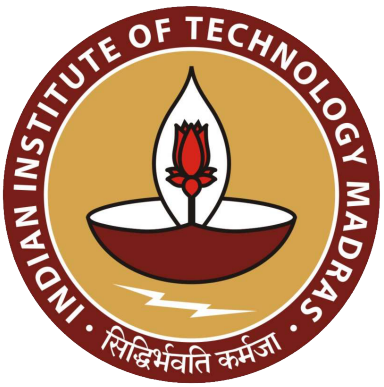
Lab Report Group 7

Jul - Nov 2023

Gajendra Processor

Submitted by
R.Yeshaswini (CS22B001)
Gurrala Abhishek (CS22B049)

Under the supervision of
Dr. Ayon Chakraborty
Assistant Professor

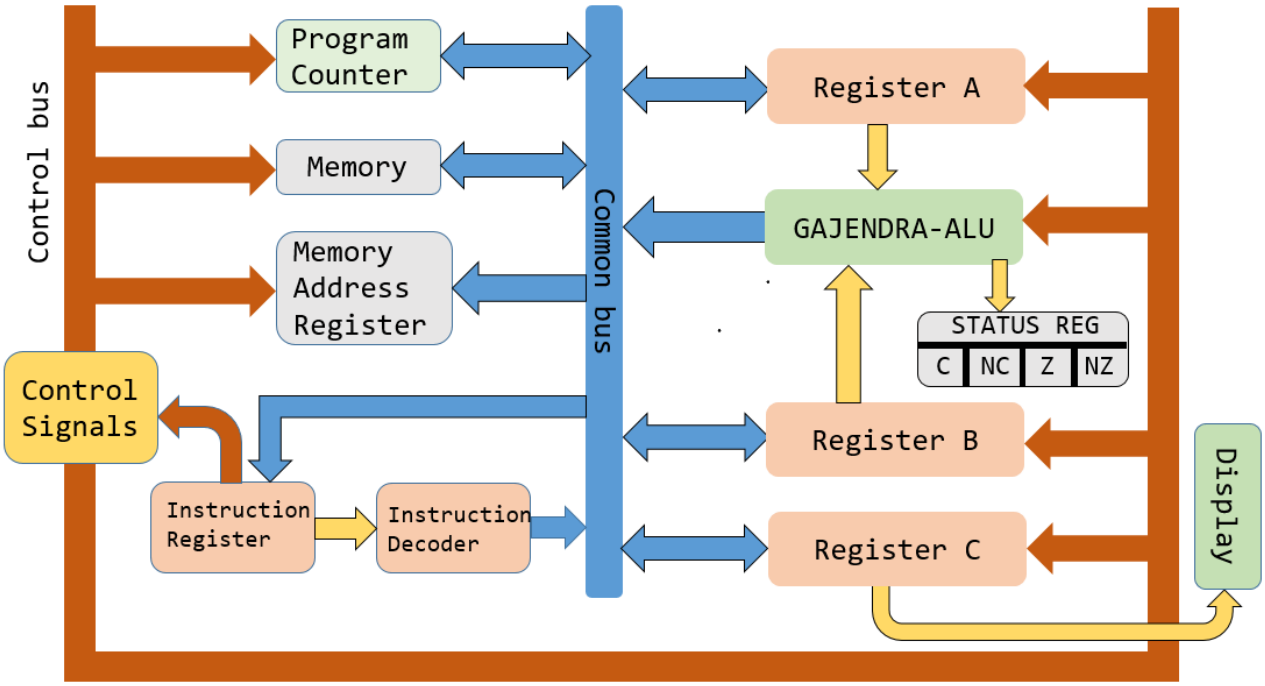


Computer Science and Engineering
Indian Institute of Technology Madras

Contents

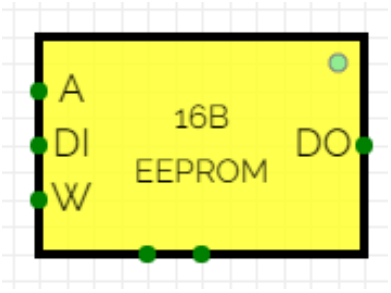
1 ARCH_GAJENDRA - Internal Components

1.1 Design Model



1.2 Memory

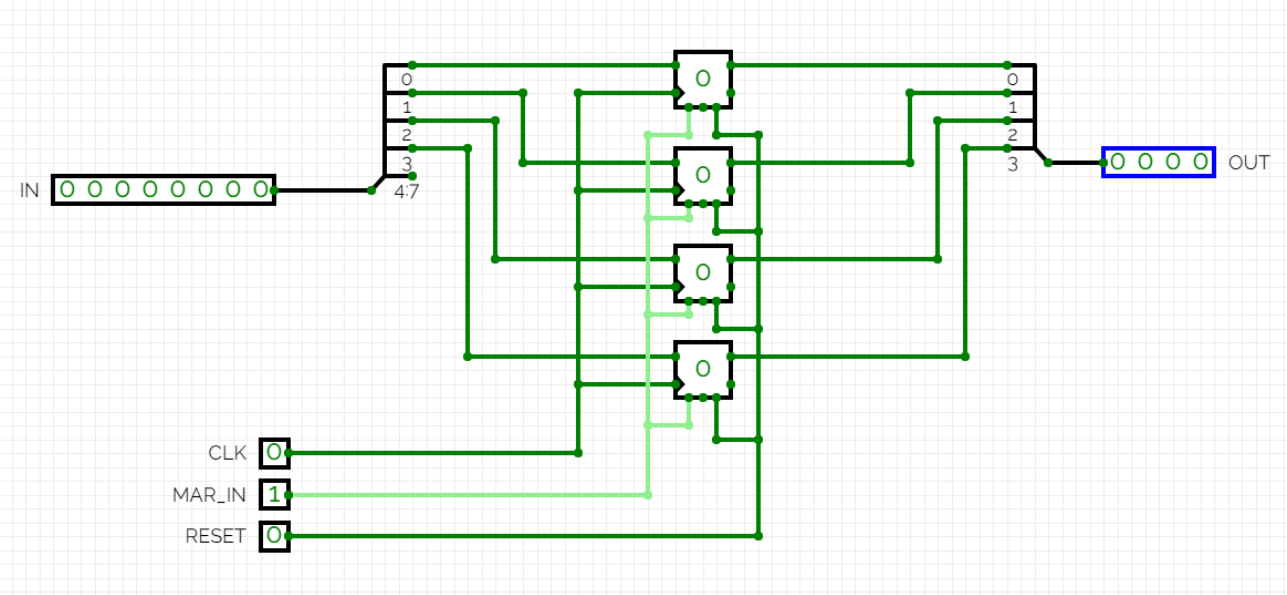
- **Design**:
  Memory is a EEPROM which contains instructions and data required for a program.
- **Circuit Diagram** :



1.3 Memory Address Register

- **Design** :
  During execution the 4-bit address from 8-bit input(4 LSB) in the Program Counter is transferred into the MAR, Then the Memory Address Register applies this 4-bit address to the Memory where a read operation is executed.

- **Circuit Diagram** :



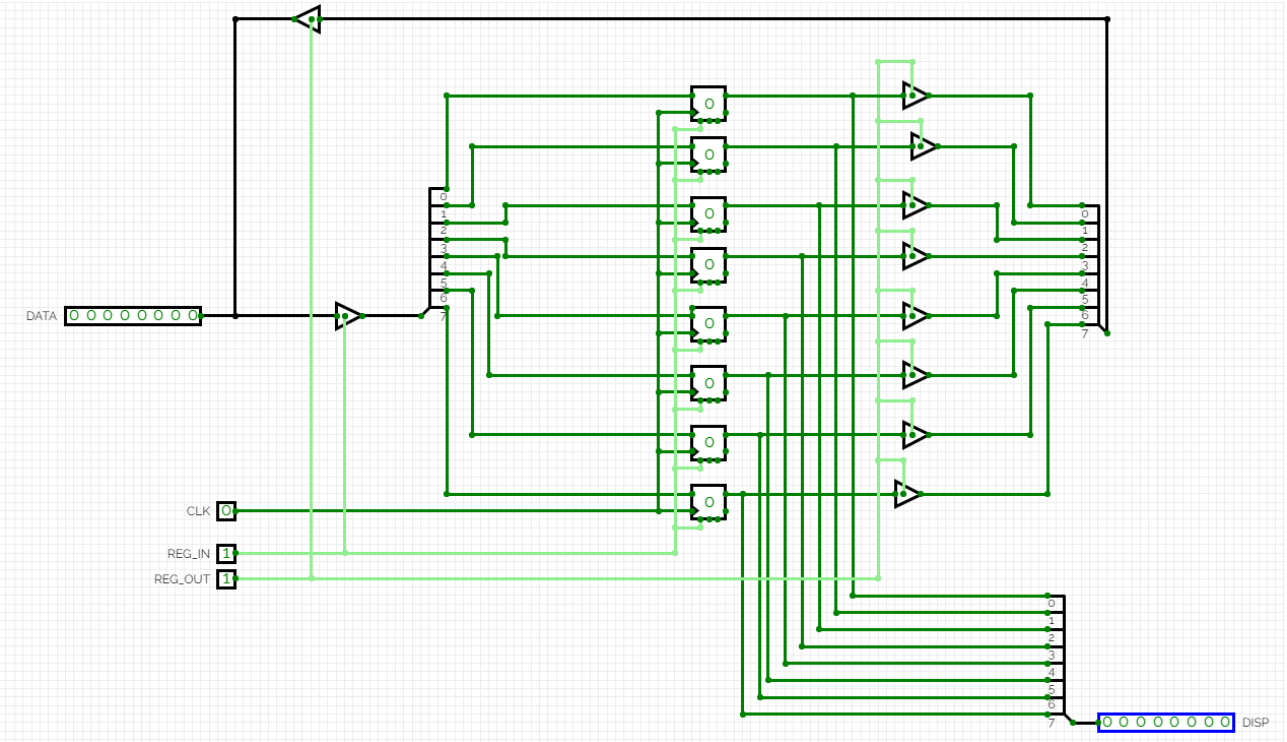
1.4 General CPU Register

- **Design** :
  It takes an 8-bit input from the common bus, storing the provided value. The 8-bit data stored within it is then presented as the output.The control inputs are REG_IN and REG_OUT.The Working of control inputs are

  1. When REG_IN is active (usually set to 1), the data on the DATA bus is taken into the register.
  2. When REG_OUT is active (usually set to 1), the data stored in the register is shown in DATA bus.
  3. The DISP output is connected to 8 bit output, and it shows the current value stored in the register. This output is continuously updated with the data in the register.

- We used three general purpose registers namely A,B and C. Third register REG_C is used in the case of swapping. Also the scrolling display is connected to REG_C. And we can see the value in the corresponding registers and bus in the flag (names are given accordingly).

- **Circuit Diagram** :

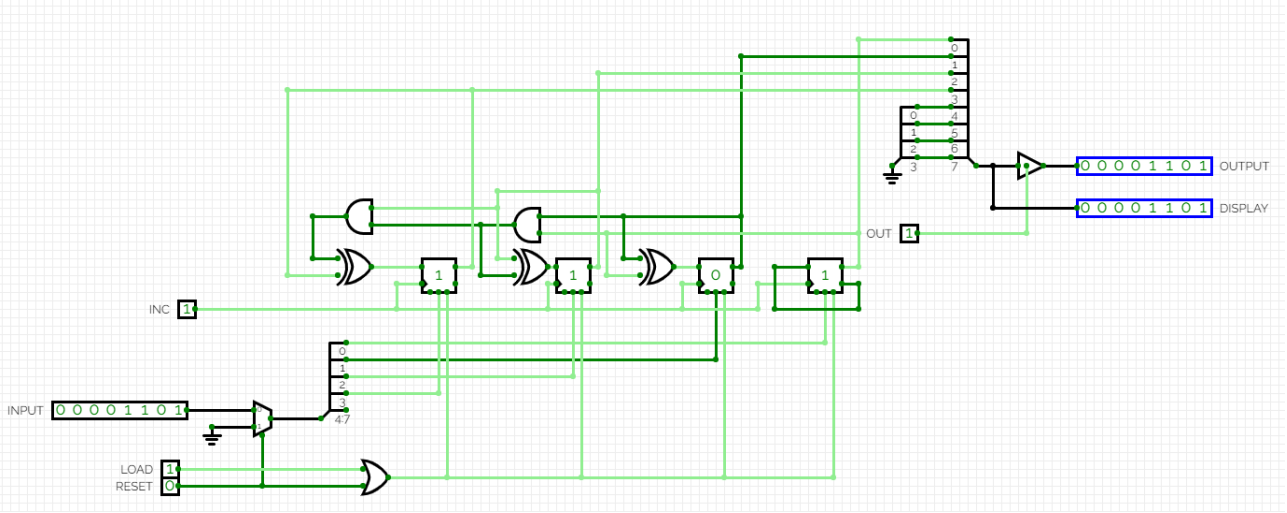


1.5 Program Counter

- **Design** :

  1. The Program Counter points to the address of the instruction in ROM, that has to be processed next.
  2. When Load of PC is set to 1, the 8-bit value from the common bus is taken as an input to the Program Counter. The 4 least significant bits (LSB) of the input are set as the output, with the 4 most significant bits (MSB) set to zero.
  3. When Reset is set to 1,all bits of input changes to 0.

- **Circuit Diagram** :



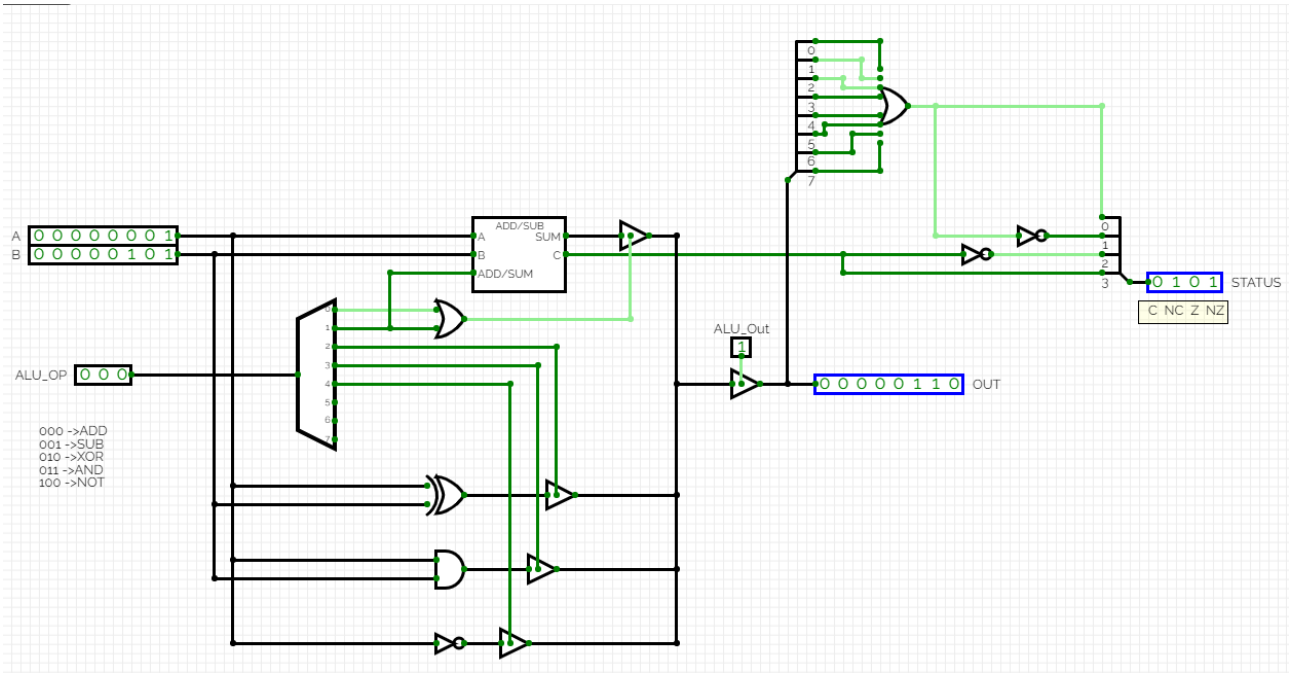
1.6 Arithmetic and Logical Unit

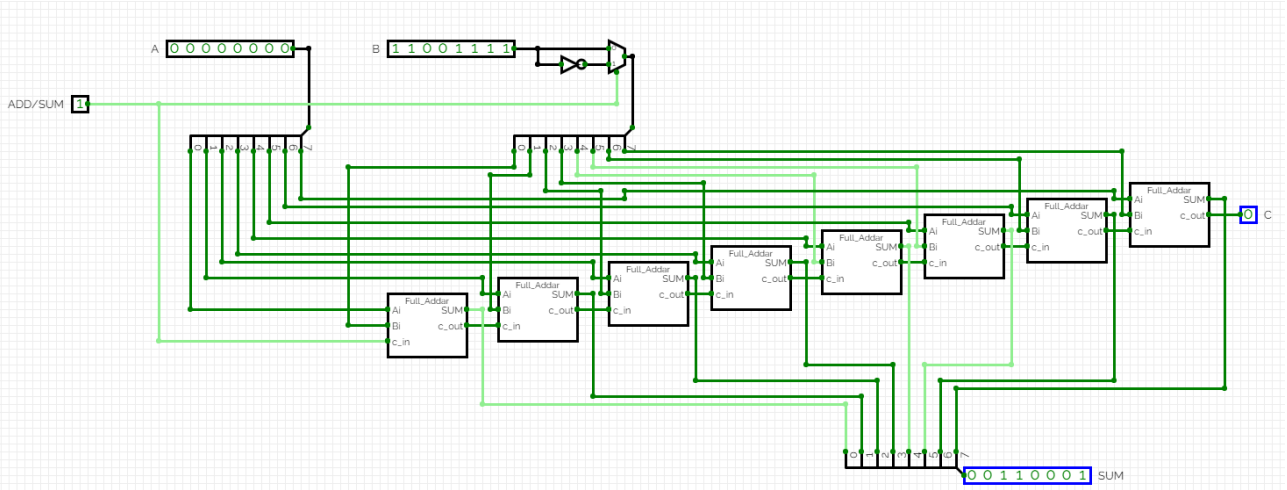
- **Design** :

  1. A simple ALU supporting 8-bit ADD,SUB,AND,XOR and NOT operations.
  2. The following flags are made from output and carry to give input of 4-bit reg_status_4.
     - Carry (C) – Set to 1 if ADD produces an output carry, else 0.
     - Not carry (NC) – Complement of (C)
     - Zero (Z) – Set to 1, if any of the operations create a result with value zero, else 0.
     - Not Zero (NZ) – Complement of (Z).
  3. This ALU can perform the following operations,but we are using only ADD and SUB (INS_DCDR gives only 2 inputs to ALU) in our Gajendra circuit.

ALU_OP	**Operation**
000	ADD
001	SUB
010	XOR
011	AND
100	NOT

- **Circuit Diagram** :



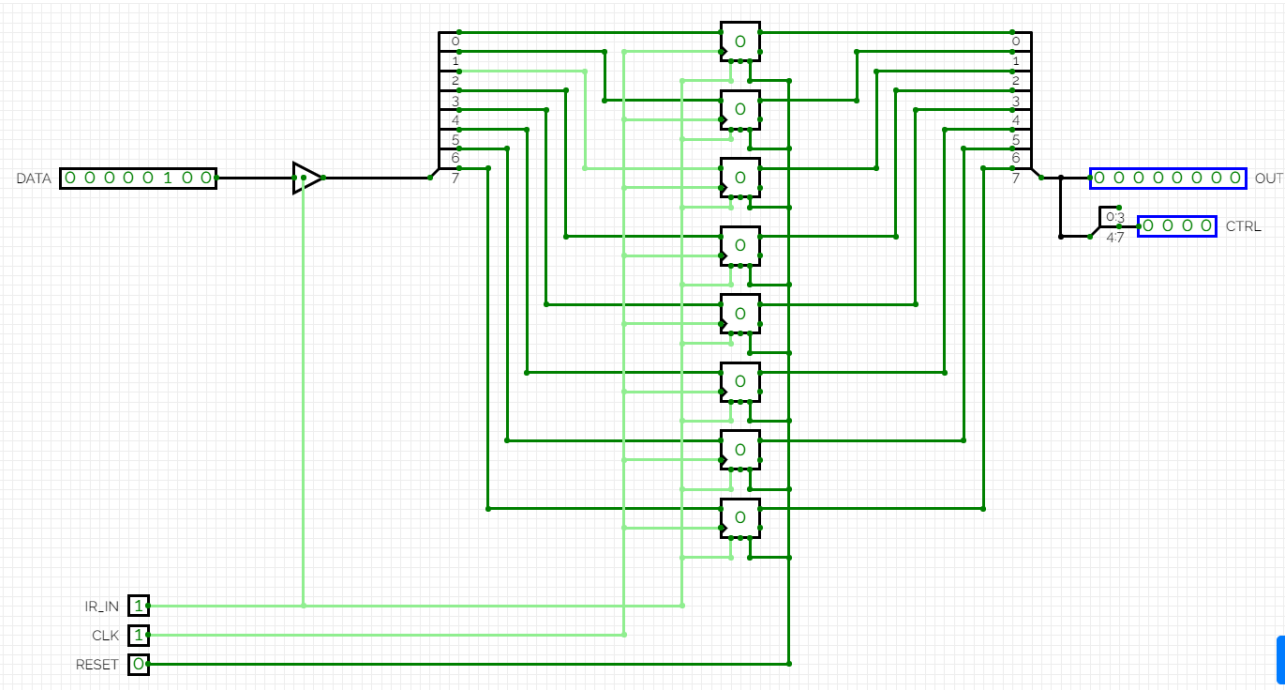


1.7 Instruction Register

- **Design** :

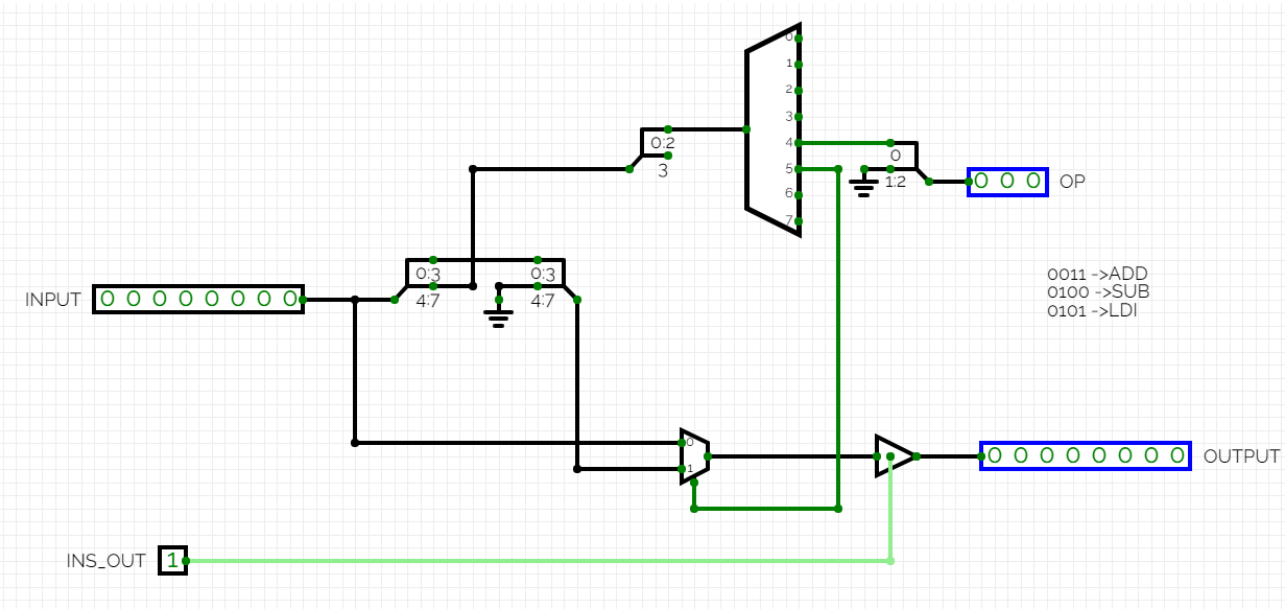
  1. When IR_in is set to 1,Input from common bus is taken and given to output and no tri state buffer is used to display output.
  2. When RESET is set to 1,Output is set to 0's.
  3. 4 Most significant digits of output are taken as CONTROL which is used as Instruction in Controller.

- **Circuit Diagram** :



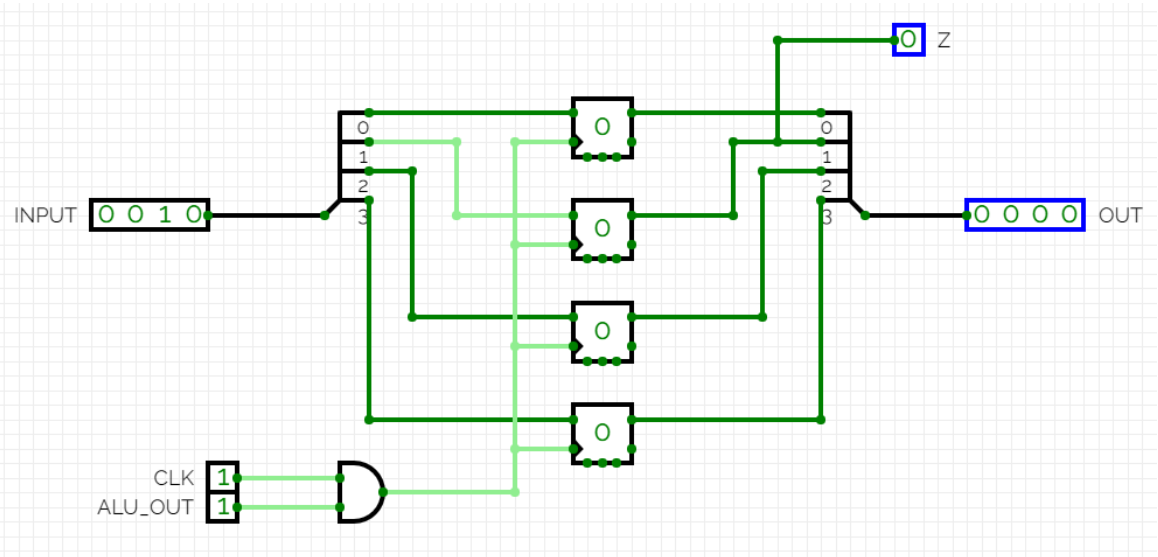
1.8 Instruction Decoder

- **Design** :
  1. Input is directly taken from reg_IR without any tri state buffer.
  2. First 4 most significant bits decide what operation to be done.
  3. When IR_OUT is 1,output is displayed.
  4. When Operation is LDI 4 Most significant bits of output becomes zeros'.
- **Circuit Diagram** :



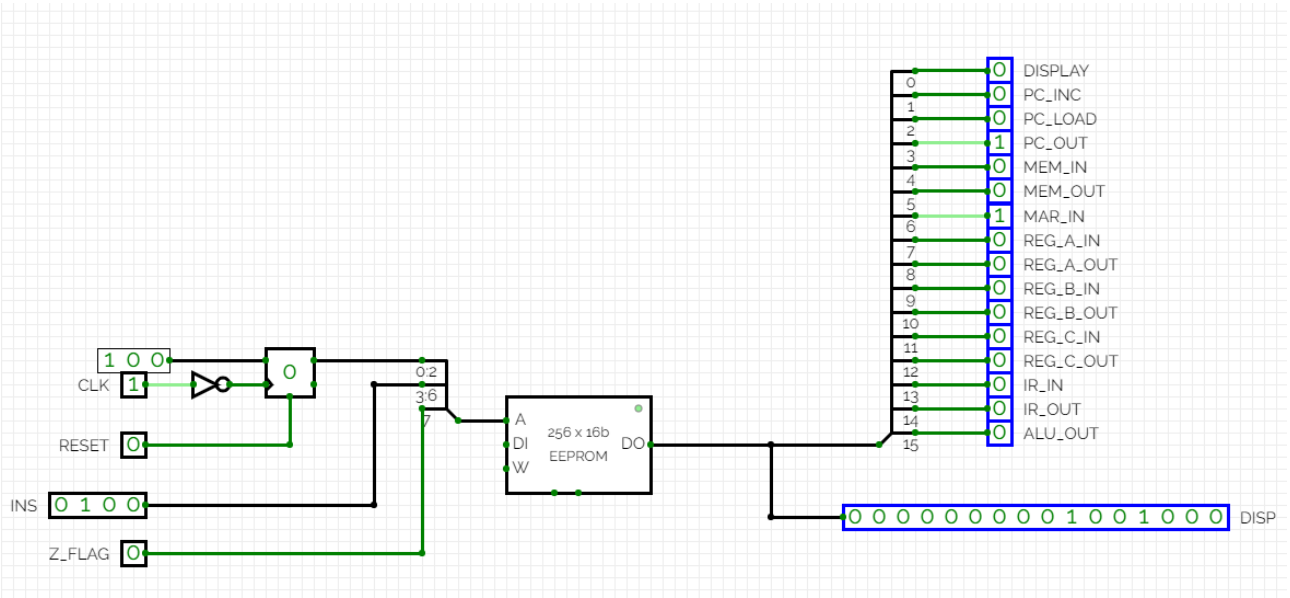
1.9 Status Register

- **Design** :
  A 4-bit status register for storing flags (C,NC,Z,NZ).
  When ALU_OUT is set to 1,Output is displayed.
- **Circuit Diagram** :



1.10 Controller

- **Design** :
  Takes an 8 bit input(Z_Flag,4 bit Instruction,3 bit T states counter) to give 16 bit output.
- **Circuit Diagram** :



2 Instruction Set and Machine codes

INSTRUCTION	Machine Code
NOP	0000 0x0
LDA	0001 0x1
STA	0010 0x2
ADD	0011 0x3
SUB	0100 0x4
LDI	0101 0x5
JMP	0110 0x6
SWAP	0111 0x7
JNZ	1000 0x8
MOVAC	1001 0x9
MOVBA	1010 0xa
MOVCB	1011 0xb
MOVAB	1100 0xc
MOVCA	1101 0xd
MOVBC	1110 0xe
HLT	1111 0xf

3 Description of Instruction Set :

3.1 NOP - No Operation

- **Description**: No operation is done.
- Operation : None

Syntax	Operands	Program Counter
NOP XXXX	X=0	PC ← PC+1

- 8 - bit opcode

0000	XXXX

3.2 LDA - Load Accumulator

- **Description**: This instruction loads the accumulator with the contents from the memory by looking at the 4 bit address provided
- Operation :
  Ra ← ROM

Syntax	Operands	Program Counter
LDA XXXX	X=0 or X=1	PC ← PC+1

- 8 - bit opcode

0001	XXXX

3.3 STA - Store At Address A

- **Description**: This instruction takes the value at accumulator and stores in memory at address A.
- Operation :
  ROM $\leftarrow$ Ra

Syntax	**Operands**	**Program Counter**
STA XXXX	X=0 or X=1	PC $\leftarrow$ PC+1

- 8 - bit opcode

0010	XXXX

3.4 ADD - Add Without Carry

- **Description**: This instruction loads the Register B with the contents from the memory by looking up at the 4 bit address provided and then adds up the contents of Accumulator and Register B and stores the final sum in Accumulator.
- Operation :
  Rb $\leftarrow$ ROM
  Ra $\leftarrow$ Ra + Rb

Syntax	**Operands**	**Program Counter**
ADD XXXX	X=0 or X=1	PC $\leftarrow$ PC+1

- 8 - bit opcode

0011	XXXX

3.5 SUB - Subtraction of positive numbers

- **Description**: This instruction loads the Register B with the contents from the memory by looking up at the 4 bit address provided and then subtracts up the contents of Register B from Accumulator and stores the final difference(non-negative) in Accumulator.
- Operation :
  Rb $\leftarrow$ ROM
  Ra $\leftarrow$ Ra $-$ Rb

Syntax	**Operands**	**Program Counter**
SUB XXXX	X=0 or X=1	PC $\leftarrow$ PC+1

- 8 - bit opcode

0100	XXXX

3.6 LDI - Load Immediate

- **Description**: Loads the 4 bit constant directly to Accumulator
- Operation :
  Ra ← k

Syntax	Operands	Program Counter
LDI XXXX	X=0 or X=1	PC ← PC+1

- 8 - bit opcode

0101	XXXX

3.7 JMP - Jump to Address

- **Description**:This Instruction takes the control to the given memory location.
- Operation :
  ROM Address

Syntax	Operands	Program Counter
JMP XXXX	X=0 or X=1	PC ← PC+1

- 8 - bit opcode

0110	XXXX

3.8 SWAP(AC) - Swap A and C

- **Description**: Swaps the contents present in Accumulator and Register C by temporarily storing contents of Register C in Register B during Swapping.
- Operation :
  Rb ← Ra
  Ra ← Rc
  Rc ← Rb

Syntax	Operands	Program Counter
SWAP XXXX	X=0	PC ← PC+1

- 8 - bit opcode

0111	XXXX

3.9 JNZ - Jump when Non-Zero

- **Description**: This Instruction takes the control to the given memory loction when the value retured after performing an operation is non-zero.
- Operation :
  if NZ==1 : ROM address : XXXX

Syntax	Operands	Program Counter
JNZ XXXX	X=0	PC ← PC+1

- 8 - bit opcode

1000	XXXX

3.10 MOVAC - Copy data from A to C

- **Description**: The contents present in Register A are loaded into the Register C
- Operation :
  Rc ← Ra

Syntax	Operands	Program Counter
MOVAC XXXX	X=0	PC ← PC+1

- 8 - bit opcode

1001	XXXX

3.11 MOVBA - Copy data from B to A

- **Description**: The contents present in Register B are loaded into the Register A.
- Operation :
  Ra ← Rb

Syntax	Operands	Program Counter
MOVBA XXXX	X=0	PC ← PC+1

- 8 - bit opcode

1010	XXXX

3.12 MOVCB - Copy data from C to B

- **Description**: The contents present in Register C are loaded into the Register B
- Operation :
  Rb ← Rc

Syntax	Operands	Program Counter
MOVCB XXXX	X=0	PC ← PC+1

- 8 - bit opcode

1011	XXXX

3.13 MOVAB - Copy data from A to B

- **Description**: The contents present in Register A are loaded into the Register B
- Operation :
  Rb ← Ra

Syntax	Operands	Program Counter
MOVAB XXXX	X=0	PC ← PC+1

- 8 - bit opcode

1100	XXXX

3.14 MOVCA - Copy data from C to A

- **Description**: The contents present in Register C are loaded into the Register A.
- Operation :
  Ra ← Rc

Syntax	Operands	Program Counter
MOVCA XXXX	X=0	PC ← PC+1

- 8 - bit opcode

1101	XXXX

3.15 MOVBC - Copy data from B to C

- **Description**: The contents present in Register B are loaded into the Register C
- Operation :
  Rc ← Rb

Syntax	Operands	Program Counter
MOVBC XXXX	X=0	PC ← PC+1

- 8 - bit opcode

1110	XXXX

3.16 HLT - End the Instruction

- **Description**: Fetch cycle stops.

Syntax	**Operands**	**Program Counter**
HLT XXXX	X=0	PC ← PC+1

- 8 - bit opcode

1111	XXXX

4 Assembly Programs implemented using the Instruction Set

4.1 Add two numbers and displaying the result

Address	Assembly Code	Machine Code
0x0	LDA 0x5	0x15
0x1	ADD 0x6	0x36
0x2	MOVAC 0x0	0x90
0x3	HLT 0x0	0xf0
0x4		0x00
0x5		0x34
0x6		0x12

- Value at address 0x5(34) is loaded to accumulator.
- Value at address 0x6(12) is added to value of accumulator(to give 46) and stored in it.
- Value in accumulator is moved to Register C for displaying.
- Instructions end here.

4.2 Adding and subtracting four numbers in some combination

Address	Assembly Code	Machine Code
0x0	LDA 0x5	0x15
0x1	SUB 0x6	0x46
0x2	ADD 0x7	0x37
0x3	SUB 0x8	0x48
0x4	HLT 0x0	0xf0
0x5		0x17
0x6		0x08
0x7		0x25
0x8		0x12

- Value at address 0x5(17) is loaded to accumulator.
- Value at address 0x6(08) is subtracted from value of accumulator(to give 0f) and stored in it.
- Value at address 0x7(25) is added to value of accumulator(to give 34) and stored in it.
- Value at address 0x8(12) is subtracted from value of accumulator(to give 22) and stored in it.
- Instructions end here.

4.3 Adding numbers from a starting address to ending address and displaying the result

Address	Assembly Code	Machine Code
0x0	LDI 0x2	0x52
0x1	ADD 0x8	0x38
0x2	ADD 0x9	0x39
0x3	ADD 0xa	0x3a
0x4	ADD 0xb	0x3b
0x5	ADD 0xc	0x3c
0x6	ADD 0xd	0x3d
0x7	HLT 0x0	0xf0
0x8		0x19
0x9		0x26
0xa		0x4b
0xb		0x04
0xc		0x10
0xd		0x09
0xe		0x00
0xf		0x00

- Value of 02 is loaded to accumulator.
- Value at address 0x8(19) is added to value of accumulator(to give 1b) and stored in it.
- Value at address 0x9(26) is added to value of accumulator(to give 41) and stored in it.
- Value at address 0xa(4b) is added to value of accumulator(to give 8c) and stored in it.
- Value at address 0xb(04) is added to value of accumulator(to give 90) and stored in it.
- Value at address 0xc(10) is added to value of accumulator(to give a0) and stored in it.
- Value at address 0xd(09) is added to value of accumulator(to give a9) and stored in it.
- Instructions end here.

4.4 Multiplication routine using repeated addition

Address	Assembly Code	Machine Code
0x0	LDA 0x9	0x19
0x1	SWAP 0x0	0x70
0x2	LDI 0x0	0x50
0x3	ADD 0x0	0x30
0x4	SWAP 0x0	0x70
0x5	SUB 0xb	0x4b
0x6	SWAP 0x0	0x70
0x7	JNZ 0x3	0x83
0x8	HLT 0x0	0xf0
0x9		0x06
0xa		0x13
0xb		0x01

- One number be x(06) and number be y(13).
- Value at address 0x9(06) is loaded to accumulator.
- Values in register A and C are interchanged. So,now register C has value 6.
- 0 is loaded to accumulator by LDI instructor.
- Value at EEPROM address 0xa(y=13) is added to accumulator(address of this addition process is 0x3)
- Values in register A and C are swapped and value at EEPROM address 0xb (01) is subtracted from accumulator(i.e, from x).
- Again values in Accumulator are swapped (i.e, register A has y and register C has x-1).
- Next instruction JNZ, it jumps to address 0x3 and repeats every intruction again from 0x3. This loop runs until Z_flag is 0.
- When flag becomes 1,JNZ stops and goes to next instruction(i.e, HLT 0x8).Now acculator contains the value x$*$y.
- Instructions end here.

5 Micro-instructions and Controller Logic Design

5.1 NOP

$1 \ll$ PC_OUT $\mid 1 \ll$ MAR_IN	T0
$1 \ll$ PC_INC $\mid 1 \ll$ MEM_OUT $\mid 1 \ll$ IR_IN	T1
0	T2
0	T3
0	T4

Micro-instruction for NOP

5.2 LDA

$1 \ll$ PC_OUT $\mid 1 \ll$ MAR_IN	T0
$1 \ll$ PC_INC $\mid 1 \ll$ MEM_OUT $\mid 1 \ll$ IR_IN	T1
$1 \ll$ IR_OUT $\mid 1 \ll$ MAR_IN	T2
$1 \ll$ MEM_OUT $\mid 1 \ll$ REGA_IN	T3
0	T4

Micro-instruction for LDA

5.3 STA

$1 \ll$ PC_OUT $\mid 1 \ll$ MAR_IN	T0
$1 \ll$ PC_INC $\mid 1 \ll$ MEM_OUT $\mid 1 \ll$ IR_IN	T1
$1 \ll$ IR_OUT $\mid 1 \ll$ MAR_IN	T2
$1 \ll$ MEM_IN $\mid 1 \ll$ REGA_OUT	T3
0	T4

Micro-instruction for STA

5.4 ADD

$1 \ll$ PC_OUT $\mid 1 \ll$ MAR_IN	T0
$1 \ll$ PC_INC $\mid 1 \ll$ MEM_OUT $\mid 1 \ll$ IR_IN	T1
$1 \ll$ IR_OUT $\mid 1 \ll$ MAR_IN	T2
$1 \ll$ MEM_OUT $\mid 1 \ll$ REGB_IN	T3
$1 \ll$ ALU_OUT $\mid 1 \ll$ REGA_IN	T4
0	T5

Micro-instruction for ADD

5.5 SUB

$1 \ll$ PC_OUT $\mid 1 \ll$ MAR_IN	T0
$1 \ll$ PC_INC $\mid 1 \ll$ MEM_OUT $\mid 1 \ll$ IR_IN	T1
$1 \ll$ IR_OUT $\mid 1 \ll$ MAR_IN	T2
$1 \ll$ MEM_OUT $\mid 1 \ll$ REGB_IN	T3
$1 \ll$ ALU_OUT $\mid 1 \ll$ REGA_IN	T4

Micro-instruction for SUB

5.6 LDI

$1 \ll$ PC_OUT $\mid 1 \ll$ MAR_IN	T0
$1 \ll$ PC_INC $\mid 1 \ll$ MEM_OUT $\mid 1 \ll$ IR_IN	T1
$1 \ll$ IR_OUT $\mid 1 \ll$ REGA_IN	T2
0	T3
0	T4

Micro-instruction for LDI

5.7 JMP

$1 \ll$ PC_OUT $\mid 1 \ll$ MAR_IN	T0
$1 \ll$ PC_INC $\mid 1 \ll$ MEM_OUT $\mid 1 \ll$ IR_IN	T1
$1 \ll$ IR_OUT $\mid 1 \ll$ PC_LOAD	T2
0	T3
0	T4

Micro-instruction for JMP

5.8 SWAP

$1 \ll$ PC_OUT $\mid 1 \ll$ MAR_IN	T0
$1 \ll$ PC_INC $\mid 1 \ll$ MEM_OUT $\mid 1 \ll$ IR_IN	T1
$1 \ll$ REGA_OUT $\mid 1 \ll$ REGB_IN	T2
$1 \ll$ REGC_OUT $\mid 1 \ll$ REGA_IN	T3
$1 \ll$ REGB_OUT $\mid 1 \ll$ REGC_IN	T4
0	T5

Micro-instruction for SWAP

5.9 JNZ

$1 \ll$ PC_OUT $\mid 1 \ll$ MAR_IN	T0
$1 \ll$ PC_INC $\mid 1 \ll$ MEM_OUT $\mid 1 \ll$ IR_IN	T1
$1 \ll$ IR_OUT $\mid 1 \ll$ PC_LOAD	T2
0	T3
0	T4

Micro-instruction for JNZ, If Flag(Z) is 0.

$1 \ll$ PC_OUT $\mid 1 \ll$ MAR_IN	T0
$1 \ll$ PC_INC $\mid 1 \ll$ MEM_OUT $\mid 1 \ll$ IR_IN	T1
0	T2
0	T3
0	T4

Micro-instruction for JNZ, If Flag(Z) is 1.

5.10 MOVAC

$1 \ll$ PC_OUT $\mid 1 \ll$ MAR_IN	T0
$1 \ll$ PC_INC $\mid 1 \ll$ MEM_OUT $\mid 1 \ll$ IR_IN	T1
$1 \ll$ REGA_OUT $\mid 1 \ll$ REGC_IN	T2
0	T3
0	T4

Micro-instruction for MOVAC

5.11 MOVBA

$1 \ll$ PC_OUT $\mid 1 \ll$ MAR_IN	T0
$1 \ll$ PC_INC $\mid 1 \ll$ MEM_OUT $\mid 1 \ll$ IR_IN	T1
$1 \ll$ REGB_OUT $\mid 1 \ll$ REGA_IN	T2
0	T3
0	T4

Micro-instruction for MOVBA

5.12 MOVCB

$1 \ll$ PC_OUT $\mid 1 \ll$ MAR_IN	T0
$1 \ll$ PC_INC $\mid 1 \ll$ MEM_OUT $\mid 1 \ll$ IR_IN	T1
$1 \ll$ REGC_OUT $\mid 1 \ll$ REGB_IN	T2
0	T3
0	T4

Micro-instruction for MOVCB

5.13 MOVAB

$1 \ll$ PC_OUT | $1 \ll$ MAR_IN	T0
$1 \ll$ PC_INC | $1 \ll$ MEM_OUT | $1 \ll$ IR_IN	T1
$1 \ll$ REGA_OUT | $1 \ll$ REGB_IN	T2
0	T3
0	T4

Micro-instruction for MOVAB

5.14 MOVCA

$1 \ll$ PC_OUT | $1 \ll$ MAR_IN	T0
$1 \ll$ PC_INC | $1 \ll$ MEM_OUT | $1 \ll$ IR_IN	T1
$1 \ll$ REGC_OUT | $1 \ll$ REGA_IN	T2
0	T3
0	T4

Micro-instruction for MOVCA

5.15 MOVBC

$1 \ll$ PC_OUT | $1 \ll$ MAR_IN	T0
$1 \ll$ PC_INC | $1 \ll$ MEM_OUT | $1 \ll$ IR_IN	T1
$1 \ll$ REGB_OUT | $1 \ll$ REGC_IN	T2
0	T3
0	T4

Micro-instruction for MOVBC

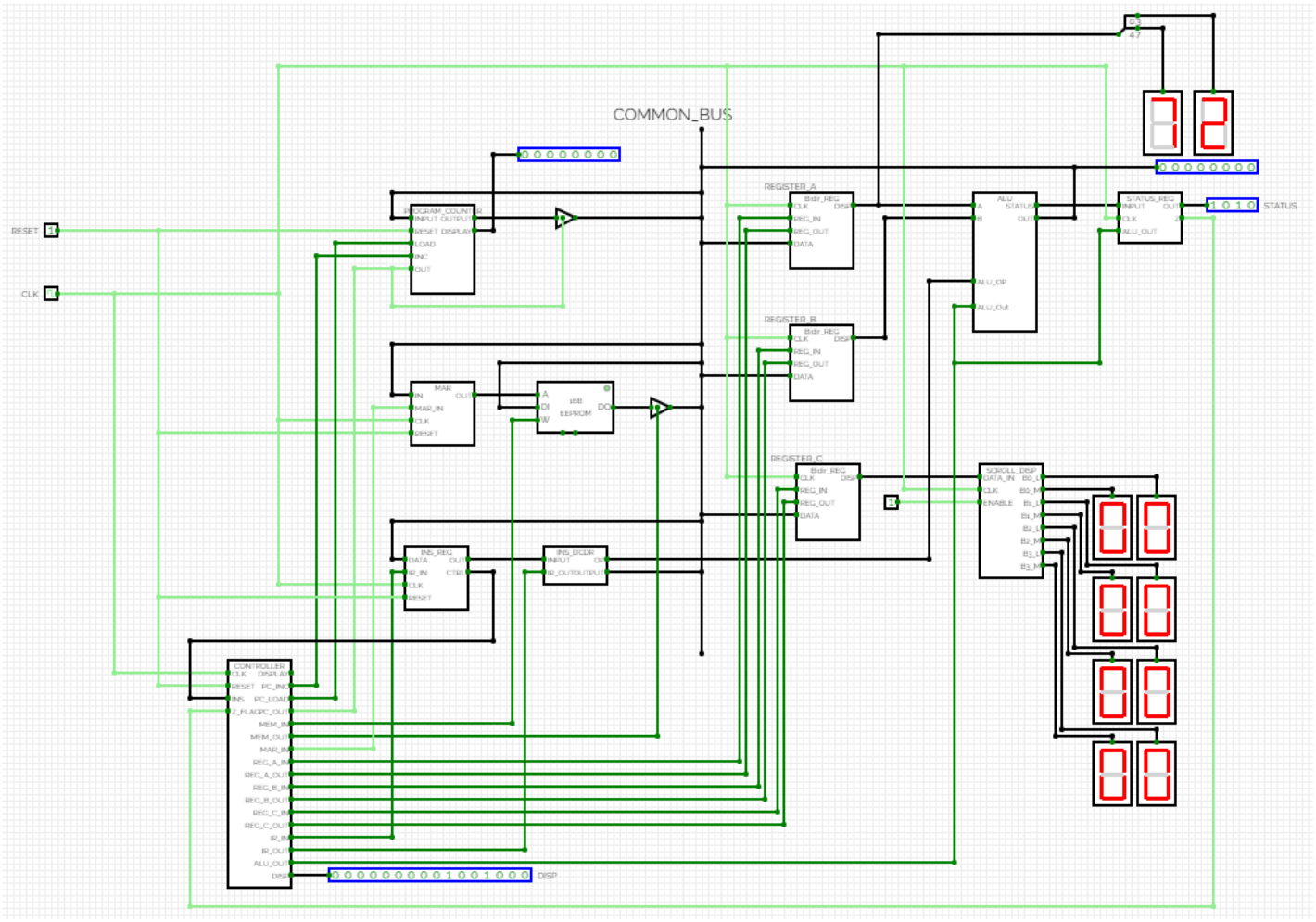
5.16 HLT

$1 \ll$ MAR_IN	T0
$1 \ll$ MEM_OUT | $1 \ll$ IR_IN	T1
0	T2
0	T3

Micro-instruction for HLT

Fetch cycle is not used in HALT instruction due to this circuit works only one time,to avoid this we have used RESET in reg_IR so that at the start of function Fetch Cycle of NOP runs.

6 Sample Programs



The Sample programs which we tried on this Circuit are,

- Add two numbers and displaying the result
  0x15,0x36,0x90,0xf0,0x00,0x34,0x12
- Adding and subtracting four numbers in some combination
  0x15,0x46,0x37,0x48,0xf0,0x17,0x08,0x25,0x12
- Adding numbers from a starting address to ending address and displaying the result
  0x52,0x38,0x39,0x3a,0x3b,0x3c,0x3d,0xf0,0x19,0x26,0x4b,0x04,0x10,0x09
- Multiplication routine using repeated addition
  0x19,0x70,0x50,0x30,0x70,0x4b,0x70,0x83,0xf0,0x06,0x13,0x01