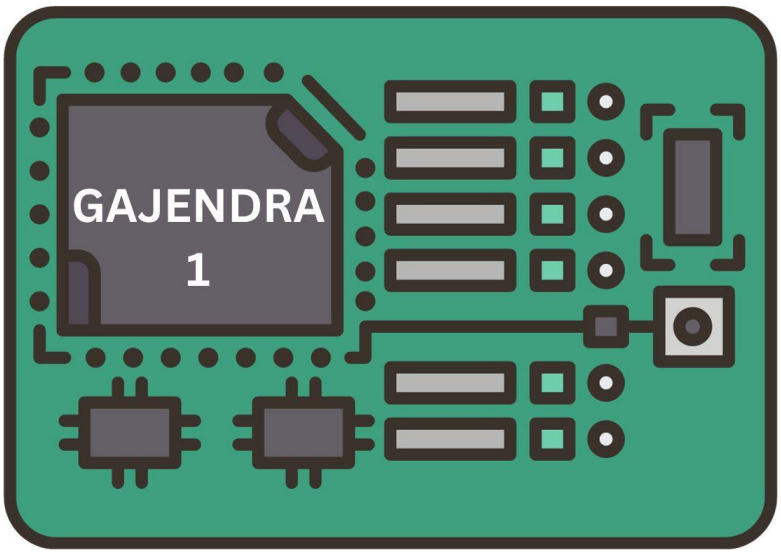
CS2310: Final Assignment

Computer System Design Lab: Gajendra-I

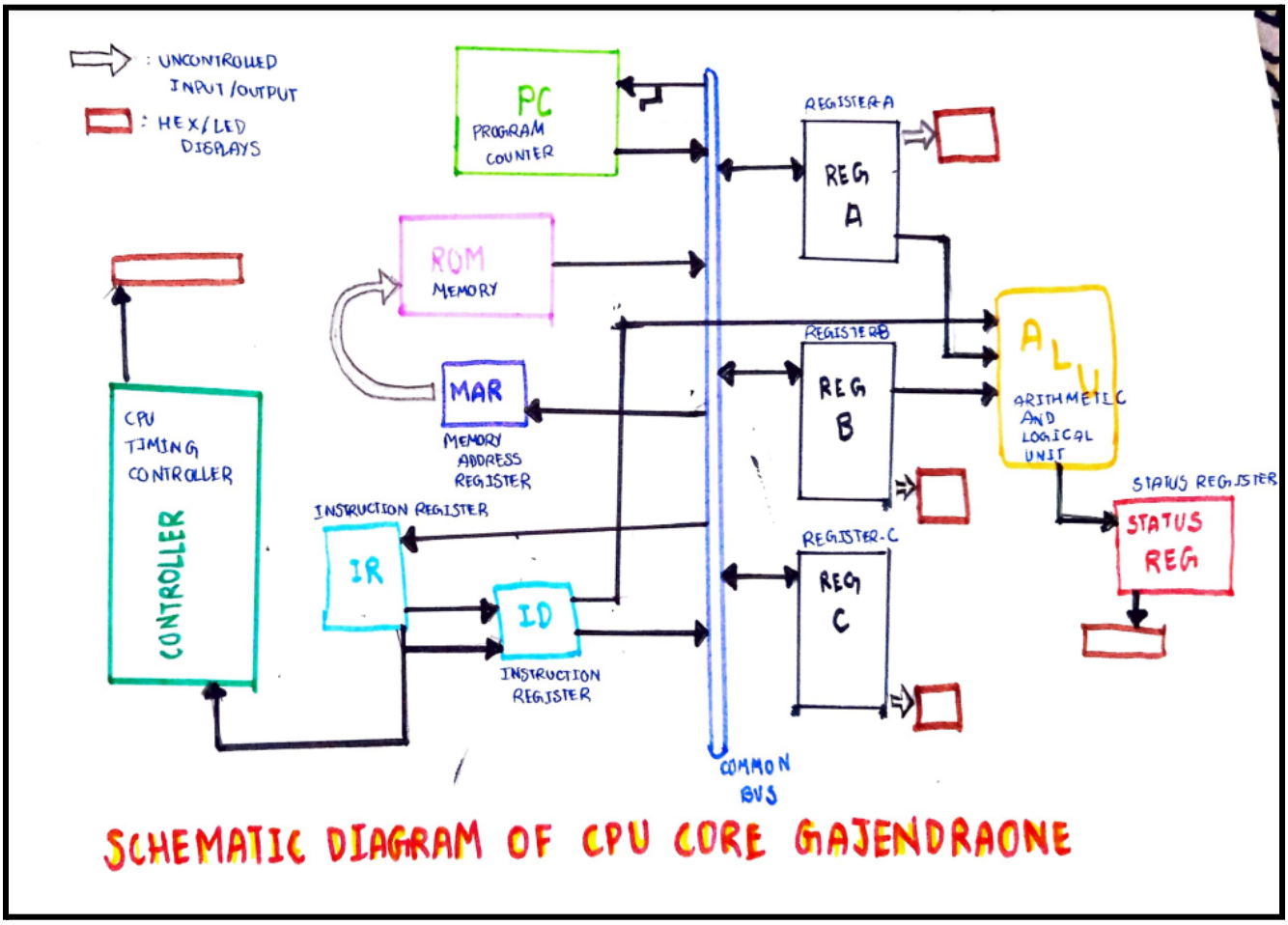


RIDDHI AGARWAL
(CS22B081)

VAISHNAVI RAJESH
(CS22B023)

SECTION-1

The schematic for our processor is as follows:



Define all the internal components and state their design along with a circuit diagram.

- **General CPU Registers (reg_cpu_8):**

  Definition:

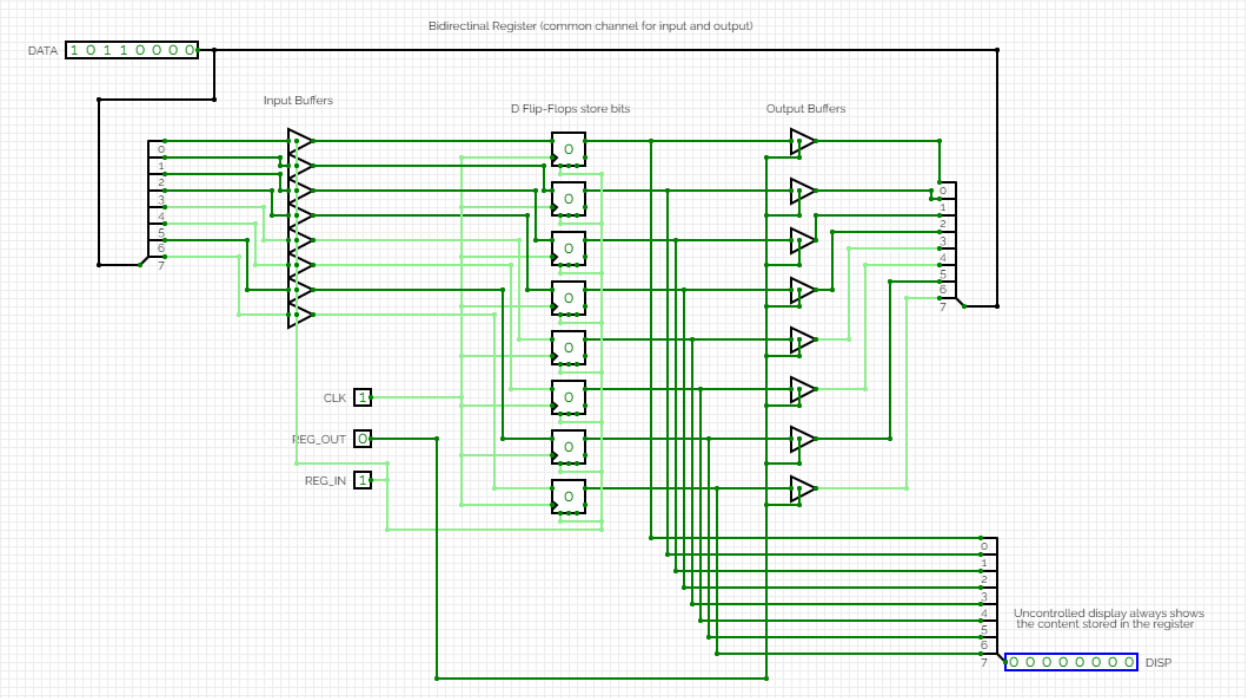
  A register serves as a quick memory for accepting, storing, and sending data that the CPU will need. A register is a collection of flip-flops.Single bit digital data is stored using flip-flops. By combining many flip-flops, the storage capacity can be extended to accommodate a huge number of bits. Here, the General purpose registers are A, B and C with a capacity of 8 bits each.We have used 3 registers, as this can enable us to swap the value between 2 registers which can be used for some software purposes.

  Connections:

  All the general purpose registers, A, B and C take and receive data from common bus (Their data is connected to bus). Additionally, we are using seven segment led hex display to see the values stored in the registers at any point of time. The REG_IN and REG_OUT of the registers are connected to their respective control bits in the controller. The values stored in register A and B are taken as inputs to the ALU. All registers are connected to the common system clock.

  Design:

  We have a bidirectional register (i.e. it has a common channel for input and output controlled by buffers internally) .The 8-bit general registers are implemented using D-flip flops. Each bit gets latched to a D-flip flop. There are tri-state buffers after data (input) is taken. These, along with the enable of registers are controlled by REG_IN bit. Only when REG_IN of A is enabled, A can store the input data. The display is uncontrolled. Similar to input, there are tri-state buffers which control the output of data from the register. Input and output channels for data is the same and this channel is connected to the common bus.



- **Instruction Register (reg_IR):**

  Definition:

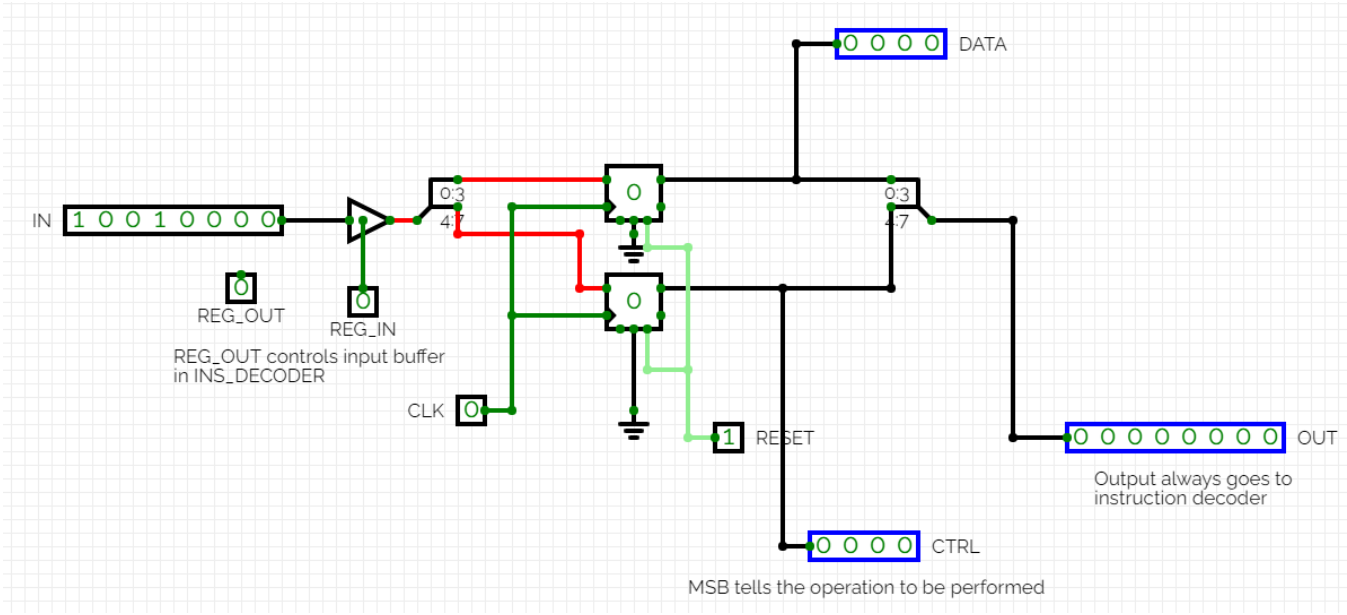
  The instruction register (IR) holds the instruction currently being executed or decoded. Each instruction to be executed is loaded into the instruction register, which holds it while it is decoded and ultimately executed, which can take several steps. In this case, each instruction takes 5 clock cycles(T-states) to be executed.

  Connections:

  The REG_IN and REG_OUT of instruction register are connected to the respective control bits of the controller. It takes input from common bus and outputs to the instruction decoder. It can hold an 8-bit instruction, out of which the first 4 bits (MSB) are input to the controller via ‘CTRL’.

  Design:

  The input of 8-bit data is controlled by a tri-state buffer. The data is then split into 2 parts 4-4. The 4 MSB bits of the input represent the ctrl (Ex: corresponding to LDA, ADD, SUB, etc) and the other 4 bits represent the data directly or memory address.



- **Memory Address Register (reg_MAR) - 4-bit addresses:**

  Definition:

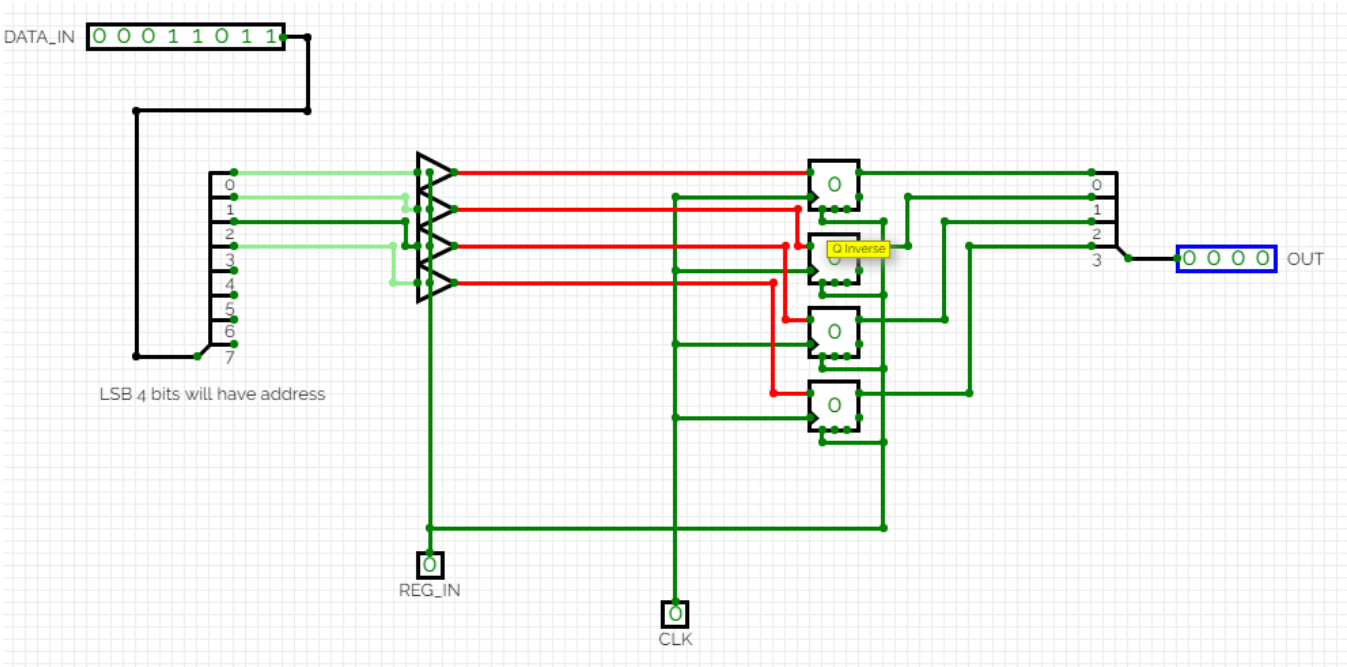
  MAR is a register that stores the memory address from which data will be fetched from the ROM. This register is used to access data and instructions from memory during the execution phase of instruction. MAR holds the memory location of data that needs to be accessed.

  Connections:

  The MAR's uncontrolled output goes to the ROM. The DATA_IN of MAR is connected to the common bus to take in an address. REG_IN bit is connected to its corresponding bit in controller. REG_IN controls the input buffer of MAR.

  Design:

  The MAR takes an 8-bit value from the common bus and ignores the most significant 4 bits, as address given to rom should have 4 bits (last 4 bits of instruction may have address). These 4 bits are then connected to D-flip flops via tri-state buffers to control the flow. The outputs of D-flip flops are then given to the ROM (4 bit output).



- **Program Counter (counter_PC) - 4-bit addresses:**

  Definition:

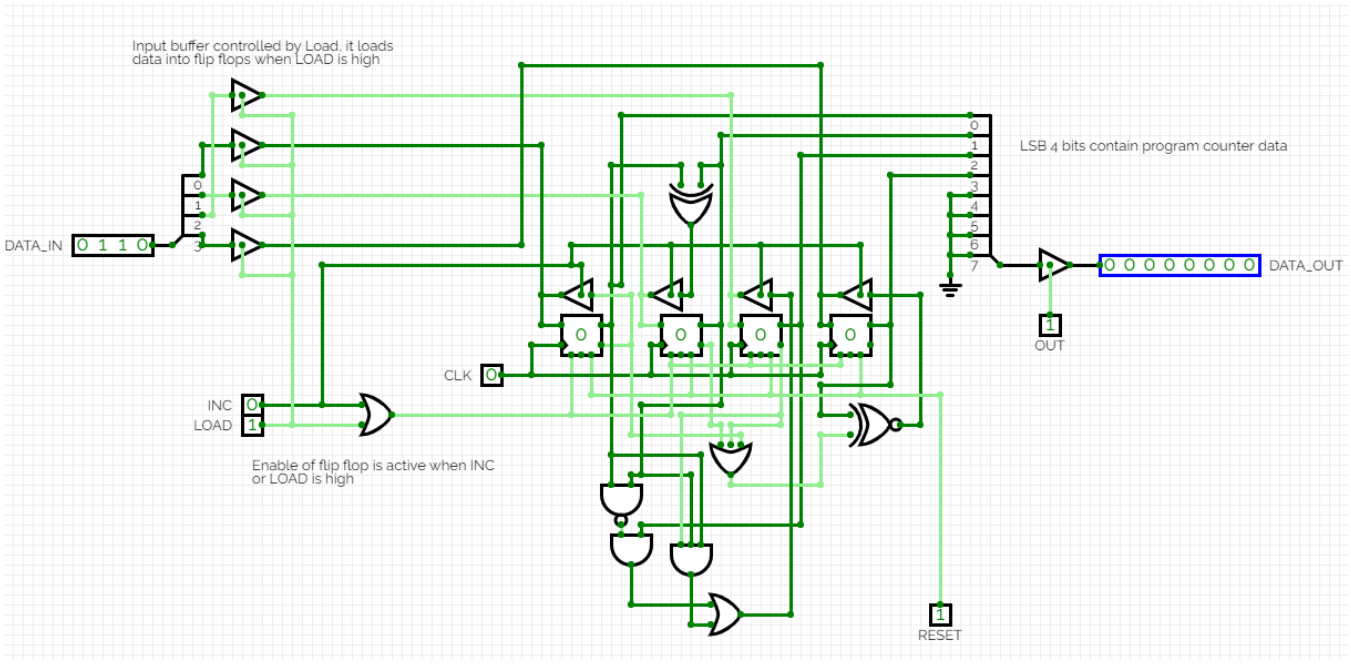
  A program counter contains the address (location) of the instruction being executed or the address of the next instruction to be executed. As each instruction is fetched, the program counter increases its stored value by 1. After each instruction is fetched, the program counter points to the next instruction in the sequence.When restart is enabled, the program counter reverts to 0.

  Connections:

  The INC,OUT and LOAD of program counter are connected to their respective control bits on controller. Both DATA_IN and DATA_OUT are connected to the common bus, but since DATA_IN is 4 bits, the LSB 4 bits of common bus is taken (which has the address).
  The program counter is connected to common clock and reset, which resets it to zero.

  Design:

  The program counter is a counter circuit implemented using D-flip flops. The program counter can either increase the value (address) by one or have a new address which is loaded. The feedback of the D-flip flops, which are responsible for increasing the value by 1, have a buffer controlled by INC. There is a 4-bit data, DATA_IN which is connected to the inputs of D-flip flops. These are controlled by tri-state buffers whose enable is connected to LOAD. The enable of D-flip flops are connected to the OR of LOAD and INC. So, either LOAD or INC is high and the value gets increased or loaded accordingly.



- **Arithmetic and Logical Unit (ALU) - supports ADD/SUB/CMP**

  Definition:

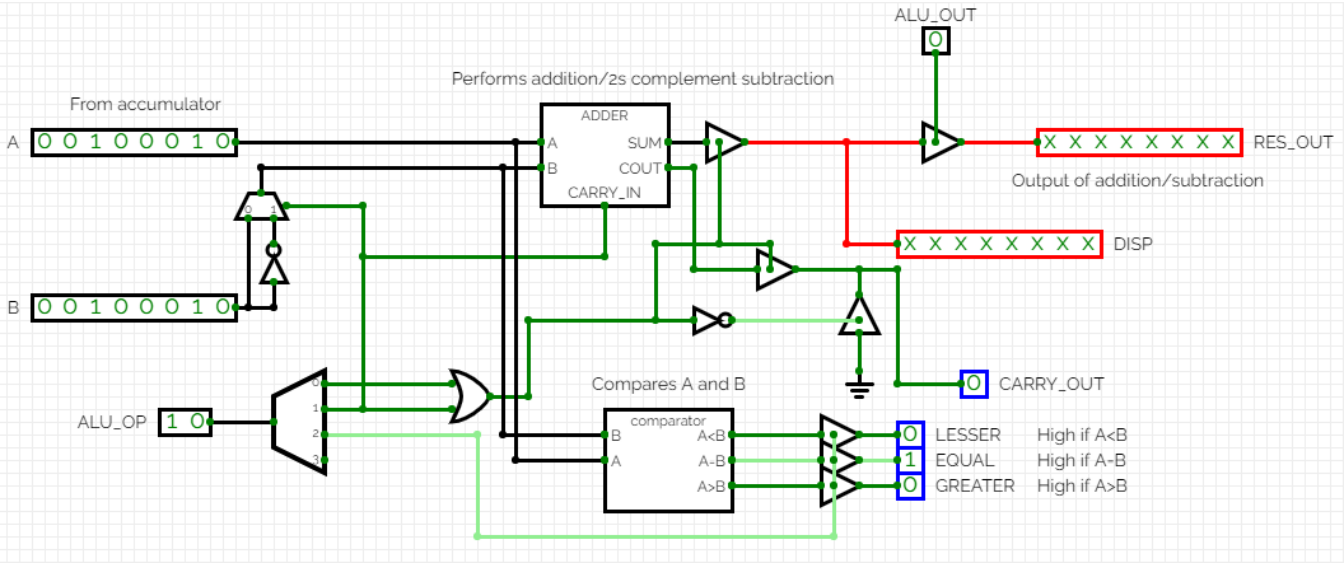
  ALU carries out arithmetic and logic operations taking instruction (op code) from instruction decoder. Here, ALU performs these operations on the data contained in register A and B.

  Connections:

  ALU takes uncontrolled inputs from registers A and B. The ALU_OP is connected to the instruction decoder. The ALU_OUT is connected to its respective control bit on controller. The ALU then sends the output, carry_out, lesser, equal, greater outputs to the status register. The output of ALU is sent to common bus. The ALU is connected to the common clock.

  Design:

  ALU_OP contains a 2-bit op code and is connected to a decoder. Then accordingly, the operation is performed. 00 for addition, 01 for subtraction, 11 for comparison. There is a multiplexer to decide whether to send B or its complement to adder for add and subtract respectively.



- **Status Register (reg_status_4)- Z, NZ, C, NC**

  Definition:

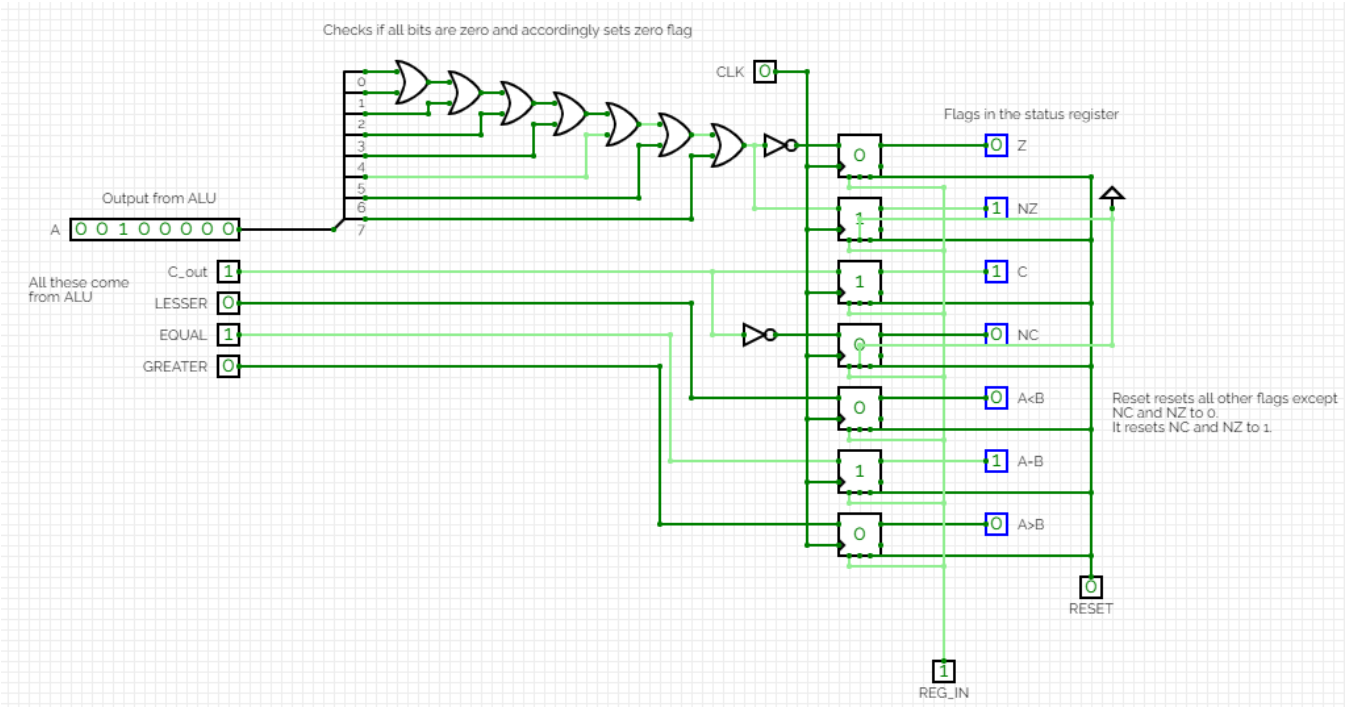
  It is a register that stores and displays zero, not_zero, carry and not_carry flags of ALU output (If output is zero, zero flag is enabled and if carry is zero, carry flag is enabled).

  Connections:

  The REG_IN of status register is controlled by the same bit which controls ALU_OUT. It receives inputs - A, CARRY_OUT, LESSER, EQUAL, GREATER from the ALU.
  It then outputs 6 flags- zero, not_zero, not_carry, lesser, equal and greater.

  Design:

  From the 8-bit input - A, zero flag is calculated by taking OR of all bits connected to a D-flip flop. Non-zero flag is the complement of the OR sent to a D-flip flop. Then, two other D-flip flops take input as CARRY_OUT and its complement for C and NC flags respectively.
  D-flip flops take inputs from LESSER, GREATER and EQUAL for their corresponding flags.
  REG_IN enables the D-flip flops. Reset resets greater, lesser, equal, carry, zero as zero.



SECTION-2

Answer:
The instruction is in the form of 8-bit opcodes consisting of (MSB) 4-bits for specifying the operation and (LSB) 4-bits for specifying address or immediate data.

- **NOP**
  Description: Does nothing
  Operation: -

Syntax	Operands	Program counter
NOP -	-	PC ← PC+1

Registers impacted: None
8-bit op code (xxxx=dont care condition)

0000 (0x0)	xxxx

- **LDA**
  Description: Loads the value present in given memory address into register A.
  Operation: reg_A ← *(address)

Syntax	Operands	Program counter
LDA address	0<=address<=15(0xF)	PC ← PC+1

Registers impacted: reg_A
8-bit op code (kkkk=address)

0001 (0x1)	kkkk

- **STA**
  Description: Loads the value present in register A into given memory address
  Operation: *(address) ← reg_A

Syntax	Operands	Program counter
STA address	0<=address<=15(0xF)	PC ← PC+1

Registers impacted: none
8-bit op code (kkkk=address)

0010 (0x2)	kkkk

- **LDI**
  Description: Loads the given value into register A.
  Operation: reg_A ← value

Syntax	Operands	Program counter
LDI value	0<=value<=15(0xF)	PC ← PC+1

Registers impacted: reg_A
8-bit op code (kkkk=value)

0011 (0x3)	kkkk

- **ADD**
  Description: Loads the value from given address into register B. Adds the value present register B to the value present in register A and stores the sum in register A.
  Operation: reg_B ← *(address)
  reg_A ← reg_A+reg_B.

Syntax	Operands	Program counter
ADD address	0<=address<=15(0xF)	PC ← PC+1

Registers impacted: reg_A, reg_B, sets flags in the status register
8-bit op code (kkkk=address)

0100 (0x4)	kkkk

- **SUB**
  Description: Loads the value from given address into register B. Subtracts the value present in register B from the value present in register A and stores the difference in register A.
  Operation: reg_B ← *(address)
  reg_A ← reg_A - reg_B.

Syntax	Operands	Program counter
SUB address	0<=address<=15 (0xF)	PC ← PC+1

Registers impacted: reg_A, reg_B, sets flags in the status register
8-bit op code (kkkk=address)

0101 (0x5)	kkkk

- **CMP**
  Description: Loads the value from given address into register B.
  Compares the values present in register A and register B.Sets flags in the status register. It sets the greater flag in the status register if the value in register A is greater than the value in register B .
  Operation: reg_B ← *(address)
  reg_A ← reg_A+reg_B

Syntax	Operands	Program counter
CMP address	0<=address<=15 (0xF)	PC ← PC+1

Registers impacted: reg_B, sets flags in the status register
8-bit op code (kkkk=address)

0110 (0x6)	kkkk

- **SWAP**
  Description: Swaps the values present in register A and register C (using register B as temporary register for swapping).
  Operation: reg_B ← reg_A
  reg_A ← reg_C
  reg_C ← reg_B

Syntax	Operands	Program counter
SWAP -	-	PC ← PC+1

Registers impacted: reg_A, reg_B, reg_C
8-bit op code (xxxx=dont care condition)

0111 (0x7)	xxxx

- **JMP**
  Description: Loads the given address into the program counter.
  Operation: PC ←address

Syntax	Operands	Program counter
JMP address	0<=address<=15 (0xF)	PC ← address

Registers impacted: - None
8-bit op code (kkkk=address)

1000 (0x8)	kkkk

- **JIG (extra)**
  Description: Loads the given address into the program counter if the greater than flag is set in the status register.
  Operation: if (greater==1) PC ← address

Syntax	Operands	Program counter
JIG address	0<=address<=15 (0xF)	if (greater==1) PC ← address

Registers impacted: - None
8-bit op code (kkkk=address)

1001 (0x9)	kkkk

- **JNZ**
  Description: Loads the given address into the program counter if the non-zero flag is set in the status register.
  Operation: if (NZ==1) PC ← address

Syntax	Operands	Program counter
JNZ address	0<=address<=15 (0xF)	if (NZ==1) PC ← address

Registers impacted: - None
8-bit op code (kkkk=address)

1010 (0xA)	kkkk

- **MOVBA**
  Description: Moves the contents of register B to register A.
  Operation: reg_A ← reg_B

Syntax	Operands	Program counter
MOVBA -	-	PC ← PC+1

Registers impacted: reg_A, reg_B
8-bit op code (xxxx=dont care condition)

1011 (0xB)	xxxx

- **MOVAB**
  Description: Moves the contents of register A to register B.
  Operation: reg_B ← reg_A

Syntax	Operands	Program counter
MOVAB -	-	PC ← PC+1

Registers impacted: reg_A, reg_B
8-bit op code (xxxx=dont care condition)

1100 (0xC)	xxxx

- **MOVAC (OUT)**
  Description: Moves the contents of register A to register C. Register C's content is displayed.
  Operation: reg_C ← reg_A

Syntax	Operands	Program counter
MOVAC -	-	PC ← PC+1

Registers impacted: reg_A, reg_C
8-bit op code (xxxx=dont care condition)

1101 (0xD)	xxxx

- **MOVCB**
  Description: Moves the contents of register C to register B.
  Operation: reg_B ← reg_C

Syntax	Operands	Program counter
MOVCB -	-	PC ← PC+1

Registers impacted:reg_B,reg_C
8-bit op code (xxxx=dont care condition)

1110 (0xE)	xxxx

- **HLT**
  Description: It freezes the program counter
  Operation: -

Syntax	Operands	Program counter
HLT -	-	-

Registers impacted: -
8-bit op code (xxxx=dont care condition)

1111 (0xF)	xxxx

SECTION-3

Some example programs

Please refer to this drive folder for the videos of running the the following programs on the CPU:
https://drive.google.com/drive/folders/1P-W1Kp2CI3zfaoVvylKzmccq5cjzqzKJ?usp=sharing

- **Add two numbers:**

0. LDA address
1. ADD address
2. MOVAC
3. HALT

Example: 1A 4B 90 FF
(refer to drive for video of running the program)

- **Subtract two numbers:**

0. LDA address
1. SUB address
2. MOVAC
3. HALT

Example: 1A 5B D0 FF
(refer to drive for video of running the program)

- **A multiplication routine using repeated addition:**

0. LDA address
1. MOVAC
2. LDI 0
3. ADD address
4. SWAP
5. SUB address(1)
6. SWAP
7. JNZ 3
8. HALT

Example: 1A D0 30 4B 70 5C 70 A3 FF
Here, memory location C has 1, A and B have the two numbers which need to be multiplied.
(refer to drive for video of running the program)

- **Alternatively add and subtract 4 numbers at memory locations A,B,C,D: numbers at A-B+C-D**

0. LDA address
1. SUB address
2. ADD address
3. SUB address
4. MOVAC(OUT)
5. HALT

Example: 1A 5B 4C 5D 90 FF

- **Integer Floor Division**

0. LDA address
1. MOVAC
2. LDI 0
3. SWAP
4. ADD address
5. CMP address
6. JIG 10
7. SWAP
8. ADD address
9. JMP 3
10. SWAP
11. ADD address
12. HALT

Example: 1D D0 30 70 4D 6E 9A 70 4F 83 70 4F FF

Here, memory address F has 1, D has the divisor and E has the dividend such that (value at E> value at D).This calculates floor(E/D).

The final value in REG_A after the program is the quotient of division.

(refer to drive for video of running the program)

- **To find greatest number among 3 numbers**
  0. LDA D
  1. MOVAC
  2. CMP E
  3. JIG (5)
  4. LDA E
  5. MOV AC
  6. CMP F
  7. JIG(9)
  8. LDA F
  9. MOV AC
  10. HALT

Example: 1D D0 6E 95 1E D0 6F 99 1F D0 FF
Here memory addresses D, E and F have the 3 numbers among which we need to find the largest number.
(refer to drive for video of running the program)

- **Program to demonstrate STA**
  0. LDI 5
  1. STA B
  2. LDI 6
  3. LDA B
  4. HALT

Example: 35 2B 36 1B FF
Here, we are loading register A with 5 and then writing this number '5' to the memory address B of EEPROM. Then we are loading '6' into register 'A'. To demonstrate STA has taken place, we load register A with the contents of memory address B. Finally, register A should have 5.

SECTION-4

The control word contains 16 bits and they are:

MSBs of control word ……LSBs of control word

HLT	PC	PC	PC	MEM	MEM	IR	IR	MAR	RA	RA	RB	RB	RC	RC	ALU
-	INC	O	L	IN	O	IN	O	IN	IN	O	IN	O	IN	O	O

T0 and T1, the first 2 T-states correspond to the fetch cycle.

14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	← Bits
PC	PC	PC	MEM	MEM	IR	IR	MAR	RA	RA	RB	RB	RC	RC	ALU	Control word
INC	O	L	IN	O	IN	O	IN	IN	O	IN	O	IN	O	O	
0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0x2080
1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0x4600

In this, the out of program counter is high and the in of memory address register. The address shown by the program counter is now in memory address register (MAR). This address goes to memory (ROM) since there is uncontrolled output to ROM from MAR. In the next step, the program counter increases ir=ts value by 1, the out of memory is high (it outputs the value stored in the address it got from MAR, to the common bus) and the in of instruction register is high (so that it can store this value from common bus).

For each of the operations, given below are the **states of the control word which are high** (1) for the first 5 T-states (T0, T1, T2, T3, T4, T5).

```
NOP:Minimum T-states required: 2
1 << PC_out | 1 << MAR_in |                    :T0
1 << PC_inc | 1 << MEM_out | 1 << IR_in |      :T1
0                                              :T2
0                                              :T3
0                                              :T4

LDA:Minimum T-states required: 4
```

```
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 << REGA_in |
0

STA: Minimum T-states required: 4
1 << PC_out | 1 << MAR_in |
1 << PC_inc | 1 << MEM_out | 1 << IR_in |
1 << IR_out | 1 << MAR_in |
1 << MEM_in | 1 << REGA_out |
0

LDI: Minimum T-states required: 3
1 << PC_out | 1 << MAR_in |
1 << PC_inc | 1 << MEM_out | 1 << IR_in |
1 << IR_out | 1 << REGA_in |
0
0

ADD: Minimum T-states required: 5
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 << REGA_in | 1 << ALU_out |

SUB: Minimum T-states required: 5
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 << REGA_in | 1 << ALU_out |

CMP: Minimum T-states required: 5
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 |
```

```
SWAP: Minimum T-states required: 5
1 << PC_out | 1 << MAR_in |
1 << PC_inc | 1 << MEM_out | 1 << IR_in |
1 << REGA_out | 1 << REGB_in |
1 << REGA_in | 1 << REGC_out |
1 << REGB_out | 1 << REGC_in |

JMP: Minimum T-states required: 3
1 << PC_out | 1 << MAR_in |
1 << PC_inc | 1 << MEM_out | 1 << IR_in |
1 << PC_load | 1 << IR_out |
0
0

JIG: Minimum T-states required: -
//yet to implement
1 << PC_out | 1 << MAR_in |
1 << PC_inc | 1 << MEM_out | 1 << IR_in |
1 << PC_load | 1 << IR_out |
0
0

JNZ: Minimum T-states required: 3
1 << PC_out | 1 << MAR_in |
1 << PC_inc | 1 << MEM_out | 1 << IR_in |
1 << PC_load | 1 << IR_out |
0
0

MOVBA: Minimum T-states required: 3
1 << PC_out | 1 << MAR_in |
1 << PC_inc | 1 << MEM_out | 1 << IR_in |
1 << REGA_in | 1 << REGB_out |
0
0

MOVAB: Minimum T-states required: 3
1 << PC_out | 1 << MAR_in |
1 << PC_inc | 1 << MEM_out | 1 << IR_in |
```

```
1 << REGA_out | 1 << REGB_in |
0
0

MOVAC: Minimum T-states required: 3
1 << PC_out | 1 << MAR_in |
1 << PC_inc | 1 << MEM_out | 1 << IR_in |
1 << REGA_out | 1 << REGC_in |
0
0

MOVCB:  Minimum T-states required: 3
1 << PC_out | 1 << MAR_in |
1 << PC_inc | 1 << MEM_out | 1 << IR_in |
1 << REGB_in | 1 << REGC_out |
0
0

HLT:  Minimum T-states required: 3
1 << MAR_in |
1 << MEM_out | 1 << IR_in |
0
0
0
```

Contents of EEPROM (control words) in cpu_timing_controller
The hexadecimal numbers (in green) are loaded into the EEPROM

Depending on the different values of NZF and Greater Flag, the content loaded into the EEPROM differs. The content of the rest of the instructions remains the same, only that of JNZ and JIG changes.

The following is the table when Greater Flag=0 and NZF=0 :

Op.No.	Operation	T0	T1	T2	T3	T4	padding		
0000	NOP	0x2080	0x4600	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000
0001	LDA	0x2080	0x4600	0x0180	0x0440	0x0000	0x0000	0x0000	0x0000
0010	STA	0x2080	0x4600	0x0180	0x0820	0x0000	0x0000	0x0000	0x0000
0011	LDI	0x2080	0x4600	0x0140	0x0000	0x0000	0x0000	0x0000	0x0000
0100	ADD	0x2080	0x4600	0x0180	0x0410	0x0041	0x0000	0x0000	0x0000
0101	SUB	0x2080	0x4600	0x0180	0x0410	0x0041	0x0000	0x0000	0x0000
0110	CMP	0x2080	0x4600	0x0180	0x0410	0x0001	0x0000	0x0000	0x0000
0111	SWAP	0x2080	0x4600	0x0030	0x0042	0x000C	0x0000	0x0000	0x0000
1000	JMP	0x2080	0x4600	0x1100	0x0000	0x0000	0x0000	0x0000	0x0000
1001	JIG	0x2080	0x4600	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000
1010	JNZ	0x2080	0x4600	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000
1011	MOVBA	0x2080	0x4600	0x0048	0x0000	0x0000	0x0000	0x0000	0x0000
1100	MOVAB	0x2080	0x4600	0x0030	0x0000	0x0000	0x0000	0x0000	0x0000
1101	MOVAC	0x2080	0x4600	0x0024	0x0000	0x0000	0x0000	0x0000	0x0000
1110	MOVCB	0x2080	0x4600	0x0012	0x0000	0x0000	0x0000	0x0000	0x0000
1111	HLT	0x0080	0x0600	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000

In case the Greater Flag=0 and NZF=1 :

Op.No.	Operation	T0	T1	T2	T3	T4	padding		
1001	JIG	0x2080	0x4600	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000
1010	JNZ	0x2080	0x4600	0x1100	0x0000	0x0000	0x0000	0x0000	0x0000

In case the Greater Flag=1 and NZF=0 :

Op.No.	Operation	T0	T1	T2	T3	T4	padding		
1001	JIG	0x2080	0x4600	0x1100	0x0000	0x0000	0x0000	0x0000	0x0000
1010	JNZ	0x2080	0x4600	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000

In case the Greater Flag=1 and NZF=1 :

Op.No.	Operation	T0	T1	T2	T3	T4	padding		
1001	JIG	0x2080	0x4600	0x1100	0x0000	0x0000	0x0000	0x0000	0x0000
1010	JNZ	0x2080	0x4600	0x1100	0x0000	0x0000	0x0000	0x0000	0x0000

SECTION-5

The System reset

- Resets the value in program counter to 0
- Resets the greater, equal, lesser, carry, zero flags to zero
- Resets the T state in controller to 0

Register A, B and C are connected to Hex Displays. The status register’s output is connected to LEDs to show the value of the flags. The controller's control word is connected to 16 LEDs corresponding to each control bit.
We have added comments (textboxes) in every subcircuit in circuitverse explaining it.

EXTRA CREDIT

SIMPLE:

Programs:

- **A multiplication routine using repeated addition:**

0. LDA address
1. MOVAC
2. LDI 0
3. ADD address
4. SWAP
5. SUB address(1)
6. SWAP
7. JNZ 3
8. HALT

Example: 1A D0 30 4B 70 5C 70 A3 FF
Here, memory location C has 1, A and B have the two numbers which need to be multiplied.
(refer to drive for video of running the program)

- **Alternatively add and subtract 4 numbers at memory locations A,B,C,D: numbers at A-B+C-D**

0. LDA address
1. SUB address
2. ADD address
3. SUB address
4. MOVAC(OUT)
5. HALT

Example: 1A 5B 4C 5D 90 FF

- **Integer Floor Division**
  0. LDA address
  1. MOVAC
  2. LDI 0
  3. SWAP
  4. ADD address
  5. CMP address
  6. JIG 10
  7. SWAP
  8. ADD address
  9. JMP 3
  10. SWAP
  11. ADD address
  12. HLT

Example: 1D D0 30 70 4D 6E 9A 70 4F 83 70 4F FF
Here, memory address F has 1, D has the divisor and E has the dividend such that (value at E> value at D).This calculates floor(E/D).
The final value in REG_A after the program is the quotient of division.
(refer to drive for video of running the program)

MODERATE:

Program to demonstrate JNZ (Jump if not zero) and JIG (Jump if greater):
(Conditional Jump Statements)

- **A multiplication routine using repeated addition:**

0. LDA address
1. MOVAC
2. LDI 0
3. ADD address
4. SWAP
5. SUB address(1)
6. SWAP
7. JNZ 3
8. HALT

Example: 1A D0 30 4B 70 5C 70 A3 FF
Here, memory location C has 1, A and B have the two numbers which need to be multiplied.
(refer to drive for video of running the program)

- **Integer Floor Division**

0. LDA address
1. MOVAC
2. LDI 0
3. SWAP
4. ADD address
5. CMP address
6. JIG 10
7. SWAP
8. ADD address
9. JMP 3
10. SWAP
11. ADD address
12. HLT

Example: 1D D0 30 70 4D 6E 9A 70 4F 83 70 4F FF
Here, memory address F has 1, D has the divisor and E has the dividend such that (value at E> value at D).This calculates floor(E/D).
The final value in REG_A after the program is the quotient of division.
(refer to drive for video of running the program)

ADVANCED:

4) a) The Design of the Finite State Machine (FSM) can be as follows: