our SLC3 microprocessor.
- The SW input in the top level file used for RTL simulations only has 10 bits [9:0] whereas in the DE-10 board, 16-bit switch values can be loaded.

Test case 1: **Basic I/O 1**

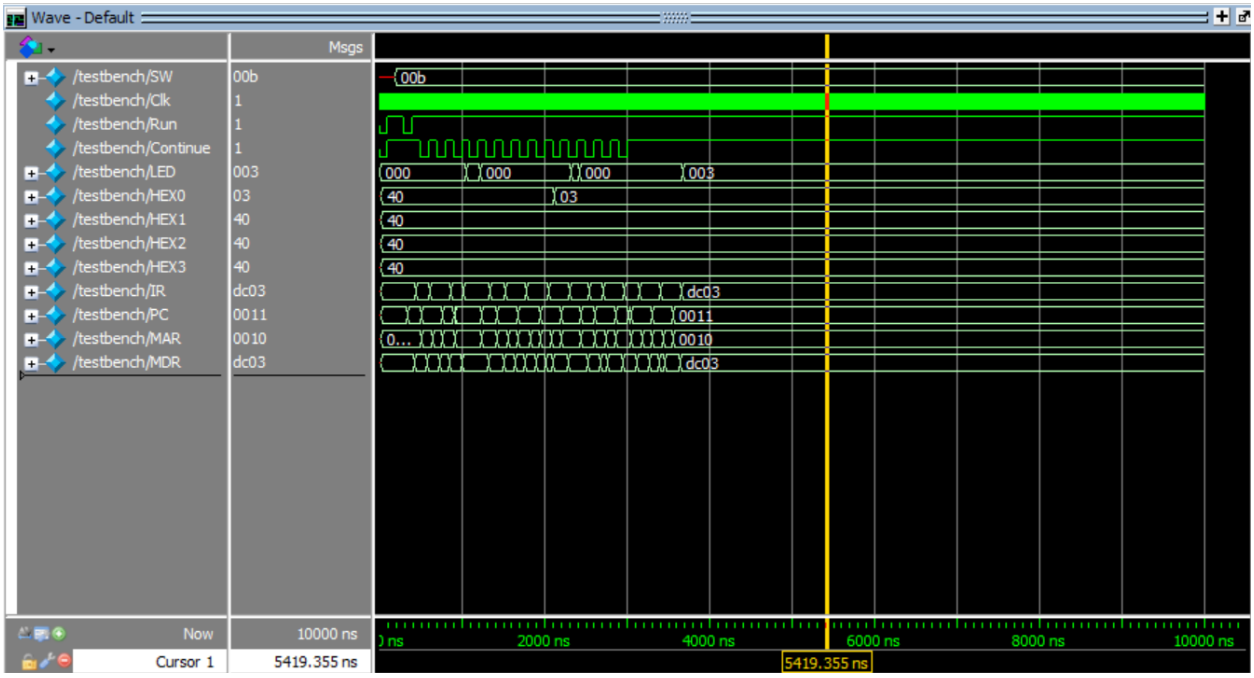


The PSA for this instruction is x003. Once Run is hit, HEX0, HEX1, HEX2, and HEX3 (6, 7, 8, 9: from least significant to most significant) will display whatever value is loaded in the switches (1). As an example, in the hex values (9 to 6), we can see that x005 gets loaded as “40” “40” “40” “12” which corresponds to x0005 in our hexdriver initiations. At the indicated line, a reset is done so all 4 hex values read “40” which corresponds to a 0 as per our hexdriver implementation.

Test case 2: **Basic I/O 2**

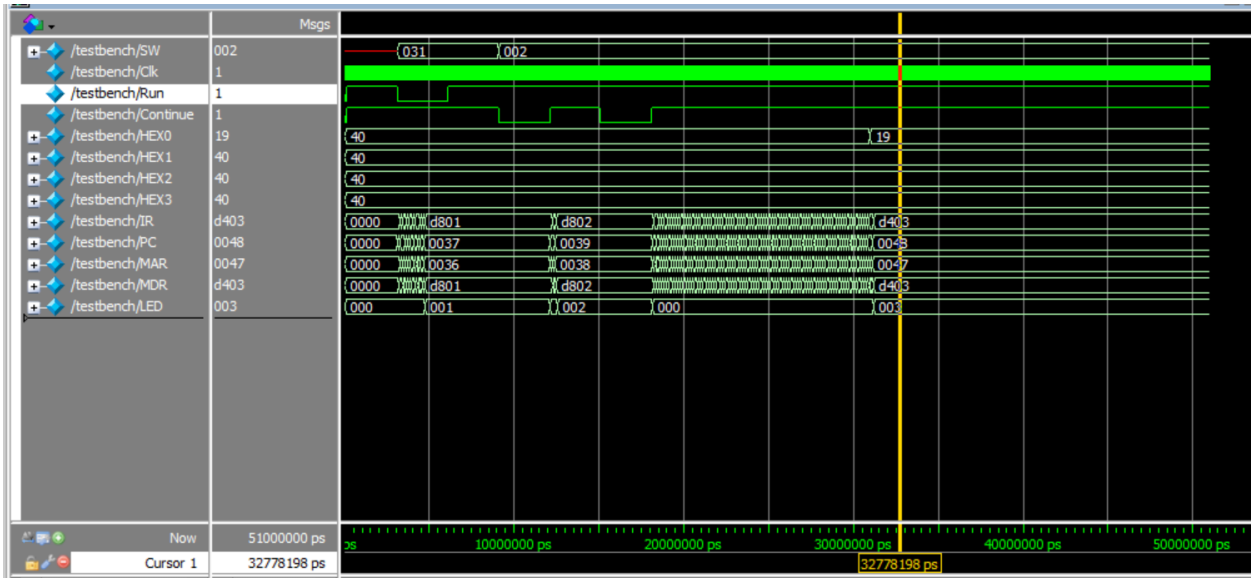
The PSA for this instruction is x003. Once Run is hit, the system is ready to accept inputs as seen by the LED (13) on the board being set to 001. Each time Continue is pressed, the value on the switches gets loaded to the hex values which appear in accordance with the initiations in hexdriver module (within the circled region, x204 gets loaded to 8,7,6 and 5 as “40” (0), “30” (2), “40” (0) and “19” (4). At the indicated line, a reset is done so all 4 hex values read “40” which corresponds to a 0 as per our hexdriver implementation.

Test case 3: **Self-modifying code**



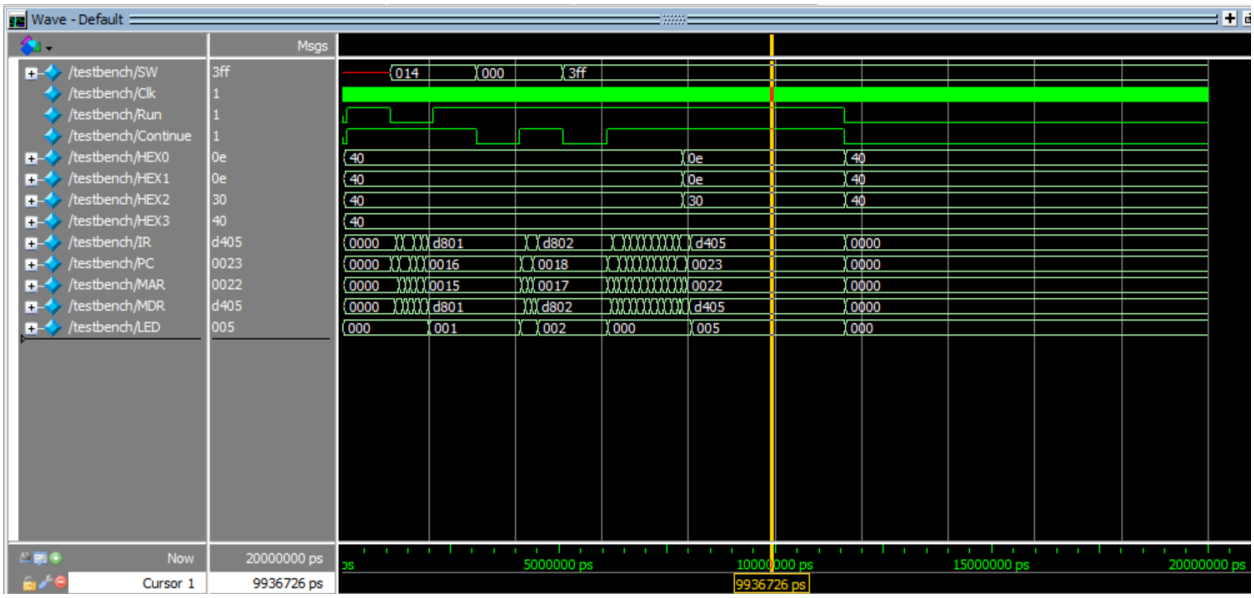
Once the PSA of x00B is loaded to the switches and run is hit, continue should go high. This starts the test case. The hex drivers are irrelevant for this test case as 9, 8, 7 and 6 read “40” “40” “40” “03”, which is x000B as it’s the only switch input. The relevant aspect of this case is the LED (5) trace, which increments by 1 with each continue that is pressed. Essentially, the LEDs on the DE-10 board will represent binary addition where the lit bits are 1s and unlit bits are 0s. The x000s are appearing as the LEDs are only being displayed during each pause state (LD.LED = 1 only in the pause states within ISDU). This specific case only goes up till 3 as some of the Continue pulses may not have registered within the runtime of the simulation trace.

Test case 3: **Multiply test**



Once the PSA (x031) is loaded and Run is hit, the value x002 is loaded on the switches and Continue is hit (first LED (9) comes on, indicated by 001). Then, x002 is loaded again with a second hit of Continue (indicated by 002 in the LED's trace). The result is within the encircled region, with 8,7,6 and 5 displaying ""40" "40" "40" "19" – corresponding to x0004. No reset is shown for this test case.

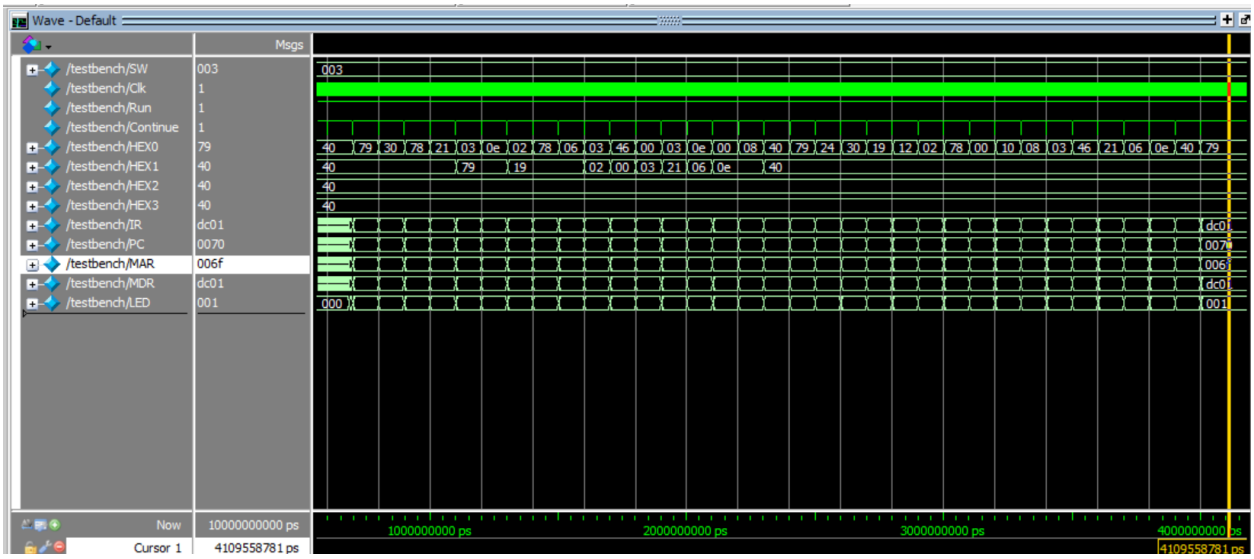
Test case 4: **XOR test:**



Once the PSA (x014) is loaded and Run is hit, the value x0000 is loaded on the switches and Continue is hit (first LED (9) comes on, indicated by 001). Then, x03FF is

loaded again with a second hit of Continue (indicated by 002 in the LED (9) trace). This XOR operation should evaluate to. The result in the boxed region, with 8,7,6 and 5 displaying “40” “30” “0e” “0e” – corresponding to x03FF. This is clearly correct as the XOR of an input with 0 returns the input. The blue line indicates a reset.

Test case 5: **Sort test**



The RTL trace for this is taken from a middle segment to show how the hex values (5,6,7 and 8) update. Once PSA = x005A is loaded and Run is executed, there are 3 possible instructions that can be executed upon loading the switches and hitting Continue: x0001 will call the “data entry” function, entering x0002 will call the “sort” function, and entering x0003 will call the “display” function. We simply look at the “display” section of the RTL trace (as indicated by x002 in SW) to observe that the pre-loaded values held by the hex values (5,6,7 and 8) have been sorted within the encircled block. This is the most comprehensive test, making use of every instruction that can be performed by our SLC3 microprocessor.

4. **Post-lab Questions:**

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

The main function of MEM2IO is to interact with the I/O of the DE10 Lite board and provide a way of interacting with memory mapped addresses. It manages all I/O with the DE10-Lite physical I/O devices, namely, the switches and 7-segment displays. Both the devices have the same address but as both of them have different functions, they can share the same address. It detects the load address 0xFFF as a load from switches instead of SRAM, further it also detects to store the address on the Hex displays rather than on storing on the SRAM.

- **What is the difference between BR and JMP instructions?**

Both these instructions have different opcodes to distinguish between the functionality it serves in LC3. The BR stands for ‘Branch’ instruction and serves more like a if/else statement in LC-3, depending on the condition codes set for ‘n, z, p’ and branching to a different section of the code if these condition codes are met. These condition codes are set depending on the instruction code of the previous statements. The JMP stands for ‘Jump’ instruction and serves as an indirect jump to a different section of the code regardless of the condition 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 purpose of the R signal is a type of a control signal that is used to make sure the system is ready to read or write or perform another function. To compensate for this lack of signal we add 4 more states in our design, which act like 4 clock cycles to ensure that memory is ready to perform the function. By adding states, we make sure that we do not need synchronizers in our design.

5. Conclusion:

The whole lab was really interesting to start with as we were implementing everything we had learnt from our freshman year in UIUC and applying and building the whole LC-3 architecture. We made sure to plan our implementation which worked out as we didn't have to debug much at the end of the lab. We faced minor issues in ISDU and the R control signals, but we were able to fix that by increasing the number of wait states and making sure our ISDU was written properly and tracing out the datapath for each and every instruction set. The instruction for the lab was well written but the work distribution between the 2 parts of the lab could be better as the first part hardly took 2 hours but the second part comparatively tool a much larger time commitment.