16-bit Microprocessor, a Subset of LC-3 ISA

1. <u>Introduction:</u>

In this lab, we designed a simple microprocessor using SystemVerilog, which is a subset of the LC-3 ISA. Like the LC-3 ISA, the microprocessor, Program Counter (PC), registers, and instructions will all be 16 bits. The Simplified LC-3 (SLC-3) will be able to fetch an instruction from memory, decode it, and then execute the instruction. The SLC-3 will have fewer instructions than the original LC-3, which means it will have a simpler FSM and a more straightforward control unit. This lab was divided into two parts. In the first part, we implement the fetch operation, which involves getting the desired instruction from the memory. In the second part, we constructed the full CPU unit to execute all the instructions.

2. <u>Description & Diagrams of SLC-3:</u>

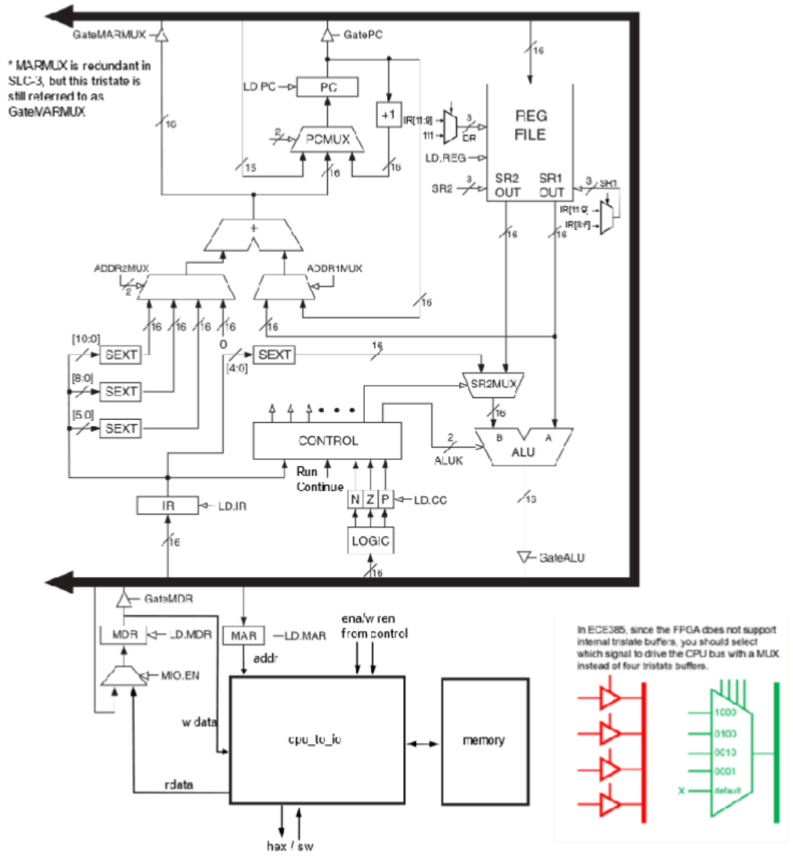
Summary of Operation

The SLC-3 follows the classic fetch-decode-execute cycle as the LC-3 ISA. The SLC-3 CPU consists of a 16-bit data bus, a control unit to control the FSM and its corresponding signals, a register unit that has 8 16-bit general-purpose registers, and an ALU. The SLC-3 supports the following arithmetic, logic, and control flow instructions: ADD, AND, NOT, BR, JMP, JSR, LDR, STR, and PAUSE. The SLC-3 CPU is able to execute the correct instruction based on the opcode, which is the 4 MSBs of the Instruction Register (IR), and is able to conditionally execute instruction through the NZP (Negative, Zero, Positive) condition codes. Then, the SLC-3 can store or load values from memory and interact with switches and LEDs of the FPGA board.

Description of Instructions

Instruction	Description
FETCH	The SLC-3 loads the PC into the Memory Address Register (MAR), increments the PC, loads the content of the MAR into the Memory Data Register (MDR), and loads the MDR into the Instruction Register (IR).
DECODE	The 4 MSBs of IR (opcode) determine in the control unit the next state to execute the desired instruction and, therefore, turn on and off the corresponding control signals. In addition, the bits [11:9] are loaded into the BEN register to set the condition sets NZP
EXECUTE	This instruction is not fixed but rather depends on the desired instruction set by DECODE
ADD	This instruction takes the contents of the outputs of the register file SR1 and SR2, which could be any of the 8 general-purpose registers, and adds them arithmetically together, then stores them in the destination register (DR) and updates the status register
ADDi	This adds the contents of SR1 with the sign-extended value of imm5, which is the bits [4:0] of IR, stores them in DR, and updates the status register
AND	This logically ANDs the contents of SR1 and SR2, stores them in DR, and updates the status register
ANDi	This logically ANDs the contents of SR1 and imm5 (bits [4:0] of IR), stores them in DR, and updates the status register
NOT	This logically negates the contents of SR1, stores it in DR, and updates the status register
BR	If any of the condition codes NZP matches the status register, it branches PC to the location of PC + PCoffset9 (bits [8:0] of IR)
JMP	It jumps the PC to the memory address in BaseR
JSR	It jumps the PC to a subroutine, storing the current PC to R7, and goes to PC+PCoffset11 (bits [10:0])
LDR	It loads the DR with the contents pointed to by the BaseR + the sign-extended offset6 (bits [5:0])
STR	It stores the contents of the SR at the memory address in BaseR + sign-extended offset6 (bits [5:0])
PAUSE	It pauses the execution until the Continue button is pressed by the user

Block Diagram of CPU



GateMARMUX
GatePC
* MARMUX is redundant in SLC-3, but this tristate is still referred to as GateVARMUX
LD.PC
PC
PCMUX
+1
REG FILE
DR
LD.REG
SR2
SR1
SR2 OUT
SR1 OUT
ADDR2MUX
ADDR1MUX
SEXT
SR2MUX
CONTROL
ALU
ALUK
Run
Continue
IR
LD.IR
LD.CC
LOGIC
GateALU
GateMDR
MDR
LD.MDR
MAR
LD.MAR
ena/w ren from control
MIO.EN
addr
w data
rdata
cpu_to_io
memory
hex / sw
In ECE385, since the FPGA does not support internal tristate buffers, you should select which signal to drive the CPU bus with a MUX instead of four tristate buffers.

Figure 1: Block diagram of the SLC-3 datapath from the lab's manual

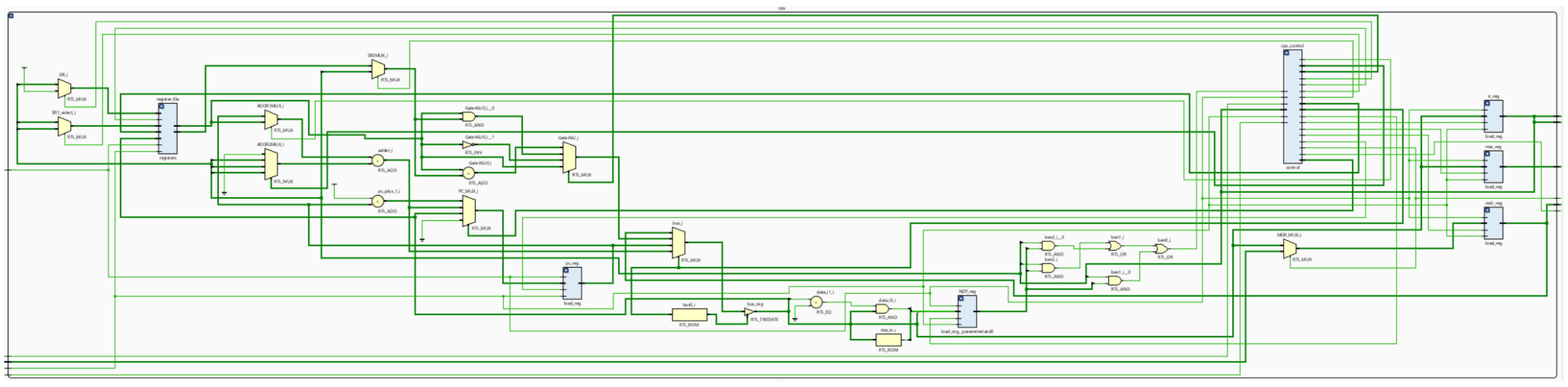


Figure 2: Top Level Block Diagram of the CPU from Vivado

Top Level Block Diagram

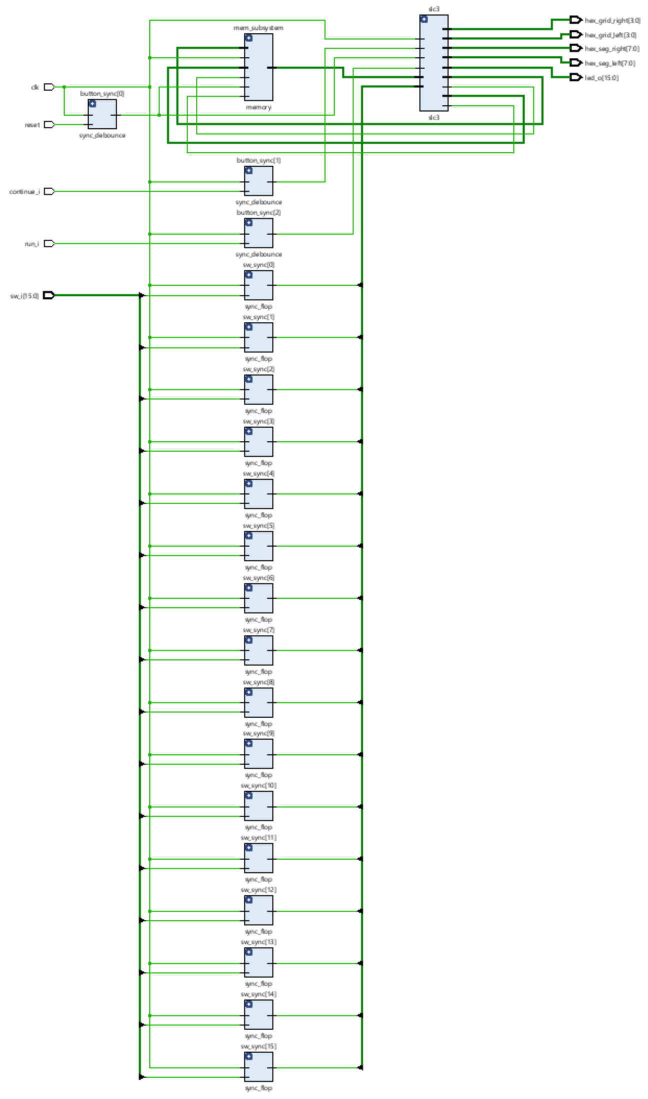
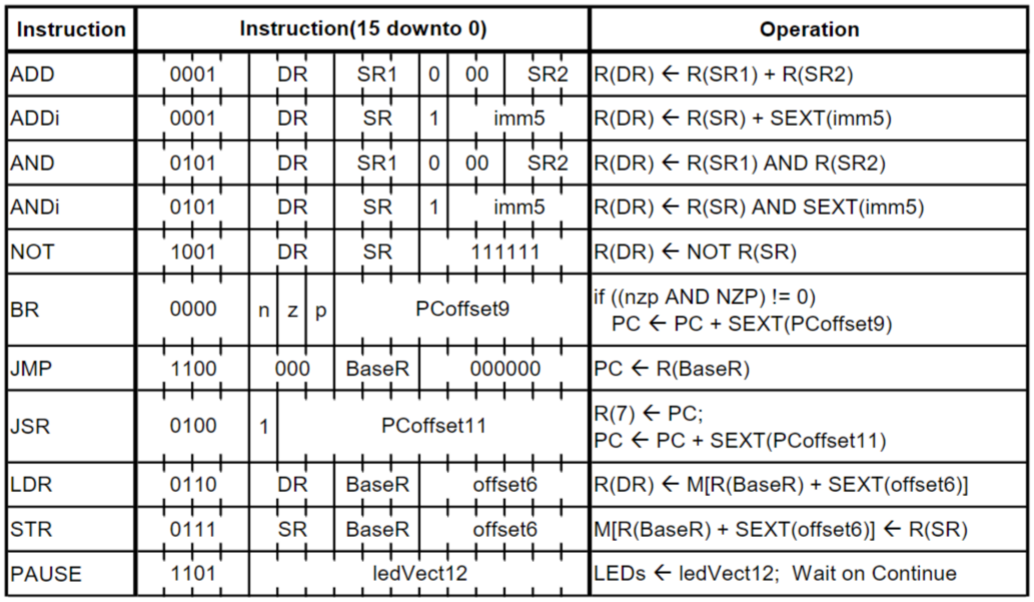


Figure 3: Block diagram of the top level with SLC-3 on the top right, memory in the middle, and the rest are sync debounce

Description of the Control Unit

The control unit of the SLC-3 takes as input the IR, run and continue buttons, and the condition sets NZP. Also, the control unit distributes all the control signals as output for each state. First, the control unit defines all the states and the corresponding control signal that should be on or off for each state. Then, the control unit puts the SLC-3 in a halt state until the run button is pressed, which makes the SLC-3 go to state 18, the start of the FETCH operation. The next state is immediately determined to be 33 to continue fetching the instruction from memory. Still, since we don't have the R (ready) signal as the LC-3, state 33 is divided into three states to wait for three clock cycles to make sure the memory value reaches the MDR. After that, the next states are directly 35 and 32 to finish fetching and decoding. In state 32, the next state is determined by a multiplexer based on the 4 MSBs of the IR (opcode), as shown in Figure 4 below, to go to the right execution state of the desired instruction.



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

Figure 4: The SLC-3 ISA table from the lab's manual

State Diagram

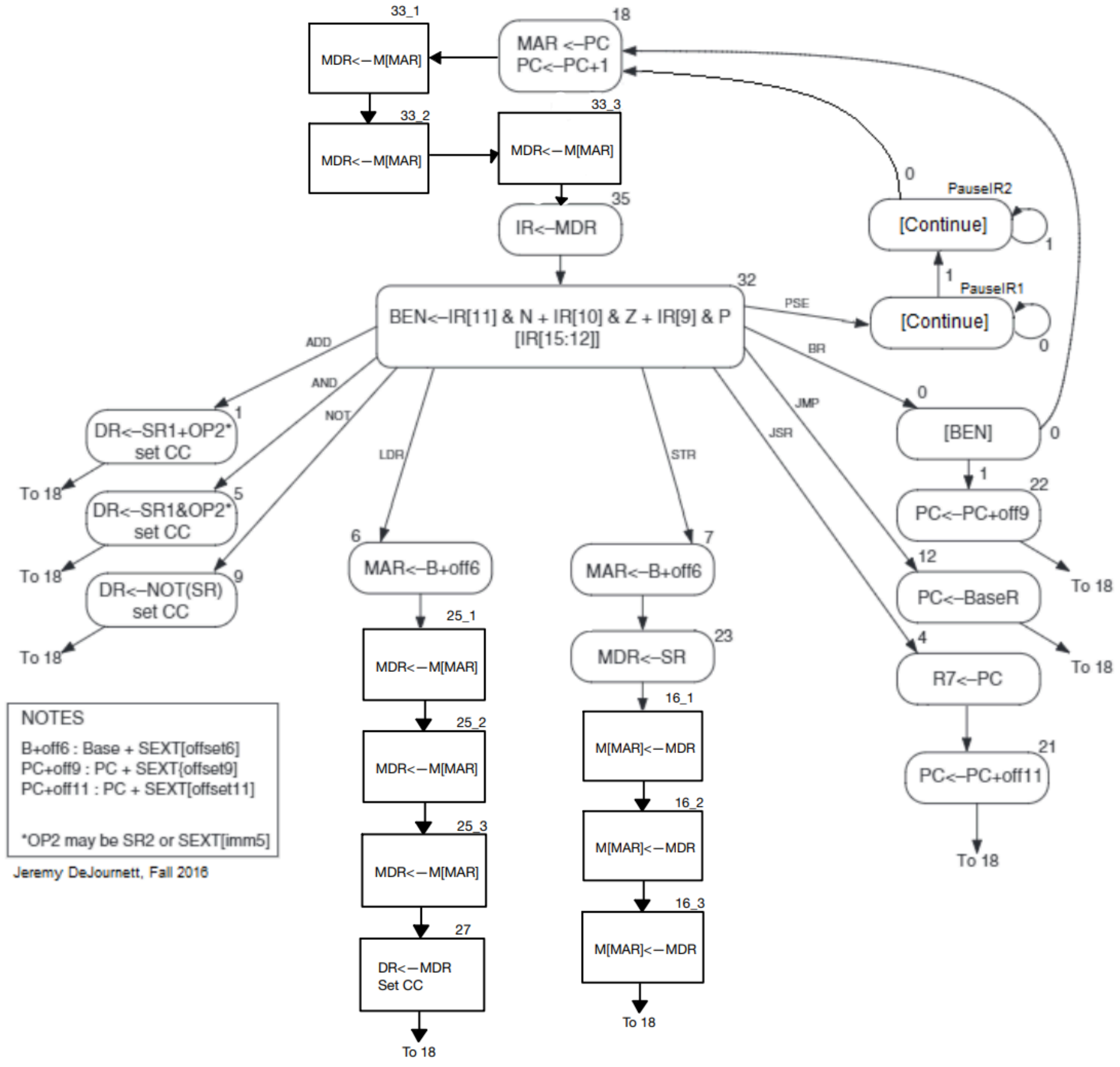


Figure 5: Modified state transition diagram for the SLC3

- **What is the difference between BR and JMP instructions?**

Both instructions manipulate the flow of the program counter and make it jump between the lines of code. However, the BR instruction is conditional while JMP is not, meaning that the PC doesn't necessarily branch from its current value when the current instruction is BR. The condition codes NZP in BR have to match those in the status register to branch. Another difference is that in BR, the provided 9-bit offset is added to the program counter, while in JMP, the value of the given register is copied into PC.

- **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 is a signal that comes from the memory or RAM to tell the CPU that the requested data, either read or write, is ready so it may proceed with its operation. This signal was originally used in the LC-3 in the DECODE, LDR, and STR (and others that we are not interested in for the SLC-3), which is whenever we needed access to the memory to load or store MDR. Our design for the SLC-3 lacks this signal, so to compensate for it, we have to wait for three clock cycles to ensure that the desired data is accessed. This compensation affects the performance of our design because instead of waiting for potentially one or two clock cycles, we are delayed to 3 clock cycles, which makes the design slower compared to the original in executing the same instruction.

3. <u>Conclusion:</u>

In this lab, we successfully implemented a simplified LC-3 microprocessor using SystemVerilog. We designed the fetch, decode, and execute stages and simulated the execution of various instructions such as ADD, AND, BR, and JMP. Our design demonstrated the ability to interact with I/O devices, load and store data, and perform arithmetic, logical, and flow manipulation operations.