Title: Design and Implementation of SAP-1 Computer using logisim

Roll: 1607106

Objectives:

- To be introduced with all the crucial ideas behind computer operations
- To develop a basic understanding of how a computer works, interacts with memory and other parts of the system like input and output
- To develop a simplified computer architecture - "SAP (Simple - As - Possible" for educational purpose

Introduction:

The SAP (Simple - As - Possible) computer is the first stage in the evolution toward modern computers. Although a primitive, SAP-1 is a big step for beginner. The main purpose of SAP-1 is to introduce all the crucial ideas behind computer operation.

Architecture:

Figure - 1 shows the architecture of SAP-1, a bus organized computer. All register outputs to the W bus

are three-state, this allows orderly transfer of data. All other registers are two state; these outputs continuously drive the boxes they are connected to.

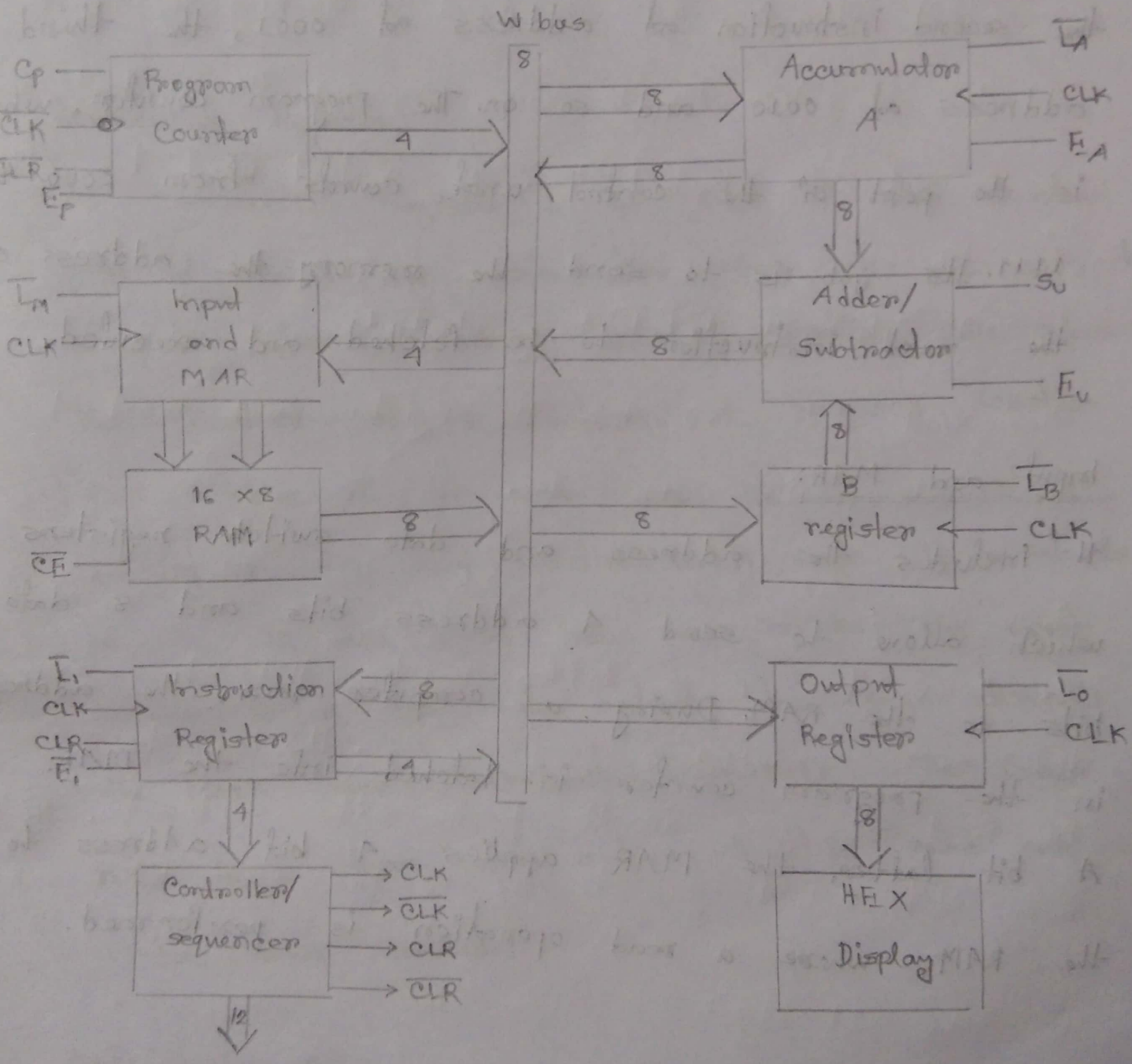


Figure-1: SAP-1 Architecture

Program Counter:

The program is stored at the beginning of the memory with the first instruction at binary address at 0000, the second instruction at address at 0001, the third address at 0010 and so on. The program counter, which is the part of the control unit, counts from 0000 to 1111. Its job is to send the memory the address of the next instruction to be fetched and executed.

Input and MAR:

It includes the address and data switch registers which allow to send 4 address bits and 8 data bits to the RAM. During a computer run, the address in the program counter is latched into the MAR. A bit latter, the MAR applies 4 bit address to the RAM, where a read operation is performed.

The RAM:

The RAM is a 16x8 static TTL RAM. During a computer run, the RAM receives the 4 bit addresses

from the MAR and a read operation is performed. In this way, the instruction or data word stored in the RAM is placed on the W bus for use in some other part of the computer.

Instruction Register:

The instruction register is the part of the control unit. To fetch an instruction from the memory, the computer does a memory read operation. This places the contents of the addressed memory location on the W-bus. At the same time, the IR is set up for loading on the next positive clock edge. The contents of the IR are split into two nibbles. The upper nibbles is a two state output that goes directly to the block labeled "Controller-Sequencer". The lower nibble is a three state output that is read onto the W-bus when needed.

Controller-Sequencer:

Before each computer run, $\overline{CLR}$ signal is send to the PC and CLR signal to the IR. This resets the PC to 0000

and wipes out the last instruction in the IR. A clock signal CLK is sent to all buffer registers, this synchronizes the operation of the computer. The 12 bits that come out of CS from a word controlling the rest of the computer. The 12 wires carrying the control word are called the control bus. The control word has the format:

$$CON = C_p E_p \overline{I}_M \overline{CE}\ \overline{I}_1 \overline{E}_1 \overline{I}_A E_A\ S_U E_U \overline{I}_B \overline{I}_O$$

This word determines how the registers will react to the next positive CLK edge.

Accumulator:

The Accumulator is a buffer register that stores immediate answers during a computer run. It has two output. The first one goes directly to the adder-subtractor. The three state output goes to the W bus when E_A is high.

The Adder-Subtractor:

When S_U is low, the sum out of the adder-subtractor is,

$$S = A + B$$

When S_U is high, the sum out of the adder-subtractor is

$$S = A - B$$

The adder-subtractor is asynchronous; this means that its contents can change as soon as the input words change. When E_U is high, these contents appear on the W bus.

B register:

The B register is also a buffer register. A low $\overline{LB}$ and positive clock edge load the word on the W bus into the B register. The two state output of the B register derives the B register.

Output Register:

At the end of a computer run, the accumulator contains the answer to the problem being solved. At this point, we need to transfer the answer to the outside world. This is where the output register is used. When E_A is high, $\overline{LO}$ is low, the next positive clock edge loads the word of the accumulator into the output register.

Hex Digit Display:

Using a seven segment display, shows the hexadecimal digit

corresponding to the four-bit input. If any of the inputs are not 0/1, then the display shows a dash ('-'). A separate one bit controls the display of the decimal point.

Instruction Set:

LDA:

LDA stands for 'load the accumulator'. A complete LDA instruction includes the hexadecimal address of the data to be loaded.

ADD:

A complete ADD instruction includes the address of the word to be added. For instance, ADD 9H means 'add the contents of memory location 9H to the accumulator contents'. The sum replaces the original contents of the accumulator.

SUB:

A complete SUB instruction includes the address of the word to be subtracted. SUB CH means 'subtract the contents of the memory location CH from the contents of the accumulator. The difference out of the adder-subtractor then replaces the original contents of

the accumulator.

OUT:

The instruction OUT tells the SAP-1 computer to transfer the accumulator contents to the output port. After OUT has been executed, the answer can be seen to the problem being solved.

HLT:

HLT instruction tells the computer to stop processing. HLT is completed by itself, we do not need to use RAM word using HLT because it does not involve memory.

Mnemonic	Op code
LDA	0000
ADD	0001
SUB	0010
OUT	1110
HLT	1111

Table-1: SAP-1 OP CODE

FETCH Cycle

The control unit generates the control words that fetch and execute each instruction. While each instruction is fetched

and executed, the computer passes through different timing states (T states), periods during which register contents change.

Ring Counter:

The ring counter has an output of

$$T = T_0 T_5 T_4 T_3 T_2 T_1$$

At the beginning of a computer run, the ring word is

$$T = 000001$$

Successive clock pulse produce ring words of

$T = 000010$

$T = 000100$

$T = 001000$

$T = 010000$

$T = 100000$

Then, the ring counter resets to 000001, and the cycle repeats. Each ring word represents one T state.

Address State.

The T_1 state is called the address state because the address in the program counter (PC) is transferred to the

memory address register (MAR) during the state. During the address state, E_p and $\overline{L_M}$ are active, all other control bits are inactive. So,

Control Word, CON = $C_p E_p \overline{L_M}\ \overline{CE}$ $\overline{L_I}\ \overline{E_I}\ \overline{L_A}\ E_A$ $S_U E_U \overline{L_B}\ \overline{L_O}$

= 0 1 0 1 1 1 1 0 0 0 1 1

Increment State

The T_2 state is called the increment state because the program counter is incremented. So, during the increment state,

Control Word, CON = $C_p E_p \overline{L_M}\ \overline{CE}$ $\overline{L_I}\ \overline{E_I}\ \overline{L_A}\ E_A$ $S_U E_U \overline{L_B}\ \overline{L_O}$

= 1 0 1 1 1 1 1 0 0 0 1 1

The C_p bit is active.

Memory State

The T_3 state is called the memory state because the addressed RAM instruction is transferred from the memory to the instruction register. During this state, $\overline{CE}$ and $\overline{L_I}$ are active and the

Control Word, CON = $C_p E_p \overline{L_M}\ \overline{CE}$ $\overline{L_I}\ \overline{E_I}\ \overline{L_A}\ E_A$ $S_U E_U \overline{L_B}\ \overline{L_O}$

= 0 0 1 0 0 1 1 0 0 0 1 1

Roll: 1607106

Execution Cycle

The next three (T_4, T_5, and T_6) are the execution cycle of SAP-1. The register transfers during the execution cycle depend on the particular instruction being executed.

LDA Routine

During T_1, T_2 & T_3 state, control words are same as Fetch cycle. During T_4 state $\overline{E_I}$ & $\overline{L_M}$ are active. During T_5 state $\overline{CE}$ and $\overline{L_A}$ are active. T_6 is a no-operation state. So,

	Control Words, $CON = C_P E_P \overline{L_M}\ \overline{CE}$	$\overline{L_I}\ \overline{E_I}\ \overline{L_A}\ E_A$	$S_U E_U \overline{L_B}\ \overline{L_O}$
T_4 =	0 0 0 1	1 0 1 0	0 0 1 1
T_5 =	0 0 1 0	1 1 0 0	0 0 1 1
T_6 =	0 0 1 1	1 1 1 0	0 0 1 1

ADD Routine

During T_4 state, $\overline{E_I}$ and $\overline{L_M}$ are active. $\overline{CE}$ and $\overline{L_B}$ are active during T_5 state. During T_6 state, E_U and $\overline{L_A}$ are active. So, Control Words,

	$CON = C_P E_P \overline{L_M}\ \overline{CE}$	$\overline{L_I}\ \overline{E_I}\ \overline{L_A}\ E_A$	$S_U E_U \overline{L_B}\ \overline{L_O}$
T_4 =	0 0 0 1	1 0 1 0	0 0 1 1
T_5 =	0 0 1 0	1 1 1 0	0 0 0 1
T_6 =	0 0 1 1	1 1 0 0	0 1 1 1

Roll: 1607106

SUB Routine

The SUB Routine is similar to the ADD Routine. During the T_6 state, a high S_U is sent to the adder – subtractor. So, the control words,

	CON = C_P E_P $\overline{L_M}$ $\overline{CE}$	$\overline{L_I}$ $\overline{E_I}$ $\overline{L_A}$ $\overline{E_A}$	S_U E_U $\overline{L_B}$ $\overline{L_O}$
T_4 =	0 0 0 1	1 0 1 0	0 0 1 1
T_5 =	0 0 1 0	1 1 1 0	0 0 0 1
T_6 =	0 0 1 1	1 1 0 0	1 1 1 1

OUT Routine

During T_4 state E_A and $\overline{L_O}$ are active. The T_5 and T_6 states are nops. So,

	CON = C_P E_P $\overline{L_M}$ $\overline{CE}$	$\overline{L_I}$ $\overline{E_I}$ $\overline{L_A}$ E_A	S_U E_U $\overline{L_B}$ $\overline{L_O}$
T_4 →	0 0 1 1	1 1 1 1	0 0 1 0
T_5 →	0 0 1 1	1 1 1 0	0 0 1 1
T_6 →	0 0 1 1	1 1 1 0	0 0 1 1

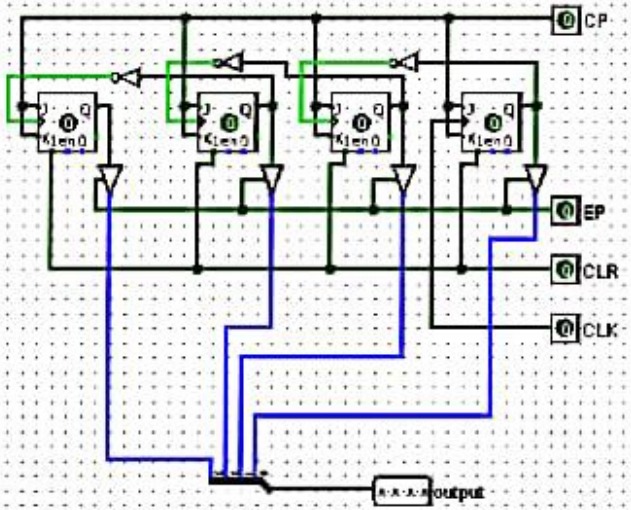
HLT

HLT does not require a control routine because no registers are involved in the execution of an HLT instruction.

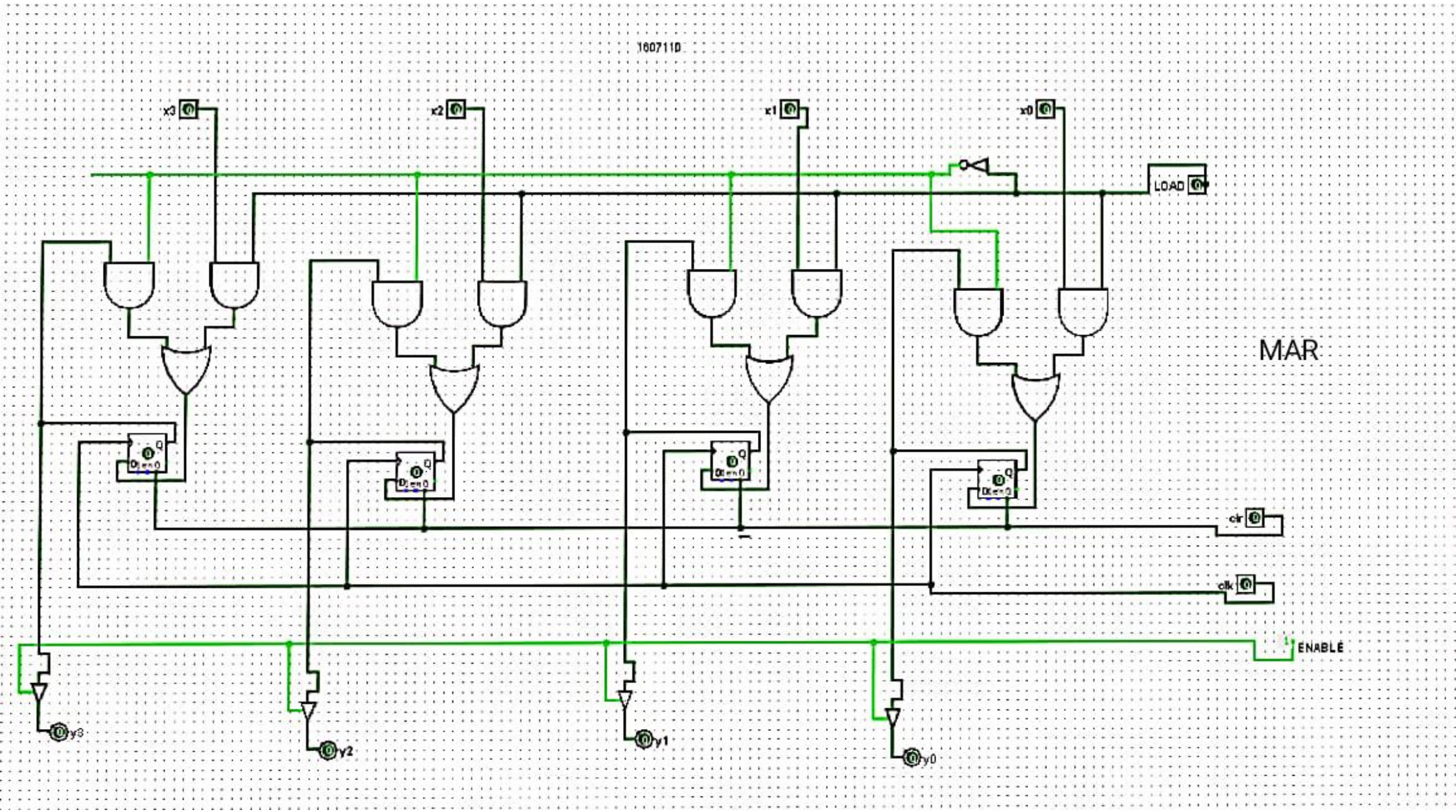
Circuit Diagram -

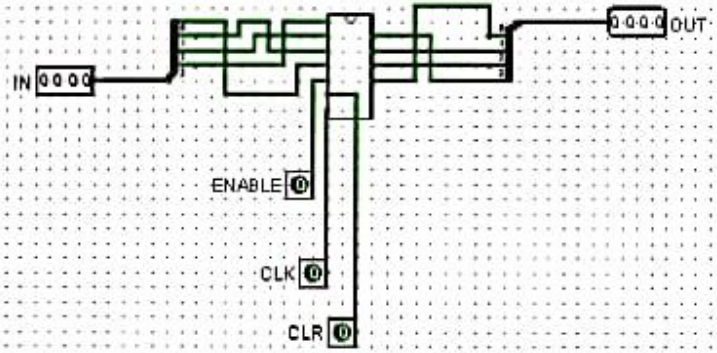
1) Program counter (PC) - Program counter is designed using JK flipflops, It has 4 inputs among them 2 input pin signal CP and EP comes from controller. Another two pins are connected to system clock and clear. It has four outputs which counts from 0 to 15.

2) Memory Address Register - MAR is four bits and used to address 16 locations in the RAM memory. It is a tri state 4 bit buffer register which loads the data from bus to register when LOAD is high then in next clock pulse data enters from bus to MAR. It's enable pin is always on so that there is no undefined state in the connected bus.



program counter


160711D
x3
x2
x1
x0
LOAD
MAR
clr
clk
ENABLE
y3
y2
y1
y0

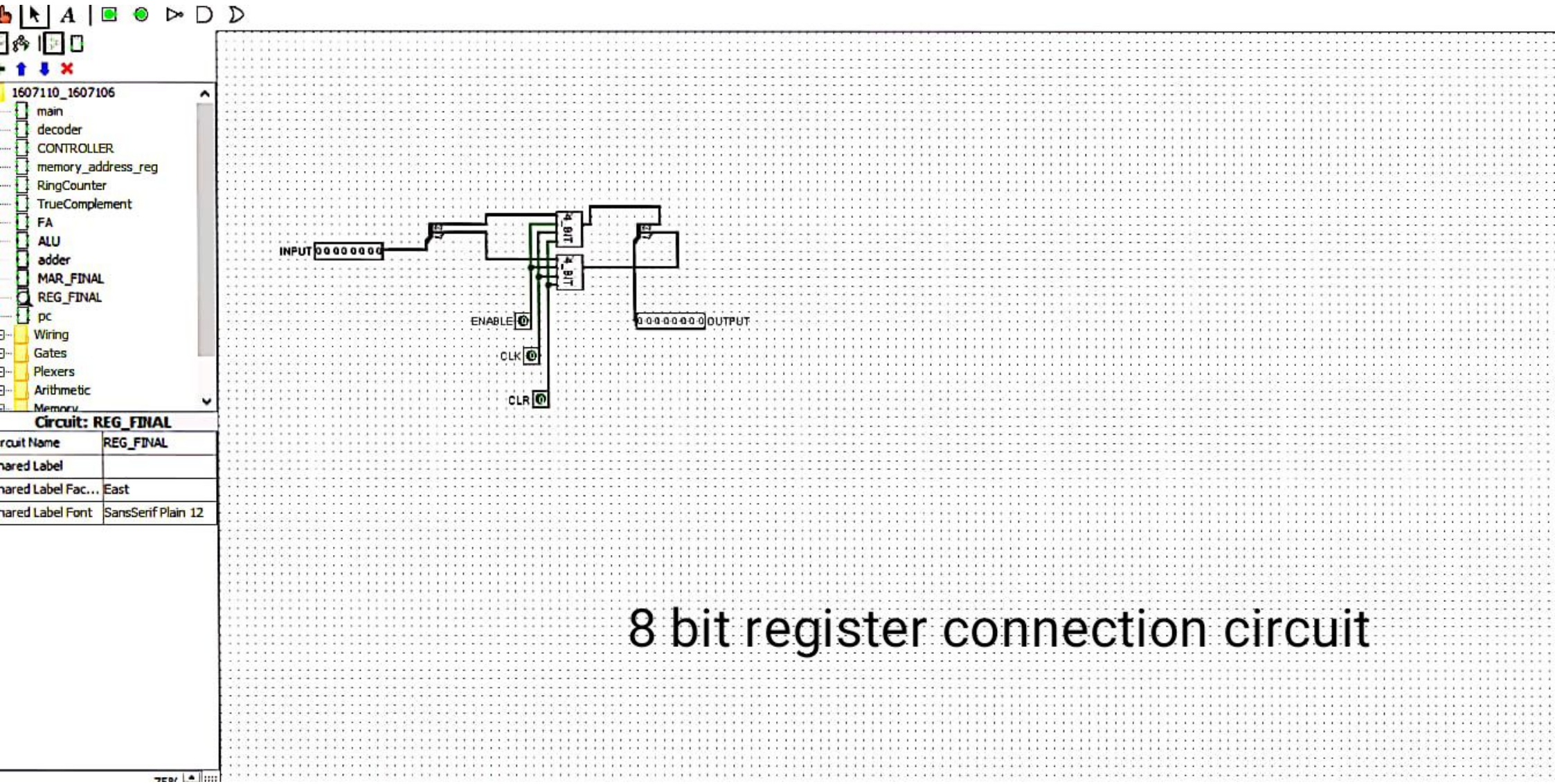


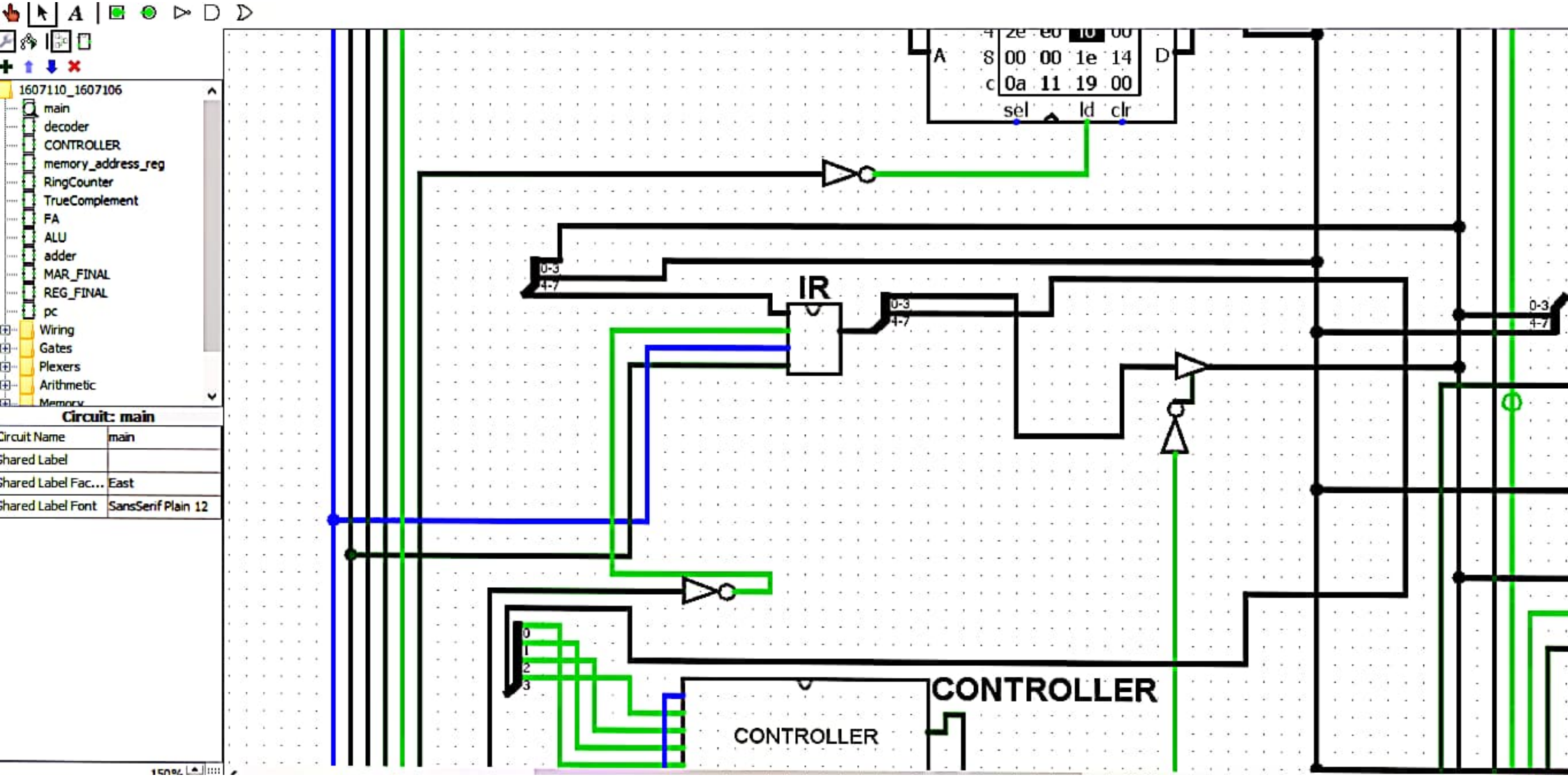
MAR connection circuit

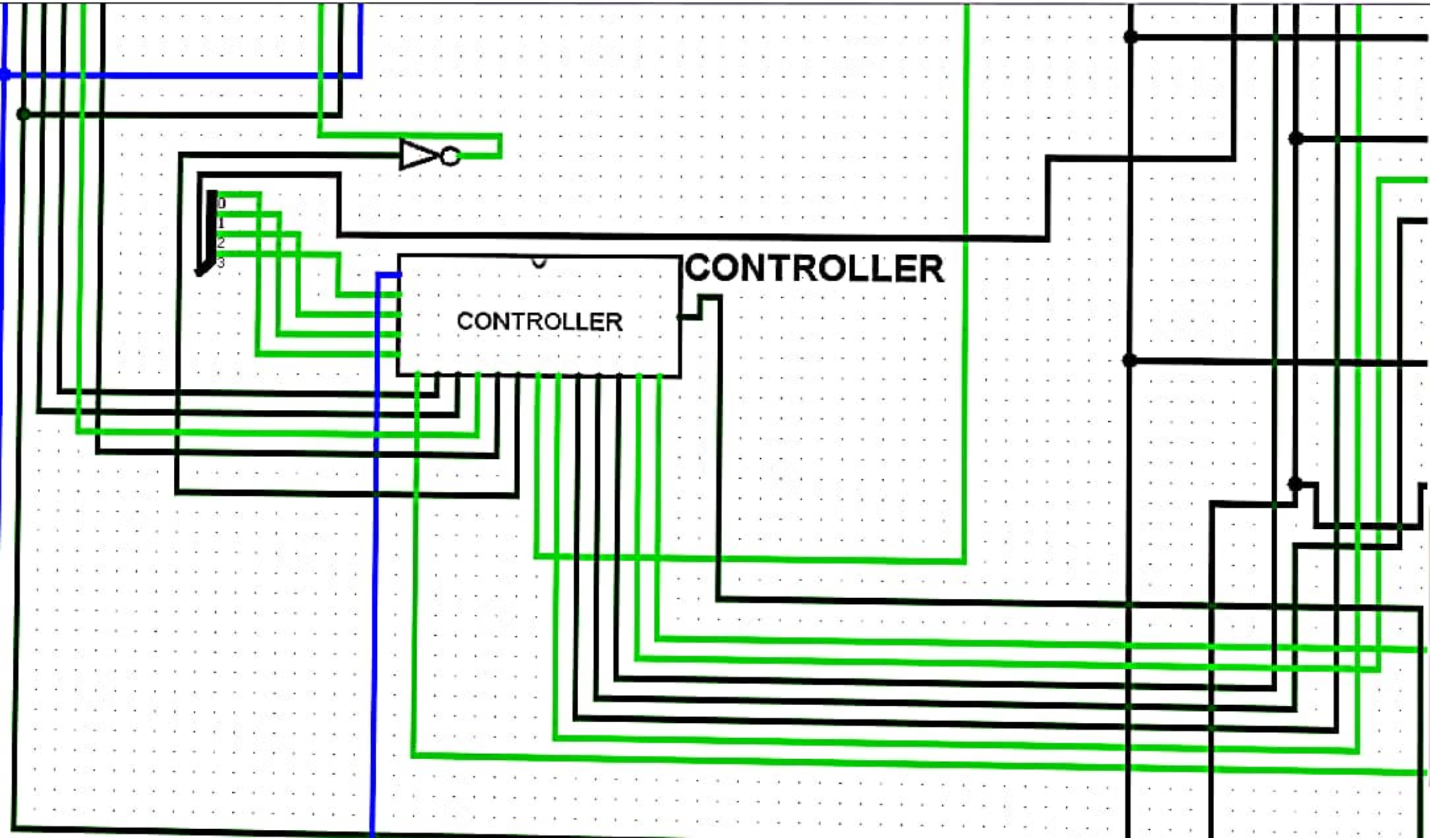
3) Instruction Register (IR) - It is a 8 bit tri state buffer register which decodes the first 4 bit of the data coming from RAM using controller. Here upper 4 nibbles are called opcode.

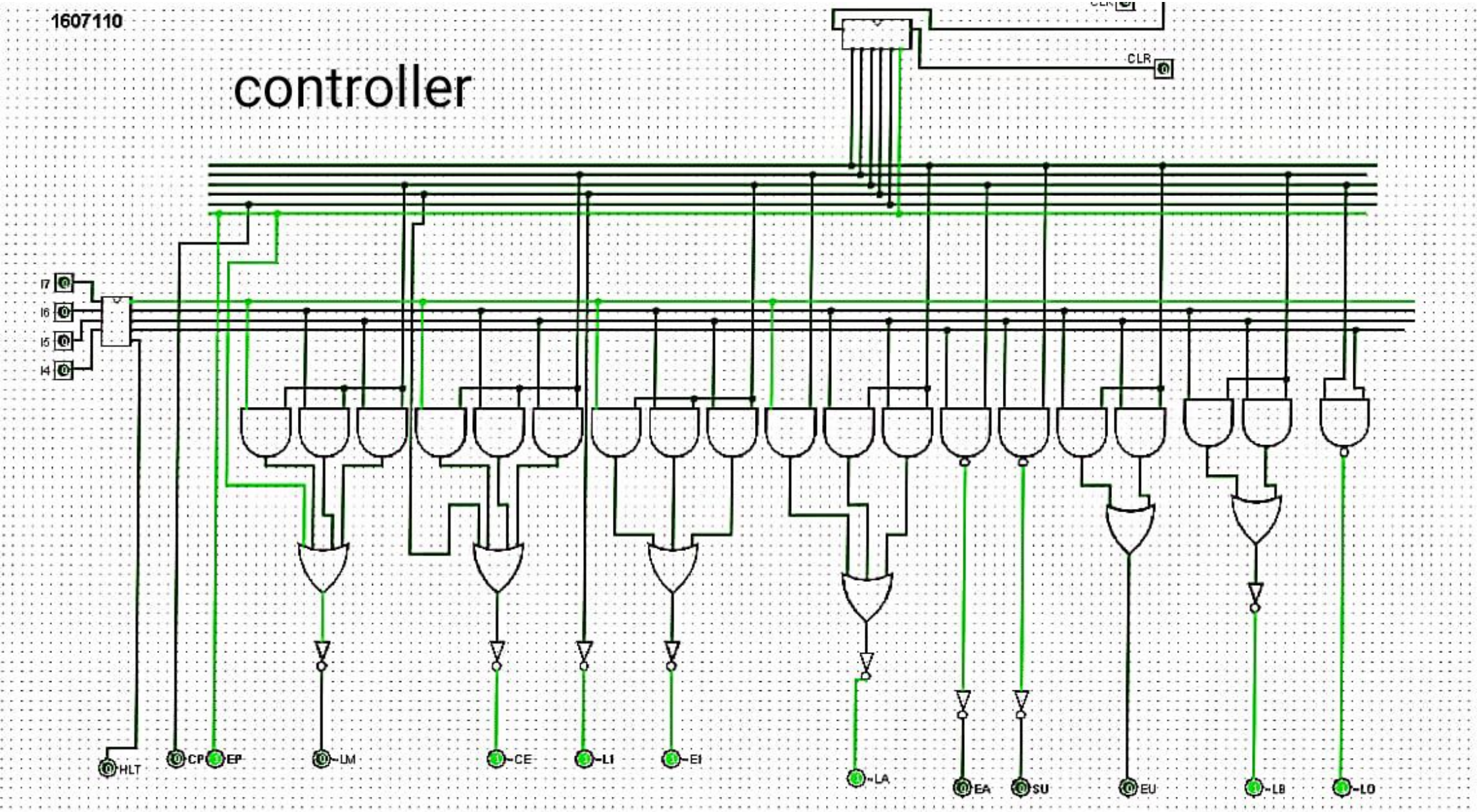
4) Controller - It is made of sub module ring counter and decoder. It generates 12 bit control word to generate signals according to the task denoted by opcode. Opcode is a input to controller. Based on operation controller generate different signals for specific tasks.

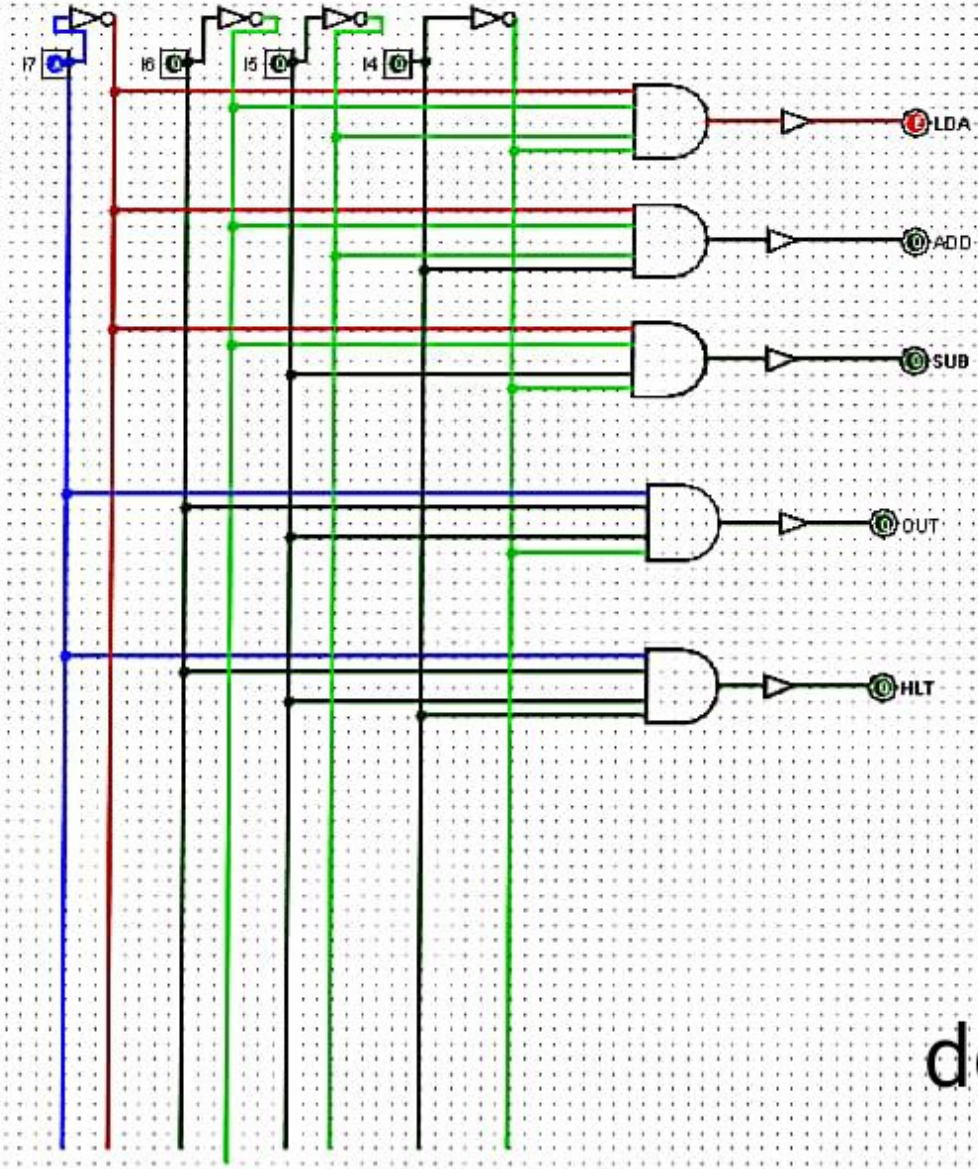
5) Decoder - It's a submodule of controller. It is used for identifying the ~~exact~~ it 'specific task assigned. For example here are 5 operations. 'Load, addition, subtraction, out and Halt.


1607110_1607106
main
decoder
CONTROLLER
memory_address_reg
RingCounter
TrueComplement
FA
ALU
adder
MAR_FINAL
REG_FINAL
pc
Wiring
Gates
Plexers
Arithmetic
Memory
Circuit: REG_FINAL
rcuit Name
REG_FINAL
nared Label
nared Label Fac...
East
nared Label Font
SansSerif Plain 12
INPUT
4_BIT
4_BIT
ENABLE
CLK
CLR
OUTPUT
8 bit register connection circuit


1607110_1607106
main
decoder
CONTROLLER
memory_address_reg
RingCounter
TrueComplement
FA
ALU
adder
MAR_FINAL
REG_FINAL
pc
Wiring
Gates
Plexers
Arithmetic
Memory
Circuit: main
Circuit Name
main
Shared Label
Shared Label Fac...
East
Shared Label Font
SansSerif Plain 12
A
D
sel
ld
clr
IR
0-3
4-7
CONTROLLER
CONTROLLER


0
1
2
3
CONTROLLER
CONTROLLER


1607110
controller
CLR
I7
I6
I5
I4
HLT
CP
EP
LM
CE
LI
EI
LA
EA
SU
EU
LB
LO



decoder

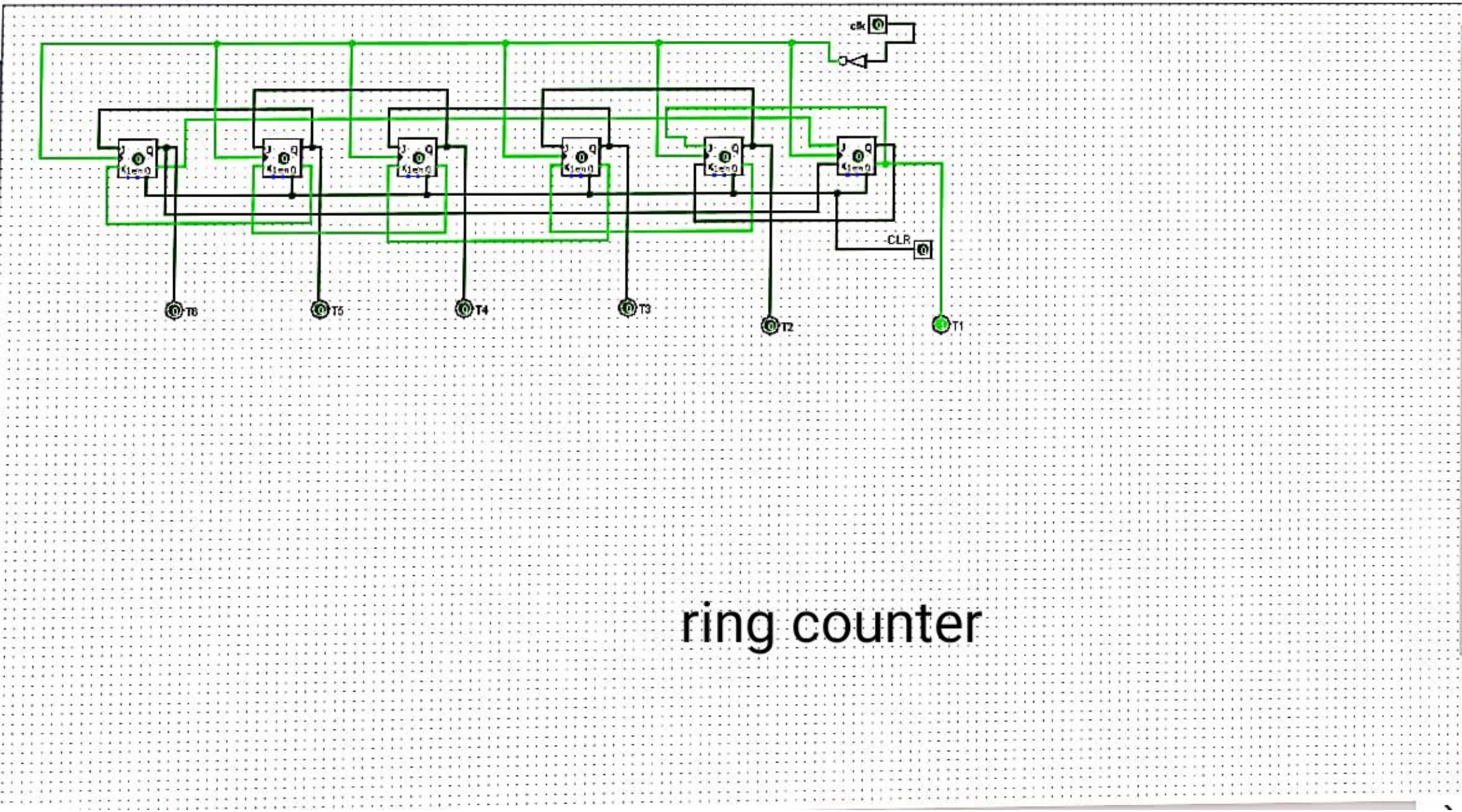
6) Ring Counter - It is designed using 6 JK flip flops and it generate counting value from T_1 to T_6. Where T_1 to T_3 is fetch cycle which is same for all instructions and T_4 to T_6 is execute cycle which is different depending instruction. It's clock is inverted from the main system clock.

7) Accumulator - It is also a 8 bit tri state buffer register. It stores the op on which operation is performed and also stores the result of the operation. ~LA signal controls the loading of data in accumulator

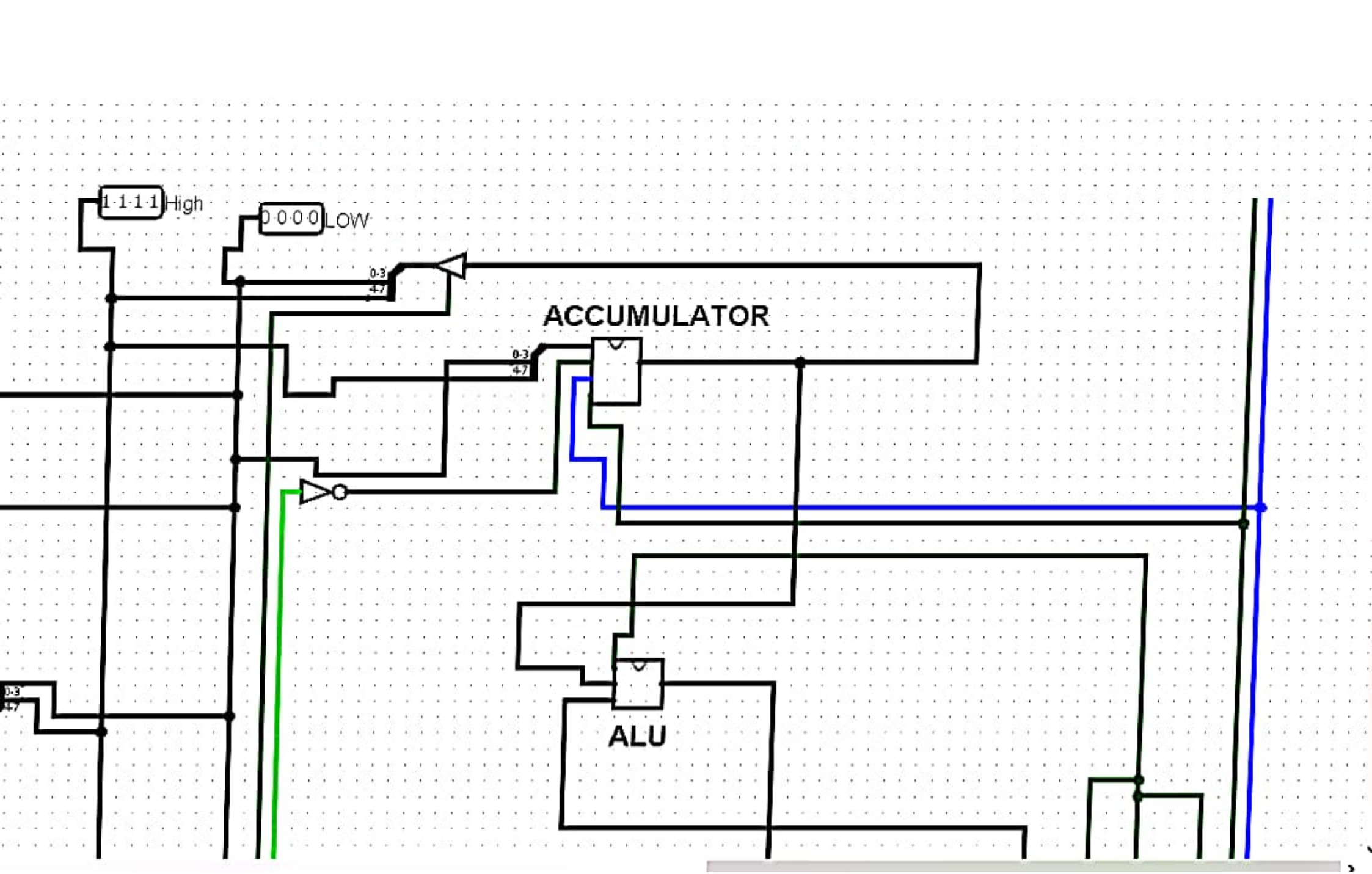
8) B register - It is a 8 bit tri state buffer register and is used to store the operand.

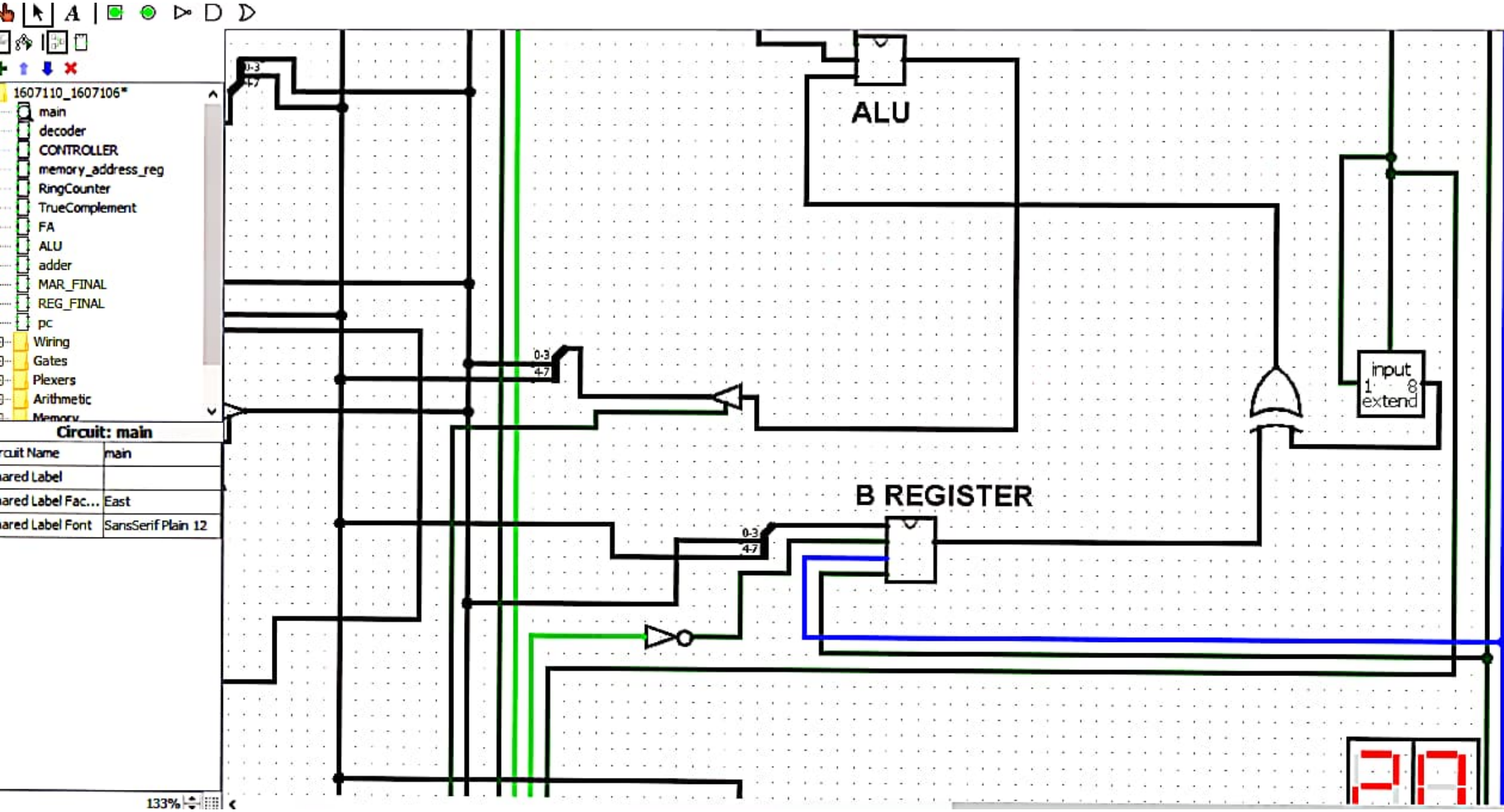
9) ALU - ALU works a 8 bit adder. It adds two 8 bit input with ~~and~~ carry and generate 8 bit output.

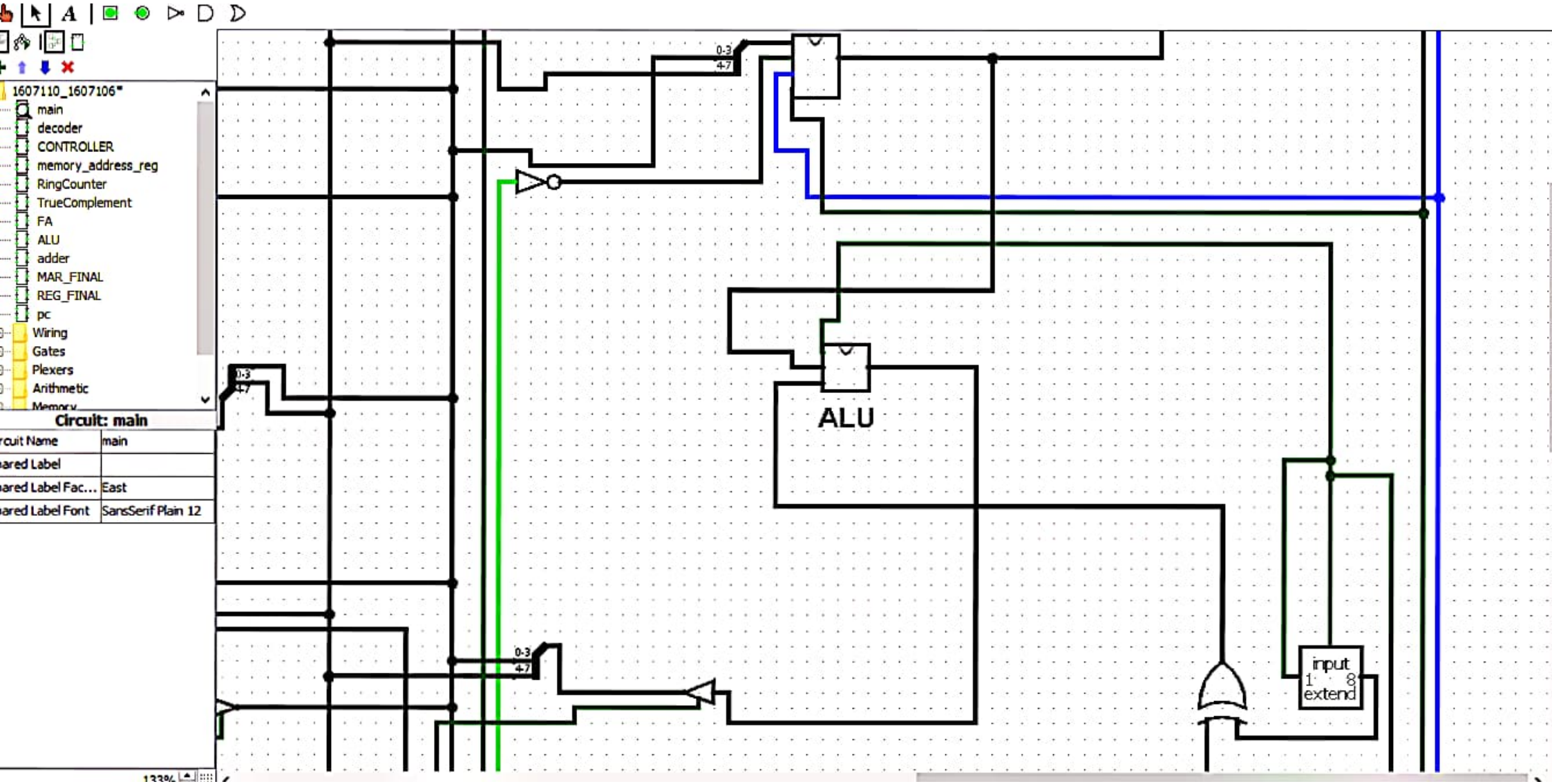
D

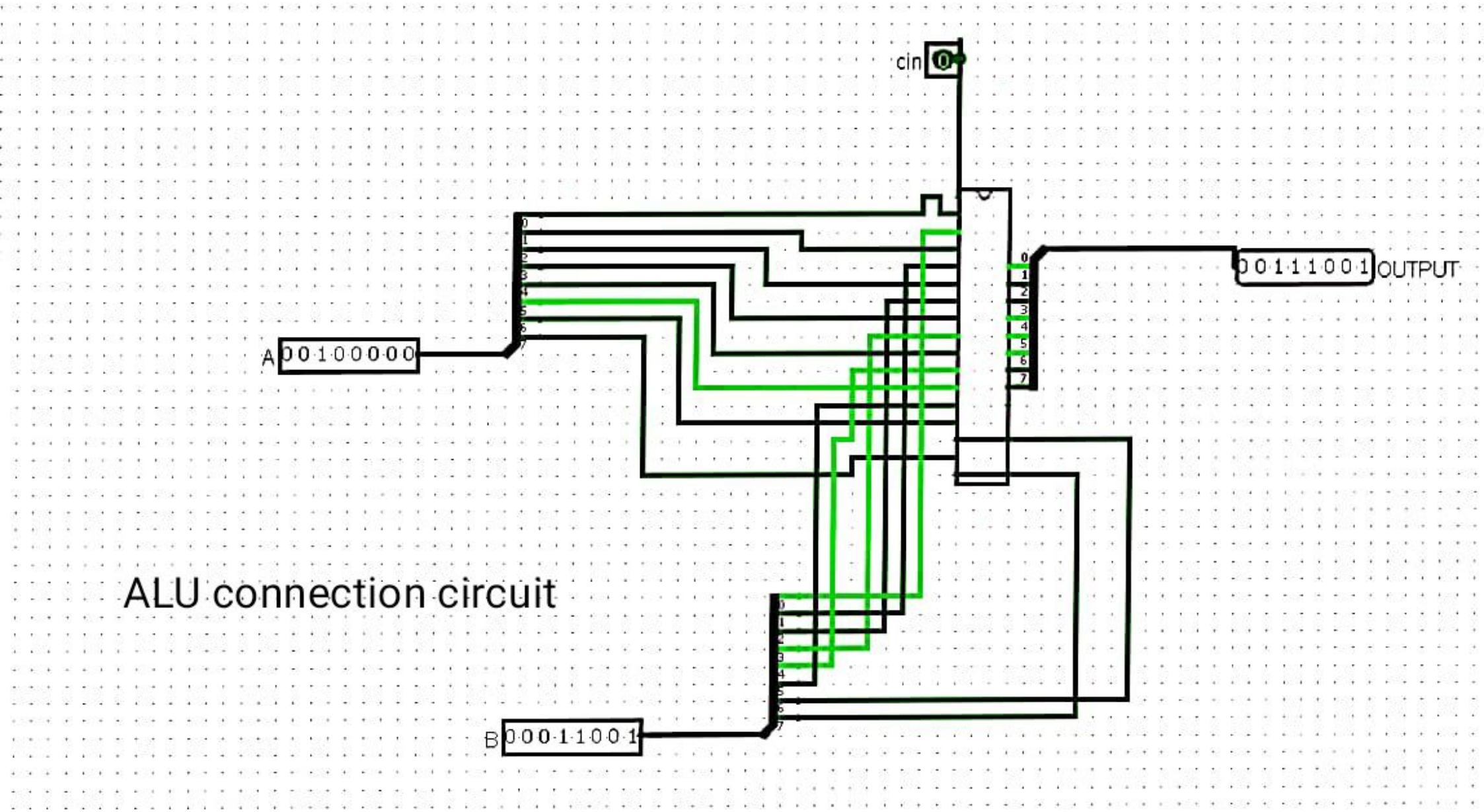


ring counter

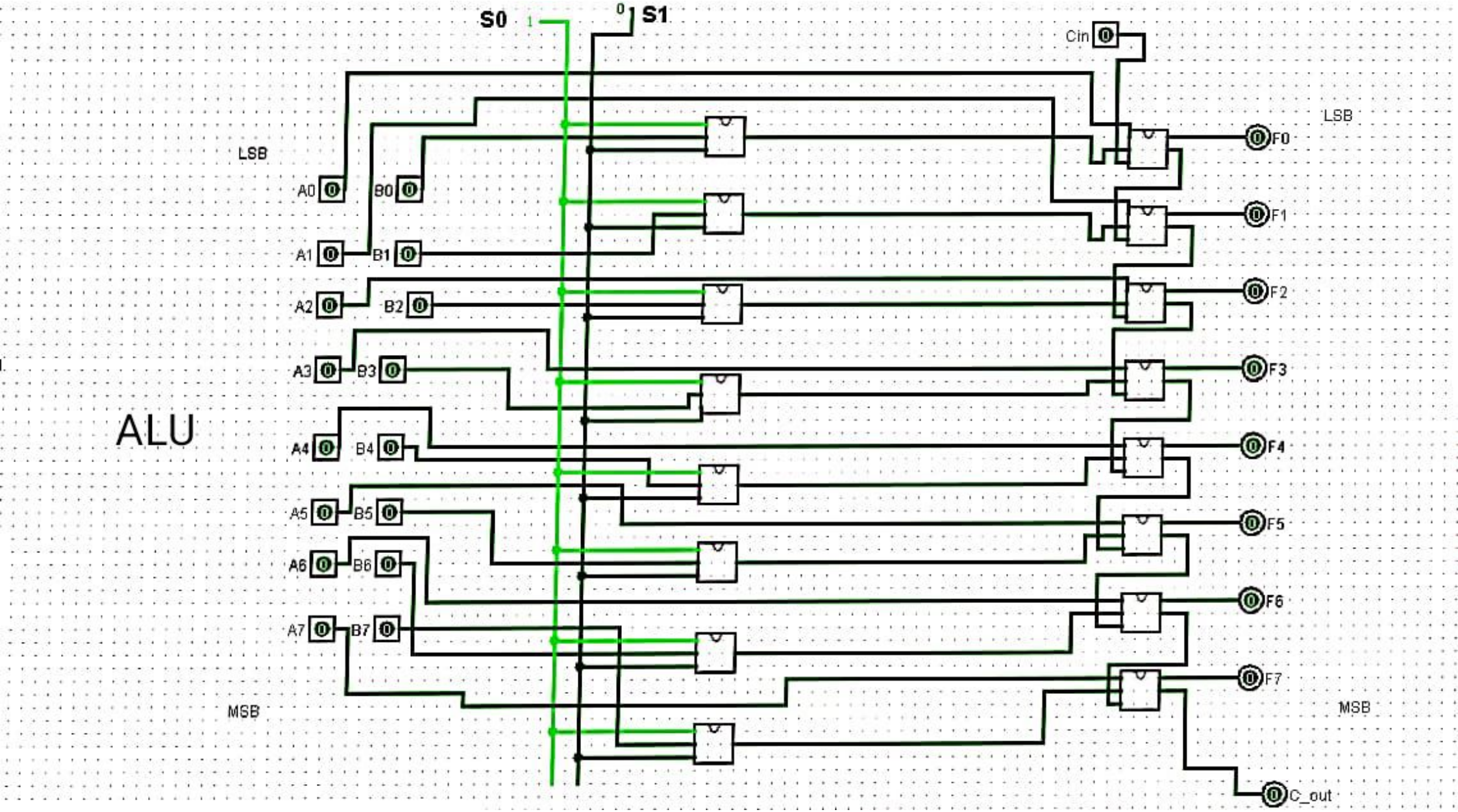

1 1 1 1 High
0 0 0 0 LOW
0-3
4-7
ACCUMULATOR
0-3
4-7
ALU

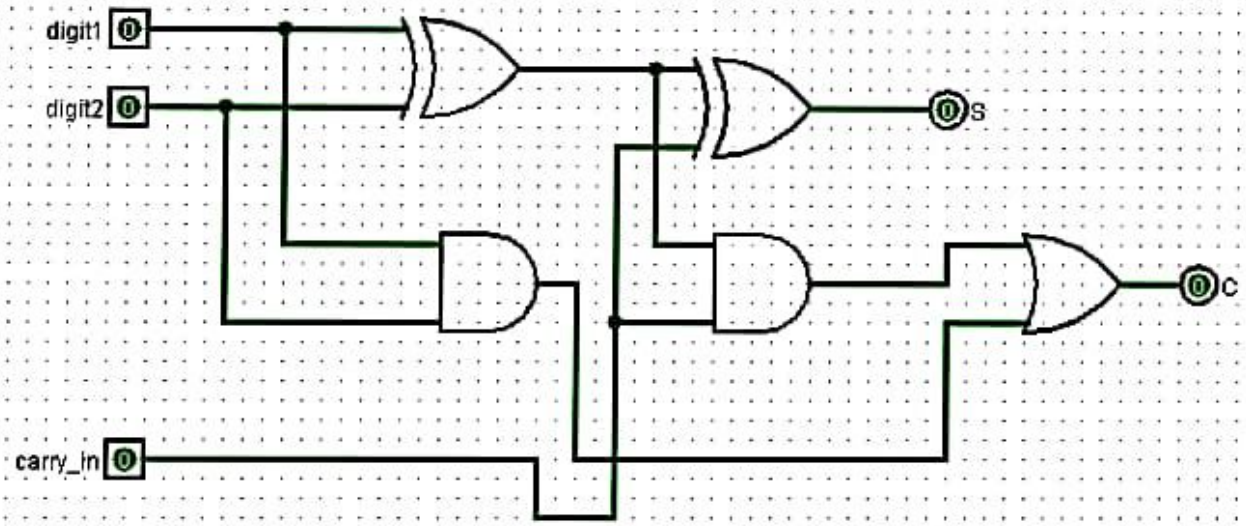

1607110_1607106*
main
decoder
CONTROLLER
memory_address_reg
RingCounter
TrueComplement
FA
ALU
adder
MAR_FINAL
REG_FINAL
pc
Wiring
Gates
Plexers
Arithmetic
Memory
Circuit: main
Circuit Name
main
Shared Label
Shared Label Fac...
East
Shared Label Font
SansSerif Plain 12
133%
ALU
B REGISTER
input
extend
0-3
4-7


1607110_1607106*
main
decoder
CONTROLLER
memory_address_reg
RingCounter
TrueComplement
FA
ALU
adder
MAR_FINAL
REG_FINAL
pc
Wiring
Gates
Plexers
Arithmetic
Circuit: main
rcuit Name
main
ared Label
ared Label Fac...
East
ared Label Font
SansSerif Plain 12
0-3
4-7
ALU
input
1 8
extend

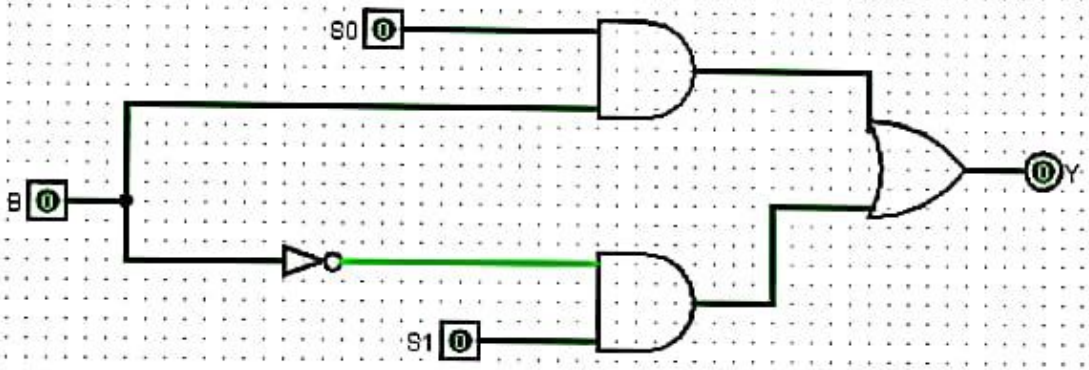


ALU connection circuit


ALU
S0
S1
Cin
LSB
MSB
A0
B0
A1
B1
A2
B2
A3
B3
A4
B4
A5
B5
A6
B6
A7
B7
F0
F1
F2
F3
F4
F5
F6
F7
C_out



full adder

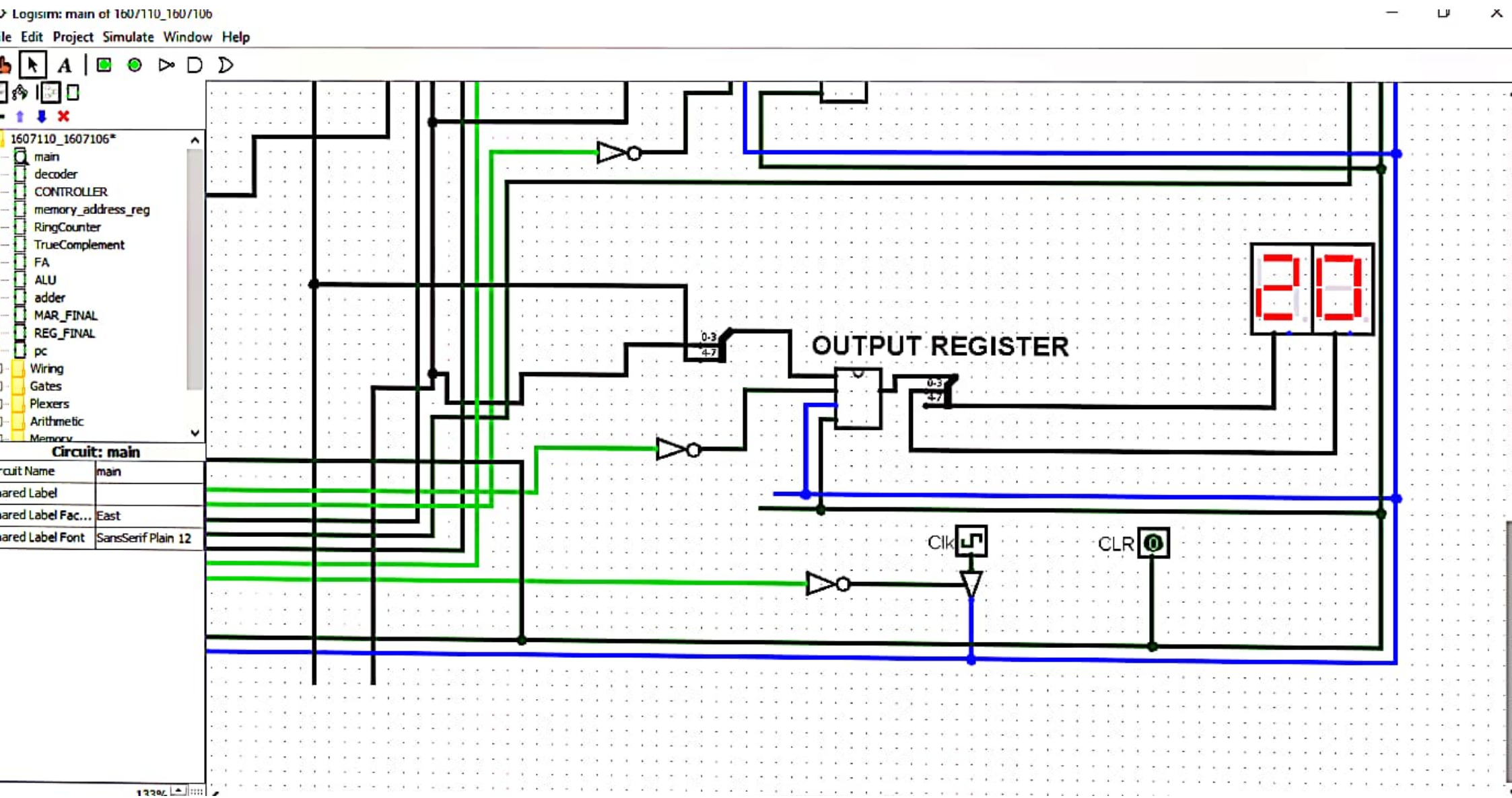


true complement circuit

here select bit S_0 and S_1 is 1 and 0 because when S_0 is 1 and S_1 is 0 this circuit works as a 8 bit full adder. When subtraction operation is needed to perform operand from B register is made 2's complement form and added with data in accumulator. XOR gate and bit extender is used for this operation.

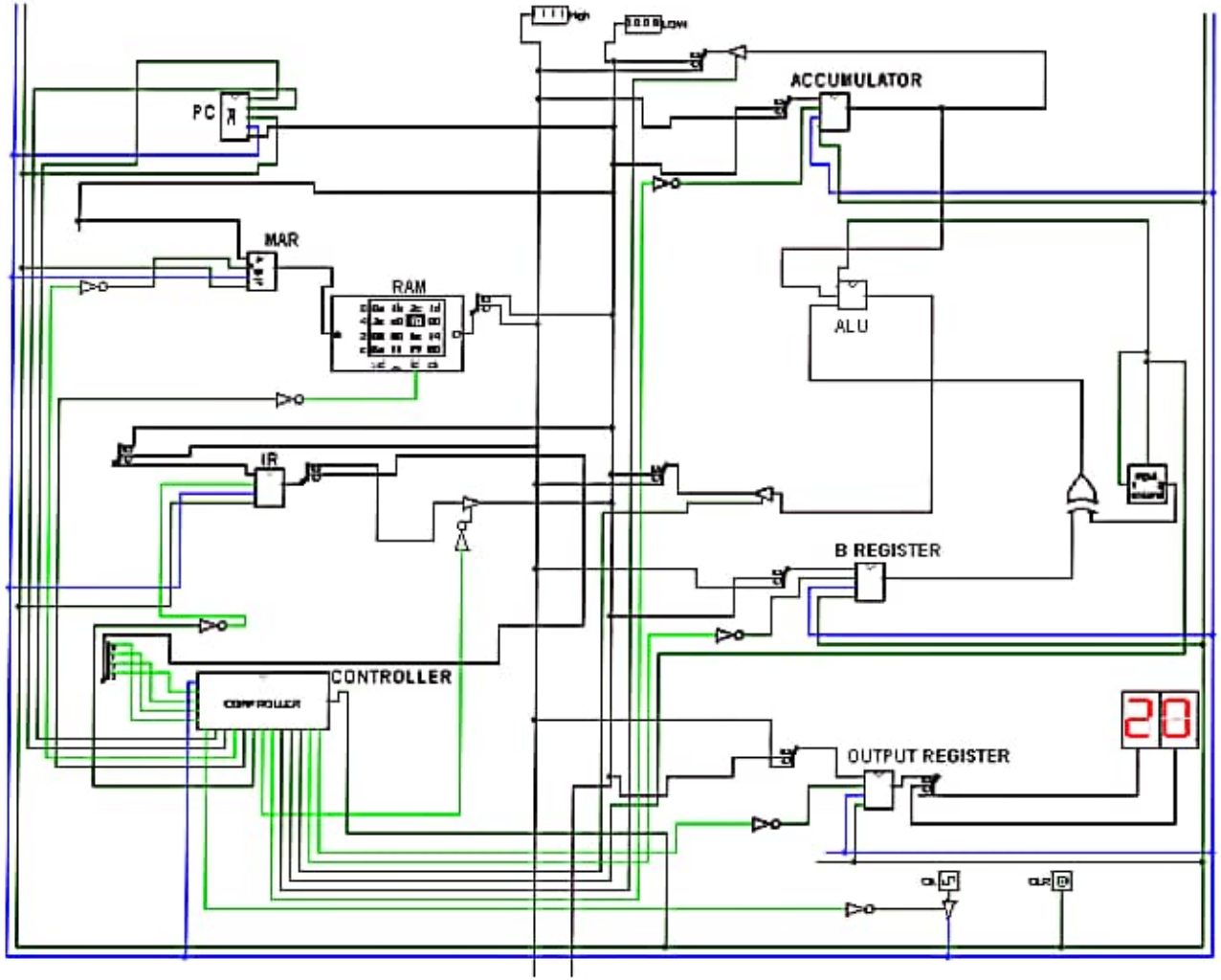
10) Output register - It is also a 8 bit tri state register which displays the 8 bit output and stores it. It is loaded when out instruction is executed.

11) Main Circuit -


Logisim: main of 1607110_1607106
le Edit Project Simulate Window Help
1607110_1607106*
main
decoder
CONTROLLER
memory_address_reg
RingCounter
TrueComplement
FA
ALU
adder
MAR_FINAL
REG_FINAL
pc
Wiring
Gates
Plexers
Arithmetic
Memory
Circuit: main
rcuit Name
main
ared Label
ared Label Fac...
East
ared Label Font
SansSerif Plain 12
OUTPUT REGISTER
0-3
4-7
Clk
CLR

1607106

1607110



Output - 1607110

30+20-10+17-25

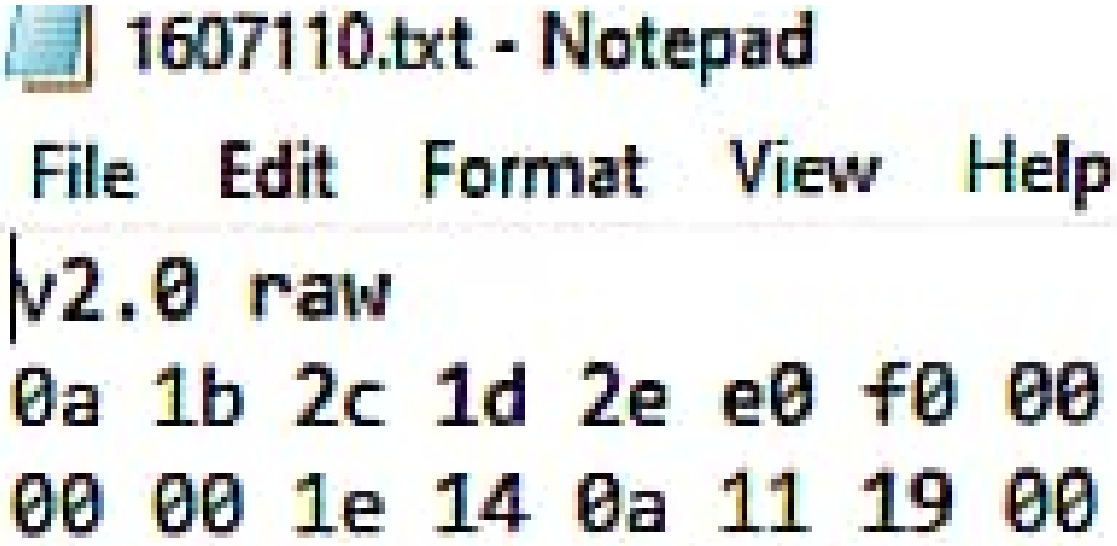
Address	Contents
0H	LDA AH
1H	ADD BH
2H	SUB CH
3H	ADD DH
4H	SUB EH
5H	OUT
6H	HLT
7H	XX H
8H	XX H
9H	XX H
AH	1E H
BH	14 H
CH	0A H
DH	11 H
EH	19 H
FH	XX H

Machine code

Address	Content
0000	0000 1010
0001	0001 1011
0010	0010 1100
0011	0001 1101
0100	0010 1110
0101	1110 xxxx
0110	1111 xxxx
0111	xxxx xxxx
1000	xxxx xxxx
1001	xxxx xxxx
1010	0001 1110
1011	0001 0100
1100	0000 1010
1101	0001 0001
1110	0001 1001
1111	xxxx xxxx

Illustrated notation

Address	content
0H	0A H
1H	1B H
2H	2C H
3H	1D H
4H	2E H
5H	EX H
6H	FX H
7H	XX H
8H	XX H
9H	XX H
AH	1E H
BH	14 H
CH	0A H
DH	11 H
EH	19 H
FH	XX H


1607110.txt - Notepad
File Edit Format View Help
v2.0 raw
0a 1b 2c 1d 2e e0 f0 00
00 00 1e 14 0a 11 19 00

SAP-1 Program for 16 + 20 + 24 - 32:

Assembly Language:

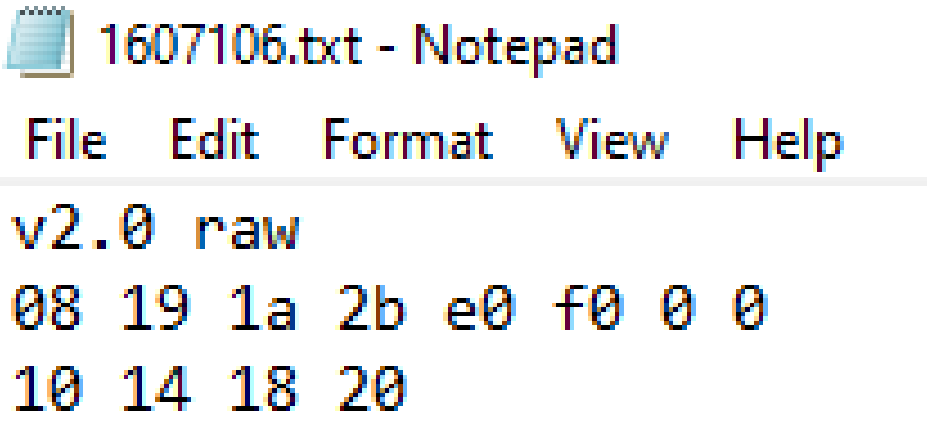
Address	Contents
0H	LDA 8H
1H	ADD 9H
2H	ADD AH
3H	SUB BH
4H	HLT
5H	OUT
6H	XX
7H	XX
8H	10H
9H	14H
AH	18H
BH	20H

Machine Language:

Address	Contents
0000	0000 1000
0001	0001 1001
0010	0001 1010
0011	0010 1011
0100	1110 XXXX
0101	1111 XXXX
0110	XXXX XXXX
0111	XXXX XXXX
1000	0001 0000
1001	0001 0100
1010	0001 1000
1011	0010 0000

Chunk Notation:

Address	Contents
0H	08 H
1H	19 H
2H	1A H
3H	2B H
4H	EXH
5H	FXH
6H	XXH
7H	XXH
8H	10 H
9H	14 H
AH	18 H
BH	20 H


1607106.txt - Notepad
File Edit Format View Help
v2.0 raw
08 19 1a 2b e0 f0 0 0
10 14 18 20

160711 0

Discussion - In this project we built a SAP 1 computer using bottom up method, where adder, register, ALU, Program counter were made of combinations of logic gates. By combining these registers and adder modules, program counter the whole SAP-1 computer was made which is able to execute a simple addition subtraction program and generate output Here all the register modules were made using tri state buffer register. logic gates and tri state switches were used to build these registers. Program counter which points the next instruction were made of flipflops. The ALU was also made of logic gates. These modules were combined to make the SAP-1 computer. Based on the instruction s generated by the instruction register control signals were generated by

the controller sequencer. For sequencing the operations a ring counter is used and designed to give 6 states T_1 to T_6 where T_1 to T_3 state is used for fetching instructions from memory and T_4 to T_6 is used for execution of each instruction. The signals are generated based on the instruction's opcode and are generated by the controller sequencer. Other modules such as A and B registers are used for storing the operands and A is used for storing the result. Output register is used for storing and displaying the result in hex display. ALU is used for performing addition and subtraction operation on given operands. The registers used in this circuit are built using 3 state buffer register, which output is in high impedence state if the

ENABLE Pin is low in the register but upper four nibble from instruction register (OPCode) so are used as inputs for controller sequencer to decide which operation to be performed, and this task is performed by decoder which also inverts these input signals (opcode). In logisim if a high impedence state is inverted it becomes erroneous. For this reason the ENABLE Pin of all registers used in this circuit are always ON so that the output of the registers are never in high impedence state.

Conclusion - Two different programs were tested using the SAP-1 circuit and it gave the accurate answer. The programs were designed using the instructions supported by the SAP-1 computer that is implemented means using LDA, ADD, SUB, OUT and HLT instruc-

-ions only. The internal transitions were also accurate. The program stoped successfully after ~~reaching~~ executing HLT instruction. The individual modules were also tested and gave correct output according to input instruction. So we can conclude that the whole design and implementation of SAP-1 computer in logisim was successful.