ECE 385

Spring 2024

Experiment #4

Lab 5: SLC-3 Computer

Sidharth Anand (sanand12) & Praveen Natarajan (pn17)
Section YD
Yash Damania
2/26/2024

Table of Contents

I. Introduction..... 3
II. Written Description and Diagrams of SLC-3.....4
III. Simulations of SLC-3..... 22
IV. Post Lab Questions..... 25
V. Conclusion..... 26

I. Introduction

The basic functionality of the SLC-3 encompasses fetching an instruction from memory, decoding the instruction to determine the operation, executing the operation, and then proceeding to the next instruction in a continual fetch-decode-execute cycle. It supports a range of operations such as ADD, AND, NOT, conditional branching (BR), jumping (JMP and JSR), and memory access instructions (LDR and STR), with both direct and immediate addressing modes. Additionally, the SLC-3 includes an I/O specification for interfacing with external devices, using memory-mapped I/O for interactions such as reading switches and controlling LEDs. A unique instruction, PAUSE, is used to pause execution for interacting with I/O devices or for debugging purposes.

The SLC-3 processor is a simplified version of the LC-3 processor. It features a reduced instruction set and simplified hardware architecture. Unlike the LC-3, the SLC-3 includes specific instructional features such as the PAUSE instruction for I/O interaction and debugging, and does not have nearly as much memory mapping functions as LC3. Overall, the SLC-3 serves as an accessible introduction to the principles underlying more complex processors like the LC-3, focusing on core concepts and practical hardware simulation skills.

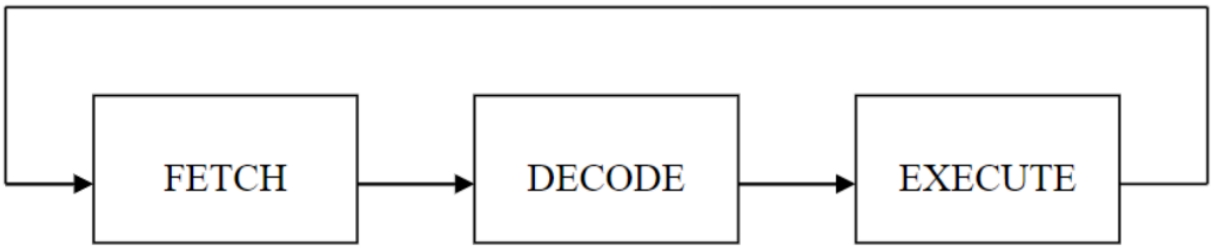
Instruction	Instruction(15 downto 0)							Operation
ADD	0001	DR	SR1	0	00	SR2		R(DR) ← R(SR1) + R(SR2)
ADDi	0001	DR	SR	1	imm5			R(DR) ← R(SR) + SEXT(imm5)
AND	0101	DR	SR1	0	00	SR2		R(DR) ← R(SR1) AND R(SR2)
ANDi	0101	DR	SR	1	imm5			R(DR) ← R(SR) AND SEXT(imm5)
NOT	1001	DR	SR	111111				R(DR) ← NOT R(SR)
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

This diagram above represents the Instruction Set Architecture (ISA) of the SLC-3 processor, listing various instructions, their bit patterns, operands, and corresponding operations. Instructions like ADD and AND support both register and immediate addressing modes, as

indicated by the presence of 'i' (immediate) in the instruction name (ADDi, ANDi). The NOT operation, uniquely represented by a series of 1s, performs a bitwise negation. Conditional branching is facilitated by the BR instruction, which utilizes status flags to determine the execution path. Unconditional jumps are achieved via the JMP instruction, directing the Program Counter (PC) to the address contained in a base register. Subroutine calls are handled by the JSR instruction, which saves the current PC to a register and then jumps to a new address. Memory access is executed through LDR (load) and STR (store) instructions, allowing data transfer between registers and memory. Lastly, the PAUSE instruction temporarily halts execution, displaying a specific LED pattern and waiting for a user input to continue. This ISA underpins the operational capabilities of the SLC-3, providing a fundamental framework for programming and interaction with the processor.

II. Written Description and Diagrams of SLC-3

The SLC-3 processor operates through three main stages: fetch, decode, and execute, which cycle continuously to operate the computer as shown in the diagram below.

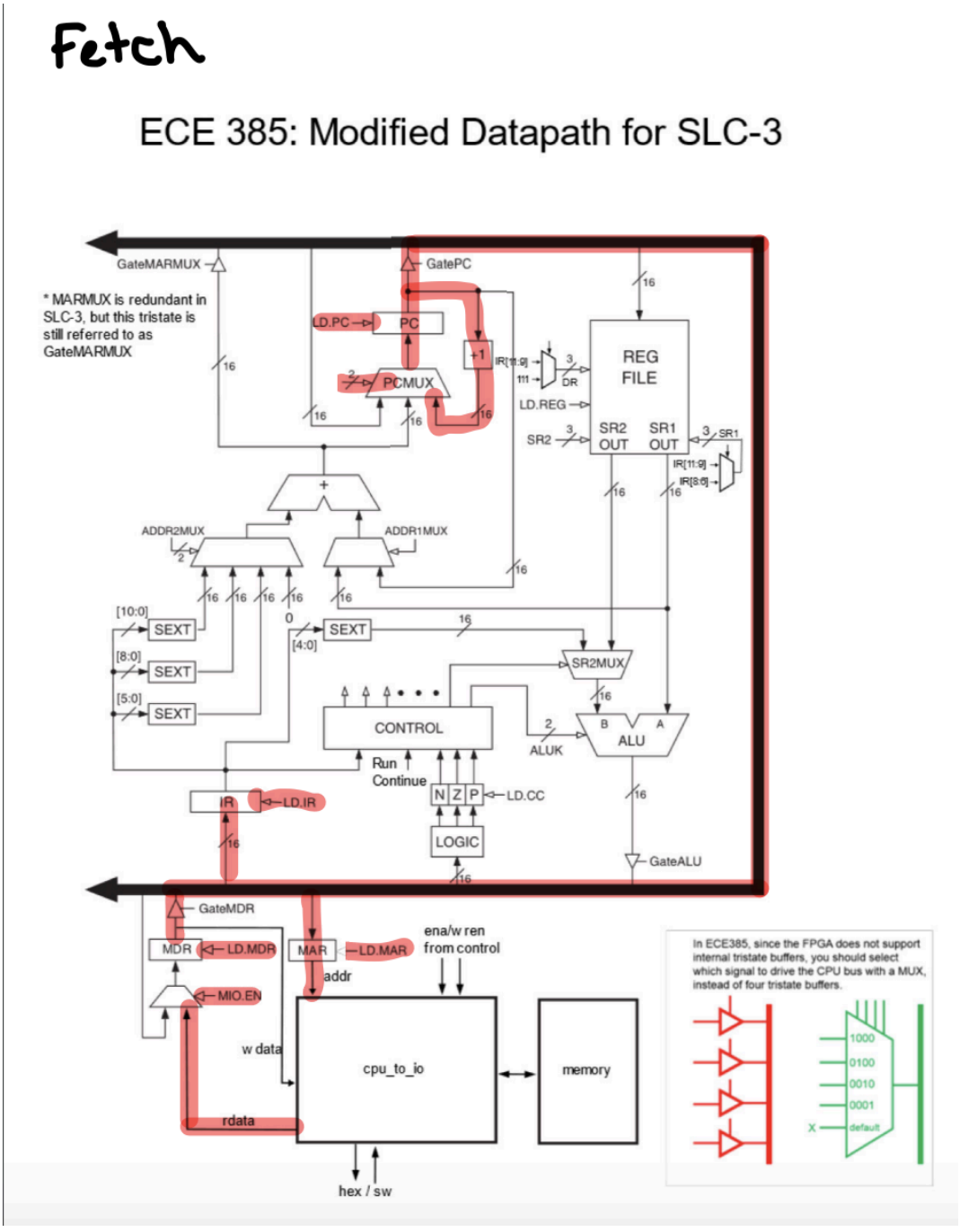


In the fetch stage, the microprocessor retrieves an instruction from memory, using the Memory Address Register to identify where to read from. This instruction is then loaded into the Memory Data Register and passed to the Instruction Register, with the Program Counter incremented to prepare for the next instruction. During the decode stage, the opcode within the instruction is interpreted by the control unit, which identifies the operation to be performed next. Finally, in the execute stage, the operation dictated by the decoded instruction is carried out, guided by signals from the control unit to the appropriate destination in registers or memory, based on the operation's requirements.

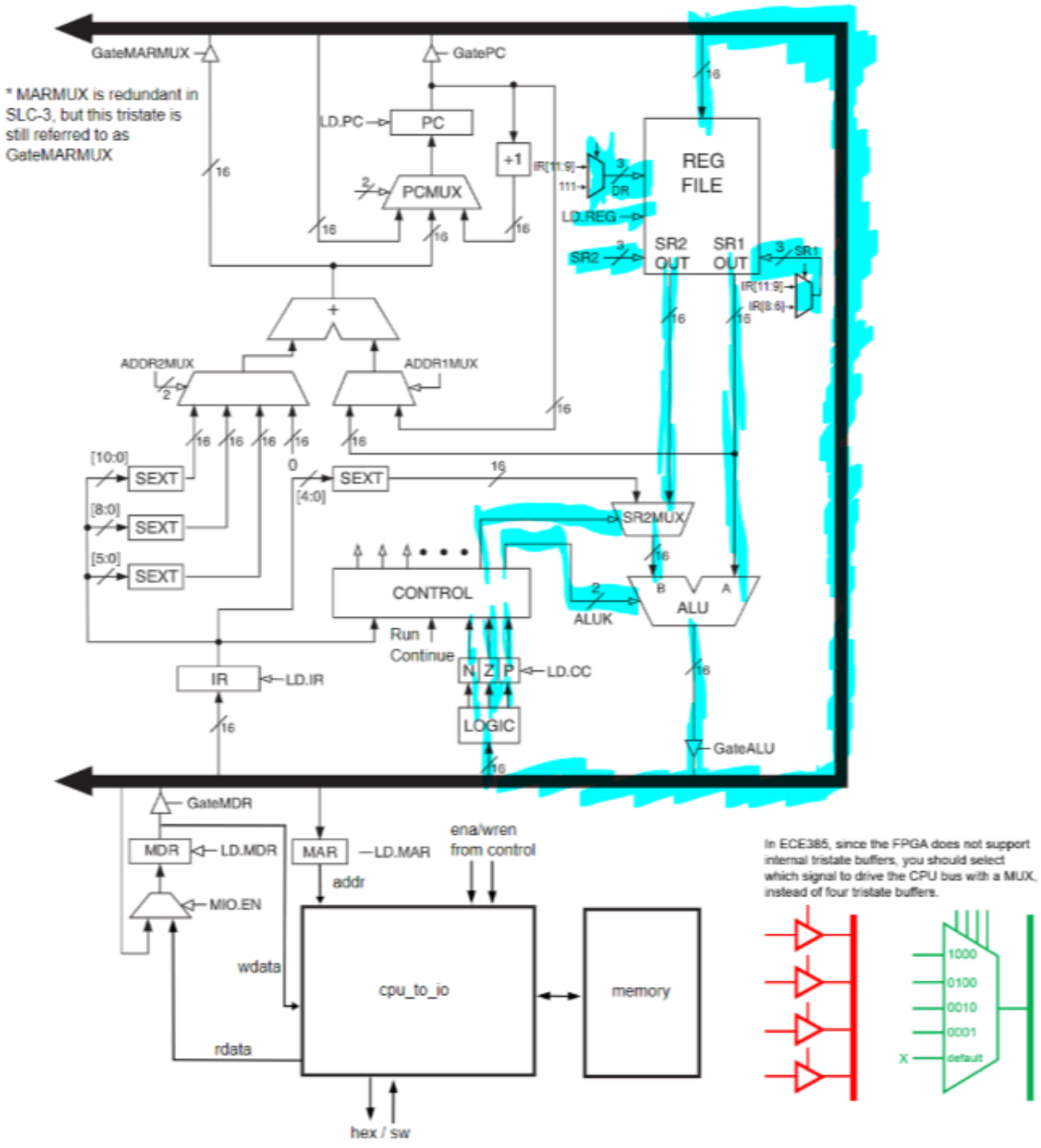
Below shows the path that the data follows for each instruction on the block diagram. It shows it for each instruction and indicates what components should be active and be in use as specified by the instruction. A similar pattern was used to write the HDL for the SLC-3 modules.

Block Diagam of Each Instruction

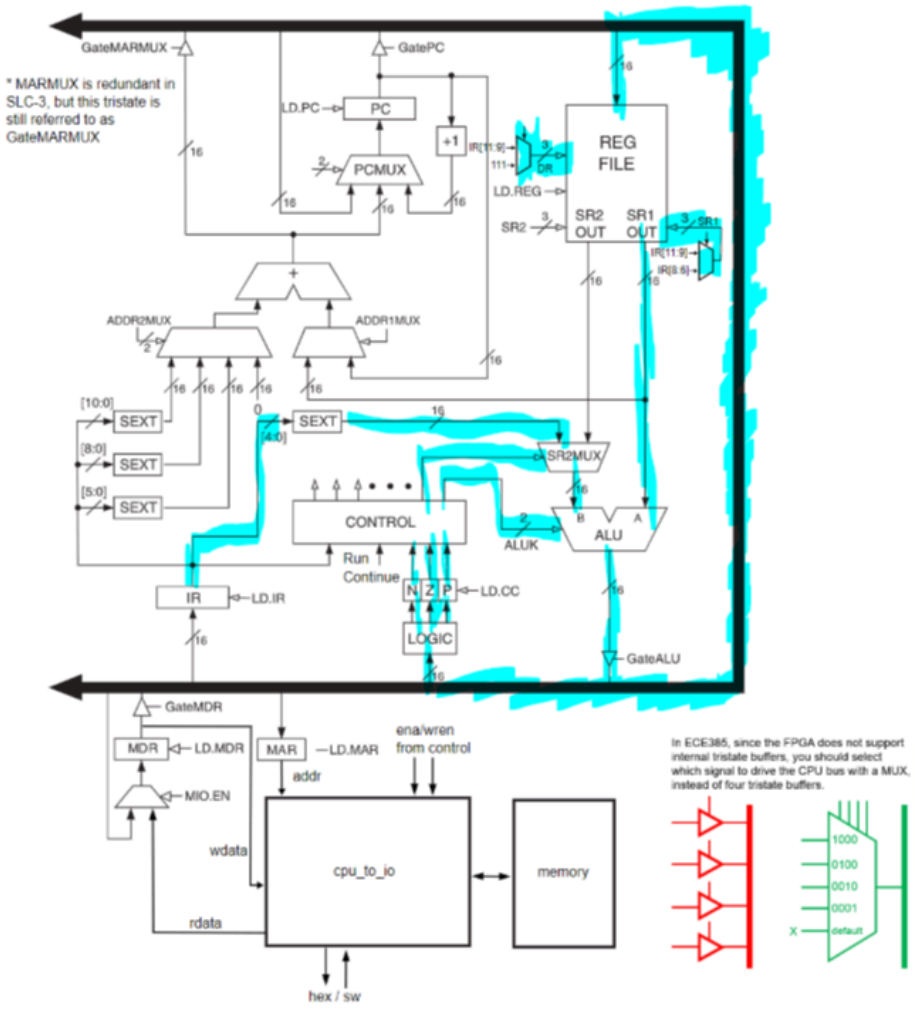
1. Fetch



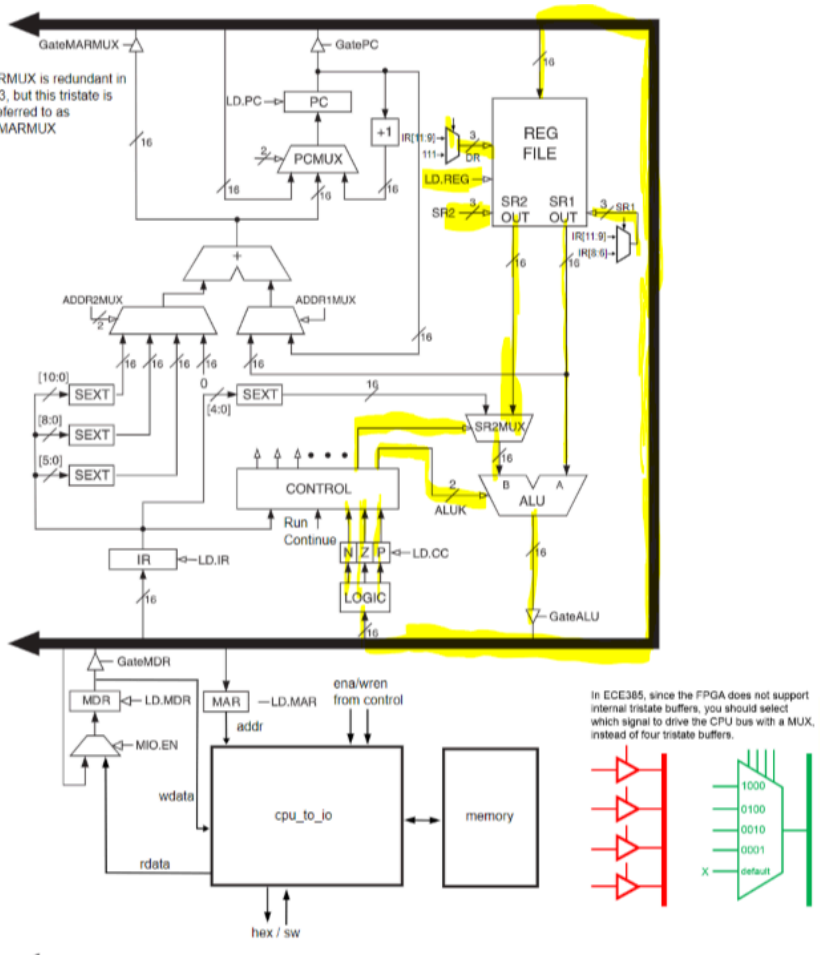
2. ADD


GateMARMUX
GatePC
* MARMUX is redundant in SLC-3, but this tristate is still referred to as GateMARMUX
LD.PC
PC
+1
PCMUX
IR[11:9]
111
DR
REG FILE
LD.REG
SR2
SR2 OUT
SR1 OUT
SR1
IR[11:9]
IR[8:6]
ADDR2MUX
ADDR1MUX
[10:0]
SEXT
[8:0]
SEXT
[5:0]
SEXT
[4:0]
SEXT
SR2MUX
CONTROL
ALUK
ALU
Run
Continue
IR
LD.IR
N Z P
LD.CC
LOGIC
GateALU
GateMDR
MDR
LD.MDR
MAR
LD.MAR
ena/wren from control
addr
MIO.EN
wdata
rdata
cpu_to_io
memory
hex / sw
In ECE385, since the FPGA does not support internal tristate buffers, you should select which signal to drive the CPU bus with a MUX, instead of four tristate buffers.
1000
0100
0010
0001
X
default

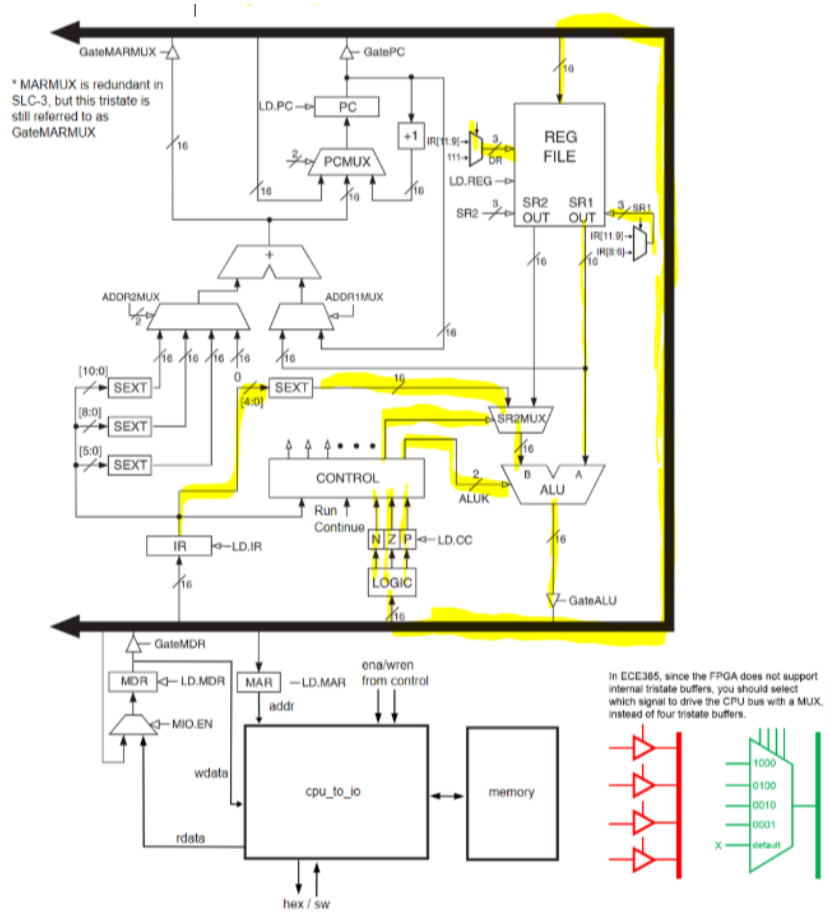
3. ADDi


GateMARMUX
* MARMUX is redundant in SLC-3, but this tristate is still referred to as GateMARMUX
GatePC
LD.PC
PC
+1
PCMUX
REG FILE
LD.REG
SR2
SR2 OUT
SR1 OUT
SR1
IR[11:9]
IR[8:6]
DR
111
ADDR2MUX
ADDR1MUX
[10:0]
[8:0]
[5:0]
SEXT
SEXT
SEXT
[4:0]
SR2MUX
CONTROL
ALUK
ALU
B
A
Run
Continue
IR
LD.IR
N Z P
LD.CC
LOGIC
GateALU
GateMDR
MDR
LD.MDR
MAR
LD.MAR
ena/wren
from control
addr
MIO.EN
wdata
rdata
cpu_to_io
memory
hex / sw
In ECE385, since the FPGA does not support internal tristate buffers, you should select which signal to drive the CPU bus with a MUX, instead of four tristate buffers.
1000
0100
0010
0001
default
X

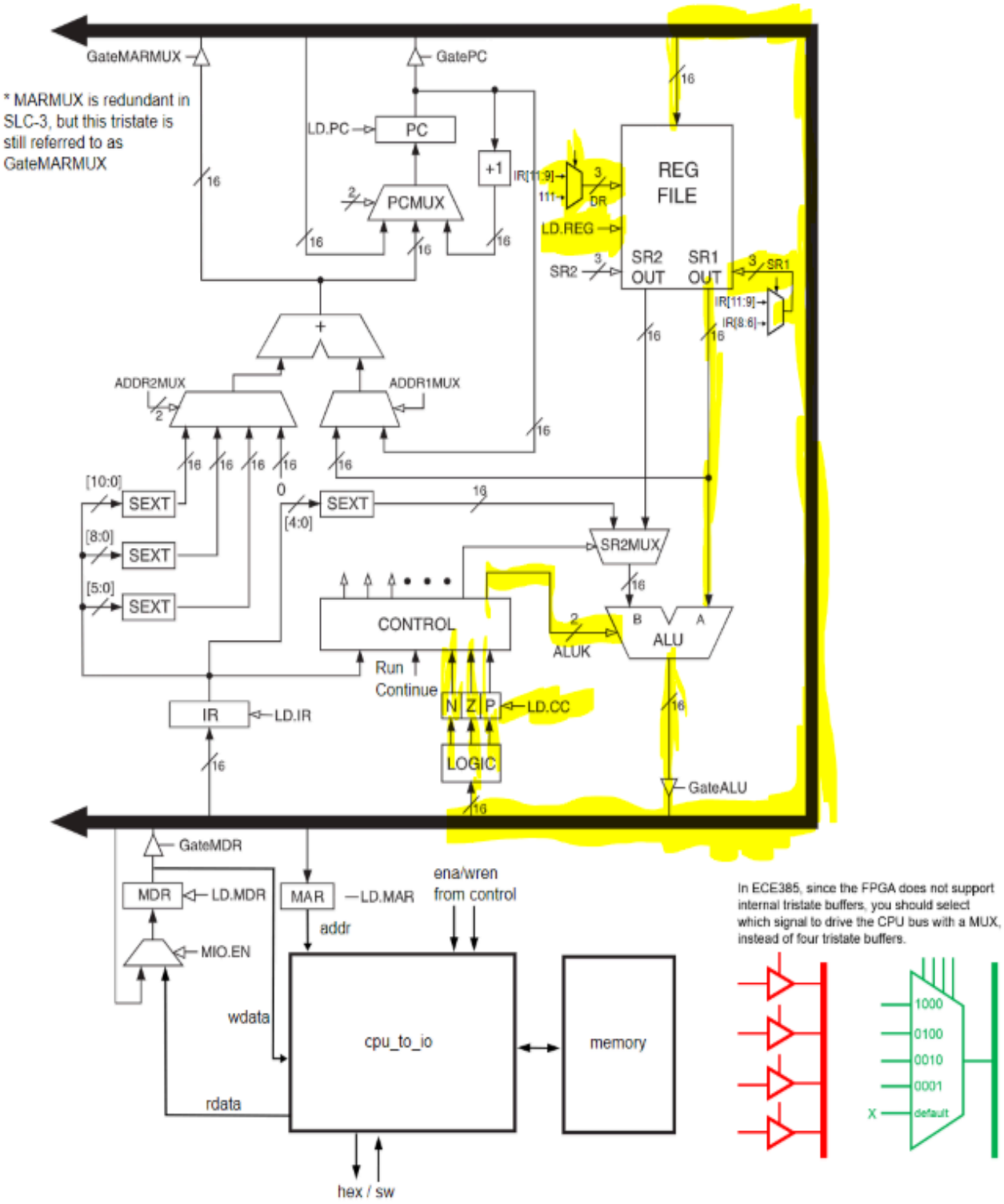
4. AND


GateMARMUX
GatePC
* MARMUX is redundant in SLC-3, but this tristate is still referred to as GateMARMUX
LD.PC
PC
+1
PCMUX
IR[11:9]
DR
REG FILE
LD.REG
SR2
SR2 OUT
SR1 OUT
SR1
IR[11:9]
IR[8:6]
ADDR2MUX
ADDR1MUX
[10:0]
[8:0]
[5:0]
SEXT
SEXT
SEXT
[4:0]
SEXT
SR2MUX
CONTROL
ALUK
B
A
ALU
Run
Continue
IR
LD.IR
N
Z
P
LD.CC
LOGIC
GateALU
GateMDR
MDR
LD.MDR
MAR
LD.MAR
ena/wren from control
addr
MIO.EN
wdata
rdata
cpu_to_io
memory
hex / sw
In ECE385, since the FPGA does not support internal tristate buffers, you should select which signal to drive the CPU bus with a MUX, instead of four tristate buffers.
1000
0100
0010
0001
X
default

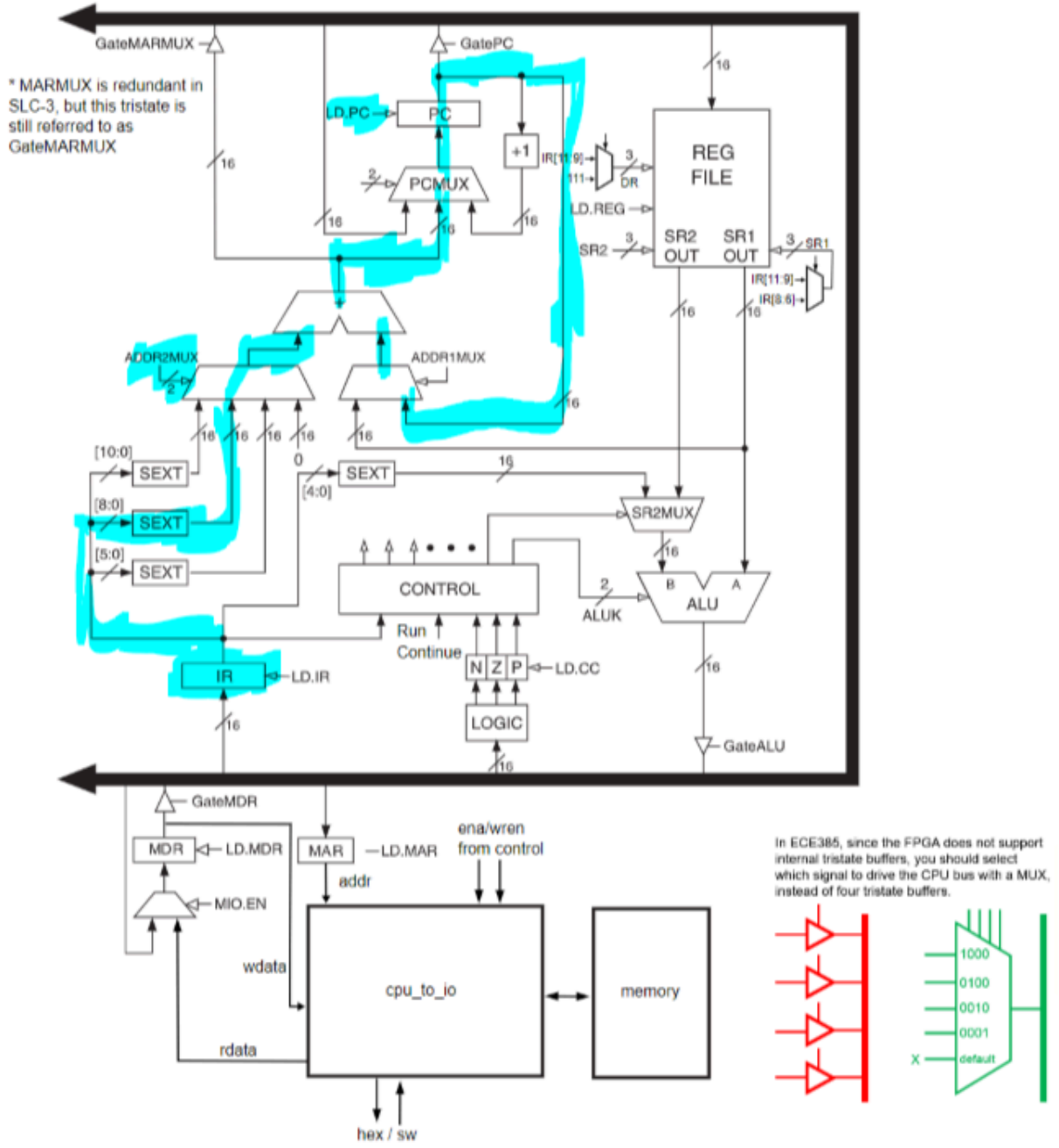
5. ANDi


GateMARMUX
* MARMUX is redundant in SLC-3, but this tristate is still referred to as GateMARMUX
GatePC
LD.PC
PC
+1
PCMUX
REG FILE
LD.REG
SR2 OUT
SR1 OUT
SR2
IR[11:9]
IR[8:6]
DR
ADDR2MUX
ADDR1MUX
SEXT
[10:0]
[8:0]
[5:0]
[4:0]
SR2MUX
CONTROL
Run
Continue
ALUK
ALU
IR
LD.IR
N Z P
LD.CC
LOGIC
GateALU
GateMDR
MDR
LD.MDR
MAR
LD.MAR
MIO.EN
addr
ena/wren
from control
wdata
rdata
cpu_to_io
memory
hex / sw
In ECE385, since the FPGA does not support internal tristate buffers, you should select which signal to drive the CPU bus with a MUX, instead of four tristate buffers.
1000
0100
0010
0001
X
default

6. NOT

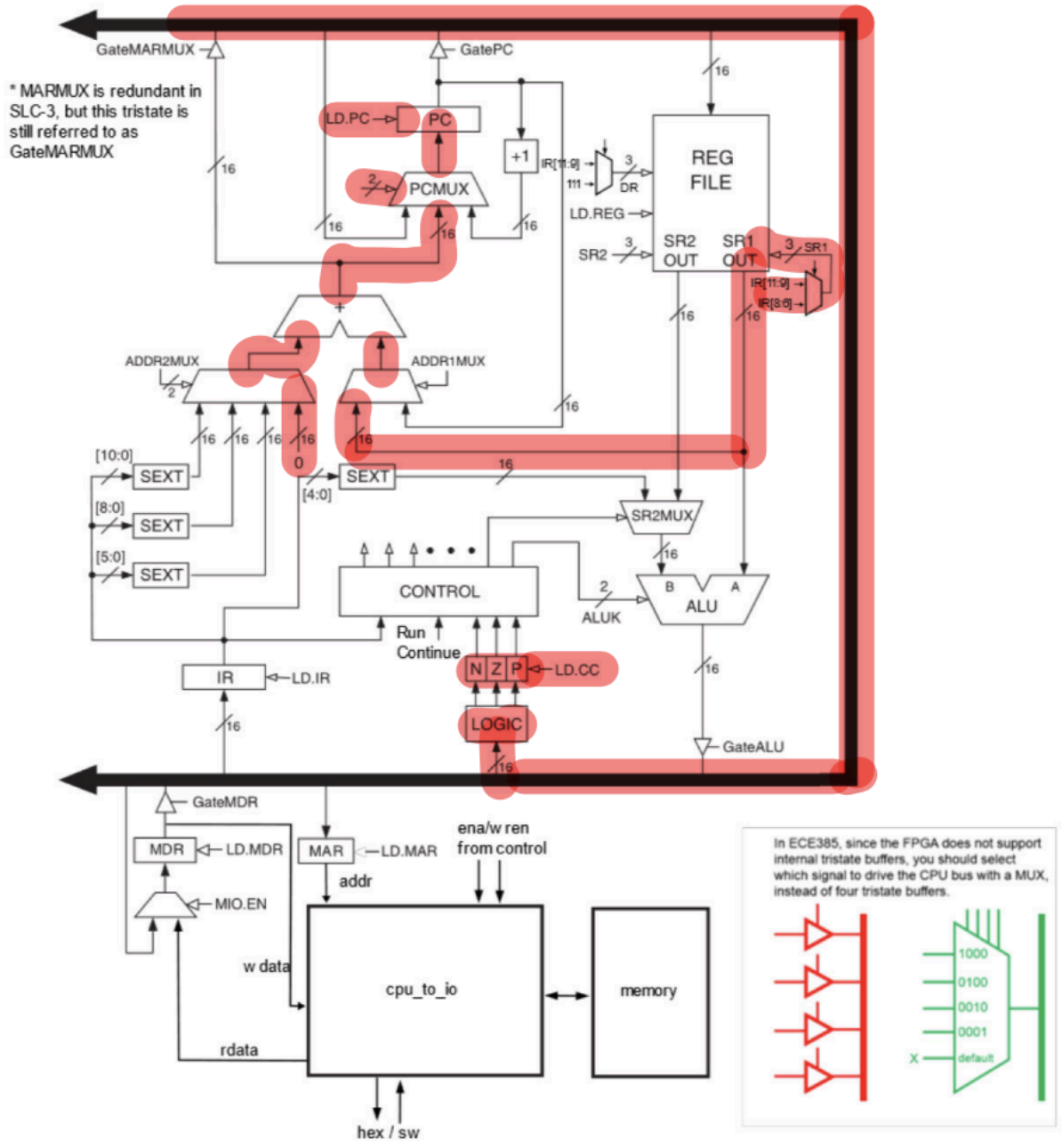

GateMARMUX
GatePC
* MARMUX is redundant in SLC-3, but this tristate is still referred to as GateMARMUX
LD.PC
PC
+1
PCMUX
IR[11:9]
111
DR
REG FILE
LD.REG
SR2 OUT
SR1 OUT
SR2
SR1
IR[11:9]
IR[8:6]
ADDR2MUX
ADDR1MUX
[10:0]
[8:0]
[5:0]
SEXT
SEXT
SEXT
[4:0]
SEXT
SR2MUX
CONTROL
ALUK
ALU
Run
Continue
IR
LD.IR
N Z P
LD.CC
LOGIC
GateALU
GateMDR
MDR
LD.MDR
MAR
LD.MAR
ena/wren
from control
addr
MIO.EN
wdata
rdata
cpu_to_io
memory
hex / sw
In ECE385, since the FPGA does not support internal tristate buffers, you should select which signal to drive the CPU bus with a MUX, instead of four tristate buffers.
1000
0100
0010
0001
X
default

7. BR


GateMARMUX
* MARMUX is redundant in SLC-3, but this tristate is still referred to as GateMARMUX
GatePC
LD.PC
PC
+1
PCMUX
IR[11:9]
111
DR
REG FILE
LD.REG
SR2
SR2 OUT
SR1 OUT
SR1
IR[11:9]
IR[8:6]
ADDR2MUX
ADDR1MUX
[10:0]
SEXT
[8:0]
SEXT
[5:0]
SEXT
0
[4:0]
SEXT
SR2MUX
CONTROL
ALUK
B
A
ALU
Run
Continue
IR
LD.IR
N
Z
P
LD.CC
LOGIC
GateALU
GateMDR
MDR
LD.MDR
MAR
LD.MAR
ena/wren
from control
addr
MIO.EN
wdata
rdata
cpu_to_io
memory
hex / sw
In ECE385, since the FPGA does not support internal tristate buffers, you should select which signal to drive the CPU bus with a MUX, instead of four tristate buffers.
1000
0100
0010
0001
X
default

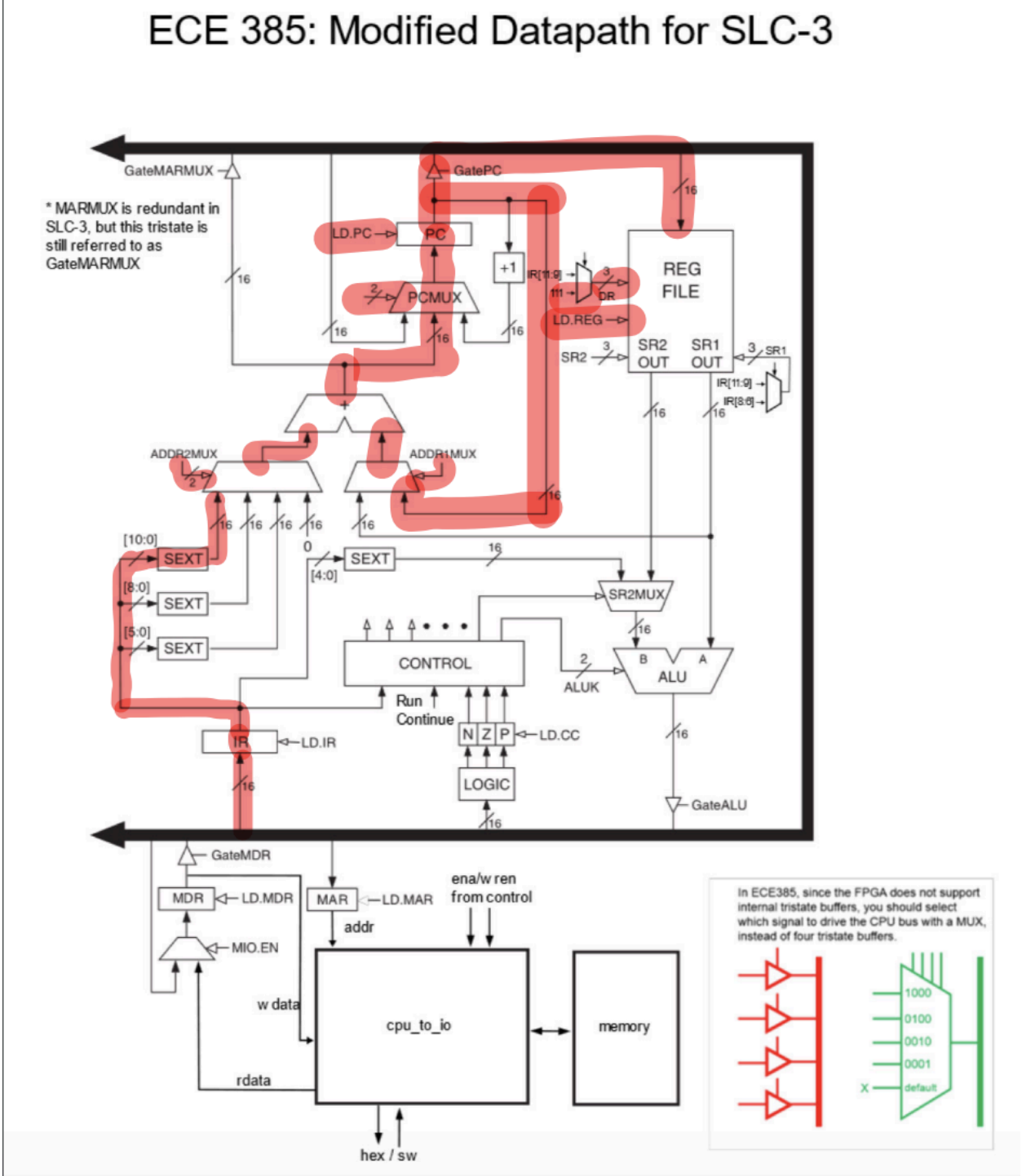
8. Jmp

Jmp


ECE 385: Modified Datapath for SLC-3
GateMARMUX
GatePC
* MARMUX is redundant in SLC-3, but this tristate is still referred to as GateMARMUX
LD.PC
PC
+1
PCMUX
IR[11:9]
111
DR
REG FILE
LD.REG
SR2
SR2 OUT
SR1 OUT
SR1
IR[11:9]
IR[8:6]
ADDR2MUX
ADDR1MUX
[10:0]
[8:0]
[5:0]
SEXT
SEXT
SEXT
0
[4:0]
SEXT
SR2MUX
CONTROL
ALUK
B
A
ALU
Run
Continue
IR
LD.IR
N Z P
LD.CC
LOGIC
GateALU
GateMDR
MDR
LD.MDR
MAR
LD.MAR
addr
ena/w ren
from control
MIO.EN
w data
rdata
cpu_to_io
memory
hex / sw
In ECE385, since the FPGA does not support internal tristate buffers, you should select which signal to drive the CPU bus with a MUX, instead of four tristate buffers.
1000
0100
0010
0001
X
default

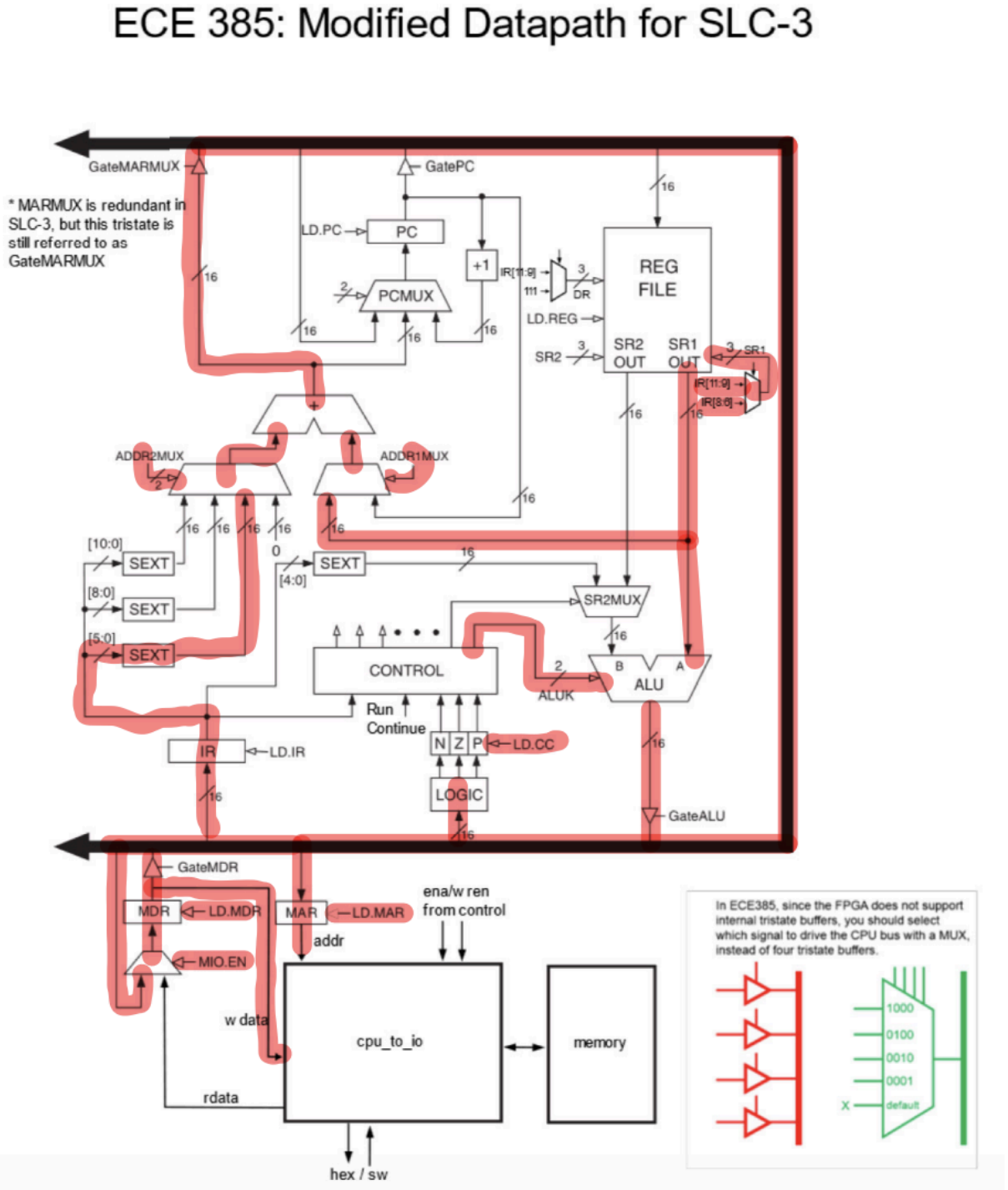
9. JSR

JSR

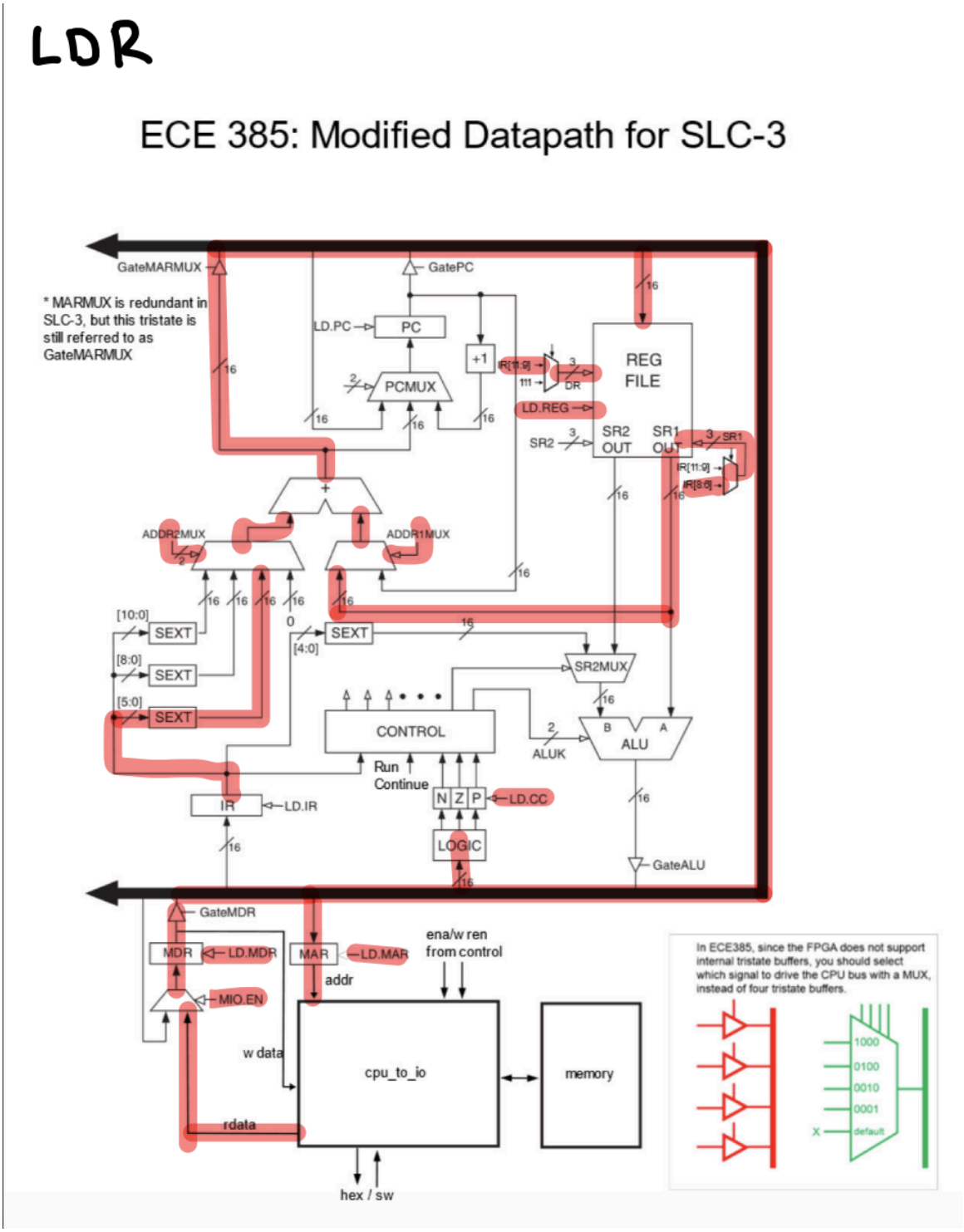

ECE 385: Modified Datapath for SLC-3
GateMARMUX
* MARMUX is redundant in SLC-3, but this tristate is still referred to as GateMARMUX
GatePC
LD.PC
PC
+1
PCMUX
IR[11:9]
111
DR
REG FILE
LD.REG
SR2
SR2 OUT
SR1 OUT
SR1
IR[11:9]
IR[8:6]
ADDR2MUX
ADDR1MUX
[10:0]
SEXT
[8:0]
SEXT
[5:0]
SEXT
0
[4:0]
SEXT
SR2MUX
CONTROL
ALUK
B
A
ALU
Run
Continue
IR
LD.IR
N
Z
P
LD.CC
LOGIC
GateALU
GateMDR
MDR
LD.MDR
MAR
LD.MAR
ena/w ren
from control
addr
MIO.EN
w data
rdata
cpu_to_io
memory
hex / sw
In ECE385, since the FPGA does not support internal tristate buffers, you should select which signal to drive the CPU bus with a MUX, instead of four tristate buffers.
1000
0100
0010
0001
X
default

10. STR

STR

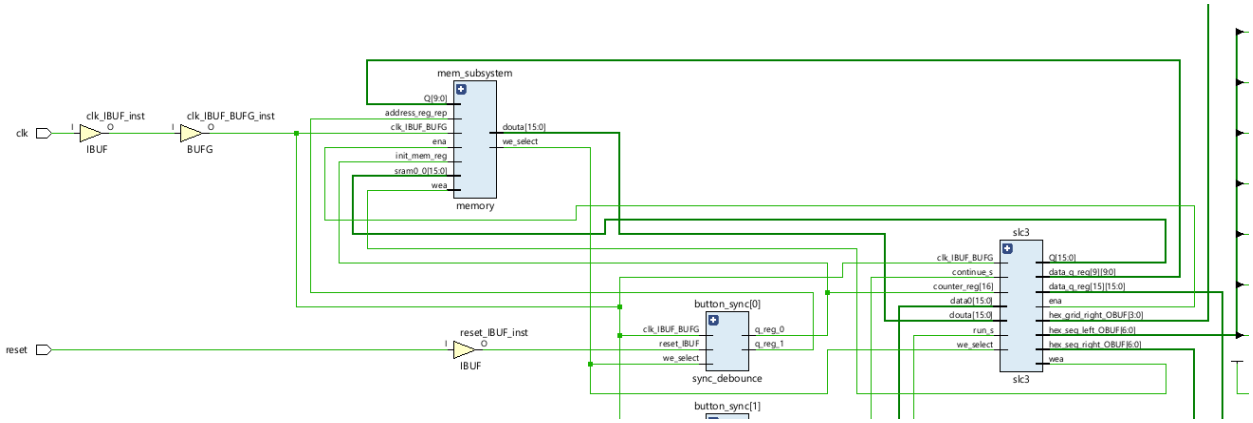

ECE 385: Modified Datapath for SLC-3
GateMARMUX
* MARMUX is redundant in SLC-3, but this tristate is still referred to as GateMARMUX
GatePC
LD.PC
PC
+1
PCMUX
REG FILE
LD.REG
SR2 OUT
SR1 OUT
SR2
SR1
IR[11:9]
IR[8:0]
DR
111
ADDR2MUX
ADDR1MUX
[10:0]
[8:0]
[5:0]
SEXT
[4:0]
SR2MUX
CONTROL
ALU
ALUK
Run
Continue
IR
LD.IR
N Z P
LD.CC
LOGIC
GateALU
GateMDR
MDR
LD.MDR
MAR
LD.MAR
MIO.EN
addr
ena/w ren from control
w data
rdata
cpu_to_io
memory
hex / sw
In ECE385, since the FPGA does not support internal tristate buffers, you should select which signal to drive the CPU bus with a MUX, instead of four tristate buffers.
1000
0100
0010
0001
X
default

11. LDR

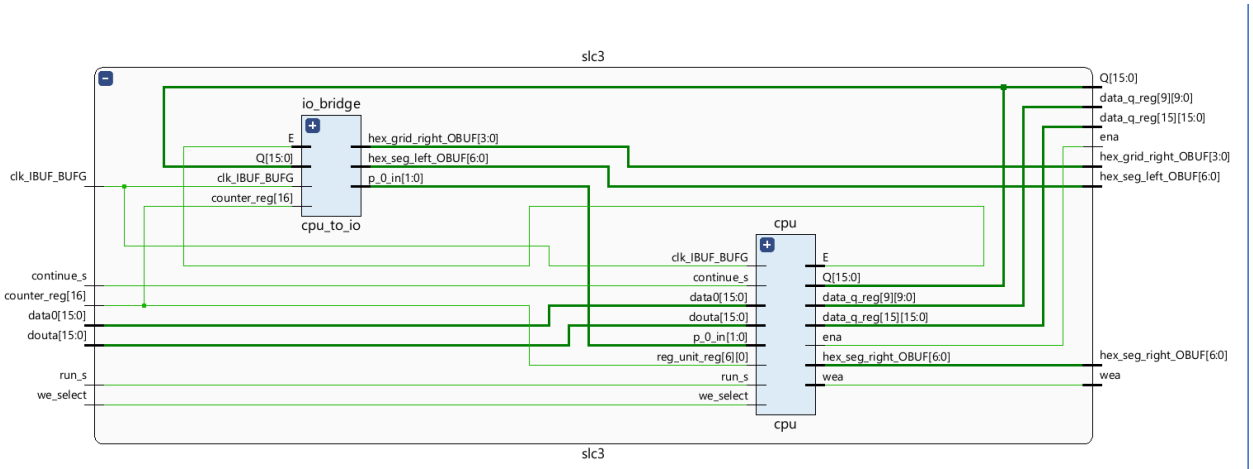


12. Pause
    a. The pause instruction is handled in the control unit so nothing changes

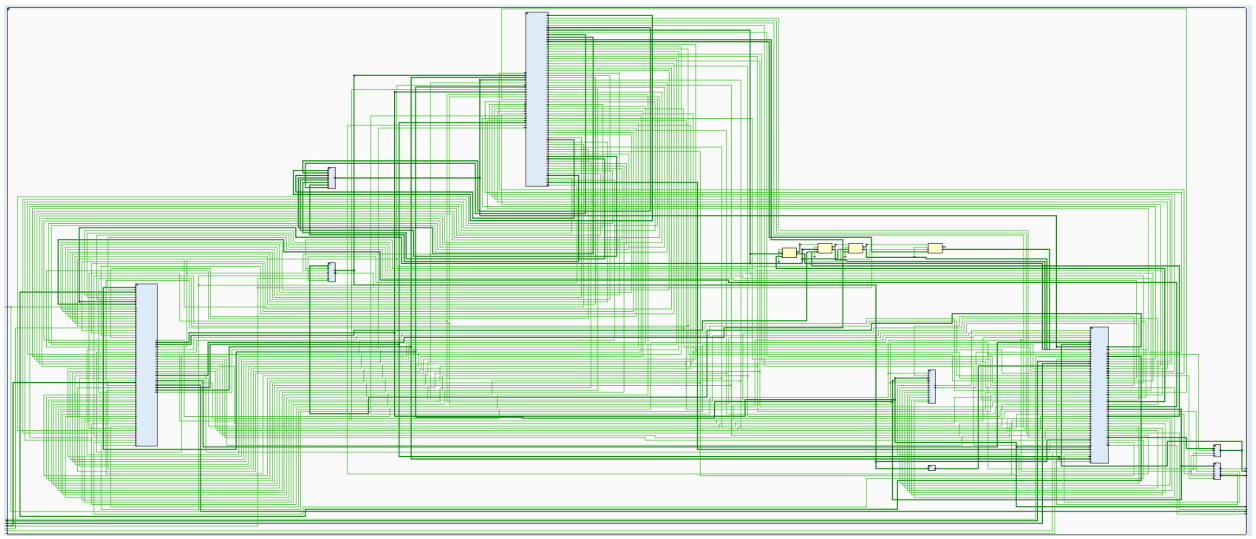
Block Diagrams of Modules



Entire Processor



SLC3 Module



CPU Module

Written description of sv modules

hex_driver.sv
Inputs: [3:0] nibble,
Outputs: [7:0] hex

Description: Takes 4 bits as input and outputs 7 bits that correspond to the outputs of the FPGA LED part. This module takes in 4 bits and displays the equivalent value in hexadecimal on the FPGA board

Purpose: The purpose of this module is to convert the binary values to hexadecimal and display them on the FPGA board for the user to see.

sync.sv
Inputs: Clk, d
Outputs: q

Description: Keeps the logic functions synchronized with the clock, allowing combinational and sequential logic to work

Purpose: Synchronizes the asynchronous signals on the FPGA

instantiate_ram.sv
Inputs: reset, clk
Outputs: [9:0] addr, wren, [15:0] data

Description: Instantiates memory into the FPGA board as well as the test contents
Purpose: Allocate memory for various test cases

alu.sv
Inputs: [15:0] A, [15:0] B, [1:0] aluk,
Outputs: [15:0] R

Description: 2 input MUX that takes in 16 bits
Purpose: The ALU unit in the datapath. Takes in the values from the SR2 mux and SR1 out

branch.sv
Inputs: reset, clk, [15:0] ir, [15:0] Bus, Ld_ben, ld_cc,
Outputs: Ben

Description: Determines if there is a need to branch instruction
Purpose: Sets the cc register to branch based on opcode from the nzp function

BusMux
Inputs: [15:0] MDR, [15:0] ALU, [15:0] PC, [15:0] MARMUX, GateMDR, GateALU, GatePC, GateMARMUX
Outputs: [15:0] BUS

Description: 5 input MUX built with case statements
Purpose: Used to regulate the BUS control signals without using tri state buffers

mux21
Inputs: [15:0] a, [15:0] b, mio_en,
Outputs: [15:0] out

Description: A 16-bit 2 to 1 mux
Purpose: 2 to 1 mux used for SR2 mux and ADDR1MUX

bit321mux
Inputs: [2:0] a,b, enablebit
Outputs: [2:0] out

Description: A 3-bit 2 to 1 mux
Purpose: Used for SR1 mux and DR mux in datapath

muxadder2
Inputs: [15:0] a, [15:0] b, [15:0] c, [15:0] d, [1:0] addr2enable,
Outputs: [15:0] out

Description: A 16-bit 4 to 2 mux
Purpose: Takes various sign extended data from the IR and chooses between them based on the select bits

pcmux.sv
Inputs: [15:0] pc_original, [15:0] BUS, [15:0] adder, pc_select
Outputs: [15:0] pc

Description: 3 input 16-bit mux to determine the new PC value
Purpose: Takes in the data from the bus, adder, and incremented PC and determines the new PC value

regfile.sv
Inputs: [15:0] bus, [2:0] sr1mux, [2:0] sr2mux, [2:0] drmux, ld_reg, clk, reset
Outputs: [15:0] regout1, [15:0] regout2

Description: Used to implement the register file and which signal to route outside (regout1 or regout2) depending on the DR and SR1 mux

Purpose: Create the 8-bit register from the register file while changing the BUS and outputting regout1 and regout2

control.sv

Inputs: clk, reset, [15:0] ir, ben, [15:0] mem_rdata, continue_i, run_i
Outputs: ld_mar, ld_mdr, ld_ir, ld_ben, ld_cc, ld_reg, ld_pc, ld_led, gate_pc, gate_mdr, gate_alu, gate_marmux, [1:0] pcmux, drmux, sr1mux, sr2mux, addr1mux, [1:0] addr2mux, [1:0], aluk, mio_en, mem_mem_ena, mem_wr_ena

Description: Implementation of the state machine. The state machine was built using the SLC-3 datapath
Purpose: Determines the next state based on the current state and the user's input. Generates control signals for the muxes based on the current state (see description for more information).

cpu_to_io.sv

Inputs: clk, reset, [15:0] cpu_addr, cpu_mem_ena, cpu_wr_ena, [15:0] cpu_wdata, [15:0] sram_rdata, [15:0] sw_i
Outputs: [15:0] cpu_rdata, [15:0] sram_addr, sram_mem_ena, sram_wr_ena, [15:0] sram_wdata, [3:0] hex_grid_o, [7:0] hex_seg_o

Description: Used to move memory between the CPU and the IO devices. Allows for the switches to be mapped to xFFFF and to display the data on hex displays
Purpose: Helps to memory map the CPU to the I/O for testing

cpu.sv

Inputs: clk, reset, run_i, continue_i, [15:0] mem_rdata
Outputs: [15:0] hex_display_debug, [15:0] led_o, [15:0] mem_wdata, [15:0] mem_addr, mem_mem_ena, mem_wr_ena

Description: Top level diagram for the entire datapath. Instantiate all of the muxes, registers, and the control unit
Purpose: Create the SLC-3 datapath by defining all of the components

slc3.sv

Inputs: clk, reset, run_i, continue_i, [15:0] sw_i, [15:0] sram_rdata
Outputs: [15:0] led_o, [7:0] hex_seg_o, [3:0] hex_grid_o, [7:0] hex_seg_debug, [3:0] hex_grid_debug, [15:0] sram_wdata, [15:0] sram_addr, sram_mem_ena, sram_wr_ena

Description: Top level module for the cpu_to_io and cpu module
Purpose: Instantiate cpu_to_io and cpu module for use in the FPGA

processor_top.sv

Inputs: clk, reset, run_i, continue_i, [15:0] sw_i
Outputs: [15:0] led_o, [7:0] hex_seg_left, [3:0] hex_grid_left, [7:0] hex_seg_right, [3:0] hex_grid_right

Description: Instantiates all of the other modules by connecting them together. Helps with the hex display to show the correct answer in the LEDs
Purpose: Connects all of the modules together by instantiating all of them

test_memory.sv
Inputs: clk, reset, [15:0] data, [9:0] address, ena, wren
Outputs: [15:0] readout

Description: Created to simulate on board memory for the FPGA. Will synthesize to a blank module
Purpose: Simulate on-chip memory for testing

types.sv
Inputs: N/A
Outputs: N/A

Description: Holds all of the memory that is written onto the FPGA chip
Purpose: Hold memory contents for the FPGA

load_reg.sv
Inputs: clk, reset, load, [DATA_WIDTH-1:0] data_i
Outputs: [DATA_WIDTH-1:0] data_q

Description: A loadable register used for later computations
Purpose: Create the register what is provided in the datapath

memory.sv
Inputs: clk, reset, [15:0] data, [9:0] address, ena, wren
Outputs: [15:0] readout

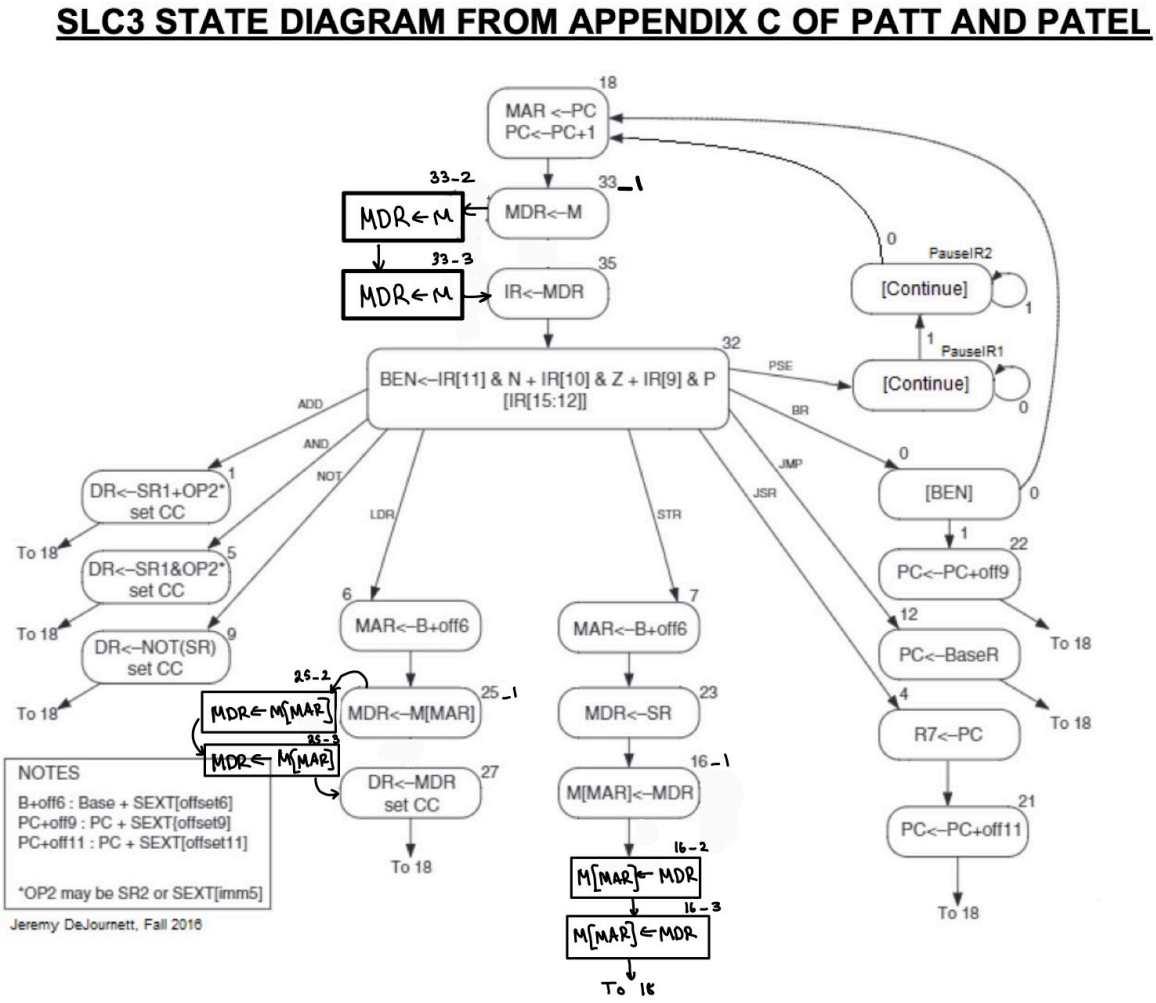
Description: Instantiates memory to be used in the FPGA
Purpose: Allocate memory on the FPGA for memory based operations on the SLC-3

Description of the operation of the control unit

The control unit plays a pivotal role in the SLC-3 processor by managing state transitions, guided by the LC-3's state diagram as detailed in Appendix C of Patt and Patel's documentation. To facilitate this, the control unit utilizes two logic variables to track the current and subsequent states. Initially, it sets default values for control signals, gates, multiplexers, and memory controls. The process then progresses through a sequence of predefined states, namely 18, 33, 35, and 32, to prepare for instruction decoding. Due to the need to accommodate logic computation delays, certain states are repeated within the sequence. At state 32, the control unit is poised to decode the incoming instruction through a selective process that examines the instruction's last four bits to identify the appropriate next state. Depending on this determination, the control unit adjusts control values to modulate the datapath's components, effectively dictating the

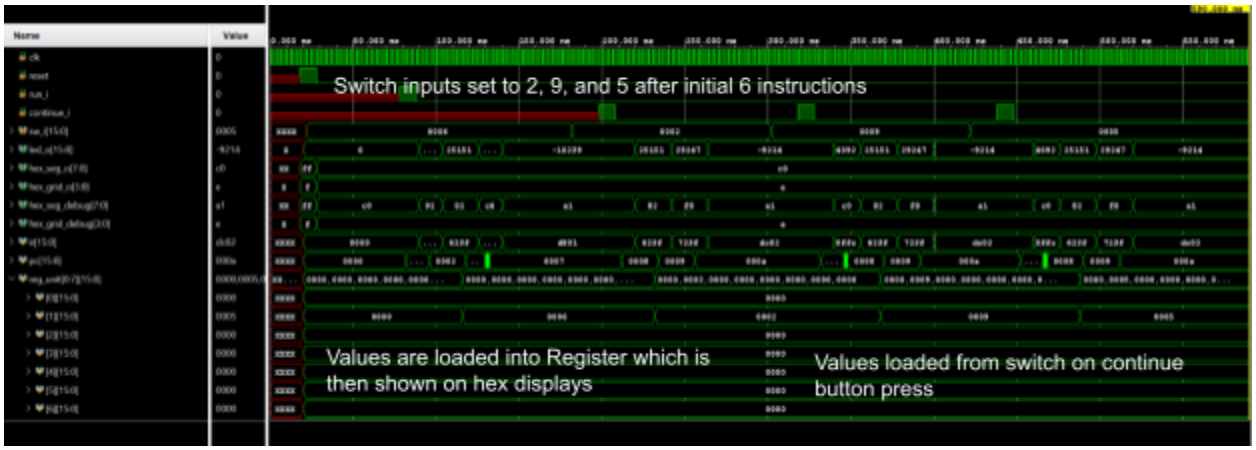
subsequent operations of the computer. Thus, the control unit is integral to determining the SLC's operational flow and functionality.

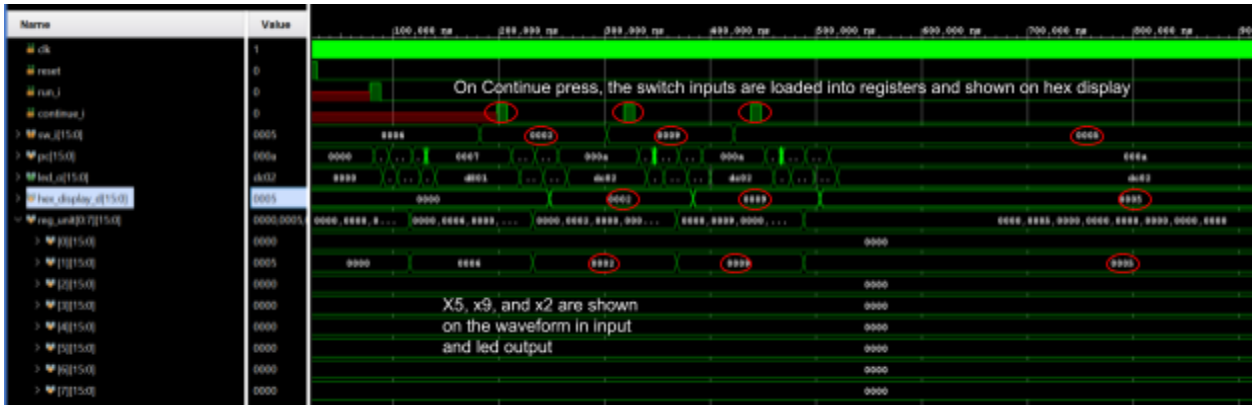
State Diagram of Control Unit



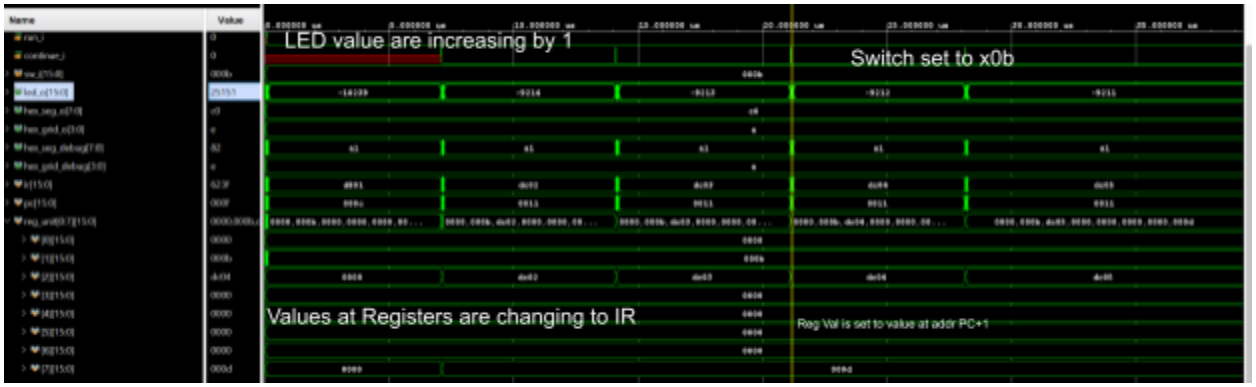
III. Simulations of SLC-3

I/O Test 2

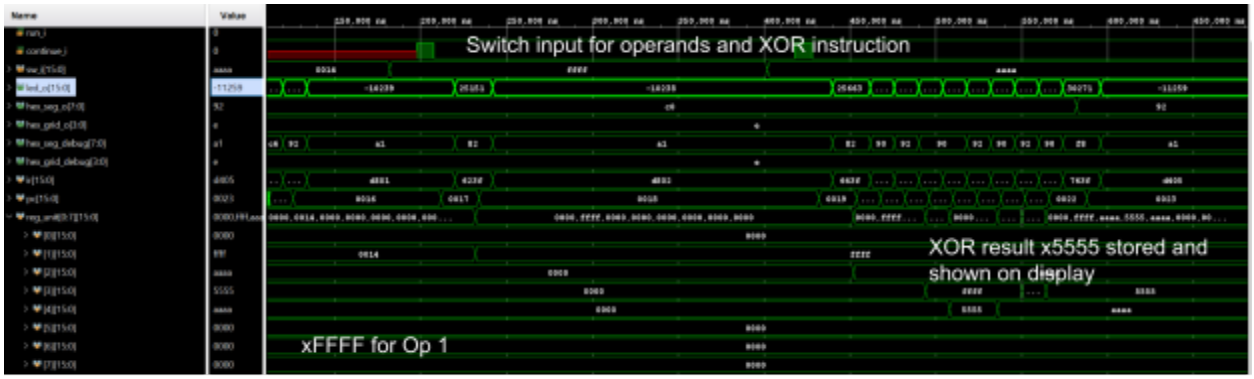




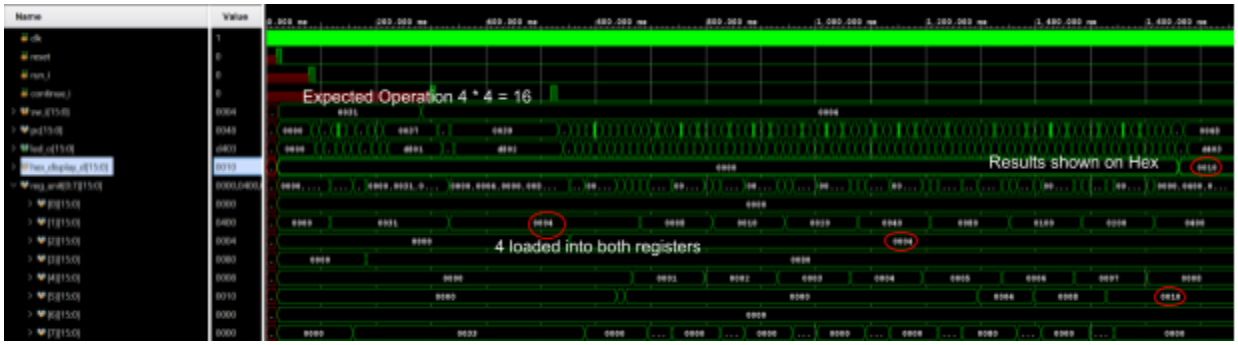
Self-Modifying Code



XOR

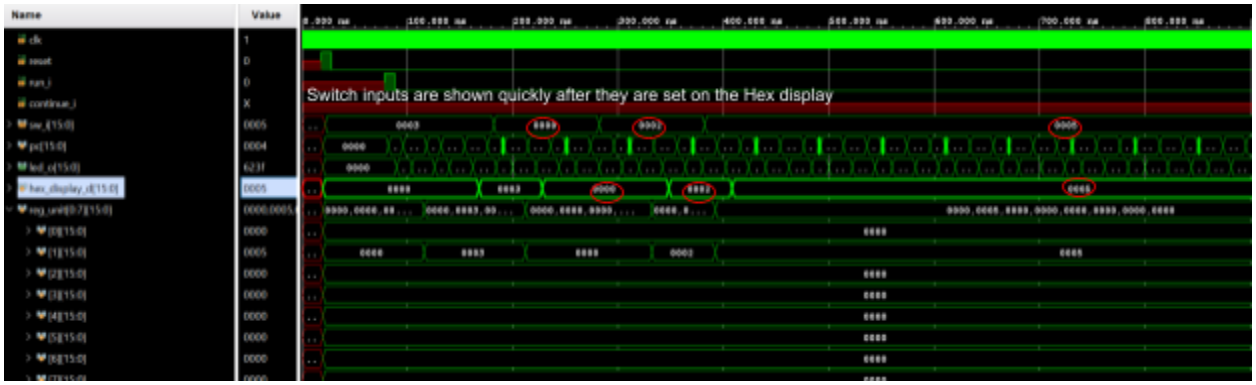


Multiplier



I/O Test 1

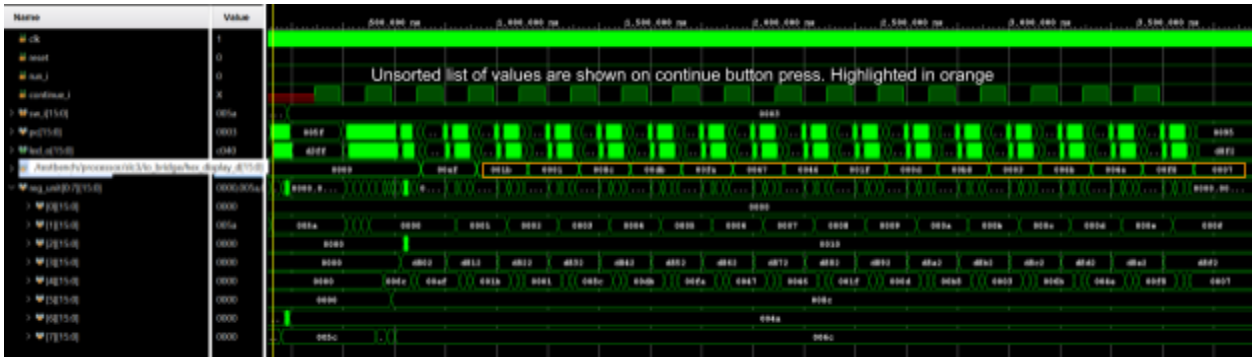

Switches set to 3, 5, 2, and 0
Switch inputs are loaded into R2 after being cleared to 0
Outputs loaded into Hex Display to be shown

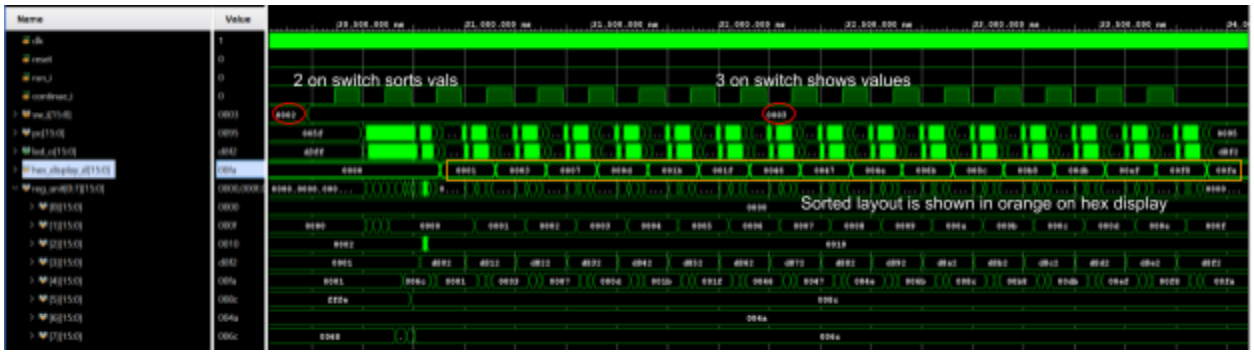

Switch inputs are shown quickly after they are set on the Hex display

Auto Count


9C is loaded to switches
R1 stores counted values to Hex during Auto Count

Sort





IV. Post Lab Questions

1. Design Resources and Statistics table

LUT	402
DSP	0
Memory(BRAM)	1
Flip Flop	346
Frequency (MHz)	97.83
Static Power (mW)	71
Dynamic Power(mW)	14
Total Power(mW)	85

2. What is the main function of CPU_TO_IO?

The CPU_TO_IO module manages the communication between the CPU and external memory-mapped peripheral devices, such as switches and 7-segment displays, by mapping a specific memory address (0xFFFF) to these I/O devices. When accessing memory location 0xFFFF, a read operation retrieves switch values into the Memory Data Register (MDR), and a write operation updates the hex displays with the value stored in the MDR. It facilitates the process of reading input from switches and displaying output on the hex displays by monitoring the memory address register (MAR) for access to this designated memory location.

3. What distinguishes the BR instruction from the JMP instruction?

The BR (Branch) instruction differs from the JMP (Jump) instruction primarily in their onditions for execution and their effects on the Program Counter (PC). The BR instruction is conditional and allows the program to branch to a different instruction based on the current status flags (N, Z, P) in the status register; it adds a sign-extended offset to the PC if the condition is met. In contrast, the JMP instruction is unconditional, transferring control directly to the address specified by a base register (BaseR) or the return address stored in a register, effectively setting the PC to this new address without any condition

4. 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 impact does this have on performance?

The R signal indicates the readiness of a memory read/write operation. It signals when the data is available for reading from or writing to memory. No R signal is compensated through additional wait states or cycles in the memory access process to ensure that the read or write operation is complete before proceeding. This compensation can impact performance by increasing the latency of memory operations, as the system must pause for a fixed number of cycles, regardless of the actual speed of the memory operation, potentially slowing down the overall execution of instructions

V. Conclusion

In Lab 5, we learned a lot about creating the SLC-3 processor, which was challenging. We faced difficulties understanding the processor's workings including the implementation of the various components like the PC and the register unit and had some coding mistakes. One area we challenges in was with the branching component. Using testbenches, we could see where we went wrong and fix our errors. We were able to identify faulty logic in the branching component

and resolve it because of the simulation. Some parts of the lab guide could be clearer, especially on the control unit and using behavioral HDL. Overall, the lab was very useful, teaching us about timing and simplifying code with behavioral HDL. It provided us with an opportunity to learn and grasp complex topics in HDL while introducing it in a friendly manner.