ECE385

Fall 2021

Experiment 6

Simple Computer SLC-3.2 in Systemverilog

Ge Yuhao, Lou Haina
D231, Nov.8, 2021
TA: Chenhao Wang

1 Introduction

In this lab, we design a SLC3 processor with some slightly changes. This SLC3 composed of a on-board memory and a processor. It is able to do basic operations like ADD, AND, XOR, and it can also LOAD from memory and STORE to memory. Besides, it can also do BRANCH which enables the looping operations needed in a sort.

2 Written Description and Diagrams of SLC-3

2.1 Summary of Operation.

We accomplish 11 operations which our SLC-3 will perform based on the opcodes of IR, including ADD, ADDi, AND, ANDi, NOT, BR, JMP, JSR, LDR, STR and PAUSE. It is a 16-bit processor that performs three operations: fetch, decode and execute. SLC-3 will first fetch an instruction from the memory, decode it to determine the type of the instruction, execute the instruction and fetch again.

2.2 Describe in words how the SLC-3 performs its functions.

There are three basic states of SLC-3: Fetch, Decode and Execute. Fetch-Decode-Execute cycle begins from fetch, with PC load into MAR register and increment of PC. After that, CPU load data from SRAM to MDR read from address in MAR, and write it to IR register. Then it comes to decode state, bit number of IR register from 9-11 will be compared with 3 bit which is already stored in the register which might be used in branch operation and next state is determined by 12-15 four bit opcodes of IR, it will be input into ISDU to output appropriate signal of next state. Finally it comes to Execute part, different instructions are performed determined by different instruction code stored in memory and user input. For ADD, ADDi, AND, ANDi, NOT instruction, the SLC-3 do computation with SR1 and SR2 or offset, and store value in DR. For BR instruction, BEN is already determined in decode state and if BEN is 1, the offset will be added to PC and execute at that memory location from next fetch operation. For LDR and STORE instructions, memory in SRAM will be read or write. For JMP and JSP instructions, PC will jump to oriented memory location directly and JSP will store PC into R7 and increase PC. For Pause instruction, the LED will light up and wait user to input value with switch, and until user press continue, it will run back to state18. Every time execute has accomplished, state will go back to s_18.

2.3 Block Diagram of slc3.sv.

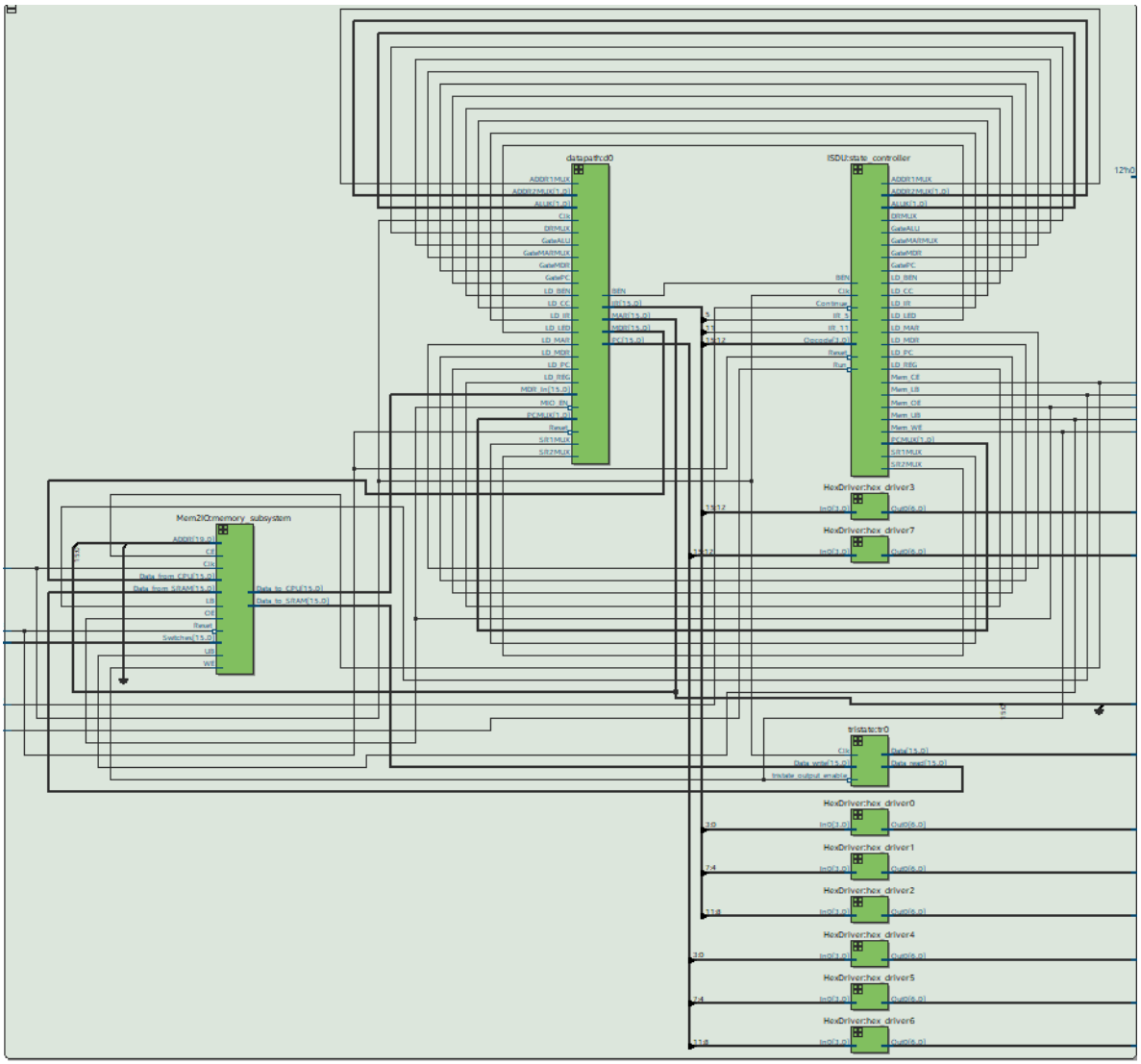

Figure 1: The Block Diagram for slc3

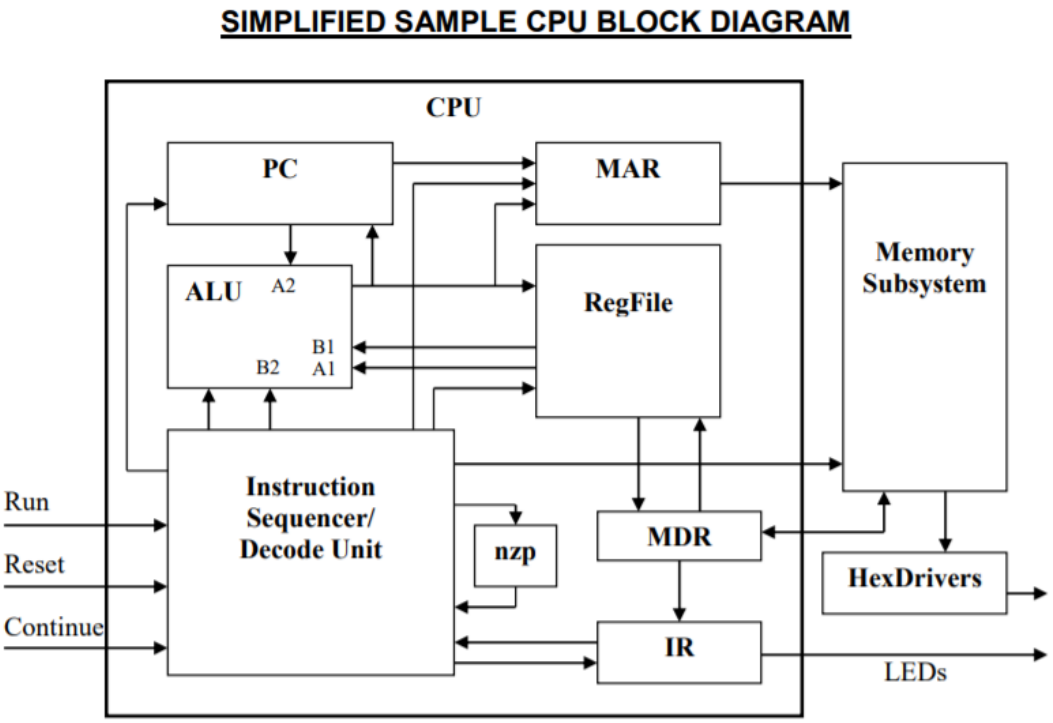

Figure 2: Simple Block Diagram for slc3

2.4 Written Description of all .sv modules

Module: ALU
Inputs: [15:0] SR1OUT, [15:0] SR2OUT, [15:0] IR40, [2:0]ALUK, SEL
Outputs: [15:0] ALUOUT
Description: Accepts SR1OUT and SR2OUT and do one of four operations on the two inputs. SEL will choose four operations, which are: AND, ADD, NOT, passSR1OUT.
Purpose: ALU works as the computation module which complete execute process for two register value and get result for register. It is the essential part for processor.

Module: tristate.sv
Inputs: Clk, tristate_output_enable, [15:0] Data_write
Outputs: [15:0] Data_read
inout wire: [15:0] Data
Description: This component works as a buffer which can either read data from SRAM to MEM2I0 or write Data to SRAM, if tristate_output_enable is not active, it will output high-impedance value.
Purpose: This module can connect both port from MEM2IO and SRAM, and accomplish both read and write operations bidirectionally through inout wire data bus.

Module: my_slc
Inputs: [15:0] S, Clk, Reset, Run, Continue
Outputs: [11:0] LED, [6:0] HEX0, [6:0] HEX1, [6:0] HEX2, [6:0] HEX3, [6:0] HEX4, [6:0] HEX5, [6:0] HEX6, [6:0] HEX7, CE, UB, LB, OE, WE, [19:0] ADDR
Description: This module contains all the SLC-3 components, and is exactly SLC-3. It contains datapath module which includes all the small components, MEM2IO module and statemachine. Hex

signal is used to display write data.
Purpose: It is used as SLC-3 processor which can process the instructions from memory.

Module: MEM2IO
Inputs: Clk, Reset, [19:0] ADDR, CE, UB, LB, 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, namely, the switches and 7-segment displays.
Purpose: This module is used for managing output and input value from switches and to hex drivers.

Module: memory_parser
Outputs: [15:0] mem_array[0:size-1]
Description: The memory contents in the test memory.
Purpose: used for simulation purpose only, take replace of SDRAM.

Module: lab6_toplevel
Inputs: [15:0] S, Clk, Reset, Run, Continue
Outputs: [11:0] LED, CE, UB, LB, OE, WE, [19:0] ADDR, [6:0] HEX0, [6:0] HEX1, [6:0] HEX2, [6:0] HEX3, [6:0] HEX4, [6:0] HEX5, [6:0] HEX6, [6:0] HEX7
Description: Top level file of the project.
Purpose: This file simulates a SLC-3 module and connect test_memory which is used to take replace of SRAM when simulates.

Module: HexDriver
Inputs: [3:0] In0
Outputs: [3:0] Out0
Description: Manage inputs to be displayed on hex.
Purpose: display data we are interested to the HEX display in FPGA board.

Module: MUX4_1
Inputs: [15:0] A, [15:0] B, [15:0] C, [15:0] D, [3:0]sel
Outputs: [15:0] Out
Description: Mux that controls which one of four gate value can go into the bus.
Purpose: this MUX is used to pick the correct gate value, which should be only opened only one gate at a time.

Module: MUX4_1_16b
Inputs: [15:0] A, [15:0] B, [15:0] C, [15:0] D, [1:0] S
Outputs: [15:0] Out
Description: Mux that takes in 4 different IR sign extended bits inputs.
Purpose: this MUX is used to pick the correct number of IR bits, which should be added with addr1mux.

Module: MUX2_1
Inputs: [15:0] A, [15:0] B, [15:0] C, [15:0] D, [1:0] S
Outputs: [15:0] Out
Description: Mux that takes in 4 different IR sign extended bits inputs.
Purpose: this MUX is used to pick the correct number of IR bits, which should be added with addr1mux.

Module: mux3_1
Inputs: [15:0] A, [15:0] B, [15:0]C, [1:0] S,
Outputs: [15:0] Out
Description: MUX used for PC so we can choose what goes into the PC.

Purpose: this MUX is used to pick the correct operation of PC updated, which should be loaded into PC register.

Module: mux2_1_3b
Inputs: [2:0] A, [2:0] B, [1:0] sel,
Outputs: [2:0] Out
Description: MUX used for SR1MUX so we can choose what goes into the SR1OUT register.
Purpose: this MUX is used to choose different IR instruction bits, when STR instruction MUX will choose IR[11:9] and else IR[8:6]

Module: NZP
Inputs: Clk, LD_CC, LD_BEN, BUS_VAL, [2:0]IR911
Outputs: BEN
Description: This is the logic of NZP loading from bus, and load into nzp register when LD_CC is high. Then it outputs AND value with instruction bits in IR to compute correct NZP values.
Purpose: For BR instruction, we need to check the condition bits to decide if we need to jump to the location or not, after set CC every time after an operation like ADD, AND and so on, we just compare the condition bit in BR instruction with last result to determine if it is negative, positive or zero.

Module: Register
Inputs: Clk, Reset, Load, [15:0] Din
Outputs: [15:0] Dout
Description: This is a 16 bit register.
Purpose: Used in MDR, MAR, and all the register which need to store 16 bits value.

Module: Register12
Inputs: Clk, Reset, Load, [11:0] Din
Outputs: [11:0] Dout
Description: This is a 12 bit register.
Purpose: Used in the register which need to store 12 bits value.

Module: Register3
Inputs: Clk, Reset, Load, [2:0] Din
Outputs: [2:0] Dout
Description: This is a simple 3 bit register.
Purpose: Used in the register which need to store 12 bits value.

Module: Register_file
Inputs: input LD_REG, Clk, Reset, [2:0] DRMUXOUT,[2:0] SR1MUXOUT, [2:0]SR2, [15:0] BUS_VAL,
Outputs: [15:0] SR2OUT,SR1OUT
Description: This is a register file which contaions register from R0 to R7, each register can be read or write with input address.
Purpose: Used like a on chip memory which can store data in R0 to R7 and read them to SR1OUT and SR2OUT

Module: MYDFF
Inputs: Clk, Reset, Load, Din
Outputs: Dout
Description: This is a simple Dflip-flop.
Purpose: Used in NZP module which load logic result of nzp into BEN fip-flop, storing value of Branch Enable bit

Module: SEXT #(N=6)
Inputs: [N-1:0]IR

Outputs: [16:0]OUT
Description: This module is used to extend sign bits of IR partial bits.
Purpose: Used when input of ADDR1MUX need to be added with offset which is defined in IR instruction

Module: datapath.sv
Inputs: LD_MAR, LD_MDR, LD_IR, LD_BEN, LD_CC, LD_REG, LD_PC, LD_LED, GatePC, GateMDR, GateALU, GateMARMUX, DRMUX, SR1MUX, SR2MUX, ADDR1MUX, MIO_EN, [1:0] PCMUX, [1:0] ADDR2MUX, [1:0] ALUK, [15:0] MDR_In, Clk, Reset
Outputs: BEN, [11:0] LED, [15:0] MAR, [15:0] MDR, [15:0] PC, [15:0] IR
Description: This module incorporates all of components in the SLC-3, and it accepts correct input signal from control unit and transfer correct data through MUX or load correct data.
Purpose: With datapath module, we successfully connect all components including control unit inside the SLC-3 and output data in LED, MAR, MDR, PC and IR, mux and register will get correct data at correct time, and has interface with MEM2IO for simulation

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, DRMUX, SR1MUX, SR2MUX, ADDR1MUX, Mem_CE, Mem_UB, Mem_LB, Mem_OE, Mem_WE, [1:0] PCMUX, [1:0] ADDR2MUX, [1:0] ALUK
Description: This is the state machine wich controls operation of the whole SLC-3, includes Fetch, Decode and Execute. It will begin at state halted and will start at S_18 if user press RUN button. Then it will begin Fetch operation. It will generate four fetch states automatically and come to decode state(S_32). On according to different instruction stored in IR. ISDU will continue to execute different state based on different opcode. Input IR_5 is used in the ADD and AND instruction to decide if CPU will add an offset or the value in register, it is controled by ISDU output signal MUX.IR_11 is used in JSP operation. Besides, with input BEN, if ISDU go to state s_00, in other words, the BR state, if BEN signal is 1, it will go to state S_22, otherwise it will go back to S_18. For Pause state, if the opcode is 1101, ISDU will come to pause after decode, and wait for user to input switch value, and only if continue is pressed by user, it will continue to run. For every state, if no input should be input, every state will output appropriate signal for SLC-3, like the selecting bits for MUX and Load signal, and state will change to the next clock after clock. If user press Reset button, it will go back to halted and clear all signal whenever the state are.
Purpose: Control unit is used to control the whole operation of SLC-3, which correct output at every state triggered by input. With ISDU, SLC-3 can generate different operations and store, load value from memory and execute instructions stored in the memory.

2.5 State Diagram of ISDU.

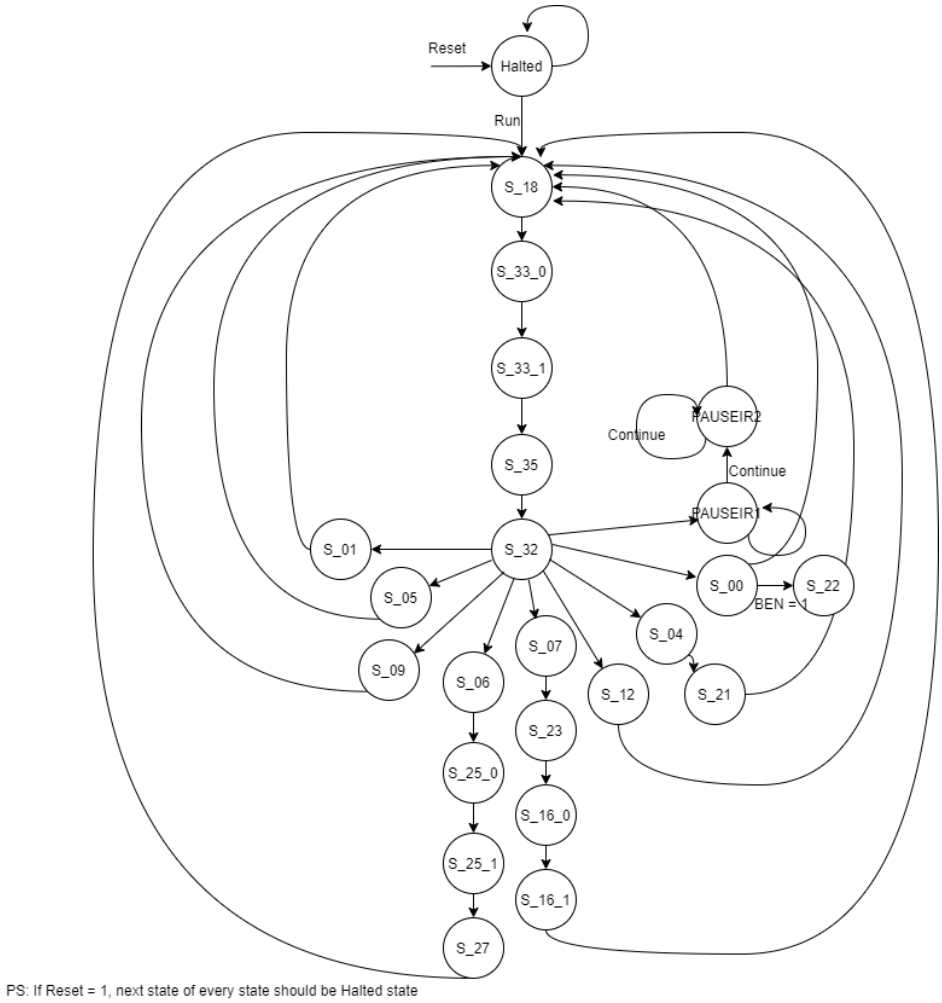

Figure 3: The State Diagram for ISDU

3 Simulations of Consecutive SLC-3 Instructions.

3.1 opLDR(R2, R0, inSW)

For this instruction, R2 should be loaded with switch value and we set switch value to be 3. We can see MAR first has instruction address x03 and read corresponding memory to MDR and IR and decode. Then MAR load with -1 to read switch value and load value to MDR, then load to R2 through bus.

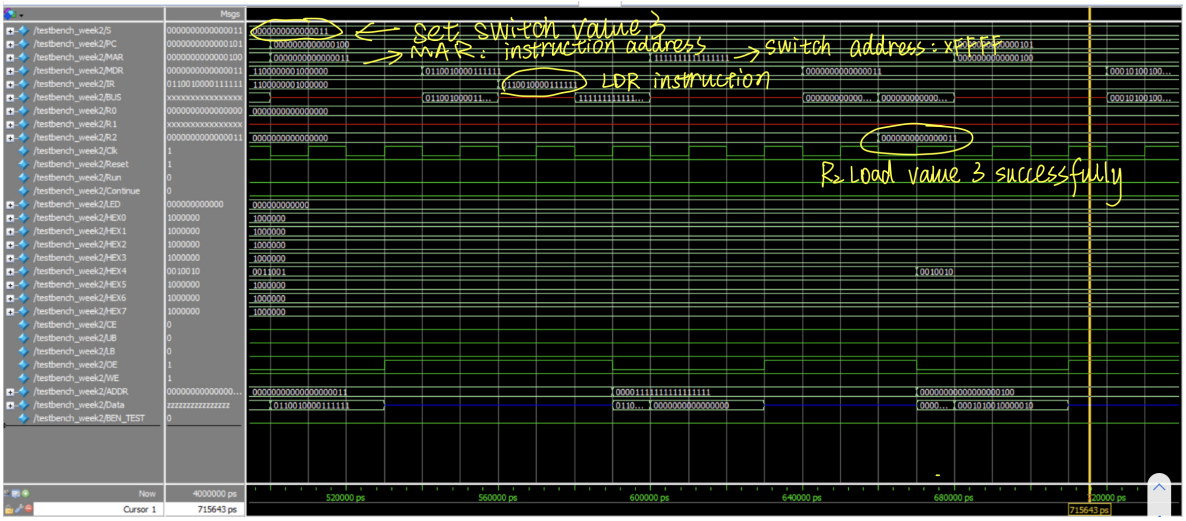

Figure 4: Simulation operation LDR for SLC-3

3.2 opADD(R2, R2, R2)

For this instruction, we add value in R2 and itself, then store the data back to R2.

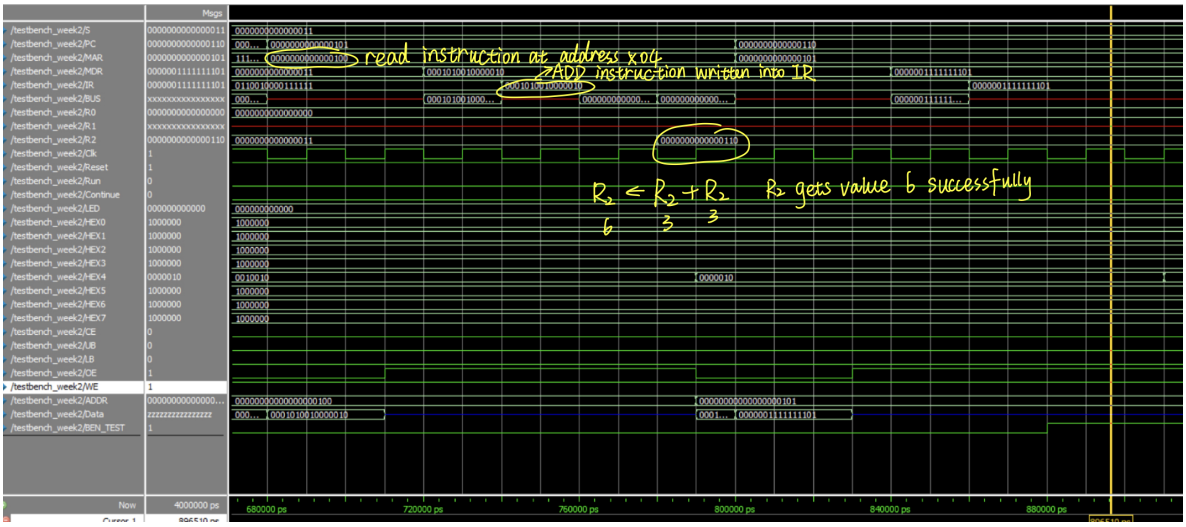

Figure 5: Simulation operation ADD for SLC-3

3.3 opBR(p, -3)

For this instruction, we set cc after we store value in R2, and if the value in R2 is positive, PC will decrease 3 and back to the first instruction address I set. After i generate add operation, the value in R2 should be 6 which is positive and PC will be set back to 3 as waveform shows.

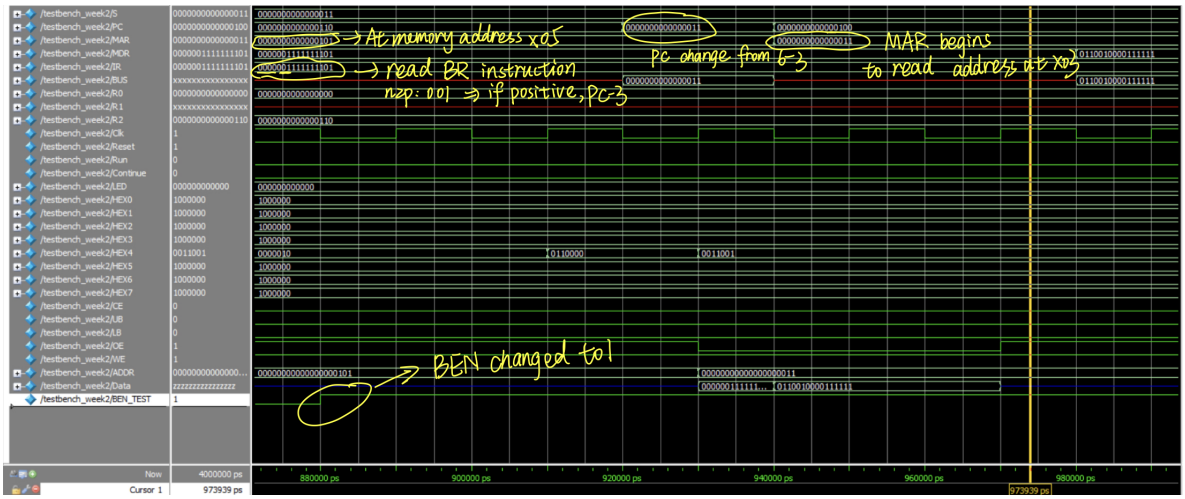

Figure 6: Simulation operation BR for SLC-3

4 Post-Lab Questions

4.1 Fill out the Design Resources and Statistics table from Post-Lab question one

LUT	735
DSP	0
Memory(BRAM)	0
Flip-Flop	283
Frequency	67.26MHz
Static Power	98.54mW
Dynamic Power	7.9mW
Total Power	178.36mW

Table 1: Design Resources and Statistics

4.2 Answer the following two questions

4.2.1 What is the function of the MEM2IO.sv module?

MEM2IO is the bridge between the SRAM module and the SLC3 processor which connect the FPGA board memory to the FPGA chip. It allows us to read data from the switch ot the memory and display the output data onto the HEX display.

4.2.2 What is the difference between the BR and JMP instructions?

JMP Jump to the location specified by the address in the *BaseR*

BR Takes the branch if any of the conditions codes match the condition stored in the status register; Otherwise continues execution. The branch location is determined by adding the sign-extended PCoffset9 to the PC.

Differences The JMP instruction changes PC without any condition while the BR needs condition codes to decide whether to jump. The JMP instruction accepts address data from register but the BR accepts address from the IR.

4.2.3 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 implication does this have for synchronization?

The R signal is used to show if the memory is ready. As we lack of the signal in our design, we add another state at the state 33, so we have two states 33-1 and 33-2. The first state 33-1 is used to add a clock circle to ensure that the memory is ready, and the second state 33-2 is used to give signals as the origin state 33 does. The state transition is no longer triggered by the signal R, and the design guarantees that there will be one clock before we go to the state 33-2, so there is no need for an additional synchronizer.

5 Conclusion

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

Our design functions perfectly. We first successfully run the FETCH operation in week one and MAR, MDR, IR works pretty well. Then in week two we realize operations like AND, XOR, and Multiplication.

5.2 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.

The guidance is very clear in this Lab. Although the Lab is kind of difficult and complicated, the guidance help a lot during the designing process. Especially the idea of the using MUX to replace the tri-state buffer design gives us inspirations how we can use replacement to achieve the same functionality. However, there are still something inconvenient in this Lab, during the debugging process, the waveform to be checked is too long and complicated, which spent us a lot of time to recognize if each operation is right, so we think an explicit given answer int the waveform would be very helpful.