CS2310
FUNDAMENTALS OF COMPUTER DESIGNING

AKSHITHA (CS22B019),ANJALI (CS22B046)

November 14, 2023

Contents

1 ARCHITECTURE

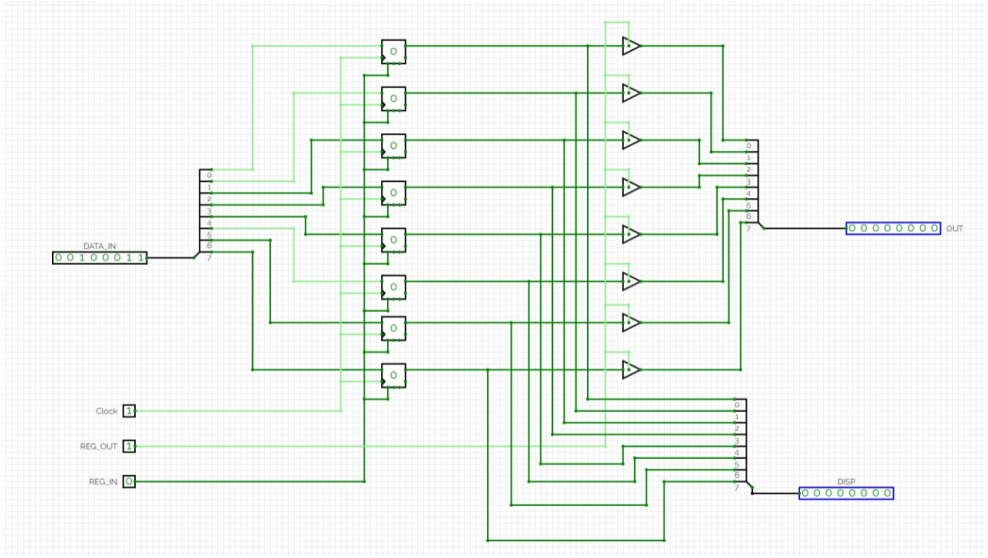
Gajendra-1 represents a computer system characterized by a bus-organized architecture.In this design,a common pathway known as bus,is employed to efficiently manage the exchange of both data and instructions among various components within the circuit.
The layout of the circuit is shown below and lets look into the components in detail:

Internal components of the gajendra-1:

- GENERAL CPU REGISTER
- INSTRUCTION REGISTER
- MEMORY ADDRESS REGISTER
- PROGRAM COUNTER
- ARITHMETIC AND LOGICAL UNIT
- STATUS REGISTER

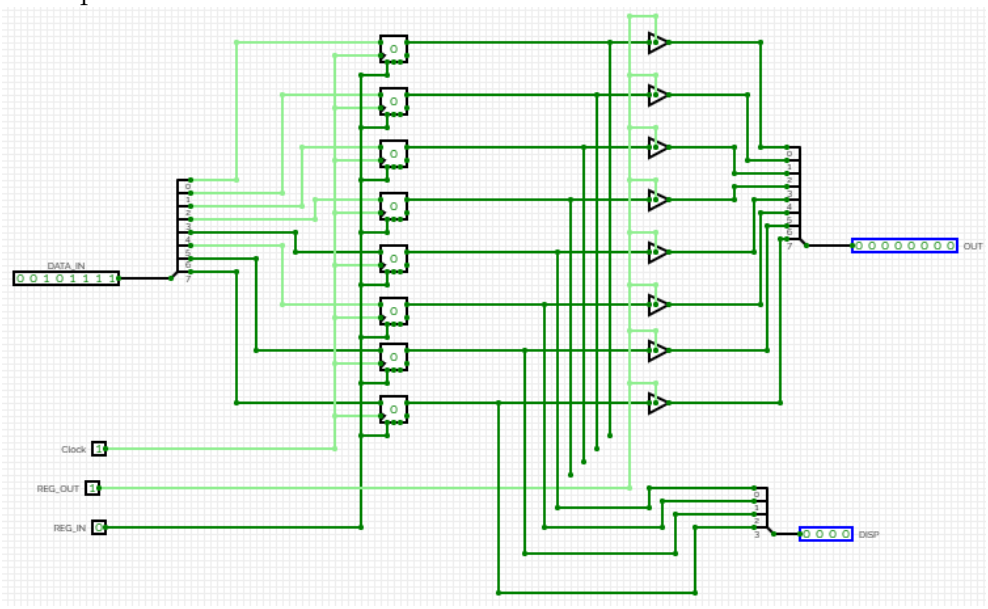
1.1 GENERAL CPU REGISTER

A register is normally used for temporary storage of a related set of bits for some operation.An 8-bit-register is constructed by integrating eight D-flipflops,all synchronized by a common clock.Additionally,eight tri-state buffers are employed in conjuction with the register.When REG_IN is set to 1, the register acquires an 8-bit input,DATA_IN,from the common bus.REG_OUT serves as an enable for tri-state buffer,regulating the output(OUT) that the register provides back to the bus.Simultaneously,the output of the register,contained in the DISP, is not subjected to external control.



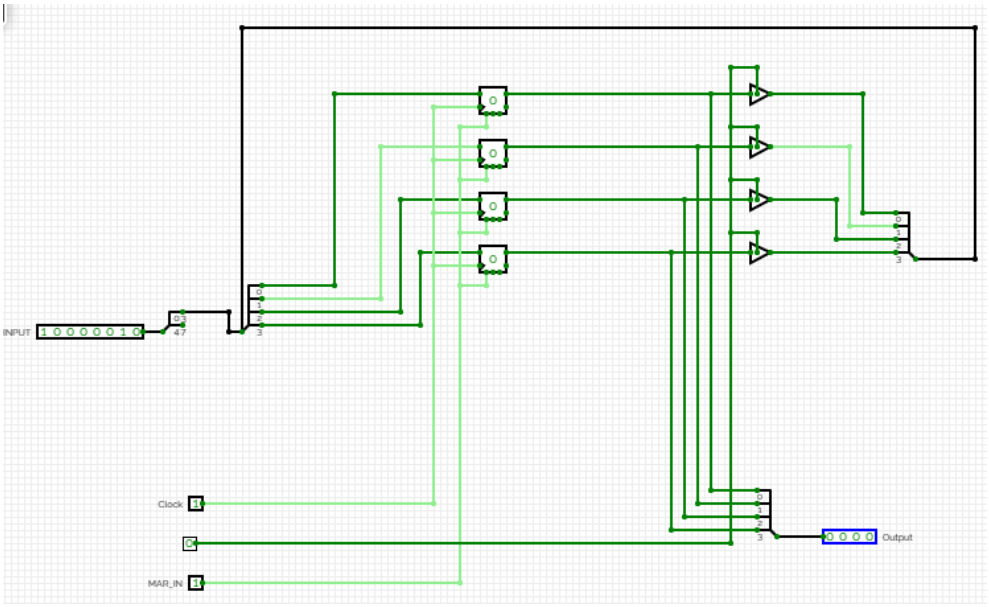
1.2 INSTRUCTION REGISTER

The instruction register (IR) serves as a temporary repository for the current instruction throughout its execution. This register receives its input, denoted as DATA_IN, from the common bus under the control of the REG_IN signal. The initial four bits of this input explicitly define the instruction to be carried out. The DISP output conveys these initial four bits to the control logic, providing the necessary instruction for the completion of the current operation.



1.3 MEMORY ADDRESS REGISTER

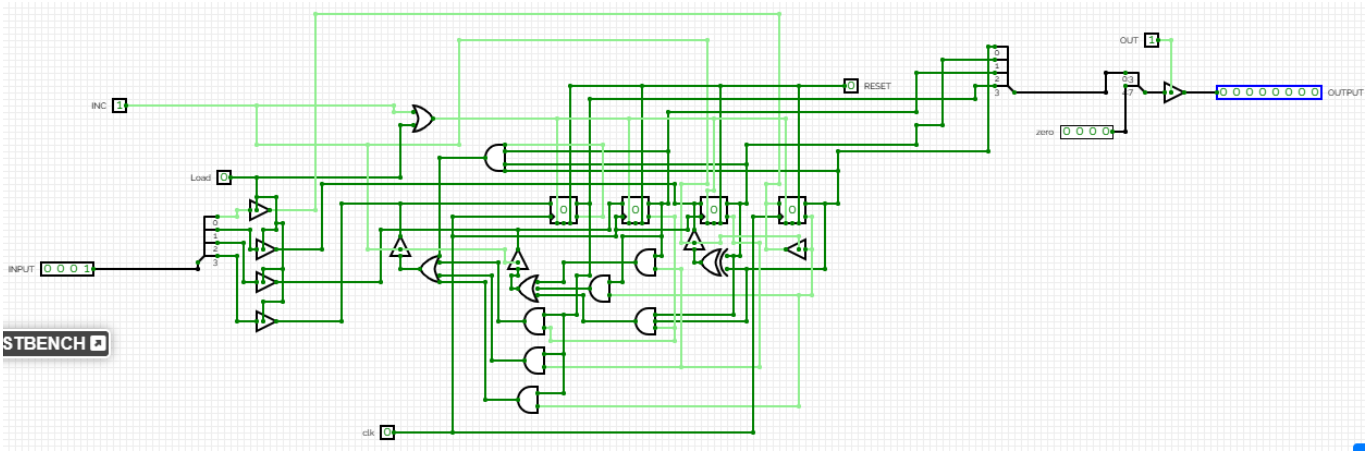
Memory Address Register (MAR) functions as a pivotal element in the memory access process.Its job is to remember the address of the specific spot in memory we want to read. It sends this address to the RAM using a 4-bit output known as Output.The program counter address smoothly transfers into the Memory Address Register (MAR) by means of Input when MAR_IN is set to 1.



1.4 PROGRAM COUNTER

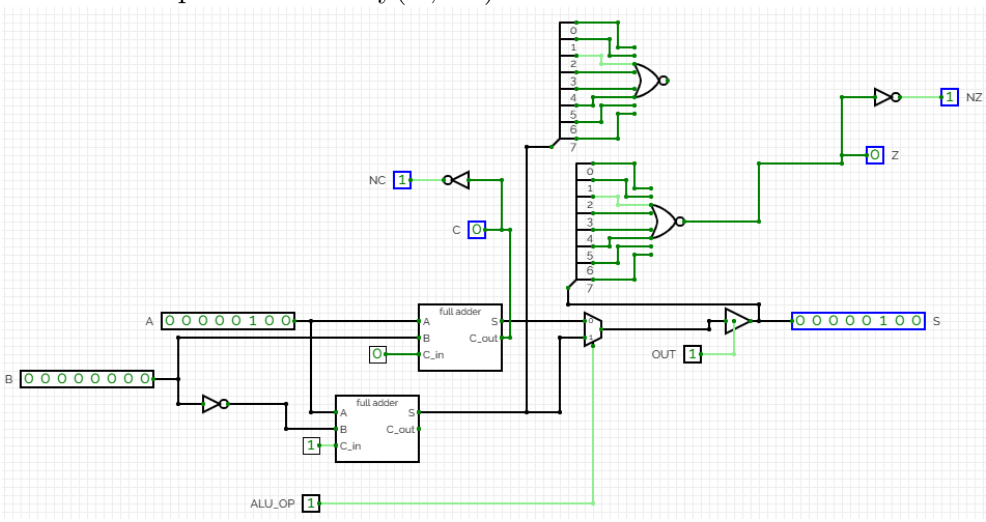
The program is initially stored at the outset of memory,with the first instruction residing at binary address 0000,the second at address 0001,and so on.The program counter is instrumental in this process,as it systematically increments from 0000 to 1111,serving the crucial role of transmitting the address of the next instruction to be fetched and executed from memory.
Control inputs of PC:

- INC: Increments the count by one given a clock pulse.
- LOAD: The PC will be loaded with the values available in the common bus.We consider the least significant 4-bits as the address.
- OUT: The PC's current count value will be output to the bus.Again,we note that only 4-bits are important- the additionally most significant 4-bits will be ignored by the rest of the circuitry
- RESET: Resets the PC count value to zero.



1.5 ARITHMETIC AND LOGICAL UNIT

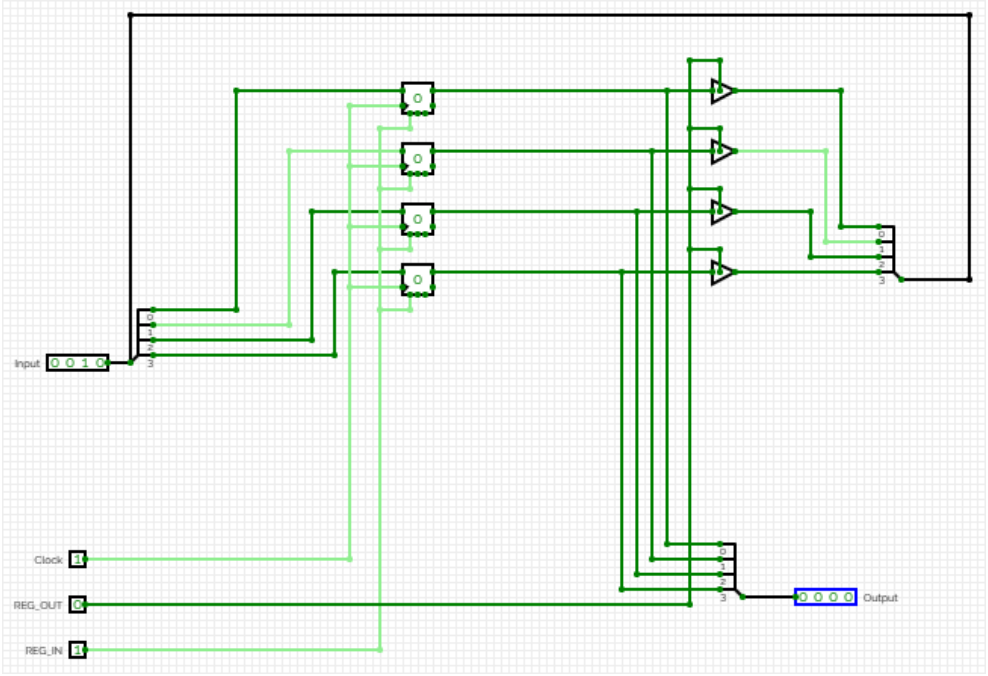
The Arithmetic Logic Unit (ALU) is responsible for executing both addition and subtraction operations. The ALU operates based on the value of ALU_OP: when ALU_OP is 1, it performs $A - B$, and when it's 0, the operation is $A + B$.When the OUT signal is set to 1, the ALU writes the result(S) to the common bus.The ALU is asynchronous, meaning its output changes immediately upon any alteration in its inputs. Additionally, the ALU is equipped to determine whether any of its operations result in a zero output(Z,NZ).it also provides the output of the carry(C,NC).



1.6 STATUS REGISTER

Status Register is a 4-bit register which stores essential flags denoted as C (Carry), NC (No Carry), Z (Zero), and NZ (Not Zero).Upon the ALU writing its output to the common bus, the Status Register takes its input from the ALU, and the resulting flags are displayed.This is done by connecting the REG_IN directly to the OUT of ALU.This configuration ensures

that the status of key arithmetic conditions is efficiently stored and displayed.



2 INSTRUCTION SET

We define our own instruction set.We assign specific machine codes for each instruction. The operation bits from the opcode needs to be passed from the instruction register (IR) to the controller to interpret what instruction needs to be executed.

Note that if K is the address then [K] denotes the value at the address.

2.1 NOP

NOP

Description:
The instruction performs no operation and increments the program counter.

Operation:
(i) No

Syntax:	**Operands:**	**Program counter:**
(i) *NOP*	None	$PC \leftarrow PC + 1$

8-bit Opcode:

0000	*XXXX*

2.2 LDA

Description:
Loads value from the given address into the Accumulator(REG_A)

Operation:
(i) $R_A \leftarrow [K]$

Syntax:	**Operands:**	**Program counter:**
(i) $LDA\ K$	$0 \leq K \leq 15$	$PC \leftarrow PC + 1$

8-bit Opcode:

0001	$0xK(XXXX)$

2.3 STA

Description:
Stores value from the Accumulator(REG_A) into the given address.

Operation:
(i) $[K] \leftarrow R_A$

Syntax:	**Operands:**	**Program counter:**
(i) $STA\ K$	$0 \leq K \leq 15$	$PC \leftarrow PC + 1$

8-bit Opcode:

0010	$0xK(XXXX)$

2.4 ADD

Description:
It updates the value stored in the accumulator by adding the value stored in the given location to the value stored in it.

Operation:
(i) (i)$R_B \leftarrow [K]$
(ii)$R_A \leftarrow R_A + R_B$

Syntax:	**Operands:**	**Program counter:**
(i) $ADD\ K$	$0 \leq K \leq 15$	$PC \leftarrow PC + 1$

8-bit Opcode:

0011	$0xK(XXXX)$

2.5 SUB

Description:
It updates the value stored in accumulator by subtracting the value stored in the given location to the value stored in it

Operation:
(i) (i)$R_B \leftarrow [K]$
(ii)$R_A \leftarrow R_A + R_B$

Syntax:	**Operands:**	**Program counter:**
(i) $SUB\ K$	$0 \leq K \leq 15$	$PC \leftarrow PC + 1$

8-bit Opcode:

0100	$0xK(XXXX)$

2.6 LDI

Description:
Loads a 4-bit constant value directly to the accumulator.

Operation:
(i) $R_A \leftarrow K$

Syntax:	**Operands:**	**Program counter:**
(i) $LDI\ K$	$0 \leq K \leq 15$	$PC \leftarrow PC + 1$

8-bit Opcode:

0101	$K(XXXX)$

2.7 JMP

Description:
The program counter jumps to the given 4-bit address unconditionally.

Operation:
(i) $PC \leftarrow K$

Syntax:	**Operands:**	**Program counter:**
(i) $JMP\ K$	$0 \leq K \leq 15$	$PC \leftarrow K$

8-bit Opcode:

0110	$0xK(XXXX)$

2.8 SWAP

Description:
Swaps the values present in REG_A and REG_C using temporary register (i.e. REG_B).

Operation:
(i) (i)$R_B \leftarrow R_A$
(ii)$R_A \leftarrow R_C$
(iii)$R_C \leftarrow R_B$

Syntax:	**Operands:**	**Program counter:**
(i) $SWAP$	–	$PC \leftarrow PC + 1$

8-bit Opcode:

0111	$XXXX$

2.9 JNZ

Description:
The program counter jumps to the given 4-bit address if the value of ALU_OUT is non zero.

Operation:
(i) $PC \leftarrow K$

Syntax:	**Operands:**	**Program counter:**
(i) $JNZ\ K$	$0 \leq K \leq 15$	$PC \leftarrow K$

8-bit Opcode:

1000	$0xK(XXXX)$

2.10 MOVAC

Description:
Copy the values present in REG_A to REG_C,the value in REG_A remains unchanged.

Operation:
(i) (i)$R_C \leftarrow R_A$

Syntax:	**Operands:**	**Program counter:**
(i) $MOVAC$	–	$PC \leftarrow PC + 1$

8-bit Opcode:

1001	$XXXX$

3 ASSEMBLY PROGRAMS

EXAMPLE 1
Adding two numbers (A, B):
Let the values of A, B be stored at locations $0xC$ and $0xD$ respectively.

Address	Assembly code	Machine Code
$0x0$	LDA $0xC$	$0x1C$
$0x1$	ADD $0xD$	$0x3D$
$0x2$	MOVAC	$0x90$

EXAMPLE 2
Adding and subtracting 4 numbers in some combination:

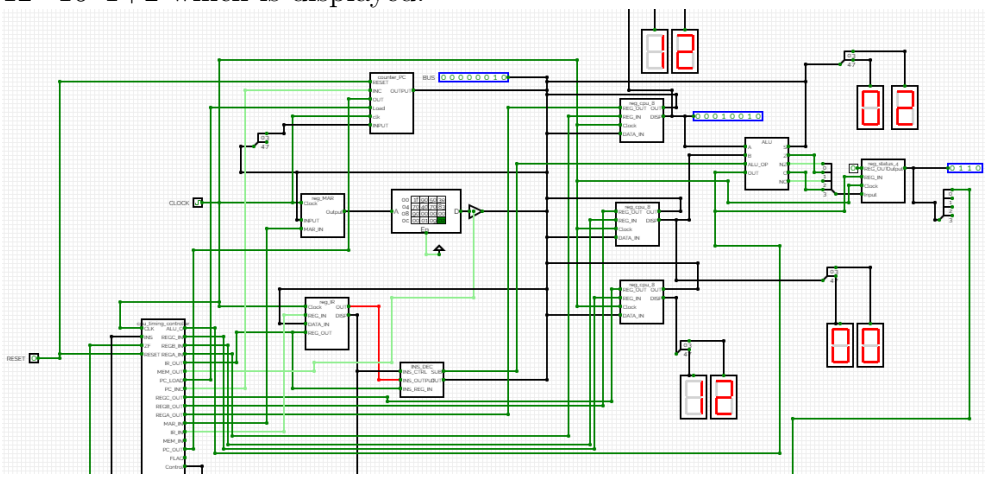
Let $E = A + B - C + D$
The values of A, B, C, D, E be stored at locations $0xC$,$0xD$,$0xE$,$0xF$ and $0xB$ respectively.

Address	Assembly code	Machine Code
$0x0$	LDA 0xC	0x1C
$0x1$	ADD 0xD	0x3D
$0x2$	SUB 0xE	0x4E
$0x3$	ADD 0xF	0x3F
$0x4$	STA 0xB	0x2B

EXAMPLE 3
Multiplication using repeated addition:
Say we want to perform $a * b$.
Let the values of $a, b, 1$ be stored at the locations $0xF$,$0xE$,oxD.

Address	Assembly code	Machine Code
0x0	LDA 0xF	0x1F
0x1	MOVAC	0x90
0x2	LDI 0	0x50
0x3	ADD 0xE	0x3E
0x4	SWAP	0x70
0x5	SUB 0xD	0x4D
0x6	SWAP	0x70
0x7	JNZ 0x3	0x83

Consider 9*2 9 is stored in location in location 0xE and 2 in location 0xF.The output is 12=16*1+2 which is displayed.



4 MICROINSTRUCTIONS AND CONTROLLER LOGIC DESIGN

We define our own control word format. The size of the control word is up to our design.We have taken size of control word as 16.
We can do this controller both hardware and software based.As of now,we have made a software design. We have to use a fixed number of T-states for all instructions to keep the design simple. Here, we use 5 T-states.

The control unit is the key to the operations formed.The control unit generates the control words that fetch and execute each instruction.While each instruction is fetched and executed,it passes through different timing states(T-states).We take 5 T-states(two fetch and three execute).These five states are called a machine cycle.

For each instruction,we define the sequence of microinstructions or control words that need to be executed in sequence which are given below:

4.1 NOP

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

The minimum number of T-states : 3

4.2 LDA

```
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<<REGA_IN
T4:  0
```

The minimum number of T-states : 4

4.3 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
```

The minimum number of T-states : 4

4.4 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
```

The minimum number of T-states : 5

4.5 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<<REGA_IN
T4:  1<<ALU_OUT | 1<<REGA_IN
```

The minimum number of T-states : 5

4.6 JMP

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

The minimum number of T-states : 3

4.7 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<<REGA_IN | 1<<REGC_OUT
T4:  1<<REGB_OUT| 1<<REGC_IN
```

The minimum number of T-states : 5

4.8 JNZ

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

The minimum number of T-states : 3

4.9 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
```

The minimum number of T-states : 3

4.10 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
```

The minimum number of T-states : 3

4.11 MOVBA

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

The minimum number of T-states : 3

4.12 MOVCB

```
T0:  1<<PC_OUT  | 1<<MAR_IN
T1:  1<<PC_INC  | 1<<MEM_OUT  | 1<<IR_IN
T2:  1<<REGC_OUT| 1<<REGB_IN
T3:  0
T4:  0
```

The minimum number of T-states : 3

4.13 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
```

The minimum number of T-states : 3

4.14 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
```

The minimum number of T-states : 3

4.15 MOVBC

```
T0:  1<<PC_OUT  | 1<<MAR_IN
T1:  1<<PC_INC  | 1<<MEM_OUT  | 1<<IR_IN
T2:  1<<REGB_OUT| 1<<REGC_IN
T3:  0
T4:  0
```

The minimum number of T-states : 3

4.16 HLT

```
T0:  1<<MAR_IN
T1:  1<<MEM_OUT  | 1<<IR_IN
T2:  1<<FLAG
T3:  0
T4:  0
```

The minimum number of T-states : 3

5 SYSTEM RESET

A system reset typically refers to a process where a circuit is brought back to its initial state or starting point.During this reboot, the processor's program counter is set to 0x0, and internal counters within the system are reset to their initial states.It involves three steps
1)Set RESET to 1
2)Toggle the clock
3)Set RESET to 0