ECE 385

Fall 2022

Lab 5

Simple Computer SLC-3.2 in SystemVerilog

Zhuoyuan Li, Shihua Cheng

Tianhao Yu

Introduction

This lab aims to construct a 16 bit LC-3 microprocessor with 16 bit program counter, 16 bit instructions and 16 bit registers using the FPGA board and SystemVerilog. The processor is capable of fetching instructions from the SRAM, which serves as memory, decoding the instruction and executing it. Instructions include ADD, ADDi, AND, ANDi, NOT, BR, JMP, JSR, LDR and STR.

Written Description and Diagrams of SLC-3

Summary of Operation

On the FPGA board, Key 0 is the Continue button and Key 1 is the Run button. For a reset, both Key 0 and Key 1 are pressed at the same time. Immediately after each reset or after the board is programmed, the InstantiateRam module automatically writes 156 16-bit instructions to the on-chip memory starting at memory address 0x0000. These are predefined test programs to be run by the SLC. Before pressing any buttons, the user needs to set the switches on the board to the starting address of the test program to be run. The starting addresses of the programs are shown in the table below.

Program	Address	Description
Basic I/O 1	x0003	Hex displays show the current value from switches.
Basic I/O 2	x0006	Hex displays show the current value from switches, but only update when Continue is pressed.
Self-Modifying Code	x000B	Hex displays show the current value from switches, only update when Continue is pressed, and the value from LEDs is incremented per press of Continue.
XOR	x0014	Enter two operands using switches, load each operand using Continue, and the result is displayed on the hex displays.
Multiplication	x0031	Enter two operands using switches, load each operand using Continue, and the result is displayed on the hex displays.
Sort	x005A	Accept an array of input values (Optional) or use the preset values and run bubble sort. Can display array values before and after sort.
“Act Once”	x002A	Display a counter that starts from zero and gets incremented by one per press of Continue.

After setting the address, press Run to run the program. This will let the control unit FSM go to Fetch state from Halted state, and the instruction at 0x0000 will be run. These instructions read the value of the switches, load the value to a register through Mem2IO, and jump to the address indicated by the switches. Then the user can provide inputs needed by the program

using the switches. The Continue button is needed to continue the program when the Pause instruction is executed. This can be indicated by the on-board 10 LEDs. The LEDs will not light up unless the current instruction is Pause. These LEDs display the lower 10 bits of the Pause instruction. The meaning of the 10 bits are defined differently in each program. The on-board hex displays will show the result of the program. Pause is used either to display a value or to accept new inputs. The programs are designed to run in infinite loops, so a reset (both keys pressed) is needed to run a different program.

How the SLC-3 performs its functions

The SLC-3 performs its functions by repeatedly running the Fetch-Decode-Execute cycle. The data operations are done in the datapath that we implemented in datapath.sv, where smaller modules such as the data bus, the register file, the ALU, and the internal registers such as PC, IR, MAR and MDR. The datapath is controlled by a control unit based on a finite state machine. In each different state, the control unit sends out different control signals for the datapath to operate different functions. The states of the finite state machine are defined inside the module ISDU.sv.

The Fetch operation is done in 5 states in our implementation. The first state is to load PC to MAR and increment PC. The next 3 states do the same thing, they read the instruction at memory address equal to MAR and load it to MDR. Three states are used because no ready signal is available so we need to wait long enough for the data to be ready. The 5th state is to load the value in MDR to IR, which is the next instruction to be executed.

The Decode operation is done in one state. In this state, the load signal of BEN is enabled and BEN is loaded from the combinational logic value of IR[11]&N + IR[10]&Z + IR[9]&P. N,Z,P are always updated in instructions ADD, AND, NOT and LDR. Next, a MUX is used to decode the first 4 bits of the instruction (the Opcode). A table showing all the instructions supported by the SLC-3 is shown below.

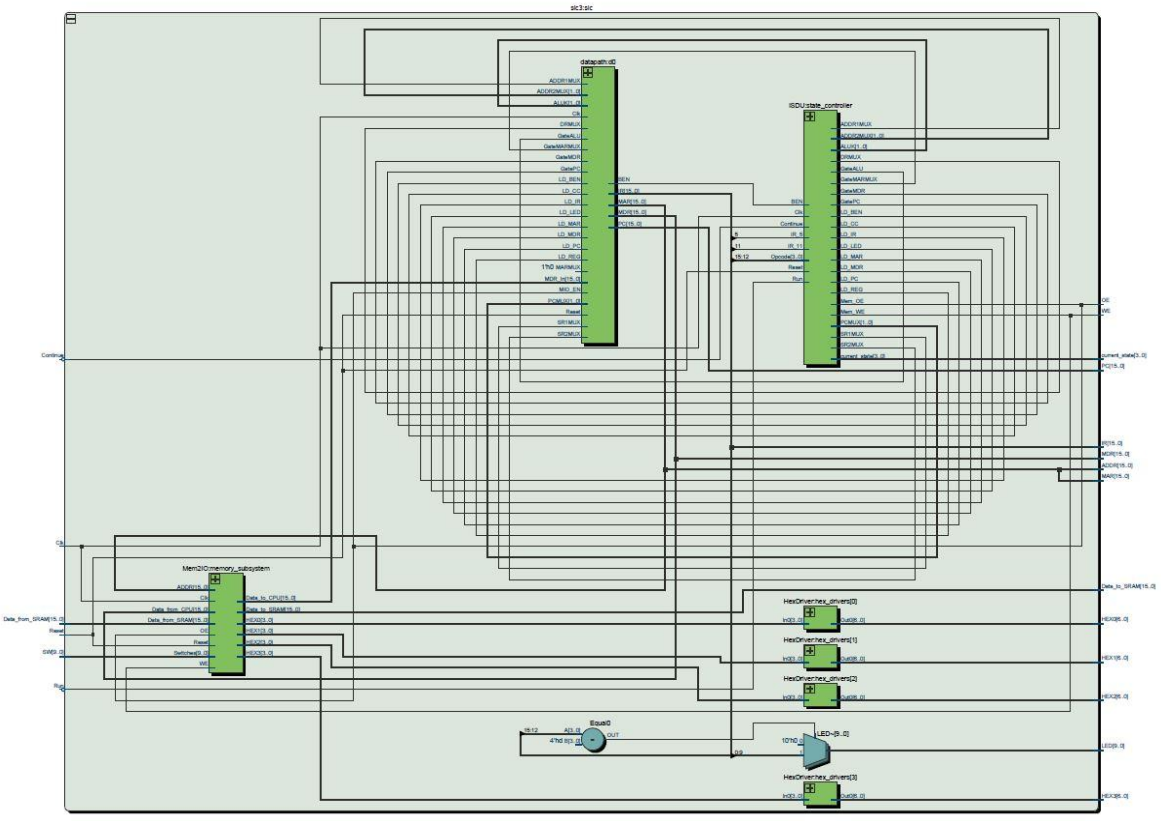
Instruction	**Instruction [15:0]**																**Operation**
ADD	0	0	0	1	DR			SR1		0	0	0	SR2				R(DR) ← R(SR1) + R(SR2)
ADDi	0	0	0	1	DR			SR		1	imm5						R(DR) ← R(SR) + SEXT(imm5)
AND	0	1	0	1	DR			SR1		0	0	0	SR2				R(DR) ← R(SR1) AND R(SR2)
ANDi	0	1	0	1	DR			SR		1	imm5						R(DR) ← R(SR) AND SEXT(imm5)
NOT	1	0	0	1	DR			SR		1	1	1	1	1	1		R(DR) ← NOT R(SR)
BR	0	0	0	0	n	z	p	PCOffset9									if ((nzp AND NZP) != 0) PC ← PC + SEXT(PCOffset9)
JMP	1	1	0	0	0	0	0	BaseR		0	0	0	0	0	0		PC ← R(BaseR)
JSR	0	1	0	0	1	PCOffset11											R(7) ← PC; PC ← PC + SEXT(PCOffset11)
LDR	0	1	1	0	DR			BaseR		Offset6							R(DR) ← M[R(BaseR)+SEXT(Offset6)]

STR	0	1	1	1	SR	BaseR	Offset6	M[R(BaseR) + SEXT(Offset6)] ← R(SR)
PSE	1	1	0	1	LedVect12			LEDs ← LedVect12; Wait on Continue

The first state of the Execute operation is determined by the opcode, as shown above. All the control signals are set to zero by default, and some are set to high during different Execute states. For example, in the first state of AND and ADD, specified control signals are SR2MUX = IR_5; GateALU = 1'b1; LD_REG = 1'b1; DRMUX = 1'b0; SR1MUX = 1'b1; LD_CC = 1'b1. The only difference is that ALUK = 2'b00 in ADD, while it is 2'b01 in AND. The next state will be the first state of Fetch, since ADD and AND only need one Execute state. All final Execute states will lead to the first state of Fetch to execute the next instruction, except the Pause instruction. After the final state of Pause, the FSM goes to PauseIR1 state, which will only go to PauseIR2 state at Continue press and then go to the first state of Fetch at Continue release. The LedVect12 in the Pause instruction is not passed to the ISDU. It is directly passed to the HexDrivers in the datapath and then passed to the on-board hex displays. In memory-related instructions such as LDR and STR, the single operation MDR ← M[MAR] / M[MAR] ← MDR is split into three states to ensure that the data from/to the memory is ready to be acquired/written. The first state of Fetch will be reached again after all states in Execute are completed so that the SLC-3 can run consecutive instructions stored in the on-chip memory.

Block Diagram of slc3.sv

The text is small and hard to read, but it is readable if zoomed in.



Written Description of all .sv modules

Module: reg_16.sv

Inputs: [15:0] D, Clk, Reset, Load

Outputs: [15:0] Data_Out

Description: This is a positive-edge triggered 16-bit register with asynchronous reset and synchronous load. When Load is high, data is loaded from D into the register on the positive edge of Clk.

Purpose: This module is used to create the register file that is used by SLC-3 operations to store values.

Module: reg_file.sv

Inputs: Clk, Reset, [15:0]Data_In, [2:0]DR, LD_REG, [2:0]SR1, [2:0]SR2

Outputs: [15:0]SR1_OUT, [15:0]SR2_OUT

Description: This is a positive-edge triggered register file containing eight 16-bit registers, R0-R7. It has asynchronous reset and synchronous load. DR, SR1, SR2 are 3-bit select signals to select from R0-R7. When LD_REG is high, data is loaded from Data_In to R(DR). SR1_OUT and SR2_OUT are connected to the output values of R(SR1) and R(SR2). These are asynchronous outputs, but the select signals SR1 and SR2 are synchronous.

Purpose: This module is used by operations of the SLC-3 to store operands, addresses or results.

Module: ALU.sv

Inputs: [15:0]A, [15:0]B, [1:0]ALUK

Outputs: [15:0]ALU_out

Description: This is the arithmetic logical unit for operations ADD, AND, and NOT. When ALUK is 2'b00, ALU_out is A + B; when ALUK is 2'b01, ALU_out is A AND B; when ALUK is 2'b10, ALU_out is NOT A; when ALUK is 2'b11, ALU_out is A.

Purpose: This module is used to perform arithmetic and logical calculations required by the ISA of SLC-3.

Module: datapath.sv

Inputs: Clk, Reset, LD_MAR, LD_MDR, LD_IR, LD_BEN, LD_CC, LD_REG, LD_PC, LD_LED, GatePC, GateMDR, GateALU, GateMARMUX, SR2MUX, ADDR1MUX, MARMUX, MIO_EN, DRMUX, SR1MUX, [1:0] PCMUX, [1:0] ADDR2MUX, [1:0] ALUK, [15:0] MDR_In

Outputs: BEN, [15:0] MAR, [15:0] MDR, [15:0] IR, [15:0] PC

Description: This is the datapath used by the SLC-3 for data operation. The datapath is controlled by the control unit, ISDU. It shares the same clock with the ISDU and the Mem2IO interface. It has asynchronous reset. The input [15:0] MDR_In comes from the memory interface, this is the data read from memory. All the other inputs are control signals that are outputs from the ISDU. The LD_* signals enable loading of the corresponding registers. Only one of the four Gate* signals can be high at a time. The Gate* signals select the source of the 16-bit data bus. SR2MUX selects whether to use SEXT(IR[4:0]) or SR2OUT as the input B of the ALU. ADDR1MUX selects whether to use PC or SR1OUT as one operand of the address adder. MARMUX was provided but was not used in the datapath. MIO_EN = 0 selects the bus as the input of the MDR, MIO_EN = 1 selects the data from memory as the input of the MDR. DRMUX selects the DR to the register file. DRMUX = 0 selects IR[11:9], while DRMUX = 1 selects 3'b111. SR1MUX selects the SR1 value, SR1MUX = 0 selects IR[11:9] and SR1MUX = 1 selects IR[8:6]. PCMUX selects the PC input value. PCMUX=2'b00 selects PC+1, 2'b01 selects data bus, 2'b10 selects the output of the address adder. ADDR2MUX selects another operand of the address adder. ADDR2MUX=2'b00 selects 16'h0000, 2'b01 selects SEXT(IR[5:0]), 2'b10 selects SEXT(IR[8:0]), 2'b11 selects SEXT(IR[10:0]). ALUK selects the type of operation of the ALU. ALUK=2'b00 selects ADD, 2'b01 selects AND, 2'b10 selects NOT, 2'b11 selects A. The datapath can achieve the expected operation at a specific FSM state according to the control signals.

Purpose: This module integrates all the smaller modules such as reg_file and ALU, and serves as a datapath where all the data operations take place. It is controlled by the ISDU through control signals, and it also sends feedbacks (IR value) to the ISDU for consecutive executions.

Module: ISDU.sv

Inputs: Clk, Reset, Run, Continue, [3:0] Opcode, IR_5, IR_11, BEN

Outputs: LD_MAR, LD_MDR, LD_IR, LD_BEN, LD_CC, LD_REG, LD_PC, LD_LED, GatePC, GateMDR, GateALU, GateMARMUX, [1:0] PCMUX, DRMUX, SR1MUX,

SR2MUX, ADDR1MUX, [1:0] ADDR2MUX, [1:0] ALUK, Mem_OE, Mem_WE, [3:0] current_state

Description & Purpose: this is written in the next section.

Module: Mem2IO.sv

Inputs: Clk, Reset, [15:0] ADDR, OE, WE, [9:0] 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 is the memory interface between the datapath and the memory. When OE = 1, Data_to_CPU will be updated to Data_from_SRAM, which is the data in the on-chip memory at address = ADDR. Otherwise, Data_to_CPU will be zero. When WE = 1, Data_to_SRAM will be connected to Data_from_CPU, and the data will be written to memory address = ADDR. A special function of this memory interface is that when ADDR = 16'hFFFF, it becomes the interface between the datapath and the on-board switches/hex displays, depending on OE/WE. It will write Data_from_CPU to {HEX3, HEX2, HEX1, HEX0} when WE = 1. When WE = 0 and OE = 1, it passes {6'b000000, Switches} to Data_to_CPU.

Purpose: This module is used as a memory interface that exchanges data between the datapath and the on-chip memory. It is also an interface that can read the input from switches and pass it to the datapath, or pass the data from the datapath to the on-board hex displays.

Module: slc3.sv

Inputs: [9:0] SW, Clk, Reset, Run, Continue, [15:0] Data_from_SRAM

Outputs: [9:0] LED, OE, WE, [6:0] HEX0, [6:0] HEX1, [6:0] HEX2, [6:0] HEX3, [15:0] ADDR, [15:0] Data_to_SRAM, [15:0] MAR, [15:0] MDR, [15:0] IR, [15:0] PC

Description: This is the SLC-3 module that creates instances of a datapath, an ISDU controller, and a Mem2IO memory interface. It connects the Mem2IO instance to the datapath, and connects the control signals from the ISDU to the datapath. Data_from_SRAM and Data_to_SRAM are connected to Mem2IO, and will be connected to the on-chip memory in the top-level entity. Four HexDrivers are also used to output the value IR[11:0] of the Pause instruction to the on-board hex displays.

Purpose: This module connects the ISDU, the datapath, and Mem2IO together.

Module: InstantiateRam.sv

Inputs: Reset, Clk

Outputs: [15:0] ADDR, [15:0] data, wren

Description: This module writes predefined instructions to the on-chip memory starting at address 0x0000 every time the board is programmed or reset signal is present. It has asynchronous reset. It writes 156 lines of 16-bit instructions to the memory. The output wren is 1 when it is writing to the memory, and 0 otherwise. ADDR is the current address being written at, and data is the current data being written.

Purpose: This module is used to initialize the test programs in the on-chip memory that the SLC-3 will run.

Module: test_memory.sv

Inputs: Clk, Reset, [15:0] data, [9:0] address, rden, wren

Outputs: [15:0] readout

Description: This is a test memory that works as well as the actual memory for testing. It has asynchronous reset. On the positive edge of the clock, data is written to the address provided by the input address if rden = 1. If wren = 1, the output readout is set to the data at address = input address.

Purpose: This module is used with the provided memory_contents.sv to work like the actual on-chip memory with InstantiateRam for testing purposes.

Module: slc3_testtop.sv & slc3_sramtop.sv

Inputs: [9:0] SW, Clk, Run, Continue

Outputs: [9:0] LED, [6:0] HEX0, [6:0] HEX1, [6:0] HEX2, [6:0] HEX3, [15:0] MDR, [15:0] MAR, [15:0] IR, [15:0] PC

Description: These are the top-level entities for compilation. In slc3_testtop, the slc3 instance is connected to a test_memory module. In slc3_sramtop, the slc3 instance is connected to the actual on-chip memory. Synchronizers are also created to avoid metastability.

Purpose: These modules link the slc3 module and the memory together so that the program can be compiled and uploaded to the FPGA board.

Module: synchronizers.sv

Inputs: Clk, d

Outputs: q

Description: This is a 1-bit flip-flop that synchronizes the asynchronous inputs such as the switches and the keys. The output q is only updated to d at positive clock edge.

Purpose: This module eliminates metastability so that asynchronous inputs will not violate flip-flop setup and hold times.

Module: HexDriver.sv

Inputs: [3:0] In0

Outputs: [6:0] Out0

Description: This is a driver that maps the 4-bit value In0 to a 7-bit value that can be written to the on-board hex displays.

Purpose: This module is used to visualize data.

Description of the operation of the ISDU

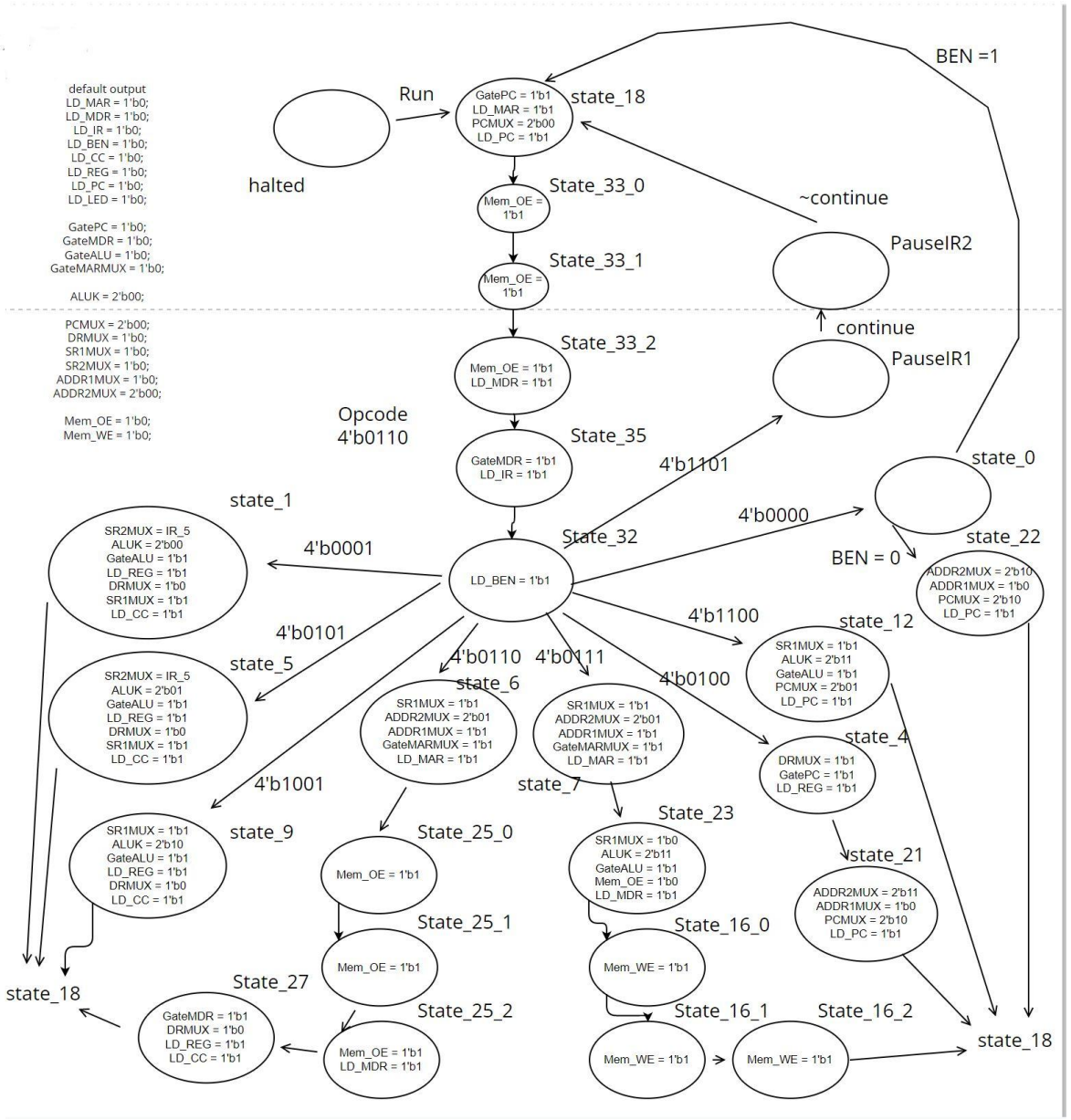
ISDU is a module that serves as the control unit within the LC3 processor. It's a finite state Meanly machine that controls which state of the LC3 is at and outputs the corresponding control signals to the registers unit, the gates, the loading signals and all kinds of muxes. It determines the next state according to its current state and the input.

It has Clk, Reset, Run, Continue, [3:0] Opcode, IR_5, IR_11, BEN as inputs, and LD_MAR, LD_MDR, LD_IR, LD_BEN, LD_CC, LD_REG, LD_PC, LD_LED, GatePC, GateMDR, GateALU, GateMARMUX, [1:0] PCMUX, DRMUX, SR1MUX, SR2MUX, ADDR1MUX, [1:0] ADDR2MUX, [1:0] ALUK, Mem_OE, Mem_WE, [3:0] current_state as outputs.

Take the ADD operation as an example. By default, all outputs are 1'b0 besides ALUK, which is 2'b00. After the run button is pressed, the machine will entere state_18 from the halted state. In state_18, the outputs are GatePC = 1'b1; LD_MAR = 1'b1; PCMUX = 2'b00; LD_PC = 1'b1, so the GatePC will open, and the PC value will enter the bus and loaded to MAR by LD.MAR, and PC will be incremented by 1 since PCMUX is 2'b00. Then the program entered state_33_1, whose outputs are Mem_OE = 1'b1, LD_MDR = 1'b1, so that the MDR can get the content in SRAM with a memory address just loaded by the MAR

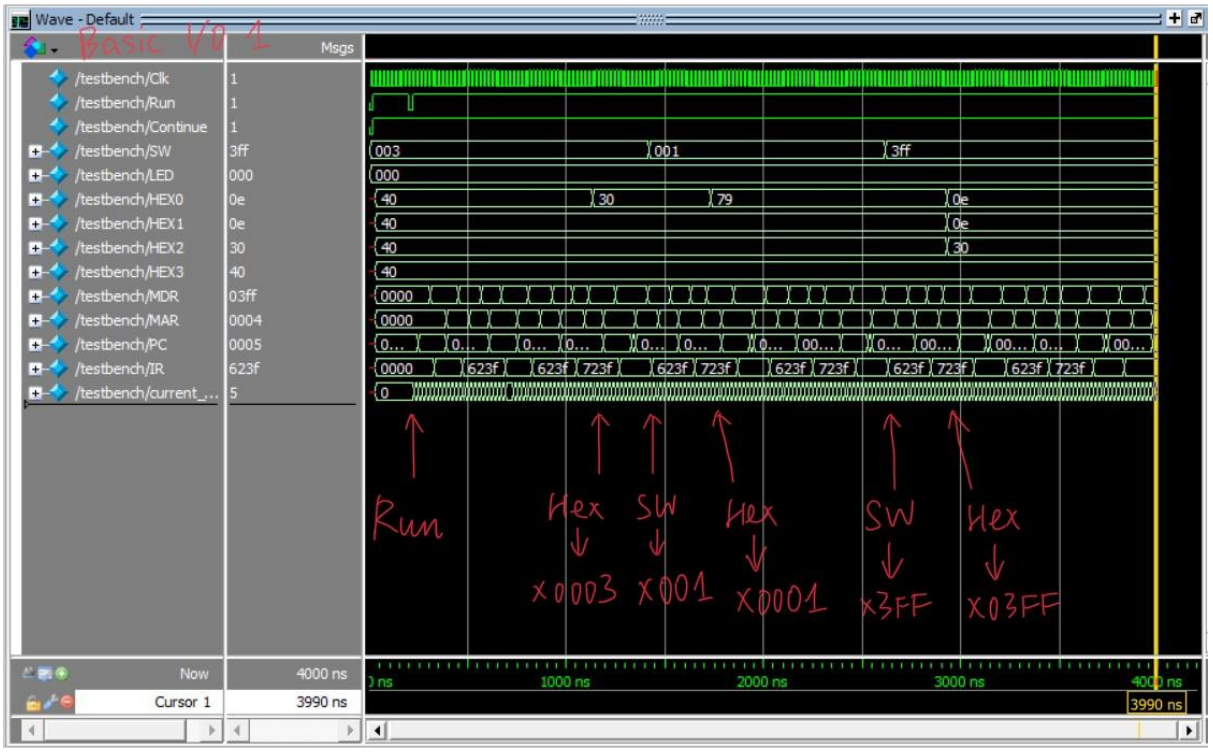
register. State_33 is splitted into state_33_0, state_33_1, state_33_2 because this gives the processor enough time to make sure the ready signal is 1. Then the processor enters the state_36, and the content in MDR will be given to IR by GateMDR = 1'b1, LD_IR = 1'b1, which sends the MDR signal to the bus and calls the IR to load the value in the bus. Then in state_32, the ISDU control unit will call the BEN to load the new nzp, and then decide which state it will transit to according to the opcode, [15:12]IR. In the case of ADD, the next state is state_01, in which the output control signals are SR2MUX = IR_5, ALUK = 2'b00, GateALU = 1'b1, LD_REG = 1'b1, DRMUX = 1'b0, SR1MUX = 1'b1, LD_CC = 1'b1. The SR2MUX will output the SR2, and ALU unit will perform an addition between SR1 and SR2 since its ALUK is 2'b00, and the GateALU will be open, causing the result of addition drives the bus, the LD_REG = 1'b1 enables the register unit to load the result of the addition, DRMUX = 1'b0 load the result from the bus to the DR specified by the [11:9]IR. SR1MUX = 1'b1 select SR1 to be the register specified by [8:6]IR. LD_CC = 1b'b1 updates the nzp. At last, it will return to state_18.

State Diagram of ISDU

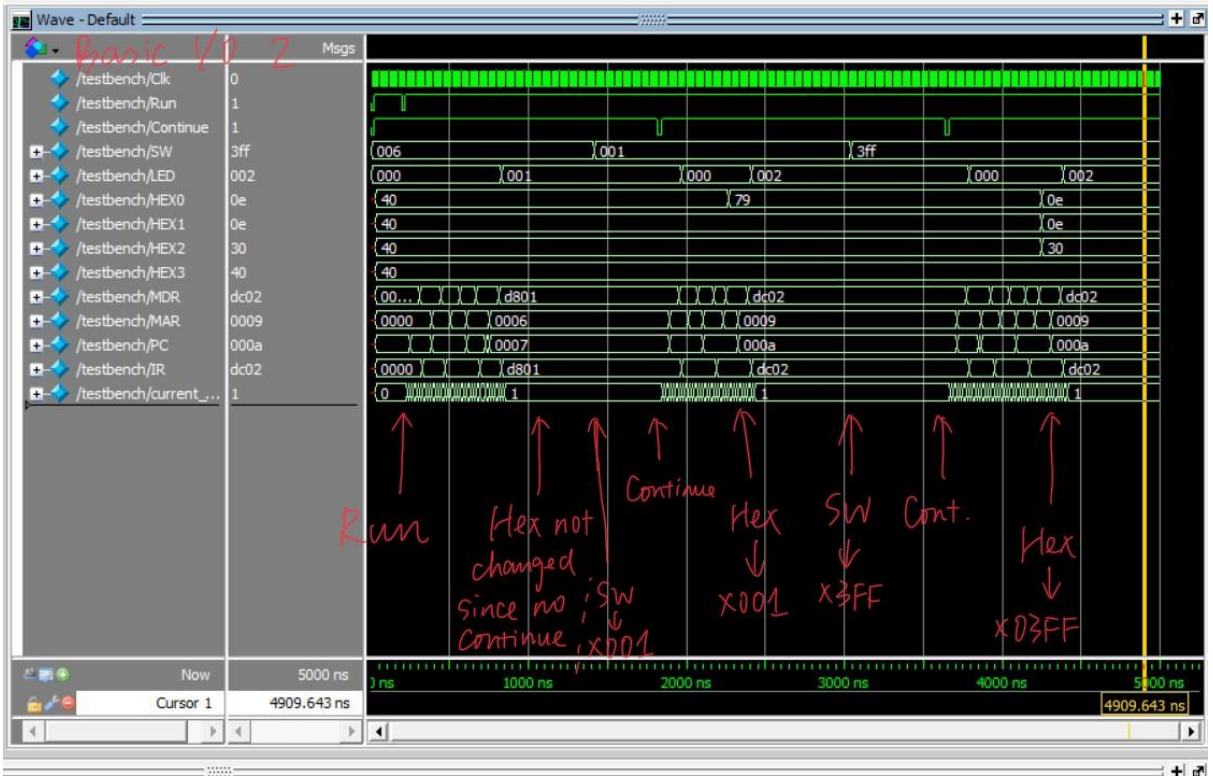


Simulations of SLC-3 Instructions

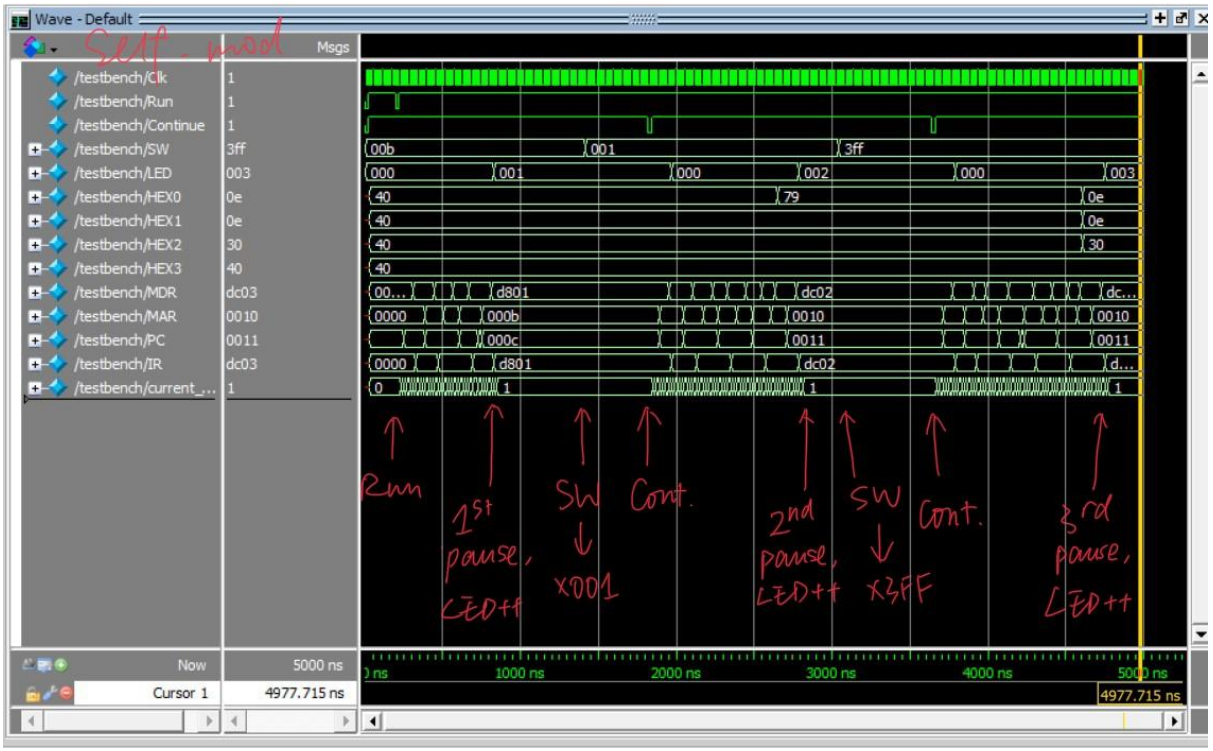
Basic I/O Test 1:



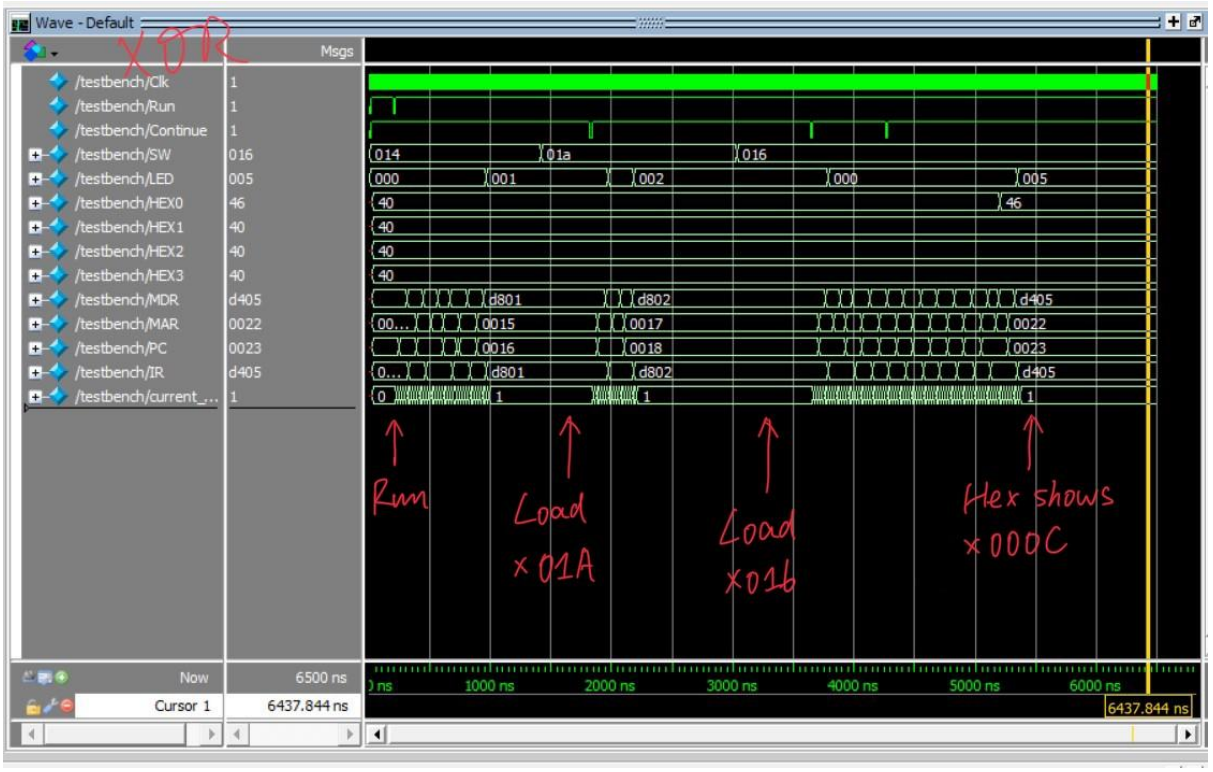
Basic I/O Test 2:



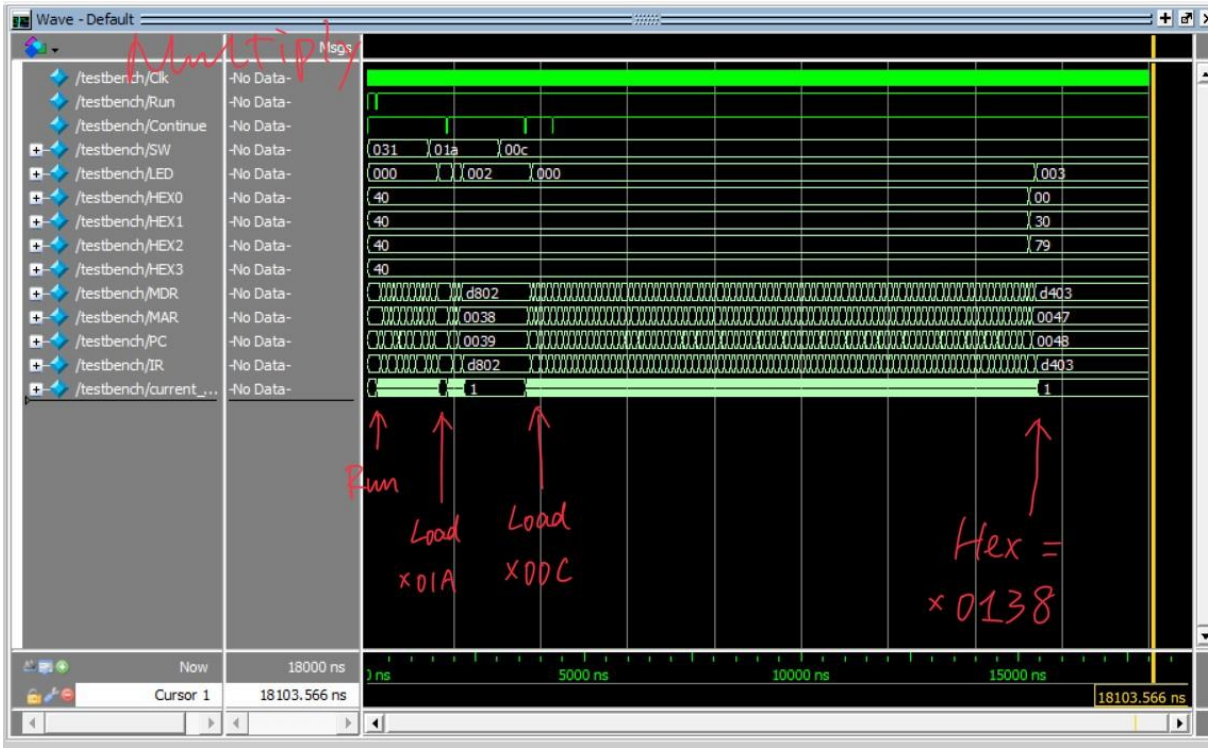
Self-Modifying Code:



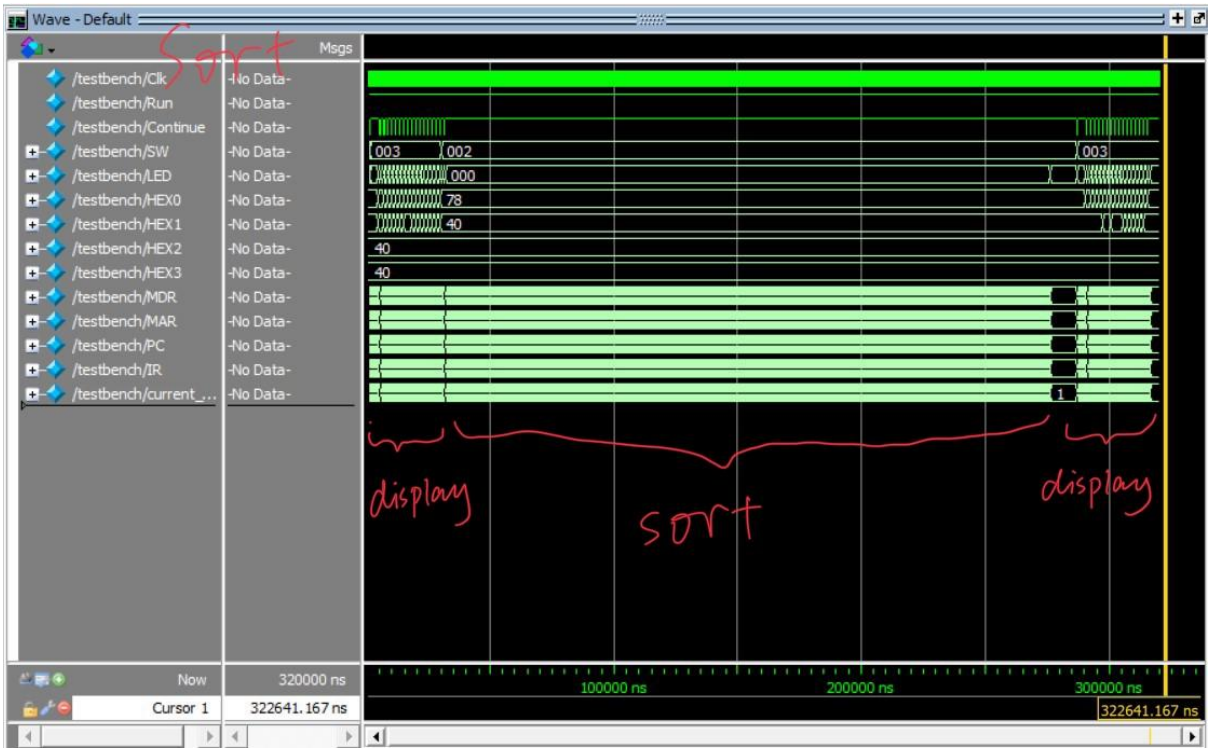
XOR: (0x1C XOR 0x16 = 0x0C)



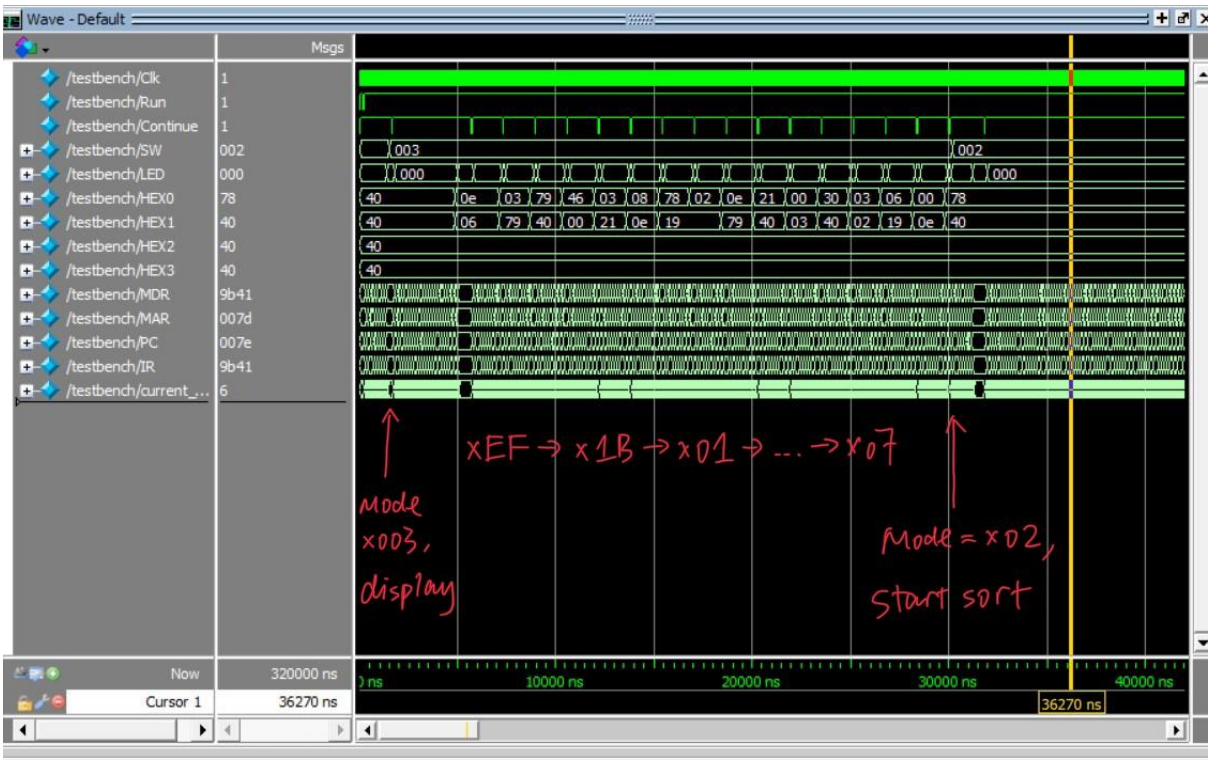
Multiply: (0x1A * 0x0C = 0d26 * 0d12 = 0d312 = 0x138)



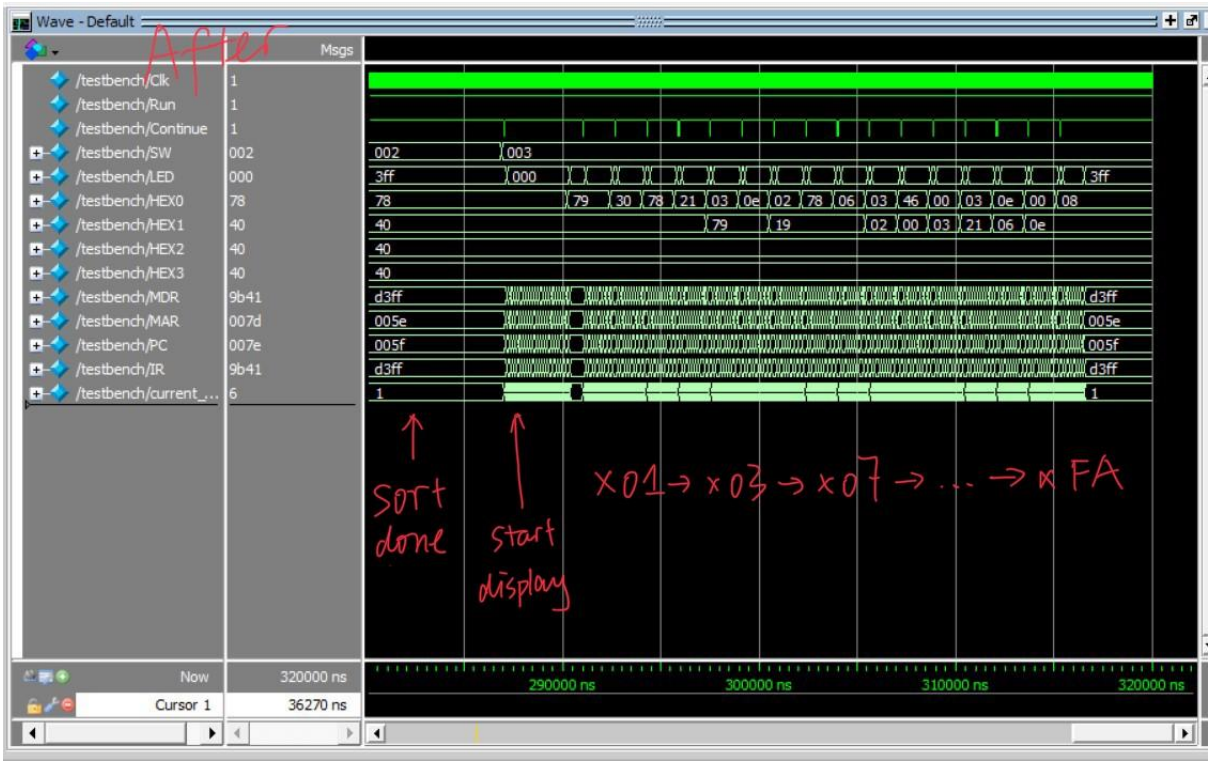
Sort: The whole waveform is too dense to see the values, so three waveforms are included. The first one is the whole waveform, the second one is before sort display, the third one is after sort display.



Before sort:



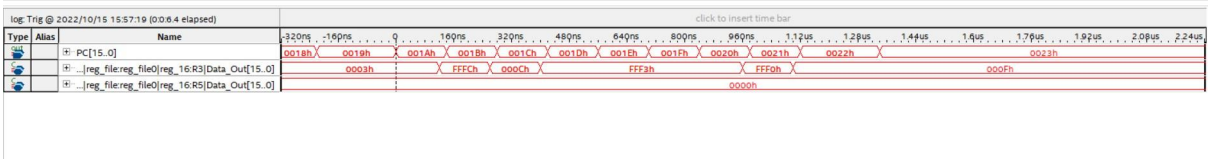
After sort:



SignalTap Simulations of XOR, Multiply and Sort

XOR

Following is the SignalTap trace of the XOR function. The two operands are 0x0003 and 0x000C, so the result should be 0x000F. The capture begins at PC = 0x001A, which is when the SLC is executing the first XOR instruction: NOT R3, R1.

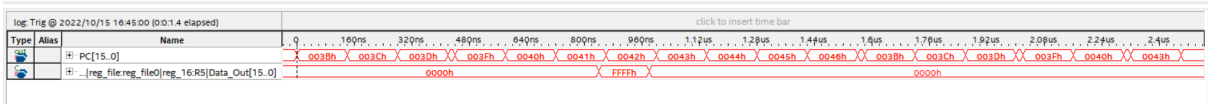


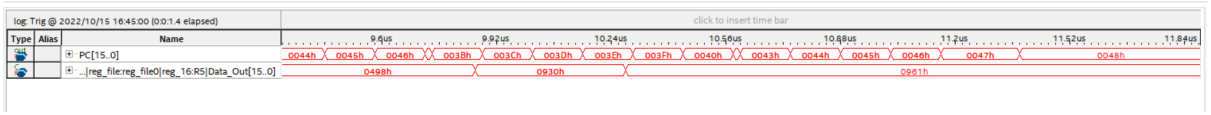
The XOR function ends just before PC = 0x0022, which is when the SLC ends the last XOR operation NOT R3, R3 and starts the instruction STR R3, R0, outHEX. This can also be verified by looking at the R3 value since the result is stored in R3. R3 turns to the desired output 0x000F just before PC = 0x0022.

This gives 0x0022 - 0x001A = **8 instructions**. According to the trace, it takes 1,120 ns to complete these instructions. The number of clock cycles is 1120ns / 20ns = **56 cycles**. This gives 8/56 * 50MHz = **7.143 MIPS**.

Multiply

The “core algorithm” starts at PC = 0x003B, which is when ADD R5, R5, R5 is executed first, and ends at PC = 0x0048, which is when the checkpoint 3 pause instruction is executed. Following is the trace of 0x0031 * 0x0031 = 0x0961. The trace is too long, therefore only the start and the end are included.

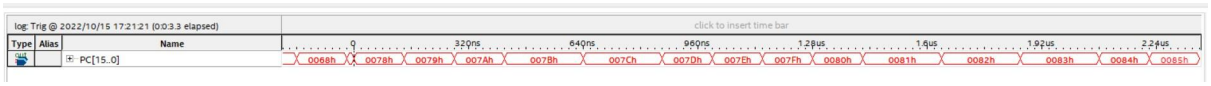




To calculate the number of instructions, we didn’t calculate by hand because some instructions are conditional and the result may be inaccurate if all conditional instructions are taken into account even if they were not executed. Instead, the SignalTap waveform was saved to a .csv file, and a Python script was used to process the data and calculate the number of instructions according to the PC value. The PC values are parsed into a list and the list is iterated. The current value is recorded, and a counter is incremented each time a different value pops up. As a result, there are a total of **97 instructions**. According to the trace, it took 11,380 ns to complete these instructions. The number of clock cycles is 11380/20 = **569 cycles**. This gives 97/569 * 50MHz = **8.524 MIPS**.

Sort

The “core algorithm” starts at PC = 0x0078, where ADD R1, R0, -16 is executed. It ends just before PC = 0x0089, where RET is executed. Following is the trace of sort executed on the provided memory values.



The number of instructions is obtained using the same method as in the multiply program. It has **1,632 instructions**. According to the trace, until PC = 0x0089, it took 242,680 ns to run. The number of clock cycles is 242,680/20 = **12,134 cycles**. This gives 1,632/12,134 * 50MHz = **6.725 MIPS**.

Average

The average MIPS is **7.464 MIPS**.

The multiply program is the fastest because it has a large number of branch instructions which are fast, and the sort program is the slowest because it has more LDR operations which involve memory so there are more states.

Post-Lab Questions

Following is a table containing the information of the SLC circuit.

LUT	1384
DSP	0
BRAM	24,704 bits
Flip-Flop	1,019
Max Frequency	63.58 MHz
Static Power	90.06 mW
Dynamic Power	15.38 mW
Total Power	124.54 mW

During 5.1, we lost one demo point for not being able to reset correctly. It was because the number of wait states was only 1, which was too small for the memory to be ready. We changed the number of wait states to 2 and the bug was fixed. During 5.2, we only

encountered one bug. We found that the LDR function was working incorrectly by running the test programs provided, and we debugged with SignalTap to observe MAR and register file values. It turned out that an unintended load was performed by the register. Then, we located the problem: the default load signals of the register file were set to don't care, but they should be zero when not used to avoid data loss. We changed the default load signals to zero, and the bug was fixed.

MEM2IO is the memory interface that exchanges data between the datapath and the memory. It can also read the switch's input and pass the input to the datapath or pass the data from datapath to hex display on the board. When OE = 1, Data_to_CPU will be updated to Data_from_SRAM, which is the data in the on-chip memory at address = ADDR. Otherwise, Data_to_CPU will be zero. When WE = 1, Data_to_SRAM will be connected to Data_from_CPU, and the data will be written to memory address = ADDR. A special function of this memory interface is that when ADDR = 16'hFFFF, it becomes the interface between the datapath and the on-board switches/hex displays, depending on OE/WE.

BR is conditional depending on NZP and nzp, whereas JMP is unconditional. BR uses a 9-bit offset, and adds it to the current PC + 1, whereas the JMP changes the PC to the content of the chosen base R.

The R signal is used to indicate that the system has finished loading the data from the SRAM to MDR, or input the data from the MDR to the SRAM. It occurs at state_25, state_16 and state_33. We compensate for the lack of such a ready signal by repeating the state three times. For example, state_25 is splitted into state_25_0, state_25_1, state_25_2, and this gives it enough time for the data to be loaded or inputted. This implies that the memory ready will be high within 3 clock cycles.

Conclusion

Overall, we were able to successfully implement all the functionalities required for the SLC-3. As discussed above, we encountered some bugs that made part of the design not able to work correctly, but we finally managed to fix them. There was a sentence in the Lab 5 description that said we did not need to change ISDU during the first week, but we actually needed to add a wait state for the memory read based on our implementation. It may be better if the description says it is possible that ISDU needs to be changed.