**ELEMENTS OF ALU**

* **1-BIT MEMORY: -**

Since the only thing inside of computers is bits, and the only thing that happens to bits is that they either turn on or turn off, then it follows that the only thing a computer can 'remember' is whether a bit was on or off. The following diagram shows one bit of computer memory.

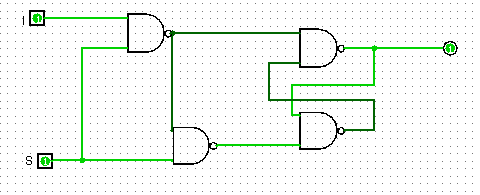
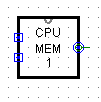
 

Fig.1: 1-bit memory

*WORKING: -* This circuit has 2 inputs ‘I’ and ‘S’. Whenever ‘S’ is set to 1, value of ‘I’ is obtained at the output. When ‘S’ is equal to zero, the circuit’s output does not change. That is, circuit remembers its previous value, hence this circuit acts like a memory that hold 1 bit.

The circuit on the right is the circuit appearance of 1-bit memory to be used in other circuits.

* **8-BIT MEMORY: -**

So, by using eight 1-bit memory units we can build a 8-bit memory structure.

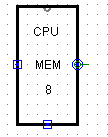
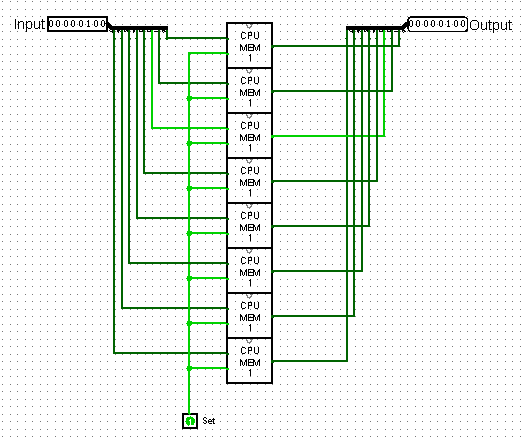


Fig.2: 8-bit memory

This circuit has 8 bits of input and 8 bits of output.

*WORKING: -* Its working is same as that of a 1-bit memory, it just contains 8 blocks of 1 bit memory.

* **REGISTER: -**

To create a register for our ALU which is used as general-purpose register, I have to incorporate two circuits. First one is the 8-bit memory that is shown before and the other circuit is the enable circuit which will control the flow of bits from memory to the output.

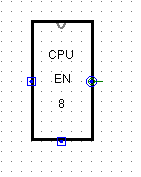
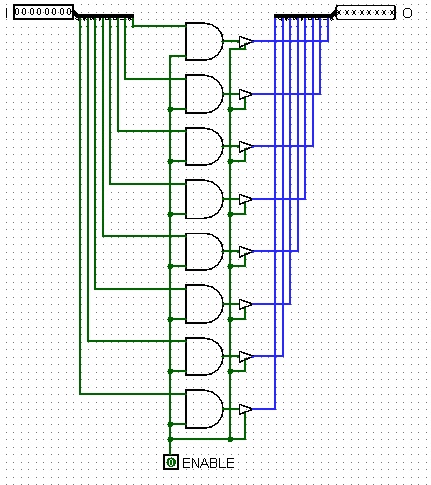


Fig.3: Enable circuit

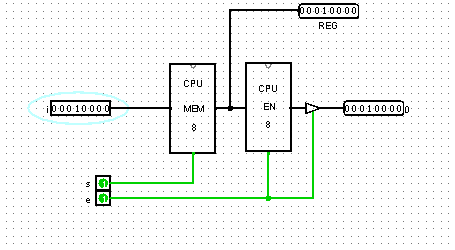
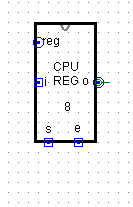
 

Fig.3: 8-bit register circuit

*WORKING: -* So, basically whenever ‘s’ is equal to one then the input register data get transferred to the 8-bit memory. To access the memory, an enable circuit is incorporated. Whenever ‘e’ (enable) is equal to 1 then the value in memory can be accessed.

* **DECODERS: -**

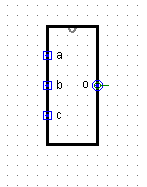
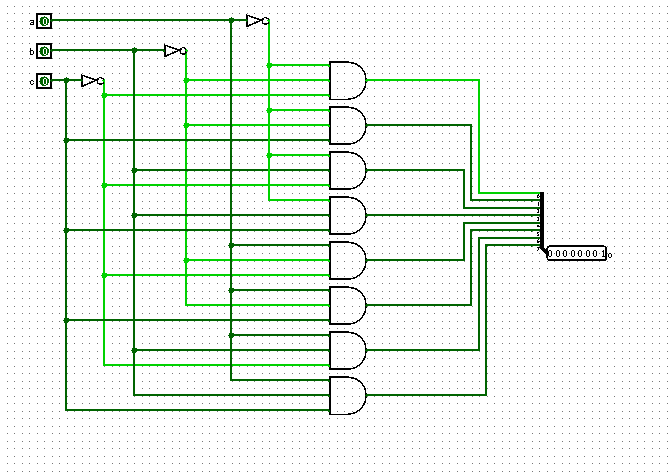
.

Fig.4: 3\*8 decoder circuit

*WORKING: -* So in a 3\*8 decoder we have 3 inputs and 8 outputs. The decimal number represented by the binary logic of input will decide which output port of the decoder circuit will be high.

* **ARITHMATIC LOGIC UNIT (ALU) COMPONENTS**

1. *ADDER CIRCUIT: -*

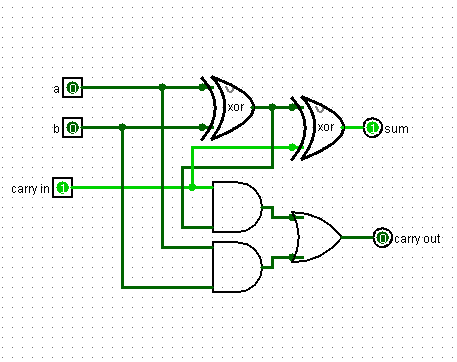


Fig.5 Full Adder for 1-bit

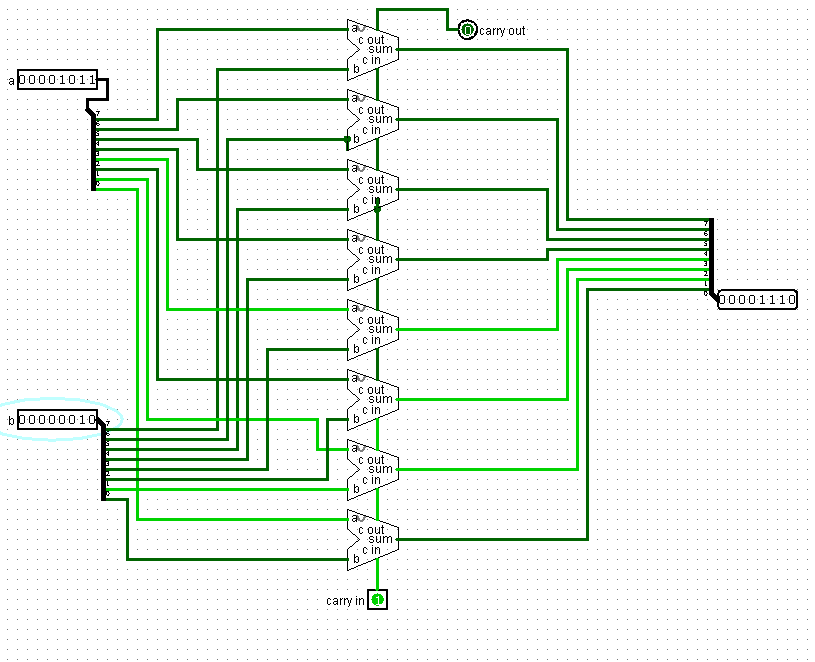


Fig.6: Full Adder for 8-bits

The basic and the main component of a ALU is the Adder circuit. In Fig.5 I have made a full adder circuit for one bit which is used to make an 8-bit full adder circuit as shown in Fig.6.

1. *SHIFT RIGHT CIRCUIT (SHR): -*

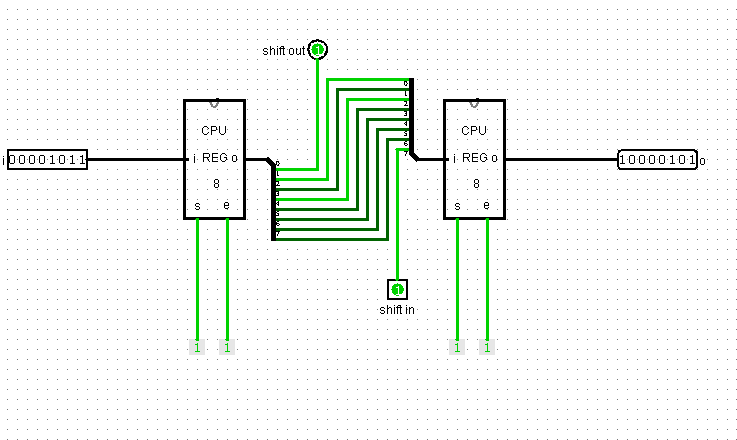


Fig.7 SHR Circuit

The SHR circuit shifts the value in register towards the right and the least significant bit is dropped to the carry out. The ALU made can perform SHR operation only on one of the registers.

1. *SHIFT LEFT CIRCUIT (SHL): -*

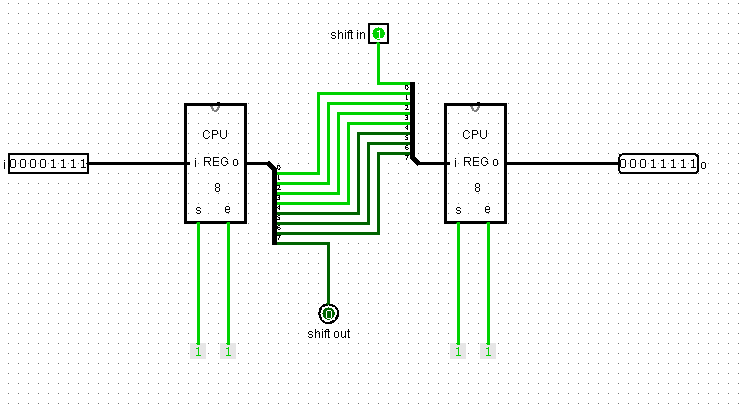


Fig.8 SHL Circuit

The SHL circuit shifts the value in register towards the left and the most significant bit is dropped to the carry out. The ALU made can perform SHR operation only on one of the registers.

1. *LOGIC OPERATIONS: -*

* *NOT OPERATION: -*

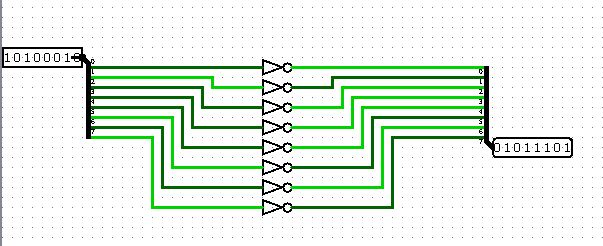


Fig.9: Not Operation Circuit (8-bit)

Similarly, as SHL and SHR, this operation can be performed on only one of the registers. This combinational circuit is formed by using 8 not gates.

* *AND OPERATION: -*

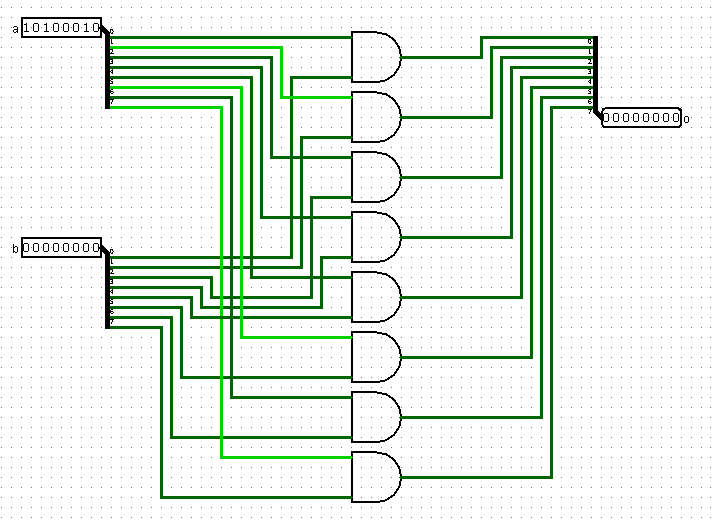


Fig.10: And operation Circuit (8-bit)

This circuit is formed by using 8 and gates and the operation can be performed on two registers.

* *OR OPERATION: -*

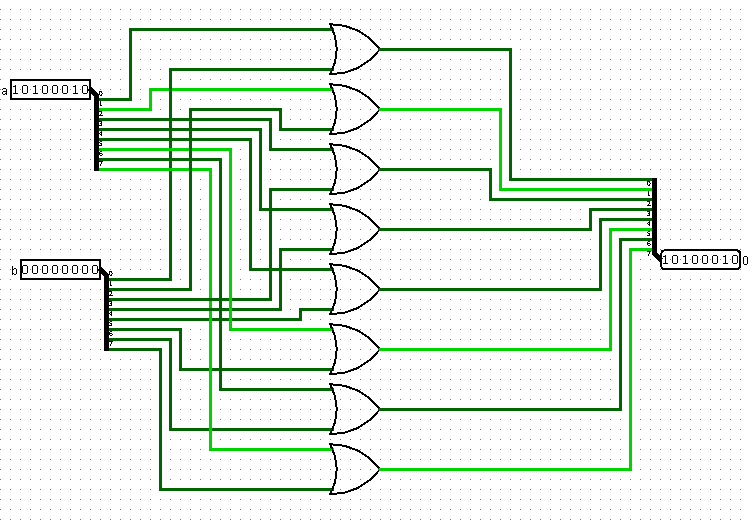


Fig.11: OR operation Circuit

This circuit is formed by using 8 OR gates and the operation can be performed on two registers.

* *XOR OPERATION: -*

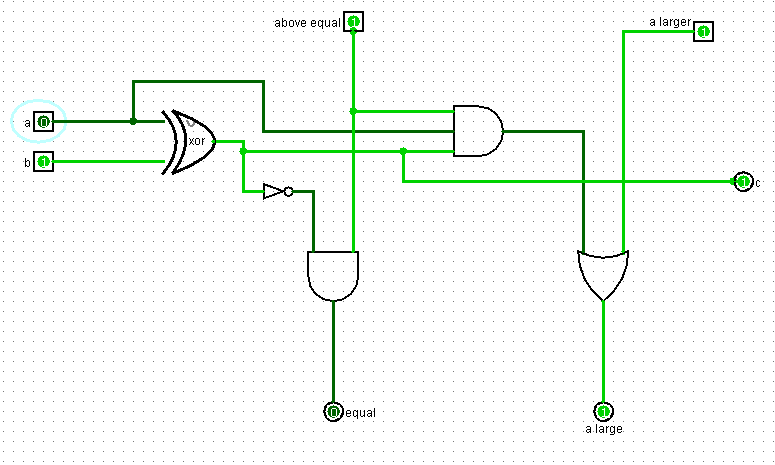


Fig.12: XOR Circuit (1- bit)

This circuit performs XOR operation and also checks if first input to the gate is larger or equal to the second input. To check for equality of bits, the output of XOR operation can be used. For equal inputs, output of XOR operation is zero. For checking if ‘a’ is greater than ‘b’ or not, there are two requirements, first is that both the inputs should be different, that is the output of the XOR gate should be 1 and the ‘a’ bit itself has to be 1. Therefore, an AND gate is required for this purpose.

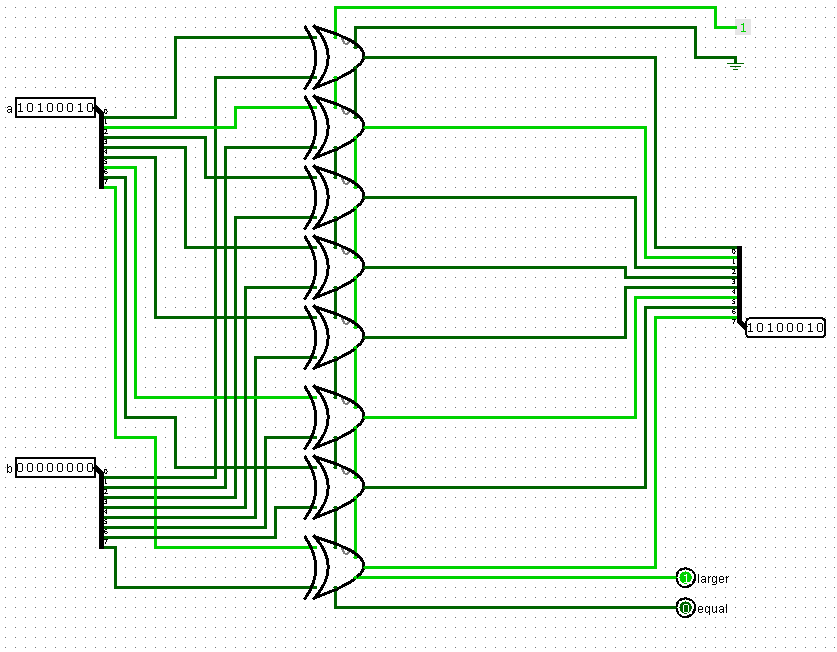


Fig.13: 8-Bit XOR Operation Circuit

* *COMPLETE ALU CIRCUIT: -*

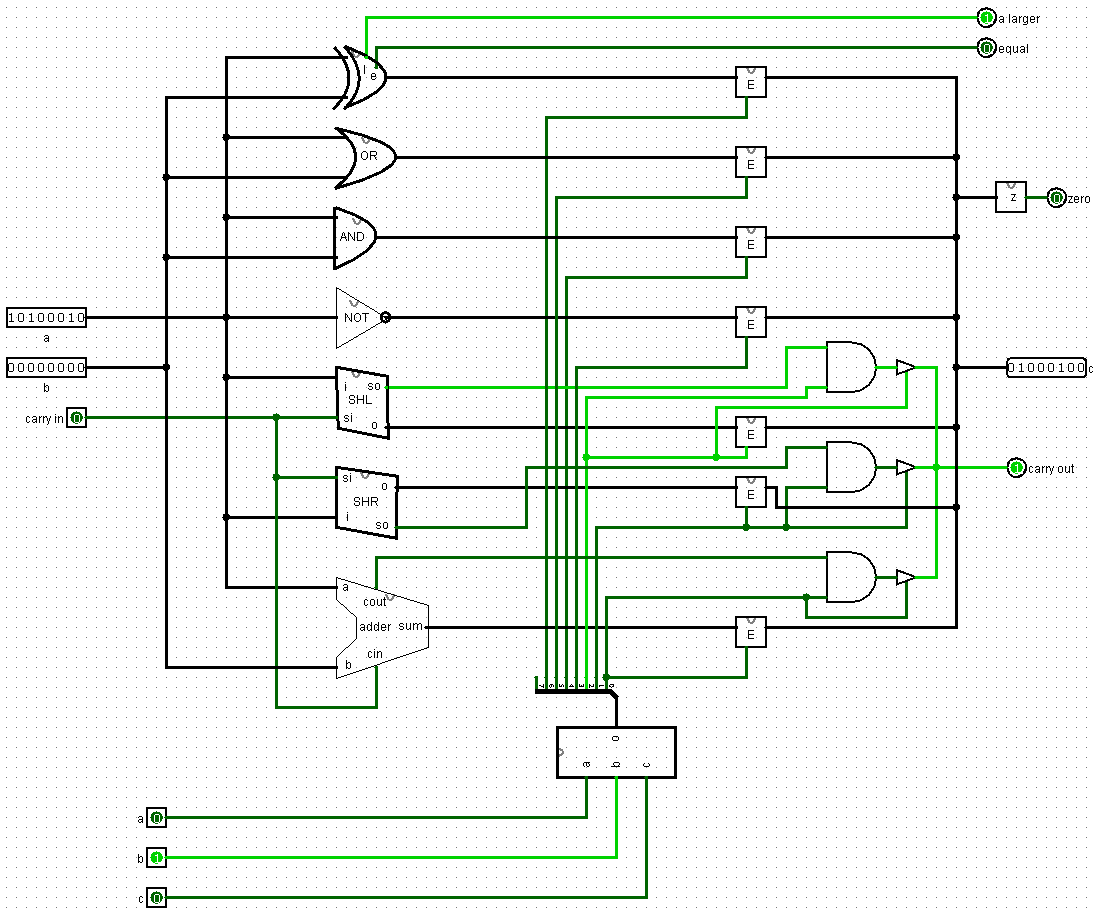


Fig.14: ALU Circuit

In this circuit ‘a’ and ‘b’ are the input registers and ‘carry in’ is also input. Operation to be performed is selected by using a 3\*8 decoder. Here you can see two registers ‘Z’ and ‘E’. Output of ‘Z’ register indicates if the output of the ALU circuit is zero. ‘E’ register is similar to the 8-bit enable circuit, only we have packed the circuit in a smaller box. 8th pin output of the decoder is unused as there are only seven functions in this circuit.

* **CLOCK**

To move data via the bus, we need first to enable the output of one and only one register, so that its electricity can travel through the bus to the inputs of other registers. Then, while the data is on the bus, we want to turn the set bit of the destination register on and off. Since the destination register captures the state of the bus at the instant that the set bit goes off, we want to make sure that it goes off before we turn off the enable bit at the first register to make sure that there are no problems.

* *CONCEPT BEHIND THE CLOCK: -*

If we show the original clock output (clk) and the delayed clock output (clk d) on a graph, they will look like this:

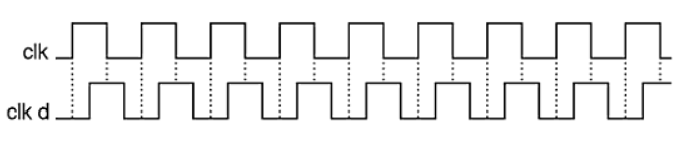
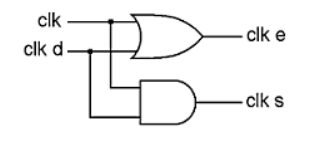


Fig.15: Timing Diagram

Here ‘clk’ represents clock signal and ‘clk d’ represents delayed clock signal which is delayed about one quarter of a cycle. If we apply the following logic to the ‘clk’ and ‘clk d’ then we will have such kind of waveforms: -



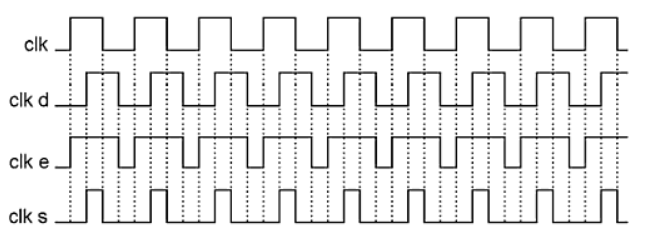


Fig.16: Timing Diagram

Here ‘clk e’ represents enable clock which is be used to enable the registers and ‘clk s’ represents set clock which is used to set the registers. Using these clocks, we can see that our purpose is fulfilled that is, enable clock is covering the entire set clock, which was our purpose.

For implementing this logic in Logisim, a JK Flip Flop is used.

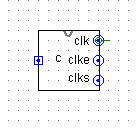
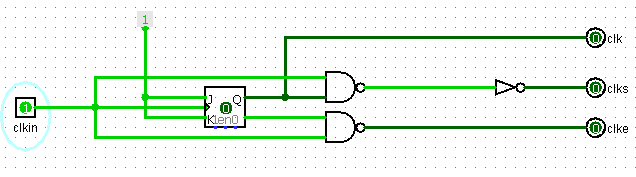
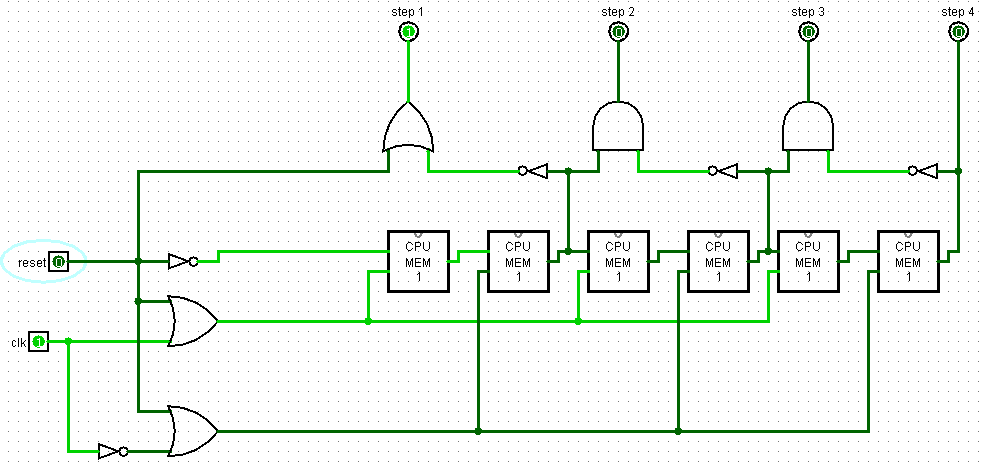


Fig.17: Clock Circuit

In this circuit as we can see J = K = 1, that is, the Flip Flop will toggle (positive edge triggered Flip Flop).

The new clock generated will have quarter the speed of original clock, but it will serve our purpose.

* **SEQUENCE COUNTER (STEPPER)**



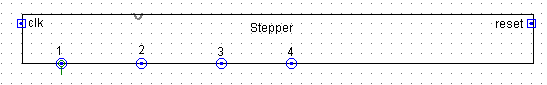


Fig.18: Sequence Counter

This sequence counter includes four steps. It has four 1-bit memory units.

*WORKING: -*

Clock signal is connected to the odd numbered memory units whereas the inverted clock signal is applied to the even numbered memory units. The input to the first memory unit is always high (until reset is high). So, when the clock goes high (inverted clock goes low) then the output of the memory unit will become high but it will not set the second memory unit because it gets set on inverted clock which is low for the time being. This will set step 1 of the stepper high as step 1 is connected to the inverted output of the second memory unit. When clock goes low, inverted clock goes high setting the second memory unit and this will turn off the step 1 (step 1 is connected to inverted output of second memory). Step 2 will become high as the output of second memory unit is high and the output of third memory unit is low (signal coming to the AND gate of step 2 is the inverted output of third memory unit). This process will continue as the clock goes ticking until it reaches step 4, which will stay on for whole time. Stepper can be reset using the reset input.

* **CPU AND CONTROL SECTION (FOR ALU OPERATIONS)**

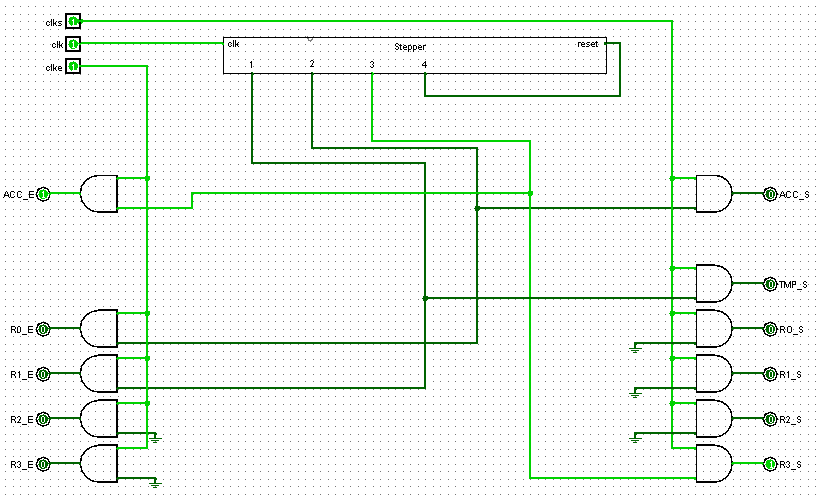


Fig.19: Control Section

Here R0\_E, R1\_E, R2\_E, R3\_E are the enables for CPU registers and ACC\_E is the enable signal for Accumulator (Accumulator register is similar to CPU register). Similarly, R0\_S, R1\_S, R2\_S, R3\_S are the set logics for registers. TMP\_S is the set logic for temporary register (similar to 8-bit memory circuit).

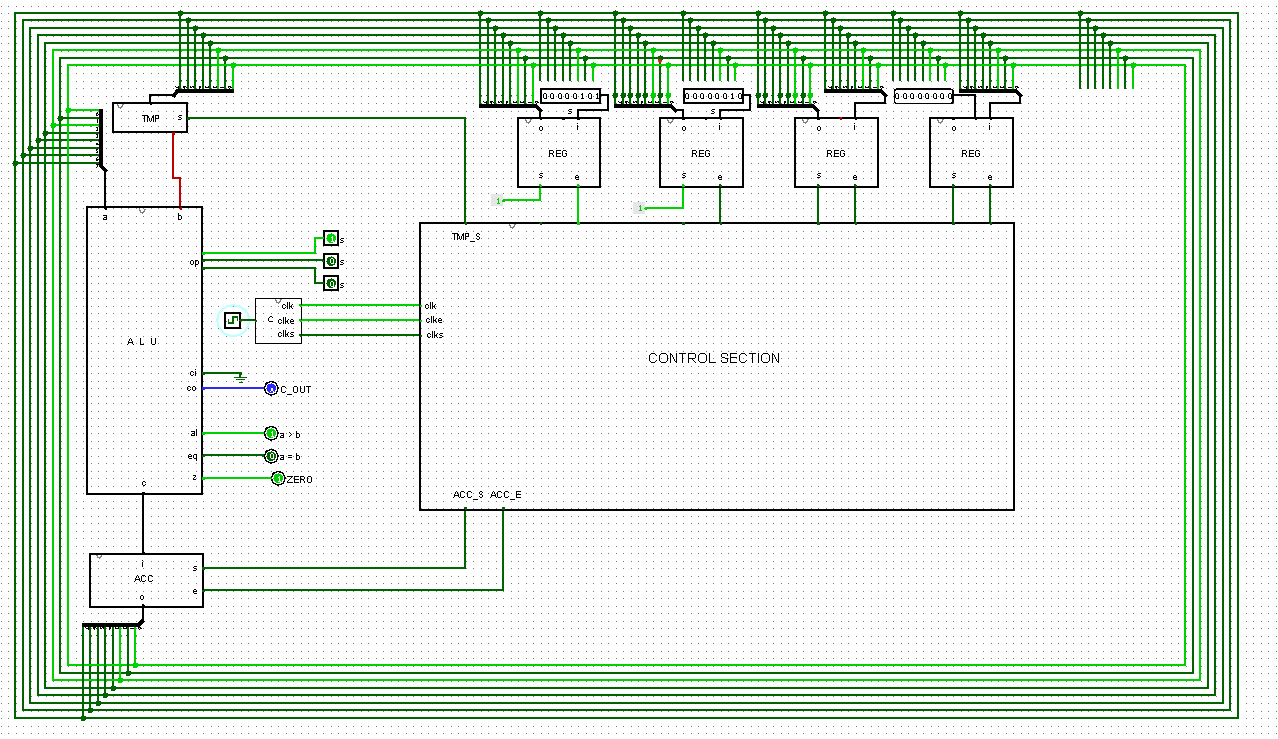


Fig.20: Complete Circuit

The green wires surrounding the Circuit is the BUS.

The control section is made such that the Bus takes input from R0 and R1 registers and fed the result of ALU operation into R3.

*WORKING: -*

STEP 1 OF SEQUENCE COUNTER: - On first clock cycle, clk e goes high which enable input of R1 goes high. On next clock pulse clk s goes high setting the set input of temporary register high which will transfer the data from R1 to the temporary register via BUS and then the data goes to the ‘b’ input of ALU. On next clock pulse clk s goes low and on the next clock pulse clk e goes low.

STEP 2 OF SEQUENCE COUNTER: - Similarly as step 1, going through 4 clock cycle, the control section first enables the R0 transferring data into ‘a’ input of ALU via BUS and then sets the Accumulator register where the result of ALU operation gets stored.

STEP 3 OF SEQUENCE COUNTER: - Going through four clock cycles, control section first enables the Accumulator register and sets the R3 register which will transfer the data of accumulator in the R3 register. And then the clk s goes low, then the clk e goes low.

STEP 4 OF SEQUENCE COUNTER: - Step 4 of sequence counter is connected to the reset pin which will reset the sequence counter.