EXPERIMENT Report #6

Simple Computer SLC-3.2 in SystemVerilog

Name: Chuangy Zhang
NetID: czhan30
Name: Homero Vazquez
NetID: hvazqu6

1. Introduction

   In this experiment, we design a simple microprocessor using SystemVerilog. We designed a subset of the LC-3 ISA, which is a 16-bit processor with 16-bits program Counter(PC), 16-bits instructions, and 16 bit register. There are three main components of the design of the processor, CPU, Memory, and input and output interface that communicate with external devices. This experiment was split up into two desert tasks.

   In the first week, we implemented the FETCH phase, in which we had to understand the structure of the memory system, and how the memory system interfaces with the CPU. We also implement all the necessary CPU entities and ISDU unit control to be able to successfully fetch the instruction line by line from the on-board memory to the cpu.

   In the second week, We implemented the DECODE and the EXECUTE phase. We extend the provided skeleton ISDU to include all the necessary state transitions and the necessary inputs/outputs in each of the states. We had to learn the specification of the LC3 and its state diagram to figure out how to assign the various control signals in each state to produce the desired operation.

2. Written Description and Diagrams of SLC-3.
   a. Summary of Operation

      The simple computer performs various operations based on the opcode. In the table below, R(X) specifies a register in the register file, addressed by the three-bit address X. SEXT(X) indicates the 2's compliment sign extension of the operand X to 16 bits. nzp is the status register mentioned above. It is a three-bit value that states whether the resulting value loaded to the register file is negative, zero, or positive. This must be updated whenever an instruction performs a write to the register file (except JSR). For all instructions, PC + 1 is implicit, unless PC is stated to get some other value. In the table, right-hand-side "PC" indicates the value of the PC register after it was incremented immediately following fetch.

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) PCPC + SEXT(PCoffset9)
JMP	1100	000	BaseR 000000			PC R(BaseR)
JSR	0100	1	PCoffset11			R(7) PC; PCPC + SEXT(PCoffset11)
LDR	0110	DR	Base R	offset6		R(DR)M[R(BaseR) + SEXT(offset6)]
STR	0111	SR	Base R	offset6		M[R(BaseR) + SEXT(offset6)]R(SR)
PAUSE	1101	ledVect12				LEDsledVect12; Wait on Continue

The IR will provide the Instruction Sequencer/Decoder with the instruction to be executed. The IR will also provide the datapath with any other necessary data. As mentioned earlier, the Instruction Sequencer/Decoder will need to generate the control signals to execute the instructions in proper order. The Instruction Sequencer/Decoder will also specify the operation to

the ALU (e.g. add, etc.). Note that each operation will take multiple cycles and the Instruction Sequencer/Decoder will need to provide signals appropriately at each cycle.

On a reset, the Instruction Sequencer/Decoder should reset to the starting “halted” state, and wait for Run to go high. The PC should be reset to zero upon a reset, where it should proceed on incrementing itself when Run is pressed for fetching the instructions line by line. The first three lines of instructions will be used to load the PC with the value on the slider switches, which indicates the starting address of the instruction(s) of interest (in the form of test programs for the demo), and the program should begin executing instructions starting at the PC. Your computer must be able to return to the halted state any time a reset signal arrives.

Instruction Summary

ADD: Adds the contents of SR1 and SR2, and stores the result to DR. Sets the status register.

ADDi: Add Immediate. Adds the contents of SR to the sign-extended value imm5, and stores the result to DR. Sets the status register.

AND: ANDs the contents of SR1 with SR2, and stores the result to DR. Sets the status register.

ANDi: And Immediate. ANDs the contents of SR with the sign-extended value imm5, and stores the result to DR. Sets the status register.

NOT: Negates SR and stores the result to DR. Sets the status register.

BR: Branch. If any of the condition codes match the condition stored in the status register, takes the branch; otherwise, continues execution. (An unconditional jump can be specified by setting NZP to 111.) Branch location is determined by adding the sign-extended PCoffset9 to the PC.

JMP:Jump. Copies memory address from BaseR to PC.

JSR:Jump to Subroutine. Stores current PC to R(7), adds sign-extended PCoffset11 to PC.

LDR: Load using Register offset addressing. Loads DR with memory contents pointed to by (BaseR + SEXT(offset6)). Sets the status register.

STR: Store using Register offset addressing. Stores the contents of SR at the memory

location pointed to by (BaseR + SEXT(offset6)).

PAUSE: Pauses execution until Continue is asserted by the user. Execution should only unpause if Continue is asserted during the current pause instruction; that is, when multiple pause instructions are encountered, only one should be “cleared” per press of Continue. While paused, ledVect12 is displayed on the board LEDs. See I/O Specification section for usage notes.

Here are the operations in more detail:

Fetch:

MAR <- PC; MAR = memory address to read the instruction from

MDR <- M(MAR); MDR = Instruction read from memory (note that M(MAR) specifies the data at address MAR in memory).

IR <- MDR; IR = Instruction to decode

PC <- (PC + 1)

Decode: Instruction Sequencer/Decoder <- IR

Execute: Perform the operation based on the signals from the InstructionSequencer/Decoder and write the result to the destination register or memory.

Fetch, Load, and Store Operations:

For Fetch, Load (LDR), and Store (STR) operations you will need to set the memory signals (see Memory Interface below) appropriately for each state of the fetch/load/store sequence. Also, notice that the RAM we use does not have an R signal indicating that a read/write operation is ready. Instead, for any states reading from or writing to RAM, we stay at those states for several clock cycles to ensure that a memory read/write operation is complete.

FETCH:

state1: MAR <- PC

state2: MDR <- M(MAR); -- *assert Read Command on the RAM*

state3: IR <- MDR;

PC <- PC+1; -- "+1" inserts an incrementer/counter instead of an adder. Go to the next state.

LOAD:

state1: MAR <- (BaseR + SEXT(offset6)) from ALU

state2: MDR <- M(MAR); -- *assert Read Command on the RAM* state3: R(DR)MDR;

STORE:

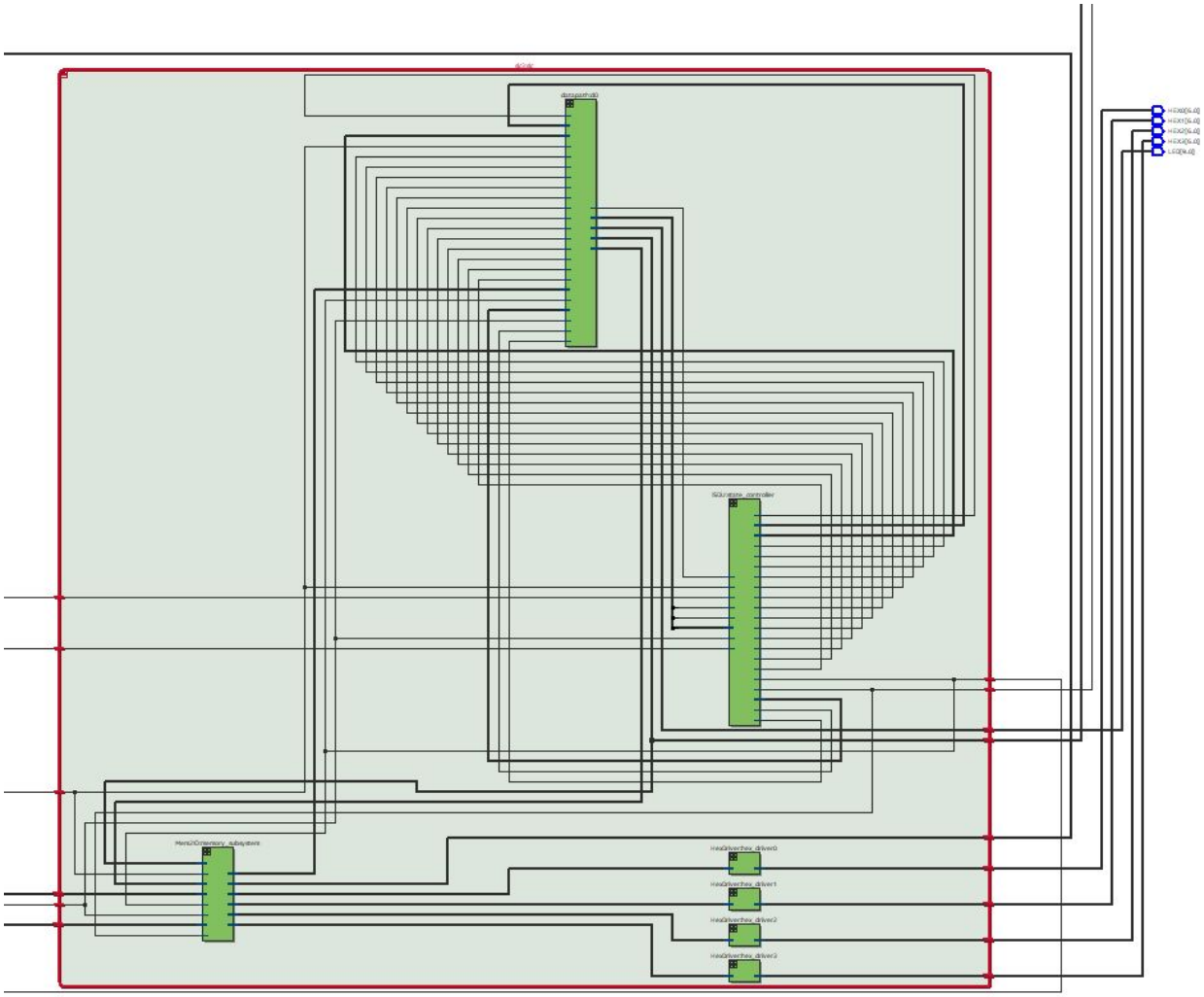
state1: MAR <- (BaseR + SEXT(offset6)) from ALU; MDRR(SR) state2: M(MAR)MDR; -- *assert Write Command on the RAM*

b. Describe how the SLC-3 performs its functions

In the first week, we implemented the FETCH phase, in which we had to understand the structure of the memory system, and how the memory system interfaces with the CPU. We also implement all the necessary CPU entities and ISDU unit control to be able to successfully fetch the instruction line by line from the on-board memory to the cpu.

In the second week, We implemented the DECODE and the EXECUTE phase. We extend the provided skeleton ISDU to include all the necessary state transitions and the necessary inputs/outputs in each of the states. We had to learn the specification of the LC3 and its state diagram to figure out how to assign the various control signals in each state to produce the desired operation.

c. Block diagram of SLC-3



d. Written Description of all .sv modules

- **Mux2to1:**

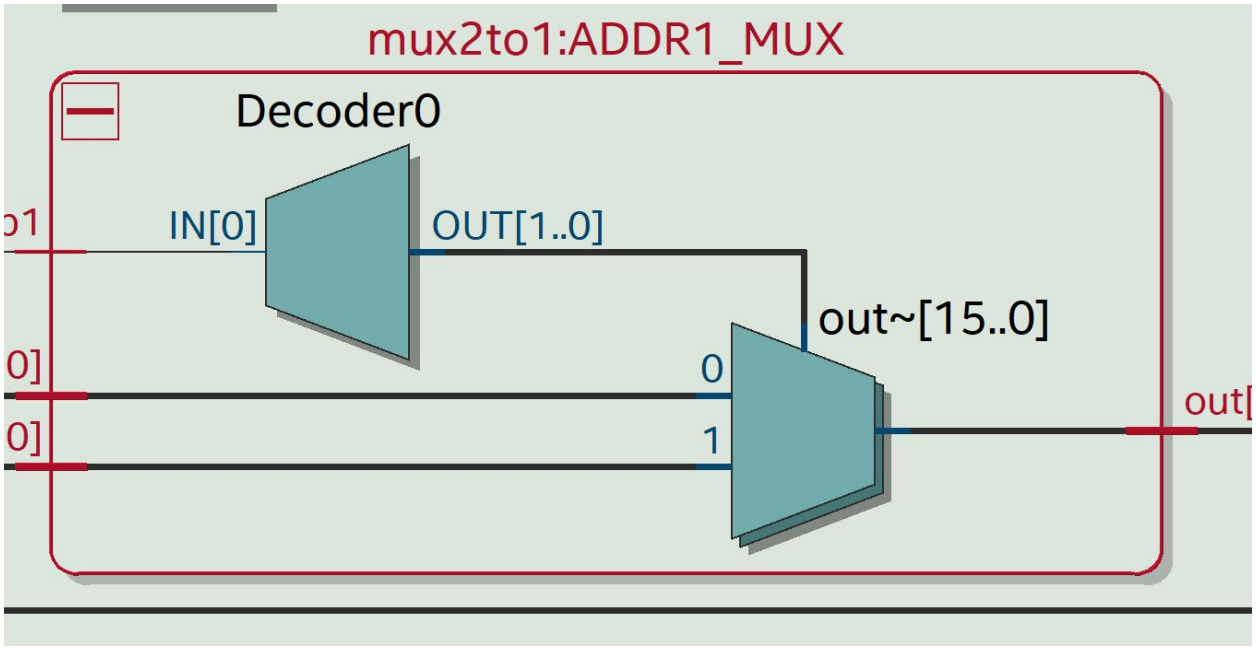
  Module : mux2to1.sv

  Inputs: [15:0] in0, [15:0] in1, sb1

  Outputs: [15:0] out

Description: This is a 2:1 Multiplexer. It takes two 16-bit inputs and the sb1 signal selects which one will be the output.

Purpose: This module is used in multiple parts of the design to select between 2 inputs. It is used for the ADD1 MUX, MIO MUX, SR2 MUX, DR MUX and, SR1 MUX



- **Mux4to1:**

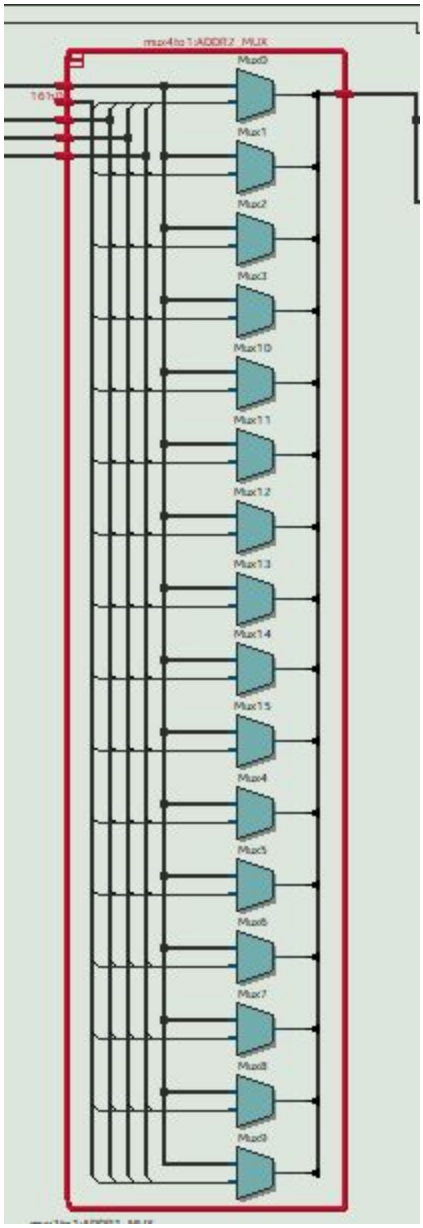
  Module : mux2to1.sv

  Inputs: [15:0] in00, [15:0] in01,[15:0] in10, [15:0] in11, [1:0]sb2

  Outputs: [15:0] out

  Description: This is a 4:1 Multiplexer. It takes four 16-bit inputs and the sb2 signal selects which one will be the output.

  Purpose: This module is used in multiple parts of the design to select between 4 inputs of data. It is used for the ADDR2 MUX, PC MUX



- **Reg_16:**

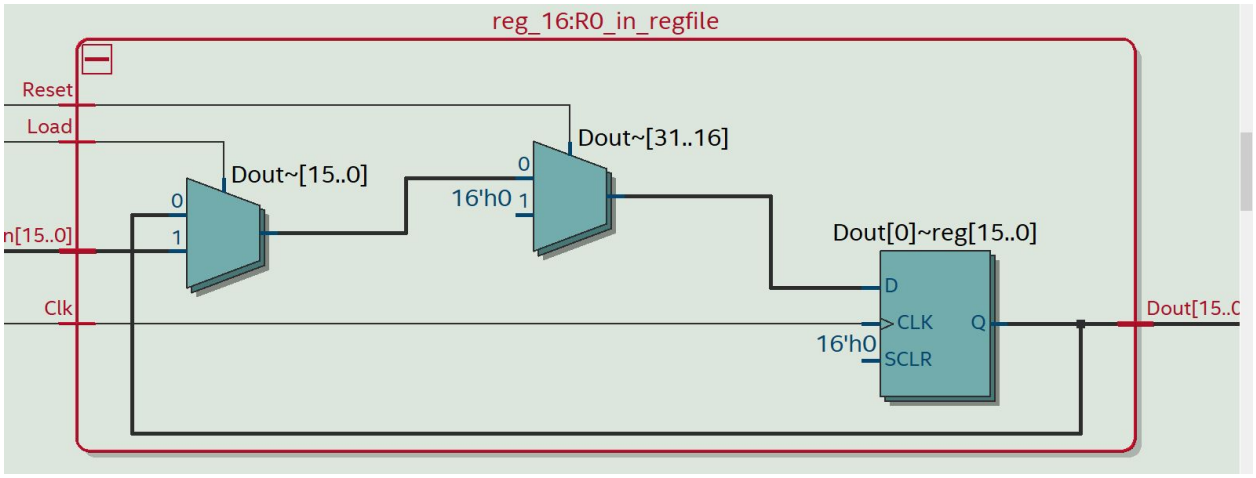
  Module : reg_16.sv

  Inputs: [15:0] Din, Clk, Reset, Load

  Outputs: [15:0] Dout

  Description: This is a positive edge triggered 16-bit register with synchronous reset, load and shift.

  Purpose: This module is used to create multiple registers in our design. We used this module for the 8 registers in the register file, the IR Register, the MAR Register, the MDR Register, and the PC Register.



- **reg_file :**

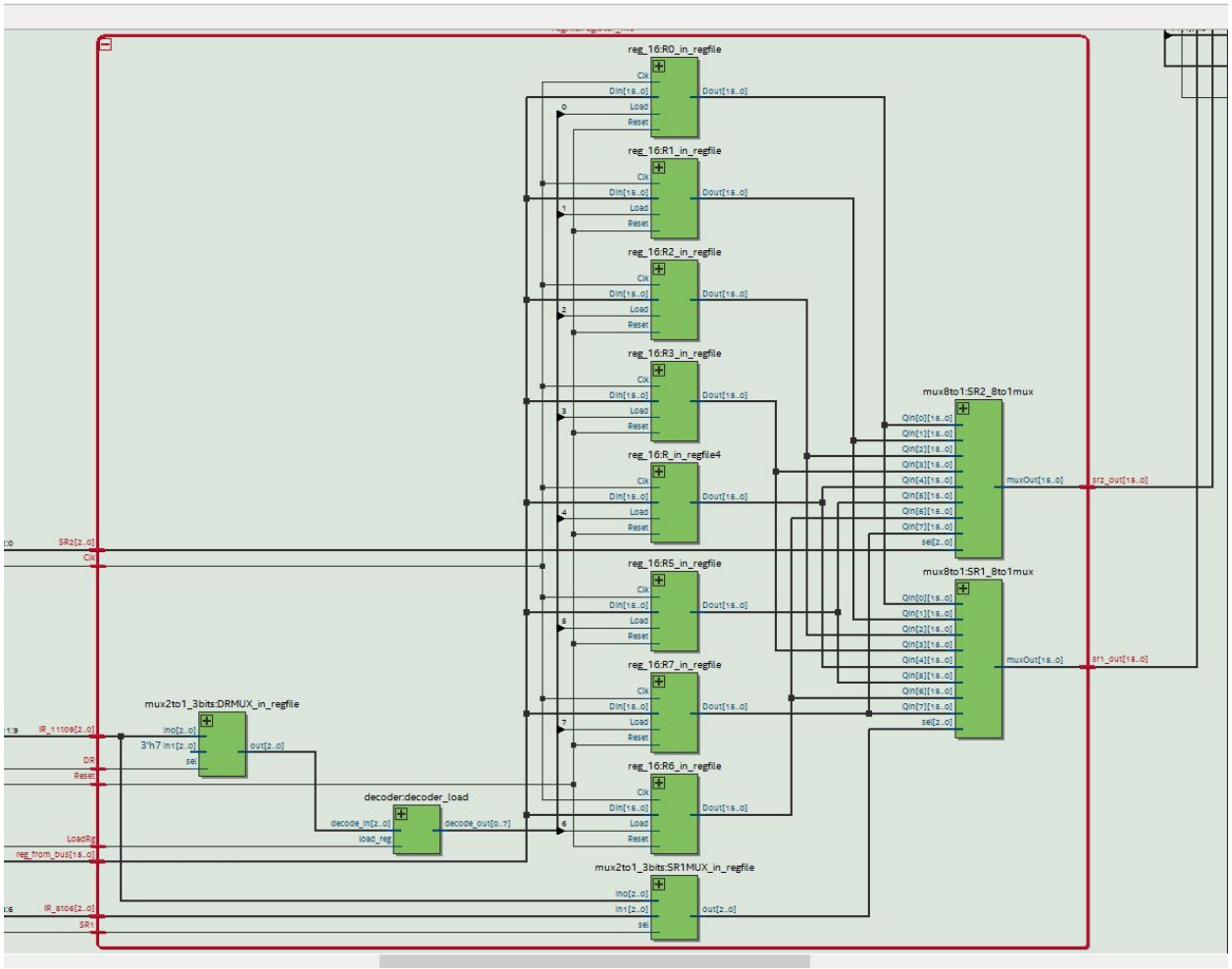
  Module : reg_file.sv

  Inputs: [15:0] reg_from_bus,[2:0] IR_11to9, [2:0] IR_8to6, Clk, Reset, Load, [2:0] SR2, DR, SR1

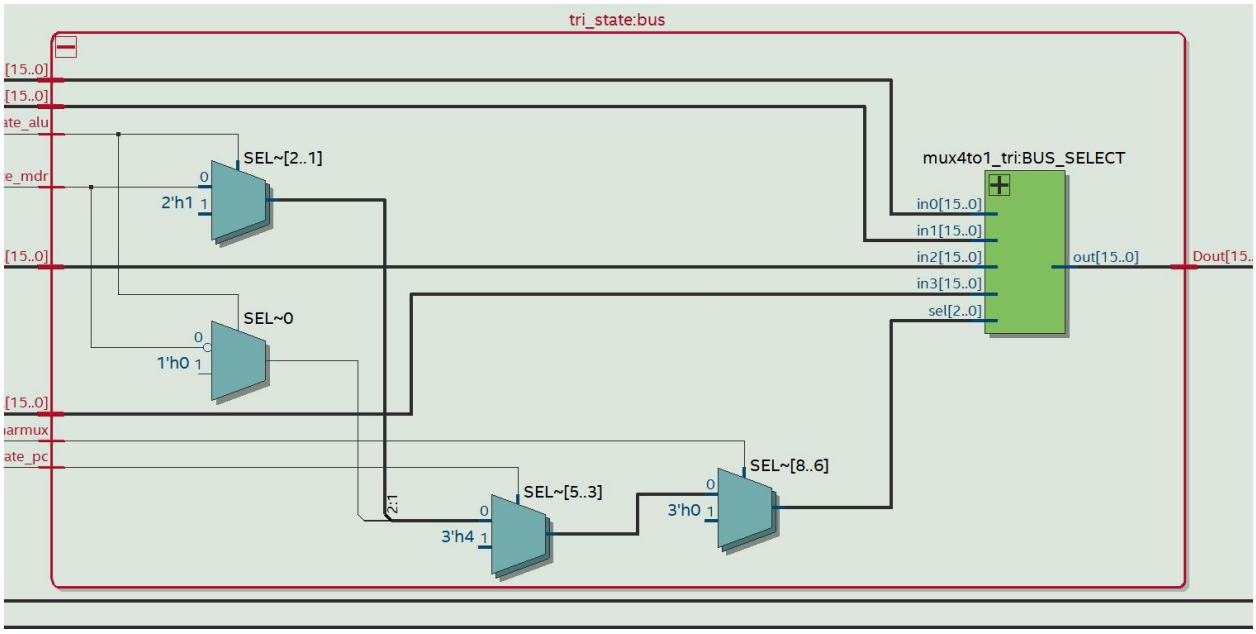
  Outputs: [15:0] sr1_out, [15:0] sr2_out

  Description: This file contains 8 registers, that depending on the inputs will be selected to either write or read data to and from them.

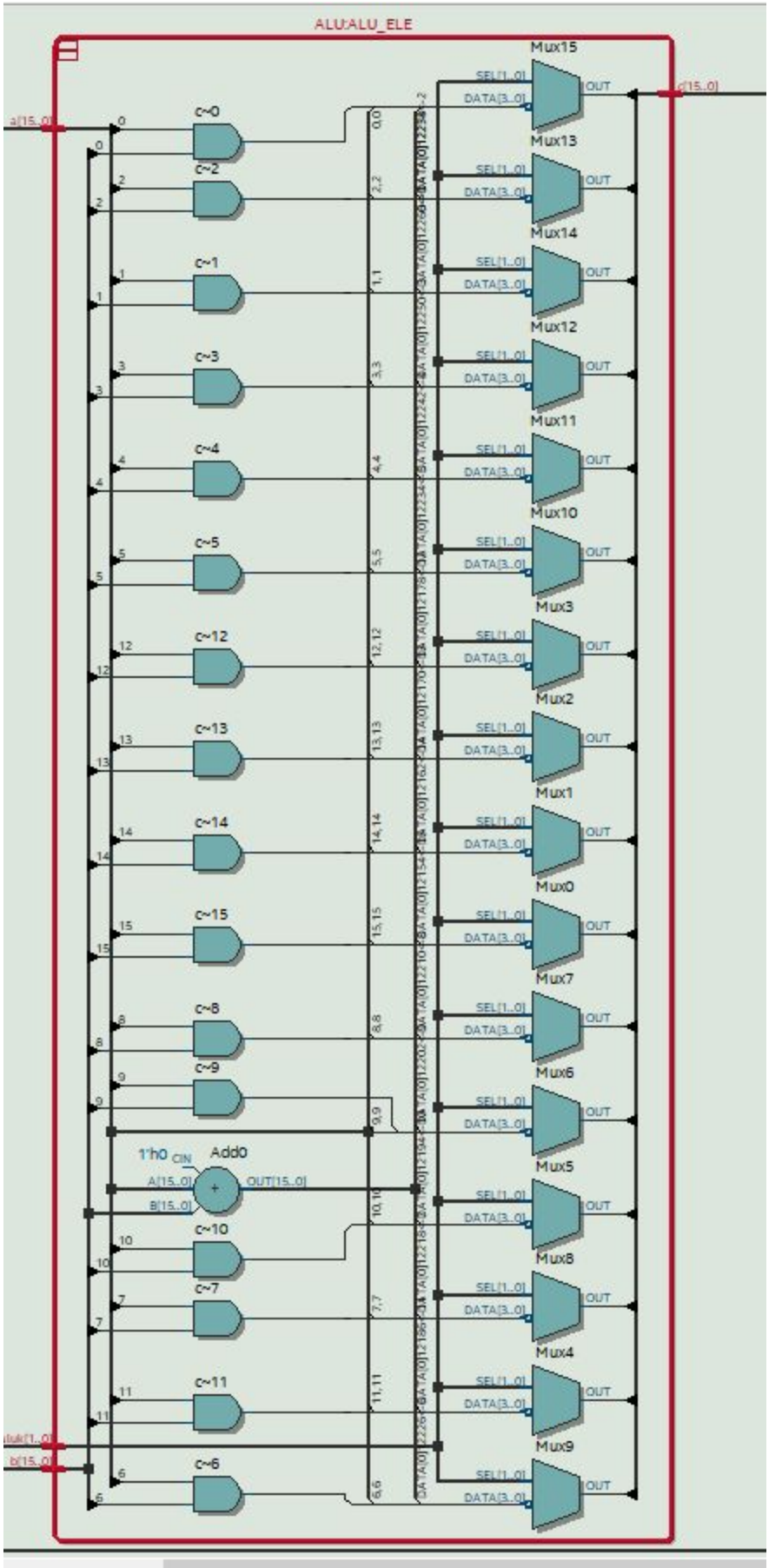
  Purpose: The purpose of this register file is to get data from the bus then get signals from the ISDU to utilize the data in the registers for multiple operations.



- **Tri_state:**
  Module : tri_state.sv
  Inputs: [15:0] MAR_data, [15:0] MDR_data, [15:0] ALU_data, [15:0] PC_data, gate_marmux, gate_pc, gate_alu, gate_mdr
  Outputs: [15:0] Dout
  Description: This file contains a 4:1 mux which will choose between four 16-bit inputs based on the value of the four 1-bit inputs which will be encoded to act as a selector for the 4:1 Mux.
  Purpose: The purpose of this module is to replace the tri state buffers on the original LC_3 so that our bus can obtain the correct data depending on the ISDU signals.



- **ALU:**
  Module : ALU.sv
  Inputs: [15:0] a, [15:0] b, [1:0] aluk
  Outputs: [15:0] c
  Description: This module has two 16-bit inputs and one 2-bit input that will select between 4 operations to be performed on the 16-bit inputs and then outputs the result.
  Purpose: The purpose of this module is to do operations either from the register file or from the SR2 MUX and output the result to the bus. The operations that this module can perform are ADD, AND, NOT and let c=a.


ALU:ALU_ELE
a[15..0]
b[15..0]
c[15..0]
c~0
c~1
c~2
c~3
c~4
c~5
c~6
c~7
c~8
c~9
c~10
c~11
c~12
c~13
c~14
c~15
1'h0
CIN
Add0
A[15..0]
B[15..0]
OUT[15..0]
Mux15
Mux13
Mux14
Mux12
Mux11
Mux10
Mux3
Mux2
Mux1
Mux0
Mux7
Mux6
Mux5
Mux8
Mux4
Mux9
SEL[1..0]
DATA[3..0]
OUT

- **ben_element:**

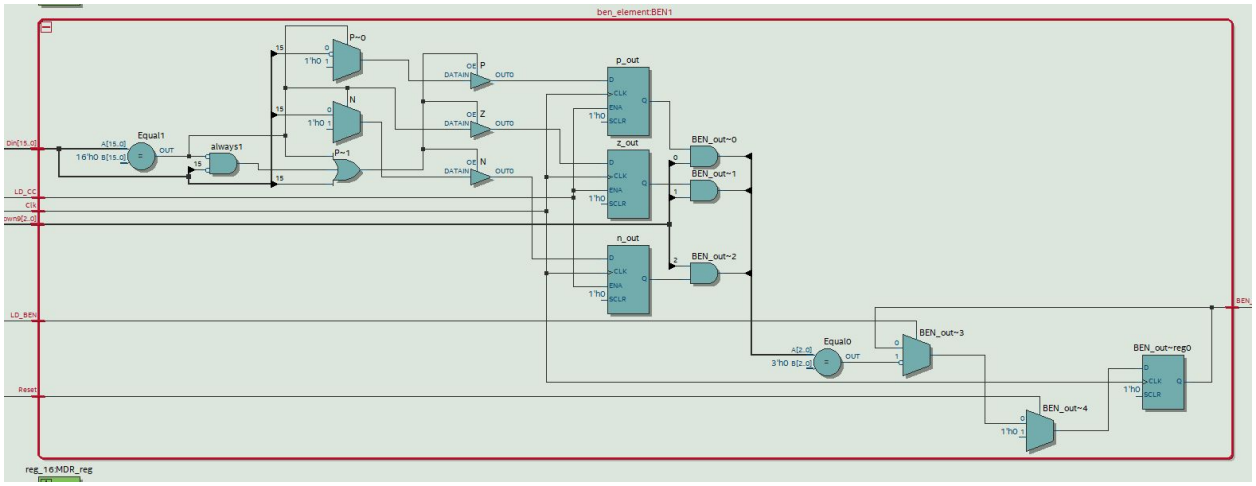
Module : ALU.sv

Inputs: [15:0] Din, [2:0] IR11down9, LD_CC, LD_BEN, Clk, Reset

Outputs: BEN_out

Description: This module contains logic to see if the data coming from the bus is either zero, positive or negative. It also checks the bits 11:9 of the IR to compare with BEN whenever the LD_BEN is on.

Purpose: The purpose of this module is to determine whether we will branch depending on the values of the instruction BEN.



- **datapath:**

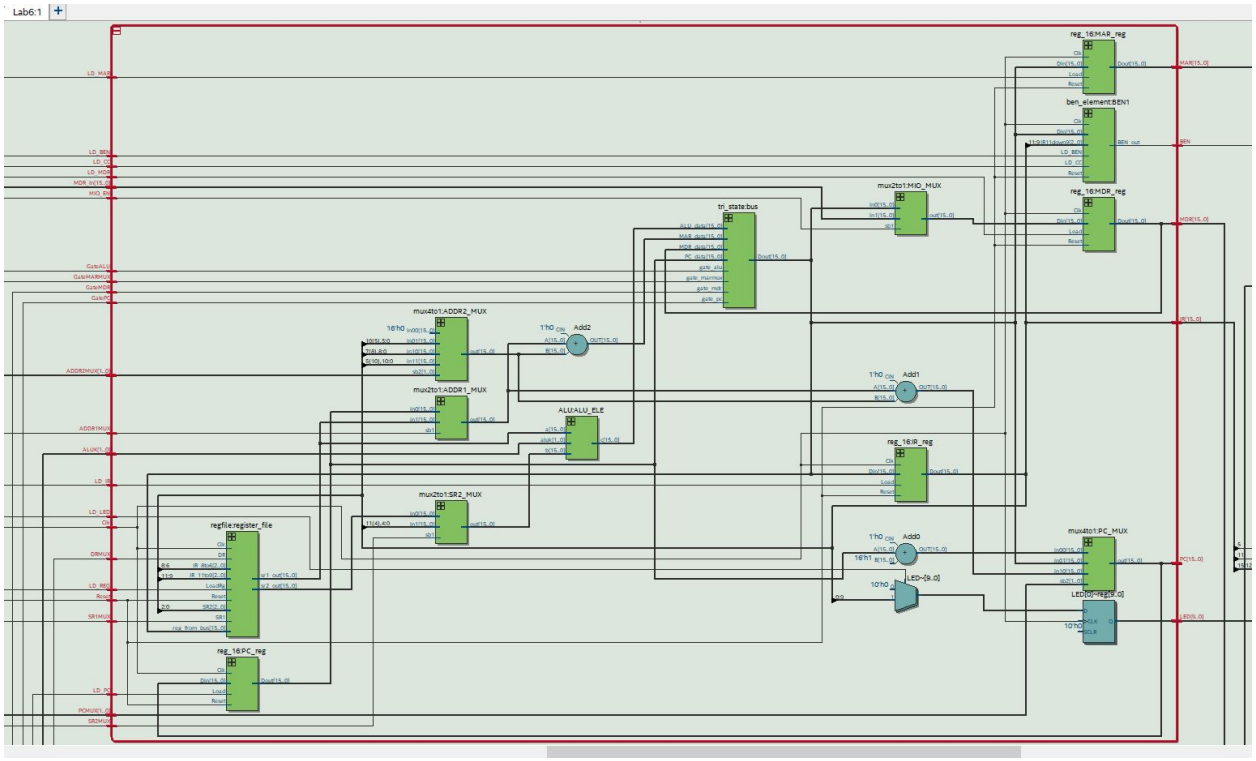
  Module: datapath.sv

  Inputs: [15:0] MDR_In, [1:0]PCMUX, [1:0] ADDR2MUX, ALUK, MIO_EN, SR2MUX, ADDR1MUX, DRMUX, SR1MUX, GatePC, GateMDR, GateALU, GateMARMUX, LD_MAR, LD_MDR, LD_IR, LD_REG, LD_PC,LD_BEN, LD_CC, LD_LED, Clk, Reset

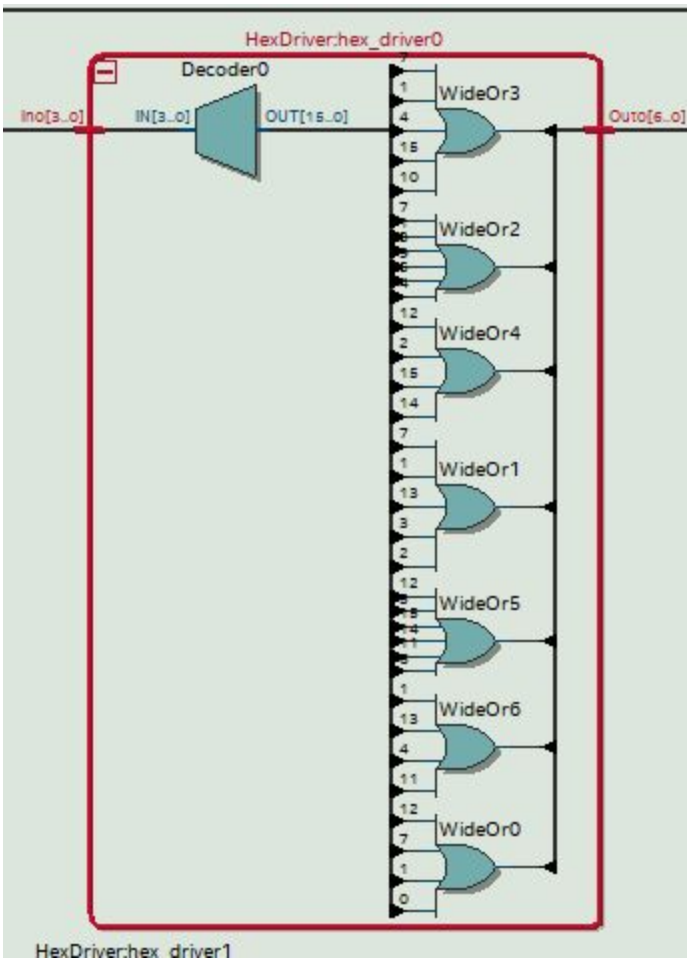
  Outputs: [15:0] MAR, [15:0] MDR, [15:0] IR, [15:0] PC , [9:0] LED, BEN

  Description: This module contains most of the files described earlier and it makes the connections between them.

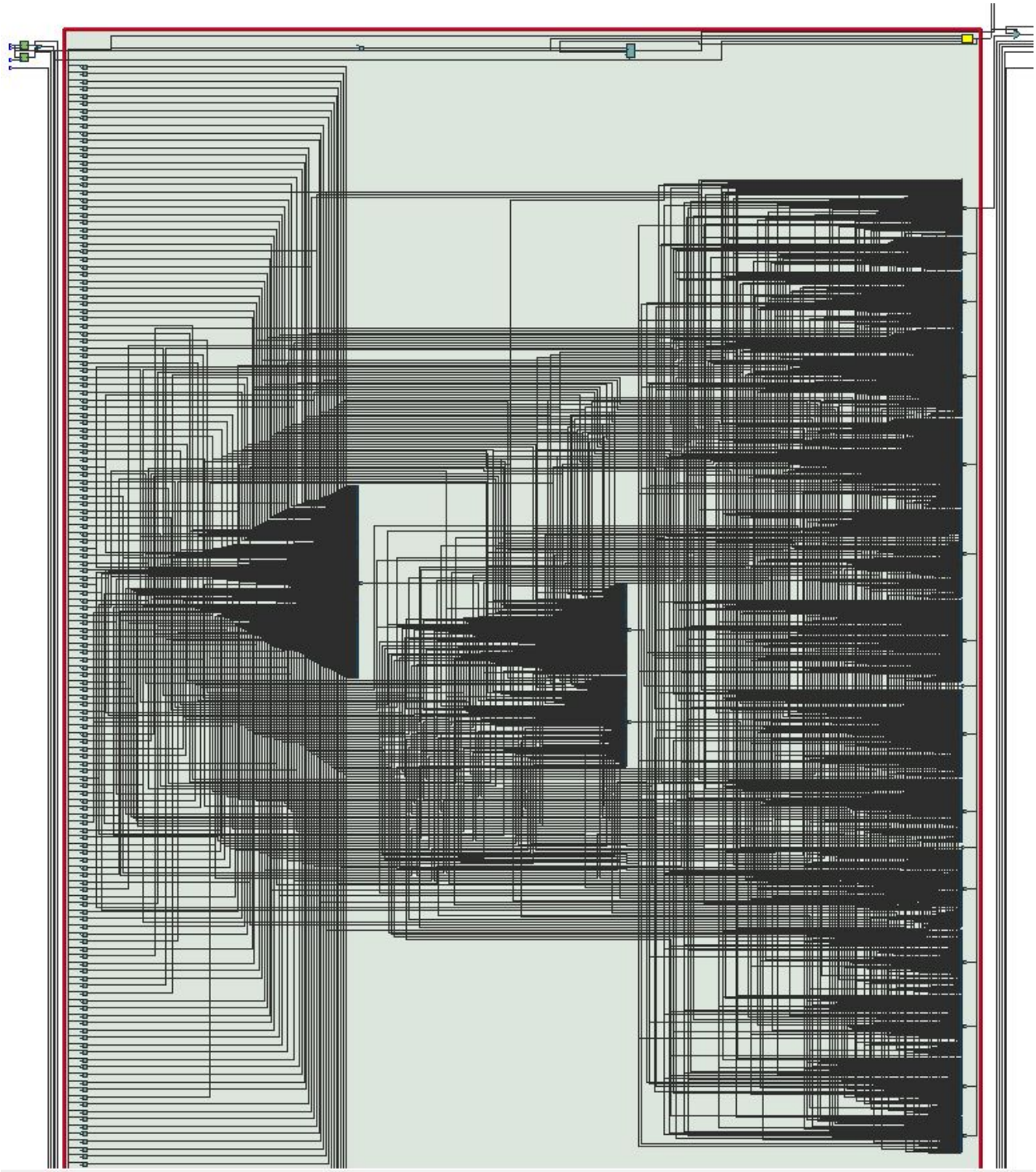
  Purpose: The purpose of this module is to connect every module in the LC-3



- **HexDriver:**
  Module: HexDriver.sv
  Inputs: [3:0] In0
  Outputs: [6:0]Out0
  Description: This module has hardcoded values that convert 4bit binary numbers from registers into hexadecimal
  Purpose: The purpose of this module is to take in 4 bit binary numbers from our registers and convert them into hexadecimal to be able to display in the seven segment displays on the FPGA Board.



- **Instantiateram:**
  Module: Instantiate.sv
  Inputs: Reset, Clk
  Outputs: [15:0] ADDR, [15:0] data, wren
  Description: This module has the addresses to the program instructions.
  Purpose: The purpose of this module is to store and write the test program instructions sequentially (from address 0 and incrementing) to the on-chip memory.

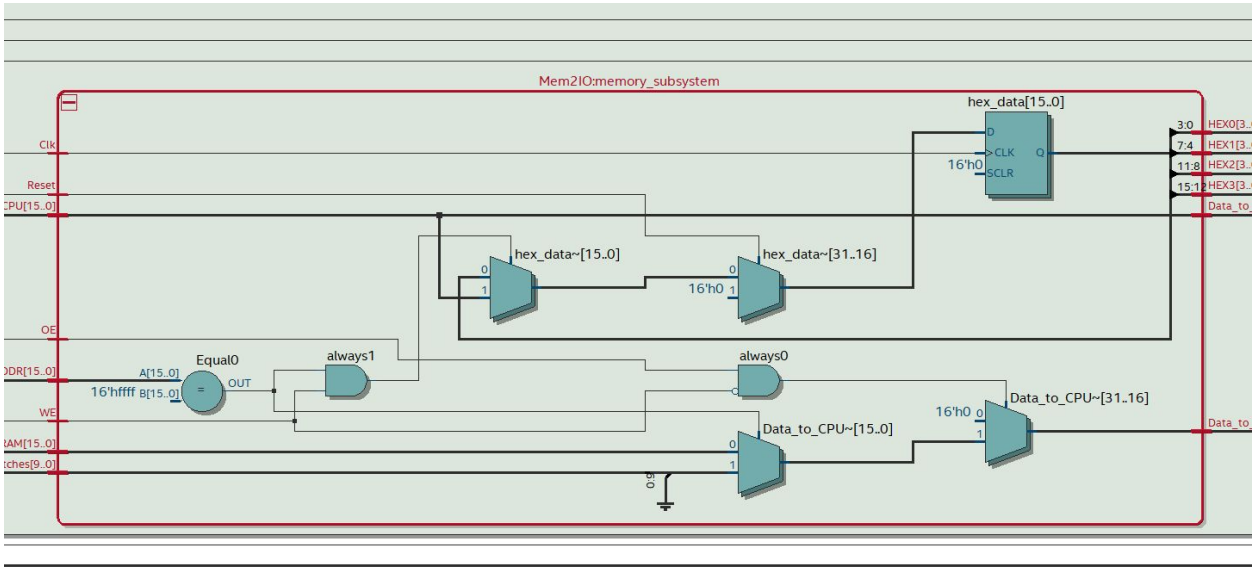


- **memory_contents:**
  Module: memory_contents.sv
  Inputs: None
  Outputs: [15:0] ADDR, [15:0] mem_array[0:256]
  Description: This module is a memory with similar behavior as the on-chip memory of the MAX10 board.
  Purpose: The purpose of this module is to act as the on-chip memory but for simulation only.

- **test_memroy:**

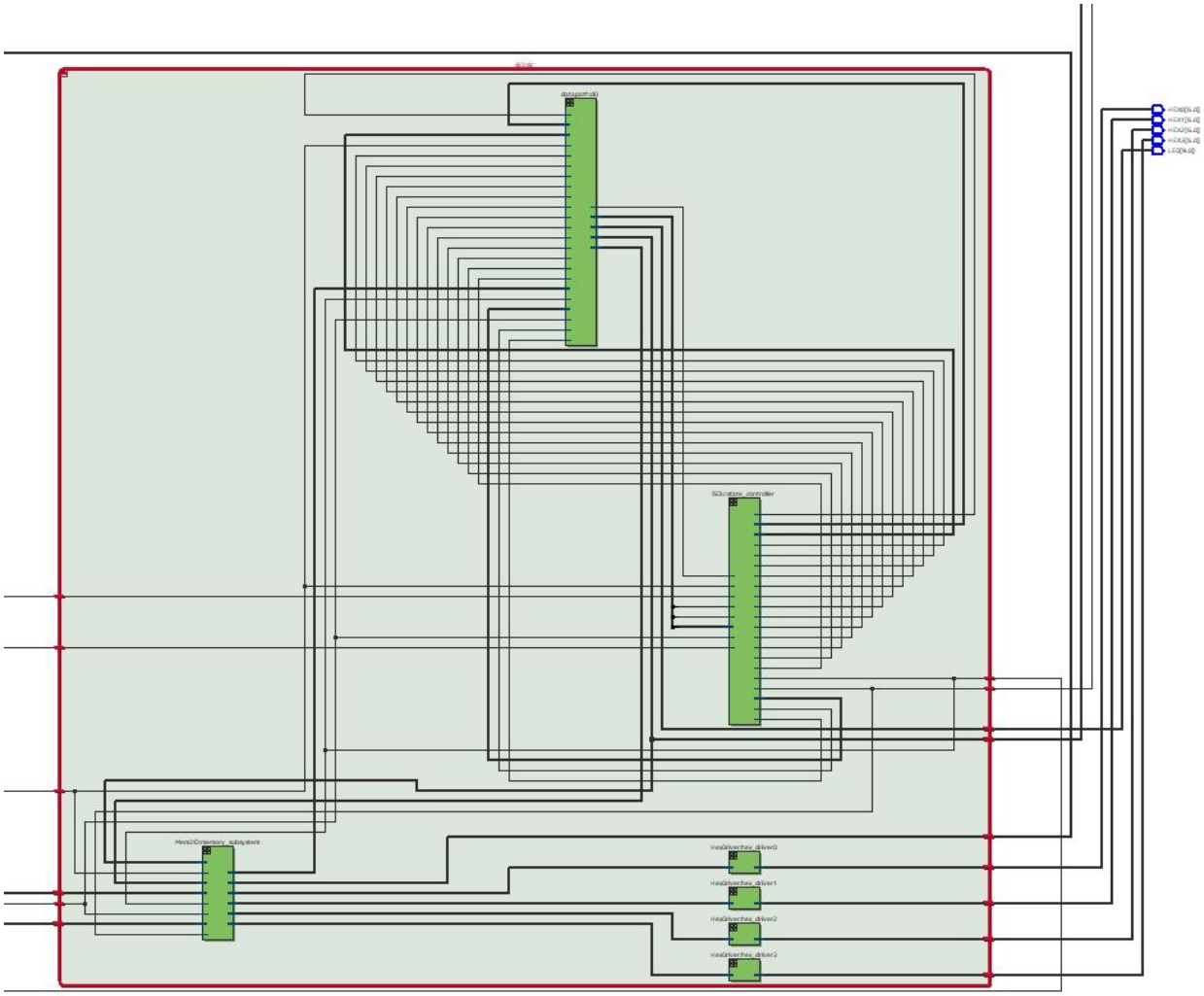
Module: test_memory.sv
Inputs: input Reset, Clk,[15:0] data,[9:0] address,rden, wren
Outputs: [15:0] readout
Description: This module contains the memory for the test memory
Purpose: The purpose of this module is to populate the test memory

- **Mem2IO:**
  Module: Mem2IO.sv
  Inputs: Clk, Reset,[15:0] ADDR, OE, WE,[9:0] Switches, [15:0] Data_from_CPU, Data_from_SRAM
  Outputs: [15:0] Data_to_CPU, Data_to_SRAM,[3:0] HEX0, HEX1, HEX2, HEX3
  Description: This module contains the interfaces for the FPGA IO and the SRAM
  Purpose: The purpose of manage all the I/O with the DE 10-lite physical I/O devices



- **slc3:**
  Module: slc3.sv
  Inputs: [9:0] SW, Clk, Reset, Run, Continue, [15:0] Data_from_SRAM,

  Outputs: [9:0] LED, OE, WE,[6:0] HEX0, HEX1, HEX2, HEX3,[15:0] ADDR, [15:0] Data_to_SRAM
  Description: This module is the top level for the entire SLC-3
  Purpose: This module is the entire interface of the system containing the datapath, ISDU, Mem2IO and HexDrivers



e. Description of ISDU

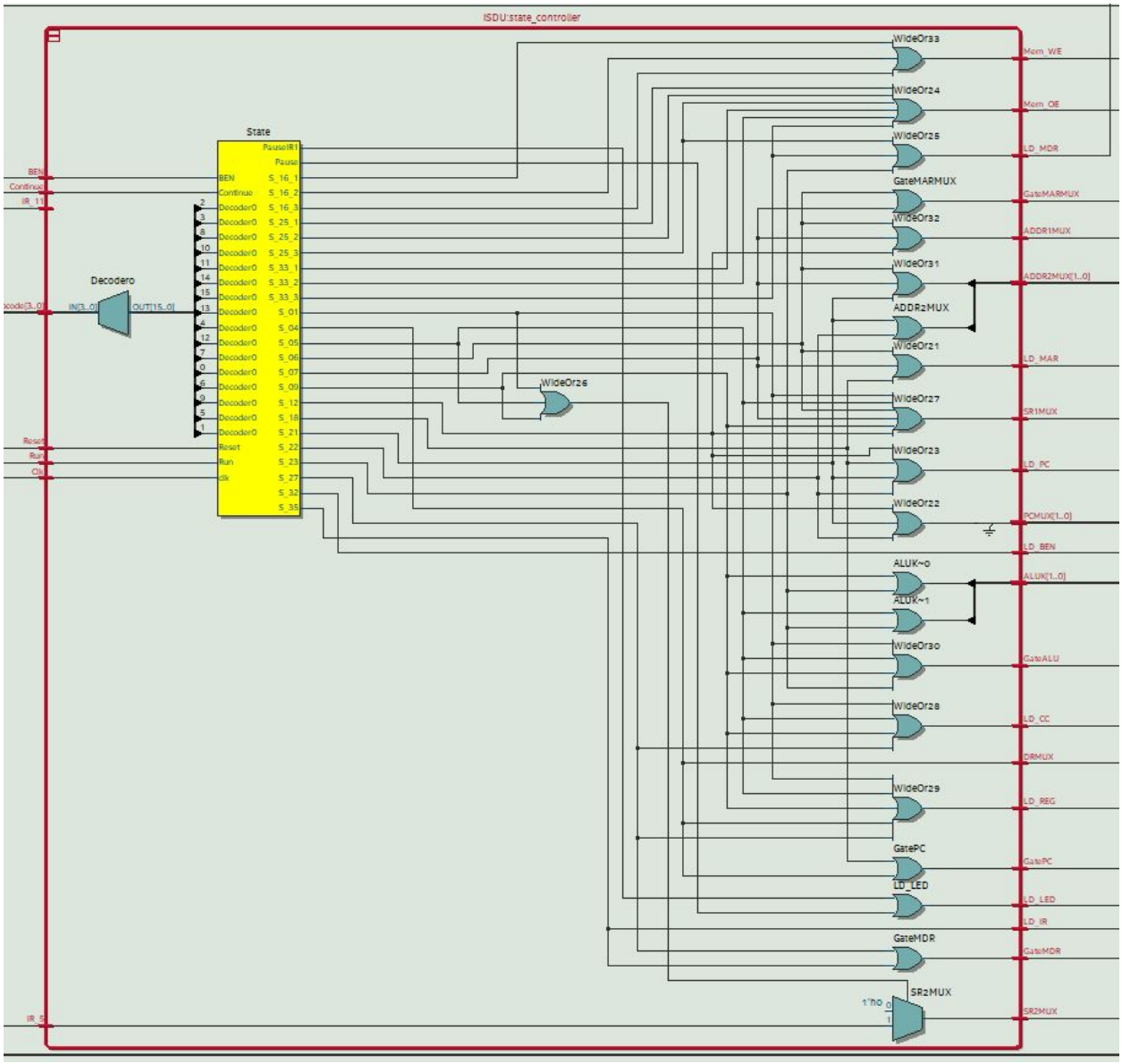
Module: ISDU.sv

Inputs: Clk, Reset, Run, Continue, [3:0] Opcode, IR_5, IR_11, BEN

Outputs: LD_MDR, LD_IR, LD_BEN, LD_CC, LD_REG, LD_PC, PC_LED,
GatePC, GateMDR, GateMAR,GateALU
[1:0] PCMUX, ADDR2MUX, ALUK,
SR1MUX, SR2MUX, ADDR1MUX, Mem_OE, MEM_WE

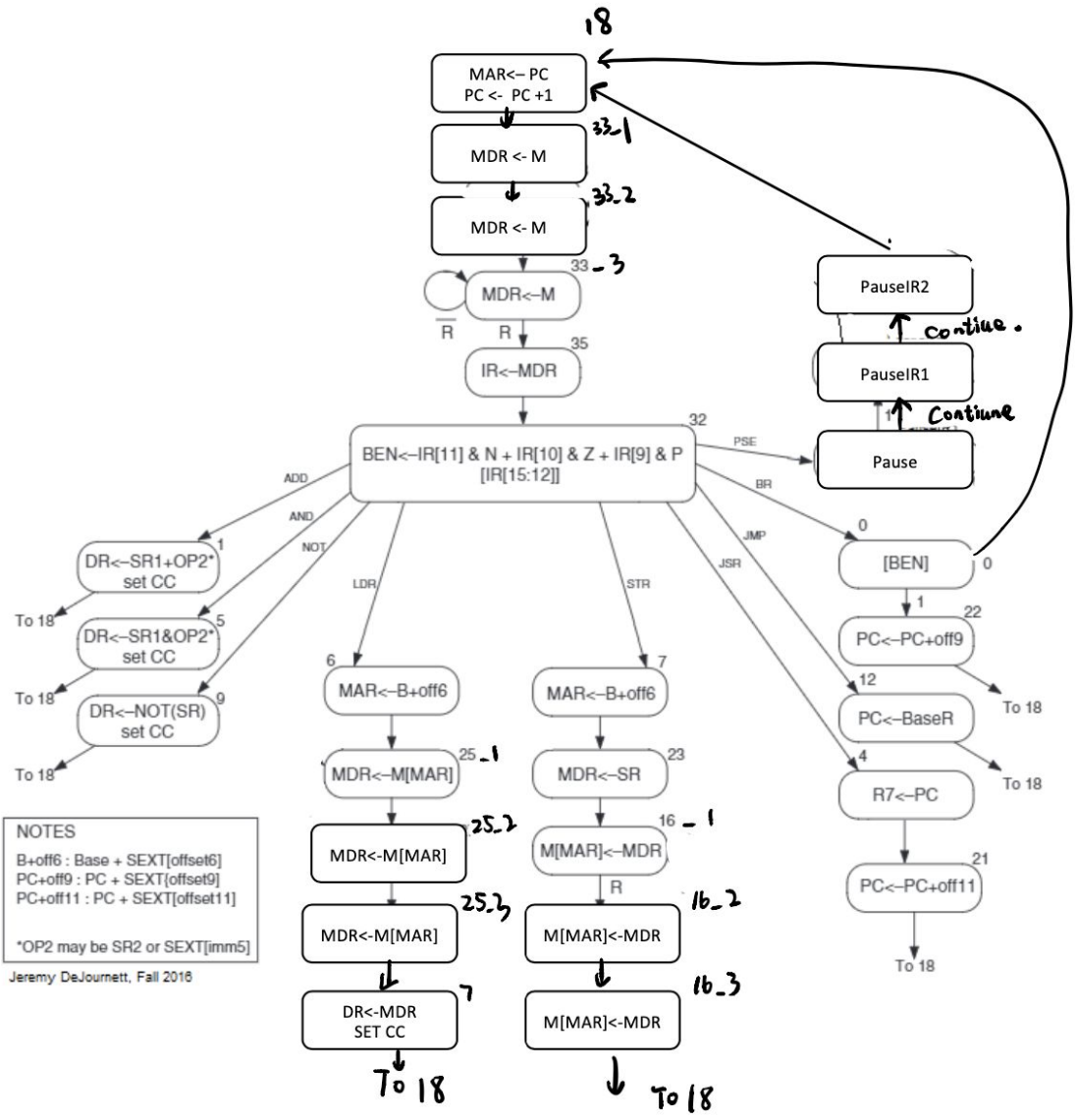
Description: This module contains all the signals that will control the datapath of the SLC-3, it has many states described below and depending on the state it will give different output values that go directly into the SLC-3 Datapath.

Purpose: The purpose of this module is to control what the SLC-3 will do depending on the instruction that comes from the pc address.



G. State Diagram of ISDU

Default signal are all set to 0;

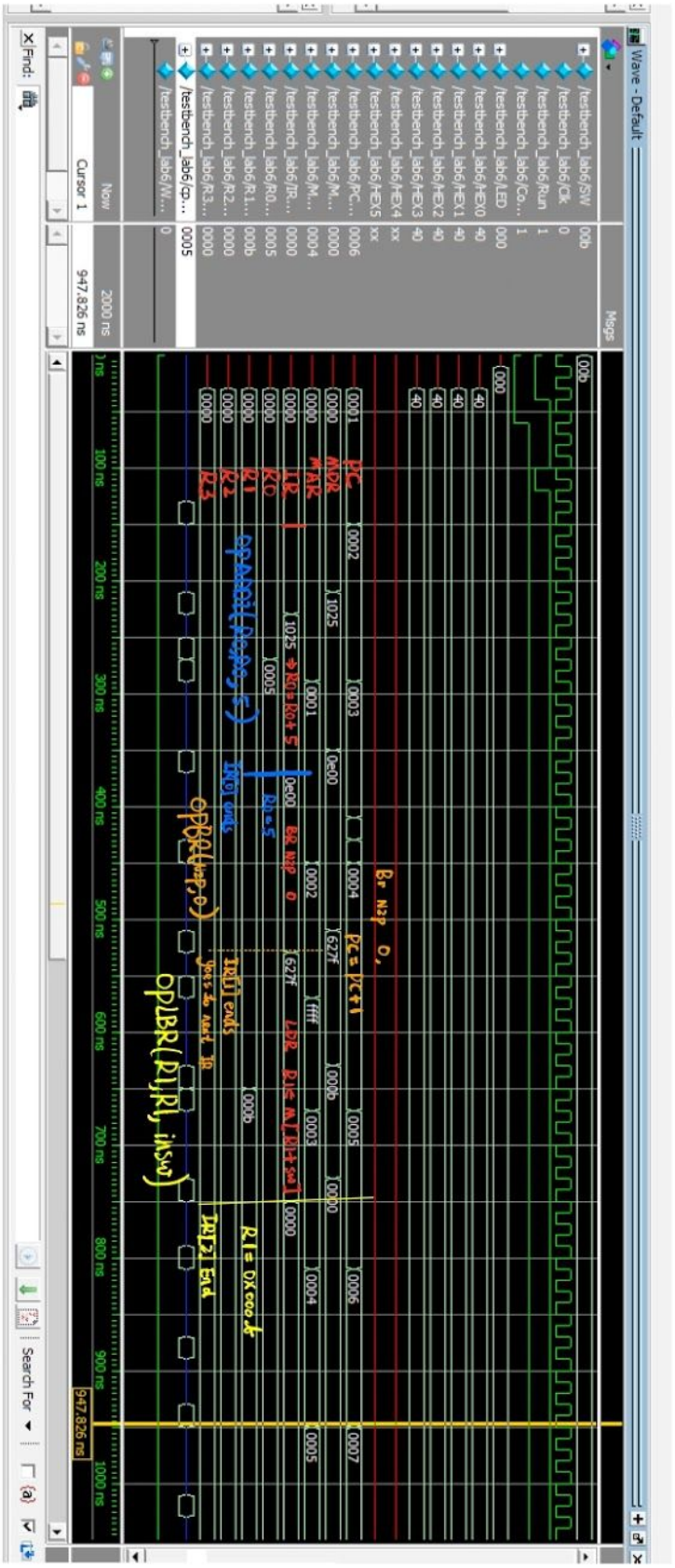


State	Signals
S_18	GatePC = 1'b1; LD_MAR = 1'b1; PCMUX = 2'b00; LD_PC = 1'b1;
S_33_1	Mem_OE = 1'b1;
S_33_2	Mem_OE = 1'b1;
S_33_3	Mem_OE = 1'b1; LD_MDR = 1'b1;
S_35	GateMDR = 1'b1; LD_IR = 1'b1;

S_32	LD_BEN = 1'b1;
S_01	SR2MUX = IR_5; SR1MUX = 1'b1; ALUK = 2'b00; LD_CC = 1'b1; LD_REG = 1'b1; GateALU = 1'b1;
S_05	SR2MUX = IR_5; ALUK = 2'b01; GateALU = 1'b1; LD_REG = 1'b1; SR1MUX = 1'b1; LD_CC = 1'b1;
S_09	SR2MUX = IR_5; ALUK = 2'b10; GateALU = 1'b1; LD_REG = 1'b1; SR1MUX = 1'b1; LD_CC = 1'b1;
S_06	SR1MUX = 1'b1; ADDR1MUX = 1'b1; ADDR2MUX = 2'b01; GateMARMUX= 1'b1; LD_MAR = 1'b1;
S_25_1	Mem_OE = 1'b1;
S_25_2	Mem_OE = 1'b1;
S_25_3	Mem_OE = 1'b1; LD_MDR = 1'b1;
S_27	GateMDR = 1'b1; LD_REG = 1'b1; LD_CC = 1'b1;
S_07	SR1MUX = 1'b1; ADDR1MUX = 1'b1; ADDR2MUX = 2'b01; LD_MAR = 1'b1; GateMARMUX = 1'b1;
S_23	ALUK = 2'b11;

	SR1MUX = 1'b0; GateALU = 1'b1; LD_MDR = 1'b1;
S_16_1	Mem_WE =1'b1;
S_16_2	Mem_WE =1'b1;
S_16_3	Mem_WE =1'b1;
S_00	No signals
S_22	ADDR2MUX = 2'b10; ADDR1MUX = 1'b0; PCMUX = 2'b10; LD_PC = 1'b1;
S_12	SR1MUX = 1'b1; ADDR2MUX = 2'b00; ADDR1MUX = 1'b1; PCMUX = 2'b10; LD_PC = 1'b1;
S_04	GatePC = 1'b1; DRMUX = 1'b1; LD_REG = 1'b1;
S_21	ADDR2MUX = 2'b11; ADDR1MUX = 1'b0; PCMUX = 2'b10; LD_PC = 1'b1;
Pause	LD_LED = 1'b1;
PauseIR1	LD_LED = 1'b1;
PauseIR2	No signals
Halted	No signals

3. **Simulations of SLC-3 Instructions**



4. Post-Lab Questions

a.

LUT	544
DSP	0
Memory	0
Flip_flop	251
Frequency	69.03MHz
Static Power	89.49 mW
Dynamic Power	6.98mW
Total Power	108.32mW

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

The MEM2IO is used for the LC-3 to interface with the on-chip RAM and the switches. If the address is 0xFFFF, then it will read the data of the switches, if the address is not 0xFFFF, then i will read the data from on-chip RAM. The main function of MEM2IO is allowing SLC-3 to interface with the on-chip RAM or the switch for our LDR and STR instructions.

c. What is the difference between BR and JMP instruction?

The biggest difference between BR and JMP instruction is, BR instruction is conditional and based on the NZP flags, on the other hand, JMP instruction is not conditional and it will jump to whatever PC value is. BRnzp does the same operation as JMP, but BRnzp only takes 9 bits of pc offset, but JMP takes 11bits of pc offset.

d. 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 to get an indication that the data read or write is completed or ready to go to the next instruction in Patt and Patel. The memory that we are working with doesn't have the ready(R) signal. We added extra clock cycles so that memory will have enough time to wait to read or write in the memory. SLC-34 doesn't rely on asynchronous signals to read or write, so we do not synchronize for it.

5. Conclusion

a. Our design awas fully functional and we were able to get it the work. But unfortunately, we didn't know how to get the internal signal to display in modelsim in the first week, because we didn't learn in the lecture, if we learned that early, we won't lose some of the points in week 1. We got all the points in week 2 and passed all the test cases. We spent a lot of time debugging on the ISDU but in the end, we got it to work.

b. I think that we would be able to save some time if there was a place on the manual where we can see common mistakes and how to fix them. Also this didn't happen to us but there was a team that didn't know that the lab files had been updated that same week. I think that if there is a way to tell everyone that the lab files were updated that will save the students so much time. We also learned that we have to learn all the knowledge in a hard way.