ECE385

Fall 2020

Experiment 6

Simple Computer SLC-3.2 in SystemVerilog

Name: Zhou Qinren
Zhang Yichi
Lab Section: LA3
TA's Name: Yu Yuqi

Nov. 10th 2020

1 Introduction

In this lab, we design and build a simplified version of LC3, SLC3. The SLC3 processor is able to fetch, decode and execute the commands in the SRAM. More specifically speaking, it is able to do basic logical operations, load data from the memory and write data into the memory.

2 Written Description and Diagrams of SLC-3

2.1 Summary of Operation

This SLC3 controls all its functions by a finite state machine(FSM). The FSM contains three main states, fetch, decode and execute while each state can be more specifically defined as more states. In the fetch state, the processor stores the PC value, the address of the instruction, into the MAR, use this address to get the corresponding instruction and stores it to the MDR which will be passed to IR later. Then in the decode state, the processor will break the instructions down to several parts. The first four bits are opcodes which defines the funtion of this instruction. The rest defines the destination registers(DR), source registers(SR), offsets and some other values needed for the funtions. In the execute state, the specific function is performed. The feasible functions includes ADD, AND, NOT, LDR, STR, BR, JSR, JMP which will be introduced in the next section.

2.2 Written Description of all .sv Modules

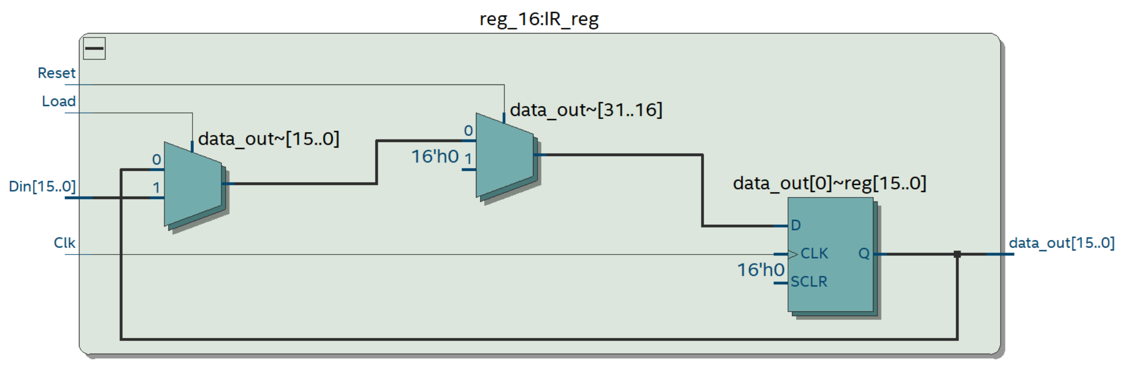

Figure 1: Reg_16.

Module: reg_16
Inputs: Clk, Reset, Load, [15:0] Din
Outputs: [15:0] data_Out
Description: This is a positive-edge triggered 16-bit register with synchronous reset, load.
Purpose: This module is a register unit which will be used to store the PC, MAR, MDR and IR values.

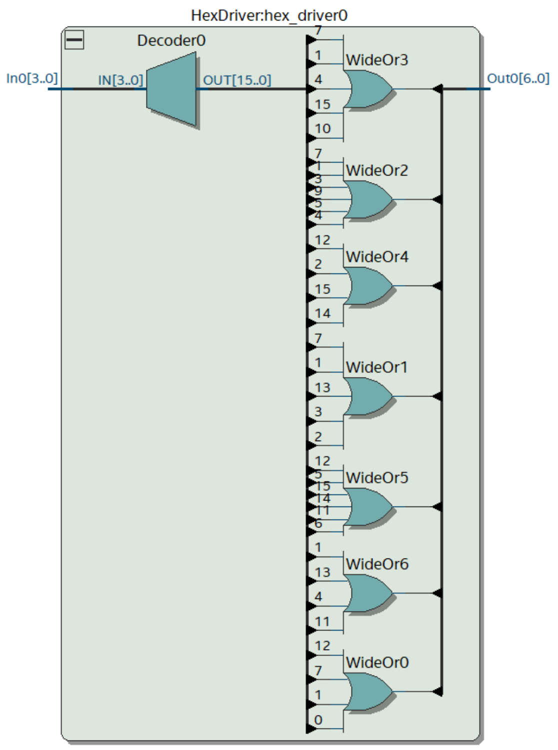

Figure 2: HexDriver.

Module: HexDriver
Inputs: [3:0] In0
Outputs: [6:0] Out0
Description: The HexDriver module takes in a 4-bit number and outputs its corresponding 7-bit number representing a hex number (one of "0123456789ABCDEF").
Purpose: HexDriver turns the result 4-bit number to its corresponding hex number for FPGA to show on the digital monitor.

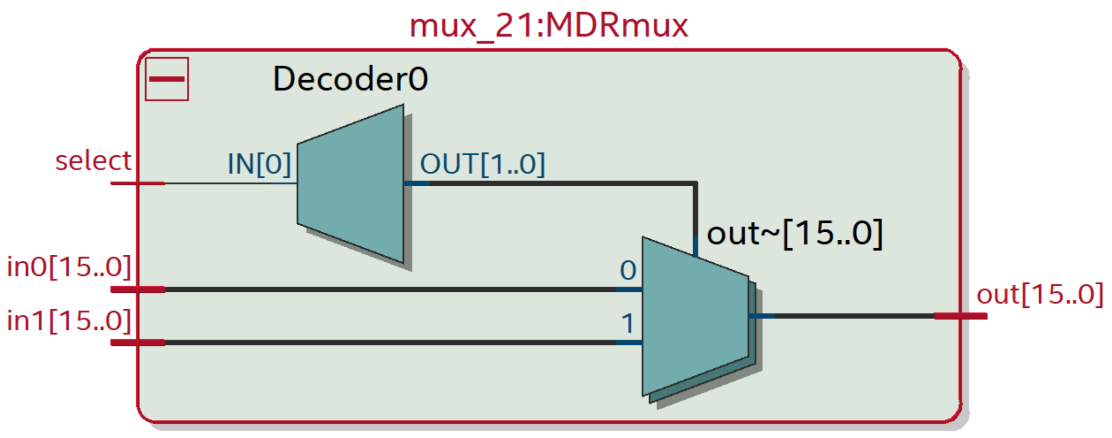

Figure 3: Mux_21.

Module: mux_21
Inputs: select, [N-1:0] in0, [N-1:0] in1
Outputs: [N-1:0] out
Description: This is a dynamic 2-to-1 mux that can take two N bits signals then select and output the one we want.
Purpose: This kind of mux is used for DRmux, SR1mux, SR2mux, ADDR1mux and MDRmux.

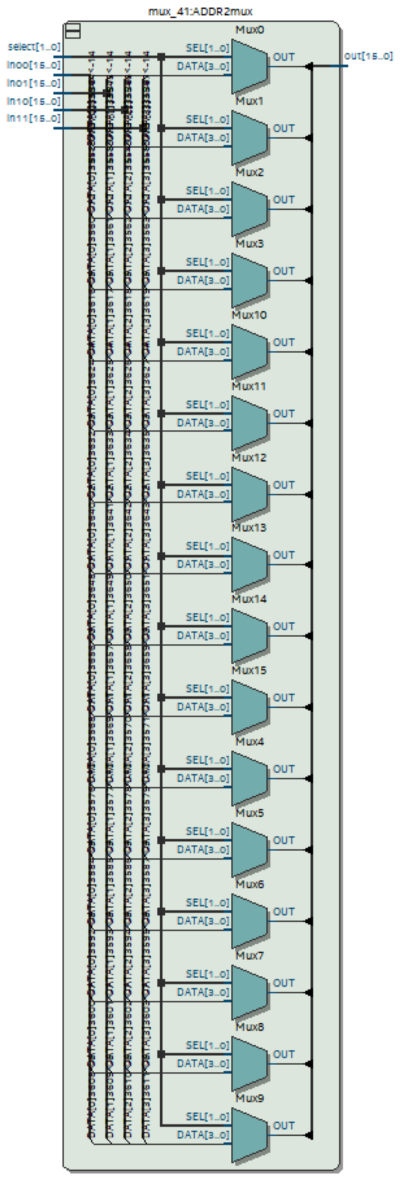

Figure 4: Mux_41.

Module: mux_41
Inputs: [1:0] select, [N-1:0] in00, [N-1:0] in01, [N-1:0] in10, [N-1:0] in11
Outputs: [N-1:0] out
Description: This is a dynamic 4-to-1 mux that can take four N bits signals then select and

output the one we want.
Purpose: This kind of mux is used for PCmux and ADDR2mux.

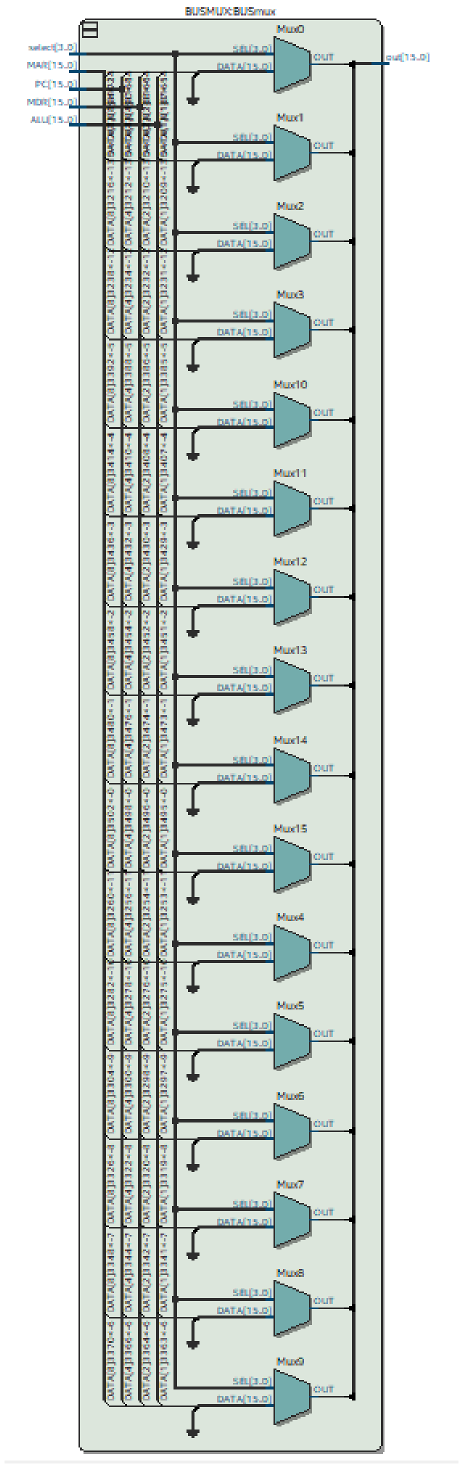

Figure 5: BUSMUX.

Module: BUSMUX

Inputs: [3:0] select, MAR, PC, MDR, ALU
Outputs: [N-1:0] out
Description: This is a 4-to-1 mux with one-hot encoding that takes four signals then select and output the one we want. One select signal corresponds to one specific input signal.
Purpose: This is a replacement of tristate buffers since the FPGA does not support internal tristate buffers. It selects which signal to be passed to the bus.

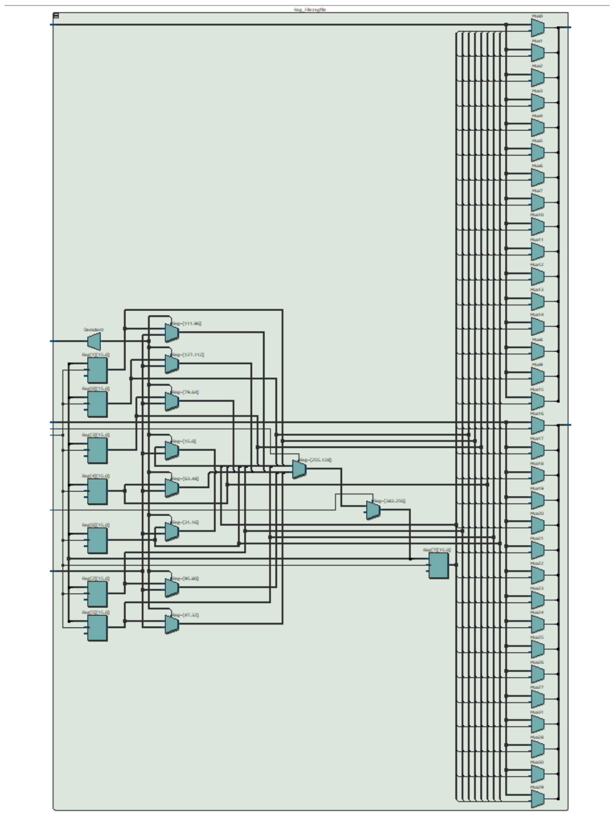

Figure 6: Reg_File.

Module: Reg_File
Inputs: [2:0] SR1, [2:0] SR2, [2:0] DR, [15:0] BUS, Clk, Reset, LD_REG
Outputs: [15:0] SR1OUT, [15:0] SR2OUT
Description: This is the combination of 8 16-bits registers that can take the values from the bus.
Purpose: This is a temporary small memory to store the value to be calculated, used or stored to the memory. It can speed up the process since reading from and writing into memory is relatively slow.

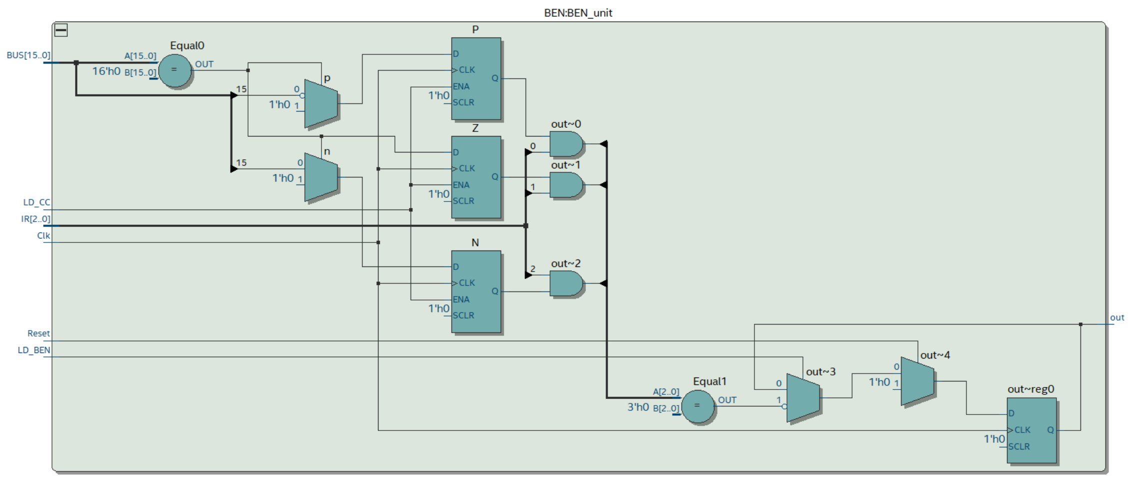

Figure 7: BEN.

Module: BEN
Inputs: [2:0] IR, [15:0] BUS, LD_BEN, LD_CC, Clk, Reset
Outputs: out
Description: This is a register that stores the sign of the values operated before (positive, negative or zero) and outputs whether the sign matches the nzp that is input from the branch instruction.
Purpose: This is specifically designed for the branch instruction to help check whether it meets the condition to branch.

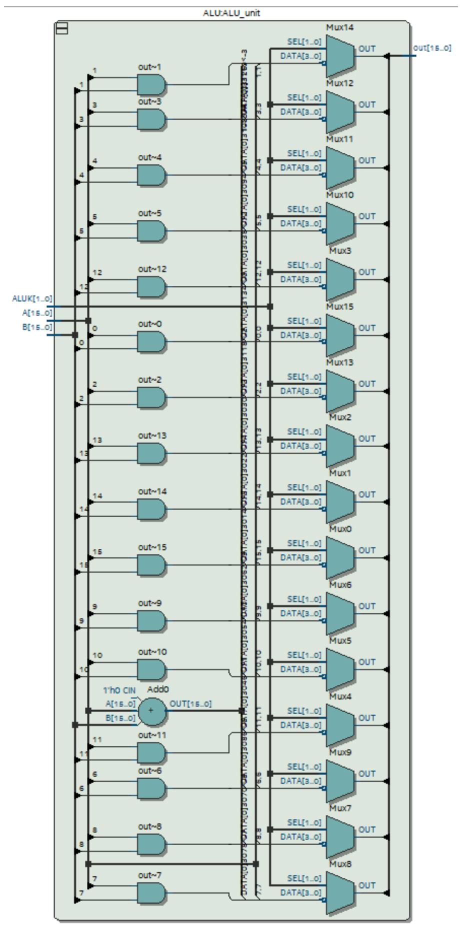

Figure 8: ALU.

Module: ALU
Inputs: [15:0] A, [15:0] B, [1:0] ALUK
Outputs: [15:0] out
Description: This unit takes two inputs, calculates with desired funtion and outputs the result.
Purpose: This is an Arithmetic Logic Unit that does the computation.

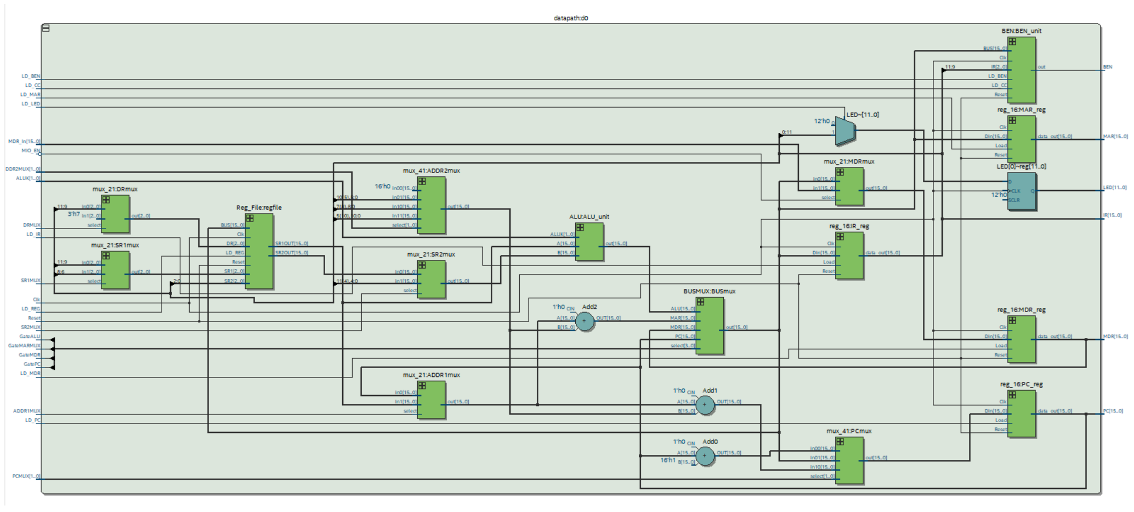

Figure 9: Datapath.

Module: datapath
Inputs: Clk, Reset, GatePC, GateMDR, GateALU, GateMARMUX, LD_MAR, LD_MDR, LD_IR, LD_BEN, LD_CC, LD_REG, LD_PC, LD_LED, [1:0] PCMUX, [1:0] ADDR2MUX, [1:0] ALUK, DRMUX, SR1MUX, SR2MUX, ADDR1MUX, MIO_EN, [15:0] MDR_In
Outputs: [15:0] MAR, [15:0] MDR, [15:0] IR, [15:0] PC, BEN, [11:0] LED
Description: This unit connects PC, MAR, MDR, BEN, ALU, control unit and Reg_File and pass data among them.
Purpose: This is the datapath of the SLC3 that transfers PC, MAR, MDR, IR, specifically, addresses, data to be computed and data to be stored.

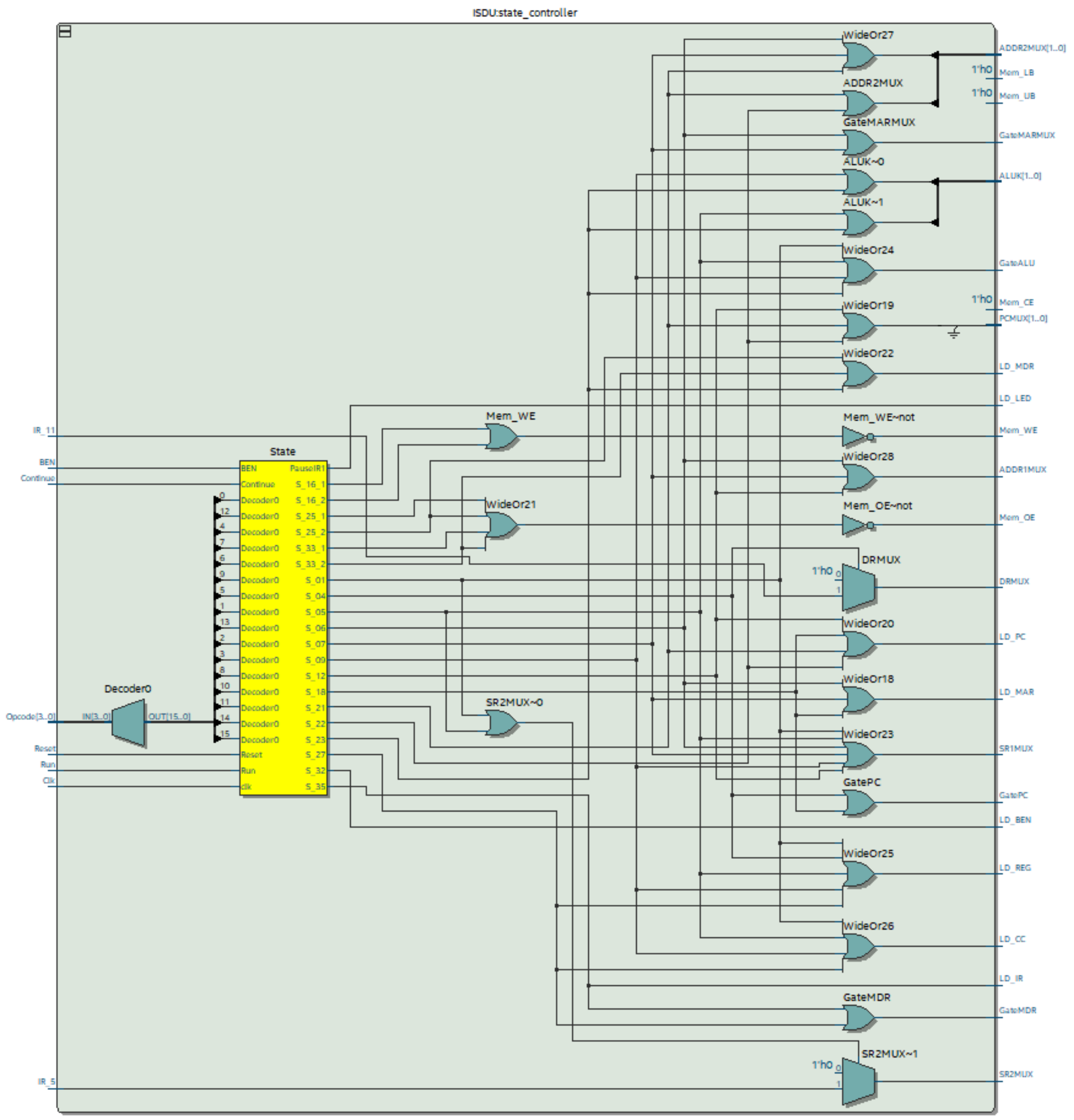

Figure 10: ISDU.

Module: ISDU
Inputs: Clk, Reset, Run, Continue, [3:0] Opcode,[N-1:0], IR_5, IR_11, BEN
Outputs: GatePC, GateMDR, GateALU, GateMARMUX, LD_MAR, LD_MDR, LD_IR, LD_BEN, LD_CC, LD_REG, LD_PC, LD_LED, [1:0] PCMUX, [1:0] ADDR2MUX, [1:0] ALUK, DRMUX, SR1MUX, SR2MUX, ADDR1MUX, Mem_CE, Mem_UB, Mem_LB, Mem_OE, Mem_WE
Description: This is an FSM with states that performs FETCH, DECODE, ADD, AND, NOT, LDR, STR, BR, JSR, JMP, PAUSE and HALT.
Purpose: This is an FSM with 24 states that controls what the SLC3 do (fetch, decode, pause or execute the functions).

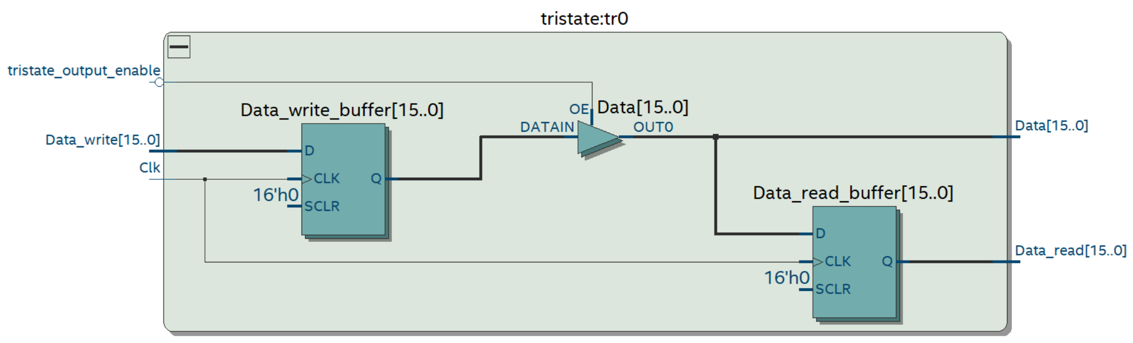

Figure 11: Tristate.

Module: tristate
Inputs: Clk, tristate_output_enable, [N-1:0] Data_write
Outputs: [N-1:0] Data_read
Inouts: [N-1:0] Data
Description: This unit exchanges data between SRAM and datapath.
Purpose: This is a tristate buffer that helps to protect the data.

Module: test_memory
Inputs: Clk, Reset, [15:0] I_O, [19:0] A, CE, UB, LB, OE, WE
Description: This is a simulated memory.
Purpose: This is only used for simulation. It performs similar to the SRAM on the DE2 board.

Module: sync
Inputs: Clk, d
Outputs: q
Description: Synchronize the inputs with no reset.
Purpose: Synchronizer for switches and buttons.

Module: sync_r0
Inputs: Clk, Reset, d
Outputs: q
Description: Synchronize the inputs with reset to 0.
Purpose: Synchronizer for D-flipflop.

Module: sync_r0
Inputs: Clk, Reset, d
Outputs: q
Description: Synchronize the inputs with reset to 0.
Purpose: Synchronizer for D-flipflop.

Module: sync_r1
Inputs: Clk, Reset, d
Outputs: q

Description: Synchronize the inputs with reset to 1.
Purpose: Synchronizer for D-flipflop.

Module: slc3
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
Inouts: [15:0] Data
Description: This module connects the memory, datapath, ISDU, and hexdrivers.
Purpose: This is the entire SLC3.

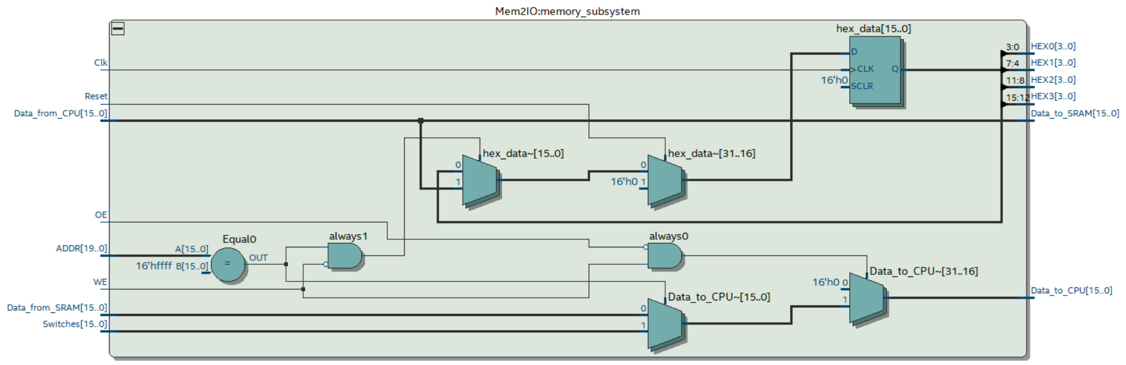

Figure 12: Mem2IO.

Module: Mem2IO
Inputs: CE, UB, LB, OE, WE, [19:0] ADDR, [15:0] Switches, Clk, Reset, [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 connects the memory, datapath, LEDs, switches and hexdrivers.
Purpose: This module controls the data between the CPU and the SRAM and the inputs (switches) and the outputs (LEDs).

2.3 Block Diagram of slc3.sv

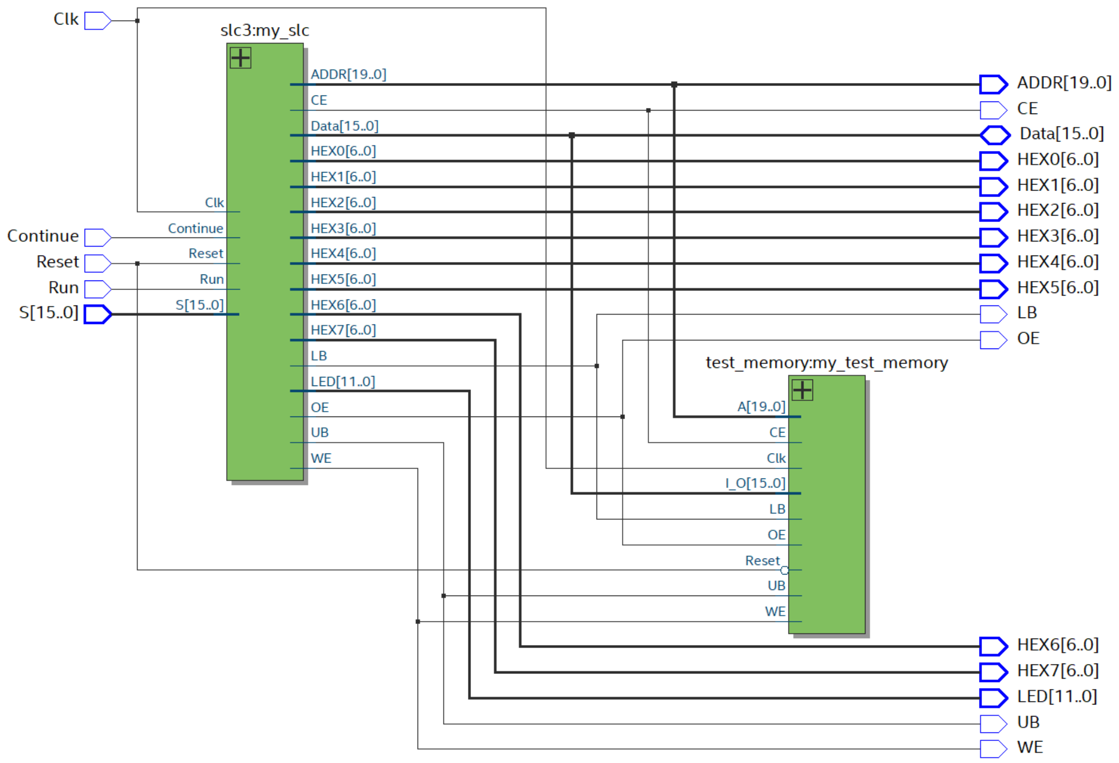

Figure 13: The block diagram of slc3.

2.4 Description of the operation of the ISDU (Instruction Sequence Decoder Unit)

ISDU is essentially a finite state machine. It generates several outputs, which can be divided into three groups: Load, Gate and Mux Select. Load is used to load the corresponding register. Gate is used to indicate which data should be connected to the bus. Mux select is used to select the input of the corresponding mux, such as the operands of ALU and the register address in register file. ISDU also deals with the memory via some memory control signal connected to MEM2IO.

After IR loads the newest instruction, the transition leads to DECODE state. Then state machine goes to different states based on the instruction. For example, if the instruction is ADD, the next state is state 1. ISDU will output the corresponding MUX Select signal to choose the destination register and source register. Moreover, ISDU will choose the operands of ALU. It will also output LD_REG, LD_CC and Gate_ALU.

2.5 State Diagram of ISDU

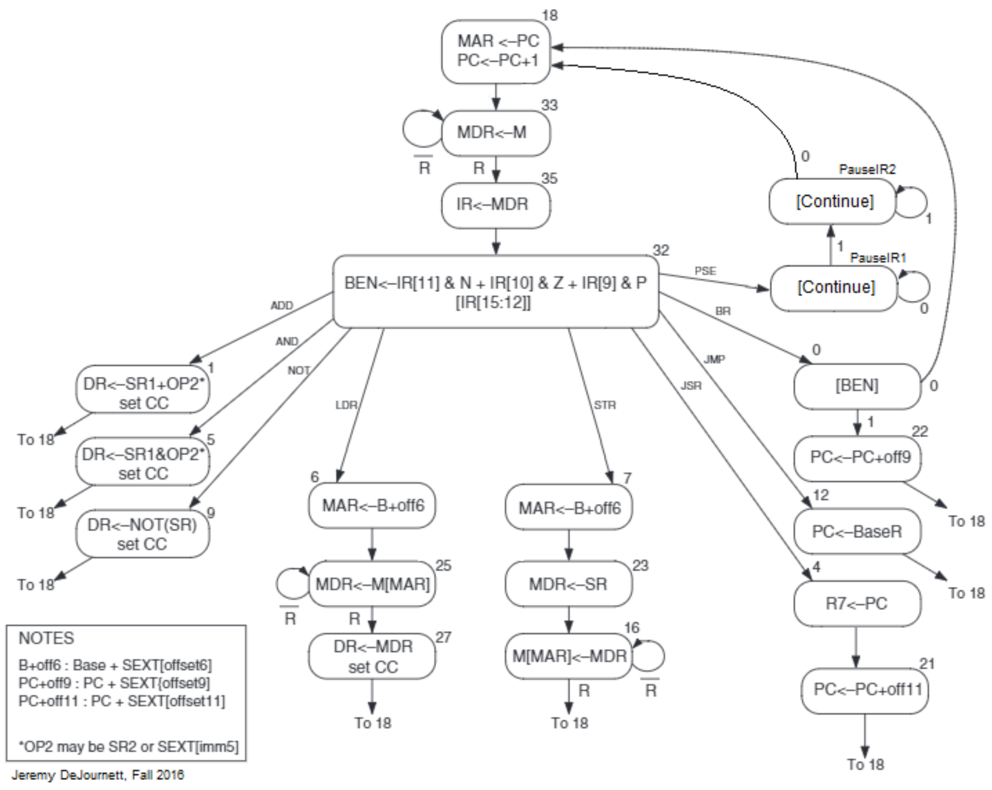

Figure 14: The state diagram of ISDU.

3 Simulations of SLC-3 Instructions

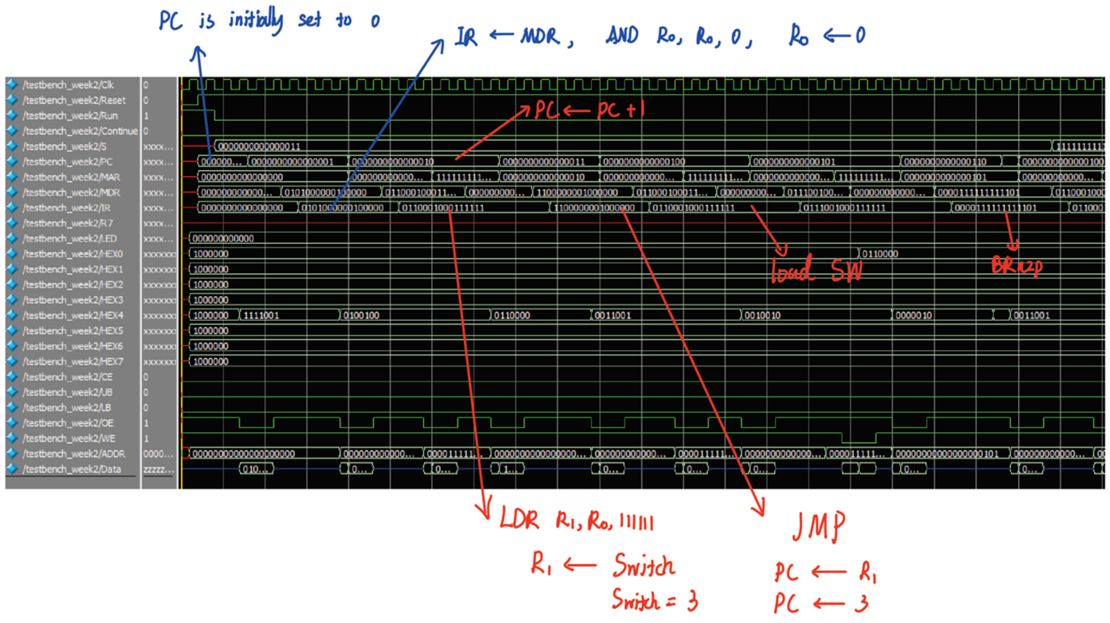

Figure 15: The simulations of SLC-3 instructions.

4 Post-Lab Questions

LUT	593
DSP	0
Memory	0
Flip-Flop	280
Frequency	65.11MHz
Static Power	98.66mW
Dynamic Power	8.54mW
Total Power	179.18mW

Table 1: Design statistics table for the multiplier.

Questions: What is the function of the MEM2IO.sv module?

Answers: MEM2IO is the interface between the memory and control unit. It can also deal with the switch input. The control unit generates some control signals to interact with the MEM2IO, which will handle the data transition between the memory.

Questions: What is the difference between the BR and JMP instructions?

BR is conditional jump; it will change PC if the condition code nzp is satisfied. JMP is unconditional jump; PC will certainly change when executing the instruction.

Questions: 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?

Answers: R signal is used to indicate that the memory is ready for read / write operation. In our design, we found that the memory will be ready in two cycles under any circumstances, thus, we assigned two states for each read or write operation. We don't have to use synchronizers, since we use states to control the memory.

5 Conclusion

5.1 Functionality

Our design is a small computer, which is capable of performing several instructions, such as ADD, AND, NOT, BR, JMP, JSR, LDR, STR, and PAUSE. The computer can read input from the switch and store data into memory. It can also perform some mathematical operation between two operands.

5.2 About Lab Manual

We are looking forward to have more materials about the FPGA, e.g. the chips on it, how it works, etc.