ECE 385

Spring 2024

Experiment 5

Simple Computer SLC-3.2 in SystemVerilog

By Wendi Wang (wendiw2)

1. **Introduction**
   a. **Basic Functionality**

      This week's lab is about designing a simple microprocessor, containing a subset of the LC-3 ISA, a 16-bit processor with 16-bit Program Counter, and 16-bit registers. There are three main components in this microprocessor, namely the central processing unit (CPU), the memory storing instructions and data, and the input/output interface that communicates with external devices. Finally, this microprocessor will perform various operations based on the opcodes, and it is able to return to the halted state any time a reset signal arrives.

   b. **Differences from ECE120/220's LC-3 processor**

      1) ECE120/220's LC-3 processor has four tristate buffers to select which signal to drive the CPU bus, while FPGA doesn't support internal tristate buffers, therefore the tristate buffers are replaced by a MUX.
      2) LC-3 processor has a MARMUX to select the address or zero-extended lower 8 bits of the instruction, but in SLC-3 this MUX is deleted and only the address is connected to the CPU bus under the control of GateMARMUX signal.
      3) LC-3 processor has a R signal to indicate whether memory reading is finished, but in SLC-3 doesn't support R signal and we need to wait for enough time to ensure memory reading is finished.
      4) SLC-3 cannot perform TRAP, LEA (load effective address instruction with opcode 1110), STI (store instruction using indirect addressing mode with opcode 1011), LDI (load instruction using indirect addressing mode with opcode 1010), ST (store instruction using PC relative addressing mode with opcode 0011), and LD (load instruction using PC relative addressing mode with opcode 0010) instructions, which are all supported in LC-3.

2. **Written Description and Diagrams of SLC-3**
   a. **Summary of Operation**

      Our SLC-3 microprocessor executes different instructions using several different opcodes. There are three components of SLC-3, namely CPU, memory, and input/output interface. As a subset of the LC-3, it has a data bus, a state machine as well as a memory (either on-chip or test memory). SLC-3 performs tasks following FETCH->DECODE->EXECUTE->FETCH (NEXT) cycle.

   b. **SLC-3 Functionality**

      In the FETCH phase, the instruction is fetched from memory following 4 steps:
      MAR <- PC (MAR stores PC's value, the address of instruction to be executed)
      MDR <- M(MAR) (MDR stores the content of MAR)
      IR <- MDR (IR stores the instruction to be executed)
      PC <- PC + 1 (PC increments itself for next FETCH)

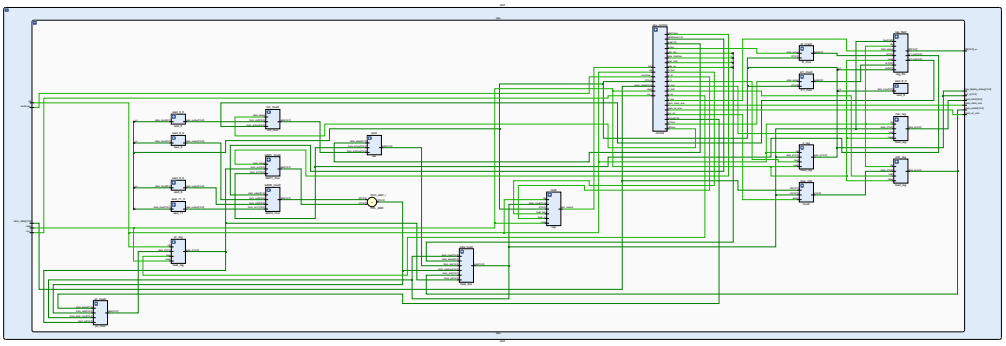
In the DECODE phase, the processor figures out which operation current instruction belongs to according to the opcode, and the control signals required to execute this instruction is fixed as well.

In the EXECUTE phase, the operation mentioned in DECODE phase is executed and the result is stored in memory or targeting register depending on the type of instruction.

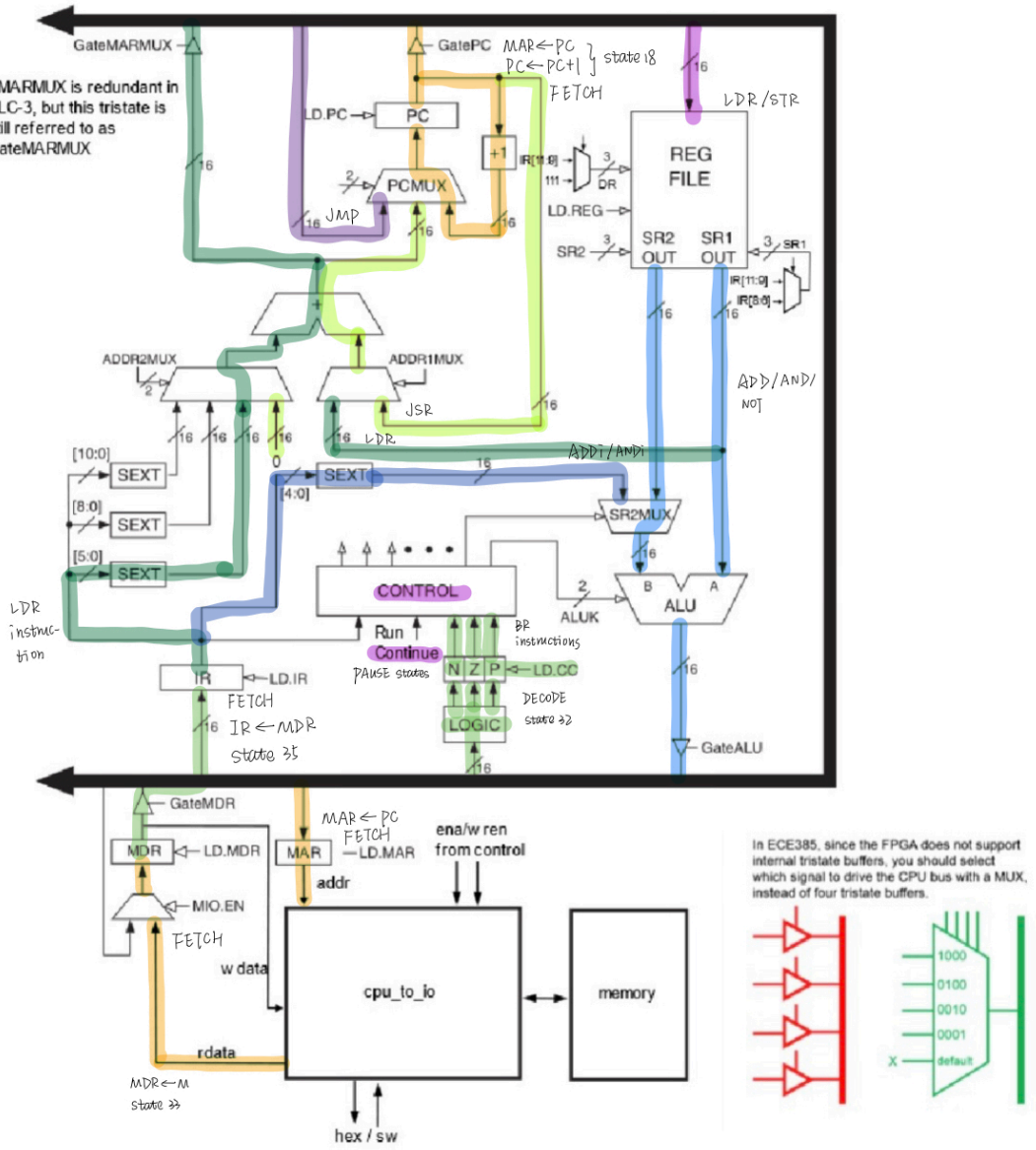
The following table demonstrates detailed instructions our SLC-3 can perform:

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

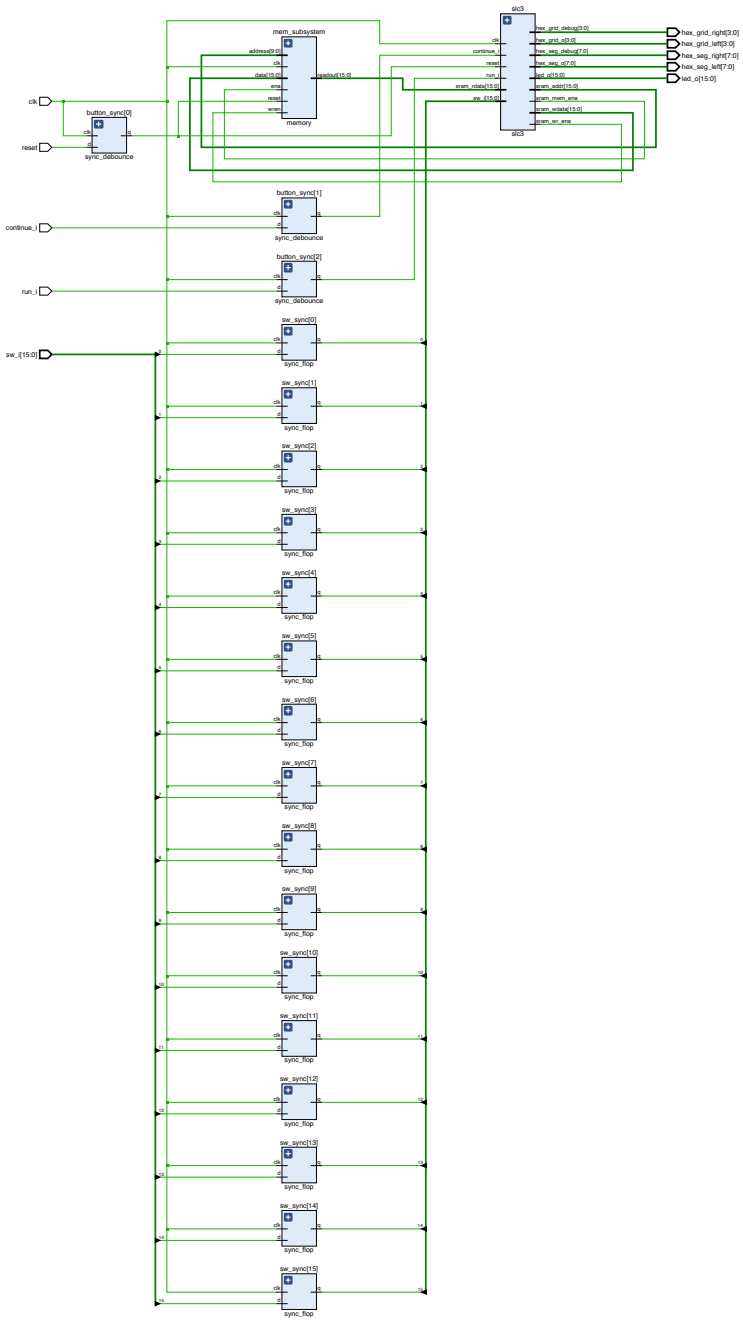
c. Block Diagram of cpu.sv



d. Block Diagram with Annotations



e. Block Diagram of the Top-level



f. Written Description of All .sv Modules

Module: control.sv

Inputs: clk, reset, ir, ben, mem_rdata, continue_i, run_i

Outputs: ld_mar, ld_mdr, ld_ir, ld_ben, ld_cc, ld_reg, ld_pc, ld_led, gate_pc, gate_mdr, gate_alu, gate_marmux, pcmux, drmux, sr1mux, sr2mux, addr1mux, addr2mux, aluk, mio_en, mem_mem_ena, mem_wr_ena

Description: This file contains a control unit of guiding the process of SLC-3. It generates several signals to control the execution of different instructions under state flows. There are several states executing specific functions depending on the opcodes. It is the state machine of SLC-3, which goes through FETCH, DECODE and all EXECUTE states.

States flow generate different values of signals for cpu.sv to use when performing EXECUTE stage.
Purpose: Generate a list of control signals to control the process of SLC-3 using instructions' opcodes.

Module: cpu_to_io.sv
Inputs: clk, reset, [15:0] cpu_addr, cpu_mem_ena, cpu_wr_ena, [15:0] cpu_wdata
Outputs: [15:0] cpu_rdata, [15:0] sram_addr, sram_mem_ena, [3:0] hex_grid_o, [7:0] hex_seg_o
Description: This file is a bridge between the CPU and physical I/O devices. It communicates between the memory and switches/HEX.
Purpose: Data transmitting between CPU and I/O devices (switches and HEX).

Module: cpu.sv
Inputs: clk, reset, run_i, continue_i, [15:0] mem_rdata
Outputs: [15:0] hex_display_debug, [15:0] led_o, [15:0] mem_wdata, [15:0] mem_addr, mem_mem_ena, mem_wr_ena
Description: This module uses all signals from control.sv to perform different kinds of operations. It defines many computation units and registers under the control of those signals as internal connections.
Purpose: Use a list of signals to perform EXECUTE stage.

Module: processor_top.sv
Inputs: clk, reset, run_i, continue_i, [15:0] sw_i

Outputs: [15:0] led_o, [7:0] hex_seg_left, [3:0] hex_grid_left, [7:0] hex_seg_right, [3:0] hex_grid_right

Description: This module contains all components in SLC-3 processor, namely CPU, memory and input/output interface that communicates with external devices.
Purpose: Provide a top-level functionality of SLC-3 microprocessor.

Module: slc3.sv
Inputs: clk, reset, run_i, continue_i, [15:0] sw_i

Outputs: [15:0] led_o, [7:0] hex_seg_o, [3:0] hex_grid_o, [7:0] hex_seg_debug, [3:0] hex_grid_debug, [15:0] sram _addr, sram_mem_ena, sram _wr_ena, [15:0] sram_wdata

Description: This module defines the CPU and CPU_TO_IO components of SLC-3 processor.
Purpose: Serving as the entity of CPU and CPU_TO_IO.

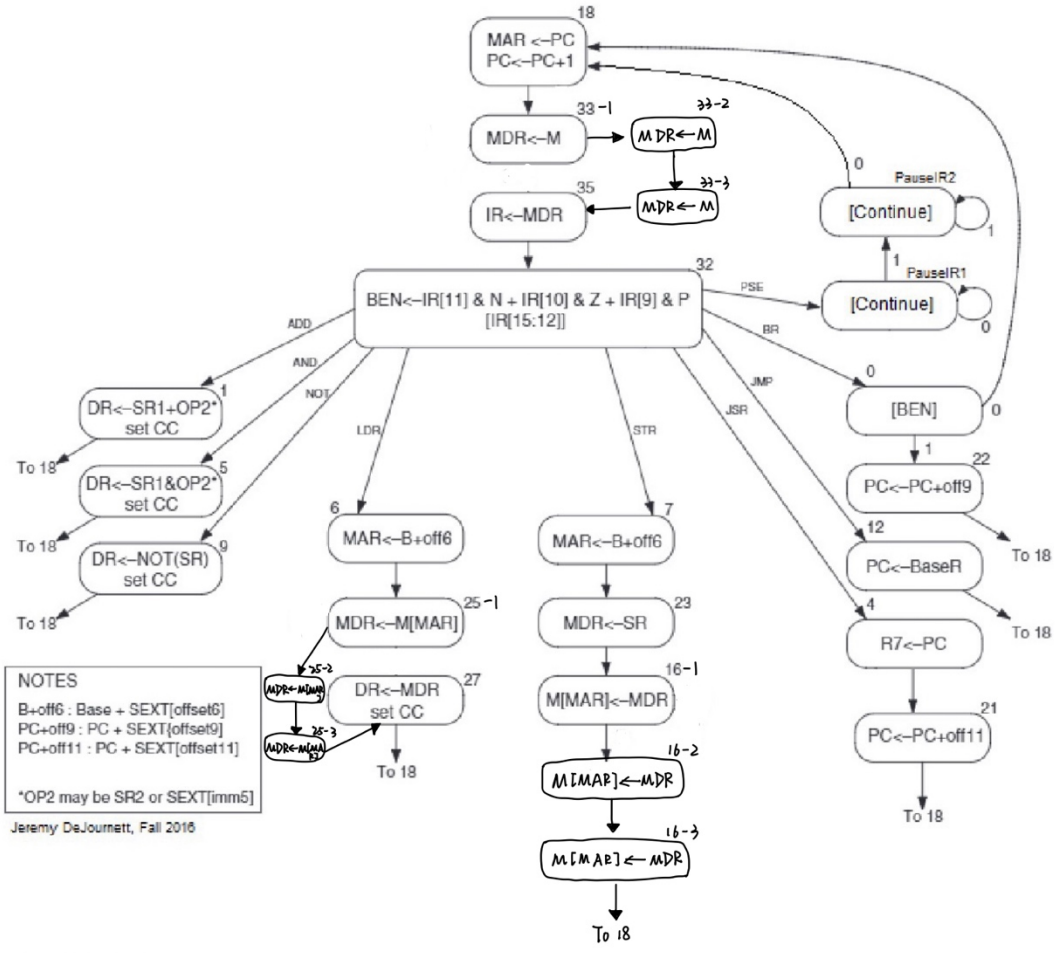
Module: hex_driver.sv
Inputs: clk, reset, [3:0] in[4]
Outputs: [7:0] hex_seg, [7:0] hex_grid
Description: This file contains a module of visualizing outputs. To be specific, it contains a module of demonstrating output values on LEDs using 7-segment method. An LED digit on the Urbana Board is divided into 7 segments, with different segment combinations we can express numbers from 0 to F in hexadecimal.
Purpose: To translate hex numbers (from 0 to F) into figures appearing on LEDs.

Module: sync.sv
Inputs: clk, d
Outputs: q
Description: This file contains a module implementing a denouncer circuit and synchronizer. It serves as a synchronizer for pushbutton and switch inputs, addressing potential instability by detecting stable input cycles using a counter. The choice of counter width is conditional, using a larger width for synthesis to optimize performance.
Purpose: Implementing a denouncer circuit and synchronizer.

Module: load_reg.sv
Inputs: clk, reset, load, [DATA_WIDTH-1:0] data_i
Outputs: [DATA_WIDTH-1:0] data_q
Description: This file contains the module of loading registers of length DATA_WIDTH. With DATA_WIDTH specified when instantiating the register and the load signal held high, this module can be reset or loaded the value in a module instantiation according to the values of its inputs.
Purpose: Load the input value to a specified register when certain signal is held high, otherwise load zero to the specified register.

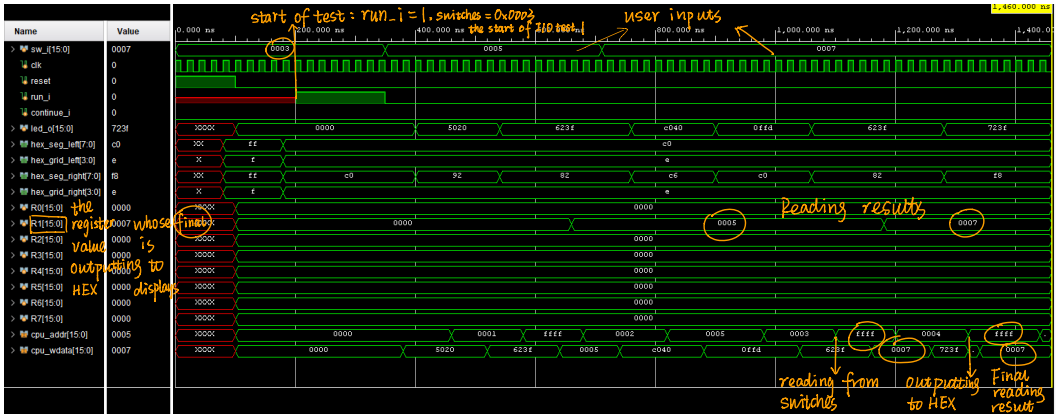
Module: memory.sv
Inputs: clk, reset, [15:0] data, [9:0] address, ena, wren
Outputs: [15:0] readout
Description: This file contains the module of on-chip memory when running synthesis or simulations.
Purpose: Define memory type (physical on-chip memory or virtual test memory) according to different task.

g. State Diagram of Control Unit

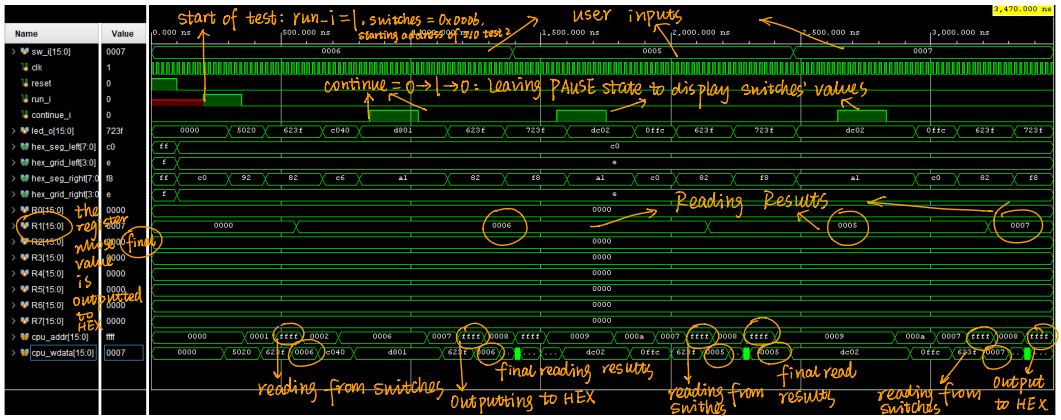


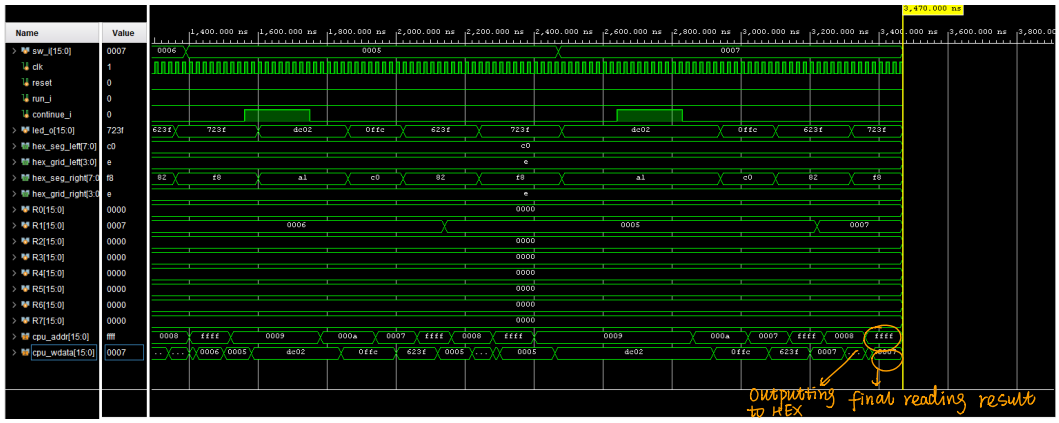
3. Simulations of SLC-3 Instructions

I/O test 1:

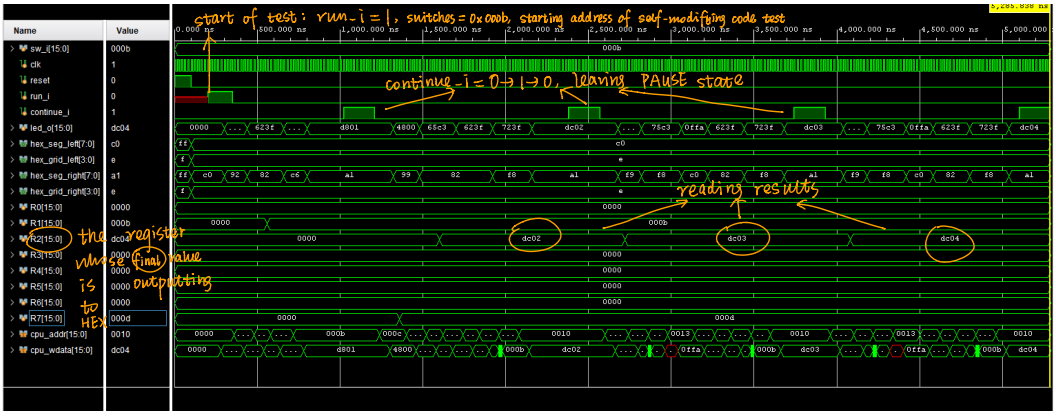


I/O test 2:

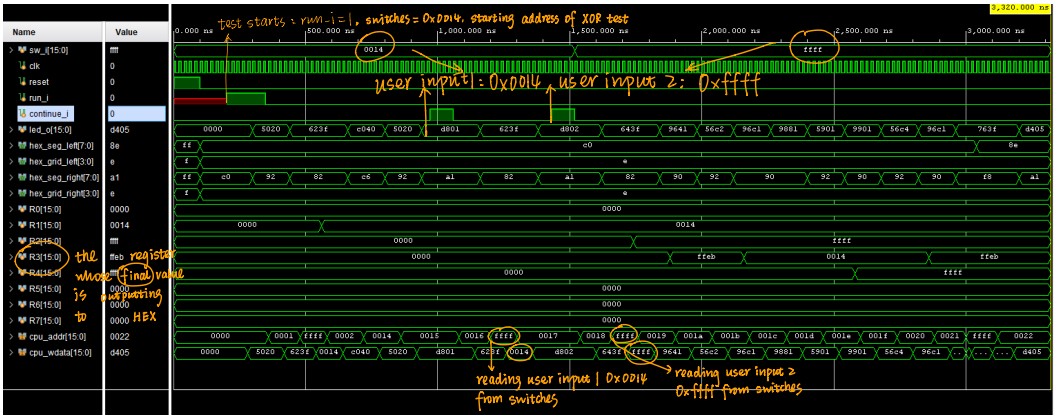


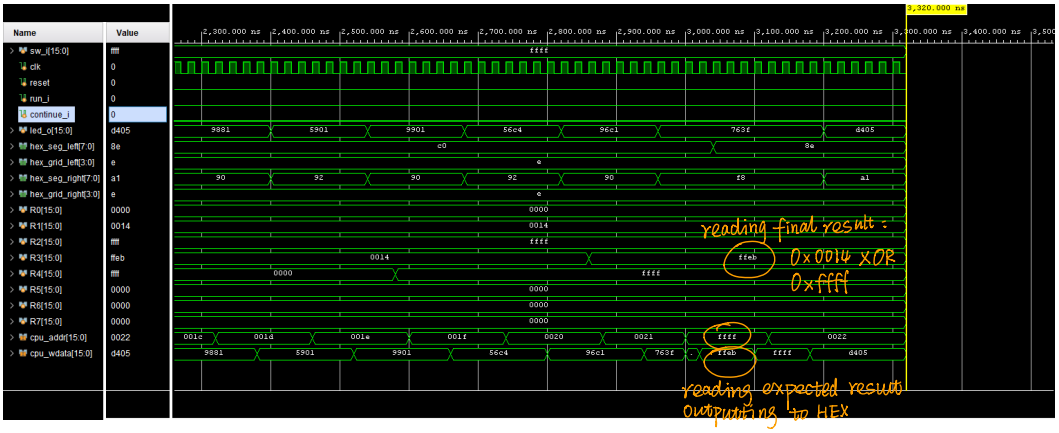


Self-modifying code test:

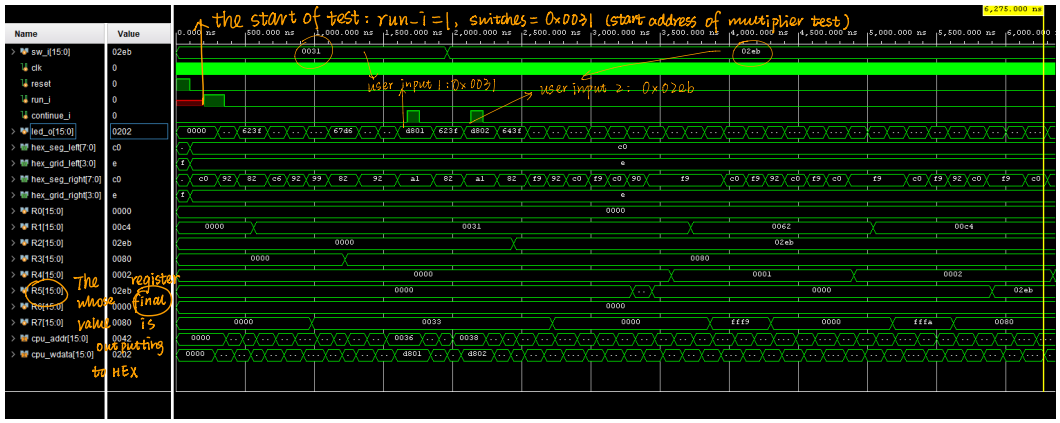


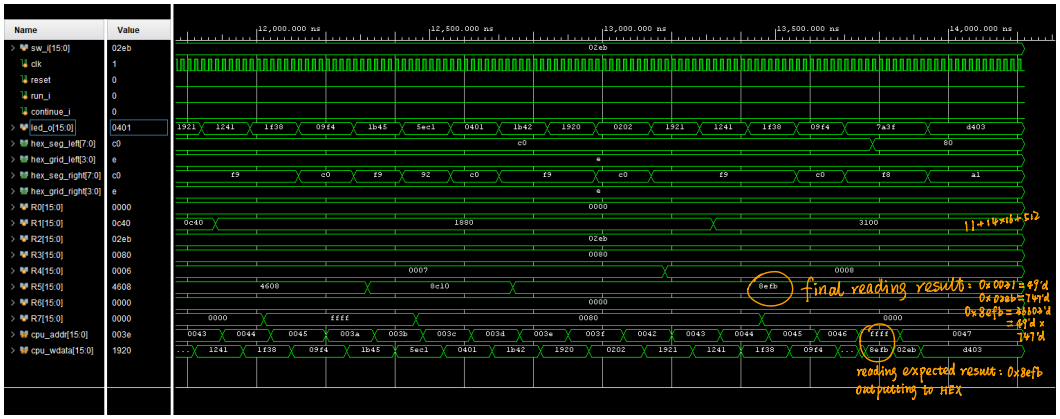
XOR test:



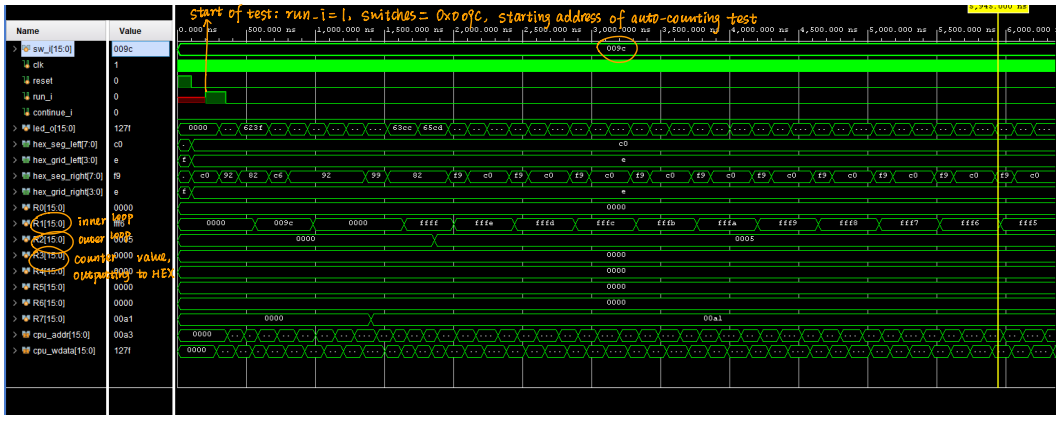


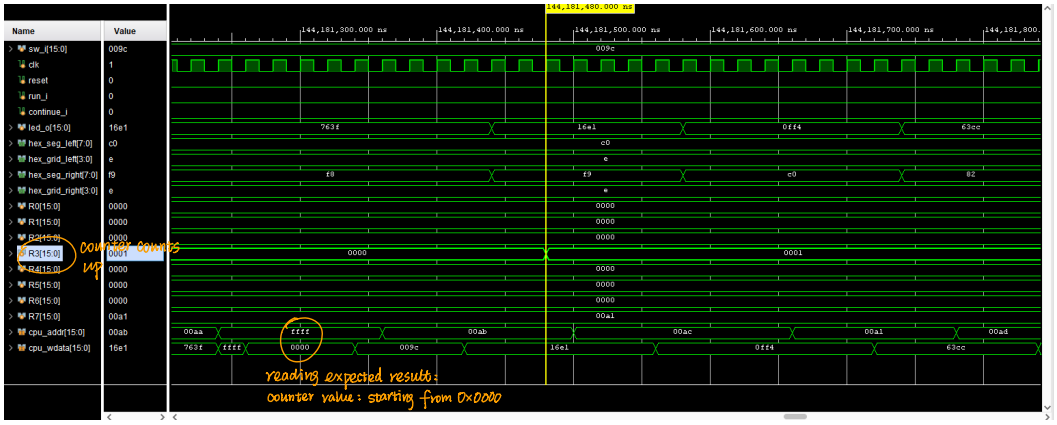
Multiplier test:

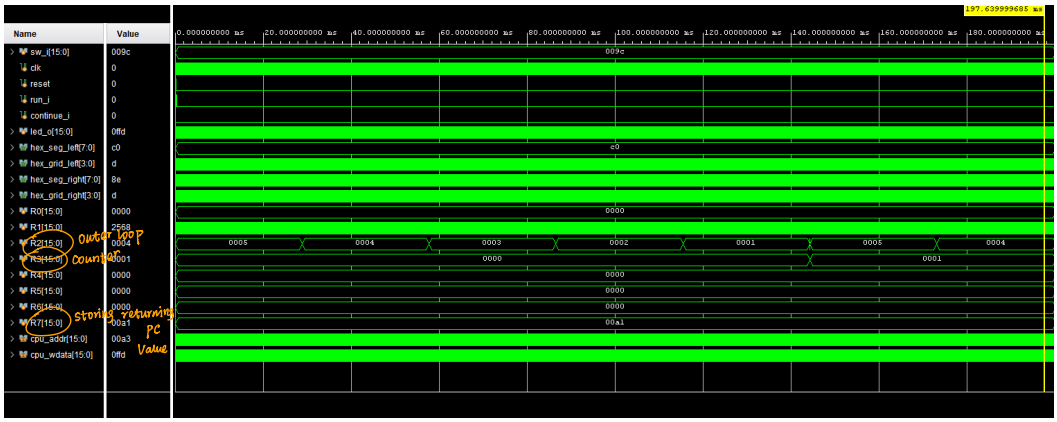




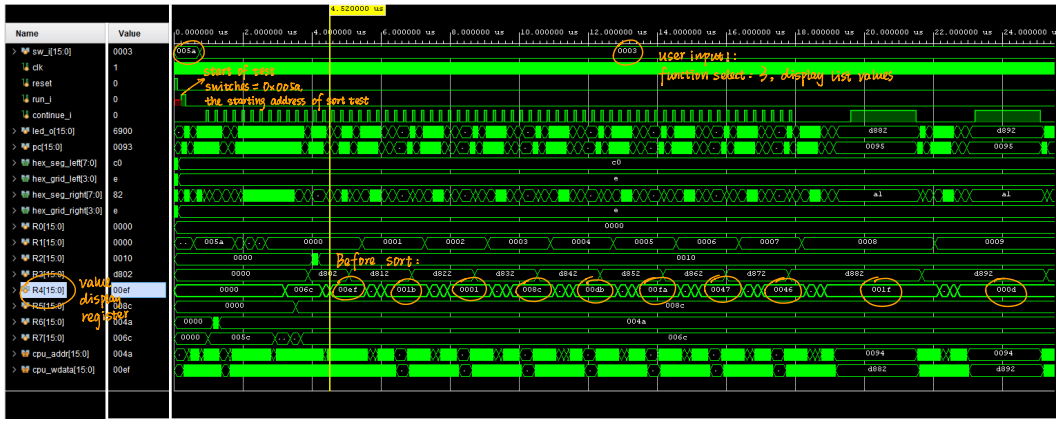
Auto-counting test:

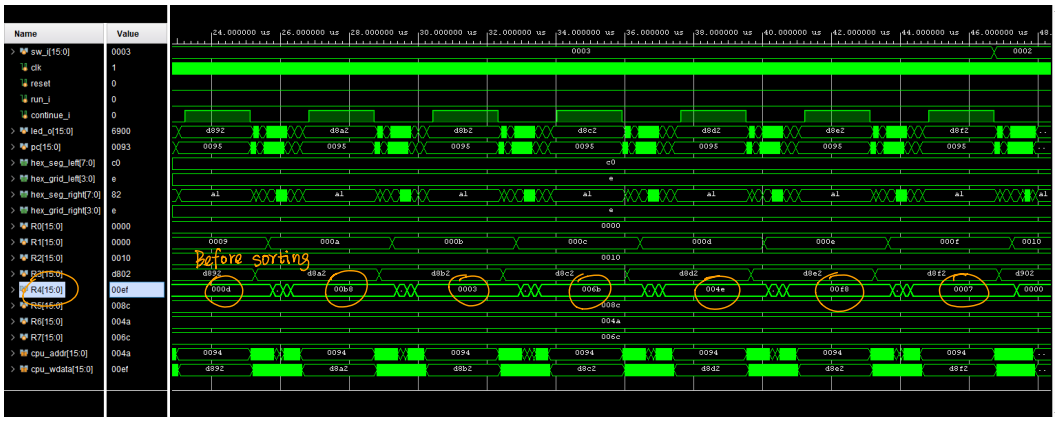


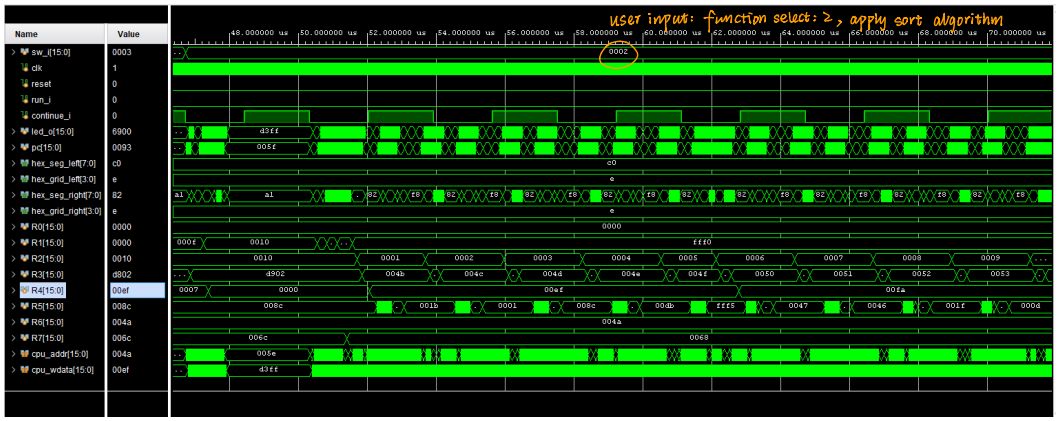

counter counts up
reading expected results:
counter value: starting from 0x0000

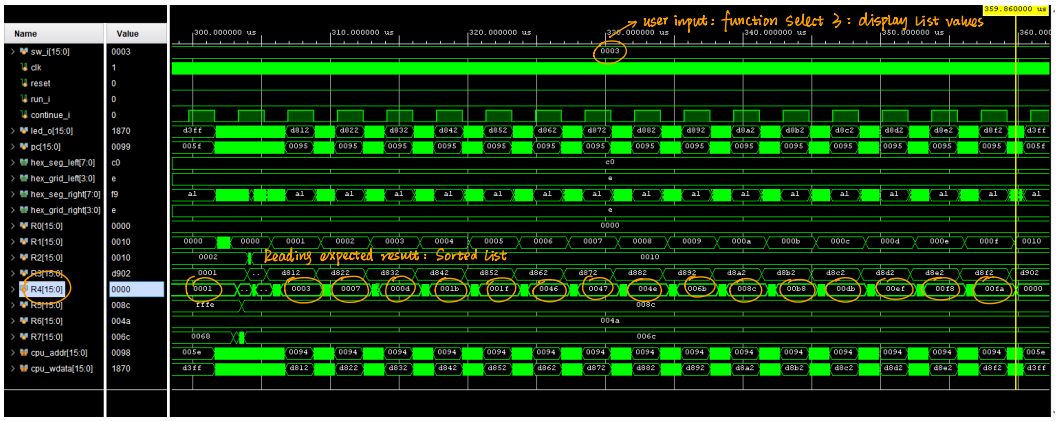

outer loop counter
storing returning PC value

Sort test:


start of sort
switches = 0x005a
the starting address of sort test
user input:
function select = 3, display list values
Before sort:
value displayed in register







4. Answers to Post-lab Questions

a. Design Statistics Table

LUT	428
DSP	0
Memory (BRAM)	0
Flip-Flop	345
Latches	0
Frequency	108.1 MHz
Static Power	0.071 W

Dynamic Power	0.015 W
Total Power	0.086 W

Problems met: Cannot output registers' values to HEXs on the board.

Solutions: Set MIO_EN signal to 0 to enable reading from the bus and then outputting to HEXs on the board.

Conclusion: We need to fully consider the matching of all possible signals' values to selecting data, thus figuring out what exact value we want a signal to be to meet our expectations.

b. What is CPU_TO_IO used for?

CPU_TO_IO is used for providing an interface between on-chip memory and physical I/O devices. Two physical I/O devices, Urbana Board Hex display and Urbana Board Switches, share the same memory address 0xFFFF since the previous one is purely output and the latter is purely input. With CPU_TO_IO as a bridge, we can read values from switches and display values through hex displays.

c. What are the differences between BR and JMP instructions?

BR instruction is a conditional jumping and JMP is an unconditional jumping. Both directly change PC values. However, BR instruction changes PC under the control of NZP signal. If the value of NZP signal matches the condition of the previous instruction, PC's value will change, while JMP instruction changes PC's value to the value stored in a register without any condition. Besides, JMP instruction can change PC to any 16-bit values, but BR instruction can only change PC using a 9-bit offset.

d. 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 impact does this have on performance?

The R signal indicates whether the memory access is finished or not, thus indicating whether the data is ready for further usage. We compensate for the lack by forcing the instructions related to memory access to wait 3 or more clock cycles before moving on. Impact on performance: Since we don't force the processor to wait when raising states, all the states related to memory access will wait for the same clock cycles, making our processor asynchronous. Sometimes we don't need to be waiting for 3 or more clock cycles before using the data, causing some waste of time since R signal can synchronously reporting readiness of the data but ours cannot.

5. Conclusion

a. Functionality of This Design

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

This week's design of SLC-3 completes all functionality requirements. This SLC-3 can perform input/output tasks, memory access as well as different calculations. Using different control signals and the datapath, this SLC-3 microprocessor can work well in executing 11 types of instructions (see the table above).

b. Summary

I think the explanation of cpu-to-io digrams (attached below) in the lab manual is intuitively clear and direct. It helps me understand the codes in cpu_to_io.sv and realizes where some bugs in my codes come from.