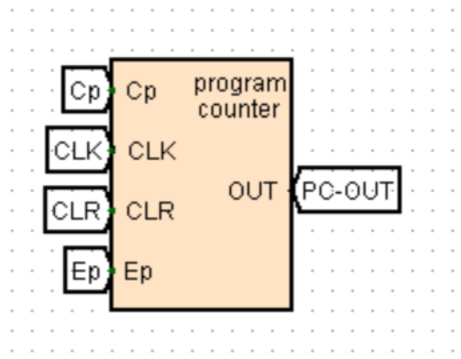
Introduction: The Simple As Possible-1 (SAP-1) architecture is an entry-level model for understanding computer design and operation principles. It is a minimalistic computer system demonstrating fundamental concepts such as instruction execution, data transfer, and control signal generation. It is an 8-bit microprocessor and is widely used to teach students the inner workings of a CPU and how its components interact to process instructions.

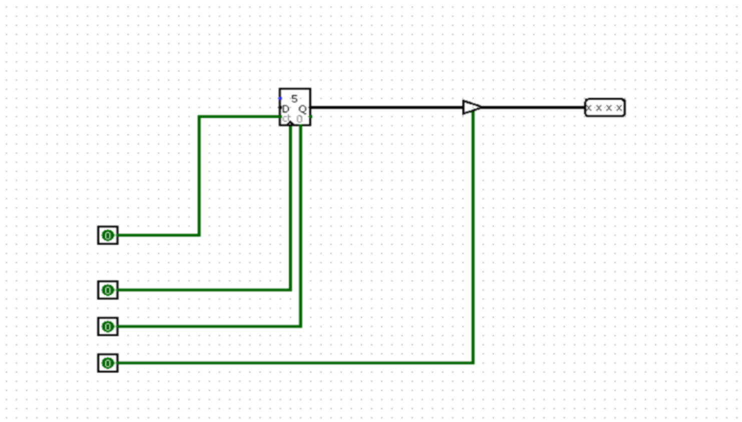
This project aimed to design and implement the SAP-1 architecture using a digital logic simulator. I chose the digital logic simulator 'Logisim' to implement this model. A unique feature of this implementation was the method used to generate the control signals required for instruction execution. Instead of using a ROM-based approach, the control signals were derived by feeding a carefully constructed truth table into Logisim's '*combinational analysis' tool*. This innovative approach simplified the design and provided a more hands-on understanding of Boolean logic and circuit design.

This report details the architecture, and design process of the SAP-1 implementation. It emphasizes the implementation of combinational logic for generating control signals and illustrates how the system was effectively tested using sample programs.

Architecture Overview: The SAP-1 architecture includes the following primary components:

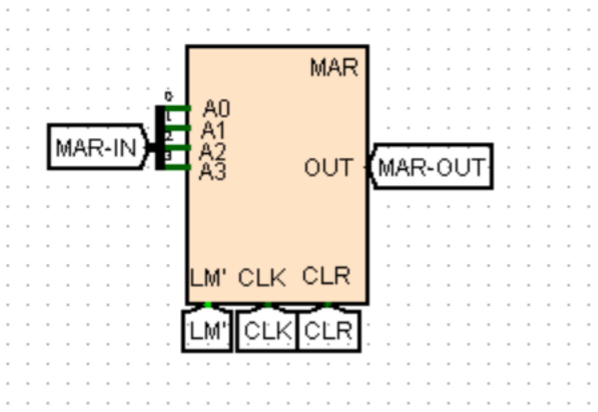
Program Counter (PC): A 4-bit counter is used to store the next instruction's address.

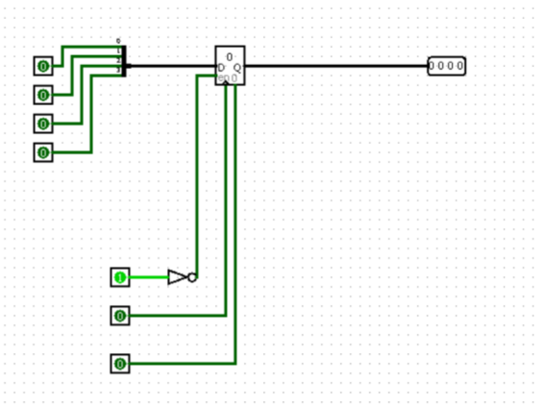




Used a ‘Counter’ component of Logisim to make the program counter. Also, used a ‘Controlled Buffer’ which gives us manual control over when to put the content of the counter in the data bus. The input of the controlled buffer is denoted as ‘Ep’. ‘Cp’ is connected to the enable pin of the counter that enables the counter for counting when ‘Cp’ is high. PC sends out the address of the next instruction to the memory address register.

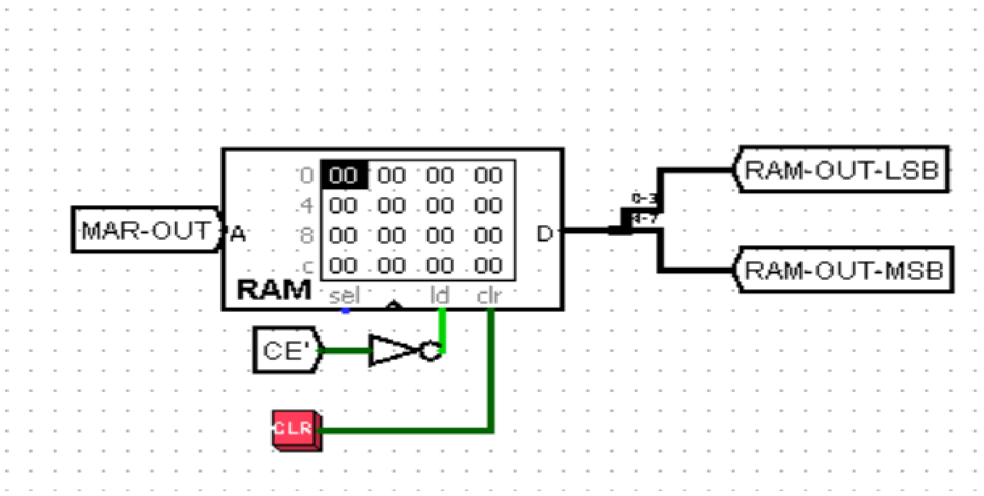
Memory Address Register (MAR): Holds the address of the memory location being accessed.





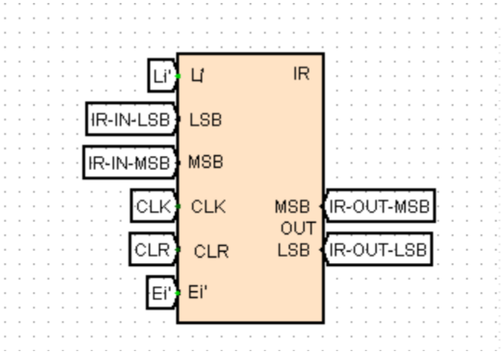
A 4-bit register is used to implement memory address register. A NOT gate and the enable pin is used to implement the LM~(complement), which enables the register to load data when LM~ is low. The MAR takes input from the PC through data bus and sends it to the random-access memory.

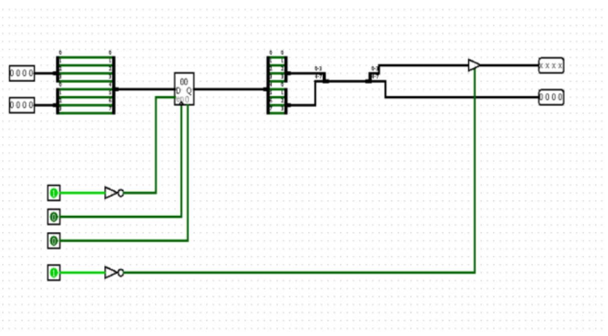
Random-access memory (RAM): Holds the instructions.



RAM holds the opcode and operands. Logisim's RAM block is used to implement this. It takes the input from the MAR to know which address to access and outputs the instruction stored in that address. Then sends the instruction to the instruction register through data bus.

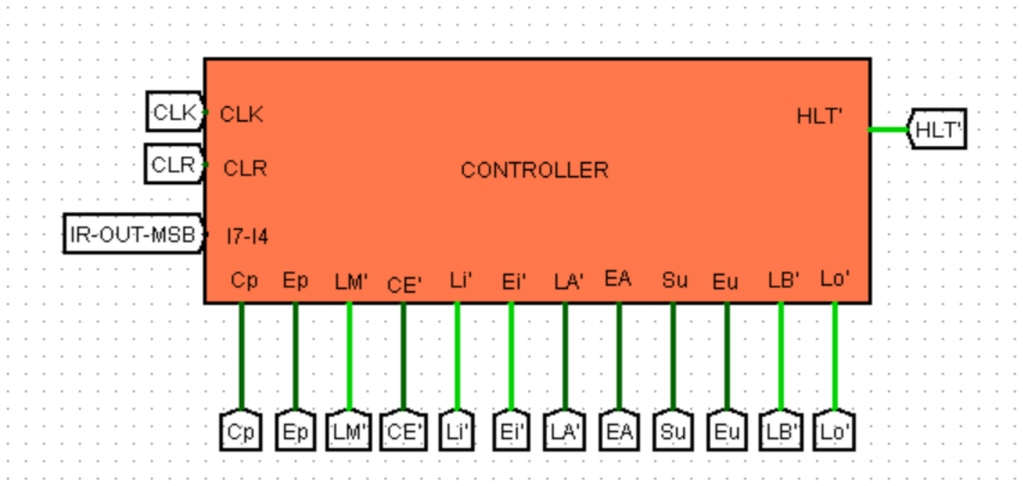
Instruction Register (IR): Temporarily stores the fetched instruction.





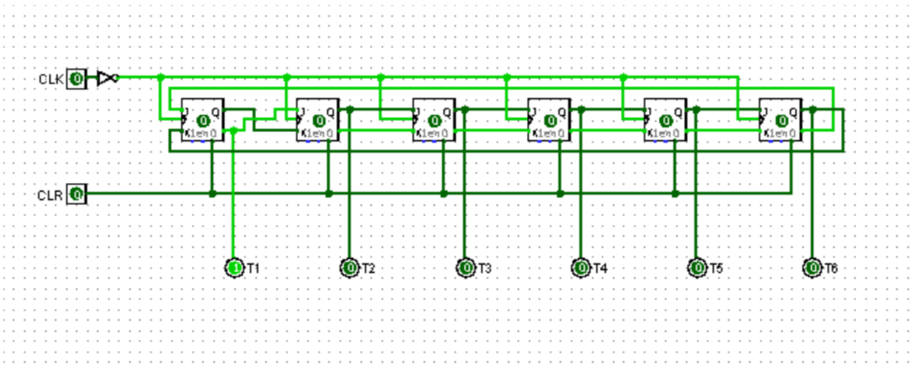
An 8-bits register block is used to make instruction register. For loading the IR, LI~(complement) is implemented using a NOT gate and the enable pin of the register. Here, the instruction's 2 parts, opcode and the address of the operands are divided. The 4-bits MSB of the instruction is defined as opcode and the 4-bits LSB is defined as the address of the operands in the RAM. So, IR outputs the MSB directly to the Control Unit and puts the LSB to the data bus for the MAR to take it as an input. And the LSB output is controlled by a 'Controlled Buffer' to decide when to put the it in the data bus.

Control Unit: Generates the required control signals to coordinate operations across the SAP-1 components.



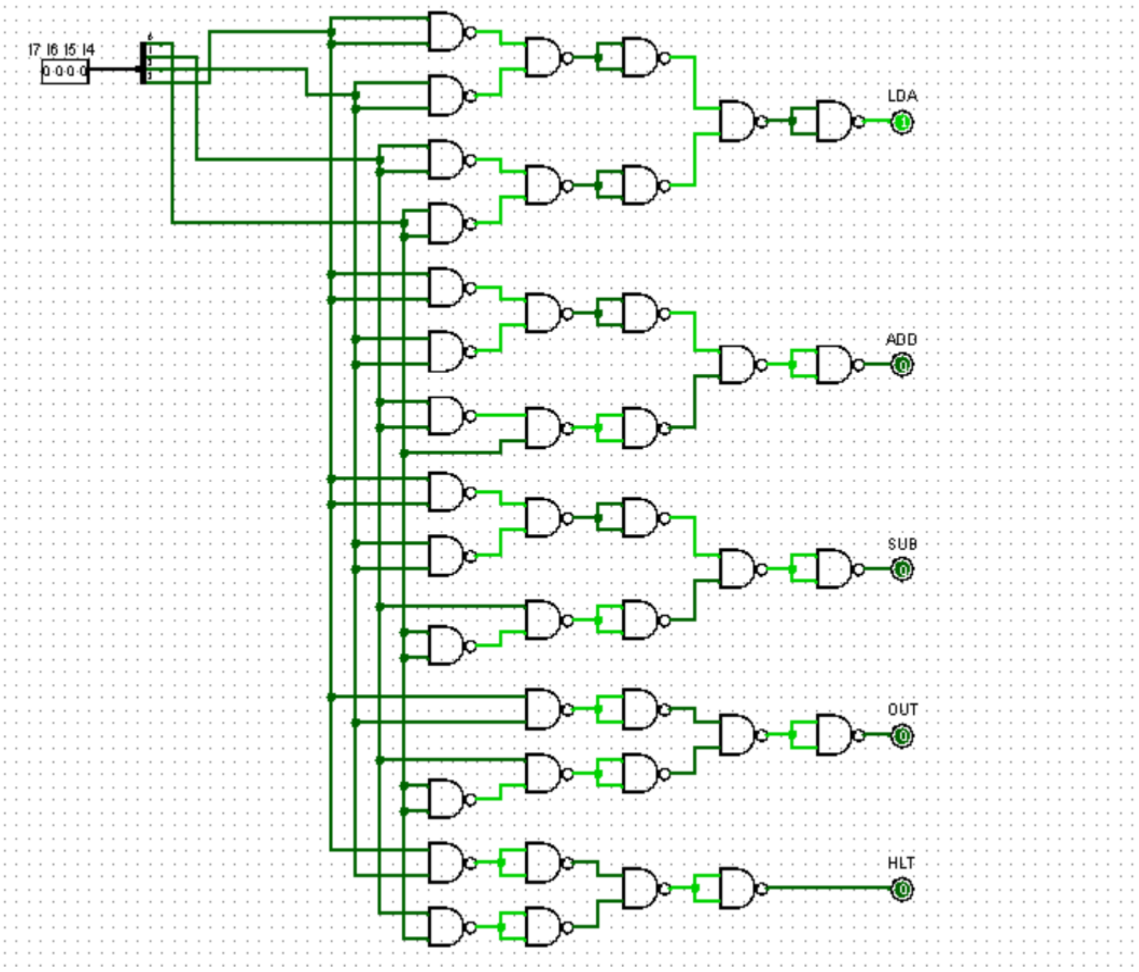
Controller is implemented with a few other components. They are:

A. **Ring Counter:** Counts from T1 to T6 states.



6 J-K flip-flops are used to make the ring counter. Ring counter counts from T1 to T6 states for each instruction.

B. **Instruction Decoder:** Decodes the instruction that came from the IR.

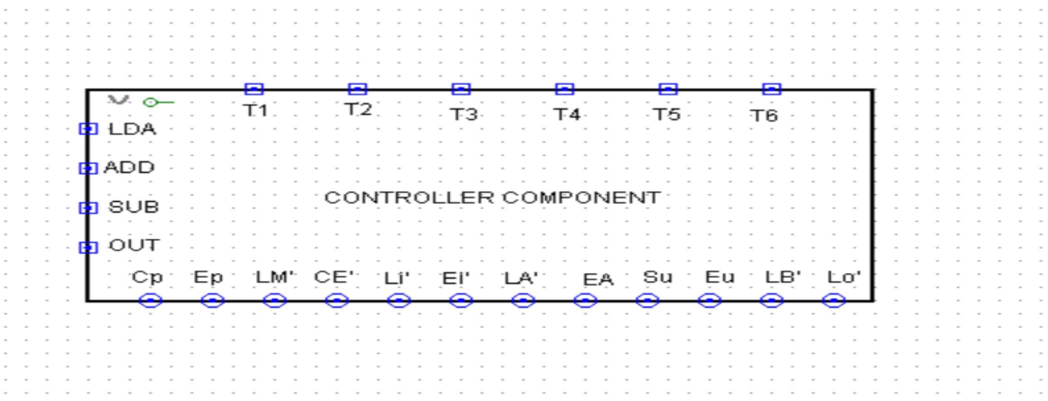


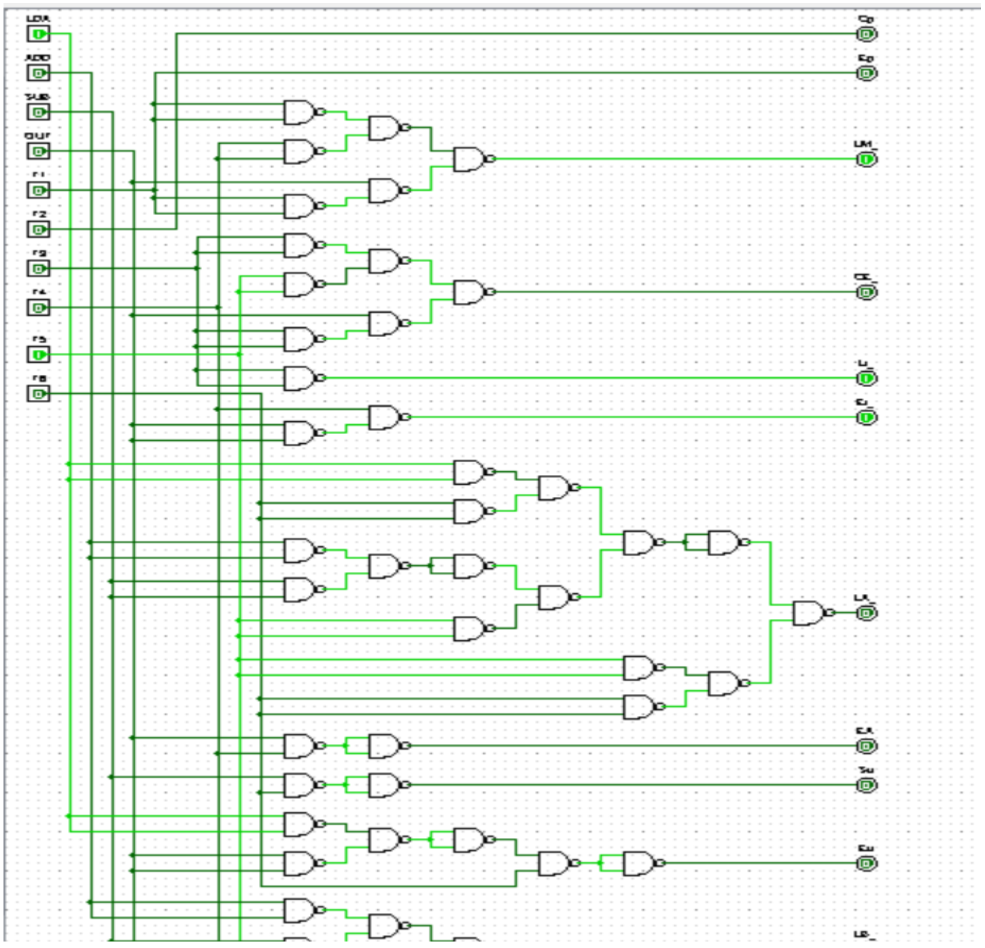
The instruction decoder is implemented using 'combinational analysis' feature in Logisim where a truth table was given as input and as output, we got a circuit according to the truth table. The instruction decoder outputs if the instruction that came from IR is whether LDA, ADD, SUB, OUT or HLT by generating active high for the specific output.

Truth Table:

I7-I4	LDA	ADD	SUB	OUT	HLT
0000	1	0	0	0	0
0001	0	1	0	0	0
0010	0	0	1	0	0
0011	DON’T CARE				
0100					
0101					
0110					
0111					
1000					
1001					
1010					
1011					
1100					
1101					
1110	0	0	0	1	0
1111	0	0	0	0	1

A. **Controller Component (Signal generator for specific instruction):** This circuit generates the signal for all the components of SAP-1.





This component takes the input from the instruction decoder and ring counter and generates the control signals for all the components in the SAP-1. This component doesn’t take HLT instruction as an input. HLT comes out from the instruction decoder and goes through a NOT gate that will serve its purpose. This component generates outputs according to the output of instruction decoder and the timing states generated by the ring counter. The circuit is implemented using a truth table.

Truth table:

T1 to T3 is same for all the instructions. Here, LDA, ADD, SUB, OUT are high one by one, not altogether.

T1-T3	Cp	Ep	LM’	CE’	LI’	EI’	LA’	EA	Su	Eu	LB’	Lo’
100	0	1	0	1	1	1	1	0	0	0	1	1
010	1	0	1	1	1	1	1	0	0	0	1	1
001	0	0	1	0	0	1	1	0	0	0	1	1

LDA: For T4-T6, LDA high. ADD, SUB, OUT are low.

T4-T6	Cp	Ep	LM’	CE’	LI’	EI’	LA’	EA	Su	Eu	LB’	Lo’
100	0	0	0	1	1	0	1	0	0	0	1	1
010	0	0	1	0	1	1	0	0	0	0	1	1
001	0	0	1	1	1	1	1	0	0	0	1	1

ADD: For T4-T6, ADD high. LDA, SUB, OUT are low.

T4-T6	Cp	Ep	LM’	CE’	LI’	EI’	LA’	EA	Su	Eu	LB’	Lo’
100	0	0	0	1	1	0	1	0	0	0	1	1
010	0	0	1	0	1	1	1	0	0	0	0	1
001	0	0	1	1	1	1	0	0	0	1	1	1

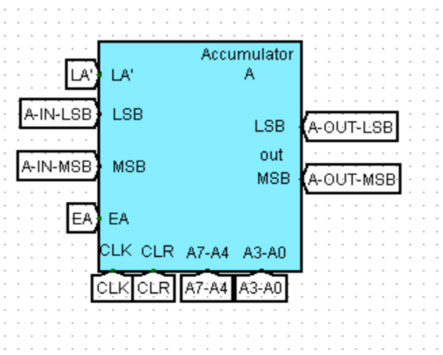
SUB: For T4-T6, SUB high. LDA, ADD, OUT are low.

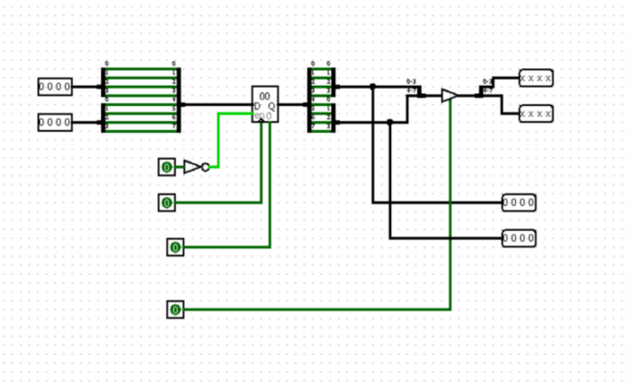
T4-T6	Cp	Ep	LM’	CE’	LI’	EI’	LA’	EA	Su	Eu	LB’	Lo’
100	0	0	0	1	1	0	1	0	0	0	1	1
010	0	0	1	0	1	1	1	0	0	0	0	1
001	0	0	1	1	1	1	0	0	1	1	1	1

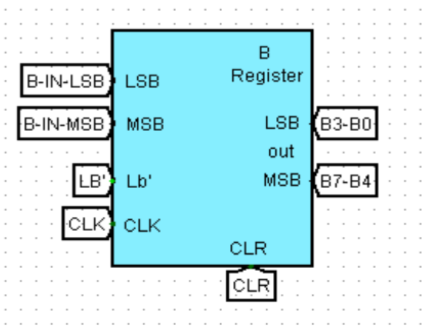
OUT: For T4-T6, OUT high. LDA, ADD, SUB are low.

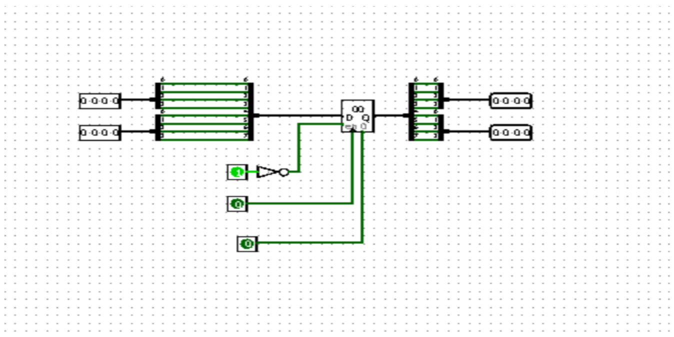
T4-T6	Cp	Ep	LM’	CE’	LI’	EI’	LA’	EA	Su	Eu	LB’	Lo’
100	0	0	1	1	1	1	1	1	0	0	1	0
010	0	0	1	1	1	1	1	0	0	0	1	1
001	0	0	1	1	1	1	1	0	0	0	1	1

Registers: General-purpose registers to store data.



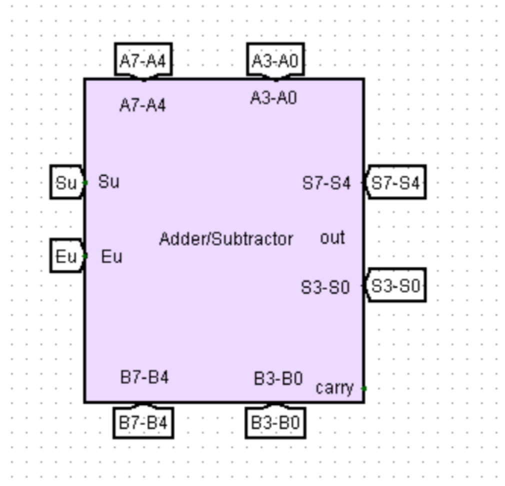


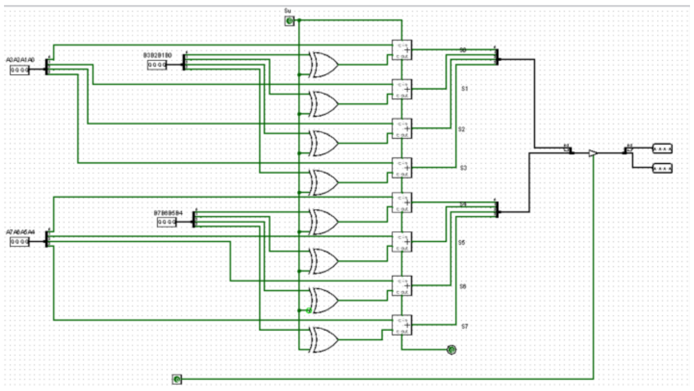




Two general purpose 8-bits registers are used in the SAP-1 model for data storing. One is accumulator and the other is B register. A controlled buffer is used with the accumulator to control when the output of the accumulator should be in the data bus.

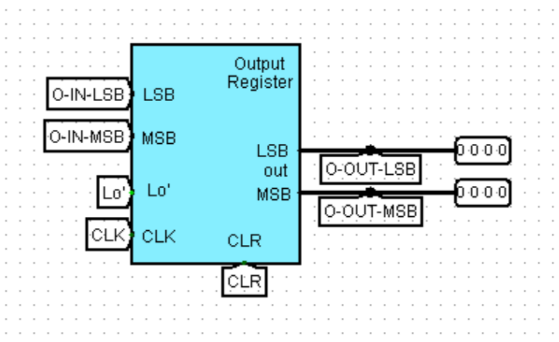
ADDER/SUBTRACTOR: Performs arithmetic operations.

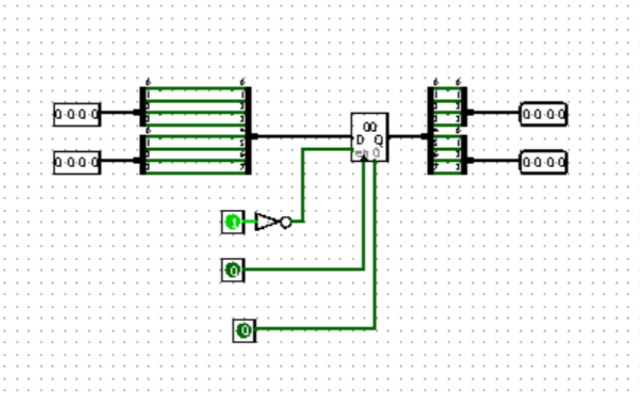




8-bits adder/subtractor is implemented using 8, 1-bit adder. Also, used XOR gate to make 2’s complement of a number for subtraction purpose. Used a controlled buffer to decide when to put the output of the adder/subtractor to the bus. This circuit takes input from accumulator and B register, performs addition or subtraction on them and stores the result on accumulator.

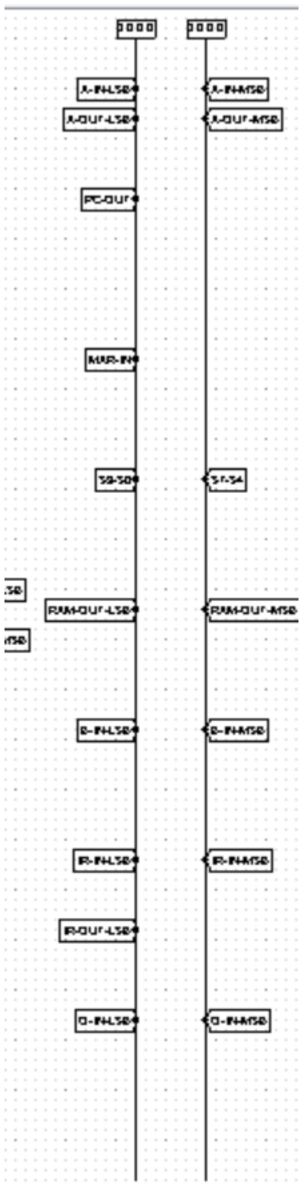
Output Register: Stroes the final result of computations and displays it using Hex Digit Display.





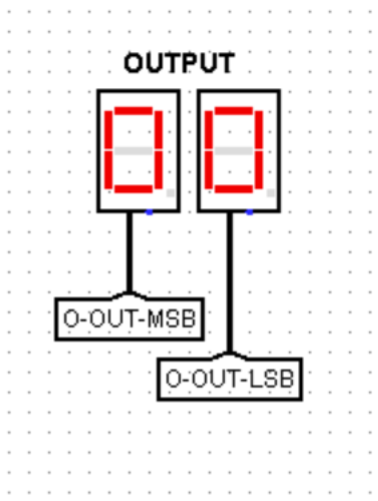
8-bits register is used for this. It takes input from the accumulator through data bus and displays the output through Hex Digit Display.

Data Bus: Transfers instruction and data between SAP-1 components.



8-bits data bus is used for transferring instructions and data between the components of SAP-1.

Hex Digit Display: Displays the output of the computations.



It takes input from the output register and displays the taken input.

Conclusion:

It has turned out to be a creative and successful use of Logisim's combinational analysis tool to implement the SAP-1 architecture and generate control signals. The architecture was significantly simplified without sacrificing system functionality by using a minimum logic circuit built from a truth table for the conventional ROM-based method. This method not only made implementation easier, but it also showed how useful and effective combinational logic is at coordinating a simple computer's actions.

This approach demonstrates a more profound, practical comprehension of control signals' function in computer design. It enables designers to see how Boolean logic powers SAP-1's real-time processes, bridging the gap between theory and practice. Additionally, the removal of ROM as a control mechanism demonstrates how design complexity can be decreased without compromising system integrity and performance.

All things considered, this project highlights the value of creative problem-solving in digital design and highlights the use of programs like Logisim to create fundamental yet efficient computational models. The SAP-1's educational value in examining the fundamentals of computer systems is confirmed by the implementation's success.