ECE385

Fall 2021

Experiment #5

Simple Computer SLC-3.2 in SystemVerlog

Haozhe Si

haozhes3

Section ABD

1. Introduction

In this lab, we designed and implemented a 16-bit 2's SLC-3 processor using SyetemVerilog on the FPGA board. This processor is based on Von Neumann architecture and realized a subset of LC-3 ISA and is able to perform 9 kinds of instruction: ADD, AND, NOT, BR, JMP, JSR, LDR, STR and PAUSE. The processor also supports the read/write of the instructions and data with the implementation of the off-chip SRAM, and has preloaded programs that can be loaded and executed easily.

2. Summary of Operation

To interact with the processor, the user can toggle the switches on the FPGA board to enter custom inputs; press both Execute and Continue buttons simultaneously to reset the state machine back to initial state; press Execute button to start the execution of instruction sequence; press Continue button to continue the program when after the PAUSE instruction. In practice, the user needs to pre-load the instructions to the SRAM so that the processor can execute the programs accordingly. The users can specify the program to execute by entering the address offset of the first instruction of the target program.

3. Functions Performing of SLC-3

To perform a function in SLC-3, the processor needs to fetch the instruction from the memory, decode the instruction to choose the corresponding instruction sequence, and execute the instruction. A cycle for Fetch-Decode-Execute needs to be performed for each instruction.

Fetch: The fetching process loads the instruction from the memory each time the previous execution is done. It requires loading Program Counter (PC) to Memory Address Register (MAR) and incrementing PC; loading the data to Memory Data Register (MDR) from the memory according to the address saved in MAR; and saving the value in MDR to the Instruction Register (IR). The value in IR represents the instruction we fetched from the memory.

Decode: Decode needs to analysis the instruction we fetched. To be specific, it will check IR[15:11], which is known as a four-bit 'opcode', and decide which sequence of operations the processor will perform later.

Execute: Execute will execute the instructions following the sequence of operations base on the values saved in source registers and immediate values as specified by the IR and save the result to the destination registers as instructed by the IR as well if necessary.

Available Instructions: The ISA we implemented for the SLC-3 is listed in the following table:

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

A more detailed description is provided by the lab manual:

ADD Adds the contents of SR1 and SR2, and stores the result to DR. Sets the status register.

ADDi Add Immediate. Adds the contents of SR to the sign-extended value imm5, and stores the result to DR. Sets the status register.

AND ANDs the contents of SR1 with SR2, and stores the result to DR. Sets the status register.

ANDi And Immediate. ANDs the contents of SR with the sign-extended value imm5, and stores the result to DR. Sets the status register.

NOT Negates SR and stores the result to DR. Sets the status register.

BR Branch. If any of the condition codes match the condition stored in the status register, takes the branch; otherwise, continues execution. (An unconditional jump can be specified by setting NZP to 111.) Branch location is determined by adding the sign-extended PCoffset9 to the PC.

JMP Jump. Copies memory address from BaseR to PC.

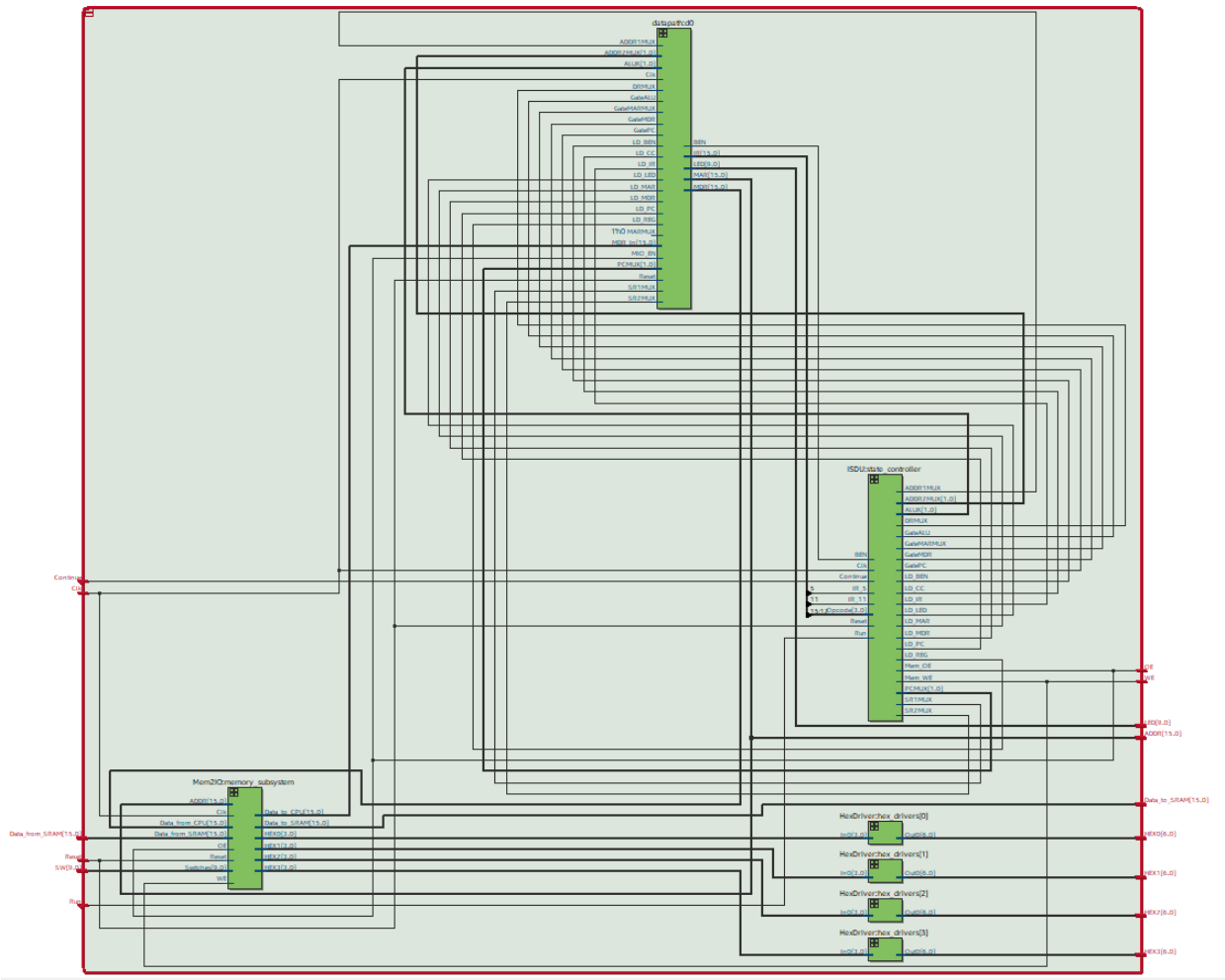
JSR Jump to Subroutine. Stores current PC to R(7), adds sign-extended PCoffset11 to PC.

LDR Load using Register offset addressing. Loads DR with memory contents pointed to by (BaseR + SEXT(offset6)). Sets the status register.

STR Store using Register offset addressing. Stores the contents of SR at the memory location pointed to by (BaseR + SEXT(offset6)).

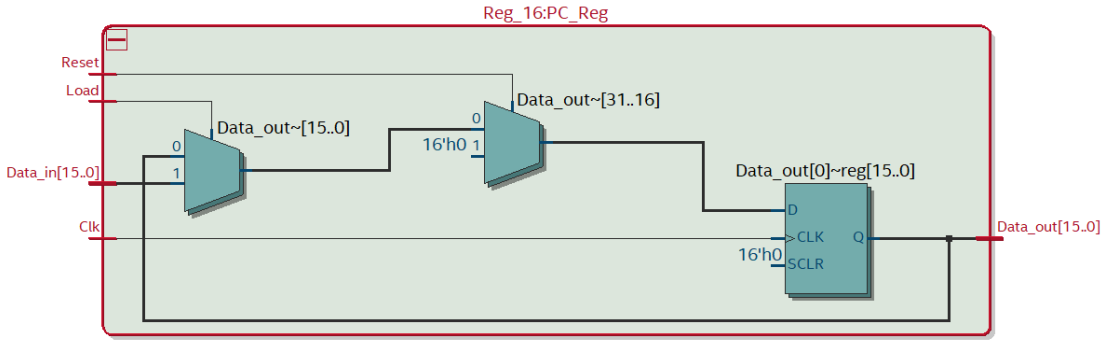
PAUSE Pauses execution until Continue is asserted by the user. Execution should only unpause if Continue is asserted during the current pause instruction; that is, when multiple pause instructions are encountered, only one should be “cleared” per press of Continue. While paused, ledVect12 is displayed on the board LEDs. See I/O Specification section for usage notes.

4. Block Diagram of SLC-3 Processor

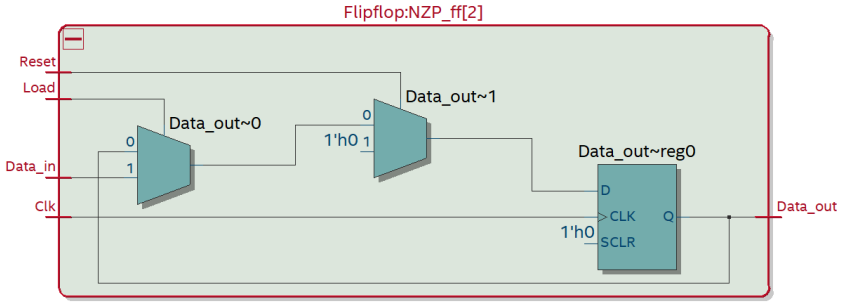


5. Module Description

Module: Reg_16
Inputs: [15:0] Data_in, Clk, Reset, Load
Outputs: [15:0] Data_out
Description: This is a positive-edge triggered 16-bit register module with asynchronous *Reset* and synchronous *Load*. When *Load* is high, data is loaded from *Data_in* into the register on the positive edge of *Clk*.
Purpose: This module is used to create the registers used in MAR, MDR, IR, PC and Register File modules.



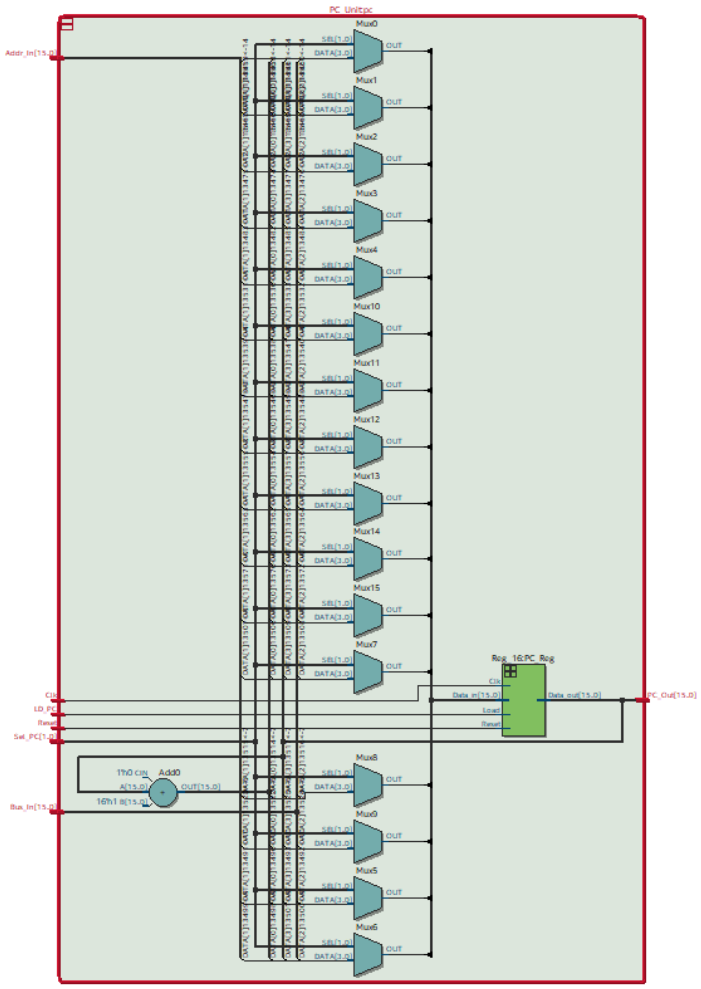
Module: Flipflop
Inputs: Data_in, Clk, Reset, Load
Outputs: Data_out
Description: This is a positive-edge triggered flip-flop module with asynchronous *Reset* and synchronous *Load*. When *Load* is high, data is loaded from *Data_in* into the register on the positive edge of *Clk*.
Purpose: This module is used to create the flipflop used in BEN module to save the NZP condition and Branch Enable signal



Module: PC_Unit
Inputs: [1:0] Sel_PC, [15:0] Bus_In, [15:0] Addr_In, Clk, Reset, Load
Outputs: [15:0] PC_Out
Description: The module consists of a 16-bit register and a set of 3-to-1 MUXes. The MUXes will select the input to the register according to the *Sel_PC* signal. The input can be: self-increment of the register value, the data from the address decoder *Addr_In*, or the data on the Bus *Bus_In*. The output of the register will be

PC_Out.

Purpose: This module will take part in the instructions requiring PC and decides the value for PC.



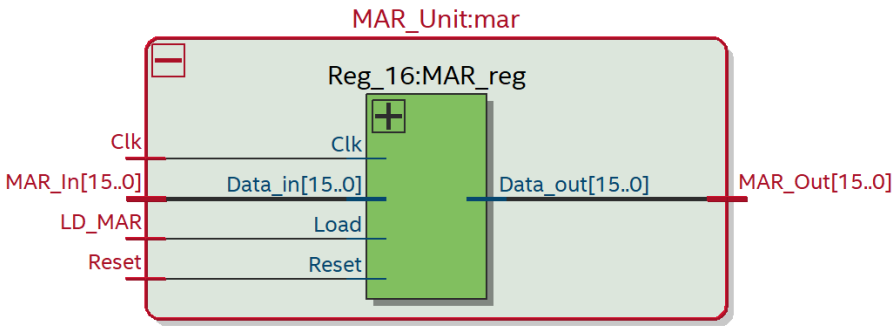
Module: MAR_Unit

Inputs: [15:0] MAR_In, Clk, Reset, LD_MAR

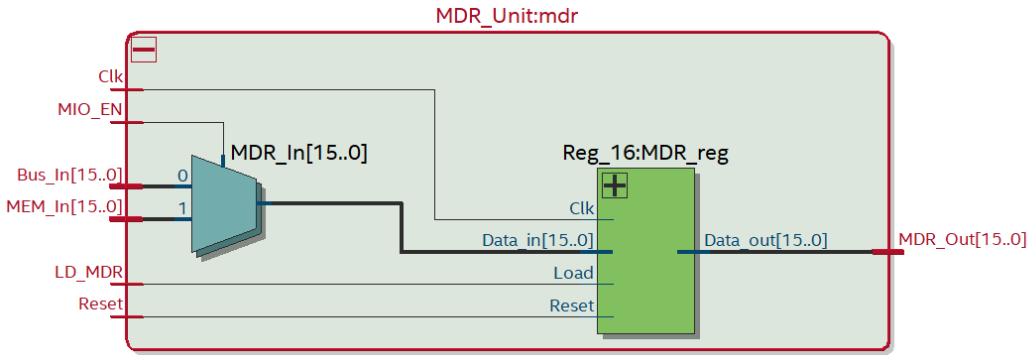
Outputs: [15:0] MAR_Out

Description: This module is a wrapper of a 16-bit register. It will load MAR value from the Bus if *LD_MAR* signal is set and output to MEM2IO and SRAM.

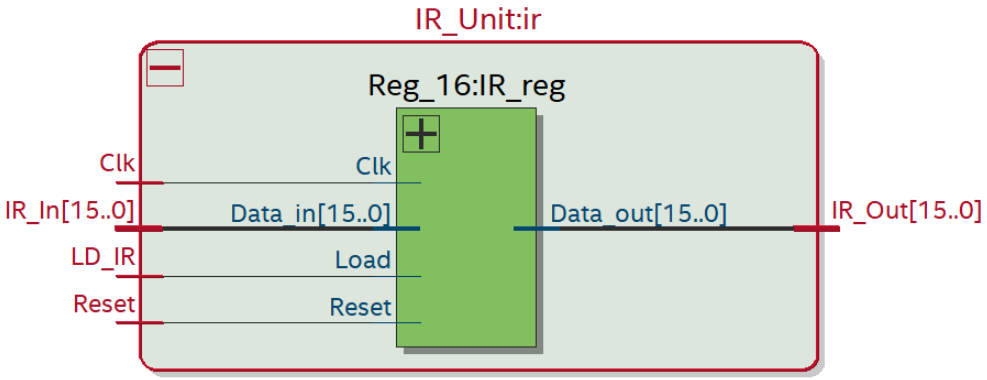
Purpose: This module is the Memory Address Register that save the memory address from the Bus.



Module: MDR_Unit
Inputs: [15:0] Mem_In, [15:0] Bus_In, Clk, Reset, MIO_EN
Outputs: [15:0] MDR_Out
Description: The module consists of a 16-bit register and a set of 2-to-1 MUXes. The MUXes will select the input to the register according to the *MIO_EN* signal. The input can be: the data from the memory interface *Mem_In*, or the data on the Bus *Bus_In*. The output of the register will be *MDR_Out.*
Purpose: This module is the Memory Data Register that save the data from Bus or memory and output them to memory or Bus.



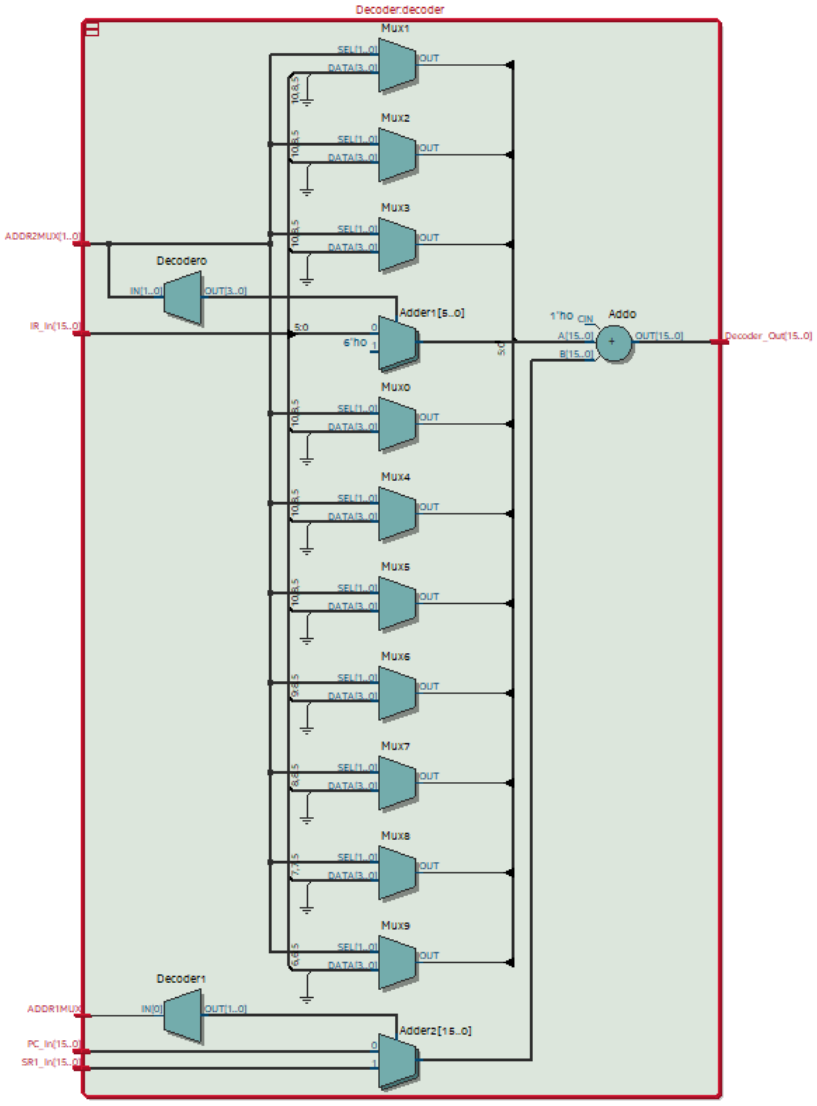
Module: IR_Unit
Inputs: [15:0] IR_In, Clk, Reset, LD_IR
Outputs: [15:0] IR_Out
Description: This module is a wrapper of a 16-bit register. It will load instructions from the Bus if *LD_IR* signal is set and output the instructions.
Purpose: This module is the Instruction Register that save the instructions from the Bus.



Module: Decoder
Inputs: [15:0] IR_In, [15:0] PC_In, [15:0] SR1_In, [1:0] ADDR2MUX, ADDR1MUX
Outputs: [15:0] Decoder_Out
Description: This module consists of a set of 4-to-1 MUXes and a set of 2-to-1 MUXes. The 4-to-1 MUXes will select different length of segments from the *IR_In* and perform the sign extend according to the *ADDR2MUX* signal; the 2-to-1

MUXes will select the value of either the *PC_In* or the *SR1_In* according to the *ADDR2MUX* signal. The sum of the outputs from the MUXes will be *Decoder_Out*.

Purpose: This module will calculate the address from IR, PC or the value in Register Files specified by instructions. The calculated address will go to the Bus or the PC module if corresponding signals are set.



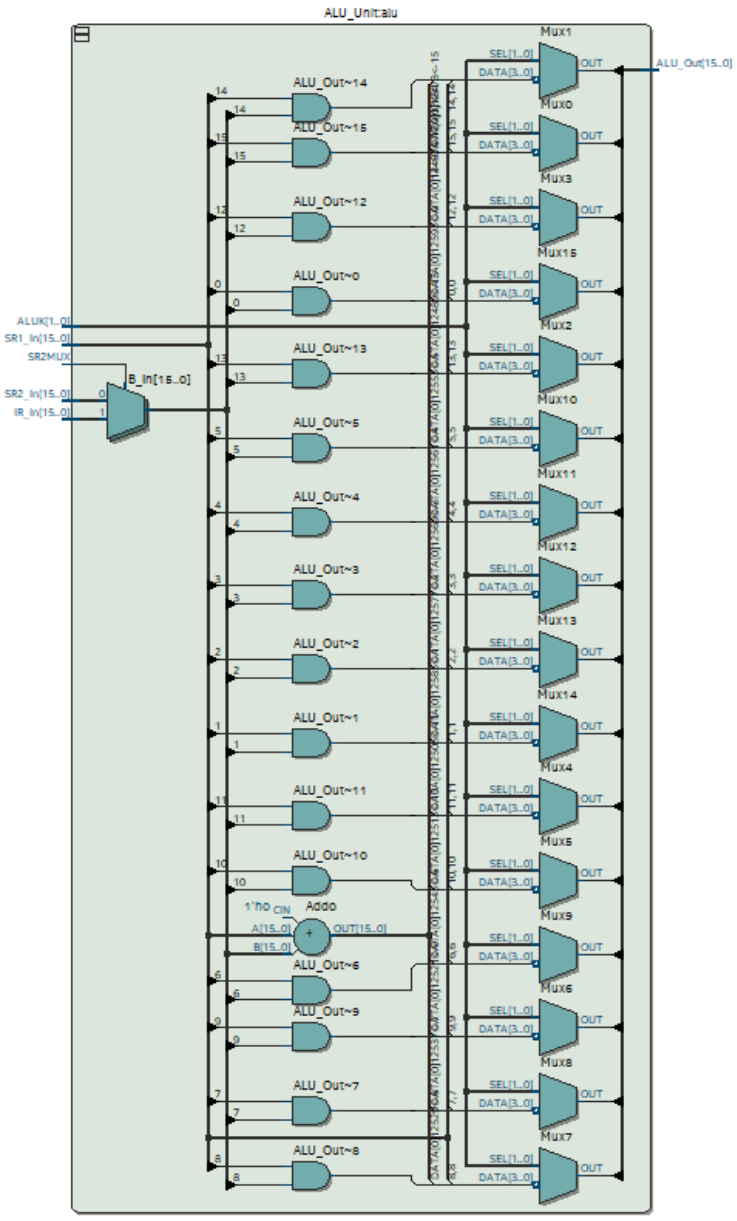
Module: ALU_Unit

Inputs: [15:0] IR_In, [15:0] SR2_In, [15:0] SR1_In, [1:0] ALUK, SR2MUX

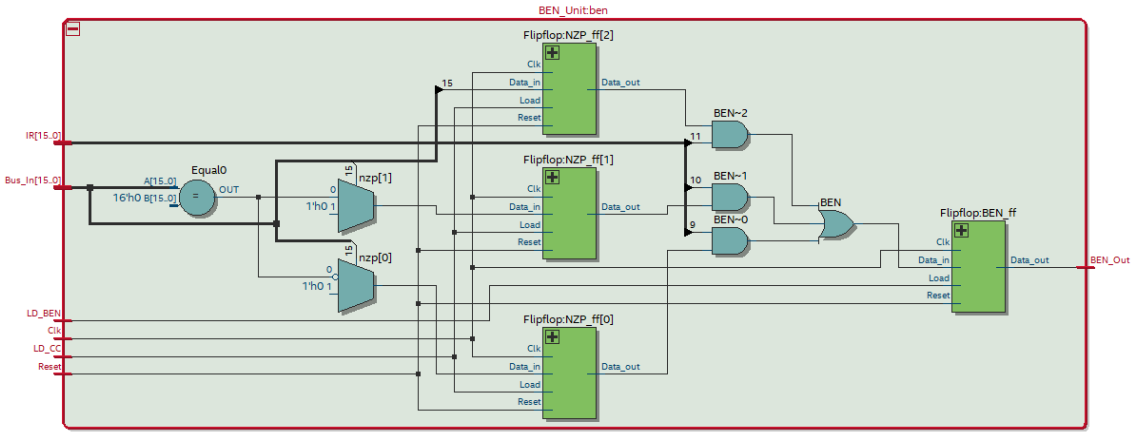
Outputs: [15:0] ALU_Out

Description: This module consists of a set of 2-to-1 MUXes and Arithmetic Logic Unit. The 2-to-1MUXes will select one of the inputs to the ALU from the sign-extended Immediate input value in *IR_In* and the value in *SR2_In*. The ALU will perform one of the four calculations: ADD, AND, NOT and Do Nothing according to the *ALUK* signal. Notice that the latter two operations will only be performed on *SR1_In*.

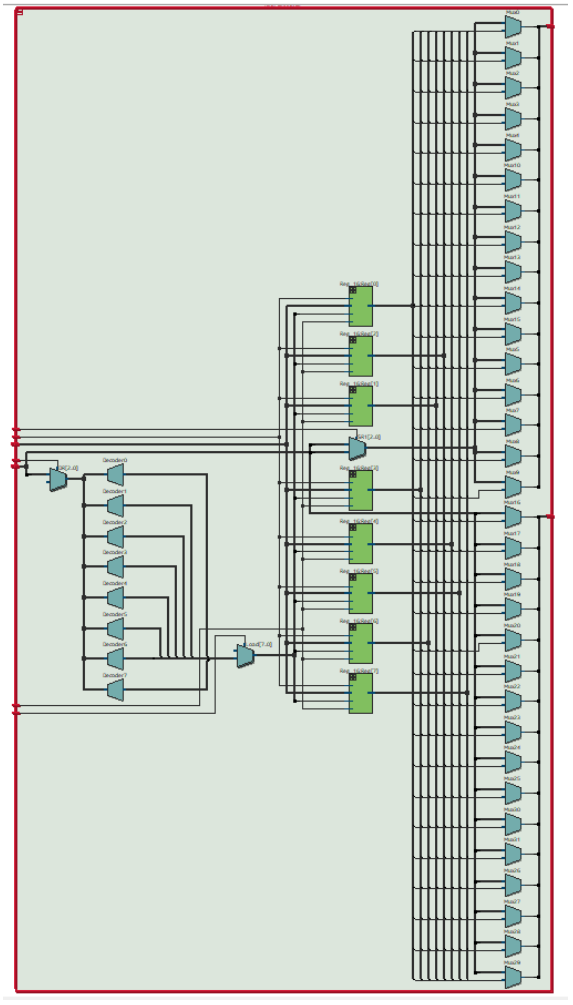
Purpose: This module is the Arithmetic Logic Unit module that selects inputs and perform calculations as the input signals specified.



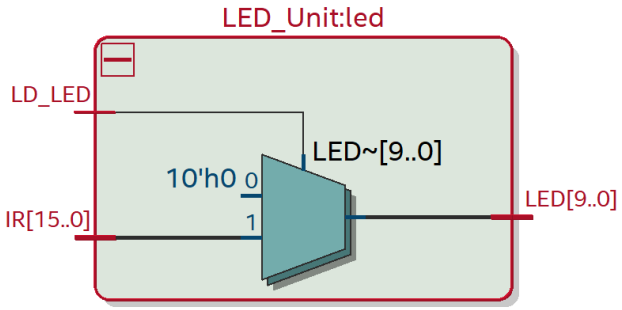
Module: BEN_Unit
Inputs: [15:0] Bus_In, [15:0] IR, Clk, Reset, LD_CC, LD_BEN
Outputs: BEN_Out
Description: The module consists of four flip-flops. The module will first determine the NZP condition of the *Bus_In* value and save them into the three NZP flip-flops respectively if *LD_CC* is set high. Then the NZP information saved in the flip-flops will be check against the NZP conditions in [11:9] *IR*. If the condition matches and *LD_BEN* is set high, the branch enable signal will be saved in the BEN flip-flop.
Purpose: The module will be used in BR instruction and is used to check whether the branch condition is satisfied.



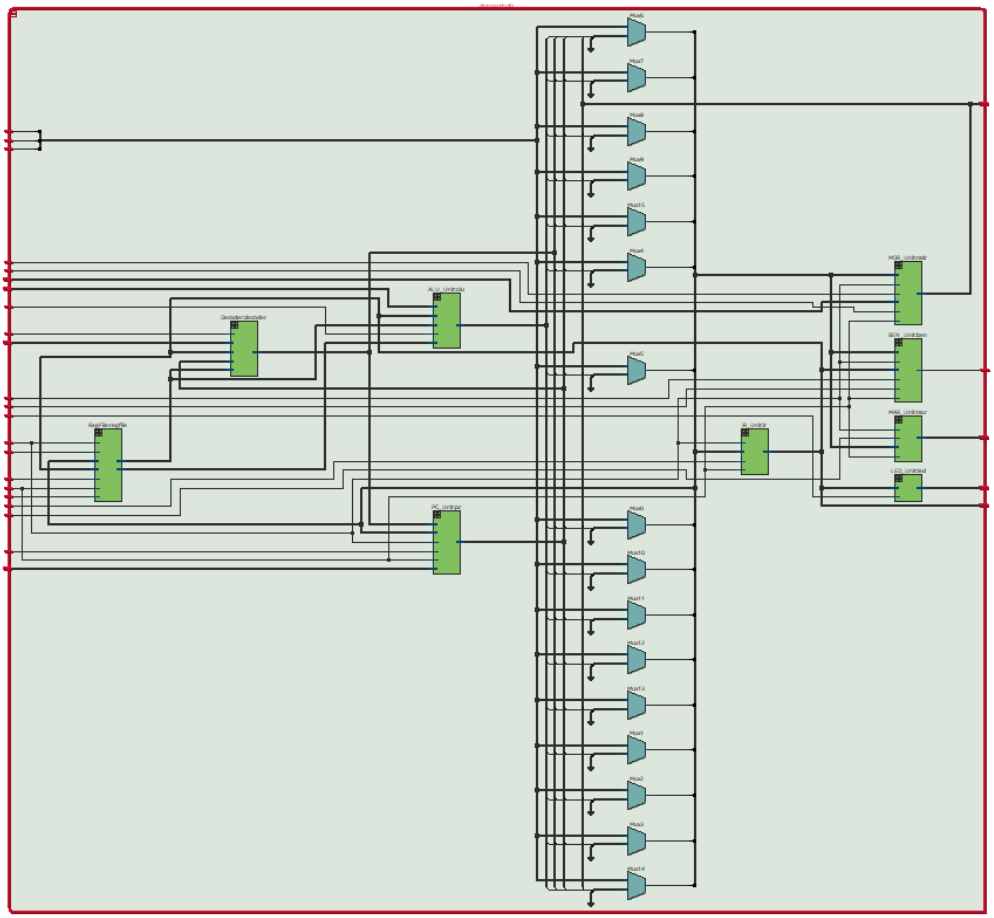
Module: RegFile
Inputs: [15:0] IR_in, [15:0] Data_in, Clk, Reset, LD_REG, DRMUX, SR1MUX,
Outputs: [15:0] SR1_out, [15:0] SR2_out
Description: The RegFile module is consists of 8 16-bit registers. The module has two sets of 2-to-1 MUXes. One set of the MUXes will select the Destination Register (DR) for the input data base on the *DRMUX* signal. The data will be loaded to the DR if the *LD_REG* signal is set high. The RegFile can have two outputs, one of the outputs is selected from one of the eight registers according to two different segments of *IR_in* specified by *SR1MUX*, the other output is selected according to [2:0] *IR_in.*
Purpose: This module is the Register File that used to save values temporarily. Data can be saved and read to the Register File as needed by the instructions.



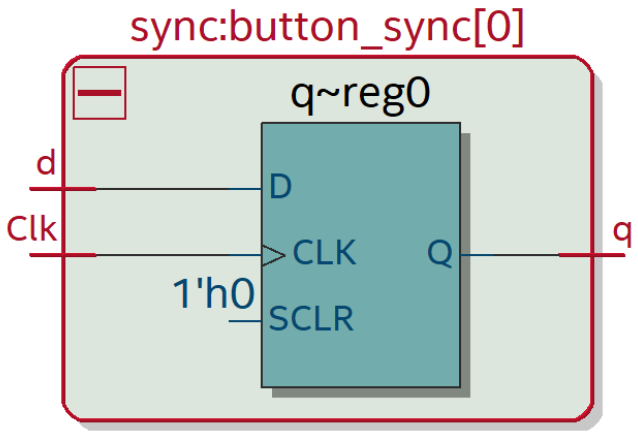
Module: LED_Unit
Inputs: [15:0] IR, LD_LED
Outputs: [9:0] LED
Description: The module will output [9:0] *IR* to *LED* if *LD_LED* is set high.
Purpose: This module is used to drive the LED to display the lowest 9 bits of IR.



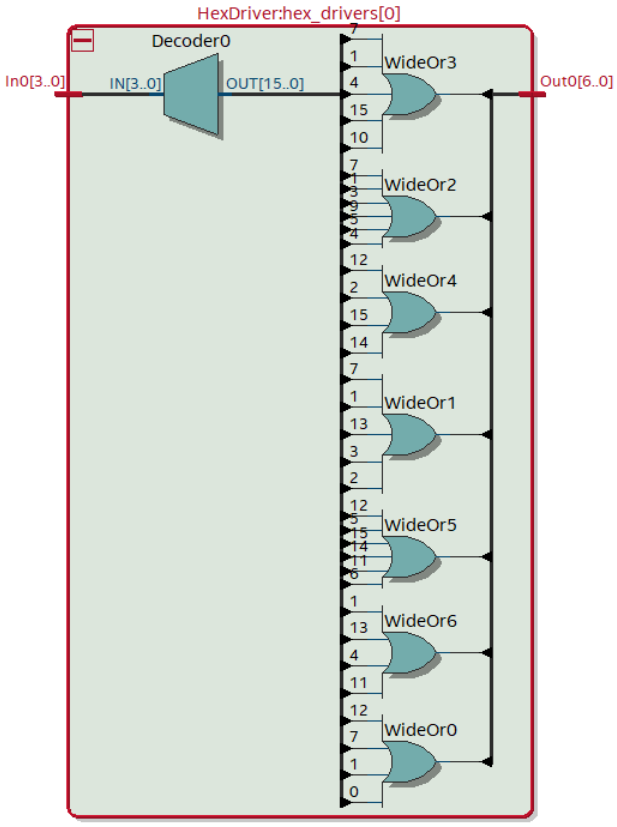
Module: datapath
Inputs: [15:0] MDR_In, [1:0] PCMUX, [1:0] ADDR2MUX, [1:0] ALUK, Clk, Reset, LD_MAR, LD_MDR, LD_IR, LD_BEN, LD_CC, LD_REG, LD_PC, LD_LED, GateMDR, GateALU, GatePC, GateMARMUX, SR2MUX, ADDR1MUX, MARMUX, MIO_EN, DRMUX, SR1MUX,
Outputs: [15:0] MAR, [15:0] MDR, [15:0] IR, [9:0] LED, BEN
Description: The module will instantiate all the modules describe above. The module will also select the value for the Bus from the outputs of MDR, ALU, PC and Decoder modules according to the gate selection signals *GateMDR*, *GateALU*, *GatePC*, *GateMARMUX*.
Purpose: This module is used to connect all the modules and the data bus except the SRAM and the memory interface.



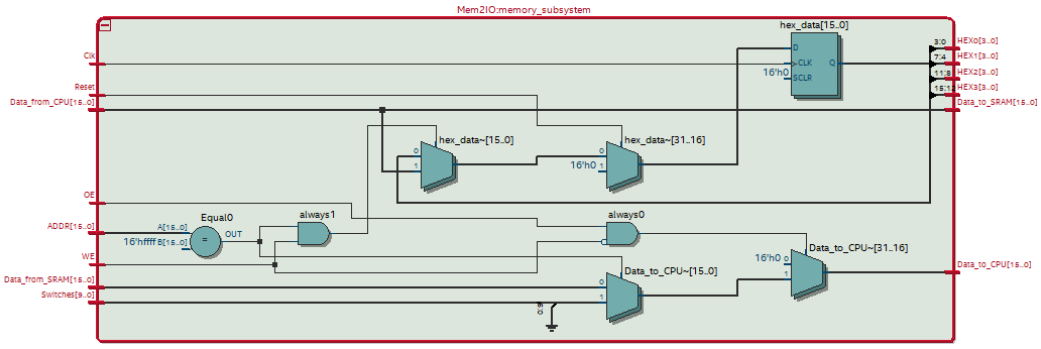
Module: sync
Inputs: Clk, d
Outputs: q
Description: This module is a positive-edge triggered flip-flop.
Purpose: This module synchronizes the asynchronous human input and reduce the metal bouncing effect in switches and buttons.



Module: HexDriver
Inputs: [3:0] In0
Outputs: [6:0] Out0
Description: The module will output the code for seven-segment display depending on the input 4-bit data.
Purpose: This module is used to translate the binary data into the code for hex display on the board.



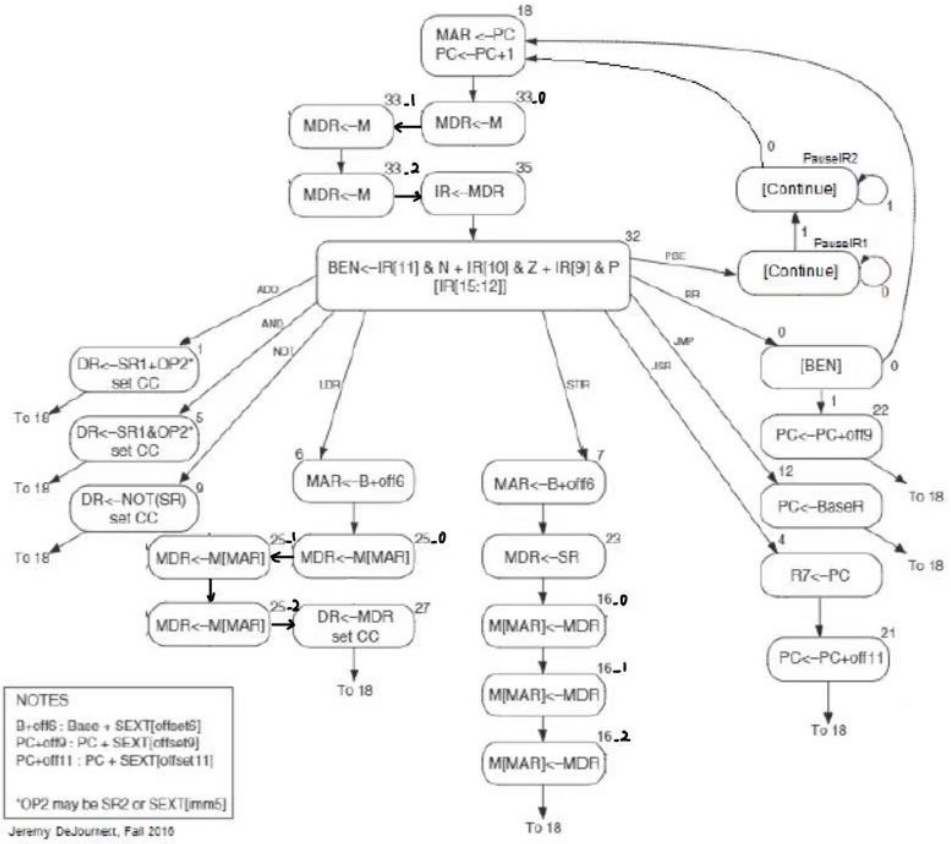
Module: Mem2IO
Inputs: [15:0] ADDR, [15:0] Data_from_CPU, [15:0] Data_from_SRAM, [9:0] Switches, Clk, Reset, OE, WE,
Outputs: [15:0] Data_to_CPU, [15:0] Data_to_SRAM, [3:0] HEX0, [3:0] HEX1, [3:0] HEX2, [3:0] HEX3
Description: The module will send out data to SRAM, CPU and Hex displays depending on the address of *ADDR*, *WE*, and *OE*.
Purpose: This module manages all I/O with the DE10-Lite physical I/O devices, namely, the switches and 7-segment displays.



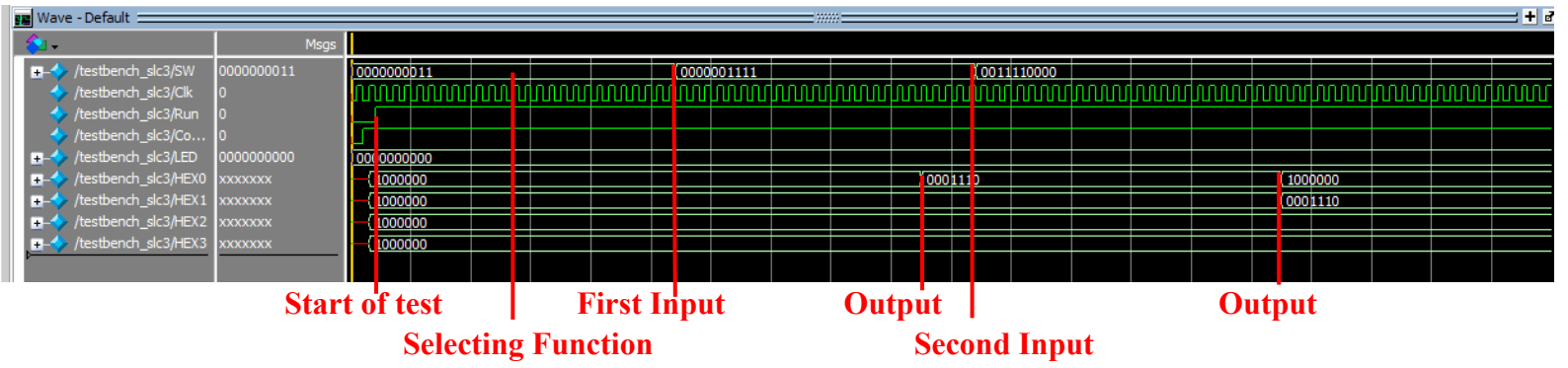
6. Instruction Sequence Decoder Unit

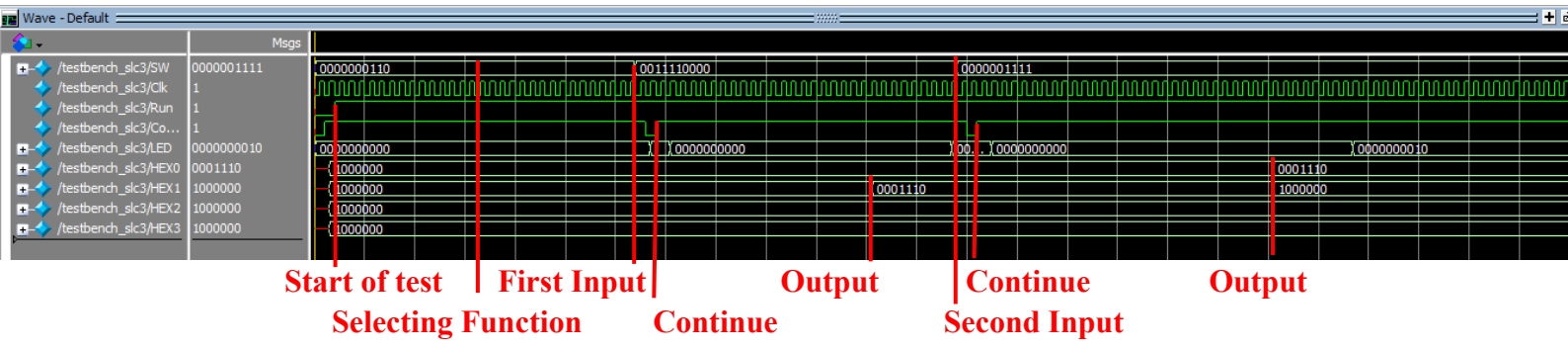
The input to the ISDU module is all the control signals used by the processor. The control signals to the processors including the Load signals to all the registers, the Selection signals to the MUXes and the Gate selection signal for registers outputting data to the Bus. To control the various components, the ISDU module needs to choose the corresponding control signals for each instruction. To be specific, for the instructions that needs to write data on the Bus to registers, the ISDU will set the Load signal to the register; for the instructions that needs to write to the Bus, the module will set the Gate for the signal; for the instructions that needs to select inputs, the ISDU will set the selection for them; if the instruction needs to interact with memory, the module will set the read/write enable for it as well. The ISDU controls the components by sending out specific combinations of control signals and such combinations are defined by the specific task it needs to perform in the instruction sequence. Therefore, an instruction is done by having the control signals properly set for each step.

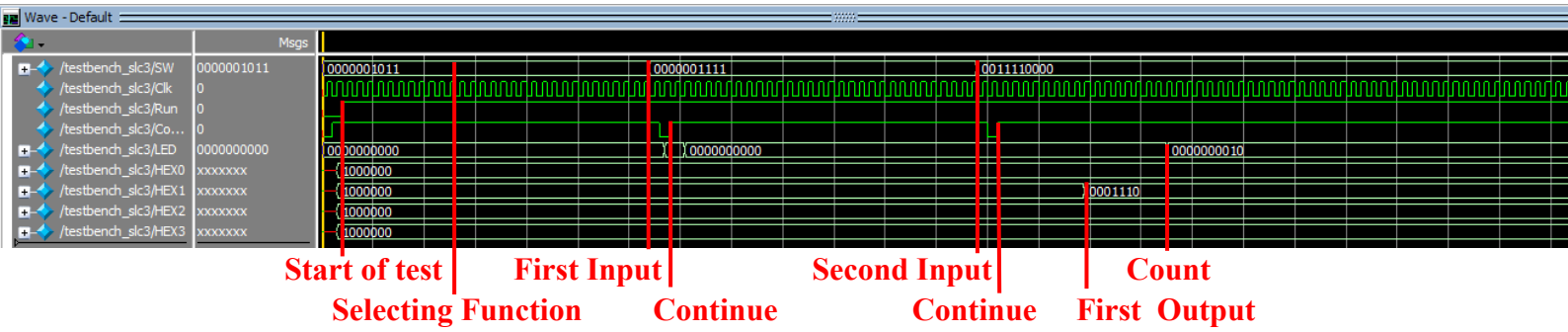
7. State Diagram

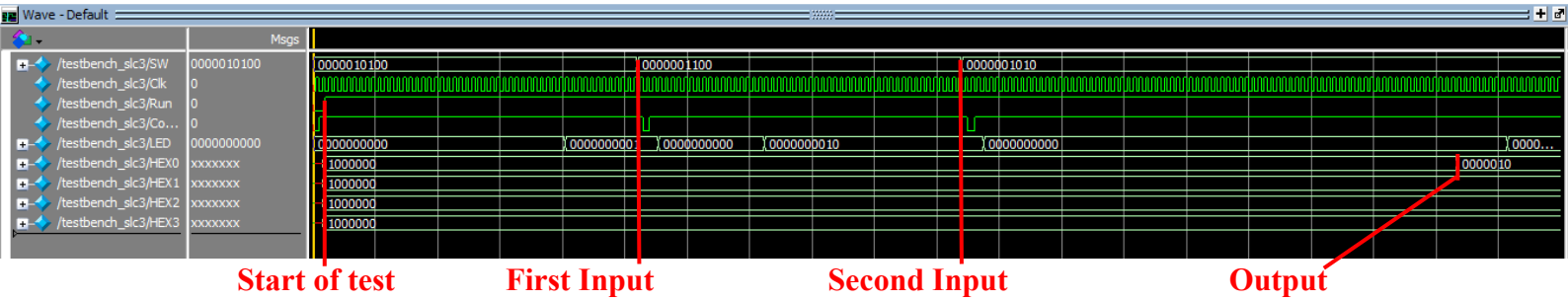


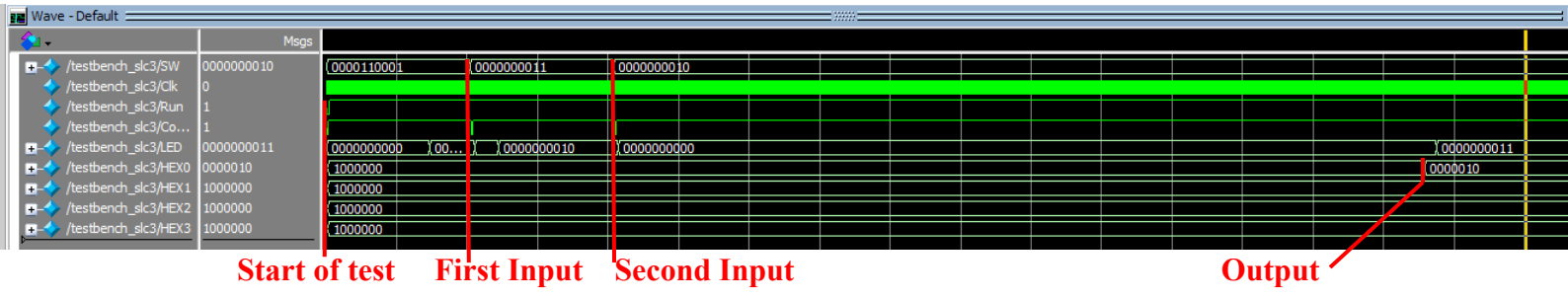
8. Simulation Trace

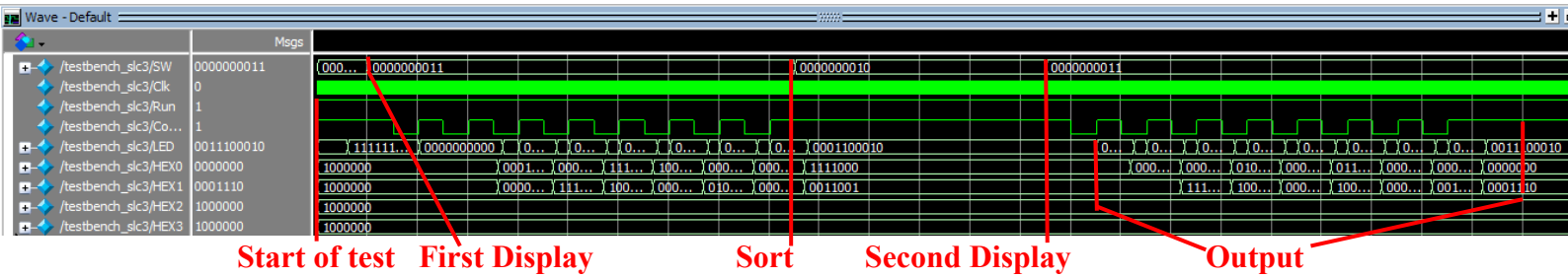












9. Post-Lab Questions

1) Design Analysis

LUT	1186
DSP	0
Memory (BRAM)	0
Flip-Flop	705
Frequency	86.81MHz
Static Power	89.95mW
Dynamic Power	1.65mW
Total Power	100.43mW

2) Function of Mem2IO

The Mem2IO serves as the interface between the processor and the external memory. It will handle the Memory-Mapped I/O base on the address and read/write enable. It can also handle the I/O of the external device to the processor like the switch and LED when the address is set to 0xFFFF.

3) **BR and JMP**

Both BR and JMP can change the PC to a specified address. However, BR is a conditional branch, so that it can only change the PC when the NZP condition is matched. It also takes in the offset of PC change. By contrast, JMP is an unconditional jump, it will jump to the address saved in the Base Register specified in the JMP instruction unconditionally.

4) **the Use of R signal**

R stands for Memory Ready. In the original LC-3 state machine, when the processor needs to access the memory, it will perform a self-loop and wait for the memory ready signal. When the job on the memory side is done, the memory ready will be set to logical high and the state machine can move on to the next state. However, in our design, we removed the signal R, but added two more states for the memory accessing process. This is because we noticed that the speed for memory read/write takes about three clock cycles. This design is also good for synchronization because is such way the memory accessing can strictly align with the clock cycle.

10.Conclusion

a) **Functionality**

The SLC-3 Processor can perform nine instructions and can access with the external memory, switches, and LEDs. With the programs being preloaded to the memory, we tested its ability of perform programs and it works just as expected.

b) **Comment on Lab Manual**

Everything is good.