ECE 385

Fall 2023

Experiment #5

Simple Computer SLC-3.2 in SystemVerilog

Ziyi Lin
Zhuoyang Lai
TA: Jerry Wang

1.Introduction

Create SLC3 (Simplified LC3) microprocessor in 16-Bit Data Path Memory-mapped I/O (only mapped peripheral is HEX displays using Mem2IO) Register File (8 registers with control) Other Registers PC, IR, MAR, MDR, nzp status register ALU and Memory Instructions Add, Sub, Logical Ops, Load, Store Control Flow instructions Branch and Jump Subroutine.

2.Written Description and Diagrams of SLC-3a

a.

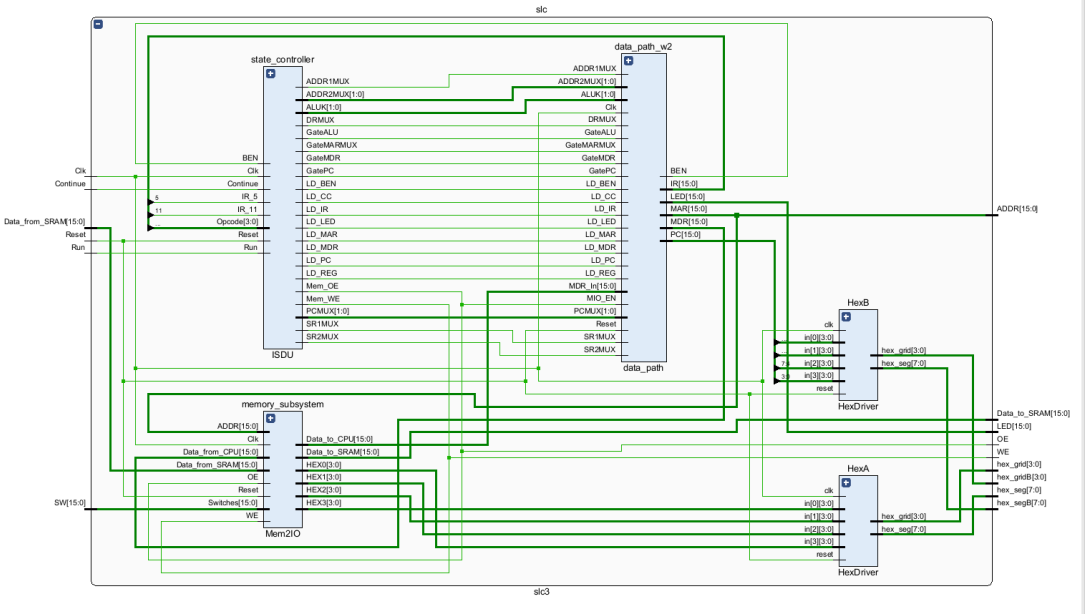
Create an SLC3 (Simplified LC3) microprocessor in SystemVerilog with a 16-bit data path, memory-mapped I/O (only mapped peripheral is HEX displays using Mem2IO), a register file (8 registers with control), and other registers such as PC, IR, MAR, MDR, and the nzp status register. Implement ALU and memory instructions including Add, Sub, Logical Operations, Load, and Store. Additionally, include control flow instructions like Branch and Jump Subroutine.

b.

instruction cycle in three main phases
Phase1: FETCH: MAR <- PC; IR <- Read Memory; PC <- PC+1;
The fetch loads the pc value into memory addresses registers (with increment of pc+1) then loads data from SRAM to memory data registers. Then write it to the IR register.
Phase2: DECODE: Decode opcode from IR; Compute Effective Address
The [15:12] bit of IR determines the type of instruction (ex. ADD, ADDi), the rest of the opcode will determine the source registers and the signal will be loaded into Mux.
Phase3: EXECUTE: Fetch Operand; Execute operation, Store Result
Based on the control signals that we decode in pahse2, all the excution will be performed relying on the state diagram. The output data will be stored either in register or memory depending on the opcode.

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 + 1 + SEXT(PCoffset9)
JMP	1100	000		BaseR	000000			PC ← R(BaseR)
JSR	0100	1	PCoffset11					R(7) ← PC + 1; PC ← PC + 1 + 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

c. (and d.)



(2.c slc3 Block Diagram)

e.(and f)

Module: slc3_sramtop

Inputs: [15:0] SW, Clk, Reset, Run, Continue,

Outputs: [15:0] LED, [7:0] hex_seg, [3:0] hex_grid, [7:0] hex_segB, [3:0] hex_gridB

Description: This module is the top level of the project which integrate everything, and output to hex displays and LEDs.

Purpose: Used as top level file.

Module: slc3_testtop
Inputs: [15:0] SW, Clk, Reset, Run, Continue,
Outputs: [15:0] LED, [7:0] hex_seg, [3:0] hex_grid, [7:0] hex_segB, [3:0] hex_gridB
Description: This module is the top level of the project which integrate everything, and output to hex displays and LEDs.
Purpose: Used as top level file(only in simluation).

Module: Instantiateram
Inputs: Reset, Clk
Outputs: [15:0] ADDR, wren, [15:0] data
Description: This is a module with prewritten memory which will load physical memory into FPGA board
Purpose: Used to test our design with I/O program preloaded in this file, it could also be modified to execute our own program to debug or anything.

Module: sync
Inputs: d, Clk
Outputs: q
Description: This module is used be synchronize the button input
Purpose: Used to synchronize the Continue and Run button

Module: sync
Inputs: d, Clk
Outputs: q
Description: This module is used be synchronize the button input
Purpose: Used to synchronize the Continue and Run button

Module: HexDriver
Inputs: clk, reset, [3:0] in[4],
Outputs: [7:0] hex_seg, [3:0] hex_grid
Description: This module is used to drive the hex display
Purpose: Used to display the output and for debug.

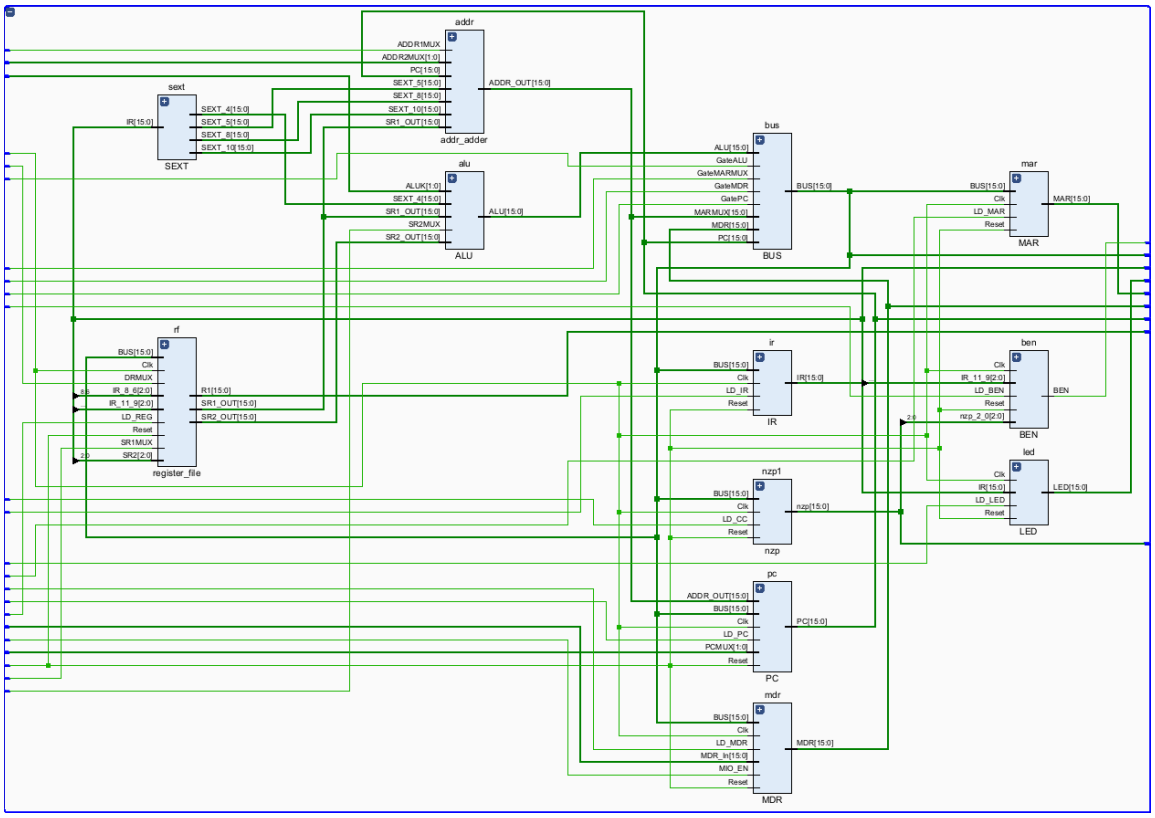
Module: data_path
Inputs:Clk, Reset, MIO_EN, [15:0]BUS, [15:0] MDR_In,
DRMUX,SR1MUX,SR2MUX,ADDR1MUX,
[1:0] PCMUX, ALUK, ADDR2MUX,

LD_MDR, LD_MAR, LD_IR,LD _PC,LD_REG,LD_CC,LD_BEN,LD_LED, GateMDR, GatePC, GateALU, GateMARMUX,

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

Description: This module includes all nearly all the components of the slc3-cpu and is driven by signals coming from control unit e.g., ISDU module

Purpose: gather nearly all the component in this file to keep the project clean and in order.



(2.e data path block diagram)

Module: MDR

Inputs: Clk,Reset, MIO_EN,15:0] BUS, [15:0] MDR_In, LD_MDR

Outputs: [15:0] MDR

Description: Include MDR register and mux near MDR

Purpose: integrate MDR and MUX for MDR to select data and output MDR which content memory data.

Module: MAR

Inputs: Clk,Reset,15:0] BUS, LD_MAR

Outputs: [15:0] MAR

Description: Include MAR register

Purpose: update and output MAR which contains memory address

Module: PC
Inputs: Clk, Reset, [1:0]PCMUX, LD_PC, [15:0] BUS, [15:0] ADDR_OUT
Outputs: [15:0] PC
Description: Include PC register and MUXs for PC register
Purpose: update and output PC which contains program count

Module: IR
Inputs: Clk,Reset,15:0] BUS, LD_IR
Outputs: [15:0] IR
Description: Include instruction register
Purpose: update and output MAR which contains instruction that is executing.

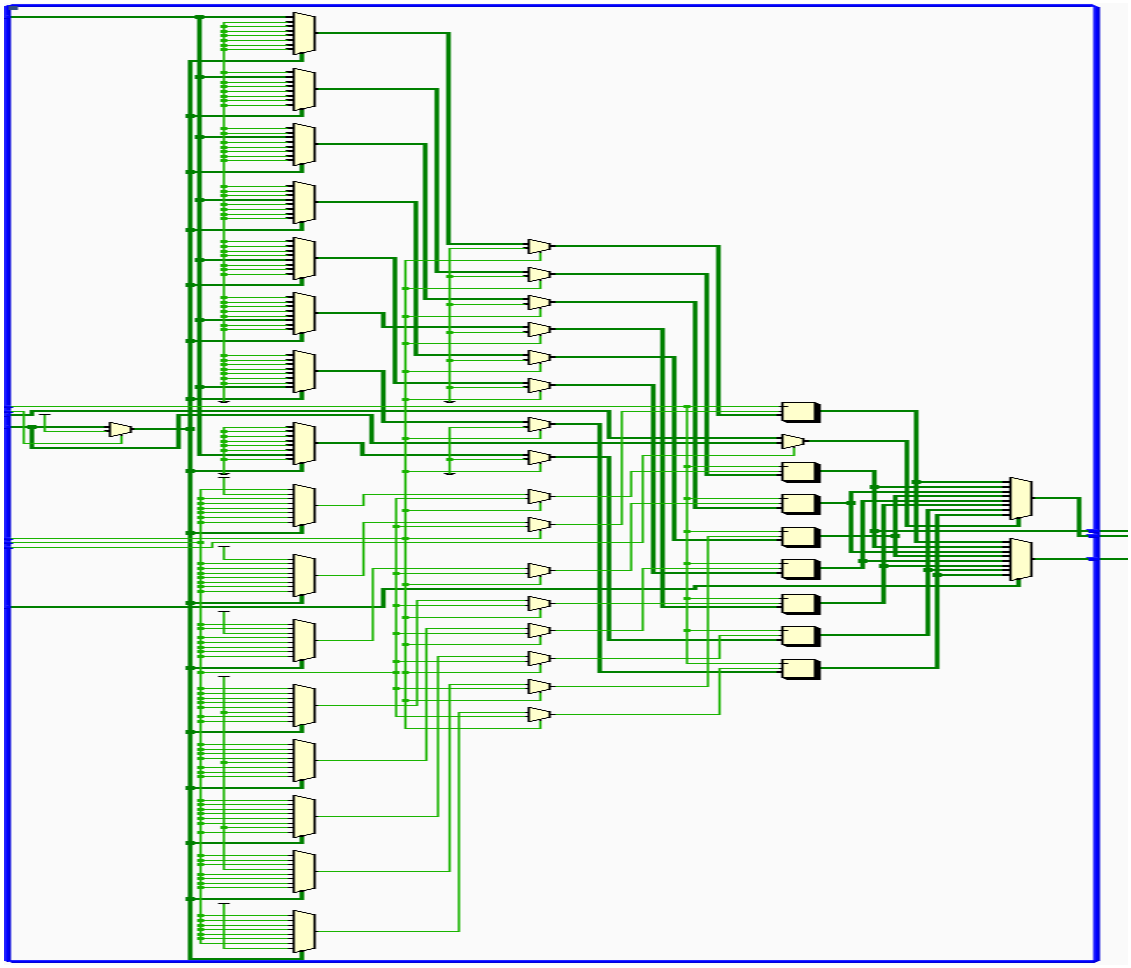
Module: BUS
Inputs: GateMDR, [15:0] MDR, GateALU, [15:0] ALU, GatePC, [15:0] PC, GateMARMUX, [15:0] MARMUX,
Outputs: [15:0] BUS
Description: it is a module which takes a MUX through four signals to output a BUS signal
Purpose: It provide a bus signal for the slc3 CPU.

Module: ALU
Inputs: [15:0] SR1_OUT, [15:0] SR2_OUT, [15:0] SEXT_4, SR2MUX, [1:0] ALUK,
Outputs: [15:0] ALU
Description: it is a module which operate a arithmetic computation and output the result
Purpose: It is provided mainly for ADD AND NOT and etc. operations.

Module: SEXT
Inputs: [15:0] IR
Outputs: [15:0] SEXT_10, [15:0] SEXT_8, [15:0] SEXT_5, [15:0] SEXT_4
Description: it outputs 16-bits signals which consist different bottom parts of IR, and the top bit are determined by the highest bit of IR you take account
Purpose: It is provided mainly for IMMs

Module: register_file
Inputs: Clk, Reset, [2:0] IR_11_9, [2:0] IR_8_6, LD_REG, [2:0] SR2, DRMUX, SR1MUX, [15:0] BUS,
Outputs: [15:0] SR1_OUT, SR2_OUT
Description: it consists of 8 16-bits register which can be read and write through MUX signal and load signal

Purpose: it is the register file of slc3.



(2.e block diagram of register file)

Module: addr_adderhe
Inputs: [15:0] SEXT_10, [15:0] SEXT_8, [15:0] SEXT_5, [15:0] PC, [15:0] SR1_OUT, [1:0]ADDR2MUX, ADDR1MUX
Outputs: [15:0] ADDR_OUT
Description: it consists an adder and a MUX to operate
Purpose: It compute the address specifically required by IR instruction

Module: nzp
Inputs: Clk, Reset, [15:0] BUS, LD_CC,
Outputs: [15:0] nzp
Description: it is a register which bottom three bits are representing negative, zero and positive
Purpose: this register is set after some instructions like ADD AND NOT operation and used for branch operation

Module: BEN
Inputs: Clk, Reset, [2:0] IR_11_9, [2:0] nzp_2_0, LD_BEN,

Outputs: BEN
Description: a 1-bit register operated by load and set by IR and nzp
Purpose: this module is to determine whether to enable branch

Module: LED
Inputs: Clk, Reset, [15:0] IR, LD_LED,
Outputs: LED
Description: a 16-bit register operated by load and set by IR
Purpose: show the IR data on the board

Module: MEM2IO
Inputs: Clk, Reset, [15:0] ADDR, OE, WE, [15:0] Switches, [15:0] Data from CPU, [15:0] Data from SRAM
Outputs: [15:0] Data to CPU, [15:0] Data to SRAM, [3:0] HEX0, [3:0] HEX1, [3:0] HEX2, [3:0] HEX3
Description: This module manages all I/O devices, e.g., the switches and 7-segment hex displays.
Purpose: This module is provided for enable output and input value from switches and to hex drivers

Module: ISDU
Inputs: Clk, Reset, Run, Continue, [3:0] Opcode, IR_5, IR_11, BEN
Outputs:LD_MAR, LD_MDR, LD_IR, LD_BEN, LD_CC, LD_REG, LD_PC, LD_LED, GatePC, GateMDR, GateALU, GateMARMUX, [1:0] PCMUX, DRMUX, SR1MUX， SR2MUX, ADDR1MUX, ADDR2MUX,ALUK, Mem_OE, Mem_WE

Description:

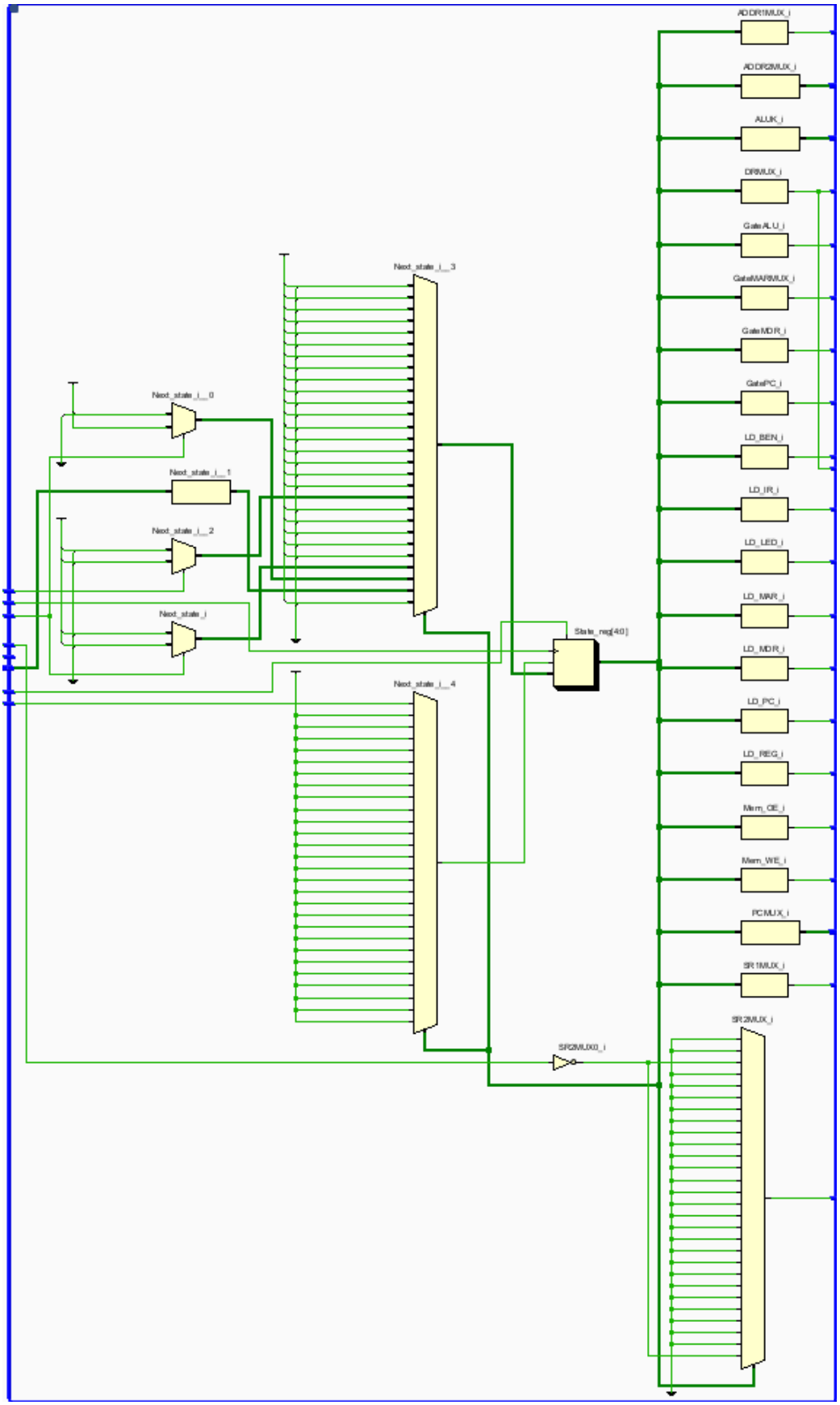
1. **Memory Fetching**: When the processor is running (Run signal active), it begins by fetching instructions from memory. It loads the memory address register (MAR) and reads data into the memory data register (MDR). The fetched instruction is then loaded into the instruction register (IR).
2. **Instruction Decoding:** The ISDU examines the fetched instruction (**IR**) to understand what operation needs to be performed. It checks the opcode to determine the type of instruction it's dealing with.
3. **Setting the Stage:** Based on the instruction, the ISDU sets the stage for different operations. For example, if it's an arithmetic operation, it prepares the arithmetic logic unit (**ALU**). If it's a memory operation, it manages data input and output from memory.
4. **Multiplexing Magic:** The ISDU uses multiplexers to select the right pieces of data and routes them to the correct destinations. Think of it as choosing different paths in a maze based on the instruction.

5. **Execution:** During certain states, like adding numbers or moving data between registers, the ISDU ensures the right components are active. It controls which registers to load, when to perform arithmetic operations, and when to read from or write to memory.

6. **Branching Decisions:** For instructions like conditional branches, the ISDU evaluates conditions. If a branch is needed, it calculates the new address for the next instruction.

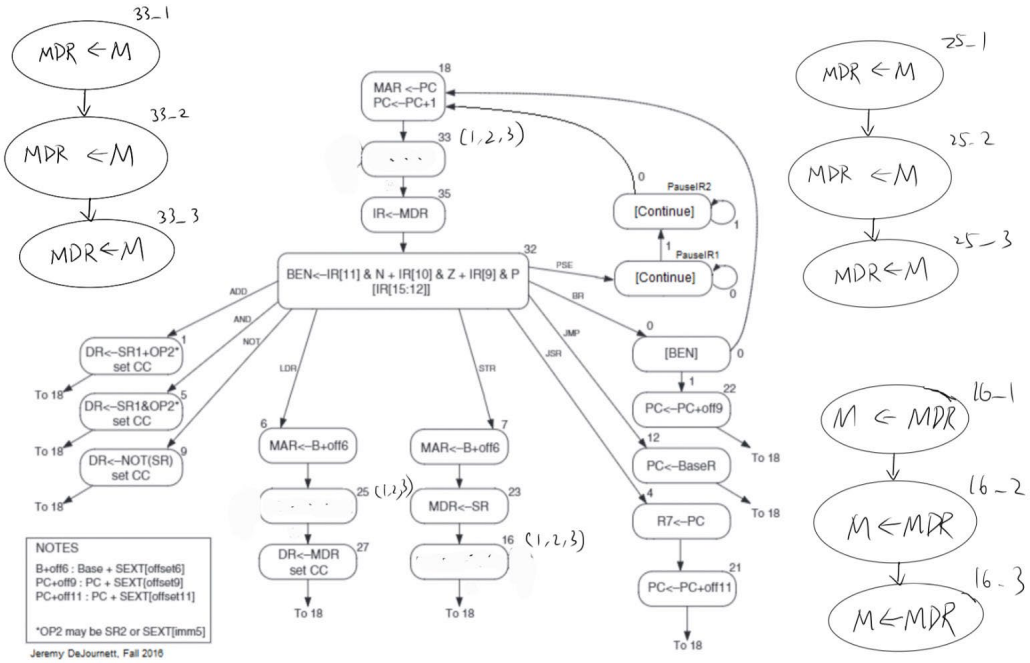
7. **Special Cases:** There are special states where the processor pauses (**PauseIR1**, **PauseIR2**). During these pauses, the contents of the instruction register are displayed for debugging.

Purpose: This module is used for outputting control signal for slc3 CPU



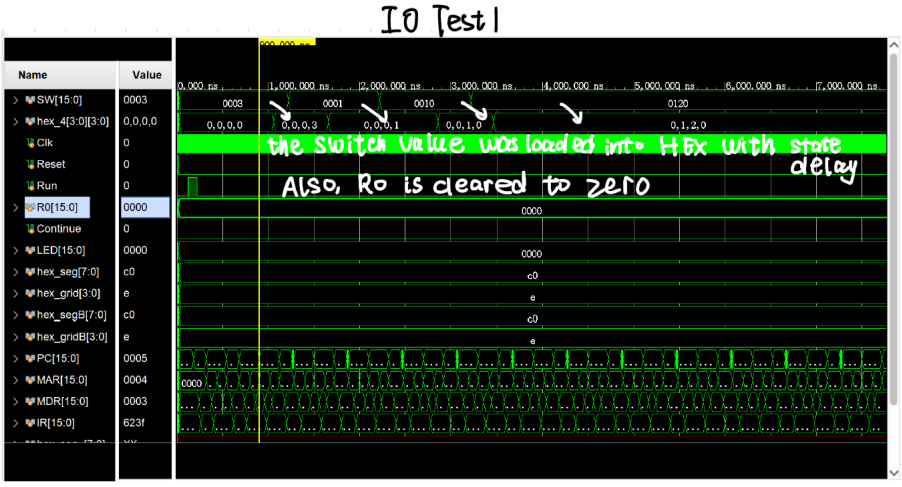
(2.e block diagram of ISDU)

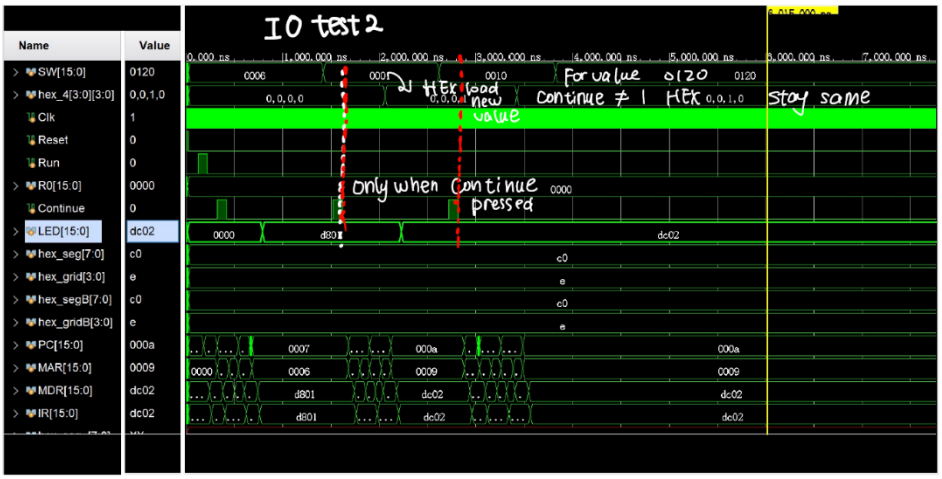
g.

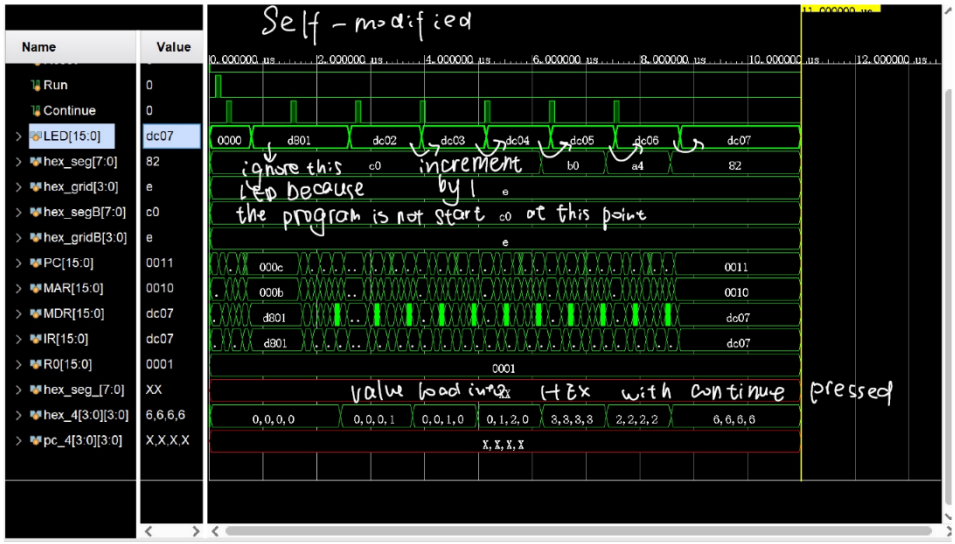


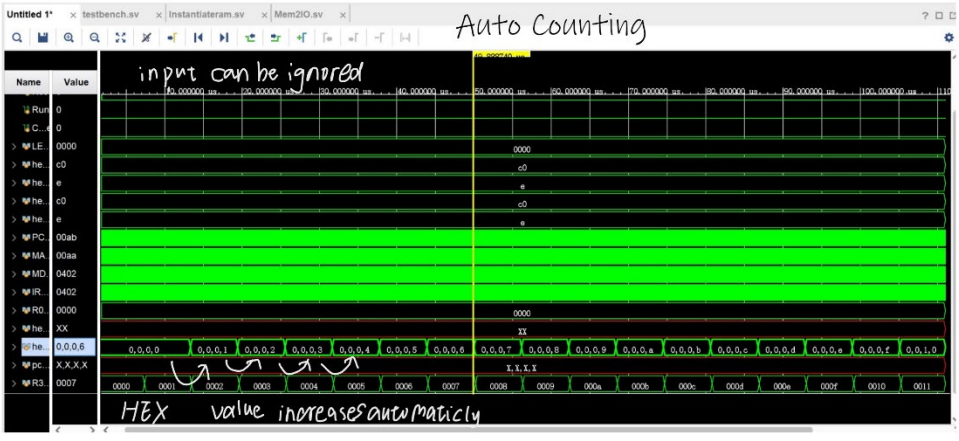
(2.g State diagram of ISDU)

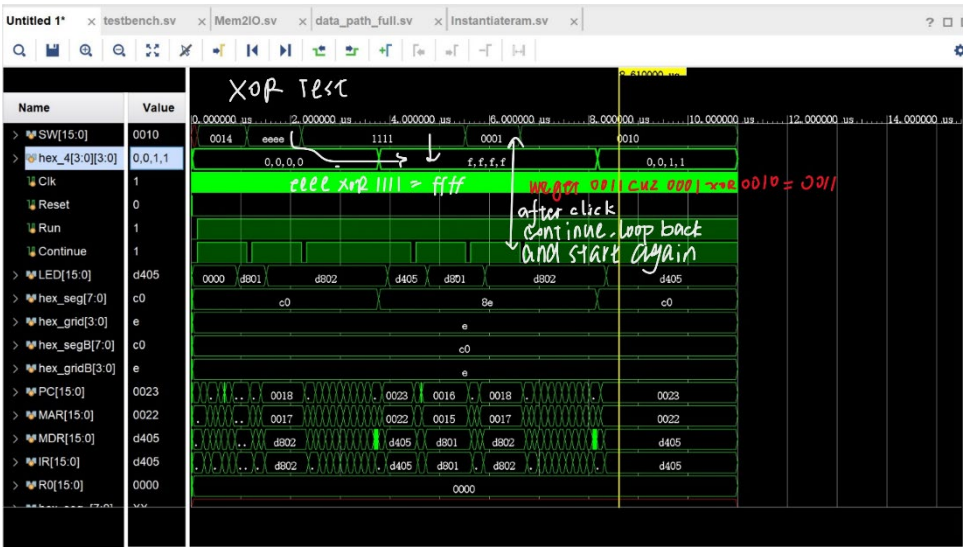
3.Simulations of SLC-3 Instructions

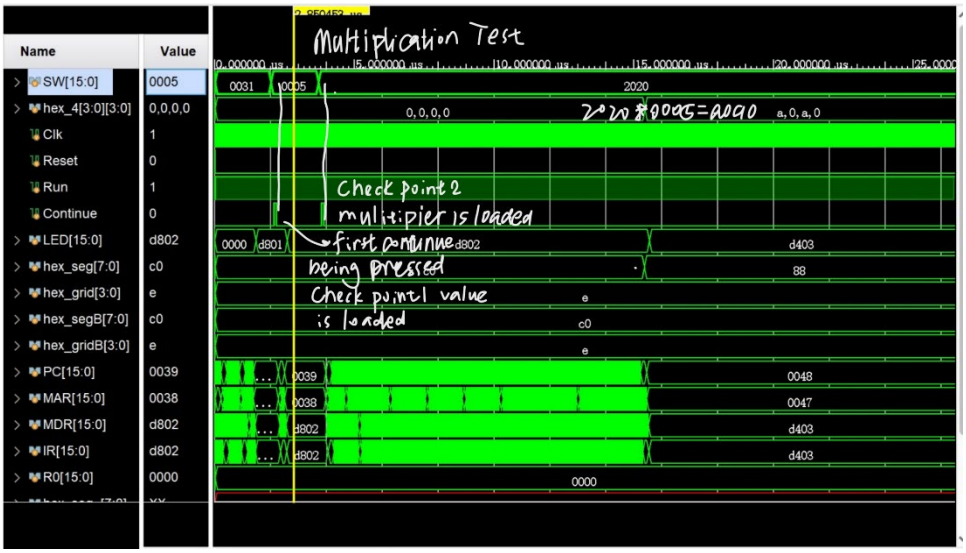


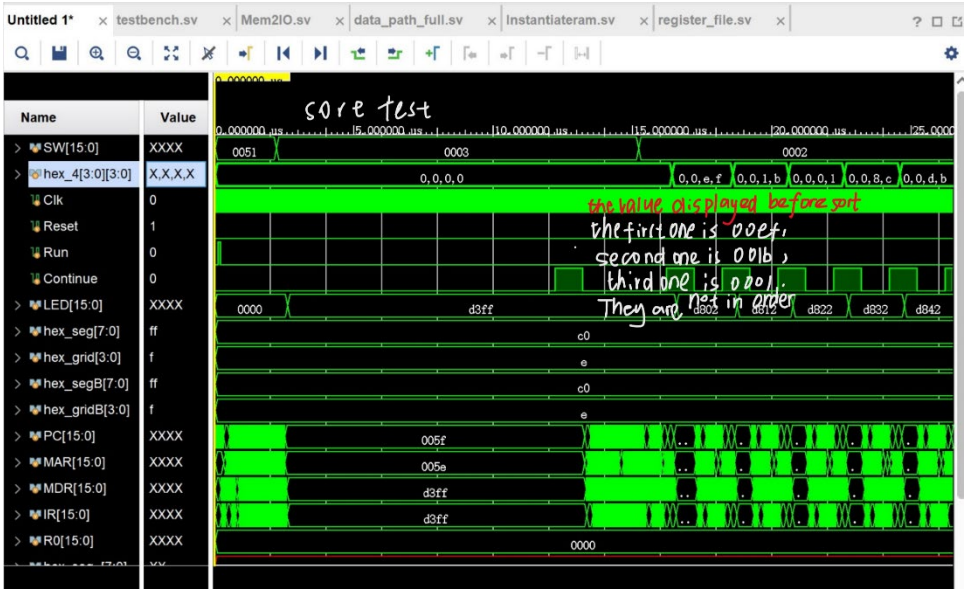

IO test2
Name
Value
SW[15:0]
hex_4[3:0][3:0]
Clk
Reset
Run
R0[15:0]
Continue
LED[15:0]
hex_seg[7:0]
hex_grid[3:0]
hex_segB[7:0]
hex_gridB[3:0]
PC[15:0]
MAR[15:0]
MDR[15:0]
IR[15:0]
0120
0,0,1,0
1
0
0
0000
0
dc02
c0
e
c0
e
000a
0009
dc02
dc02
HEX load new value
For value 0120 Continue ≠ 1 HEX 0,0,1,0 stay same
only when Continue pressed

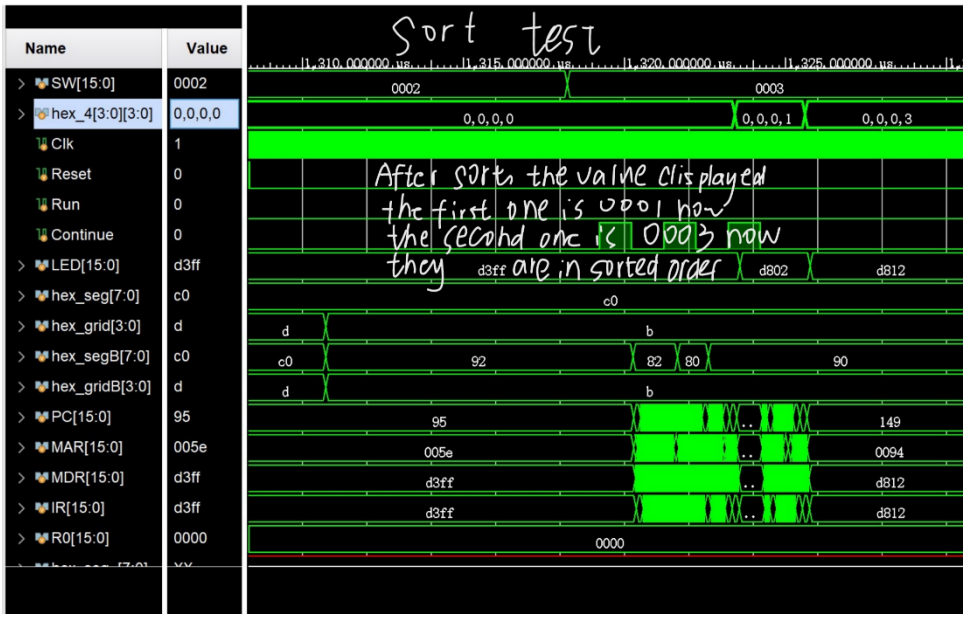

Self-modified
Name
Value
Run
Continue
LED[15:0]
hex_seg[7:0]
hex_grid[3:0]
hex_segB[7:0]
hex_gridB[3:0]
PC[15:0]
MAR[15:0]
MDR[15:0]
IR[15:0]
R0[15:0]
hex_seg_[7:0]
hex_4[3:0][3:0]
pc_4[3:0][3:0]
0
0
dc07
82
e
c0
e
0011
0010
dc07
dc07
0001
XX
6,6,6,6
X,X,X,X
ignore this step because the program is not start at this point
increment by 1
value load in HEX with continue pressed


Auto Counting
input can be ignored
HEX value increases automaticly


Untitled 1*
testbench.sv
Mem2IO.sv
data_path_full.sv
Instantiateram.sv
Name
Value
XOR Test
eeee xor 1111 = ffff
after click
continue, loop back
and start again
SW[15:0]
hex_4[3:0][3:0]
Clk
Reset
Run
Continue
LED[15:0]
hex_seg[7:0]
hex_grid[3:0]
hex_segB[7:0]
hex_gridB[3:0]
PC[15:0]
MAR[15:0]
MDR[15:0]
IR[15:0]
R0[15:0]


Name
Value
Multiplication Test
SW[15:0]
hex_4[3:0][3:0]
Clk
Reset
Run
Continue
LED[15:0]
hex_seg[7:0]
hex_grid[3:0]
hex_segB[7:0]
hex_gridB[3:0]
PC[15:0]
MAR[15:0]
MDR[15:0]
IR[15:0]
R0[15:0]
Check point 2
multipier is loaded
first continue
being pressed
Check point1 value
is loaded


Untitled 1*
testbench.sv
Mem2IO.sv
data_path_full.sv
Instantiateram.sv
register_file.sv
Name
Value
sore test
SW[15:0]
hex_4[3:0][3:0]
Clk
Reset
Run
Continue
LED[15:0]
hex_seg[7:0]
hex_grid[3:0]
hex_segB[7:0]
hex_gridB[3:0]
PC[15:0]
MAR[15:0]
MDR[15:0]
IR[15:0]
R0[15:0]
the value displayed before sort
the first one is 00ef,
second one is 001b,
third one is 0001.
They are not in order



4.Post-Lab Questions

LUT	473
DSP	0
Memory (BRAM)	0.50
Flip-Flop	295
Latches*	0
Frequency	897.666MHZ
Static Power	0.071W
Dynamic Power	0.009W
Total Power	0.08W

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

MEM2IO control the data flow between the switches, processor and memory. 0xFFFF is the key word to inditify whether to interact with I/O or memory. WE and OE are used to determine read and write instructions. If OE is high and WE is low and ADDR is FFFF, the value of switch is loaded into Data_to_CPU, else if adder is not FFFF, the value of SRAM is loaded into Data_to_CPU. When OE and WE(write_enable) is high, we write output as switch value if adder is FFFF else if adder is not FFFF, output is loaded from data_from_CPU. This module is important for MDR to load the right value. MEM2IO can display data from switches to HEX also display value coming out of processor to HEX.

•What is the difference between BR and JMP instructions?

The first differnece is the condition for PC to change, for BR, only when nzp（opcode）

and NZP（load with previous operation）are high can PC change it's value. The second difference is for JMP, PC can change to any 16 bits value based on BaseR, But for BR, the value for PC can only change with a 9 bits offset.

• 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?

Since memory access can take multiple cycles, read State continues to execute until the ready signal from the memory（R） is asserted, indicating that the memory access has completed. To compensate for lack of signal, we use three wait state which gives enough time to read from memory. The state is based on Clk signals which will not influence the synchronization. However, R signal is not based on Clk, which is not synchronization.

5. Conclusion

a.

All the functions is operating correctly. The hardest bug we encountered is that all the similation is correct but the inplementation on FPGA board is not functioning. It would always ended with with a strange PC value. Then we change the context of Instantiateram to find out which operation leads FPGA out of track. However, we find that no matter what operation we change, it will always gets wrong at a certain step, so there must be something wrong with the fetch operation. We change the wait state of state 33 then all the problem solved.

b.

This lab is excellent; it has helped me review LC3 and has greatly improved my proficiency in System Verilog coding.