ECE 385

Fall 2020

Experiment 6

SLC-3

Zhicong Fan/Xin Jin
AB2/Online
Yucheng Liang

Introduction:

In this lab, we designed a SLC-3 using System Verilog on FPGA. It was a simplified version of LC-3, which was consisted of a 16-bit processor, several 16-bit instructions, and several 16-bit registers.

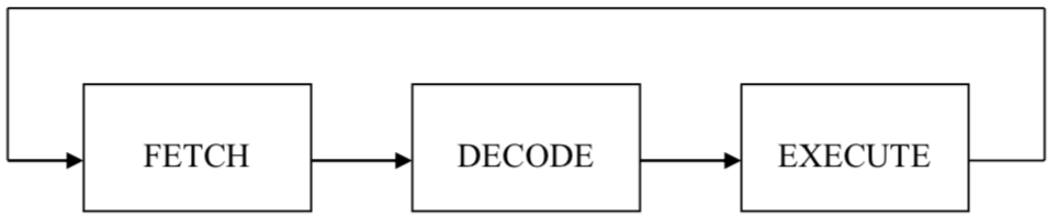
SLC-3:

Summary of Operation:

Our design was a simplified version of LC-3. It was able to accomplish most of the basic functionalities of the LC-3 such as the Boolean AND, ADD, and NOT, and the basic memory read and write (LDR and STR). In addition, it was also able to ask for the user inputs and perform different programs based on the user inputs (JSR, JMP, and BR).

SLC-3:

The SLC-3 was able to execute the instructions by following procedure below:



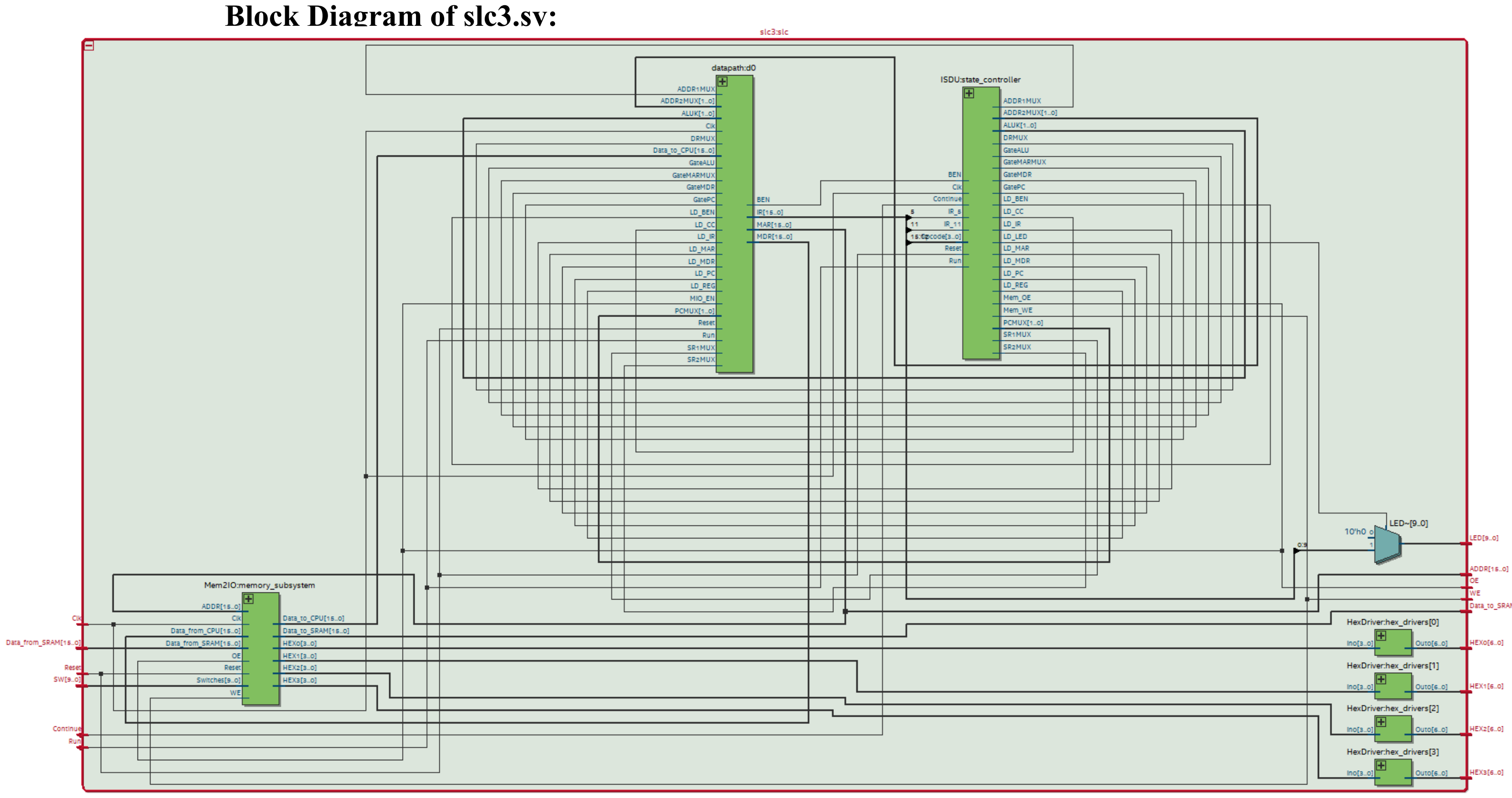
The computer will first fetch an instruction from the memory in the FETCH state. The current line to fetched will be tracked by PC which points to the address of the current instruction. The instruction address will be loaded in register MAR (Memory Address Register), and the content will be loaded into MDR (Memory Data Register). Then, the instruction will be loaded into IR (instruction register). In the DECODE state, the instruction in the IR state will be spared into several pieces. Each piece specifies the type of operation, the use of registers and some of the meta data. Finally, the EXECUTE state executes the instruction. After the computer done with the current instruction, it will go back to the FETCH state to perform next instruction.

Summary of the instructions:

- ADD: Adds the contents of SR1 and SR2, 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.
- 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. 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 pressed by the user at pause states. While paused, ledVect12 is displayed on the board LEDs.

Block Diagram of slc3.sv:



.sv Modules:

Module: 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
Description: This module synchronized the user inputs and declared the modules for the CPU (slc) and the memories (instaRam and ram0).
Purpose: The top level of the program. Initialized the CPU and the memory.

Module: SLC3.sv
Inputs: [9:0] SW, Clk, Reset, Run, Continue, [15:0] Data_from_SRAM
Outputs: OE, WE, [6:0] HEX0, [6:0] HEX1, [6:0] HEX2, [6:0] HEX3, [9:0] LED, [15:0] ADDR, [15:0] Data_to_SRAM
Description: contains three main modules: datapath, mem2io, and isdu, which controls

the data flow, the interaction between the input and output to the memory, and state of the circuit respectively.
Purpose: the CPU of the SLC-3, which is responsible for all of the computtaions.

Module: Instantiateram.sv
Inputs: Reset, Clk
Outputs: [15:0] ADDR, wren, [15:0] data
Description: this is the content of the on-chip program, which includes the address of the data and the content at each address.
Purpose: instantiate the on-chip memory

Module: datapath.sv
Inputs: Clk, Reset, Run, LD_MAR, LD_MDR, LD_IR, LD_PC, LD_BEN, LD_CC, LD_REG, MIO_EN, GatePC, GateMDR, GateALU, GateMARMUX, SR1MUX, SR2MUX, DRMUX, ADDR1MUX, [1:0] PCMUX, [1:0] ADDR2MUX, [1:0] ALUK, [15:0] Data_to_CPU
Outputs: [15:0] MAR, [15:0] MDR, [15:0] IR, [15:0] PC, BEN
Description: based on the inputs, the module will determine the what contents should be loaded into MAR, MDR, PC, and IR. All of the internal logics are controlled by the load enable flags, and gate enable determines which data will be loaded into bus. MUX select pins controls the outputs of the corresponding MUXs.
Purpose: determine the flow of the data in the circuit based on the inputs.

Module: ALU.sv
Inputs: [15:0] Input_A, [15:0] Input_B, [1:0] ALUK
Outputs: [15:0] ALU_out
Description: ALUK is the select pin for the ALU, which controls the outputs of the two input signals. Four types of operations are: ADD, AND, NOT, and input A.
Purpose: control the type of arithmetic-logic operation for the two input signals.

Module: BEN.sv
Inputs: Clk, LD_BEN, In
Outputs: BEN
Description: BEN will be updated to be In if LD_BEN is high, otherwise remain its original value.
Purpose: updates or hold the value in BEN

Module: MUXs.sv
Inputs: [15:0] IR, [15:0] PC, SR1MUX_select, SR2MUX_select, DRMUX_select, ADDR1MUX_select, [1:0] ADDR2MUX_select, [15:0] SR1_out, [15:0] SR2_out
Outputs: [2:0] SR1_Mux, [2:0] DRMUX, [15:0] ADDRMUX_out, [15:0] SR2
Description: determine the inputs and the outputs of the MUXs based on the select pins. The MUXs include SR1MUX, SR2MUX, DRMUX, ADDR1MUX, and ADDR2MUX. The output of SR1MUX will be the input for ADDR1MUX.

Purpose: get the outputs from the MUXs and connects the outputs to inputs between MUXs.

Module: HexDriver.sv
Inputs: [3:0] In0
Outputs: [6:0] Out0
Description: Convert 4 bits input data to a 7 bits board readable data.
Purpose: Convert the result to board readable hex decimal number.

Module: IR.sv
Inputs: Clk, Reset, [15:0] In, LD_IR
Outputs: [15:0] IR
Description: Reset clears the content of IR and LD_IR update the content of IR to be In. In other cases, the IR retains its value.
Purpose: determine the content of IR

Module: MAR.sv
Inputs: Clk, Reset, [15:0] In, LD_MAR
Outputs: [15:0] MAR
Description: Reset clears the content of MAR and LD_MAR update the content of MAR to be In.
Purpose: determine the content of MAR.

Module: MDR.sv
Inputs: Clk, Reset, [15:0] In, LD_MDR, [15:0] Data_to_CPU, MIO_EN
Outputs: [15:0] MDR
Description: Reset clears the content of MDR and LD_MAR update the content of MAR to be In, or to be Data_to_CPU if MIO_EN is high. In other cases, MDR retains its value.
Purpose: determine the content of MDR.

Module: bus_MUXs.sv
Inputs: GatePC, GateMDR, GateALU, GateMARMUX, [15:0] PC, [15:0] MDR, [15:0] MAR, [15:0] ALU
Outputs: [15:0] Out
Description: put either MAR, PC, MDR, or ALU on the bus if the corresponding gate is open. Otherwise set the value on bus to be unknown.
Purpose: determine the data on the bus.

Module: PC.sv
Inputs: Clk, Reset, [15:0] In, [1:0] PCMUX, LD_PC, [15:0] ADDRMUX_out
Outputs: [15:0] PC
Description: Reset clears the content of PC and LD_PC update the content of PC to be PC + 1, In, or ADDRMUX_out based on the PCMUX. In other cases, MDR retains its

value.
Purpose: determine the content of the PC

Module: register.sv
Inputs: Clk, Reset, Ld_REG, [15:0] In, [15:0] IR, [2:0] DRMUX, [2:0] SR1MUX
Outputs: [15:0] SR1, [15:0] SR2
Description: DRMUX specifies load enable for the target register SR1, IR[2:0] specifies the target register SR2, and SR1MUX specifies the target register SR1. Reset will clear the SR1 register, Ld_REG specifies whether the target register load enable should be set to high. If so, it will be updated to be In.
Purpose: read from/write to the registers

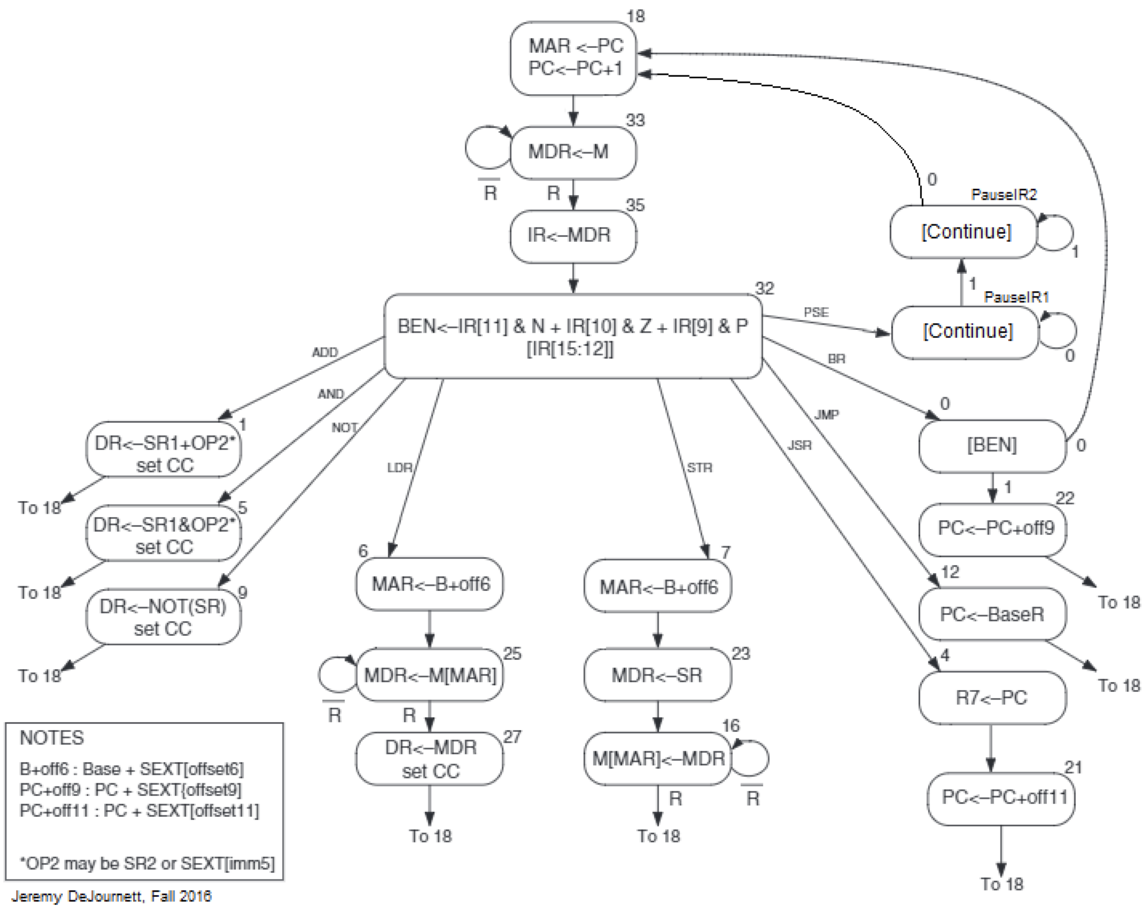
Module: set_CC.sv
Inputs: Clk, LD_CC, [15:0] In, [15:0] IR,
Outputs: nzp_logic
Description: Store the CC for operations that will set CC(LD_CC). In BR operation, the nzp in the opcode will be compared to the CC, if the condition meets, nzp_logic will be set to high.
Purpose: store the CC and keep track of the branch condition

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: (same answer in the post-lab question) MEM2IO controls the interactions between the memory and the inputs (switches), outputs (HEXes) and CPU. It will load the switch value to CPU when only read is enabled and the address is 0xFFFF. It will load the data from memory to CPU if only read is enabled and address is not 0xFFFF. It will write the data from CPU to memory if the write is enabled.
Purpose: control the interactions between the input/output signal and the memory.

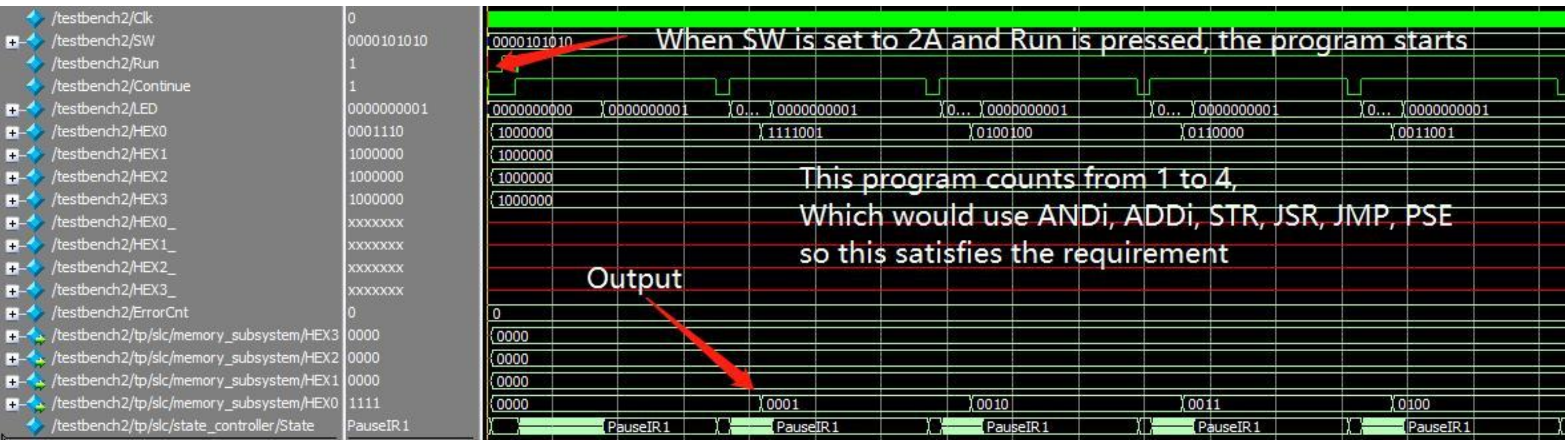
<u>ISDU:</u>

This module controls the state of the SLC-3. As shown in the state diagram below, the states 18, 33, and 35 are the FETCH state, state 32 is the DECODE state, and the rest of the states are the EXECUTE state. The next state is mostly determined on the current state and the content of IR. Except for state 32, the next state is depending on the opcode, the next states of the other states are purely depended on the current state. At each state, the flags are set as the outputs. For example, states 1, 5, 9, and 27 will need to set CC, so at those states, the LD_CC signal will need to be set high. States like 33, 25, and 16 need to access the memory, so the MEM_OE or MEM_WE will be set to high, which indicate the read/write enable. For more specific information, please refer to the ISDU.sv file.

<u>State Diagram for ISDU:</u>



<u>Simulations of SLC-3 Instructions:</u>



When SW is set to 2A and Run is pressed, the program starts
This program counts from 1 to 4,
Which would use ANDi, ADDi, STR, JSR, JMP, PSE
so this satisfies the requirement
Output
/testbench2/Clk 0
/testbench2/SW 0000101010
/testbench2/Run 1
/testbench2/Continue 1
/testbench2/LED 0000000001
/testbench2/HEX0 0001110
/testbench2/HEX1 1000000
/testbench2/HEX2 1000000
/testbench2/HEX3 1000000
/testbench2/HEX0_ xxxxxxx
/testbench2/HEX1_ xxxxxxx
/testbench2/HEX2_ xxxxxxx
/testbench2/HEX3_ xxxxxxx
/testbench2/ErrorCnt 0
/testbench2/tp/slc/memory_subsystem/HEX3 0000
/testbench2/tp/slc/memory_subsystem/HEX2 0000
/testbench2/tp/slc/memory_subsystem/HEX1 0000
/testbench2/tp/slc/memory_subsystem/HEX0 1111
/testbench2/tp/slc/state_controller/State PauseIR1
0001
0010
0011
0100

Post-Lab Question:

Q: Refer to the Design Resources and Statistics in IQT.17-19 and complete the following design statistics table.

A:

LUT	862
DSP	0
Memory (BRAM)	0
Flip-Flop	258
Frequency	86.78MHz
Static Power	89.94mW
Dynamic Power	0.00mW
Total Power	98.63mW

Q: What is MEM2IO used for, i.e. what is its main function?

A: MEM2IO controls the interactions between the memory and the inputs (switches), outputs (HEXes) and CPU. It will load the switch value to CPU when only read is enabled and the address is 0xFFFF. It will load the data from memory to CPU if only read is enabled and address is not 0xFFFF. It will write the data from CPU to memory if the write is enabled.

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

A: BR instruction will only change the PC, if the branch condition (nzp) is met. Operations such as ADD or AND will set CC. BR will only update the PC if the branch condition specified by the user matches the CC, otherwise the branch operation will be ignored. On the other hand, JMP instruction will update the PC based on the given register unconditionally.

Q: 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?

A: Since reading from and writing to memory are relatively slow compare to other operations in the computer, R signal is used to tell the processor that the slow operations like read or write operation have finished, so the processor could proceed on the next operation, otherwise it may cause the read/write conflicts. We compensate for the lack of R signal in SLC-3 by waiting for extra 2 states whenever the processor is reading or writing from the memory. Since we potentially will need to wait for longer time in the extra redundant states, the speed of accessing the memory will be much slower, but this ensures that the memory is synchronized with the rest of the circuit.

Conclusion:

Bug Encountered:

1. For the state 32, 16, and 25, we initially only waited for one extra states, but it turns out it is not long enough, which caused weird outputs. The issue was solved by waiting for one extra states.
2. When we do the simulation, the outputs were all wrong, but it worked perfectly on the board. We found that the issue was because the continue was pressed too early in the testbench. The issue was solved by pressing the continue at the correct pause state.

Lab Manuel Issues:

Pro:

1. The state diagram and the Datapath diagram is super helpful.

Con:

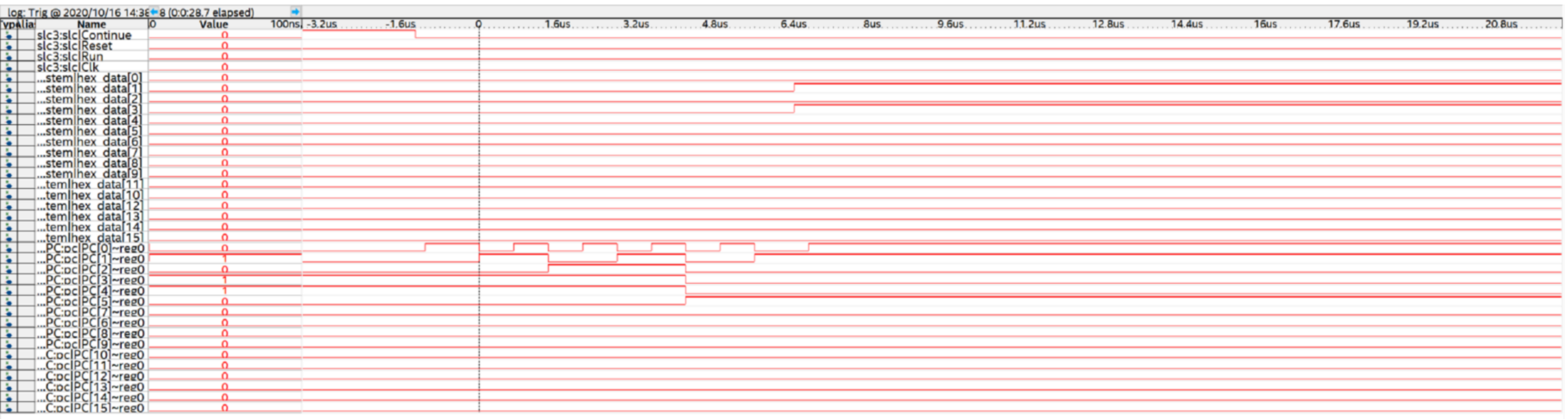
1. More comments in the provided code will be appreciated.
2. Instructions on how to use some advanced functions in simulation will be appreciated, such as adding waves.
3. It is not very clear how does the memory work since the ram part is provided.

Summary:

In this lab, we designed a SLC-3 processor, which is the simplified version of LC-3. We designed the datapath of the whole circuit and completed the provided control unit. The state machine in this lab is much more complicated and mature than the one in the previous labs. In the end, our circuit supports the following instructions: ADD, AND, NOT, BR, JSR, JMP, LDR, STR, and PSE, and we had experienced the CPU design for the simplified little computer 3.

Extra Credit:

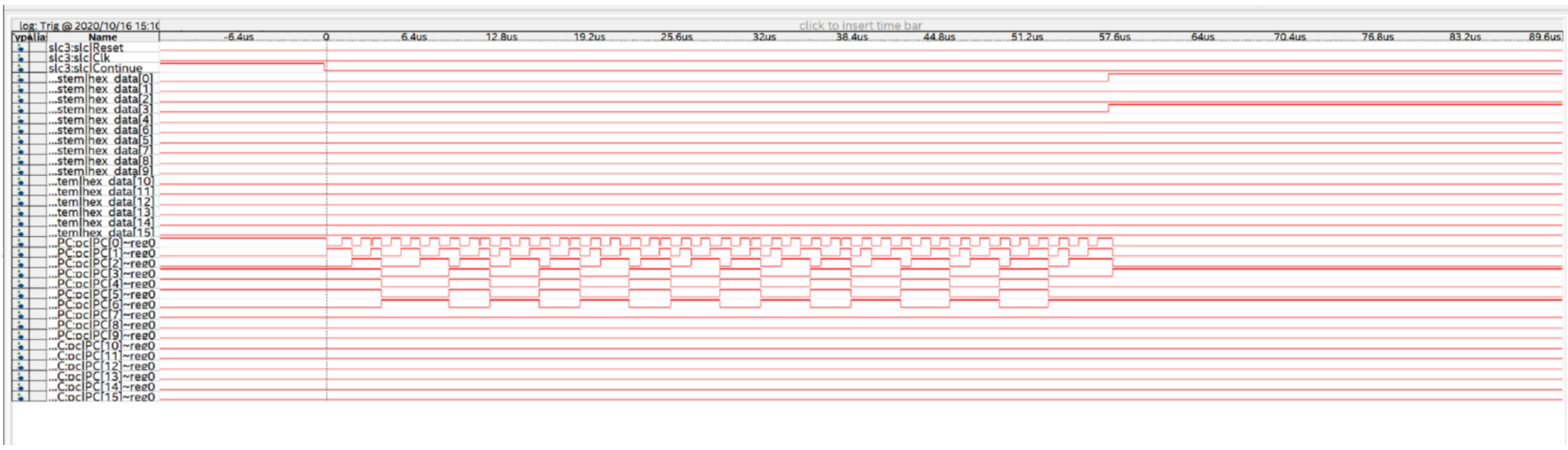
XOR:



From 0ns to 6.4ns, there are 9 instructions, which means: 9/6.4 = 1.40625 so 1.406 millions of instructions per second.

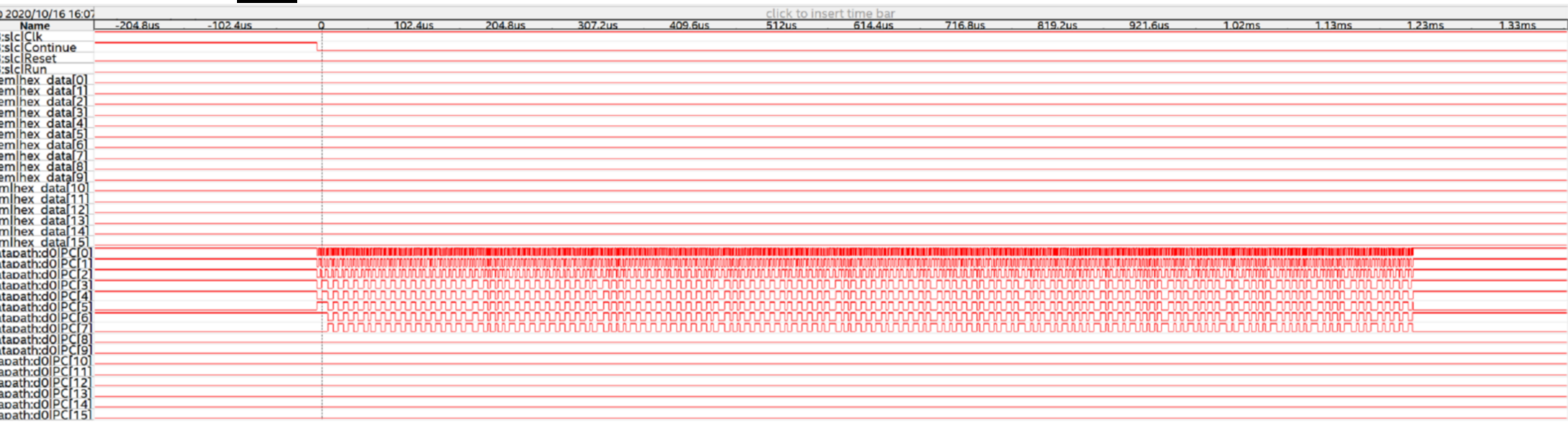
We start to count from 0 and found 9 instructions.

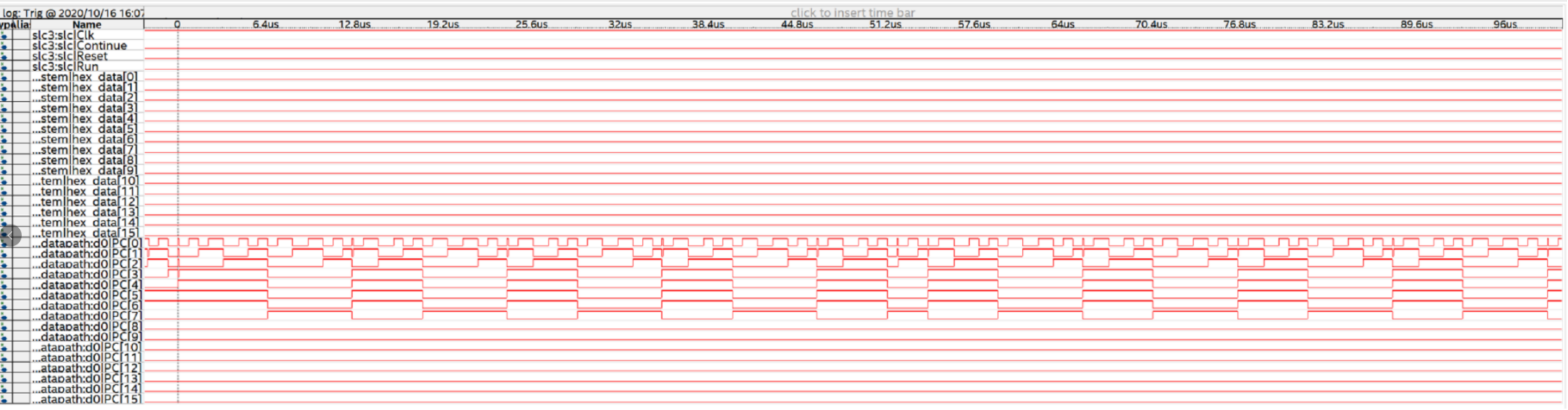
Multiplier:



From 0ns to 57.6ns, there are 84 instructions, which means: 84/57.6 = 1.45833 so 1.45833 millions of instructions per second.

Sort:





From 0ns to 57.6ns, there are 74 instructions, which means: 74/57.6 = 1.284722 so 1.284722 millions of instructions per second.

We counted how many times the pc has increased and since the last one is too long, we still use 57.6 as the time. We cannot directly use the last pc to minus the starting pc because the jmp operation would change the value of pc which would make the final pc value much smaller than actual implementation.