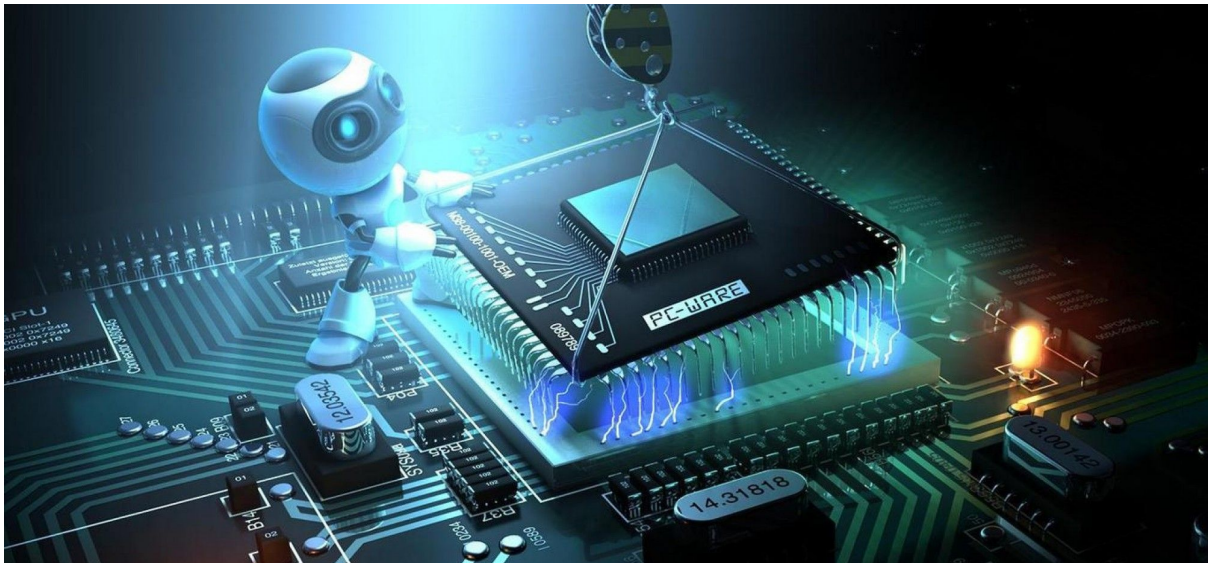
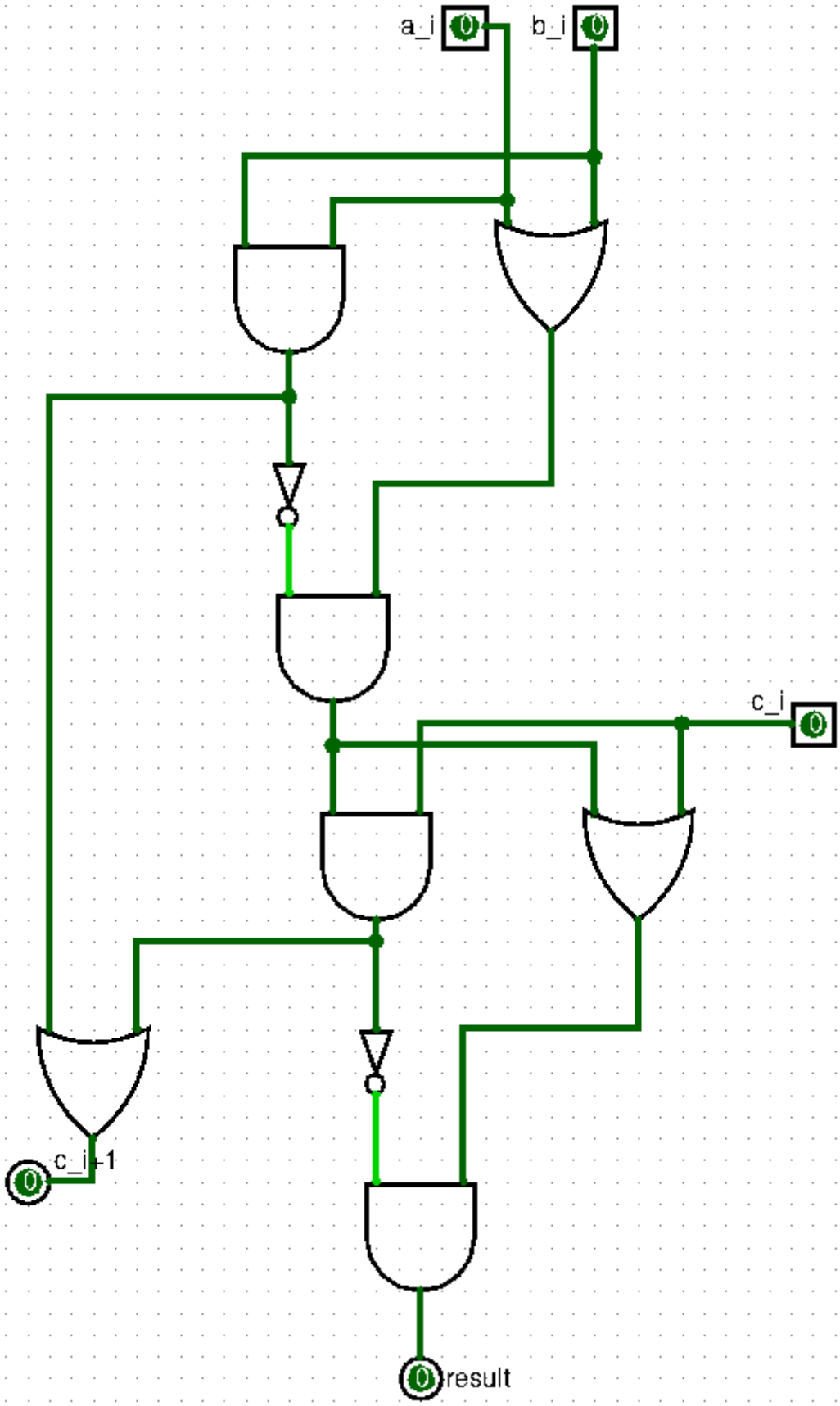



# CSE-331 Homework 3 Report

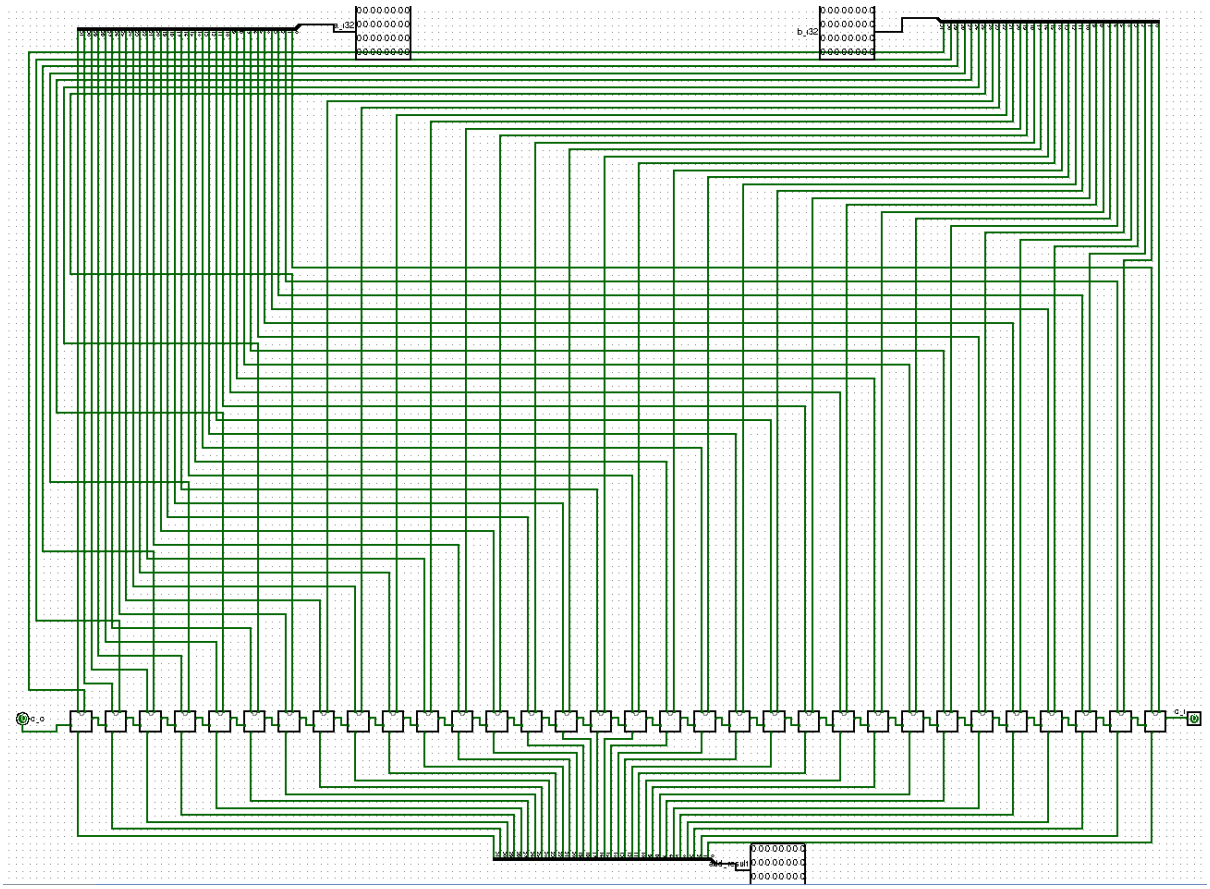
| | |
|---|---|
| Computer Organization | HomeWork Lecture |
| Created | @Dec 20, 2020 |
| Number | 171044036 |
| Property | Baran Hasan Bozduman |

In this project i designed adder shifter by myself it also works for overflowed numbers. I can start with introduce my components

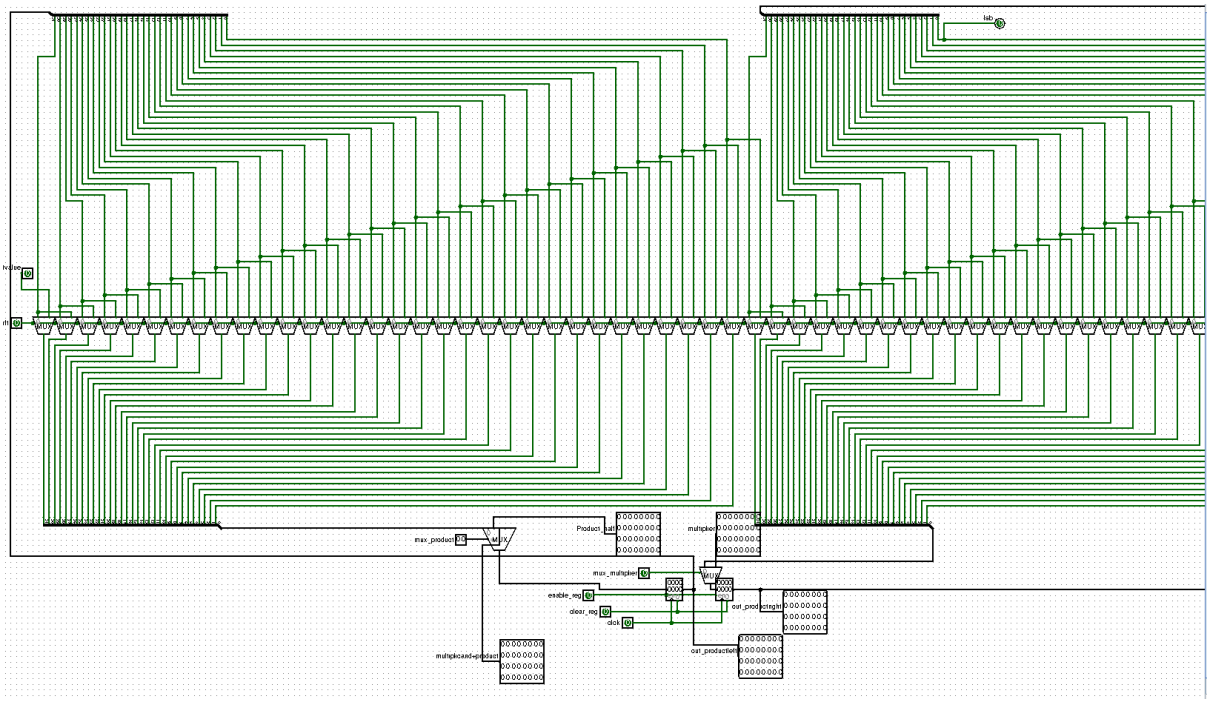
**1-Bit Adder**


a_i
b_i
c_i
c_i-1
result

## 32-Bit Adder

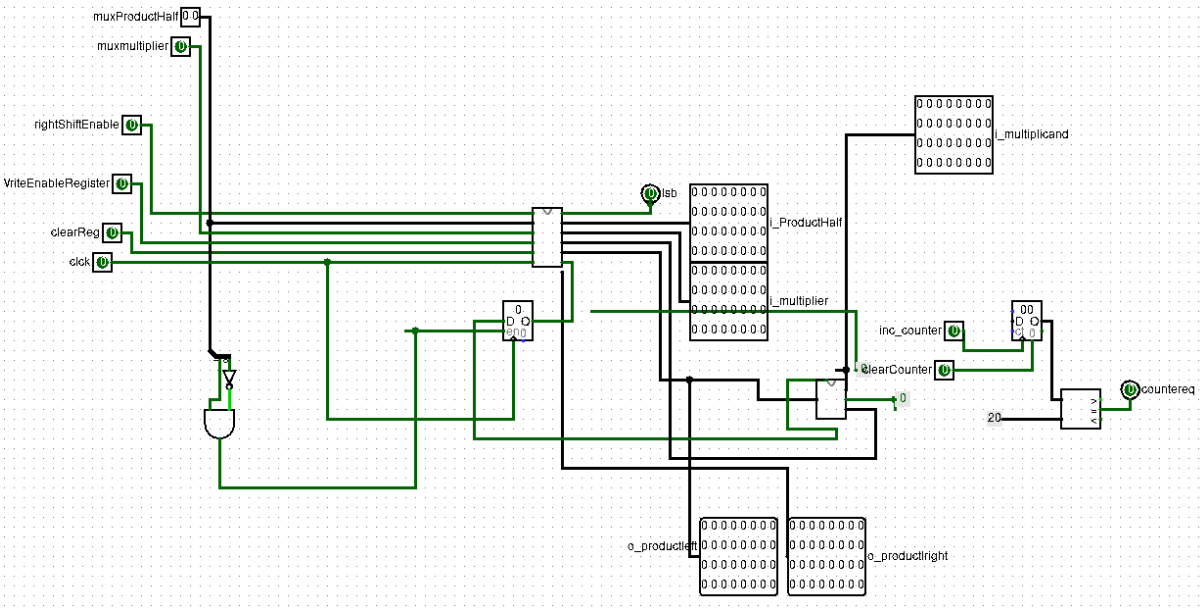


## Shifter Register



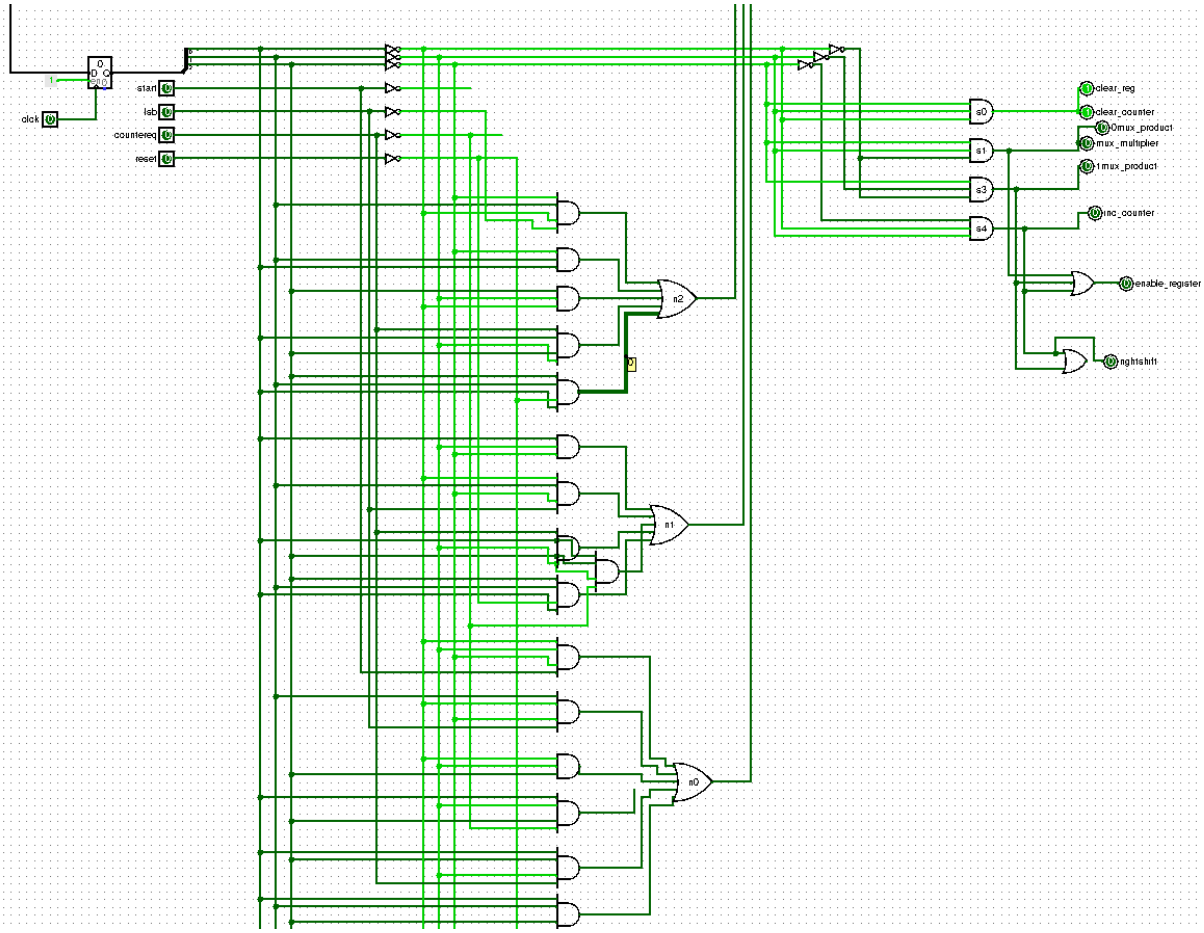
As you see above i designed a shifter with muxes so i can arrange the signals with using control unit i can decide at the curent state it will make a shift or not if it will, right shift can be 1 and accouding to overflow bit i send shift value so we can also do arithmetic shift by deciding muxes when we come to register inputs as default it gets shifted values, and also we use initial values and summation result for left register

**Datapath**



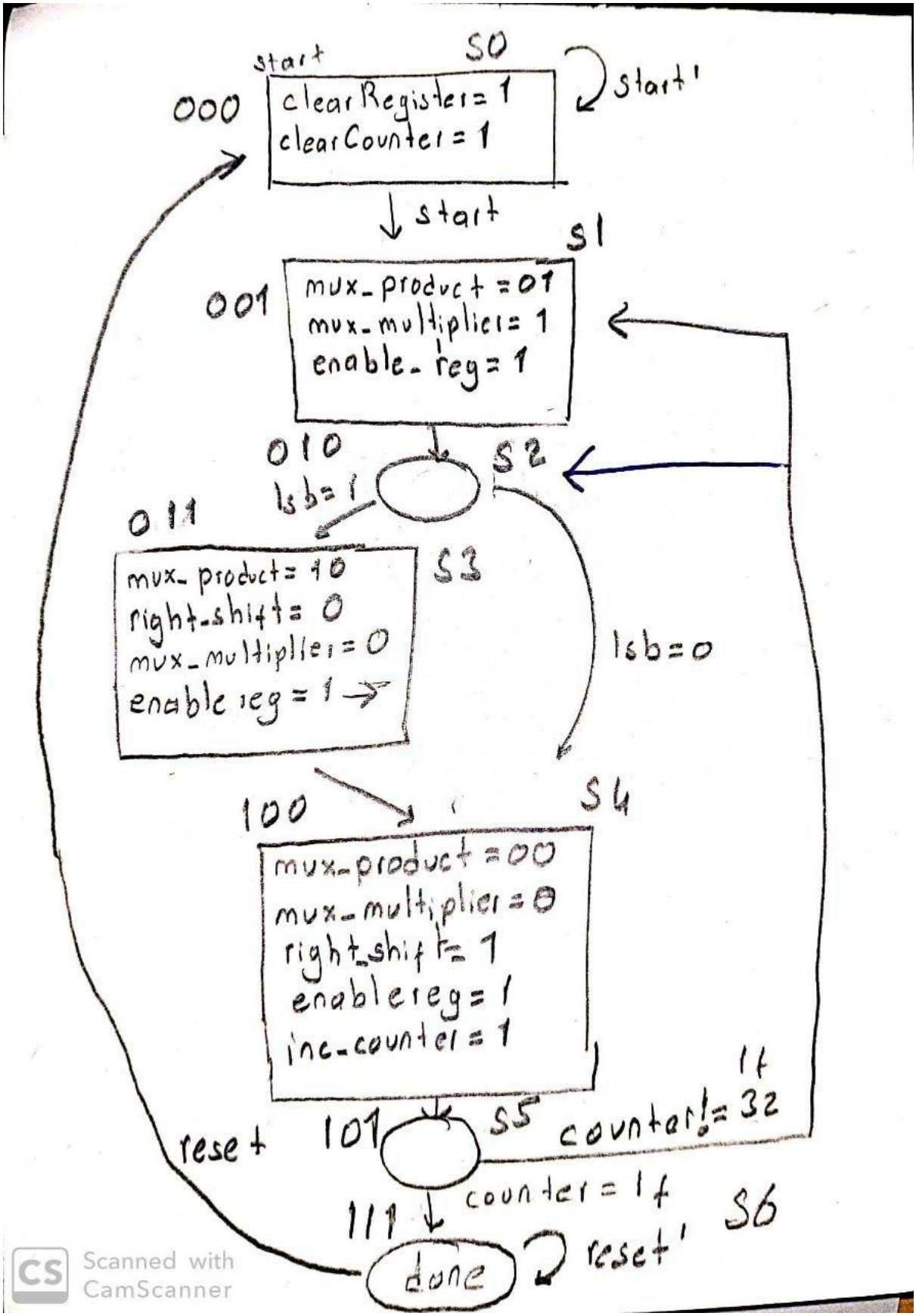

You can see extra registerin datapath also for the overflow situations so if to pretend overflow situations i keep the carry out bit in a gregisster so when we make a shift the carry out bit comes beginning of the number

**Control Unit**

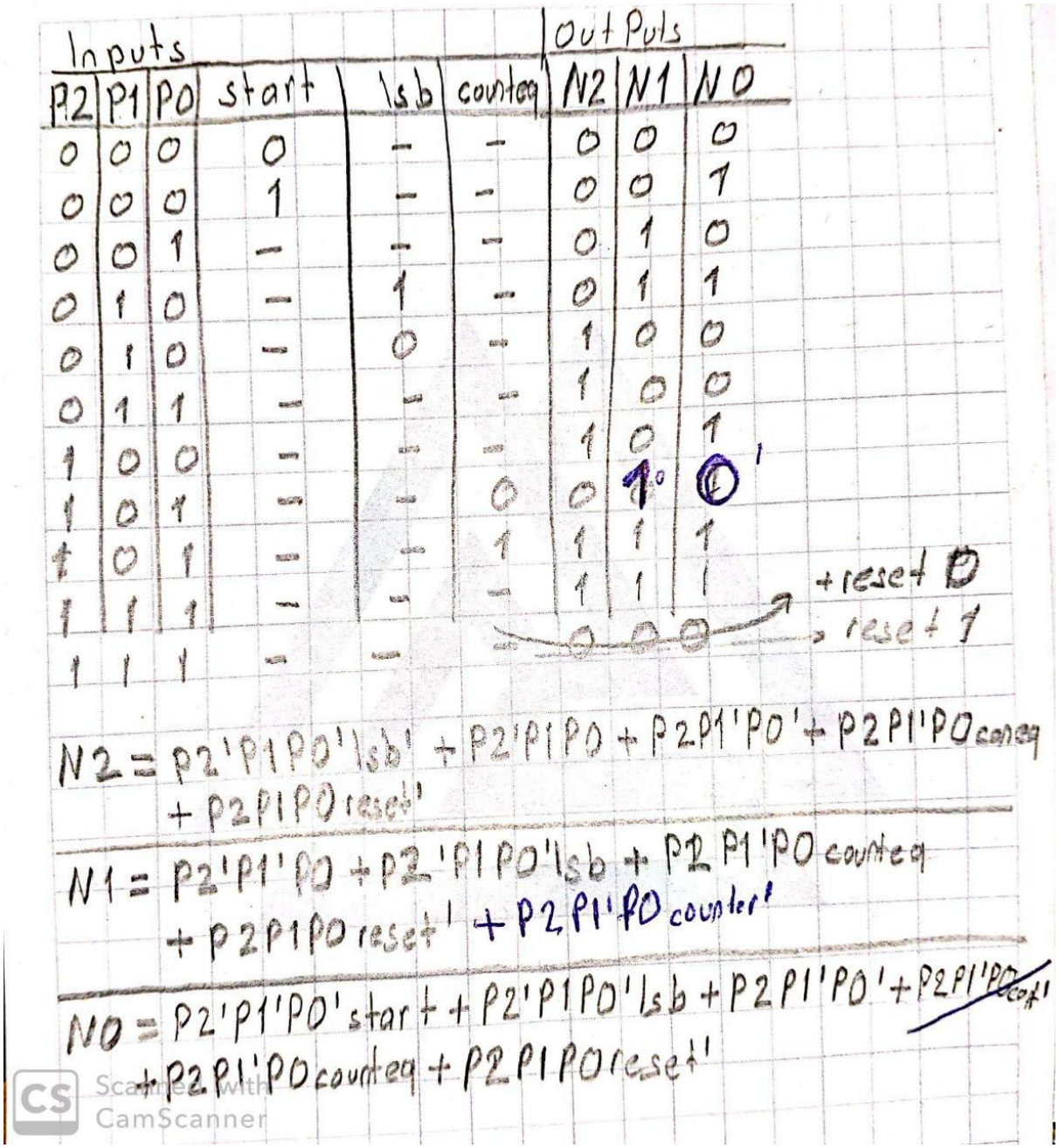



I arranged the signals according to my states

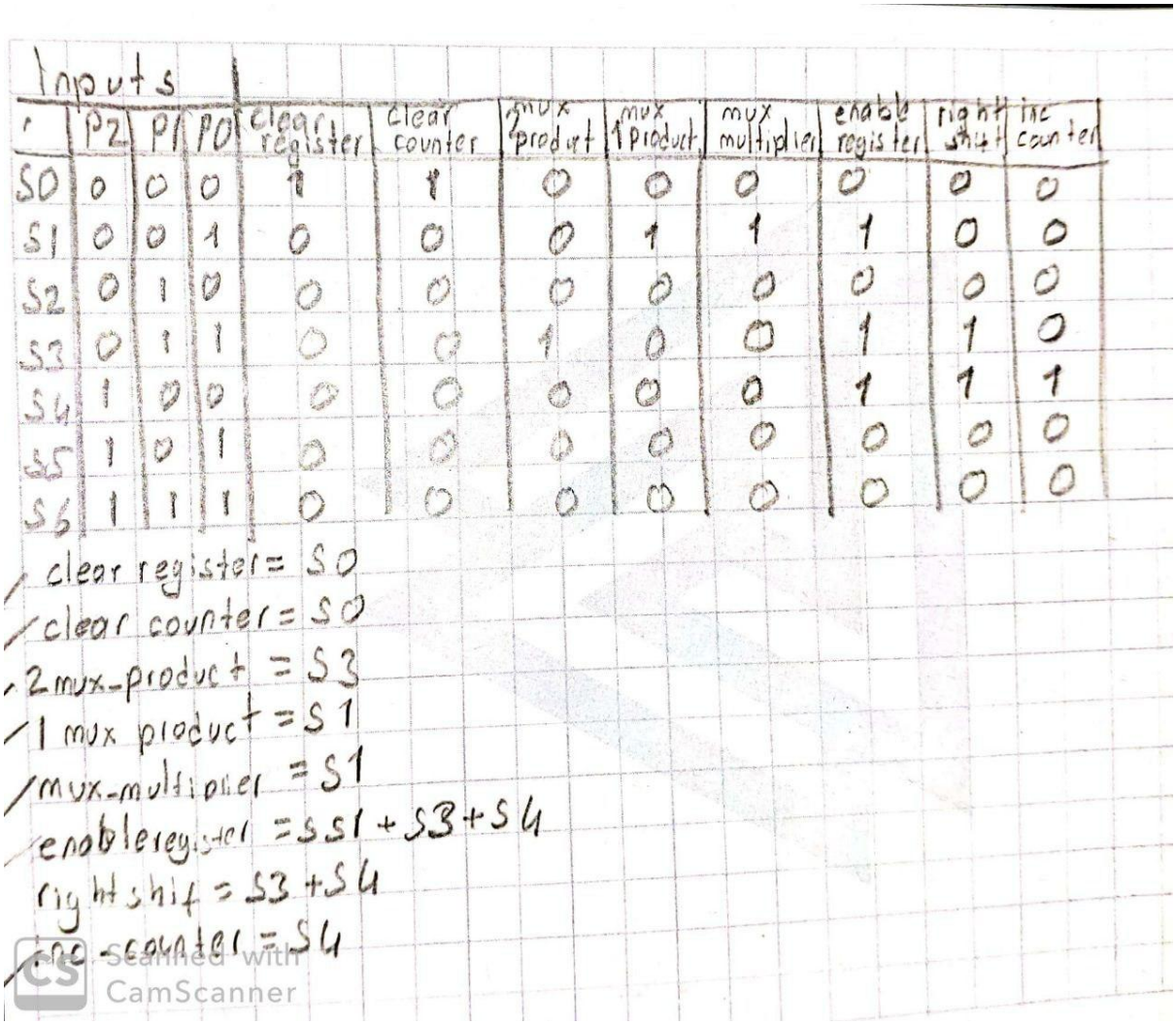
**Finite State Machine And Program Flow**




to determine next state here is my truth table below




And to determine signals according to what state are we here my other table



Inputs

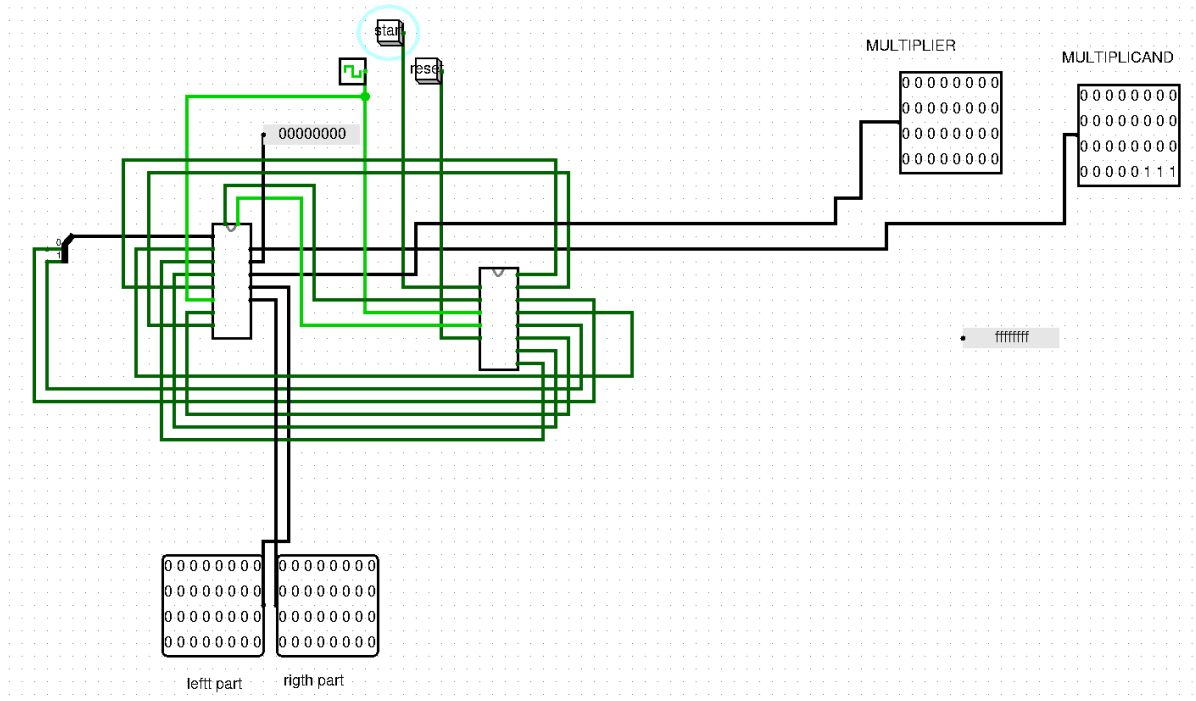
| | P2 | P1 | P0 | clear register | clear counter | 2mux product | 1mux Product | mux multiplier | enable register | right shift | inc counter |
|---|---|---|---|---|---|---|---|---|---|---|---|
| S0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| S1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| S2 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| S3 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| S4 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| S5 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| S6 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

clear register = S0
clear counter = S0
2mux-product = S3
1 mux product = S1
mux-multiplier = S1
enableregister = S1 + S3 + S4
right shift = S3 + S4
inc-counter = S4

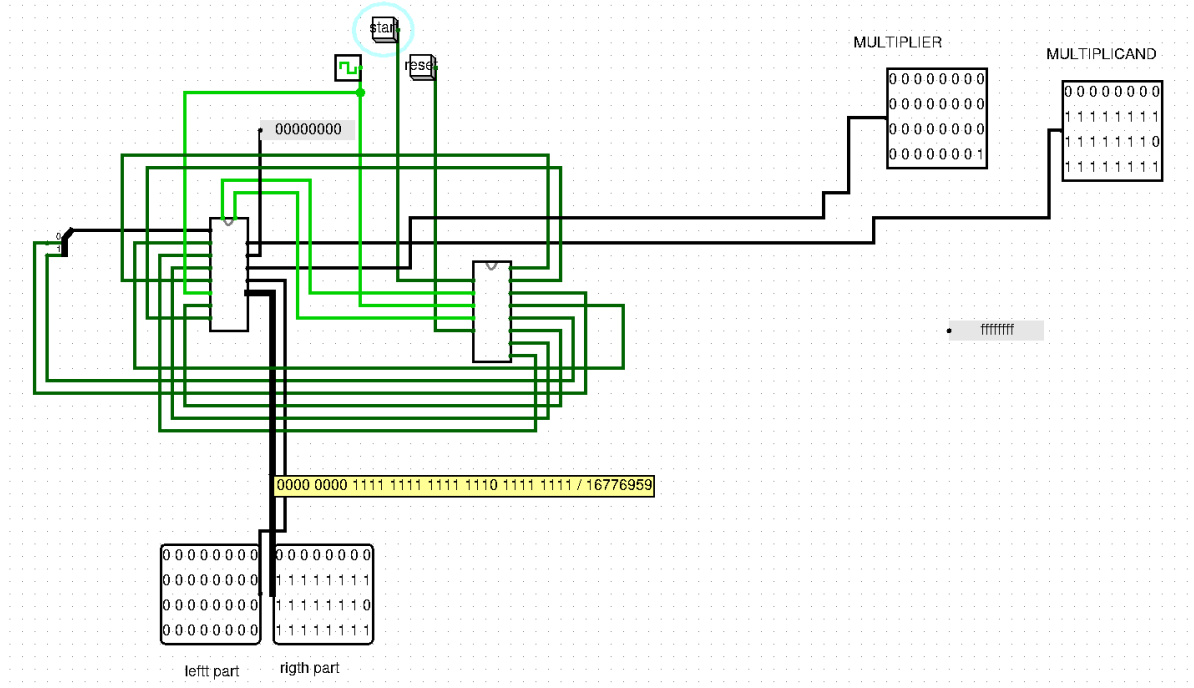


# TESTS

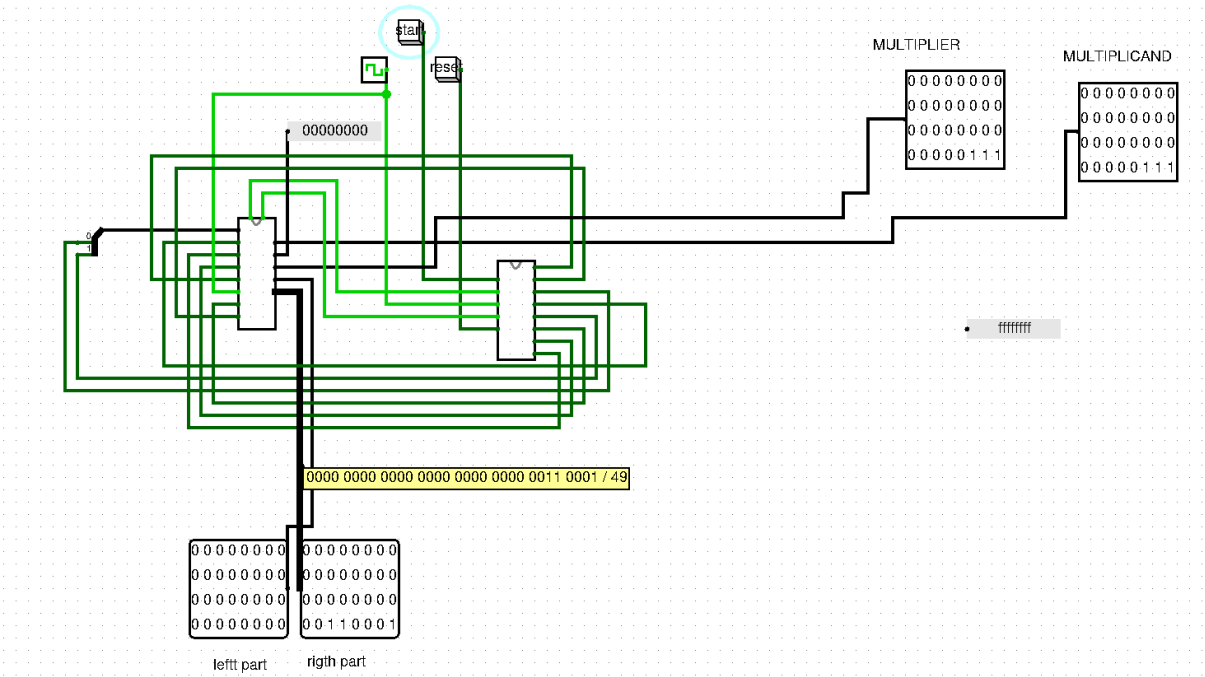
**1- ONE SIDE ZERO**



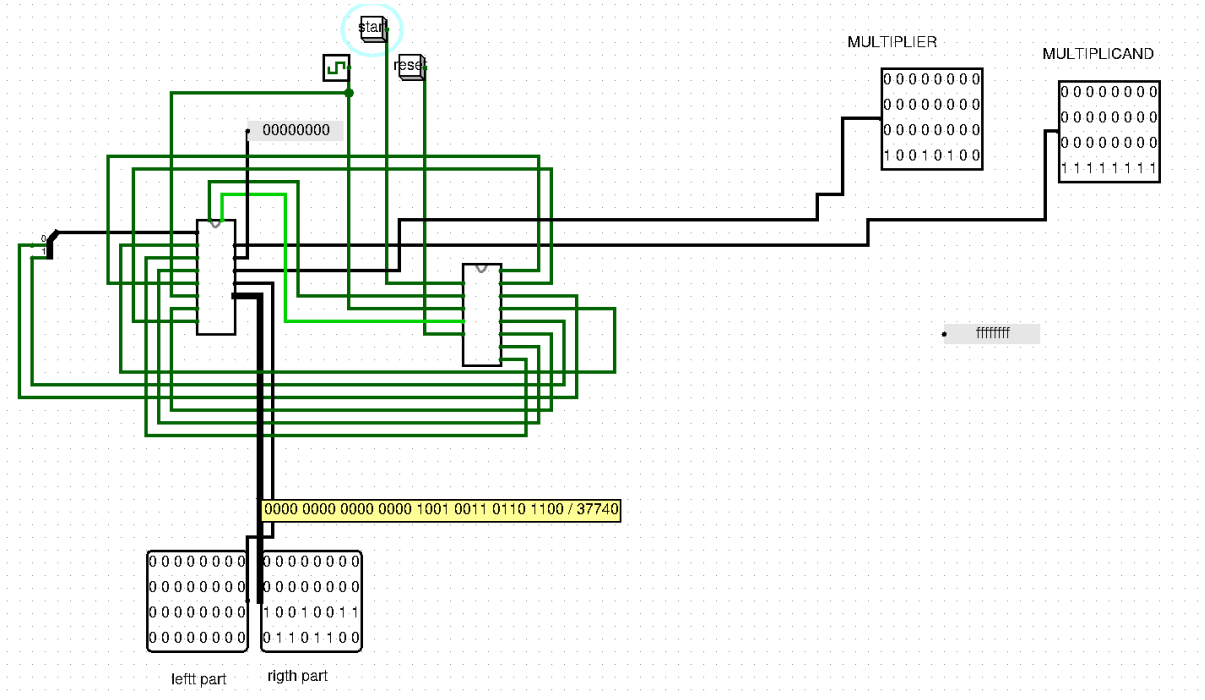

## 2-ONE SIDE 1



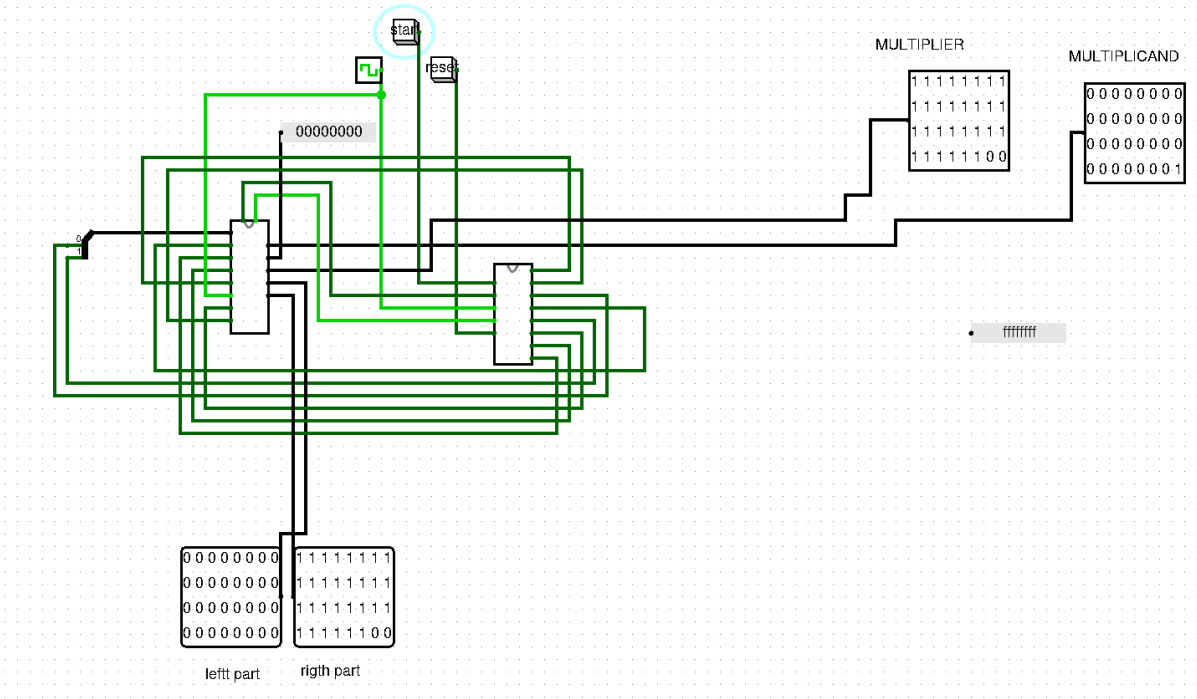

## 3-THERE SOME NUMS BOTH SIDE




## 4-TEST CASE




## 5-TEST CASE




## 6- TEST OVERFLOW