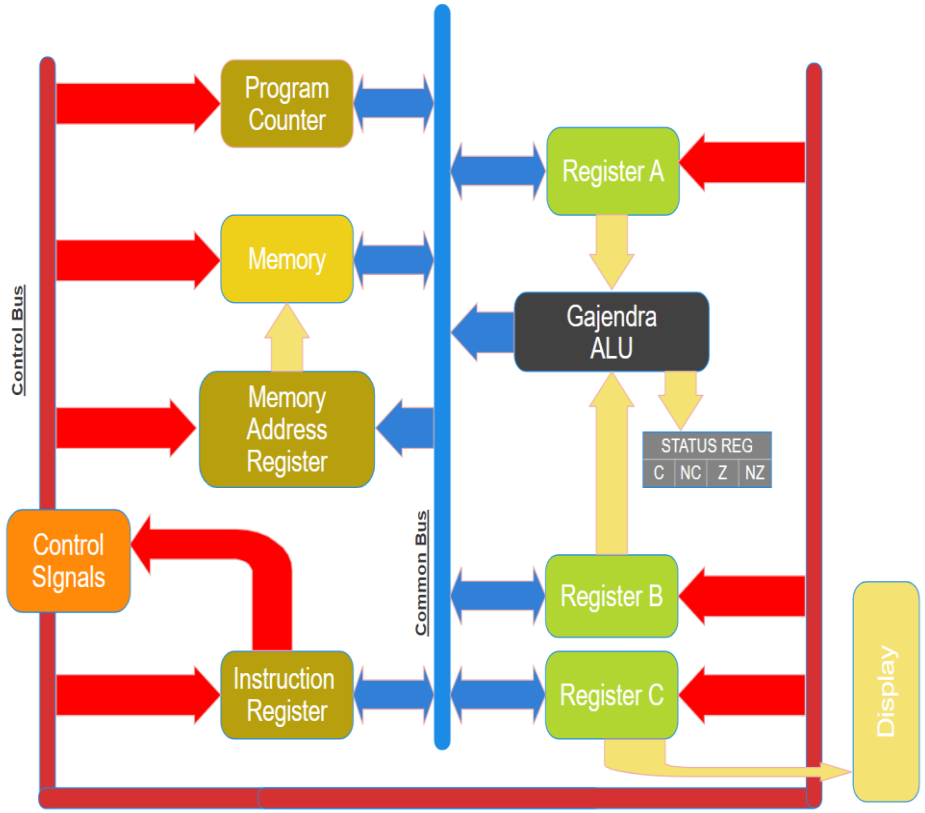
Computer System Design Gajendra-1

By Shreyanshu and Rohan

Introduction



Our designed CPU swiftly performs diverse operations like swapping, addition, subtraction based on user inputs, promptly displaying accurate results for seamless user interaction and computational efficiency.

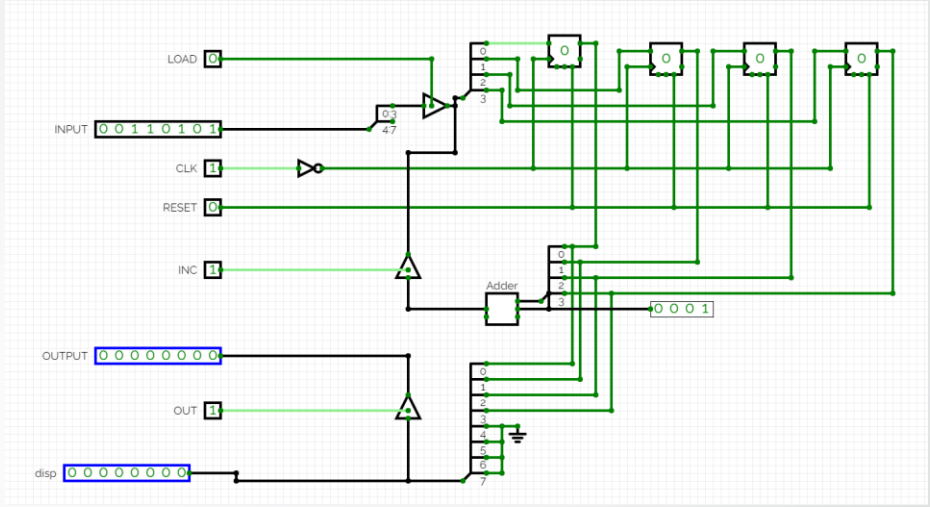
Utilizing components such as Registers, ALU, ROM, EEPROM, and Counters, our CPU automates operations. Controlled by a dedicated controller, precise coordination of control signals ensures efficient execution, underscoring the integration of key elements for optimized functionality.

COMPONENTS IN OUR CPU

- **Program Counter (counter_PC)** :

We have made this with flip-flops, an adder and tri-state buffers to control the flow of data. The input pins are LOAD, INPUT, CLK, RESET, INC, OUT and the output pins are OUTPUT , DISP.

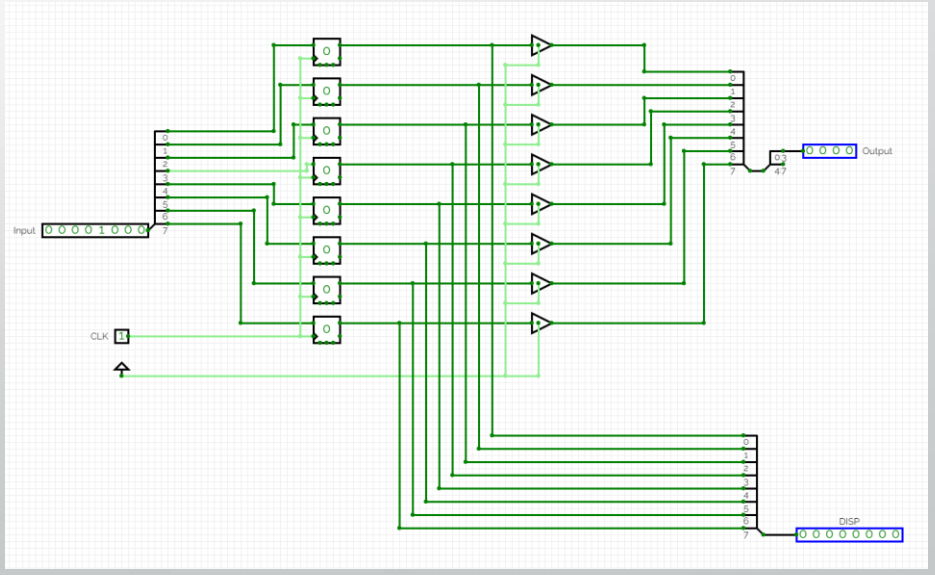
When LOAD is 1, it loads the INPUT value into the counter. When RESET is 1, it is set to oooo by default. When INC is 1, counter increments its value by 1. When OUT is 1, it sends the OUTPUT to the common bus.



- **Memory Address Register (reg_MAR)** :

We have made this with flip-flops and tri-state buffers to control the flow of data. The input pins are INPUT, CLK and the output pins are OUTPUT , DISP.

It takes input from the common bus and stores the 4 least significant bits of the input (LSB). It is directly connected to the memory, i.e. ROM.



- **Register A (Accumulator) (GEN_REG A)** :

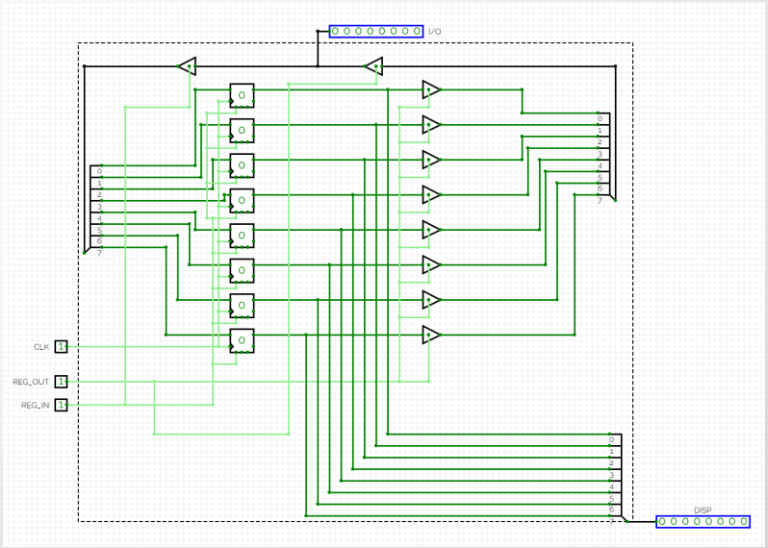
Register made from flip-flops and tri-state buffers. Input pins are CLK, REG_IN, REG_OUT and output pins are I/O and DISP.

When REG_IN is 1 and REG_OUT is 0, I/O stores input from common bus. When REG_IN is 0 and REG_OUT is 1, I/O pushes stored value to common bus. Both can't be 1 at the same time as it would give contention error. The value given by ALU is stored in the accumulator.

- **Register B (GEN_REG B)** :

Register made from flip-flops and tri-state buffers. Input pins are CLK, REG_IN, REG_OUT and output pins are I/O and DISP.

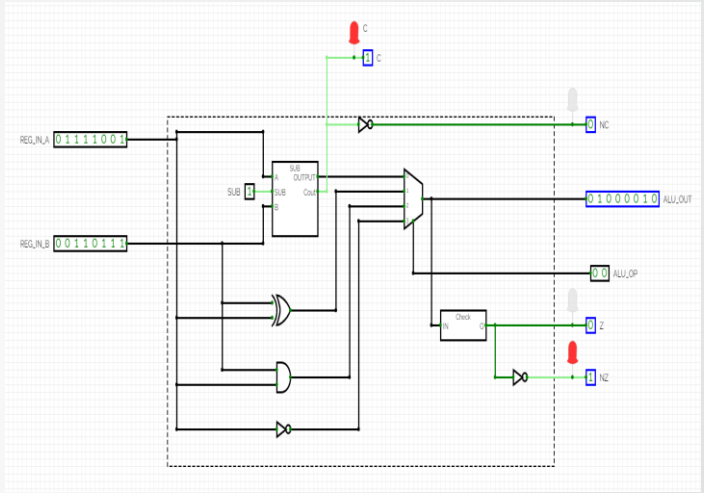
When REG_IN is 1 and REG_OUT is 0, I/O stores input from common bus. When REG_IN is 0 and REG_OUT is 1, I/O pushes stored value to common bus. Both can't be 1 at the same time as it would give contention error. The value temporarily stored to be inputted in ALU is stored in B.



GEN_REG

- **Arithmetic Logic Unit (ALU)** :

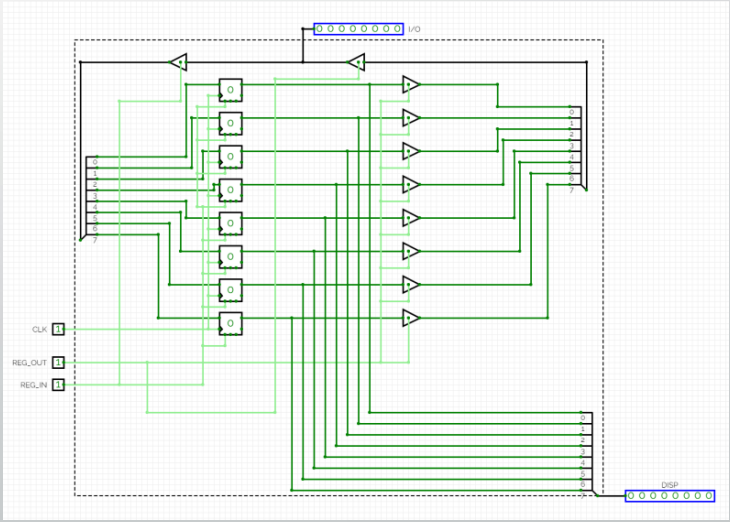
We have made our ALU with an 8-bit Adder, which can be used a subtractor too, along with some other gates and a multiplexer to decide which operation to perform. We have an in-built check module (status register) which measures zero/non-zero and carry/no carry. Input pins are REG_IN A, REG_IN B, ALU_OP, SUB and output pins are ALU_OUT, C, NC, Z, NZ.



ALU_OP is a 2-bit input which acts as select line for multiplexer and chooses which operation to do.

- **Register C(GEN_REG C)** :

Register made from flip-flops and tri-state buffers. Input pins are CLK, REG_IN, REG_OUT and output pins are I/O and DISP.

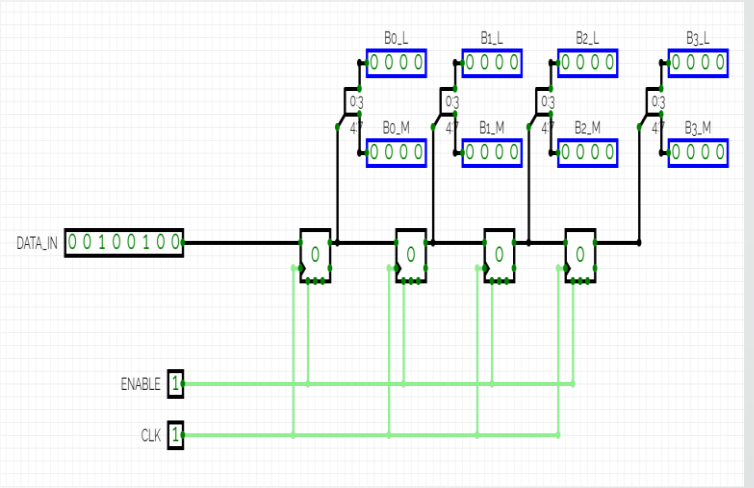


When REG_IN is 1 and REG_OUT is 0, I/O stores input from common bus. When REG_IN is 0 and REG_OUT is 1, I/O pushes stored value to common bus. Both can't be 1 at the same time as it would give contention error. The value stored in it is the final output and is displayed through the SCROLL DISPLAY.

- **Scroll Display (SCROLL_DISP)** :

It's made up from flip-flops and acts like a shift register. It pushes the outputs down as new inputs are provided. In the main circuit, we have connected it to 8 seven-segment displays to clearly show the shifting.

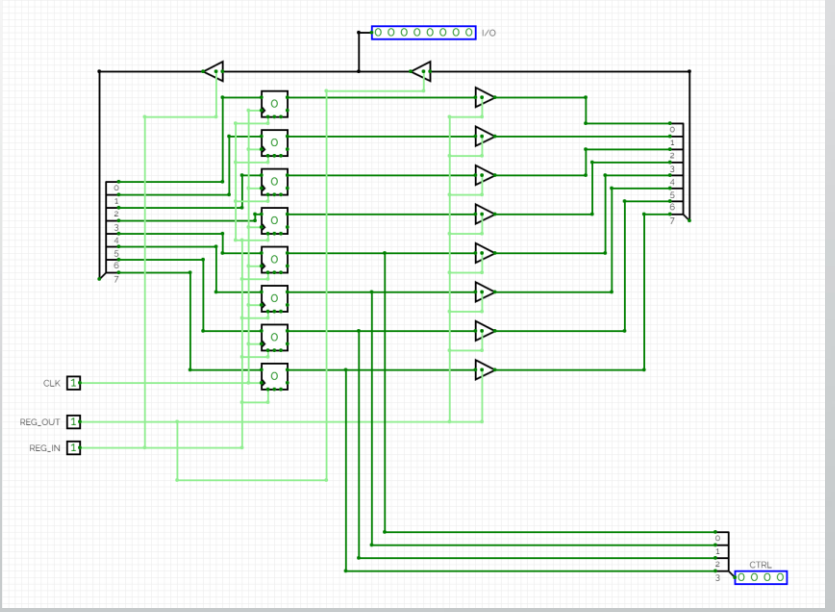
Input pins are DATA_IN, ENABLE, CLK and output pins are B0_L, B1_L, B2_L, B3_L, B0_M, B1_M, B2_M, B3_M.



- **Instruction Register (reg_IR)** :

It's made up from flip-flops and acts like a register. Input pins are CLK, REG_OUT, REG_IN and output pins are I/O, CTRL.

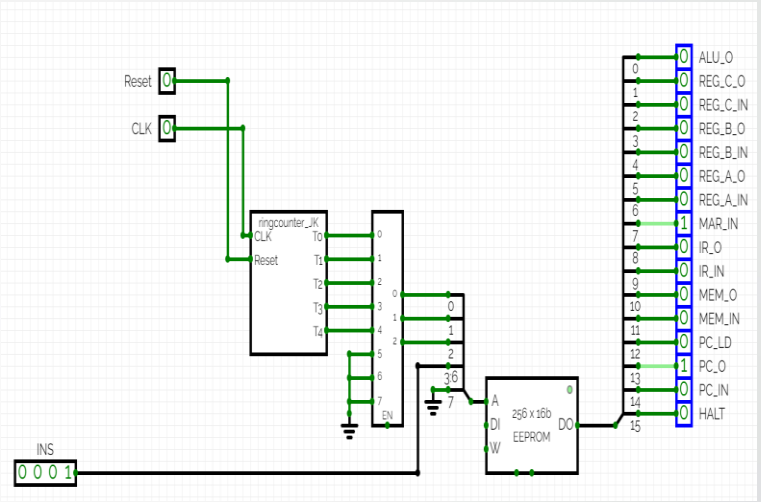
When REG_IN is 1 it takes input from the common bus (8-bit op-code) and stores it. When REG_OUT is 1 it pushes the output into the common bus. The CTRL pin gives the 4 most significant bits (MSB) of the op-code(instruction op-code) to the controller as soon as REG_IN is 1.



- **Controller** (cpu_timing_controller):

This the micro-instruction generator of the CPU. We have used Ring counter, Encoder and an EEPROM for this. Input pins are RESET, CLK, INS and output pins are ALU_O, REG_C_O…….PC_IN, HLT. These output pins generate a particular micro-instruction, i.e. a T-state. Combination of these T-states make up a full instruction. E.g. LDA, etc.

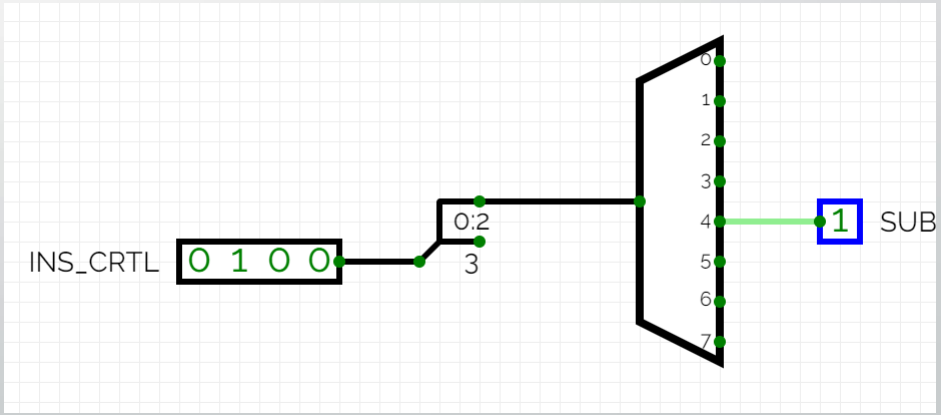
INS give the MSB (instruction op-code) and helps in selecting the appropriate row in EEPROM and then the T-states of that run accordingly to execute that particular instruction.



- **Instruction Decoder (Ins_Decoder)** :

This basically tells the ALU when to do subtraction and when to do addition. We have used a decoder for this. Input pin is INS_CTRL and output pin is SUB.

It takes the 4-bit MSB in and when it equals 0100, the decoder sets the SUB pin to 1 and gives this to the ALU's SUB pin.



INSTRUCTION SET

NOP – No Operation

- Description :

Performs no operation, all values retained in registers. Program counter increments by 1.

- Operation :

No operation

-

SYNTAX	OPERANDS	PROGRAM COUNTER
NOP	0<=X<=15	PC <- PC + 1

- 8-bit OP-code :

0000	XXXX

LDA – Load Accumulator

- Description :

Loads the value stored in the address LSB of the op-code into the accumulator.

- Operation :

A <- d

SYNTAX	OPERANDS	PROGRAM COUNTER
LDA	0<=d<=15	PC <- PC + 1

- 8-bit OP-code :

0001	dddd

ADD – Add without Carry

- Description :

Adds the value in the LSB address to the accumulator without the C flag and stores it in the accumulator.

- Operation :

A <- A + d

Adds 2 registers without the C flag and stores it in the accumulator.

SYNTAX	OPERANDS	PROGRAM COUNTER
ADD	0<=d<=15	PC <- PC + 1

- 8-bit OP-code :

0011	dddd

SUB – Subtract without carry

- Description :

Subtracts the value in the LSB address from the accumulator without the C flag and stores it in the accumulator.

- Operation :

A <- A – d

SYNTAX	OPERANDS	PROGRAM COUNTER
SUB	0<=d<=15	PC <- PC + 1

- 8-bit OP-code :

0100	dddd

LDI – Load Immediate

- Description :

Loads the immediate value in the LSB of OP-code into the accumulator.

No need to access data address.

- Operation :

A <- d

SYNTAX	OPERANDS	PROGRAM COUNTER
LDI	0<=d<=15	PC <- PC + 1

- 8-bit OP-code :

0101	dddd

JMP – Jump to address

- Description :

Loads Program counter with the address given in LSB such that ROM points to this address.

- Operation :

PC <- d

-

SYNTAX	OPERANDS	PROGRAM COUNTER
JMP	0<=d<=15	PC <- PC + 1

- 8-bit OP-code :

0110	dddd

SWAPAC – Swaps A with C

- Description :

Swaps the value of the accumulator and register C using B as temporary storer.

- Operation :

B <- A A <- C C <- B

-

SYNTAX	OPERANDS	PROGRAM COUNTER
SWAPAC	0<=X<=15	PC <- PC + 1

- 8-bit OP-code :

0111	XXXX

MOVAC – Moves A to C

- Description :

Moves the value stored in accumulator to register C.

- Operation :

C <- A

-

SYNTAX	OPERANDS	PROGRAM COUNTER
MOVAC	0<=X<=15	PC <- PC + 1

- 8-bit OP-code :

1001	XXXX

Similarly we can write for MOVAB, etc.

Instruction Set

Assembly	Machine Code	Assembly	Machine Code
NOP	0000 0x0	MOVAC	1001 0x9
LDA	0001 0x1	MOVBA	1010 0xA
ADD	0011 0x3	MOVCB	1011 0xB
SUB	0100 0x4	MOVAB	1100 0xC
LDI	0101 0x5	MOVCA	1101 0xD
JMP	0110 0x6	MOVBC	1110 0xE
SWAPAC	0111 0x7		

Example Assembly Programs

- Adding values from addresses 0x3 and 0x4 to A:

Control ROM data: 0x33 , 0x34 , 0x11 , 0x22

Commands : ADD 0x3 ADD 0x4

- Adding value in 0x6 , subtracting value in 0x7 , adding value in 0x8 , subtracting value in 0x9

Control ROM data: 0x36 , 0x47 , 0x38 , 0x49

Commands : ADD 0x6 SUB 0x7 ADD 0x8 SUB 0x9

- Adding numbers from address 0x8 to 0xD and displaying the result:

Control ROM: 0x38 , 0x39 , 0x3A , 0x3B , 0x3C , 0x3D

Commands : ADD 0x8 ADD 0x9 ADD 0xA ADD 0xB ADD 0xC ADD 0xD

Microinstructions and Controller Logic Design

```
NOP:    1 << PC_out | 1 << MAR_in
        1 << PC_inc | 1 << MEM_out | 1 << IR_in
        0
        0
        0
```

```
LDA:    1 << PC_out | 1 << MAR_in
        1 << PC_inc | 1 << MEM_out | 1 << IR_in
        1 << IR_out | 1 << MAR_in
        1 << MEM_out | 1 << RA_in
        0
```

```
ADD:    1 << PC_out | 1 << MAR_in
        1 << PC_inc | 1 << MEM_out | 1 << IR_in
        1 << IR_out | 1 << MAR_in
        1 << MEM_out | 1 << REGB_in
        1 << ALU_out | 1 << REGA_in
```

```
SUB:    1 << PC_out | 1 << MAR_in
        1 << PC_inc | 1 << MEM_out | 1 << IR_in
        1 << IR_out | 1 << MAR_in
        1 << MEM_out | 1 << REGB_in
        1 << ALU_out | 1 << REGA_in
```

```
LDI:    1 << PC_out | 1 << MAR_in
        1 << PC_inc | 1 << MEM_out | 1 << IR_in
        1 << IR_out | 1 << RA_in
        0
        0
```

```
JMP:    1 << PC_out | 1 << MAR_in
        1 << PC_inc | 1 << MEM_out | 1 << IR_in
        1 << PC_load | 1 << IR_out
        0
        0
```

```
SWAPAC: 1 << PC_out | 1 << MAR_in
        1 << PC_inc | 1 << MEM_out | 1 << IR_in
        1 << RA_out | 1 << RB_in
        1 << RA_in | 1 << RC_out
        1 << RB_out | 1 << RC_in
```

```
MOVAC:  1 << PC_out | 1 << MAR_in
        1 << PC_inc | 1 << MEM_out | 1 << IR_in
        1 << RA_out | 1 << RC_in
        0
        0
```

```
MOVBA:  1 << PC_out | 1 << MAR_in
        1 << PC_inc | 1 << MEM_out | 1 << IR_in
        1 << RB_out | 1 << RA_in
        0
        0
```

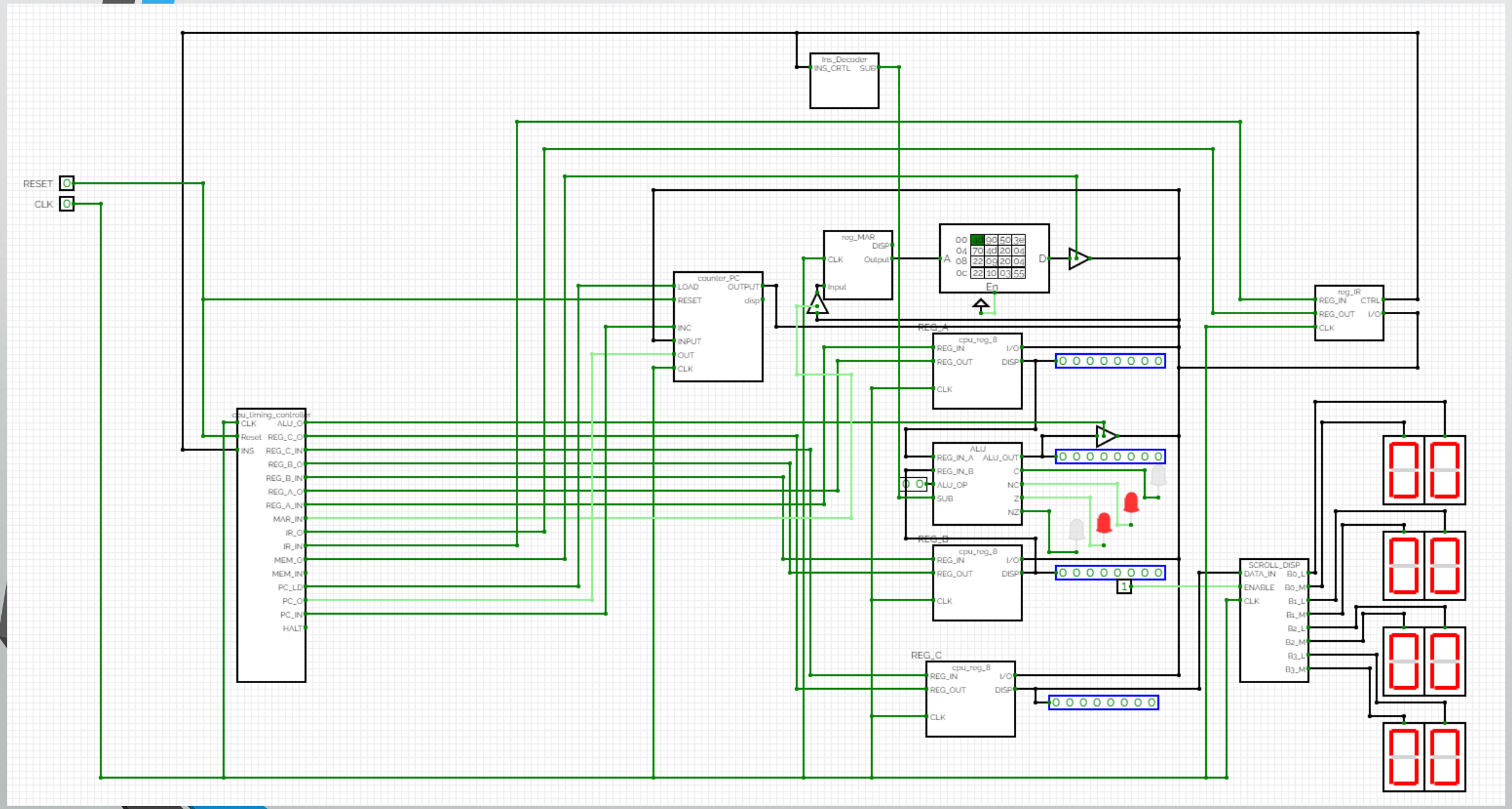
```
MOVCB:  1 << PC_out | 1 << MAR_in
        1 << PC_inc | 1 << MEM_out | 1 << IR_in
        1 << RC_out | 1 << RB_in
        0
        0
```

```
MOVAB:  1 << PC_out | 1 << MAR_in
        1 << PC_inc | 1 << MEM_out | 1 << IR_in
        1 << RA_out | 1 << RB_in
        0
        0
```

```
MOVCA:   1 << PC_out | 1 << MAR_in
         1 << PC_inc | 1 << MEM_out | 1 << IR_in
         1 << RC_out | 1 << RA_in
         0
         0
```

```
MOVBC:   1 << PC_out | 1 << MAR_in
         1 << PC_inc | 1 << MEM_out | 1 << IR_in
         1 << RB_out | 1 << RC_in
         0
         0
```

CPU_CORE_ARCH_GAJENDRA FINAL CIRCUIT


RESET
CLK
Ins_Decoder
INS_CRTL
SUB
reg_MAR
CLK
Output
Input
DISP
counter_PC
LOAD
RESET
INC
INPUT
OUT
CLK
OUTPUT
disp
A
D
En
reg_IR
REG_IN
REG_OUT
CLK
CTRL
I/O
REG_A
cpu_reg_8
REG_IN
REG_OUT
CLK
I/O
DISP
ALU
REG_IN_A
REG_IN_B
ALU_OP
SUB
ALU_OUT
C
NC
Z
NZ
REG_B
REG_C
cpu_timing_controller
CLK
Reset
INS
ALU_O
REG_C_O
REG_C_IN
REG_B_O
REG_B_IN
REG_A_O
REG_A_IN
MAR_IN
IR_O
IR_IN
MEM_O
MEM_IN
PC_LD
PC_O
PC_IN
HALT
SCROLL_DISP
DATA_IN
ENABLE
CLK
B0_L
B0_M
B1_L
B1_M
B2_L
B2_M
B3_L
B3_M