ECE385 Spring 2023 – Lab 5 Report

Ziyuan Chen, Weijie Liang | ziyuanc3, weijiel4

All caps in Times New Roman (e.g., ADD*) are operations. All caps in Consolas (e.g.,* ADD*) are states.*

Introduction

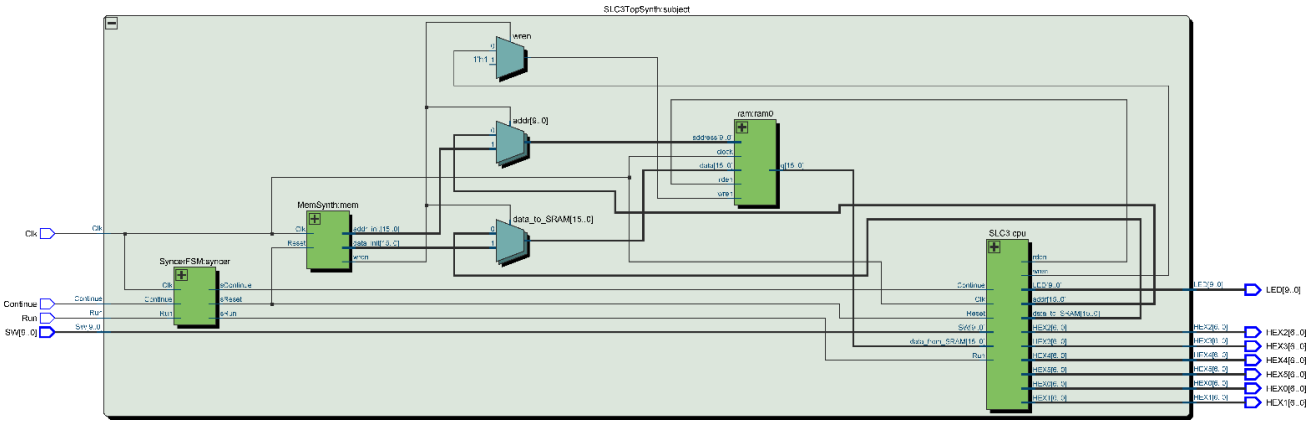
In this lab, we build a simplified SLC-3.2 Microprocessor with von Neumann architecture, 16-bit bus, and an instruction set including Add, And, Not, Load and Store (register addressing mode), Branch, Jump, Jump to Subroutine, and Pause. Starting from the user-specified Program Counter (PC) value, it executes instructions pre-loaded into the on-chip memory and writes the results back to the memory. This results in its capability to run self-modifying code. The user inputs values from the switches and reads output from the HEX digits.

Operations of the SLC-3 Processor

All the instruction cycles start with the FETCH and DECODE phases where the processor reads the instruction and determines its actions in the EXECUTE phase based on the opcode. Note that the notions of FETCH OPERAND and STORE RESULT are simplified into the EXECUTE phase which refers to the branches below DECODE.

- FETCH – 5 clock cycles. PC is simultaneously loaded into MAR and incremented. Reading from memory takes **3 cycles** (input buffer, query, output buffer). Instruction in MDR is transferred into IR In the last cycle.
- DECODE – 1 clock cycle. Assisted by an embedded ROM in the *original* LC-3 (whose principle resembles the FPGA LUT). Concurrently computes CC.
- EXEC. ADD, AND, NOT – 1 clock cycle. Computing the result and updating the Register File are performed simultaneously.
- EXEC. LDR – 5 clock cycles. It takes one cycle to compute the address. Note that the base register and 6-bit offset are not summed in the ALU but in a dedicated MAR adder. Exporting the memory content to DR takes another cycle.
- EXEC. STR – 3 clock cycles. Resembles LDR except that MDR is written into instead of read from, and that memory access only takes **1 cycle** since there is no need to wait for the MDR; we can move on and leave the work to RAM.

- EXEC. JMP – 1 clock cycle. Base register is loaded into PC.
- EXEC. JSR – 2 clock cycles. PC is backed up in R7 before being incremented by the 11-bit offset hardcoded into the instruction.
- EXEC. BR – 2 clock cycles. Checking the condition takes a dedicated cycle since BR is only allocated one opcode of `0000`, even though *no control signal is set in this cycle*. If true, PC is incremented by a 9-bit offset (shorter than that in JSR).
- EXEC. PSR – depends on the user. Immediately enters state `PSR1` after DECODE. The user can alter the switches or read from HEX at this point. Pressing the `Continue` key moves the ISDU into `PSR2`. Releasing the key moves it into `FCH1`.



Top-level block diagram of SLC-3, generated by the RTL Viewer in Quartus Prime 18.1 Lite

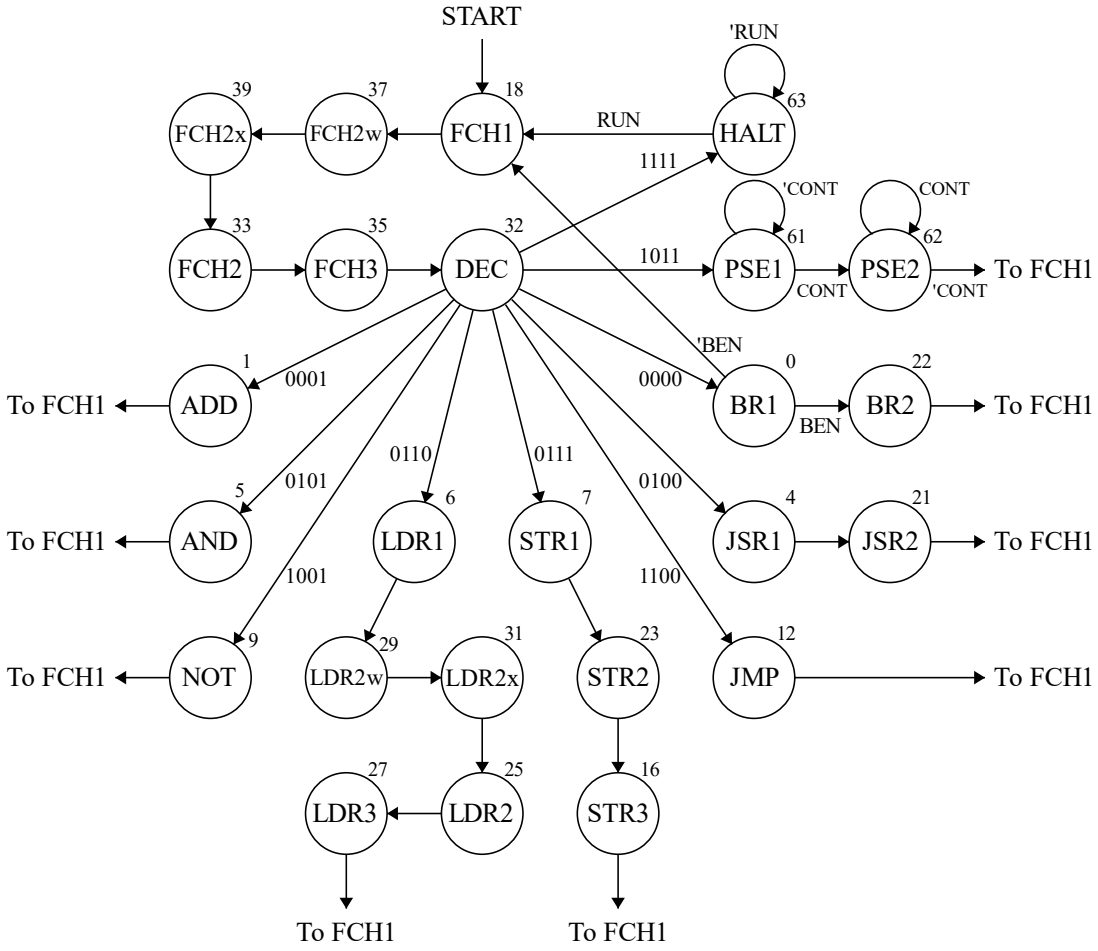
Operations of the ISDU Control Unit

The Indexed Service Data Unit (ISDU) serves as a finite state machine that orchestrates all of SLC-3's operations. Aside from the clock, it accepts the `Run` and `Continue` keys and parts of the Instruction Register (namely `IR[15:12]`, the opcode, and `IR[5]`, the immediate number selector in algorithmic instructions) as input and outputs the control signals (namely 8 `LD`s, 4 `Gate`s, 7 `MUX`es, `ALUK`, `rden`, and `wren`). The `LD` signals specify data destination, the `Gate`s control the source, and the `MUX`es determine the flow.

Next state is affected by keys and IR components. The key signals are active low and synchronized with the clock. However, instead of using the regular `Syncer`s, we implement a `SyncerFSM` to override the order of releasing keys: the FSM can only move back to `IDLE` only when neither key is pressed. Shown below are the TTL expressions and control signals for each state (a.k.a. ROM content in the *original* LC-3), along with a state transition diagram.

State	TTL Expression	State	TTL Expression
ADD	DR ← SR1 + OP2, **setCC**	LDR1	MAR ← BaseR + Offset6
AND	DR ← SR1 & OP2, **setCC**	STR1	MAR ← BaseR + Offset6
NOT	DR ← ~ SR, **setCC**	JSR2	PC ← PC + Offset11
LDR3	DR ← MDR, **setCC**	BR2	PC ← PC + Offset9
FCH3	IR ← MDR	FCH2* LDR2*	MDR ← M[MAR]
JMP	PC ← BaseR	STR3	M[MAR] ← MDR
JSR1	R7 ← PC	PSE1	LED ← LEDVect12, [CONT]
FCH1	MAR ← PC, PC ← PC + 1	PSE2	[CONT]
STR2	MDR ← SR	HALT	[RUN]
DEC	BEN ← (Logic)	BR1	[BEN]

State	LD_IR	LD_PC	LD_MAR	LD_MDR	LD_REG	LD_CC	LD_BEN	LD_LED	GatePC	GateMARMUX	GateMDR	GateALU	PCMUX	MARMUX	DRMUX	SR1MUX	SR2MUX	ADDR1MUX	ADDR2MUX	ALUK	rden	wren
FCH1		1	1						1				00									
FCH2w																					1	
FCH2x																					1	
FCH2				1																	1	
FCH3	1										1											
DEC							1															
ADD					1	1						1			0	1	IR5			00		
AND					1	1						1			0	1	IR5			01		
NOT					1	1						1			0	1				10		
LDR1			1							1						1		1	01			
LDR2w																					1	
LDR2x																					1	
LDR2				1																	1	
LDR3					1	1					1				0							
STR1			1							1						1		1	01			
STR2				1								1				0				11		
STR3																						1
JMP		1											10			1		1	00			
JSR1					1				1						1							
JSR2		1											10					0	11			
BR1																						
BR2		1											10					0	10			
PSE1								1		1				1								
PSE2																						
HALT																						



SystemVerilog Modules – *Exported signals for simulated inspection are not included*

New Filename	Old Filename
`top.sv`	–
`topsim.sv`	`slc3_testtop.sv`
`topsynth.sv`	`slc3.sramtop.sv`
`slc3.sv`	`slc3.sv`
`slc3isdu.sv`	`ISDU.sv`
`slc3pkg.sv`	`SLC3_2.sv`
`mem2io.sv`	`Mem2IO.sv`
`memsim.sv`	`test_memory.sv` `memory_contents.sv`
`memsynth.sv`	`Instantiateram.sv`
`utils.sv`	`synchronizers.sv`

The macro `{TOP_INTERFACE}` is defined as

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

Outputs: `HEX*[6:0] LED[9:0]`

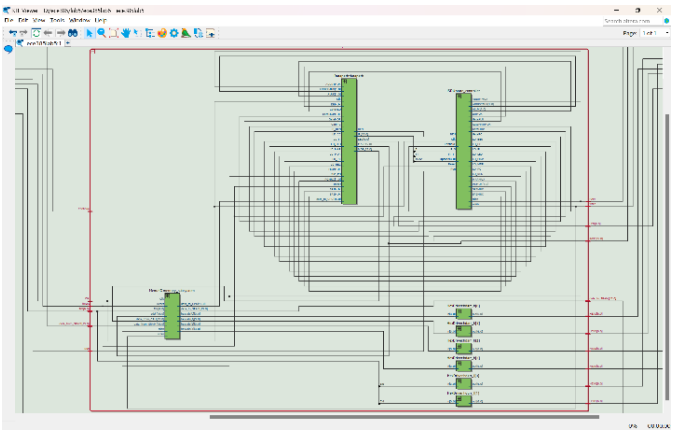
SLC3Top*	**Inputs:**	{TOP_INTERFACE}
	Outputs:	{TOP_INTERFACE}
	Description:	Includes SLC3, MemSim or MemSynth for initialization, peripheral SyncerFSM, and an additional ram *for synthesis*.
	Purpose:	Top-level design entity that envelopes the system.
SLC3	**Inputs:**	{TOP_INTERFACE} DataFromSRAM[15:0] Reset
	Outputs:	{TOP_INTERFACE} DataToSRAM[15:0] addr *en
	Description:	Includes Datapath, ISDU, Mem2IO, and HexDriver. Hosts global registers like IR, PC, MAR, MDR, and RegFile. • Also accepts the calculated Reset from SyncerFSM • Hosts BEN (for ISDU), but CC is hosted in Datapath • rden and wren are exported to SLC3Top*
	Purpose:	Wraps up the processor and provides memory interface.
Datapath	**Inputs:**	DataToCPU[15:0] LD* Gate* *MUX ALUK *en Reset Clk
	Outputs:	LED[9:0] BEN
	Description:	Includes global register *drivers*, ALU, MUXes, and BUS. Also accepts Reset to clear the registers when requested,
	Purpose:	Core datapath that drives the registers.
ISDU	**Inputs:**	IR[15:12] IR[5] BEN Run Continue Reset Clk
	Outputs:	LD* Gate* *MUX ALUK *en
	Description:	Implements the FSM that tracks states and provides signals hardcoded in memory. BEN is calculated in Datapath and hosted in SLC3 and. Reset comes from SLC3TOP*.
	Purpose:	Control unit that orchestrates the instruction cycles.
MemSim	**Inputs:**	addr[9:0] data[15:0] rden wren Reset Clk
	Outputs:	readout[15:0]
	Description:	Simulated MemSynth equivalent that functions *like* a RAM. Embeds a MemSimParser that handles memory requests.

`MemSynth`	**Inputs:**	`Reset Clk`
	Outputs:	`addr_init[15:0] data_init[15:0] wren`
	Description:	Initializes `ram` (created with Intel FPGA IP for RAM) with predefined contents. `wren` is asserted only at `mem_write` phase in the internal FSM.
`Mem2IO`	**Inputs:**	`DataFromCPU[15:0] DataFromSRAM[15:0] addr[15:0] SW[9:0] rden wren Reset Clk`
	Outputs:	`DataToCPU[15:0] DataToSRAM[15:0] [4]hexvals[3:0]`
	Description:	Inspects query address and overrides data flow from/to CPU and SRAM by `SW` or `LED` when requested, usually before or after the PAUSE instructions.
	Purpose:	Redirects memory access at `0xFF` to external I/O devices.
`Reg(WIDTH)`	**Inputs:**	`D[WIDTH-1:0] LD Reset Clk`
	Outputs:	`Q[WIDTH-1:0]`
	Description:	*Synchronously* loads `D` (`LD` asserted) or `0` (`Reset` asserted) into `Q` at positive edges of `Clk`. `Reset` overrides `LD`.
`RegFile`	**Inputs:**	`in[15:0] SR*[2:0] DR[2:0] LD_REG Reset Clk`
	Outputs:	`out*[15:0] [8]regs[15:0]`
	Description:	Packed version for eight 16-bit registers. `regs[SR*]` is assigned to `out*`. This is a *driver* that has no internal flip-flops.
`MUX*(WIDTH)`	**Inputs:**	`in*[WIDTH-1:0] S[n:0]`
	Outputs:	`out[WIDTH-1:0]`
	Description:	Basic multiplexer.
`ALU`	**Inputs:**	`A[15:0] B[15:0] ALUK[1:0]`
	Outputs:	`S[15:0]`
	Description:	Multifunctional arithmetic unit that outputs `A + B` at `ALUK` = `00`, `A & B` at `01`, `~A` at `10`, and `A` at `11`.

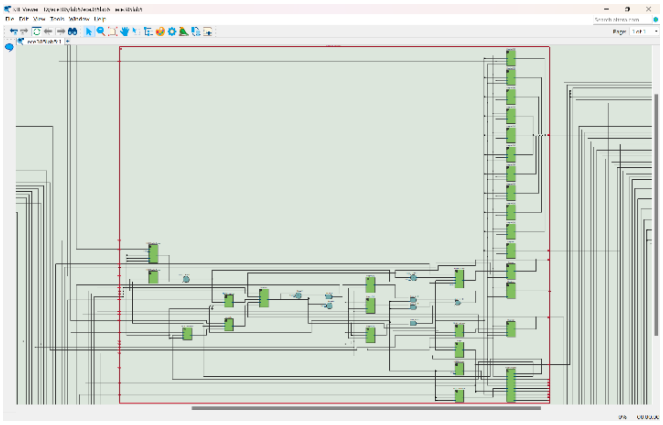
`Bus` **Inputs:** `in*[15:0] gates[3:0]`

Outputs: `out[15:0]`

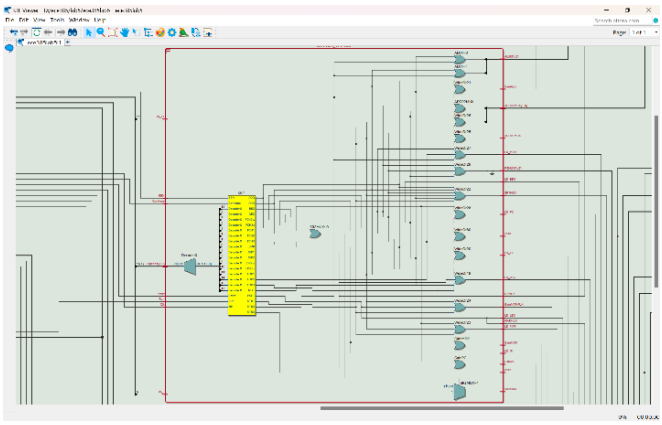
Description: MUX-based equivalent to four packaged three-state buffers. `gates` is one-hot encoded (multiple `1` outputs `16'bX`).



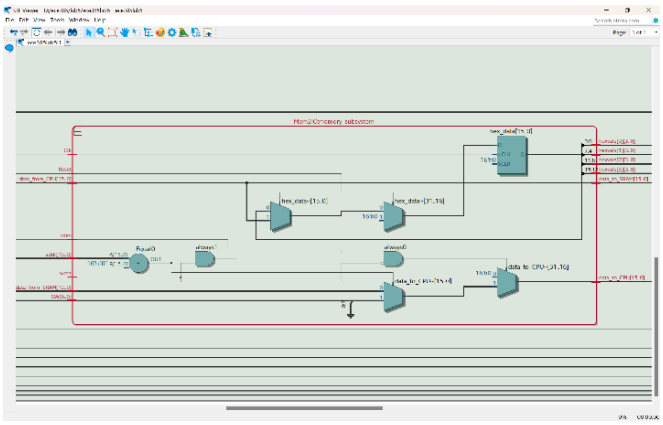
`SLC3` *Wrapper*



`Datapath`



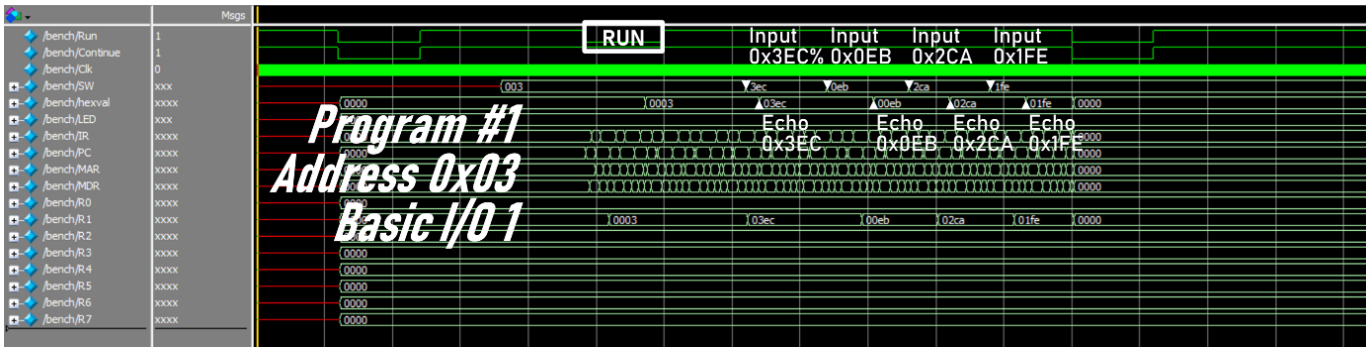
`ISDU`

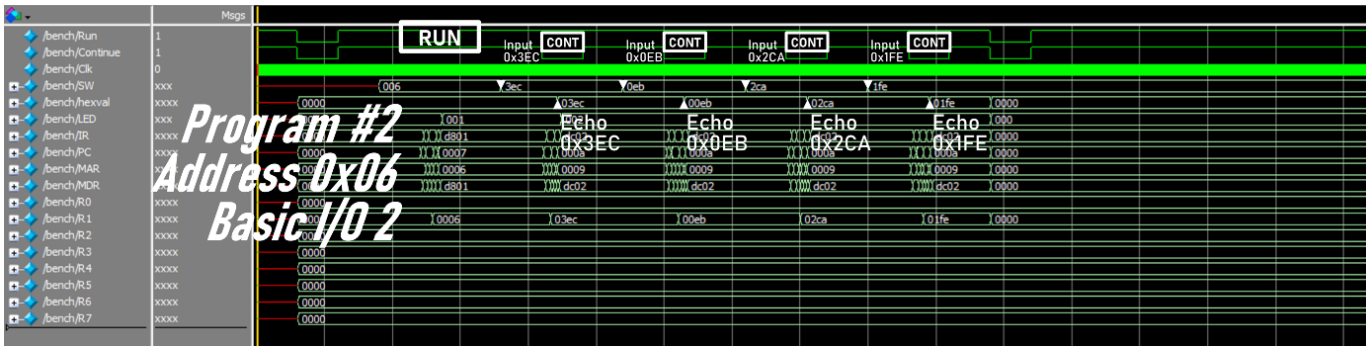


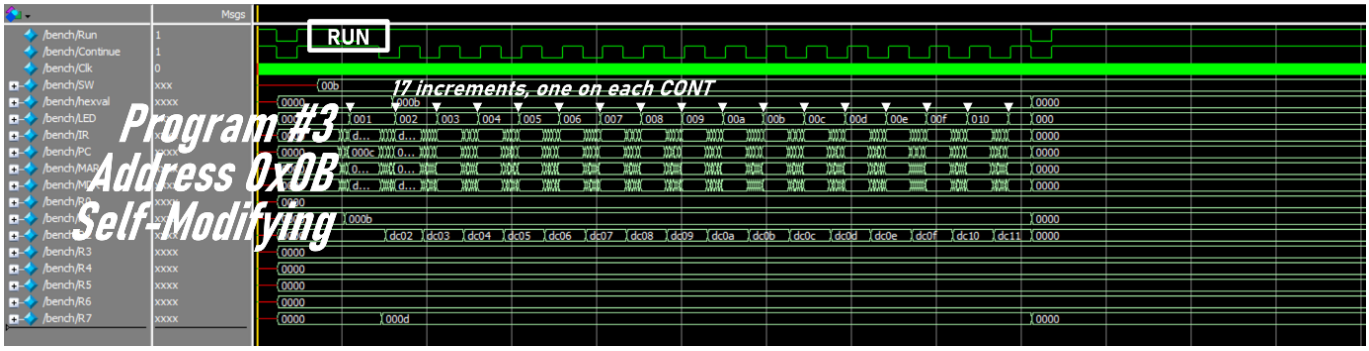
`Mem2IO`

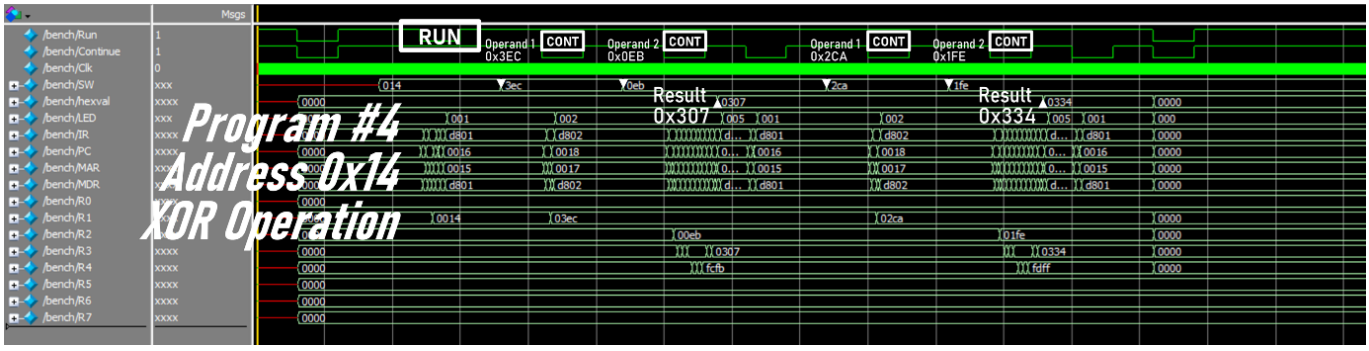
Expanded RTL diagrams of individual modules below `SLC3`

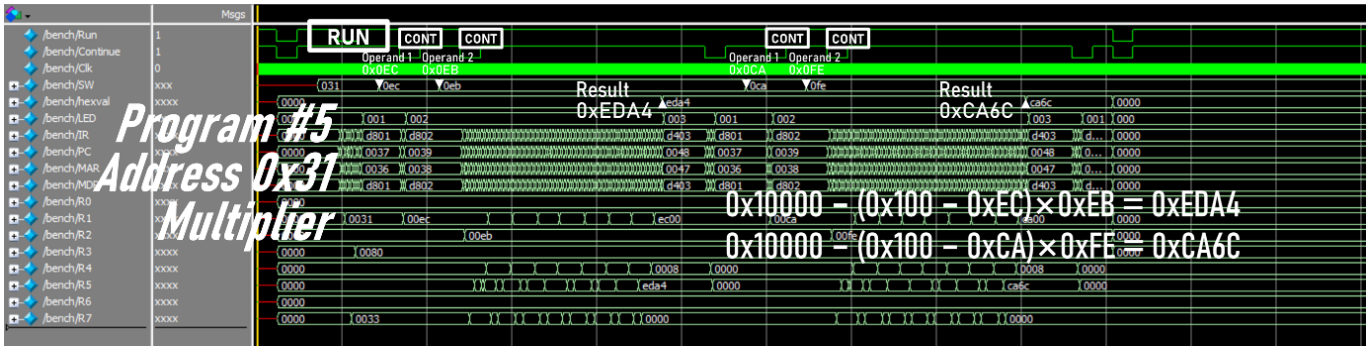
Simulation Traces – *using Lumetta's* `ECEB CAFE` *dataset from ECE120 practice exams*

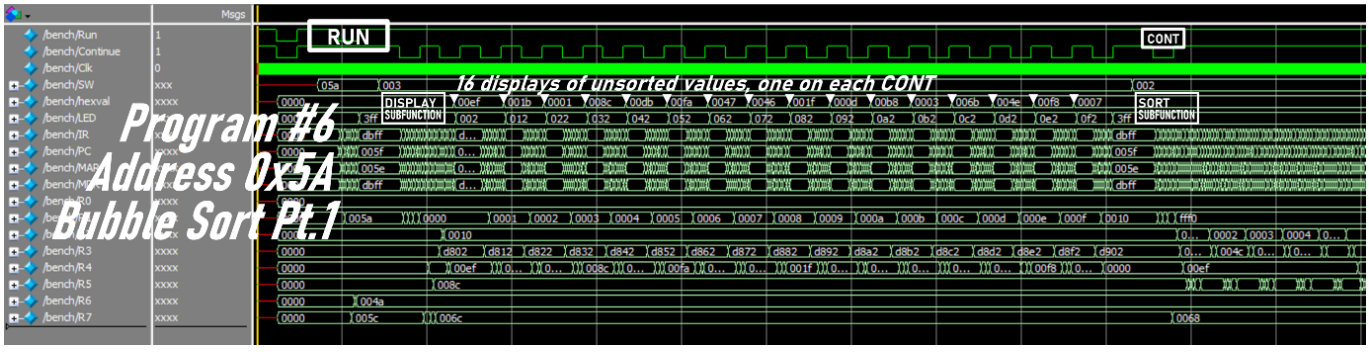


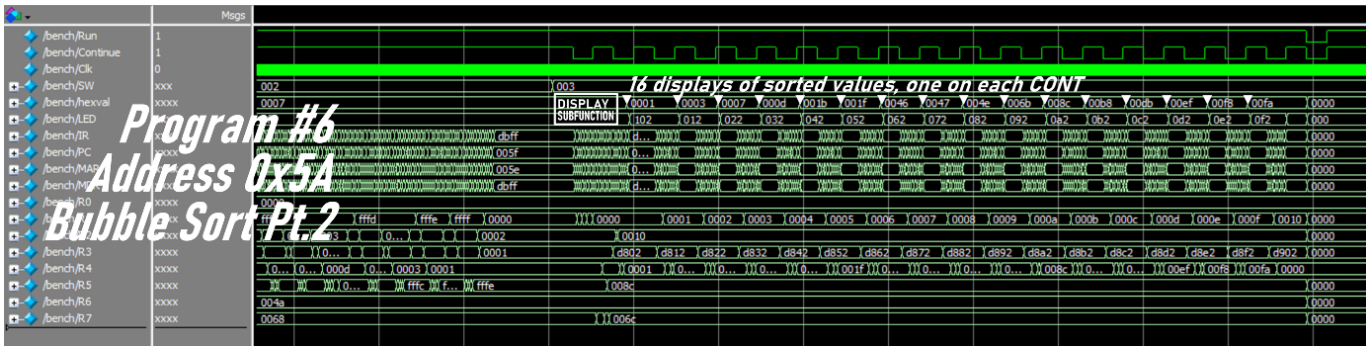

Program #2
Address 0x06
Basic I/O 2
RUN
Input 0x3EC
Input 0x0EB
Input 0x2CA
Input 0x1FE
CONT
Echo 0x3EC
Echo 0x0EB
Echo 0x2CA
Echo 0x1FE


Program #3
Address 0x0B
Self-Modifying
RUN
17 increments, one on each CONT


Program #4
Address 0x14
XOR Operation
RUN
Operand 1 0x3EC
Operand 2 0x0EB
Operand 1 0x2CA
Operand 2 0x1FE
CONT
Result 0x307
Result 0x334


Program #5
Address 0x31
Multiplier
RUN
CONT
Operand 1 0xEC
Operand 2 0xEB
Operand 1 0xCA
Operand 2 0xFE
Result 0xEDA4
Result 0xCA6C
0x10000 − (0x100 − 0xEC) × 0xEB = 0xEDA4
0x10000 − (0x100 − 0xCA) × 0xFE = 0xCA6C


Program #6
Address 0x5A
Bubble Sort Pt.1
RUN
CONT
16 displays of unsorted values, one on each CONT
DISPLAY SUBFUNCTION
SORT SUBFUNCTION

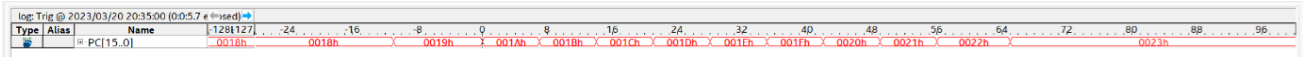


MIPS Analysis – *clock frequency is 86.84MHz, average processing speed is <u>12.403 MIPS</u>*

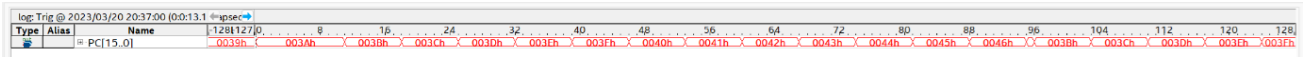
Program	Expression	Algorithm Range	# Instr.	# Cycles	MIPS
XOR	`0xEC XOR 0xEB`	`0x19` ~ `0x20` (Core) `0x19` ~ `0x21` (Peri.) `0x14` ~ `0x23` (Full)	9	65	12.024
MULT	`0xEC MULT 0xEB`	`0x3A` ~ `0x45` (Core) `0x39` ~ `0x46` (Peri.) `0x31` ~ `0x48` (Full)	81	591	11.902

SORT	SORT(
	`0xEF, 0x1B, 0x01, 0x8C,`
	`0xDB, 0xFA, 0x47, 0x46,`
	`0x1F, 0x0D, 0xB8, 0x03,`
	`0x6B, 0x4E, 0xF8, 0x07`)
		`0x77` ~ `0x87` (Core)
		`0x77` ~ `0x87` (Peri.)
		`0x77` ~ `0x88` (Full)
			1,818
			77, 78
			(79 ~ 85) * 16
			86, 87, 78
			(79 ~ 85) * 15
			86, 87, 78
			……
			86, 87, 78
			(79 ~ 85) * 1
			86, 87, 88
				11,886	13.282

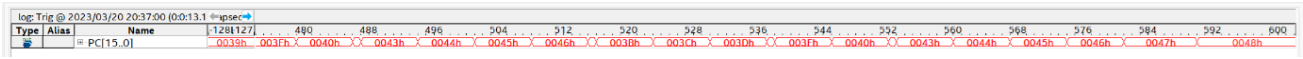
XOR



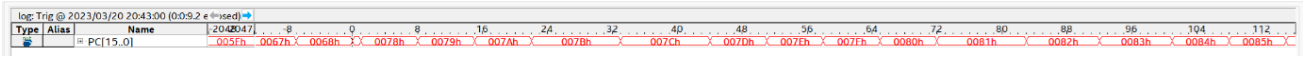
MULT Head



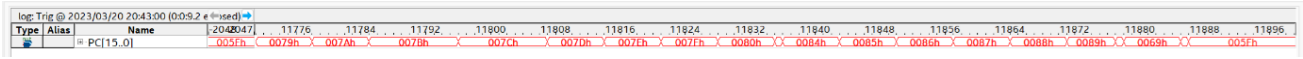
MULT Tail



SORT Head



SORT Tail



Post-Lab Problems

- Design Resources and Statistics

	LUT	DSP	Flip-Flop	Memory (BRAM)	Frequency /MHz	Static Power/mW	Dynamic Power/mW	Total Power/mW
Multiplier	1,044	0	270	16,384	86.84	89.94	0.00	98.70

We tried pushing the design to work with a 100MHz, 50% Duty Cycle clock but failed.

- The function of `Mem2IO`

This module manages I/O with DE10-Lite physical devices, namely, the switches and the 7-segment display. Input from switches is mapped to address `0xFFFF`, and output from address `0xFFFF` is mapped to Hex Display. The module detects load from the mapped address and redirects `LDR` and `STR` instructions to query peripheral devices.

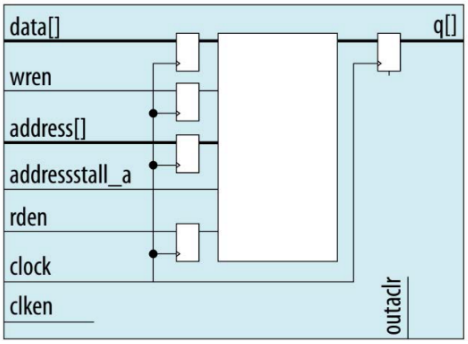
Physical I/O Device	**Type**	**Memory Address**	**"Memory Contents"**
DE10 Board Hex Display	Output	`0xFFFF`	`[6]HEX[7:0]`
DE10 Board Switches	Input	`0xFFFF`	`SW[9:0]`

- Difference between `BR` and `JMP`

Both instructions unconditionally change the value of `PC` but in different ways. `BR` adds a `SEXT`ed 9-bit offset to `PC`. The jumping range is restricted to (`PC-0x0100`, `PC+0x00FF`). `JMP` directly loads value in `BaseR` to `PC`. The range is extended to (`0x0000`, `0xFFFF`).

- Memory clocking issues: on the `R` signal

Reading value from memory takes much time, usually several clock cycles. In the *original* LC-3, the `R` signal tells us whether the value from MDR is ready. In the simplified SLC-3, we assume this process always takes 3 clock cycles and wait for 3 states before reading from MDR as a result of the 3-level register delay (buffered input, query, buffered output).



Conclusion

In this lab, we complete the SLC-3.2 design by implementing the ISDU finite state machine, which follows the Fetch-Decode-Execute cycles and waits for `RUN` keypress at `HALT` states. Each state corresponds to a set of control signals that goes into the datapath. We also implement the datapath by putting together registers, ALU, bus, and MUXes with different sizes. The correctness and performance of this minicomputer is evaluated on 6 test programs.