GAJENDRA PROCESSOR

JULY-NOV 2023

PROF: AYON CHAKRABORTY

Submitted by:
1)Kanakamedala Vibhav Chowdary CS22B028
2)Pranjal Nandakumar CS22B042

CONTENTS

- ADD
- SUB
- LDI
- JMP
- SWAP
- JNZ
- MOVAC
- MOVCA
- OUT
- LSL
- LSR
- MOVAB
- HLT

6. Implementation

INTERNAL COMPONENTS

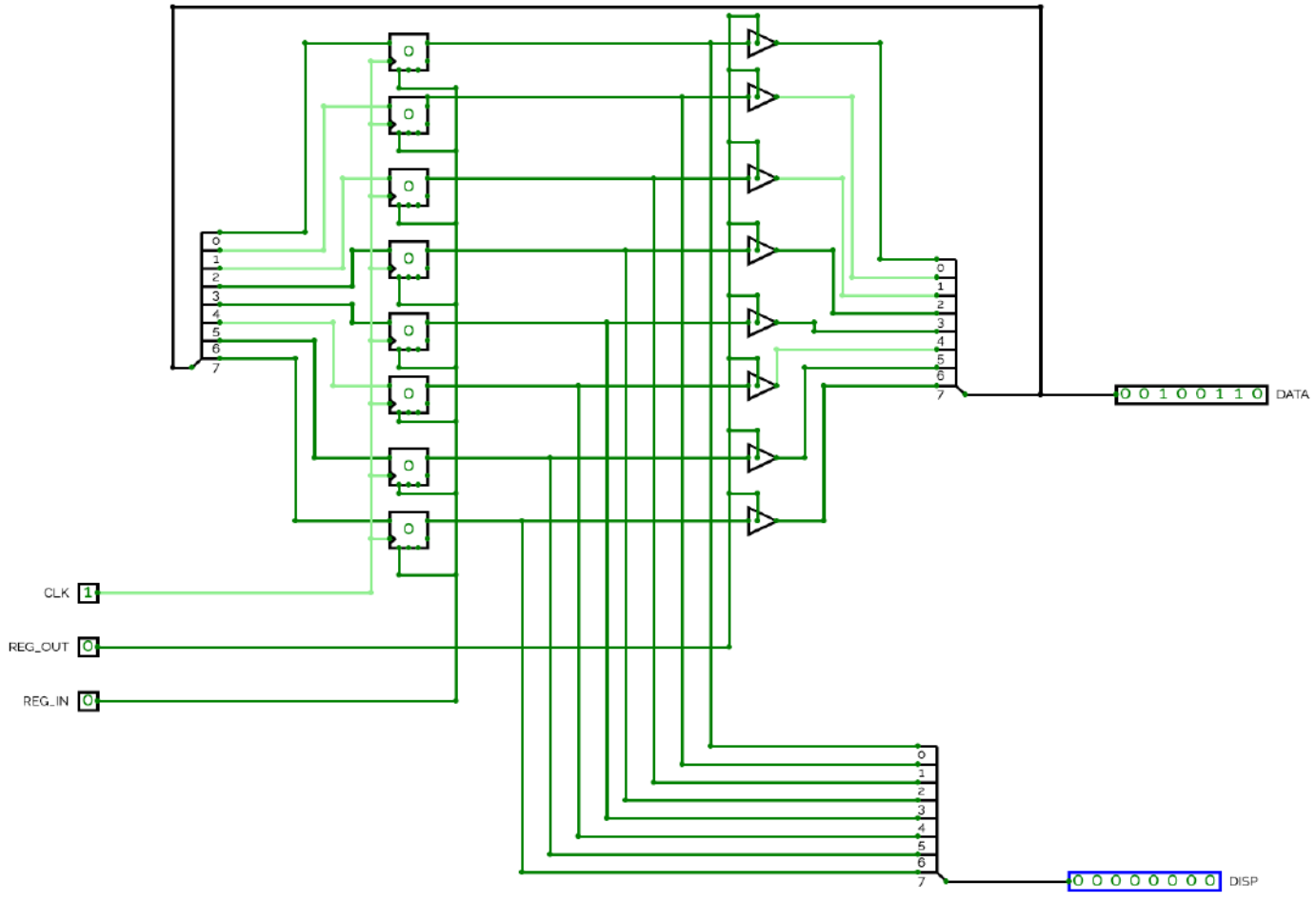
1. General CPU Register:

- **Design:**

  It is a Bi-Directional Register, Both its input and output are connected to the common bus. we use D-flip flops to store data. It has a DISP which outputs the value stored in the register. It has 2 control signals i.e. REG_IN and REG_OUT.

  1. When REG_IN is 1 it takes 8-bit input from the common bus and stores it in the flip-flop.
  2. When REG_OUT is 1 the 8-bit data stored in the flip-flop is passed into the common bus.

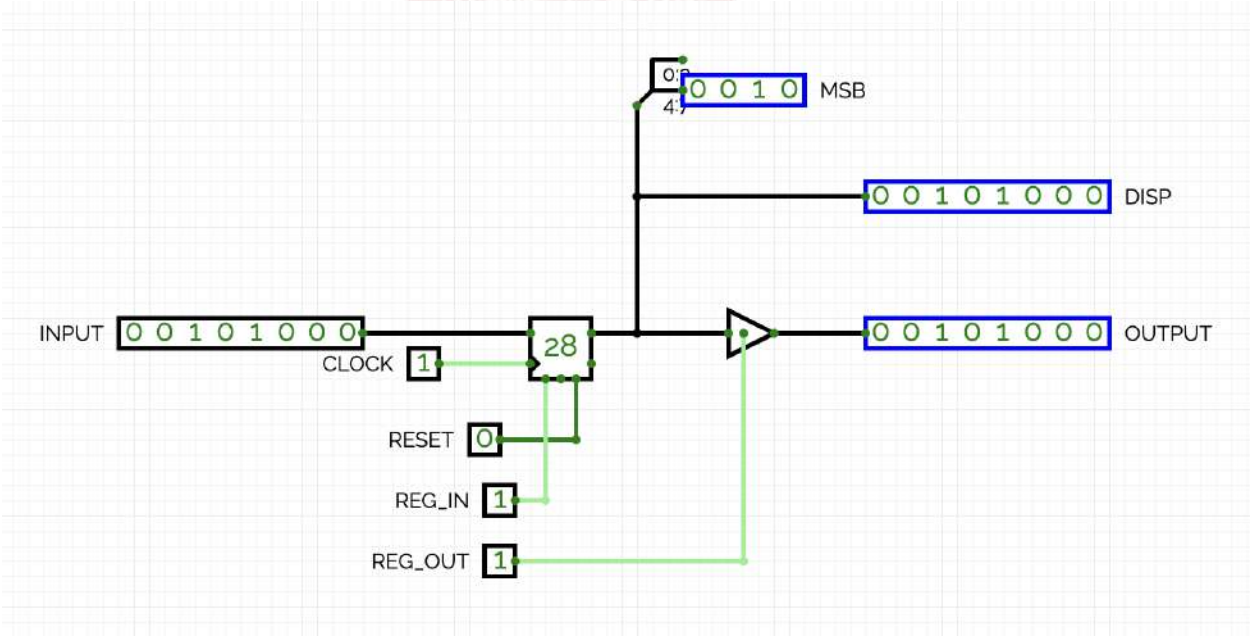
- **Circuit Diagram:**



- There are 3 General CPU Registers Accumulator (A), B, C.
- The accumulator is the register that is used to carry the sum or difference over multiple instructions
- REG_B is used to get input from memory. This is the only register that communicates with memory firsthand.
- REG_C is used as an output register and as well as a temporary register in swapping Accumulator and B.

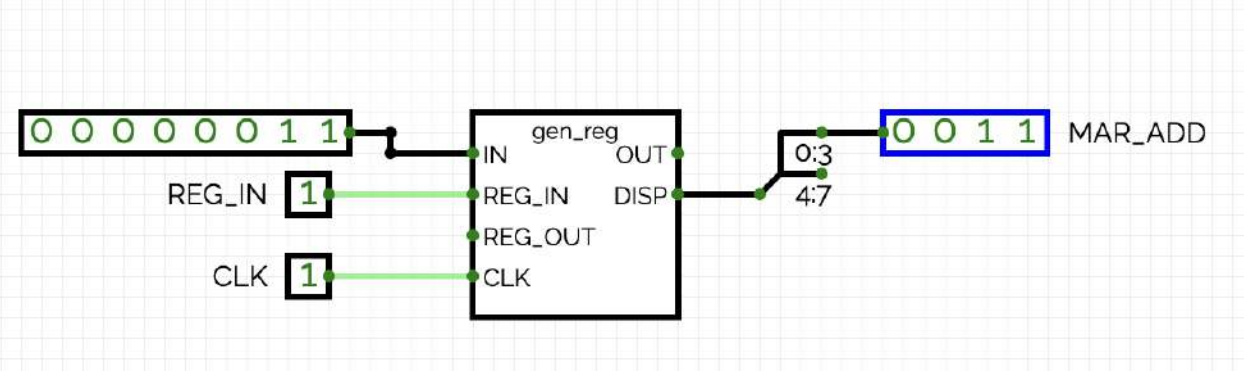
2. Instruction Register:

- **Design:**
  Instruction Register Design is similar to that of the General CPU Register except that IR is not bidirectional. We use s 8-bit flip-flop to store data. It has the same control words as that of a General CPU Register. It takes input from the common bus and sends the 4 MSBs to both the instruction decoder and the Controller. It can even send addresses for certain instructions to MAR if needed in certain instruction. It has also reset word which resets the value in flip flop to 0.
- **Circuit Diagram:**



3. Memory Address Register:

- **Design:**
  It is a 4-bit register that stores the address to be sent to the ROM in the processor. It is built using an 8-bit register. when REG_IN is active it takes 4 LSB's from 8-bit input and stores it in the register.MAR_ADD is always active and points to the correct address in ROM. The existence of this is very important because when the program counter sends the address to the ROM, it should simultaneously output it to the control bus. But since 2 components cannot talk to the common bus, we need a register to store the address and this is where MAR comes in.
- **Circuit Diagram:**

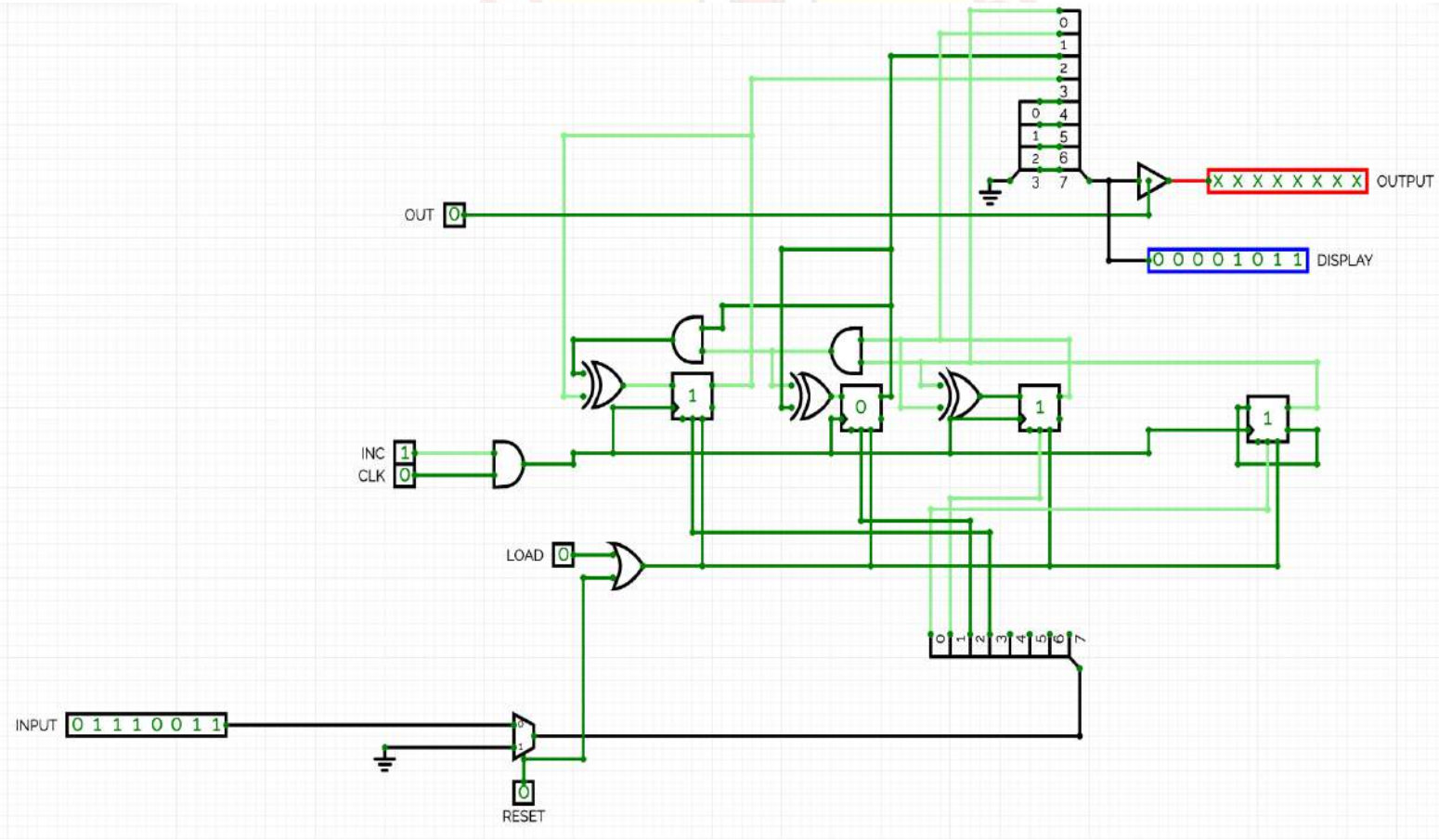


4. Program Counter:

- **Design:**

  The function of the program counter is to send the address of the instruction that is stored in the ROM.PC_LOAD is used to load the address from the common bus. It takes only the 4 LSBs from the 8-bit data. When OUT is active the 4-bit address stored in flip flops combined with 0000(Since 8-bit input should be sent to the common bus) is sent to the common bus line. When RESET is active the values in flip flops are reset to 0. When In changes from 0 to 1 the value in flip flops increases by 1. In Summary, the Program counter points to the next instruction after every instruction cycle.

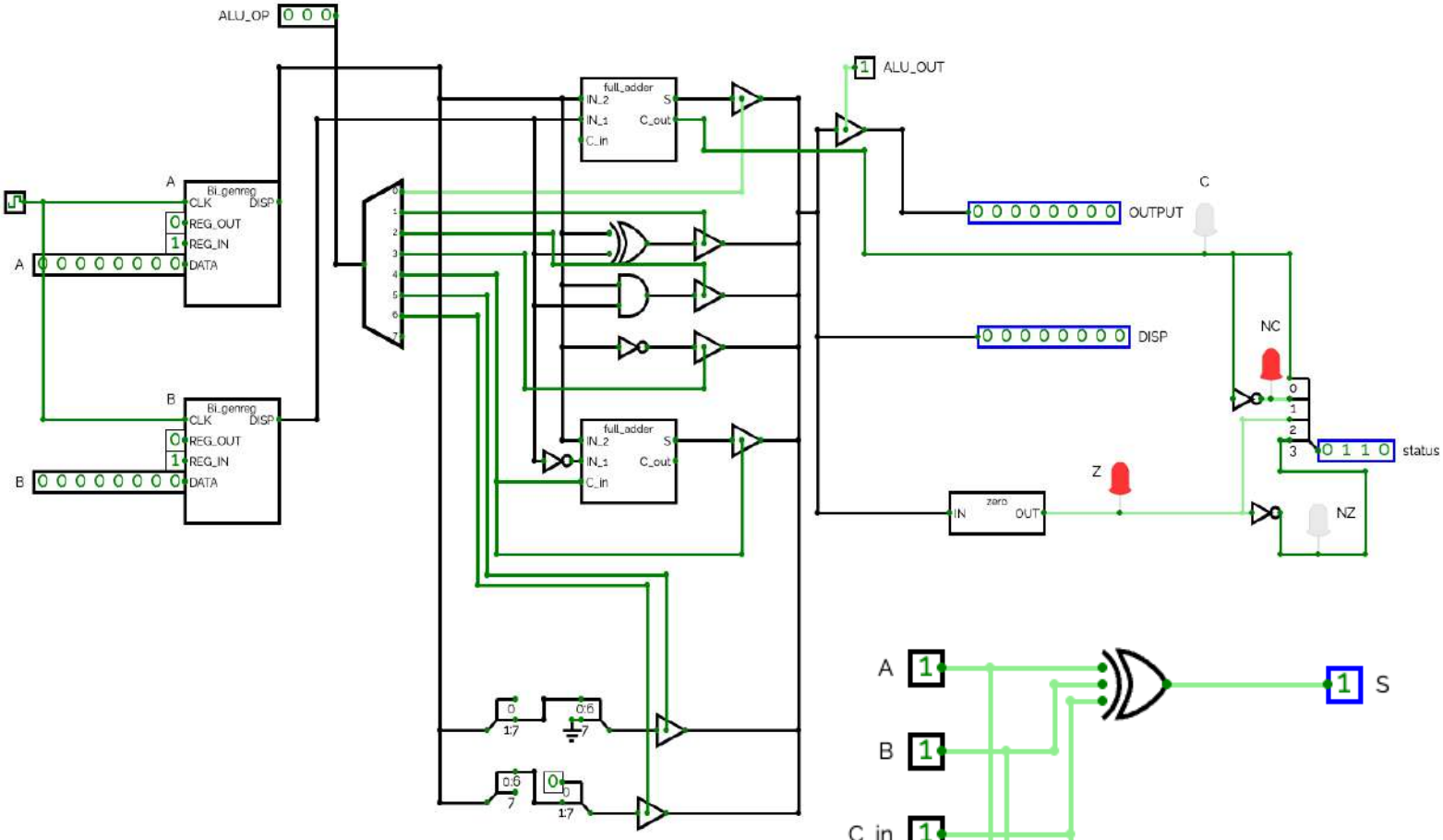
- **Circuit Diagram:**

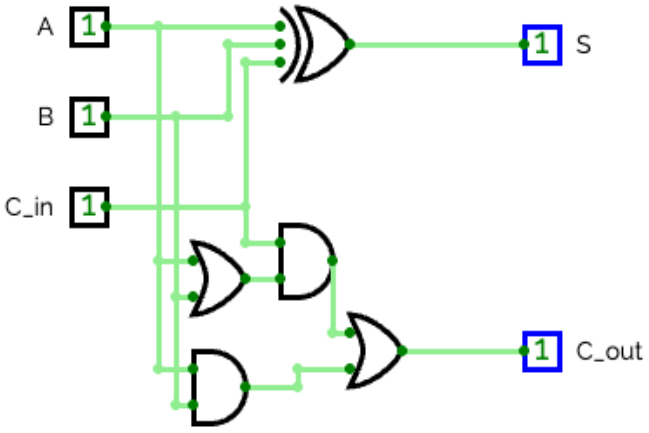


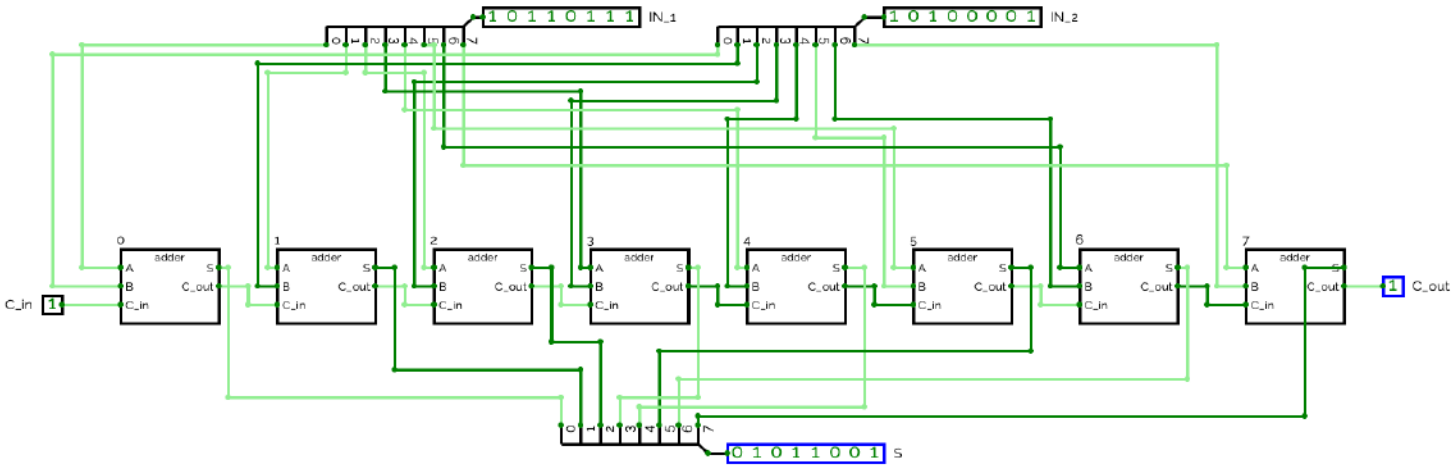
5. Arithmetic Logic Unit:

- **Design:**
  - ➢ ALU is one of the most important components in any processor. Here ALU can support 8 instructions. They are
    1. 000 - Addition is performed
    2. 001 - XOR is performed
    3. 010 - AND is performed
    4. 011 - NOT of A is performed
    5. 100 - SUB is performed
    6. 101 - Right shift of A
    7. 110 - Left shift of A
    8. 111 - Comparat
  - ➢ The instruction for this will be sent by the instruction decoder when an instruction that requires the operation of ALU is needed. For addition, decrements, and subtraction, we are using 8 1-bit full adders.
  - ➢ XOR, AND, NOT are performed by using the respective gates.
  - ➢ The right shift and left shift are performed by using a splitter.
  - ➢ When an operation is performed in ALU it sets 4 status registers namely C, NC, Z, NZ.
  - ➢ We are using C and Z flags to send them to the controller for doing conditional instructions.

- **Circuit Design:**

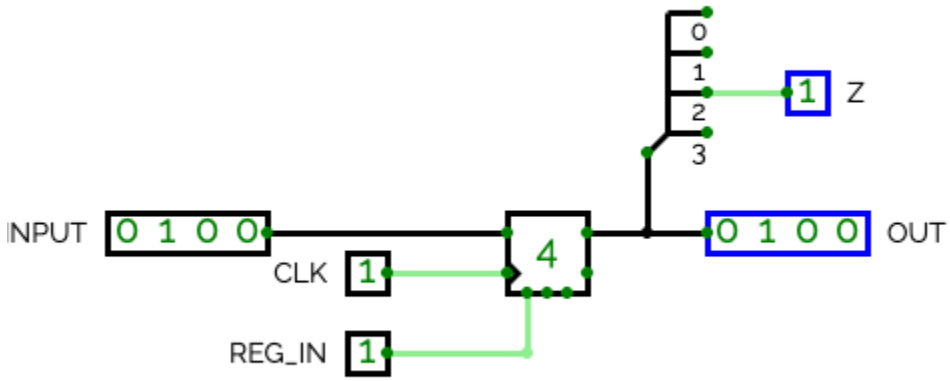






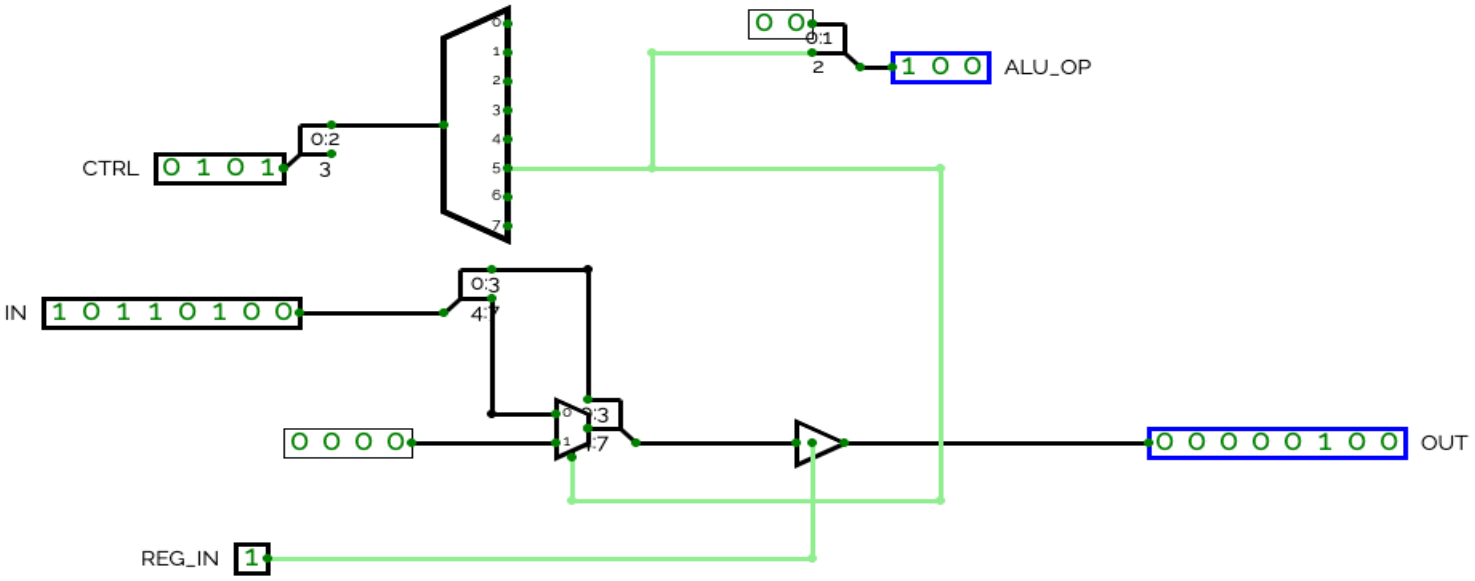
6. Status Register:

- **Design:**
  It is implemented using a 4-bit register. When REG_IN is 1 data is stored in the register. The data from ALU is stored in the status register and is used in the controller for doing conditional instructions like JNZ and JG. Status Register is essential in our processor design because the flags that it stores are useful for making conditional instructions, and the utility of these instructions is very high for supporting the multiplication of two numbers.
- **Circuit Diagram:**



7. Instruction Decoder:

- **Design:**
  Input is directly taken from reg IR without any tri-state buffer. The first 4 most significant bits decide what operation to be done. When IR OUT is 1, output is displayed. When Operation is LDI 4 Most significant bits of output become zeros'.
- **Circuit Diagram:**


0 0
0:1
2
1 0 0
ALU_OP
CTRL
0 1 0 1
0:2
3
IN
1 0 1 1 0 1 0 0
0:3
4:7
0 0 0 0
0:3
4:7
0 0 0 0 0 1 0 0
OUT
REG_IN
1

INSTRUCTION SET AND MACHINE CODE

Instruction	Mnemonic	Operands	Description	Operation
0000	NOP	—	No operation	—
0001	LDA	Ra, Ad	Load Accumulator	Ra ← ROM(Ad)
0010	STA	Ra, Ad	Store at Address	ROM(Ad) ← Ra
0011	ADD	Ra, Rb, Ad	Add without carry	Rb ← ROM(Ad) Ra ← Ra + Rb
0100	SUB	Ra, Rb, Ad	Subtract positive numbers	Rb ← ROM(Ad) Ra ← Ra - Rb
0101	LDI	Ra, K	Load Immediate	Ra ← k
0110	JMP	Ad	Jump to Address	PC ← ROM(Ad)
0111	SWAP	Ra, Rc	Swap values in registers	Rb ← Ra Ra ← Rc Rc ← Rb
1000	JNZ	Ad	Jump when the flag is non-zero	PC ← ROM(Ad)
1001	MOVAC	Ra, Rc	Move data from A to C	Rc ← Ra
1010	MOVCA	Ra, Rc	Move Data from C to A	Ra ← Rc
1011	OUT	Rc	Display on Hex Display	Rc ← Ra
1100	LSL	Ra	Logical shift left	Ra ← Ra << 1
1101	LSR	Ra	Logical shift right	Ra ← Ra >> 1
1110	CMP	Ra, Rb	Compare	
1111	HALT	—	Halt	—

DESCRIPTION OF INSTRUCTION SET

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

2. LDA - LOAD ACCUMULATOR:

- **Description:** This instruction loads the Accumulator with the contents from the memory by looking up at the 4-bit address provided.
- **Operation:**
    1. Ra ← ROM(XXXX)

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

- 8 -bit opcode

0001	XXXX

3. STA - STORE AT ADDRESS A:

- **Description:** This instruction takes the value at the accumulator and stores it in memory at address A.
- **Operation:**
    1. ROM ← Ra

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

- 8-bit opcode

0010	XXXX

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 the Accumulator.
- **Operation:**
    1. Rb ← ROM(XXXX)
    2. Ra ← Ra + Rb

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

- 8-bit opcode

0011	XXXX

5. SUB - SUBTRACTION OF POSITIVE NUMBERS:

- **Description:** This instruction loads 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:**
  1. Rb ← ROM
  2. Ra ← Ra−Rb

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

- 8-bit opcode

0100	XXXX

6. LDI - LOAD IMMEDIATE:

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

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

- 8-bit opcode

0101	XXXX

7. JMP - JUMP TO ADDRESS:

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

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

- 8-bit opcode

0110	XXXX

8. SWAP(AC) - SWAP A and C:

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

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

- 8-bit opcode

0111	XXXX

9. JNZ - Jump when Non-Zero:

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

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

- 8-bit opcode

1000	XXXX

10. MOVAC - Copy data from A to C:

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

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

- 8-bit opcode

1001	XXXX

11. MOVCA - Copy data from C to A:

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

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

- 8-bit opcode

1010	XXXX

12. OUT – Display on Hex display:

- **Description:** Copies the contents of Accumulator to Register-C and shows the value on the Hex display attached to it.
- **Operation:**
  1. Rc ← Ra

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

- 8-bit opcode

1011	XXXX

13. LSL – Logical Shift Left:

- **Description:** This instruction performs the logical left shift to the contents present in the Accumulator. Used to perform multiplication by 2.
- **Operation:**
    1. Ra ← Ra << 1

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

- 8-bit opcode

1100	XXXX

14. LSR – Logical Shift Right:

- **Description:** This instruction performs the logical right shift to the contents present in the Accumulator.Used to perform division by 2.
- **Operation:**
    1. Ra ← Ra >> 1

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

- 8-bit opcode

1101	XXXX

15. MOVAB - Copy Data from A to B:

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

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

- 8-bit opcode

1110	XXXX

16. HALT – Halt the program:

- **Description:** Fetch cycle stops.

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

- 8-bit opcode

1111	XXXX

ASSEMBLY PROGRAMS IMPLEMENTED USING INSTRUCTION SET

a. Adding two numbers and displaying result

Address	Assembly code	Machine code
0x0	LDA 0x5	0x15
0x1	ADD 0x6	0x36
0x2	OUT 0X0	0xb0
0x3	HALT 0X0	0xf0
0x4		0x05
0x5		0x18

- First, The content of ROM at the address 0x5 is loaded to Accumulator and the content at the address 0x6 is loaded to Register-B and then the ALU performs addition and stores it in A.
- Finally, the sum is loaded to Register-C and displayed on Hex Display. And the program halts.

b. Adding and subtracting four numbers in some combination

Address	Assembly code	Machine code
0x0	LDI 0X5	0x15
0x1	SUB 0X6	0x46
0x2	ADD 0X7	0x37
0x3	SUB 0X8	0x48
0x4	HALT 0X9	0xf0
0x5		0x34
0x6		0x12
0x7		0x21
0x8		0x15

- First, the value 0x34 is loaded directly to accumulator and then 0x12 is subtracted from it.
- Then the data at address 0x7 is added(0x21) and finally the data at 0x8 (0x15) is subtracted from the result.
- Finally the program Halts.

c. Multiplication of two numbers using repeated addition.

Address	Assembly code	Machine code
0x0	LDA 0x9	0x19
0x1	SWAP 0x0	0x70
0x2	LDI 0x0	0x50
0x3	ADD 0xa	0x3a
0x4	SWAP 0x0	0x70
0x5	SUB 0xb	0x4b
0x6	SWAP 0x0	0x70
0x7	JNZ 0x3	0x83
0x8	HLT 0x0	0xf0
0x9		0x11
0xa		0x05
0xb		0x01

- First, We load a data 0x11 into the Accumulator then we swap A and C.Then we store another value 0x05 in A.We decrease its value by 1 and swap again.Then we jump to address 0x3 and recursively add until one value becomes 0.
- If a value becomes 0 then zero flag becomes 1 and jump doesnt occur and finally the program Halts.

INDIAN INSTITUTE OF TECHNOLOGY MADRAS
सिद्धिर्भवति कर्मजा

MICRO-INSTRUCTIONS AND CONTROLLER DESIGN

1. NOP:

T0 : 1 ≪ PC_OUT | 1 ≪ MAR_IN
T1 : 1≪PC_INC | 1≪MEM_OUT | 1≪IR_IN
T2 : 0
T3 : 0
T4 : 0

2. LDA:

T0 : 1 ≪ PC_OUT | 1 ≪ MAR_IN
T1 : 1≪PC_INC | 1≪MEM_OUT | 1≪IR_IN
T2 : IR_OUT | 1 ≪ MAR_IN
T3 : MEM_OUT | 1 ≪ REGA_IN
T4 : 0

3. STA:

T0 : 1 ≪ PC_OUT | 1 ≪ MAR_IN
T1 : 1≪PC_INC | 1≪MEM_OUT | 1≪IR_IN
T2 : 1 ≪ IR_OUT | 1 ≪ MAR_IN
T3 : 1 ≪ MEM_IN | 1 ≪ REGA_OUT
T4 : 0

4. ADD:

T0 : 1 ≪ PC_OUT | 1 ≪ MAR_IN
T1 : 1≪PC_INC | 1≪MEM_OUT | 1≪IR_IN
T2 : 1 ≪ IR_OUT | 1 ≪ MAR_IN

T3 : 1 ≪ MEM_OUT | 1 ≪ REGB_IN
T4 : 1 ≪ ALU_OUT | 1 ≪ REGA _IN

5. SUB:

T0 : 1 ≪ PC_OUT | 1 ≪ MAR_IN
T1 : 1≪PC_INC | 1≪MEM_OUT | 1≪IR_IN
T2 : 1≪ IR_OUT | 1 ≪ MAR_IN
T3 : 1 ≪ MEM_OUT | 1 ≪ REGB_IN
T4 : 1 ≪ ALU_OUT | 1 ≪ REGA_IN

6. LDI:

T0 : 1 ≪ PC_OUT | 1 ≪ MAR_IN
T1 : 1≪PC_INC | 1≪MEM_OUT | 1≪IR_IN
T2 : 1 ≪ IR_OUT | 1 ≪ REGA_IN
T3 : 0
T4 : 0

7. JMP:

T0 : 1 ≪ PC_OUT | 1 ≪ MAR_IN
T1 : 1≪PC_INC | 1≪MEM_OUT | 1≪IR_IN
T2 : 1 ≪ IR_OUT | 1 ≪ PC_LOAD
T3 : 0
T4 : 0

8. SWAP:

T0 : 1 ≪ PC_OUT | 1 ≪ MAR_IN
T1 : 1≪PC_INC | 1≪MEM_OUT | 1≪IR_IN
T2 : 1 ≪ REGA_OUT | 1 ≪ REGB_IN
T3 : 1 ≪ REGC_OUT | 1 ≪ REGA_IN

T4 : 1 ≪ REGB_OUT | 1 ≪ REGC_IN

9. JNZ:

If flag =0:

T0 : 1 ≪ PC_OUT | 1 ≪ MAR_IN
T1 : 1≪PC_INC | 1≪MEM_OUT | 1≪IR_IN
T2 : 1 ≪ IR_OUT | 1 ≪ PC_LOAD
T3 : 0
T4 : 0

Else

T0 : 1 ≪ PC_OUT | 1 ≪ MAR_IN
T1 : 1≪PC_INC | 1≪MEM_OUT | 1≪IR_IN
T2 : 0
T3 : 0
T4 : 0

10. MOVAC:

T0 : 1 ≪ PC_OUT | 1 ≪ MAR_IN
T1 : 1≪PC_INC | 1≪MEM_OUT | 1≪IR_IN
T2 : 1 ≪ REGA_OUT | 1 ≪ REGC_IN
T3 : 0
T4 : 0

11. MOVCA:

T0 : 1 ≪ PC_OUT | 1 ≪ MAR_IN
T1 : 1≪PC_INC | 1≪MEM_OUT | 1≪IR_IN
T2 : 1 ≪ REGC_OUT | 1 ≪ REGA_IN
T3 : 0
T4 : 0

12. OUT:

T0 : 1 ≪ PC_OUT | 1 ≪ MAR_IN
T1 : 1≪PC_INC | 1≪MEM_OUT | 1≪IR_IN
T2 : 1 << REGA_OUT | 1 << REGC_IN
T3 : 1 << REGC_OUT
T4 : 0

13. LSL:

T0 : 1 ≪ PC_OUT | 1 ≪ MAR_IN
T1 : 1≪PC_INC | 1≪MEM_OUT | 1≪IR_IN
T2 : 1 << ALU_OUT | 1 << REGA_IN
T3 : 0
T4 : 0

14. LSR:

T0 : 1 ≪ PC_OUT | 1 ≪ MAR_IN
T1 : 1≪PC_INC | 1≪MEM_OUT | 1≪IR_IN
T2 : 1 << ALU_OUT | 1 << REGA_IN
T3 : 0
T4 : 0

15. MOVAB:

T0 : 1 ≪ PC_OUT | 1 ≪ MAR_IN
T1 : 1≪PC_INC | 1≪MEM_OUT | 1≪IR_IN
T2 : 1 ≪ REGA_OUT | 1 ≪ REGB_IN

T3 : 0
T4 : 0

16. HLT:

T0 : 1 ≪ MAR_IN
T1 : 1≪MEM_OUT | 1≪IR_IN
T2 : 0
T3 : 0
T4 : 0

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.