ECE 385

Fall 2021

Experiment #6

Lab 6

Simple Computer SLC-3.2 in SystemVerilog

Guanshujie Fu 3190110666
Xiaomin Qiu 3190110717
Lab Section D231 18:00 – 21:00
TA: Yuchuan Zhu

1. Introduction

a) Summarize the basic functionality of the SLC-3 processor

The SLC-3 processor is a subset of the LC-3 ISA, a 16-bit processor with 16-bit Program Counter. The processor could load/store data, jump to specific instructions, play arithmetic and logic calculation, and can be paused. It takes instructions from RAM and executes them with data registers.

2. Written Description and Diagrams of SLC-3

a) Summary of Operation

i. Describe in words how the SLC-3 performs its functions. In particular, you should describe the Fetch-Decode-Execute cycle as well as the various instructions the processor can perform.

The SLC-3 has three stages, *FETCH, DECODE* and *EXCUTE*, it could perform *ADD, ADDi, AND, ANDi, NOT, BR, JMP, JSR, LDR, STR, PAUSE* commands based on the instructions stored in SRAM.

The first stage is FETCH. The SLC-3 will store the value in PC register to MAR, the memory address to read the instruction from. Then store M[MAR] into MDR, the instruction read form the memory, and then pull the MDR value into the IR register which provides instructions and data path with other necessary data. To finish Fetch, the PC register will be incremented.

The second stage is DECODE The SLC-3 will read the instructions stored in IR and send them to the Instruction Sequencer/Decoder which will decode it to determine the type of the instruction and generate the control signals to execute in proper order. IR [15:12] specifies the opcodes and IR[11:0] will be used to determine the registers address or offset number.

The third stage is EXCUTE. The SLC-3 will perform the operation based on the signals from the Instruction Sequencer/Decoder and write the result to the destination or memory. Operations include sending signal to ALU to perform logic calculations or loading/storing to the SRAM or JUMP/Branch to the specific locations.

Then the SLC-3 will fetch again.

b) Written Description of all .sv Modules

Modules	**Synchronizers.sv**
Inputs	*logic [15:0]* S, *logic* Clk, Reset, Run, Continue, *inout* wire [15:0] Data
Outputs	*logic* [11:0] LED, *logic* [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7, logic CE, UB, LB, OE, WE, logic [19:0] ADDR,
Description	These are built by D flip-flops. Data is stored until it is on the positive edge of clock and then data will go from D to Q.
Purpose	These are synchronizers required for bringing asynchronous signals into the FPGA and make them executing signal synchronously.

Modules	**SEXT.sv**
Inputs	*input logic* [15:0] IR
Outputs	*output logic* [15:0] SEXT0_10, *output logic* [15:0] SEXT0_8,

	output logic [15:0] SEXT0_5, *output logic* [15:0] SEXT0_4
Description	It uses the highest prior bit to decide whether to extend data by 1 and then check if the data is zero.
Purpose	Used to determine the offset.

Modules	**reg_file.sv**
Inputs	input logic Clk, Reset, input logic Load, input logic [2:0] DRMUX, SR1, SR2, input logic [15:0] Bus_in
Outputs	output logic [15:0] SR1_out, SR2_out
Description	This module contains registers and MUX which determine the register's value and output it.
Purpose	Act as R0 to R7, the temporary registers to hold temporary values which the LC3 program can use to perform operations such as addition, store, load, write etc.

Modules	**logic_nzp.sv**
Inputs	input logic Clk, Reset, input logic [15:0] data_in
Outputs	output logic [2:0] nzp
Description	This module is the Branch Enable unit for the instruction BR. IR[11:9] are the N, Z, P bits, if the status register match the condition codes, it will set BEN to the result.
Purpose	This module determines whether an branch operation should be excuted based on the status register's number. If it matches with the conditions given in the BR instruction, then the PC is changed.

Modules	**HexDriver.sv**
Inputs	input logic [3:0] In0
Outputs	output logic [6:0] Out0
Description	This is a hex driver that transfers a 4-bit signal to represent a hex display on the FPGA. The 4 bits in are needed to have designations for hex values 0 to F.
Purpose	It helps to present the output hex-value on the board.

Modules	**tristate.sv**
Inputs	input logic Clk, input logic tristate_output_enable, input logic [N-1:0] Data_write
Outputs	output logic [N-1:0] Data_read, (inout wire [N-1:0] Data)
Description	This module is a tristate buffer which allow data reads and writes to the SRAM.
Purpose	It is used to transmit data to and from MEM2IO and SRAM.

Modules	**test_memory.sv**
Inputs	input Clk,

	input Reset, inout [15:0] I_O, input [19:0] A, input CE, UB, LB, OE, WE
Description	This module is the top level of a simulating memory.
Purpose	This memory has similar behavior to the SRAM IC on the DE2 board. This file is for simulations only. In simulation, this memory is guaranteed to work at least as well as the actual memory (that is, the actual memory may require more careful treatment than this test memory). At synthesis, this will be synthesized into a blank module.

Modules	**slc3.sv**
Inputs	input logic [15:0] S, input logic Clk, Reset, Run, Continue
Outputs	output logic [11:0] LED, output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7, output logic CE, UB, LB, OE, WE, output logic [19:0] ADDR, inout wire [15:0] Data
Description	This is the layout of the entire SLC3 processor. It instantiates datapath, MEM2IO, the ISDU, and several other modules and serves as the central glue to hold them all together using the same variable names.
Purpose	It is designed to simulate LC3 and glues all the elements of the device together.

Modules	**register.sv**
Inputs	input logic Clk, Reset, Load input logic [15:0] data_in,
Outputs	output logic [15:0] data_out
Description	This module holds 16-bits registers and 12-bits LED register.
Purpose	The registers can be used to hold values for the LED and other module need registers.

Modules	**mux.sv**
Inputs	input logic SEL, input logic [k-1:0] A, B
Outputs	output logic [k-1:0] Out
Description	It includes all muxes, we default that input signals A, B, C, D... are selcted as 0, 1, 2, 3... It includes 2-to-1 MUX, 4-to-1 MUX, MUX_bus as mux work as the gate of bus line.
Purpose	In this module, for 2 or 4 inputs of the same bit length, the control signal would determine which one is needed and output it through the 2-to-1 MUX or 4-to1 MUX. The gate of the databus is also included in this module.

Modules	**memory_contents.sv**
Description	Store the memory content that may be simulated.
Purpose	The memory contents in the test memory.

Modules	**Mem2IO.sv**
Inputs	input logic Clk, Reset, input logic [19:0] ADDR, input logic CE, UB, LB, OE, WE, input logic [15:0] Switches, input logic [15:0] Data_from_CPU, Data_from_SRAM,
Outputs	output logic [15:0] Data_to_CPU, Data_to_SRAM output logic [3:0] HEX0, HEX1, HEX2, HEX3)
Description	This module is made of several muxes, it inputs the address from the MAR and can write and read memory based on the address and the WE and OE bits in the ISDU.
Purpose	This allows data transfer between the SLC3 processor datapath and the memory content in SRAM. It also allows data input from the switches as well as data output to the hexdrivers. It allows modification.

Modules	**lab6_toplevel.sv**
Inputs	input logic [15:0] S, input logic Clk, Reset, Run, Continue
Outputs	output logic [11:0] LED, output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7, output logic CE, UB, LB, OE, WE, output logic [19:0] ADDR, inout wire [15:0] Data
Description	This is the top level of lab6. It instantiates an instance of slc3 as well as test_memory.
Purpose	It combines SLC-3 processor and the available Test_memory.

Modules	**ISDU.sv**
Inputs	input logic Clk, Reset, Run, Continue, input logic[3:0] Opcode, input logic IR_5, input logic IR_11, input logic BEN,
Outputs	output logic LD_MAR, LD_MDR, LD_IR, LD_BEN, LD_CC, LD_REG, LD_PC, LD_LED, output logic GatePC, GateMDR, GateALU, GateMARMUX, output logic [1:0] PCMUX, output logic DRMUX, SR1MUX, SR2MUX, ADDR1MUX, output logic [1:0] ADDR2MUX, ALUK, output logic Mem_CE, Mem_UB, Mem_LB, Mem_OE, Mem_WE
Description	This module is the state machine FSM which control the signals on muxes, registers, gates, regfiles to manage the SLC-3 operation process based on instructions.

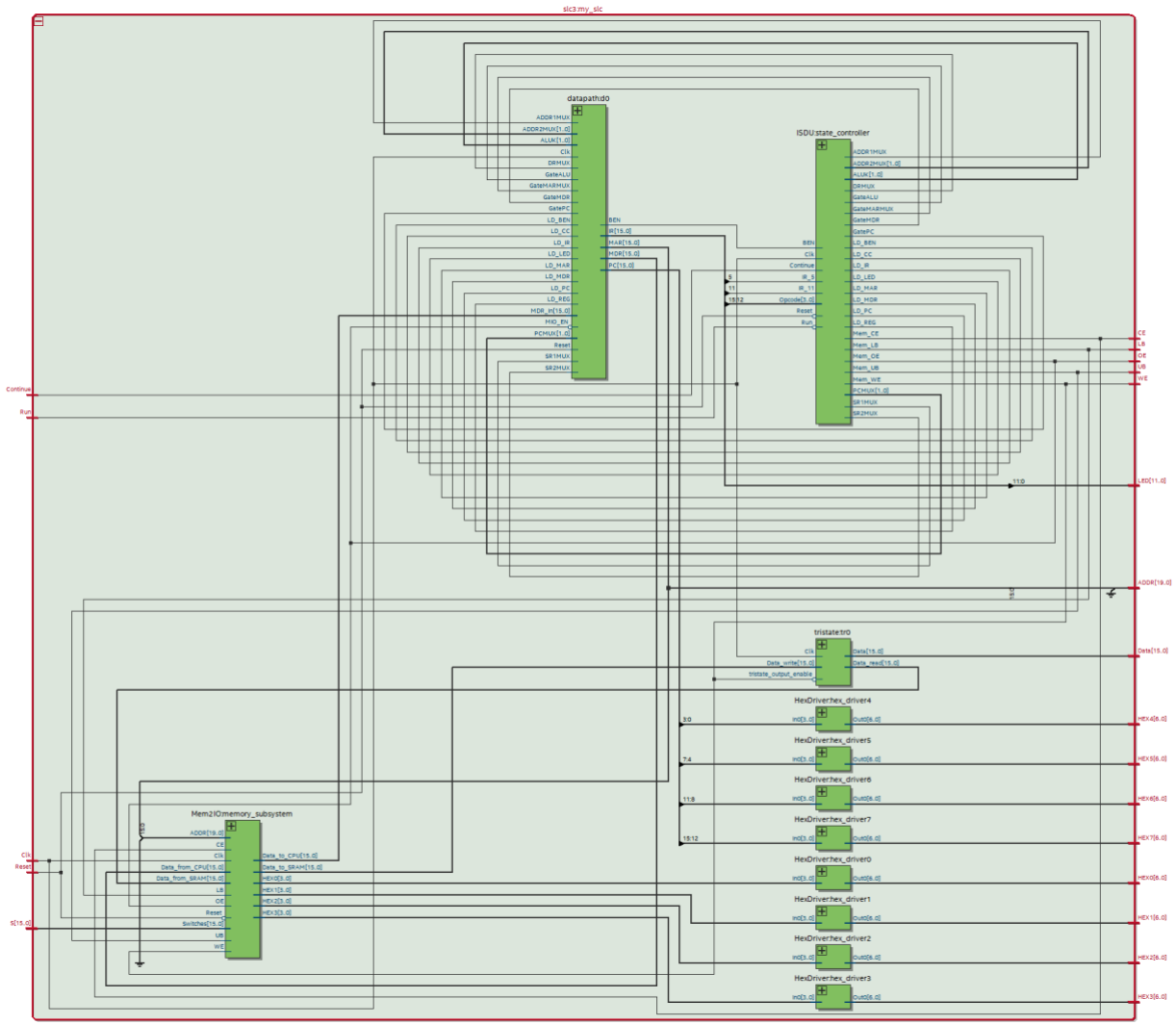
Purpose	It output control signals to contol datapath and other modules though the different states.

Modules	**datapath.sv**
Inputs	logic Clk, Reset, logic LD_MAR, LD_MDR, LD_IR, LD_BEN, LD_CC, LD_REG, LD_PC, LD_LED, logic GatePC, GateMDR, GateALU, GateMARMUX, logic [1:0] PCMUX, ADDR2MUX, ALUK, logic DRMUX, SR1MUX, SR2MUX, ADDR1MUX, logic MIO_EN, logic [15:0] MDR_In,
Outputs	logic BEN, logic [11:0] LED, logic [15:0] IR, logic [15:0] MAR, logic [15:0] MDR, logic [15:0] PC
Description	This module holds all registers, gates, MUXs, MUXs' select bits and the data bus.
Purpose	The module is used for data transfers and instruction decoding. It performs all operations associated with the signals from ISDU and outputs the results.

Modules	**ALU.sv**
Inputs	logic [1:0] ALUK logic [15:0] A, logic [15:0] B,
Outputs	logic [15:0] ALU_out
Description	This is an Arithmetic Logic Unitbuild with some gates and a 4-to-1 MUX.
Purpose	It allows basic arithmetic operations and logic operations and it could perform ADD, AND, NOT, or pass one of the input through it depending on the value in ALUK.

c) *Block Diagram of slc3.sv*

i. *This diagram should represent the placement of all your modules in the **slc3.sv**.*



d) *Description of the operation of the ISDU (Instruction Sequence Decoder Unit)*

i. *Named ISDU.sv, this is the control unit for the SLC-3. Describe in words how the ISDU controls the various components of the SLC-3 based on the current instruction.*

It controls the various components of the SLC-3 through various enable bits that are specified in each of the states. As the state diagram of the ISDU shows, the signals would enable or disable the modules based on instructions. Examples of these enable bits are for selecting MUX values, load or store, etc. For example, as the STR instruction is decoded at state 32, the ISDU will next change to stage 7 with signals that first enable RegFile to output baseR, then enable the ALU to ADD the baseR with offset6, then set LD.MAR the to load this value into MAR. After this, it will change to state 23 with signal MIO.EN and LD.MDR to load SR value to MDR. When completed, it will shift to state 16 with signals to store MDR to M[MAR]. When everything completed with complete signal, it will change to state 18 and FETCH stage again.

ADD and ADDi: IR [15:12] = 0001. It will enter state S_01, execute ADD, add two values and store the result in target register. So ALUK=00, SR2MUX=IR5, SR1MUX=1, LD_CC=1, LD_REG=1, GateALU=1. if IR5=1, SR2MUX will choose imm5 (IR[4:0]). if IR5=0, it will choose register (IR [2:0]). Then the state machine will return to s_18.

AND and ANDi: IR [15:12]=0101. It will turn to state S_05, execute AND, AND two values and store the result in target register. So ALUK=01, SR2MUX=IR5, SR1MUX=1, LD_CC=1, LD_REG=1, GateALU=1. if IR5=1, SR2MUX will choose imm5 (IR [4:0]). if IR5=0, it will choose register (IR [2:0]). Then the state machine will return to s_18.

NOT: IR [15:12]=1001, It will turn to state S_09, execute NOT, reverse every bit in SR1 and store it in target register. So LD_CC=1, LD_REG=1, GateALU=1, ALUK=10, SR1MUX=1. Then the state machine will return to s_18.

LDR: IR [15:12]=0110, It will turn to state S_06. First, store the address [Base + IR[5:0]] to MAR , LD_MAR=1, GateMARMUX=1, SR1MUX=1, ADDR1MUX=1, ADDR2MUX=01. Then it will enter the state s_25_1 the same as s_33_1 to read the content of MAR to MDR, so Mem_OE = 0. It will enter the state s_25_2 to wait for one clock cycle.Finally, enter state s_27 to store MDR to DR and set the status register, so GateMDR = 1, LD_REG = 1, LD_CC = 1; Then it returns back to s_18.

STR:IR [15:12]=0111, it will turn to state s_7 to store the address [Base +IR[5:0]] to MAR, so, ADDR2MUX = 01, ADDR1MUX = 1, SR1MUX = 1, GateMARMUX = 1, LD_MAR = 1. Then it will enter the state s_22 to store SR to MDR, so, ADDR1MUX = 0, ADDR2MUX = 10, PCMUX =10, LD_PC = 1. Finally it will enter state s_16_1 to write MDR to the address of MAR, so, Mem_WE = 0. Since the writing operation to the memory also needs more time, we add a state s_16_2 to do the same step. Then it returns back to s_18.

JSR: IR [15:12]=0100, it will enter s_4 to store PC value to R7, so, GatePC = 1, LD_REG = 1, DRMUX = 1. Then PC is changed to PC+IR[10:0]. Then, it returns to s_18. If the Opcode is 1100(JMP), so, GatePC = 1, LD_MAR = 1, LD_PC = 1. It enters state s_12 to store the content of Base register to PC, so, SR1MUX = 1, ADDR1MUX = 1, PCMUX = 10, LD_PC = 1. then it returns to s_18.

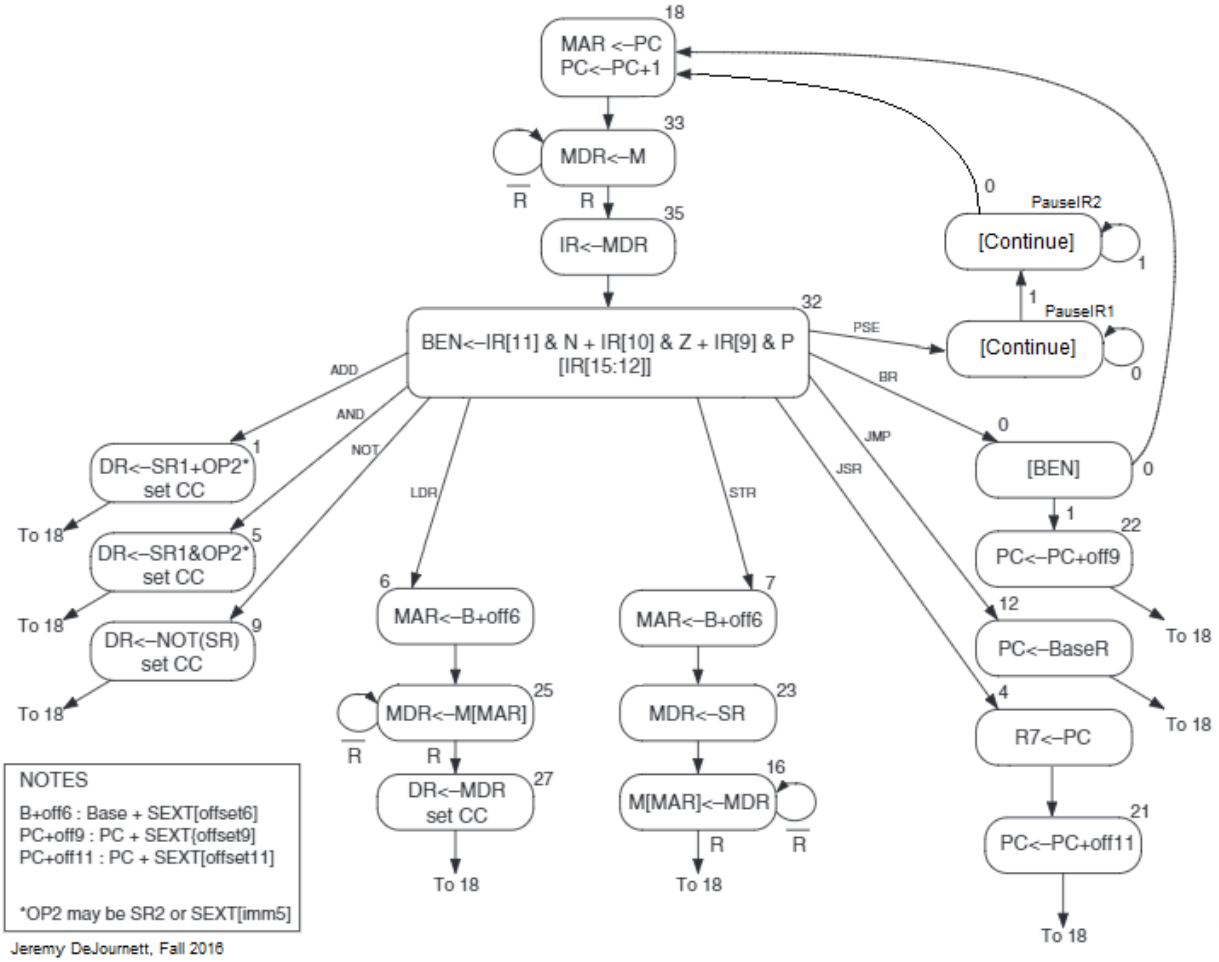
BR: IR [15:12]=0000, it will enter the intermediate state s_0. If the signal BEN is 0, it will return to s_18. Otherwise, it will enter state s_22 to change the PC value to PC+IR[10:0], so, ADDR2MUX = 10, PCMUX = 10, LD_PC = 1. and then it returns to s_18.

PAUSE: IR [15:12]=1101, it will enter the state PauseIR1 to stop the execution. If continue is 0, it will stay at the state, and IR[11:0] will be displayed on LEDs. If the continue is 1, it will enter PauseIR2. Now, if the continue button is still pressed, it will stay at PauseIR2 state until continue is 0 to return to state s_18.

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

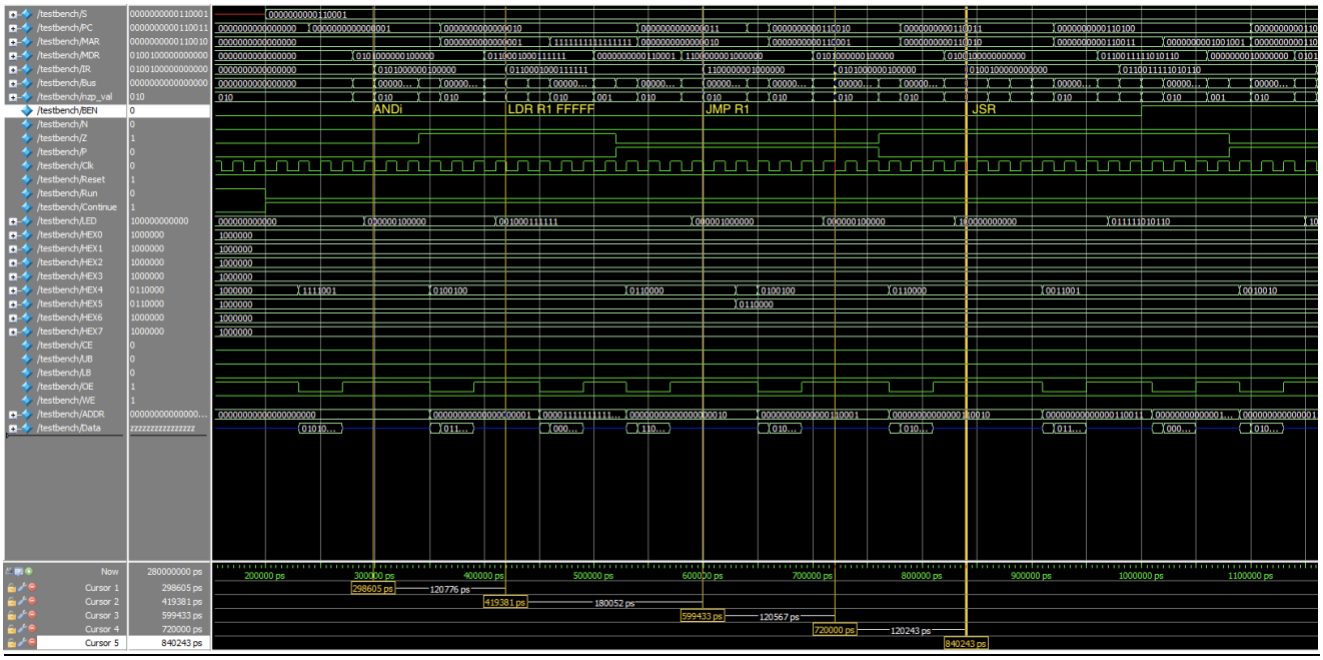
e) *State Diagram of ISDU*

i. *This should represent all states present in the ISDU and their transitions.*



3. Simulations of SLC-3 Instructions

a) *Simulate the completion of 3 instructions from the following groups: ADD/ADDi/AND/ANDi/NOT; BR/JMP/JSR; LDR/STR. For example, consecutively simulating ADD, BR and then LDR would be an acceptable simulation. You must annotate this diagram (for instance, label where instructions begin, where the answer is stored, etc.)*



As shown above, we perform ANDi, LDR, JMP and JSR. Firstly, ANDi is performed such that R0 is initialized to 0. LDR correctly load the value in Switch into R1, which can be seen in the value of Bus. JMP and JSR correctly load the value in R1 into PC and jump to the expected position.

4. Post-Lab Questions

a) *Fill out the Design Resources and Statistics table from Post-Lab question one*

LUT	588
DSP	0
Memory (BRAM)	0
Flip-Flop	274
Frequency	67.78MHz
Static Power	98.64mW
Dynamic Power	0.00mW
Total Power	172.29mW

b) *Answer the following three questions*

i. *What is the function of the MEM2IO.sv module?*

The Men2IO is used to manage the data from CPU and memory.it detect the content from

address 0xFFFF, switches as output instead of loading into memory. Also, it fetches the content out as HEX display, during which it stores the value in internal register and displays it instead of storing to SRAM.

ii. *What is the difference between the BR and JMP instructions?*

JMP changes PC unconditionally, while BR requires that the condition numbers matching the branch nzp bits stored in registers.

iii. *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 R signal is used to ensure that the memory read or write is ready. When memory read or write is finished, the memory will send R signal to indicate the operation has To compensate for the lack of the signal in our design, we use additional states for the state 33, thus we have 33_1 and 33_2 so that the two clock cycles could ensure the memory is ready.
This should have minimal impact on synchronization.

5. Conclusion

a) *Discuss functionality of your design. If parts of your design didn't work, discuss what could be done to fix it*

Our design works well and pass all test shown in testbench. It makes some mistakes in bubble sort test at first, but we fixed it soon.

b) *Was there anything ambiguous, incorrect, or unnecessarily difficult in the lab manual or given materials which can be improved for next semester? You can also specify what we did right, so it doesn't get changed.*

Nope.