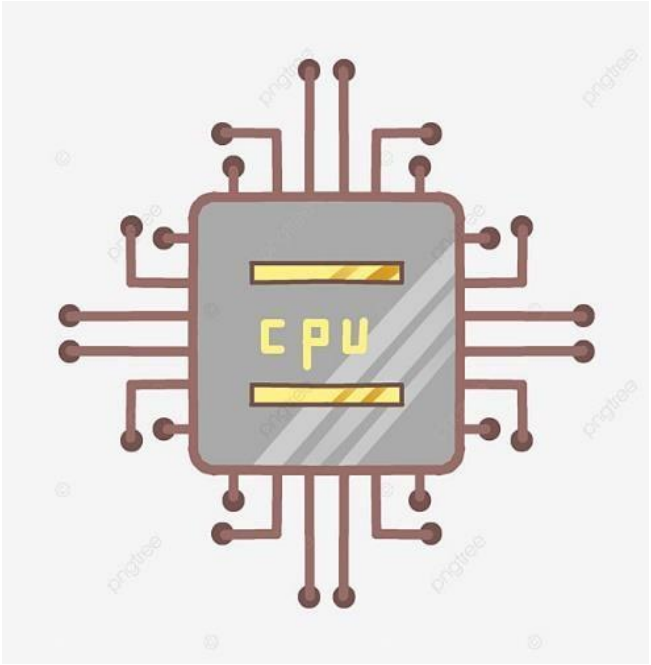


GAJENDRA-ALU

Simple Processor

SECTION-1:

Description of INTERNAL COMPONENTS

SECTION-2:

Description of INSTRUCTIONS SET **SECTION-**

3:

Examples of Assembly Programs implemented using the Instruction Set

SECTION-4:

Micro-instructions and Controller Logic Design

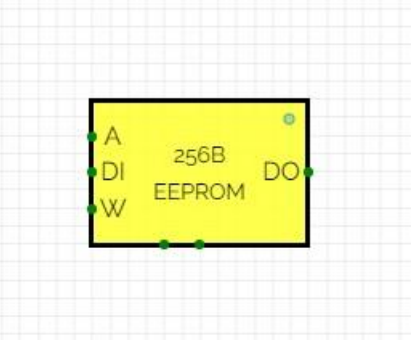
I. Internal Components

1.Memory

Design:

Memory is a EPROM which contains instructions and data required for a program.

Circuit:



2. General CPU Register

Design:

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

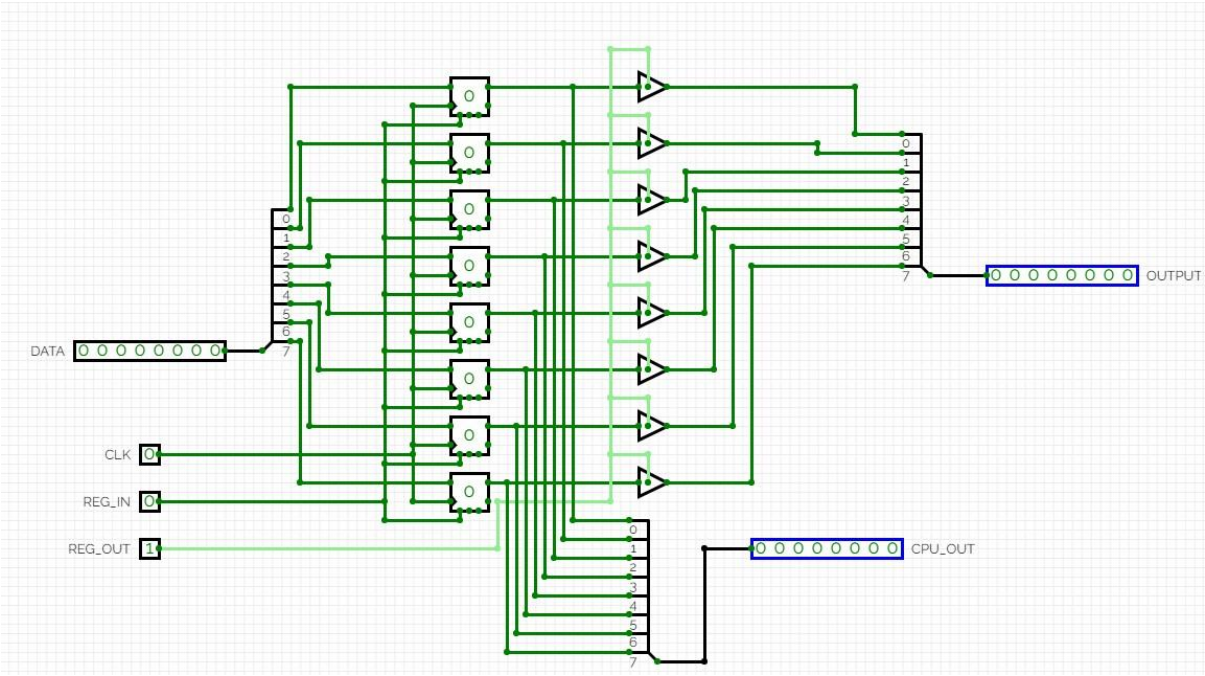
The Working of control inputs are

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 8bit output, and it shows the current value stored in the register. This output is continuously updated with the data in the register

Circuit:

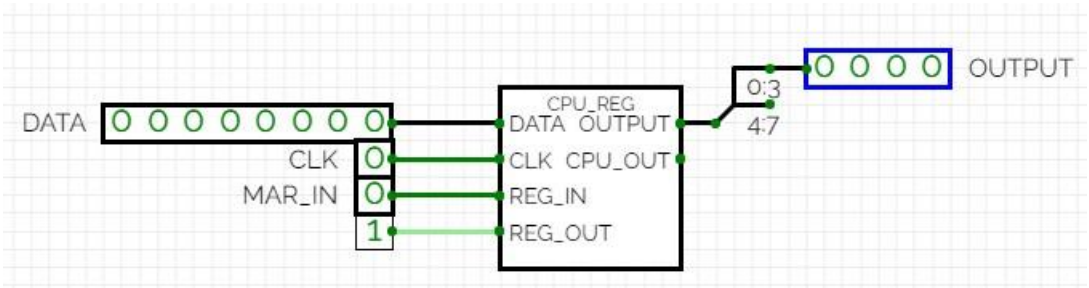


3. Memory Address Register

Design:

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

Circuit:

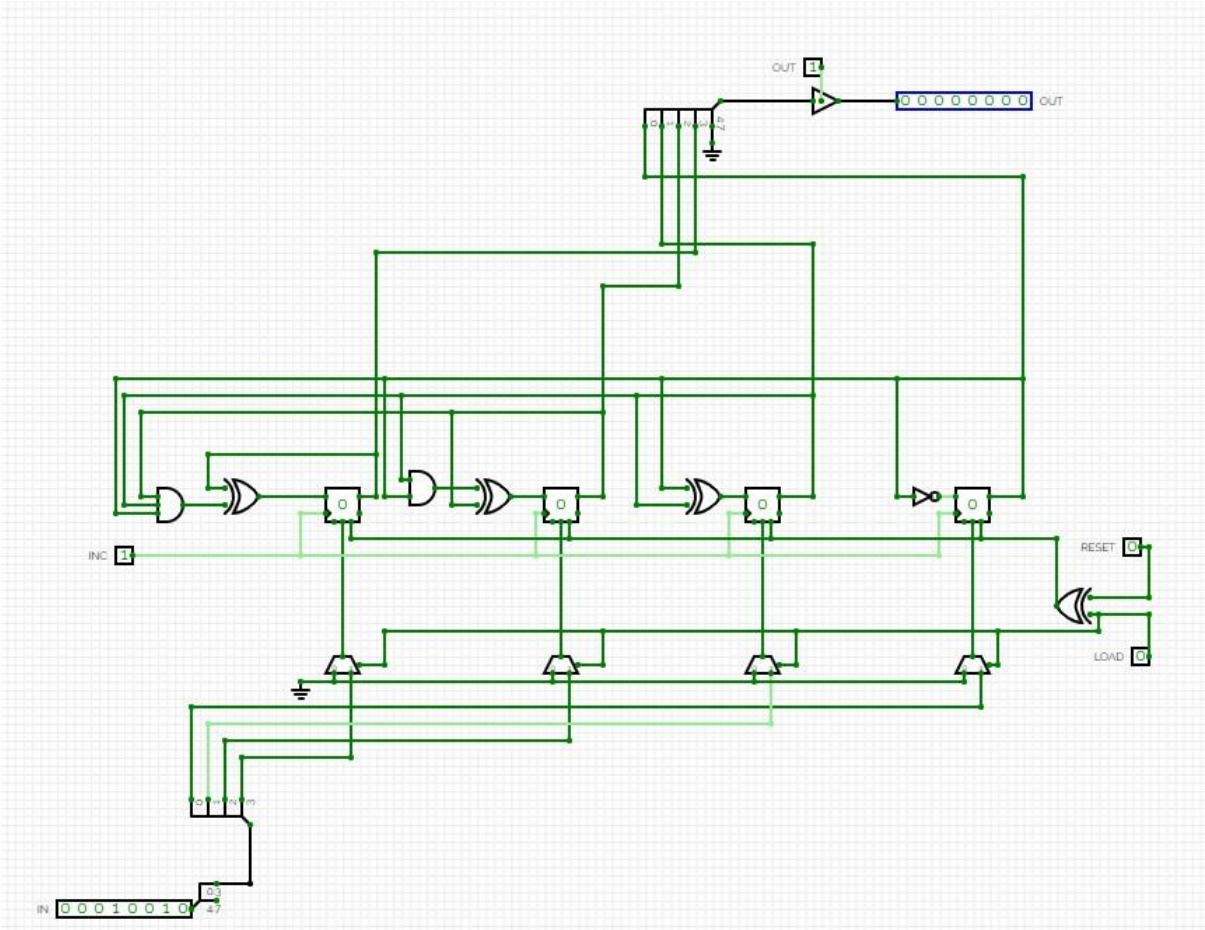


4.Program Counter

Design:

1. The Program Counter points to the address of the instruction in ROM, that has to be processed next.
2. When PC Load 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.

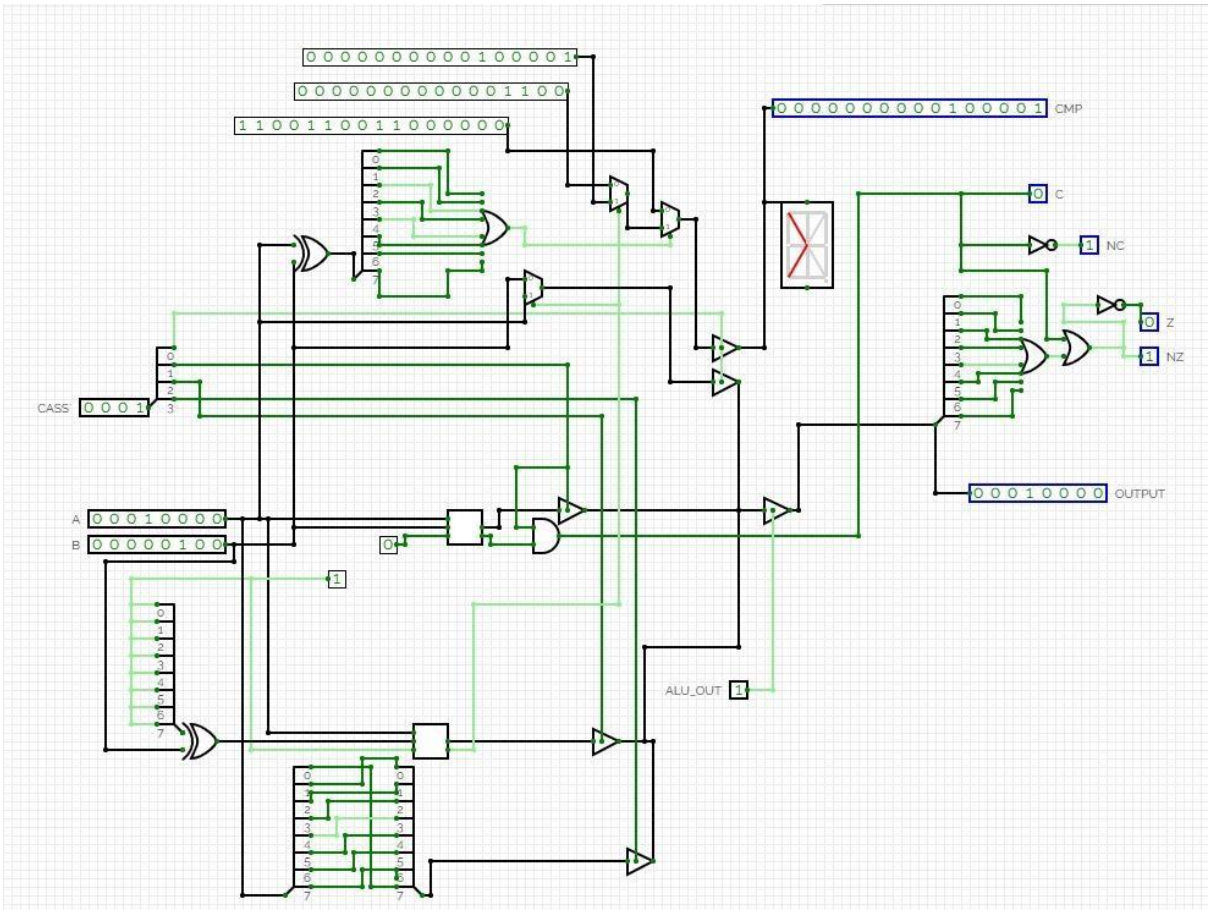
Circuit:



5. Arithmetic and Logical Unit (ALU)

Design:

Circuit:

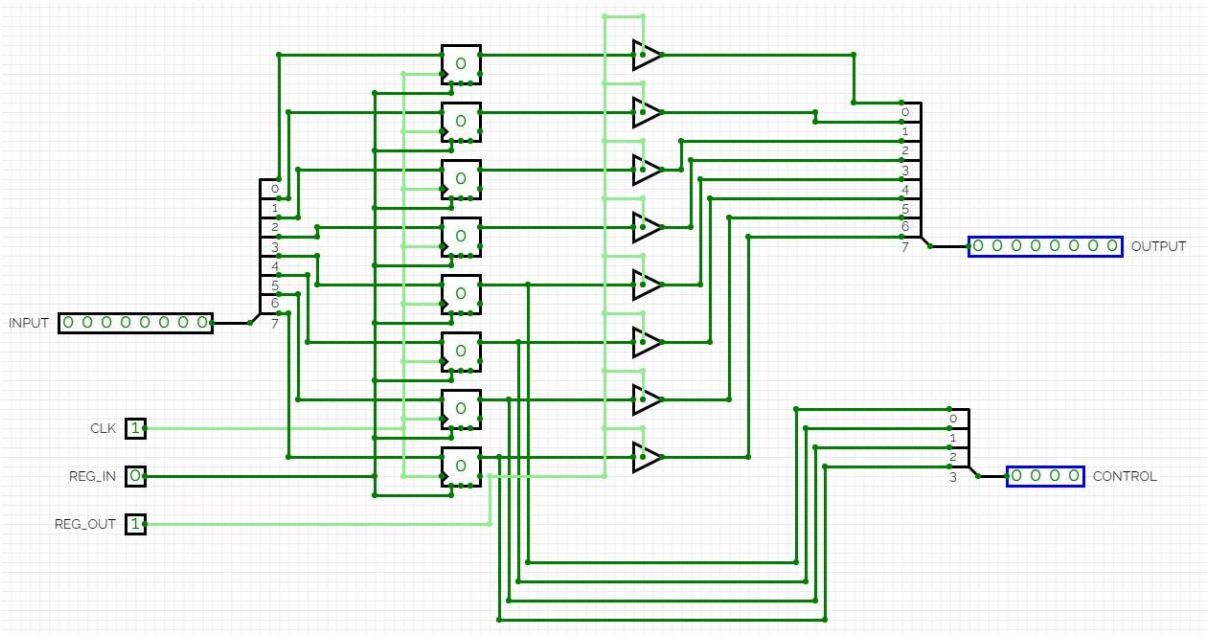


6.Instruction Register

Design:

1. When REG_IN is set to 1, Input from common bus is taken and given to register.
2. When REG_OUT is set to 1, Input from register is taken output.
3. 4 Most significant digits of output are taken as CONTROL which is used as Instruction in Controller.

Circuit:



7. Instruction Decoder

Design:

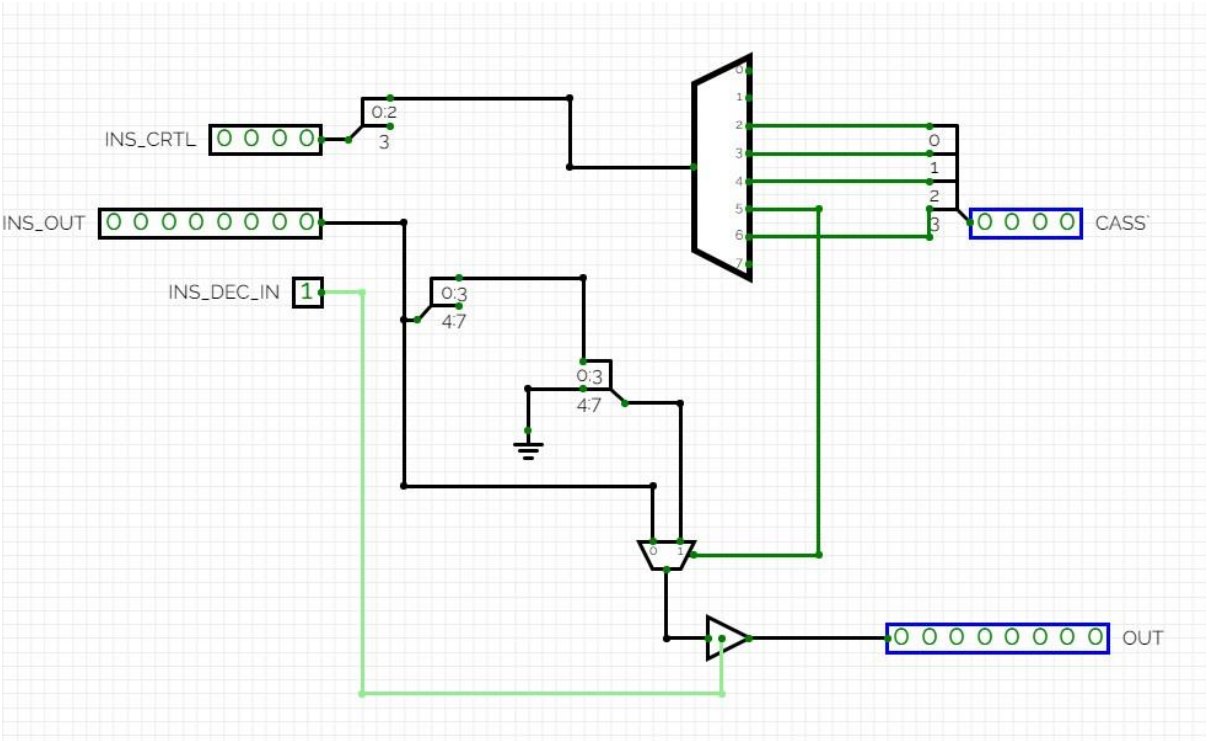
1. Input is directly taken from Instruction Register.

2. First 4 most significant bits decide what operation to be done.

3. When REG_OUT is 1, output is displayed.

4. When Operation is LDI 4 Most significant bits of output becomes zero' s.

Circuit:

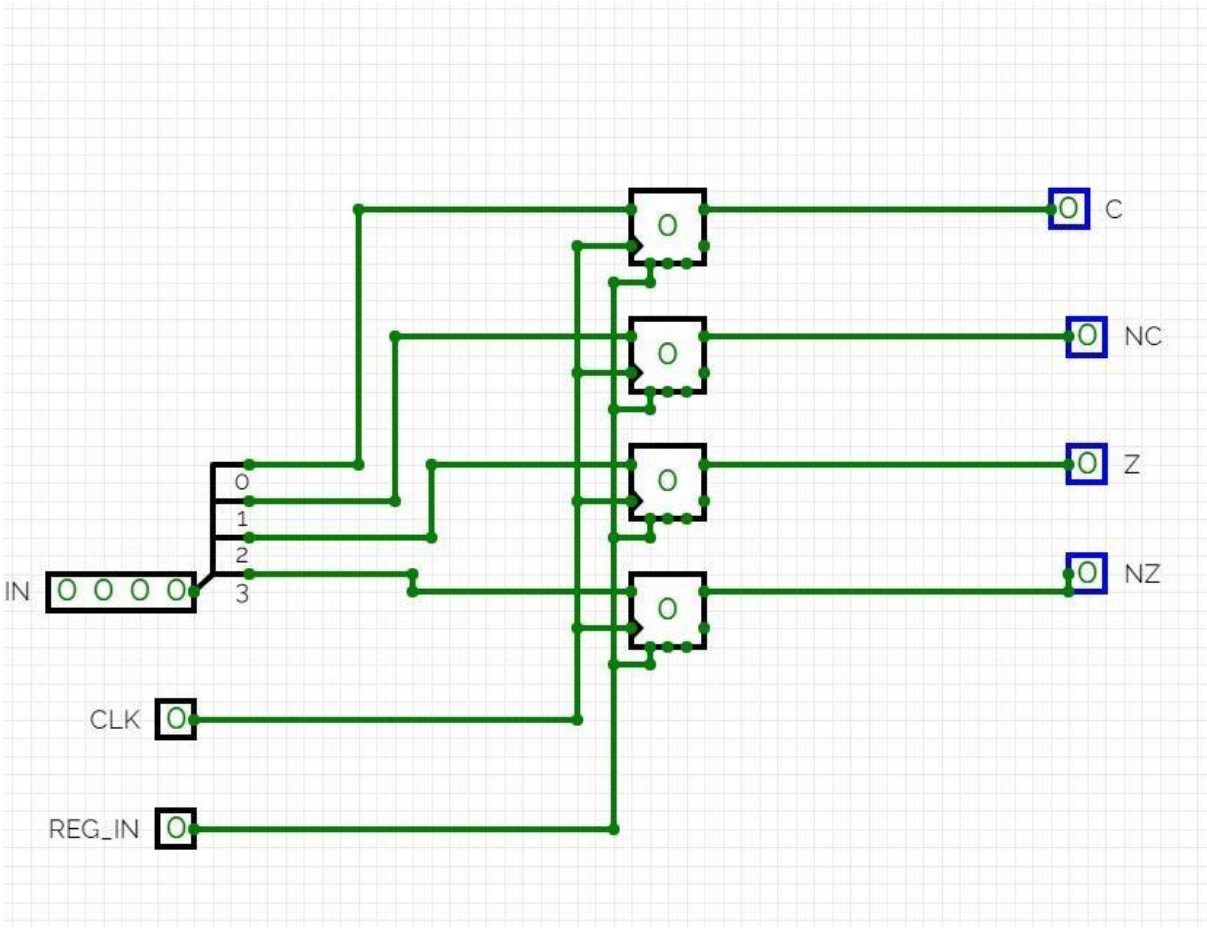


8.Status Register

Design:

A 4-bit status register for storing flags (C, NC, Z, NZ).

Circuit:



controller

2.1 Instruction and Machine Code

INSTUCTIONS	MACHINE CODE
NOP	0000 0X0
LDA	0001 0X1
STA	0010 0X2
ADD	0011 0X3
SUB	0100 0X4

(i)

LDI	0101 0X5
JMP	0110 0X6
SWAP	0111 0X7
JNZ	1000 0X8
MOVAC	1001 0X9
CMP	1010 0XA
MOVCB	1011 0XB
MOVAB	1100 0XC
MOVCA	1101 0XD
SHIFT(LR)	1110 0XE
HLT	1111 0XF

Description of Instruction Set:

NOP - No Operation

Description No operation is done.

Operation

No Operation

Syntax:	Operands:	Program counter:
(ii) NOP XXXX	x=0	PC←PC+1

8 Bit- Op-Code:

0000	XXXX

LDA – Load to Accumulator

Description

Loads Register A with the value of the common bus (the value obtained from the memory). Operation

(i) $R_A \leftarrow x$

Syntax:	Operands:	Program counter:
(ii) LDA XXXX	0≤ x ≤255	PC←PC+1

8 Bit- Op-Code:

0001	XXXX

STA - Store At Address A Description

This instruction takes the value at accumulator and stores in memory at address A.

(i)

Operation

(i)

ROM(Memory) ← Ra

Syntax:	Operands:	Program counter:
(ii) STA XXXX	0≤ x ≤255	PC←PC+1

8 Bit- Op-Code:

0001	XXXX

ADD – Add without carry

Description

Adds the value in register B to the value in register A and places the result in register A. Operation

(i) $R_B \leftarrow x$

$R_A \leftarrow R_A + R_B$

Syntax:	Operands:	Program counter:
(ii) ADD XXXX	0≤ x ≤255	PC←PC+1

8 Bit- Op-Code:

0011	XXXX

SUB – subtraction Description

Subtracts the value in register B to the value in register A and places the result in register A (if the result is negative we place the 2's complement).

Operation

$R_B \leftarrow x$

(i)

$R_A \leftarrow R_A - R_B$

Syntax:	Operands:	Program counter:
(ii) SUB XXXX	0≤ x ≤255	PC←PC+1

8 Bit- Op-Code:

0100	XXXX

LDI – Load Immediate Description

Loads Register A with the value of the common bus.

Operation

(i) $R_A \leftarrow X$

Syntax:	Operands:	Program counter:
(ii) LDI XXXX	x=0	PC-PC+11

8 Bit- Op-Code:

0101	XXXX

JMP- Jumps to Address Description

This Instruction takes the control to the given memory location.

Operation

(i) Jumps to the given address

Syntax:	Operands:	Program counter:
(ii) JMP XXXX	0≤ x ≤255	PC←PC+1

8 Bit- Op-Code:

0110	XXXX

SWAP(AB)- Swap A and B

Description

Swaps the contents present in Accumulator and Register B by temporarily storing contents of Register B in Register C during Swapping.

Operation

(i) Rc ← Rb

Rb ← Ra

Ra ← Rc

Syntax:	Operands:	Program counter:
(ii) SWP XXXX	0≤ x ≤255	PC←PC+1

8 Bit- Op-Code:

0111	XXXX

JNZ – Jump when Non-Zero

Description

This Instruction takes the control to the given memory location when the value returned after performing an operation is non-zero.

Operation

(i) if NZ==1: ROM address: XXXX

	Syntax:	Operands:	Program counter:
(ii)	JNZ XXXX	0≤ x ≤255	PC←PC+1

8 Bit- Op-Code:

1000	XXXX

MOVAC - Copy data from A to C

Description

The contents present in Register A are loaded into the Register C

Operation

(i) Rc ← Ra

	Syntax:	Operands:	Program counter:
(ii)	MOVAC XXXX	x=0	PC←PC+1

8 Bit- Op-Code:

1001	XXXX

CMP – Compares the values of A and B Description

Compares the values stored in register A and register B and display the result and shows the symbol as well as the larger will be loaded into register C.

Operation

(i)

	Syntax:	Operands:	Program counter:
(ii)	CMP XXXX	0≤x≤255	PC←PC+1

8 Bit- Op-Code:

1010	XXXX

MOVCB - Copy data from C to B

Description

The contents present in Register C are loaded into the Register B.

Operation

(i) $R_B \leftarrow R_C$

	Syntax:	Operands:	Program counter:
(ii)	MOVCB XXXX	0≤x≤255	PC←PC+1

8 Bit- Op-Code:

1011	XXXX

MOVAB - Copy data from A to B

Description:

The contents present in Register A are loaded into the Register B.

Operation

(i) $R_B \leftarrow R_A$

	Syntax:	Operands:	Program counter:
(ii)	MOVAB XXXX	0≤x≤255	PC←PC+1

8 Bit- Op-Code:

1100	XXXX

MOVCA - Copy data from C to A

Description:

The contents present in Register C are loaded into the Register A.

Operation

(i) $R_C \leftarrow R_A$

	Syntax:	Operands:	Program counter:
(ii)	MOVCA XXXX	0≤x≤255	PC←PC+1

8 Bit- Op-Code:

1101	XXXX

SHIFT(LR)- shift each bit towards right Description

The contents present in Register A are shifted by one bit and output into A again

Operation

(i) $R_B \leftarrow Max(R_A,R_B)$

	Syntax:	Operands:	Program counter:
(ii)	SHIFT XXXX	0≤x≤255	PC←PC+1

8 Bit- Op-Code:

1110	XXXX

HLT - End the Instruction

Description

Fetch cycle stops.

Operation

(i)

Syntax:	Operands:	Program counter:
(ii) HLT XXXX	0≤x≤255	PC←PC+1

8 Bit- Op-Code:

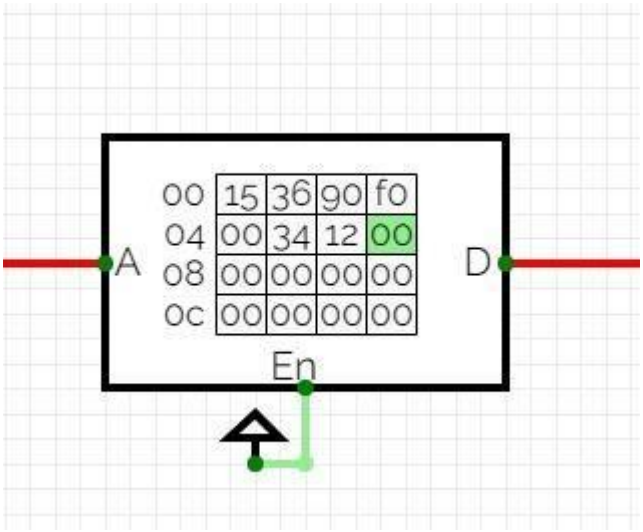
1111	XXXX

3 Assembly Programs implemented using the Instruction Set

3.1 Adding two numbers and displaying the result

INSTRUCTIONS	ASSEMBLY CODE	MACHINE CODE
0X1	LDA 0X5	0X15
0X2	ADD 0X6	0X36
0X3	MOVAC 0X0	0X90
0X4	HLT 0X0	0XF0

0X5		0X00
0X6		0X34
0X7		0X12



1.Value at address 0X5(34) is loaded to accumulator.

2.Value at address 0X6(12) is added to value of accumulator (to give 46) and stored in it.

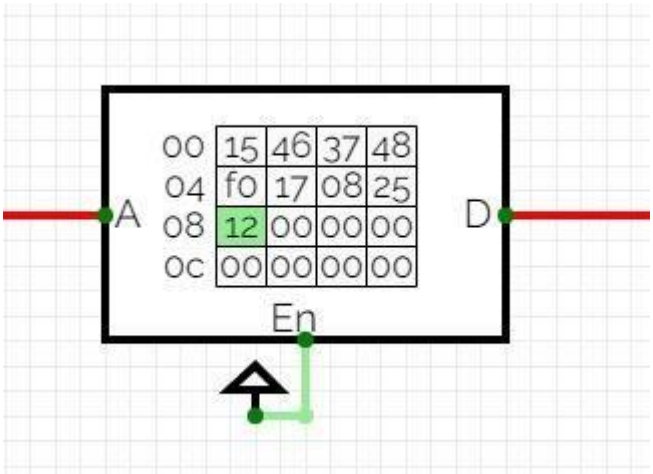
3.Value in accumulator is moved to Register C for displaying.

4.Instructions end here.

3.2 Adding and subtracting four numbers in some combination

INSTRUCTIONS	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.

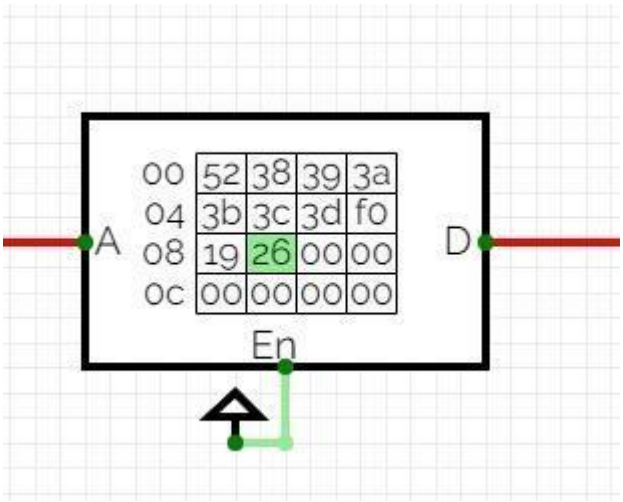
Value at address 0X8(12) is subtracted from value of accumulator (to give 22) and stored in it.

Instructions end here.

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

INSTRUCTIONS	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.

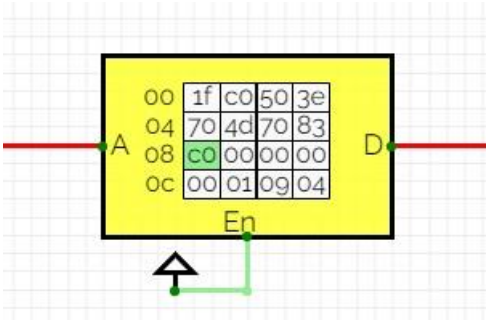
Value at address 0Xc (10) is added to value of accumulator (to give a0) and stored in it.

Value at address 0Xd (09) is added to value of accumulator (to give a9) and stored in it.

Instructions end here

3.4 Multiplication of two numbers using repeated addition

INSTRUCTIONS	ASSEMBLY CODE	MACHINE CODE
0X0	LDA 0xf	0x1f
0X1	MOVEAB 0x0	0xC0
0X2	LDI 0x0	0x50
0X3	ADD 0xe	0x3e
0X4	SWAP 0x0	0x70
0X5	SUB 0xd	0x4d
0X6	SWAP 0x0	0x70
0X7	JNZ 0x3	0x83
0X8	MOVEAB 0x0	0x90
0X9	HLT 0x0	0xf0
0Xa		
0Xb		
0Xc		
0Xd		0x1
0Xe		0X9
0Xf		0X4



Value at address 0xf (04) is loaded to accumulator.

Value at accumulator is moved to Register B.

Accumulator value is initialised to zero value.

Value at address 0xe (09) is added to accumulator stored in it.

Swapping of accumulator and Register B value.

Now, accumulator value is 4(Say X) , which is reduced to 3 using SUB . Also , checked if accumulator value become zero or not by zeroflag.

Swapping of accumulator and Register B value.

Until the value X become zero, the program will run by loop by jumping to the address 0X3.

When X =0, comes out of loop .

The value at B is moved to accumulator which is 9x4=36.

Halt will stop the program.

Instructions end here.

4 SECTION 4 - Micro-instructions and Controller Logic Design

4.1 NOP

1 ≪ PC OUT | 1 ≪ MAR IN	T0
1 ≪	T1
0	T2
0 PC INC | 1 ≪ MEM OUT | 1 ≪ IR IN	T3

0	T4

Micro-instruction for NOP

4.2LDA

1 ≪ PC OUT | 1 ≪ MAR IN	T0
1 ≪ PC INC | 1 ≪ MEM OUT | 1 ≪ IR IN	T1
1 ≪ IR OUT | 1 ≪ MAR IN	T2
1 ≪ MEM OUT | 1 ≪ REGA IN	T3
0	T4

Micro-instruction for LDA

4.3 STA

1 ≪ PC OUT | 1 ≪ MAR IN	T0
1 ≪ PC INC | 1 ≪ MEM OUT | 1 ≪ IR IN	T1
1 ≪ IR OUT | 1 ≪ MAR IN	T2
1 ≪ MEM IN | 1 ≪ REGA OUT	T3
0	T4

Micro-instruction for STA

5.4 ADD

1 ≪ PC OUT | 1 ≪ MAR IN	T0
1 ≪ PC INC | 1 ≪ MEM OUT | 1 ≪ IR IN	T1
1 ≪ IR OUT | 1 ≪ MAR IN	T2
1 ≪ MEM OUT | 1 ≪ REGB IN	T3
1 ≪ ALU OUT | 1 ≪ REGA IN	T4
0	T5

Micro-instruction for ADD

4.5 SUB

1 ≪ PC OUT | 1 ≪ MAR IN	T0
1 ≪ PC INC | 1 ≪ MEM OUT | 1 ≪ IR IN	T1
1 ≪ IR OUT | 1 ≪ MAR IN	T2
1 ≪ MEM OUT | 1 ≪ REGB IN	T3
1 ≪ ALU OUT | 1 ≪ REGA IN	T4

Micro-instruction for SUB

4.6LDI

1 ≪ PC OUT | 1 ≪ MAR IN	T0
1 ≪ PC INC | 1 ≪ MEM OUT | 1 ≪ IR IN	T1
1 ≪ IR OUT | 1 ≪ REGA IN	T2
0	T3

0	T4

Micro-instruction for LDI

4.7 JMP

1 ≪ PC OUT | 1 ≪ MAR IN	T0
1 ≪ PC INC | 1 ≪ MEM OUT | 1 ≪ IR IN	T1
1 ≪ IR OUT | 1 ≪ PC LOAD	T2
0	T3
0	T4

Micro-instruction for JMP

4.8 SWAP

1 ≪ PC OUT | 1 ≪ MAR IN	T0
1 ≪ PC INC | 1 ≪ MEM OUT | 1 ≪ IR IN	T1
1 ≪ REGA OUT | 1 ≪ REGC_IN	T2
1 ≪ REGB_OUT | 1 ≪ REGA_IN	T3
1 ≪ REGC OUT | 1 ≪ REGB IN	T4
0	T5

Micro-instruction for SWAP

4.9 JNZ

1 ≪ PC OUT | 1 ≪ MAR IN	T0
1 ≪ PC INC | 1 ≪ MEM OUT | 1 ≪ IR IN	T1
1 ≪ IR OUT | 1 ≪ PC LOAD	T2
0	T3
0	T4

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

1 ≪ PC OUT | 1 ≪ MAR IN	T0
1 ≪ PC INC | 1 ≪ MEM OUT | 1 ≪ IR IN	T1

0	T2
0	T3
0	T4

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

4.10 MOVAC

1 ≪ PC OUT | 1 ≪ MAR IN	T0
1 ≪ PC INC | 1 ≪ MEM OUT | 1 ≪ IR IN	T1
1 ≪ REGA OUT | 1 ≪ REGC_IN	T2
0	T3
0	T4

Micro-instruction for MOVAC

4.11 COMP

1 ≪ PC OUT | 1 ≪ MAR IN	T0
1 ≪ PC INC | 1 ≪ MEM OUT | 1 ≪ IR IN	T1
1 ≪ REGC_IN | 1 ≪ ALU_OUT	T2
0	T3
0	T4

Micro-instruction for MOVBA

4.12 MOVCB

1 ≪ PC OUT | 1 ≪ MAR IN	T0
1 ≪ PC INC | 1 ≪ MEM OUT | 1 ≪ IR IN	T1
1 ≪ REGC OUT | 1 ≪ REGB IN	T2

0	T3
0	T4

Micro-instruction for MOVCB

4.13 MOVAB

1 ≪ PC OUT | 1 ≪ MAR IN	T0
1 ≪ PC INC | 1 ≪ MEM OUT | 1 ≪ IR IN	T1
1 ≪ REGA OUT | 1 ≪ REGB_IN	T2
0	T3
0	T4

Micro-instruction for MOVAB

4.14 MOVCA

1 ≪ PC OUT | 1 ≪ MAR IN	T0
1 ≪ PC INC | 1 ≪ MEM OUT | 1 ≪ IR IN	T1
1 ≪ REGC OUT | 1 ≪ REGA_IN	T2
0	T3
0	T4

Micro-instruction for MOVCA

4.15 SHIFT(LR)

1 ≪ PC OUT | 1 ≪ MAR IN	T0
1 ≪ PC INC | 1 ≪ MEM OUT | 1 ≪ IR IN	T1
1 ≪ REGA_IN | 1 ≪ ALU_OUT	T2
0	T3
0	T4

Micro-instruction for MOVBC

4.16 HLT

1 ≪ MAR IN	T0
1 ≪ MEM_OUT | 1 ≪ 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 Section 5

6.1 Subsection 5.1

6.2 Subsection 5.2