Gajendra-I Processor

Lab Report

submitted by

Chaitanya (CS22B036), Rushi(CS22B040)

Under the supervision
of

Dr.Ayon Chakraborty

Computer Science and Engineering

Indian Institute of Technology Madras

Jul - Nov 2023

Contents

1 Architecture **4**
1.1 Program Counter(PC) 4
1.2 Memory 4
1.3 Memory Address Register(MAR) 5
1.4 Instruction Register(INS-REG) 6
1.5 Instruction Decoder(INS-DEC) 6
1.6 Arithmetic and Logical Unit(ALU) 7
1.7 ALU Status Register 8
1.8 Registers 8
1.8.1 Register-A(Accumulator) 9
1.8.2 Register-B 9
1.8.3 Register-C 9
1.9 Control Processor 9

2 Instruction Set **10**
2.1 NOP 10
2.2 LDA 10
2.3 STA 11
2.4 ADD 11
2.5 SUB 11
2.6 LDI 12
2.7 JMP 12
2.8 OUT 12

3 Internal Implementation Of Each Instruction **13**
3.1 NOP 13
3.2 LDA 13
3.3 STA 14
3.4 ADD 14
3.5 SUB 14
3.6 LDI 15
3.7 JMP 15
3.8 OUT 15
3.9 JNZ 16
3.10 SWAP 16
3.11 MOVAB 16
3.12 HLT 17

4 Sample Tasks **17**
4.1 Task 1 17
4.2 Task 2 17
4.3 Task 3 18
4.4 Task 4 18

List of Figures

1 Program Counter 4
2 Memory 5
3 Memory Access Register 5
4 Instruction Register 6
5 Instruction Decoder 7
6 Arithmetic and Logical Unit 7
7 ALU Status Register 8
8 Registers 9
9 Control Processor 10

1 Architecture

1.1 Program Counter(PC)

The program of a processor is stored at the begin of the memory, instruction by instruction and the program counter helps to traverse through the instruction set/program. For describing it's internal design we can say that it is a basic counter performing some additional operations.
DESIGN: A four bit register.(created using D-flip-flops)
Here, the program counter constitutes the following inputs to help us perform this task:

1. INC
   When switched on, it increments the address pointer once with each machine cycle.

2. LOAD
   When switched on, it allows PC to load data from the common bus.

3. DATA-IN
   When LOAD is switched on, this input allows the last 4 bits of the data to get loaded from the common bus to the PC.

4. RESET
   When switched on, it resets the PC to the default state of 0.

5. OUT
   When switched on, it gives the address stored to the common bus through the OUT outlet.

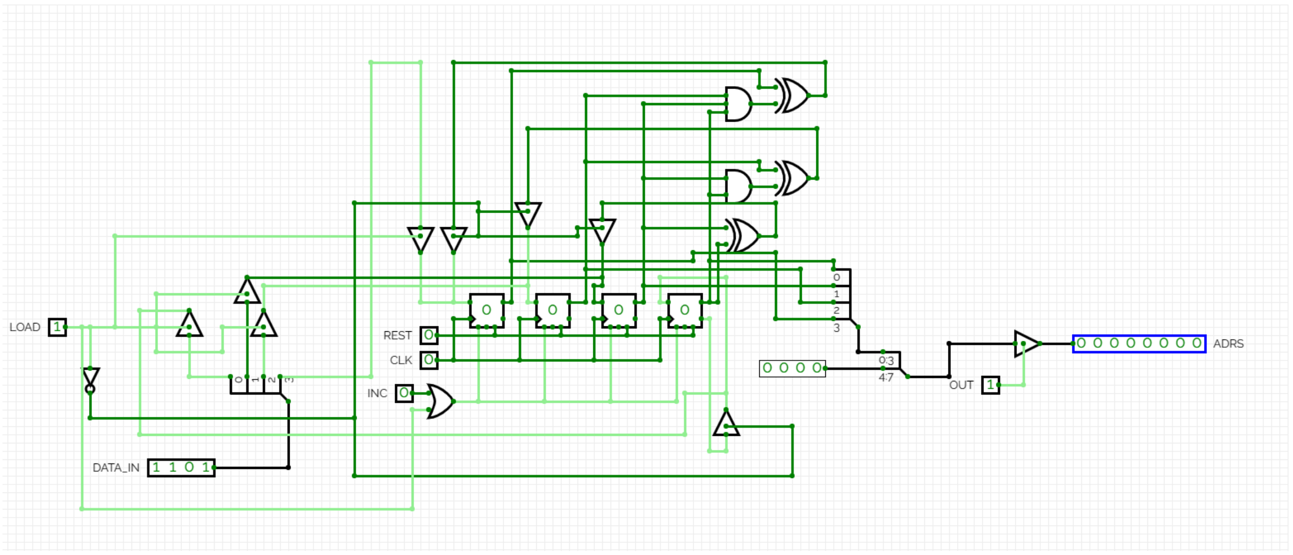


Figure 1: Program Counter

1.2 Memory

Memory stores the program at the beginning and the required data in the remaining locations. In this model we used an EEPROM which provides us the flexibility to perform operations for both reading from the memory and writing to the memory. It takes in the following parameters:

1. ADDRESS
   It takes in the address of the memory location where the required data of a particular task is stored.

2. DATA-IN
   It takes in the data from the common bus and writes it to the memory corresponding to the address when WRITE is turned on.

3. WRITE
   It is like a gate for writing to the memory.

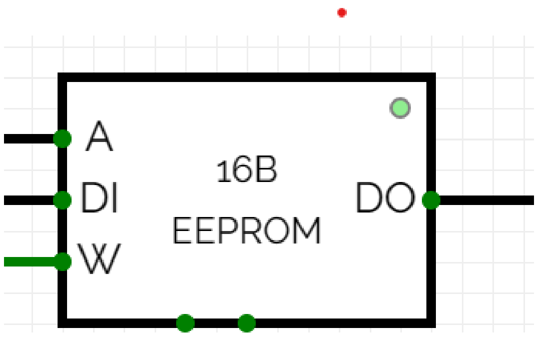


Figure 2: Memory

1.3 Memory Address Register(MAR)

It holds the address of the memory location which is presently in use. MAR is required for all operations requiring memory access.
DESIGN: A four bit register.(created using D-flip-flops)
Parameters:

1. MAR$_{in}$
   When it is turned on, the address from the common bus is loaded to MAR.

2. IN
   Takes in the actual address in the data from common bus when MAR$_{in}$ is turned on. The output OUT is connected to the memory which helps us to access the required address.

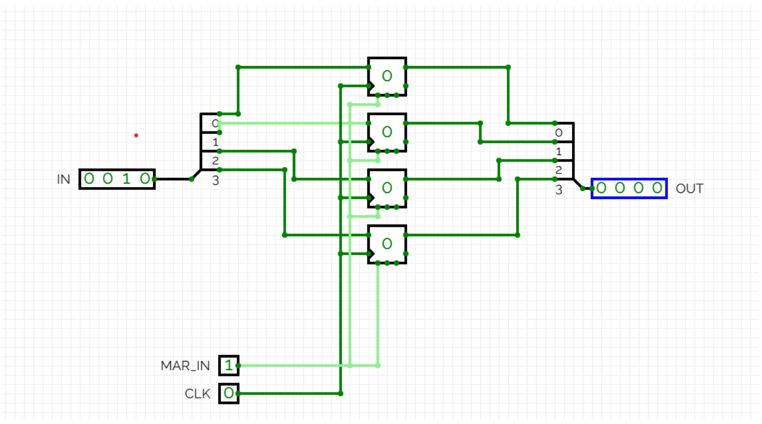


Figure 3: Memory Access Register

1.4 Instruction Register(INS-REG)

We encode instructions as 8-bit numbers for writing the program. The instruction register stores the 8-bit where the most significant 4 bits denote the instruction.
DESIGN: An 8-bit register(created using 8 D-flip-flops).
Parameters:

1. REG_{out}
   When turned on, it gives the number stored to the instruction decoder.

2. REG_{in}
   When turned on, it allow the register to store the data in the common bus.

3. IN
   The inflow of the data happens through this inlet.

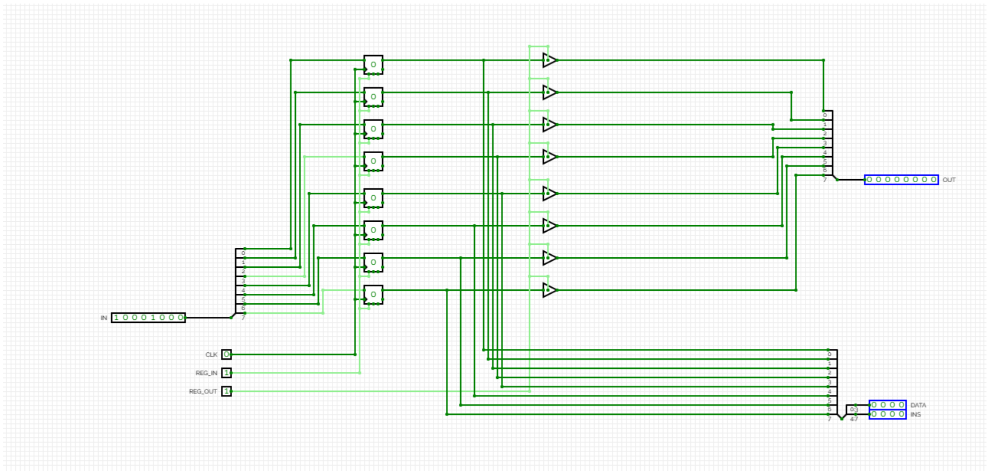


Figure 4: Instruction Register

1.5 Instruction Decoder(INS-DEC)

It controls the output of the instruction register and channels them to their respective places.
DESIGN: A decoder along with a multiplexer to perform operations of some instructions carefully.
Parameters:

1. IR_{ctrl}
   It takes in the bits which store the information about the type of instruction.

2. ID_{OUT}
   When turned on, it allows ID to output data to the common bus.

3. IR_{out}
   It takes in the output from the instruction register.

When SUB is turned on, it asks the ALU to perform subtraction operation, otherwise ALU performs the addition operation.
The output of ID is passed to the common bus through the OUT outlet.

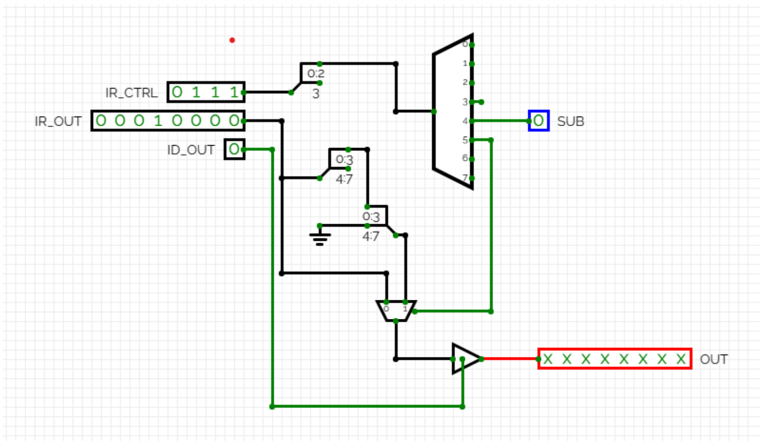


Figure 5: Instruction Decoder

1.6 Arithmetic and Logical Unit(ALU)

The ALU performs arithmetic operations. In this design the ALU is limited for computing the results for addition and subtraction.
DESIGN: Contains to 8-bit full adders one for addition and the other for subtraction. When AS is turned on the subtraction result is given as the output otherwise the addition result is given.
Parameters:

1. AS
   It takes in the value from the SUB output of ID. When turned on it outputs the subtraction result.

2. A
   Takes in the data from Register A.

3. B
   Takes in the data from Register B.

4. ALU$_{out}$
   It controls the output of the ALU to the common bus.

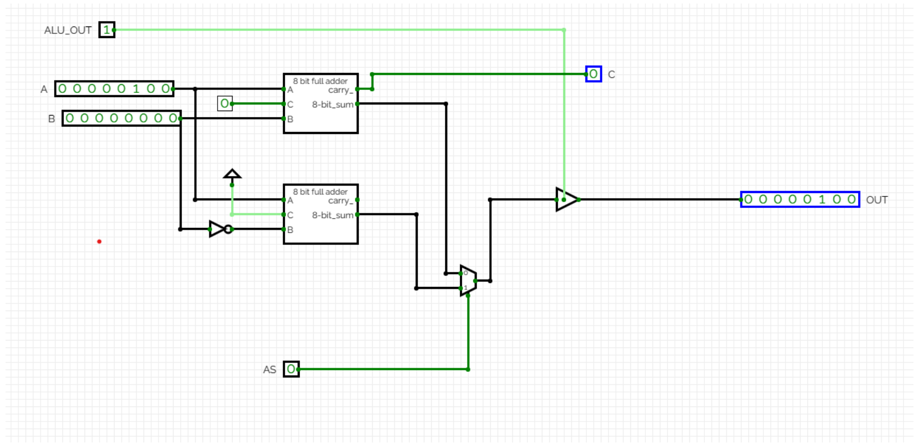


Figure 6: Arithmetic and Logical Unit

1.7 ALU Status Register

The output to the ALU is fetched by the ALU Status Register. It helps in performing conditional jumps by providing information about the output.
DESIGN: A 4-bit register where each bit starting from the least significant bit indicates Non-Zero Flag, Zero-Flag, Non-Carry Flag and Carry Flag respectively.
Parameters:

1. IN
   It takes in the sum digit of the result of the ALU.

2. C
   It takes in the carry digit of the ALU.

NZ, Z, NC, C provides information regarding the output.

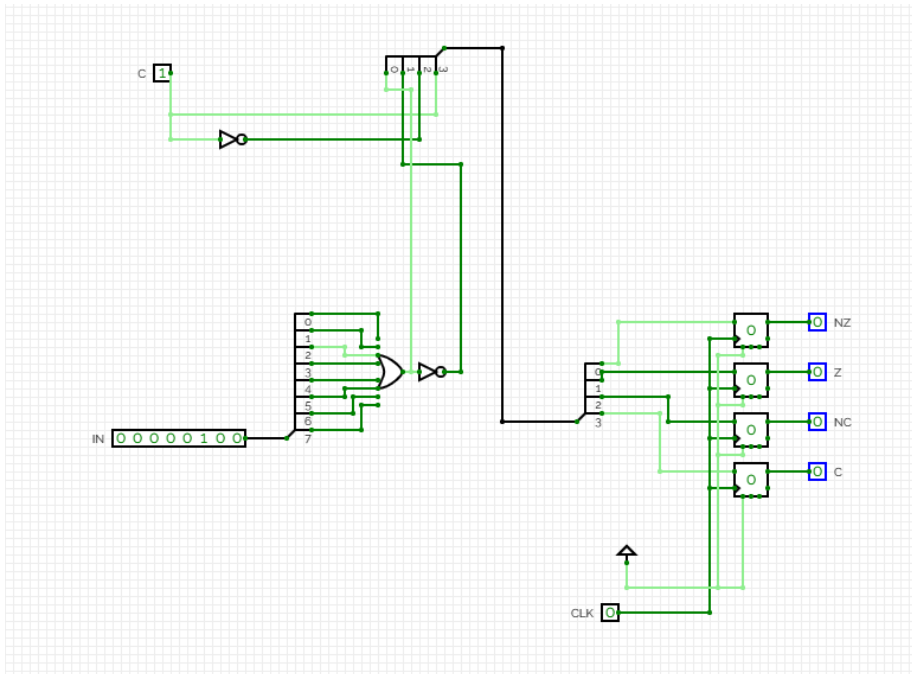


Figure 7: ALU Status Register

1.8 Registers

Registers are used to storing data for temporary or near immediate use. These are used for using data for ALU operations etc. In this design we mainly use 8-bit registers for using temporary data.
DESIGN: A simple 8-bit register with a common wire for both input and output controlled using REG$_{in}$ and REG$_{out}$.
Parameters:

1. REG$_{in}$
   If turned on, it takes in the input from the common bus.

2. REG$_{out}$
   If turned on, it gives the number stored to the common bus.

3. Data$_{i/o}$
   The data input/output is done using this wire.

DISP can be used to show the number stored in the register.

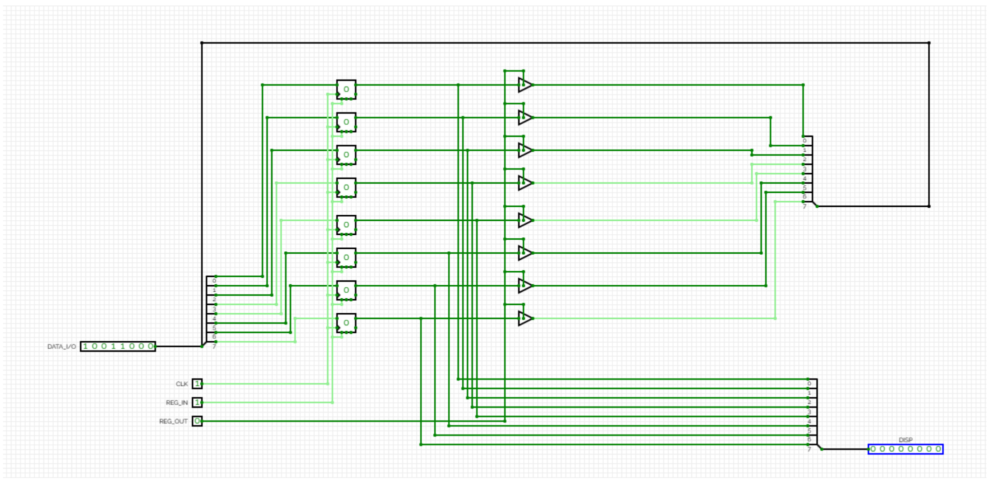


Figure 8: Registers

1.8.1 Register-A(Accumulator)

It can be thought of being the main register of this processor. We retrieve data from the memory and load it to this register. ALU performs operations and loads this result to the Accumulator.

1.8.2 Register-B

This register is used as temporary memory. When we perform add or subtract operation we store the second operand in Register B. Contents of register A, B are passed to the ALU later on for calculation.

1.8.3 Register-C

This register is mainly used to display the data. When we use the OUT instruction, we load the data to register C. This register has a hex display attached to it.

1.9 Control Processor

Control Processor is responsible for complete implementation of the program by stepwise implementation of each instruction(which itself requires stepwise implementation of several control words).
DESIGN: Ring Counter for traversing through all the T-states/subinstructions, an EEPROM which stores control words for every T-state for each instruction and an encoder.
EEPROM Programming: The EEPROM is loaded in such a way that each instruction is allotted 8 memory locations where each memory location corresponds to each subinstruction.

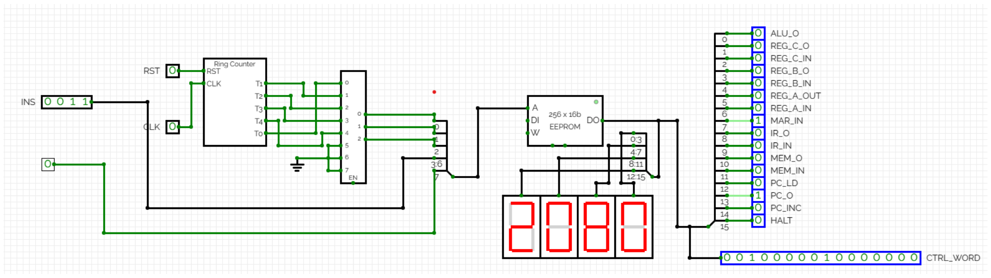


Figure 9: Control Processor

2 Instruction Set

2.1 NOP

Description

Loads the Instruction Register and increments the Program counter. No other operation is carried out.

Operation

(i) None

	Syntax:	Operands:	Program Counter
(i)	NOP X	None	PC 🡐 PC + 1

8-bit op-code:

0000	XXXX

2.2 LDA

Description

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

Operation

(i) R_A 🡐 (X)

	Syntax:	Operands:	Program Counter
(i)	LDA X	$0 \leq X \leq 15$	PC 🡐 PC + 1

8-bit op-code:

0001	0xX(xxxx)

2.3 STA

Description

It writes the value in Register A to the memory. (In the location whose address is provided along with the instruction)

Operation

(i) $(X) \leftarrow R_A$

	Syntax:	Operands:	Program Counter
(i)	STA X	$0 \leq X \leq 15$	$PC \leftarrow PC + 1$

8-bit op-code:

0010	0xX(xxxx)

2.4 ADD

Description

Add the value at the given address to accumulator.

Operation

(i) $R_B \leftarrow (X)$
(ii) $R_A \leftarrow R_A + R_B$

	Syntax:	Operands:	Program Counter
(i)	ADD X	$0 \leq X \leq 15$	$PC \leftarrow PC + 1$

8-bit op-code:

0011	0xX(xxxx)

2.5 SUB

Description

Subtract the value at the given address from the accumulator.

Operation

(i) $R_B \leftarrow (X)$
(ii) $R_A \leftarrow R_A - R_B$

	Syntax:	Operands:	Program Counter
(i)	ADD X	$0 \leq X \leq 15$	$PC \leftarrow PC + 1$

8-bit op-code:

0100	0xX(xxxx)

2.6 LDI

Description

Loads the value given along with the instruction to the Register A.

Operation

(i) $R_A \leftarrow X$

	Syntax:	Operands:	Program Counter
(i)	LDI X	$0 \leq X \leq 15$	$PC \leftarrow PC + 1$

8-bit op-code:

0101	0xX(xxxx)

2.7 JMP

Description

Jumps to the instruction of the program which is present in the given address.

Operation

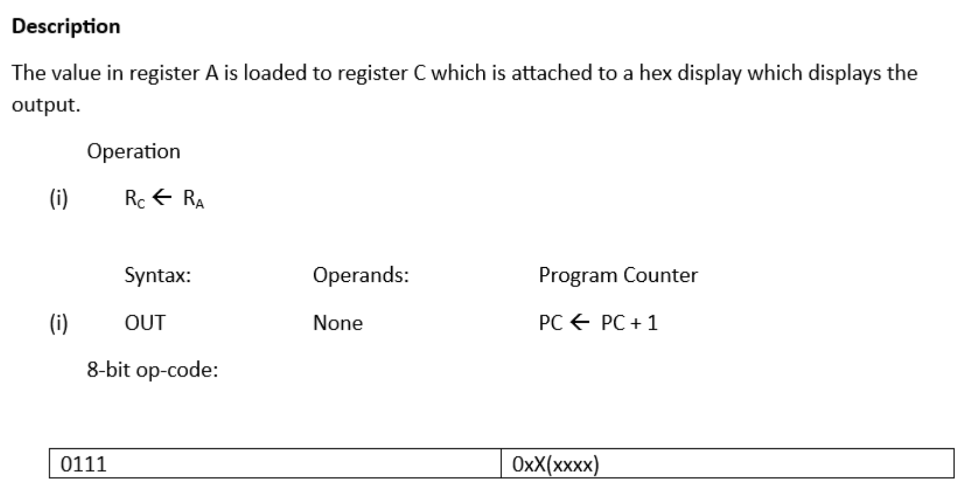
(i) $PC \leftarrow (X)$

	Syntax:	Operands:	Program Counter
(i)	JMP X	$0 \leq X \leq 15$	$PC \leftarrow X$

8-bit op-code:

0110	0xX(xxxx)

2.8 OUT


Description

The value in register A is loaded to register C which is attached to a hex display which displays the output.

Operation

(i) $R_C \leftarrow R_A$

	Syntax:	Operands:	Program Counter
(i)	OUT	None	PC ← PC + 1

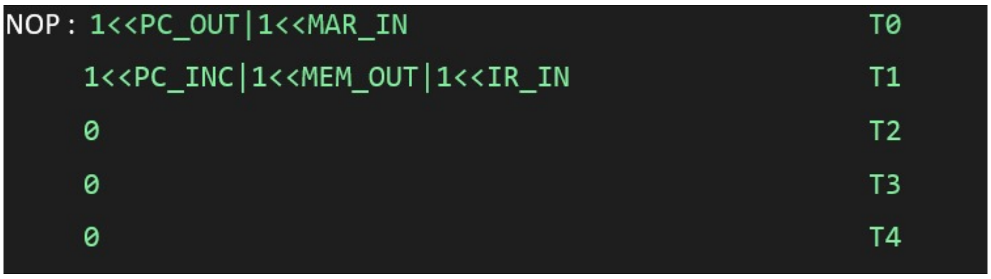
8-bit op-code:

0111	0xX(xxxx)

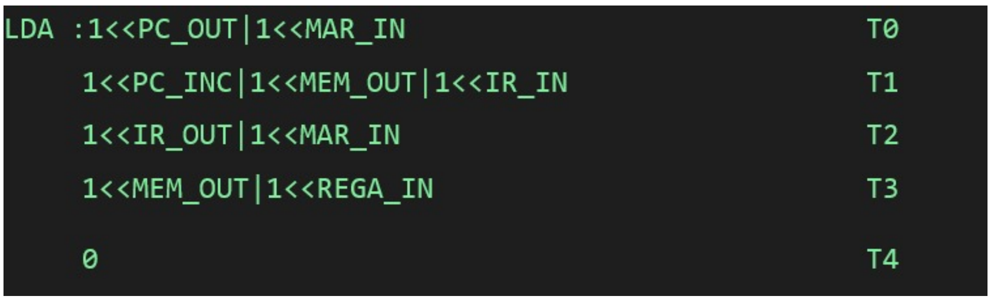
other instructions like **JNZ, SWAP, HALT, MOV** can also be performed using this cpu.

3 Internal Implementation Of Each Instruction

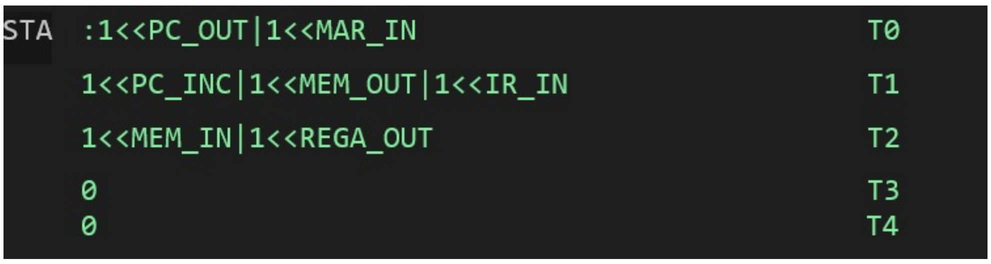
3.1 NOP


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

3.2 LDA

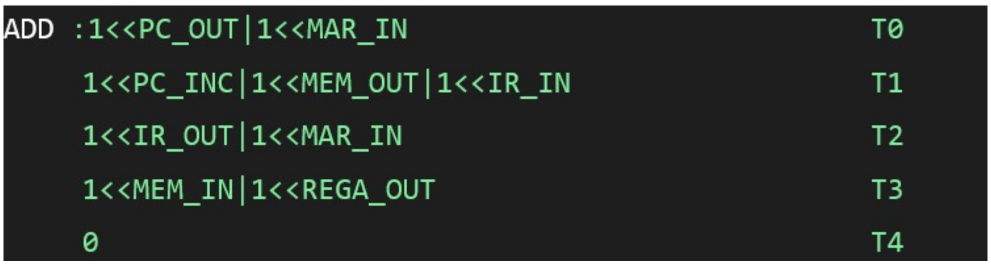

```
LDA :1<<PC_OUT|1<<MAR_IN                                    T0
     1<<PC_INC|1<<MEM_OUT|1<<IR_IN                          T1
     1<<IR_OUT|1<<MAR_IN                                    T2
     1<<MEM_OUT|1<<REGA_IN                                  T3

     0                                                      T4
```

3.3 STA



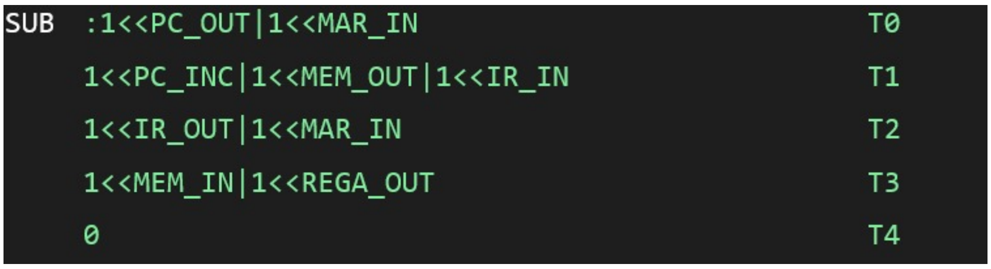
```
STA  :1<<PC_OUT|1<<MAR_IN                              T0
      1<<PC_INC|1<<MEM_OUT|1<<IR_IN                    T1
      1<<MEM_IN|1<<REGA_OUT                            T2
      0                                                T3
      0                                                T4
```

3.4 ADD



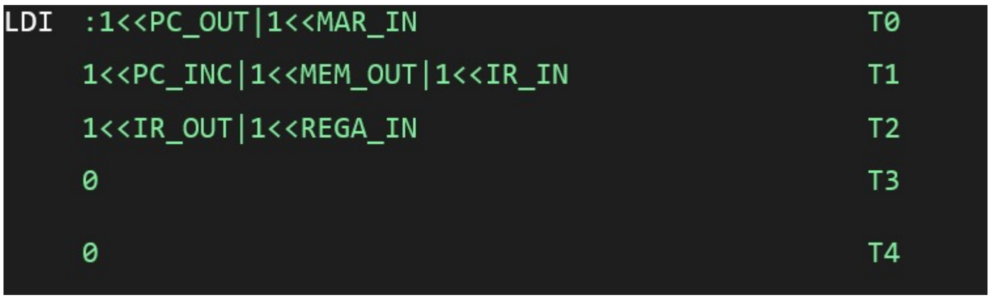
```
ADD  :1<<PC_OUT|1<<MAR_IN                              T0
      1<<PC_INC|1<<MEM_OUT|1<<IR_IN                    T1
      1<<IR_OUT|1<<MAR_IN                              T2
      1<<MEM_IN|1<<REGA_OUT                            T3
      0                                                T4
```

3.5 SUB



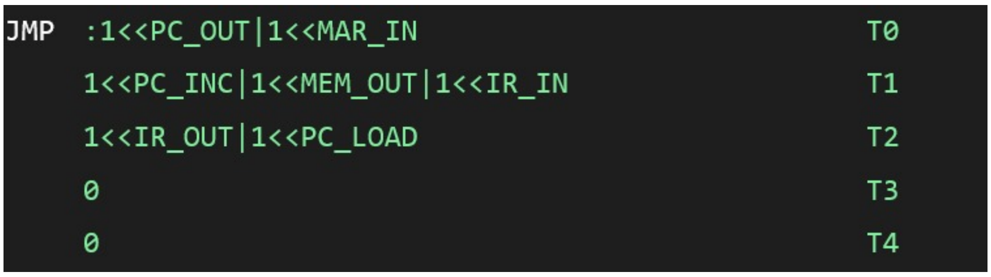
```
SUB  :1<<PC_OUT|1<<MAR_IN                              T0
      1<<PC_INC|1<<MEM_OUT|1<<IR_IN                    T1
      1<<IR_OUT|1<<MAR_IN                              T2
      1<<MEM_IN|1<<REGA_OUT                            T3
      0                                                T4
```

3.6 LDI



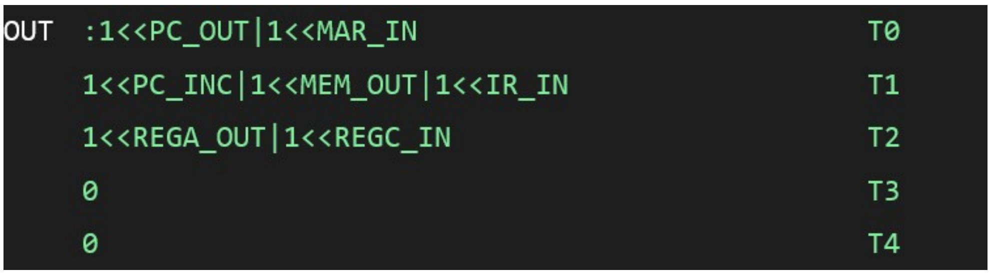
```
LDI  :1<<PC_OUT|1<<MAR_IN                              T0
      1<<PC_INC|1<<MEM_OUT|1<<IR_IN                     T1
      1<<IR_OUT|1<<REGA_IN                              T2
      0                                                 T3
      0                                                 T4
```

3.7 JMP



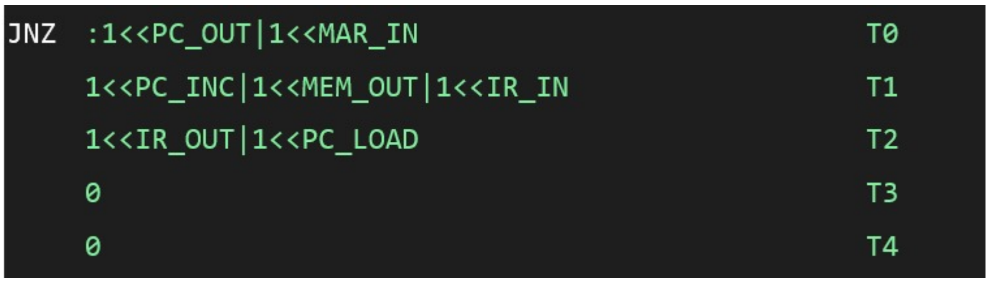
```
JMP  :1<<PC_OUT|1<<MAR_IN                              T0
      1<<PC_INC|1<<MEM_OUT|1<<IR_IN                     T1
      1<<IR_OUT|1<<PC_LOAD                              T2
      0                                                 T3
      0                                                 T4
```

3.8 OUT



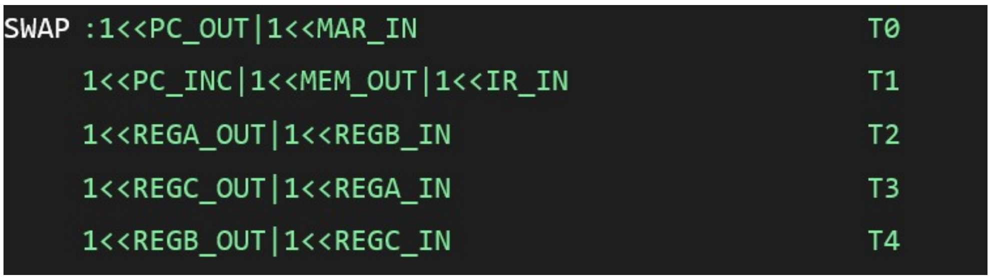
```
OUT  :1<<PC_OUT|1<<MAR_IN                              T0
      1<<PC_INC|1<<MEM_OUT|1<<IR_IN                     T1
      1<<REGA_OUT|1<<REGC_IN                            T2
      0                                                 T3
      0                                                 T4
```

3.9 JNZ



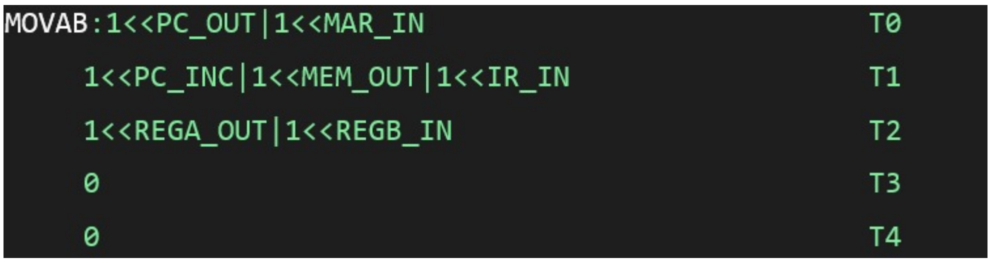
```
JNZ  :1<<PC_OUT|1<<MAR_IN                               T0
      1<<PC_INC|1<<MEM_OUT|1<<IR_IN                     T1
      1<<IR_OUT|1<<PC_LOAD                              T2
      0                                                 T3
      0                                                 T4
```

3.10 SWAP



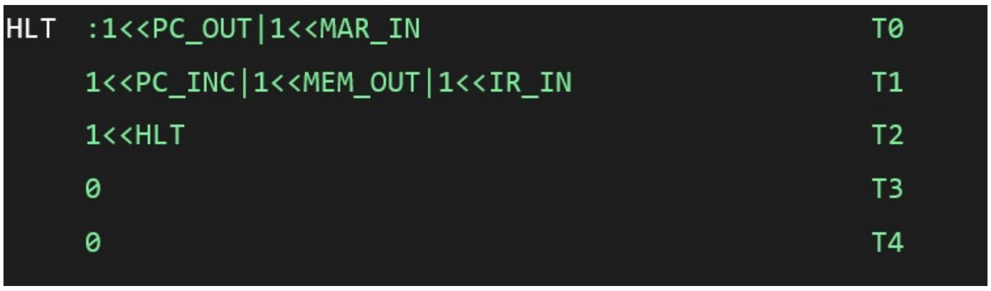
```
SWAP :1<<PC_OUT|1<<MAR_IN                               T0
      1<<PC_INC|1<<MEM_OUT|1<<IR_IN                     T1
      1<<REGA_OUT|1<<REGB_IN                            T2
      1<<REGC_OUT|1<<REGA_IN                            T3
      1<<REGB_OUT|1<<REGC_IN                            T4
```

3.11 MOVAB



```
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
```

3.12 HLT



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

4 Sample Tasks

4.1 Task 1

Given $A = 4$, $B = 3$ and $C = 1$.
Write the assembly code for the following operations:
$A = A + B \longrightarrow$ Display A $\longrightarrow A = A - C \longrightarrow$ Display A
Solution:

```
LDI 4
ADD 3
OUT
SUB 1
OUT
HALT
```

4.2 Task 2

Given $A = 3$, $B = 4$, $C = 9$ and $D = 2$. Here A is stored in location 0x0A, B is stored in location 0x0B, C is stored in location 0x0C and D is stored in location 0x0D. E, F are stored in 0x0E and 0x10. The instructions of the program are stored line by line starting from 0x00.
Write the assembly code for the following operations:
$E = A + B \longrightarrow F = C - D \longrightarrow E = E - F \longrightarrow$ Display E $\longrightarrow$ Repeating the same procedure from step 1.
Solution:

```
LDA A
ADD B
STA E
LDA C
SUB D
STA 10
LDA E
SUB F
STA E
OUT
JMP 0
HALT
```

4.3 Task 3

Given $A = 4$, $B = 3$. Calculate $A \times B$ and give the output.A, B are in 0x0A, 0x0B respectively. Value at 0x0E is 1. 0x0C can be used to store temporary value. Initially 0x0C has 0.

Solution:

```
LDA C
ADD A
STA C
LDA B
SUB E
STA B
JNZ 1
LDA C
OUT
HALT
```

4.4 Task 4

Swap values in memory 0x0A, 0x0B.

Solution:

```
LDA A
MOVAB
LDA B
STA A
SWAPAB
STA B
HALT
```