COMPUTER ARCHITECTURE (ECT-312)

PROJECT - CONJOINT PROCESSOR

<u>Submitted to</u>:

Mr. Rakesh Bairathi

<u>Submitted by</u>:

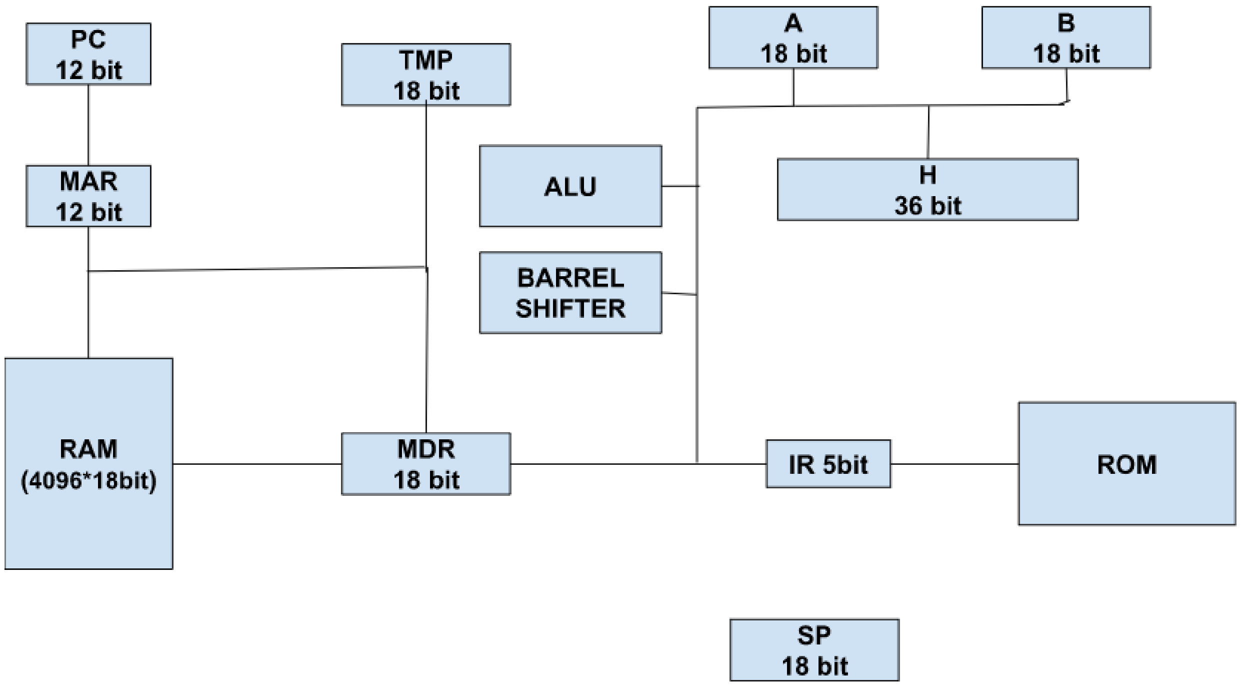
N. Rohith	(2020UEC1672)
V.V Sirisha Malla	(2020UEC1697)
Sanchit Dadhich	(2020UEC1670)
Kartik Rathor	(2020UEC1656)
Y. Dileep Kumar	(2020UEC1762)

Malaviya National Institute of Technology

ELECTRONICS AND COMMUNICATION DEPARTMENT

2022-2023

Processor Architecture Layout:



FLAGS:

N	Z	C	F

INSTRUCTION SET:

I 1 bit	OPCODE 5 bit	ADDRESS 12 bit

Instruction set:

Conjoint processor culminates the MRI (Memory referencing Instructions) Non-MRI instruction and IMMEDIATE instructions sets which can be used as per the user requirements. The below table shows the MRI, Non-MRI and Immediate instructions.

MRI	Non- MRI	IMMEDIATE
ADD	**ADDN**	**ROR**
SUB	**SUBN**	**ROL**
LDA	**MOV**	**ASR**
STA	**SWAP**	**RRX**
	CMP (Compare)	**RLX**
	CL (Clear)	**ADI**
	CM (Complement)	**SBI**
	HLT	**CPI**
	MUL	**MVI**
	DIV	**PUSH**
		POP

Unique features of conjoint processor!

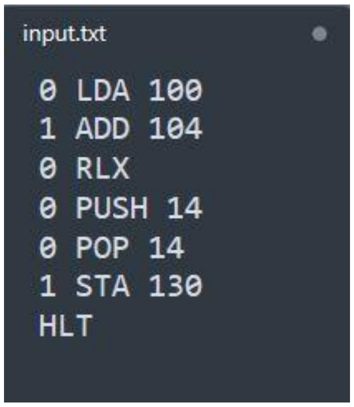
- Usage of OPCODE 31 for Non-MRI instructions.
- Implementing ROR/ROL/ASR/RLX/RRX with the help of Barrel shifter (Hardware).
  - ROR/ROL can be implemented logically using operations in the ALU but providing a hardware for the functions that are frequently used helps to improve the average computation time.
- Additional specification for the registers using the indirect bit in Non-MRI instructions.
  - Non-MRI instruction generally don't use indirect bit. We use it to specify which register is intended for the given operation thereby effectively doubling the number of instruction and providing some direct functionalities.
- Using TMP register for temporary holding data and addresses while other calculations are performed.
  - Generally, TMP is used to store the addresses for processing but conjoint uses it for intermediate storage and processing of data.
- Interrupt Handling: Handling three types of interrupts:
  - Division by zero: In this interrupt we increase the divisor to 1 and divide by 1. In this way we handle the interrupt.
  - Address less than Code Area Size(100): In this interrupt we don't do any operation, but simply move on to next instruction.
  - Rotations > 3: In this interrupt we truncate the rotations by performing the modulo 3operation to bring the number of rotations within the bounds of rotations.

- We are giving the input in the form of the string and we are doing the task of the assembler by taking the string and decoding it to the opcodes and taking the address as an integer value and dealing the instruction and output is also generated as integer value.

Example on Working of Processor:

1.

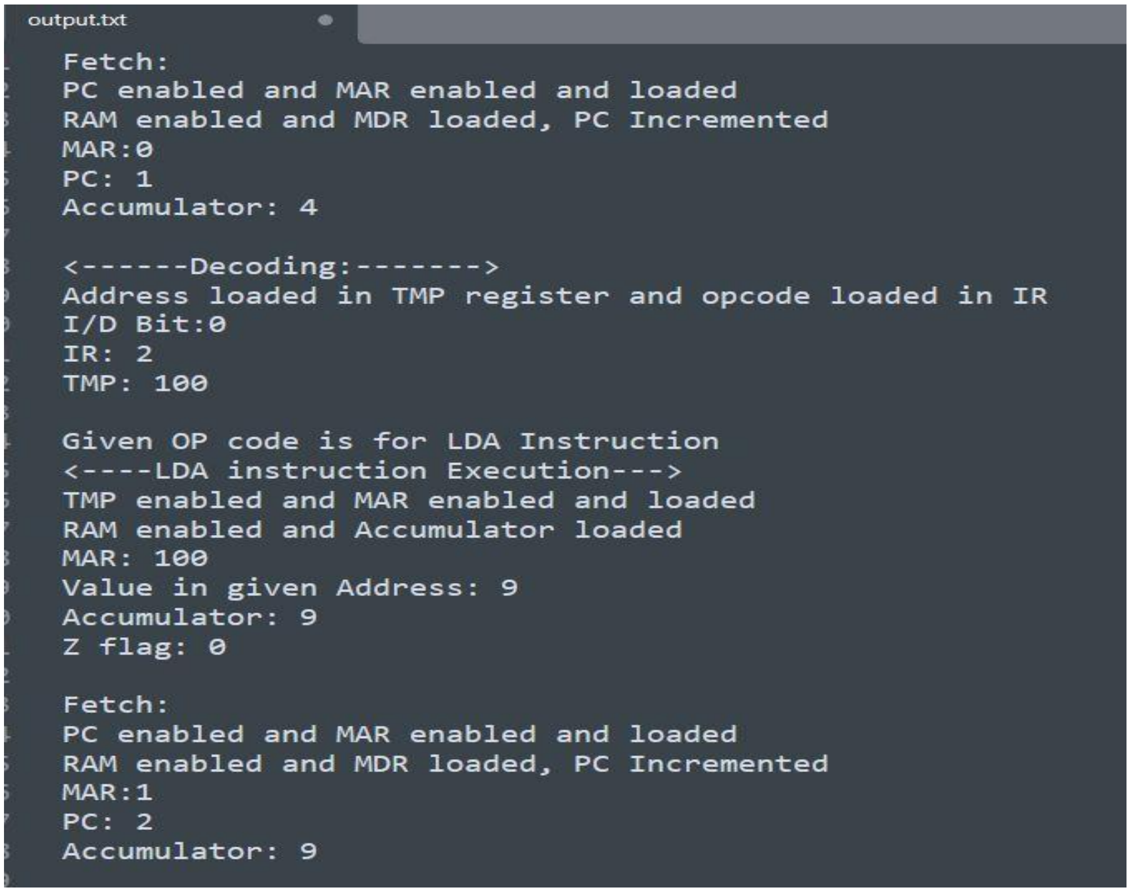
Input:



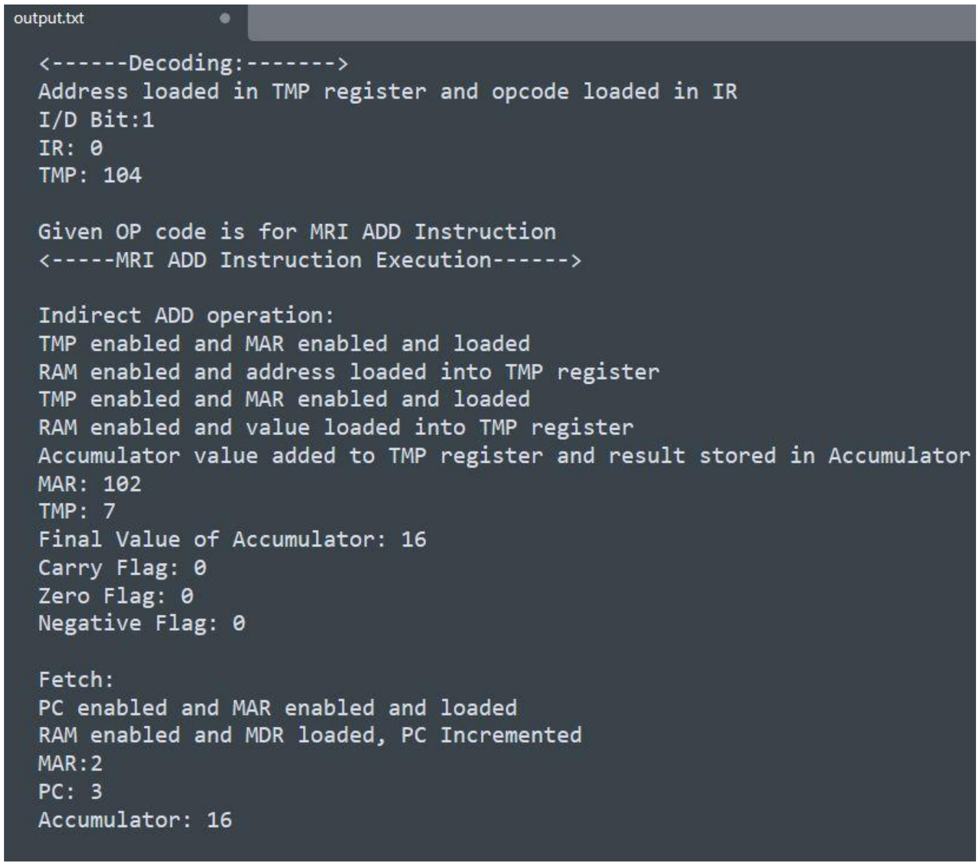
input.txt

```
0 LDA 100
1 ADD 104
0 RLX
0 PUSH 14
0 POP 14
1 STA 130
HLT
```

Output:

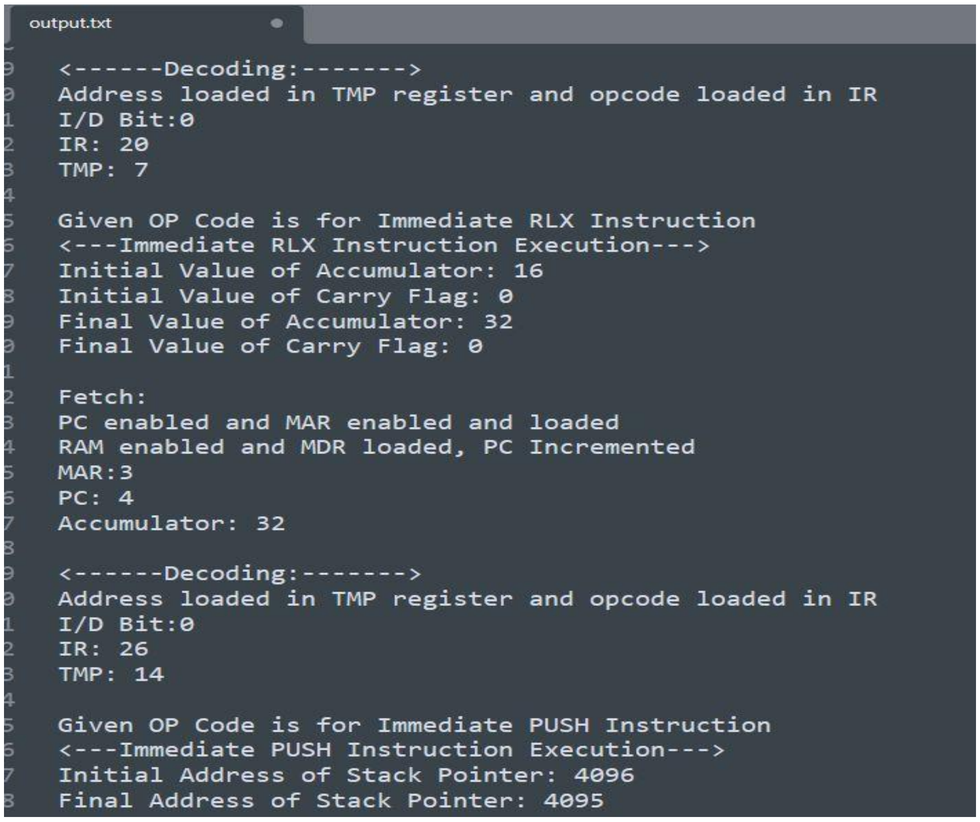

output.txt

```
Fetch:
PC enabled and MAR enabled and loaded
RAM enabled and MDR loaded, PC Incremented
MAR:0
PC: 1
Accumulator: 4

<------Decoding:------->
Address loaded in TMP register and opcode loaded in IR
I/D Bit:0
IR: 2
TMP: 100

Given OP code is for LDA Instruction
<----LDA instruction Execution--->
TMP enabled and MAR enabled and loaded
RAM enabled and Accumulator loaded
MAR: 100
Value in given Address: 9
Accumulator: 9
Z flag: 0

Fetch:
PC enabled and MAR enabled and loaded
RAM enabled and MDR loaded, PC Incremented
MAR:1
PC: 2
Accumulator: 9
```



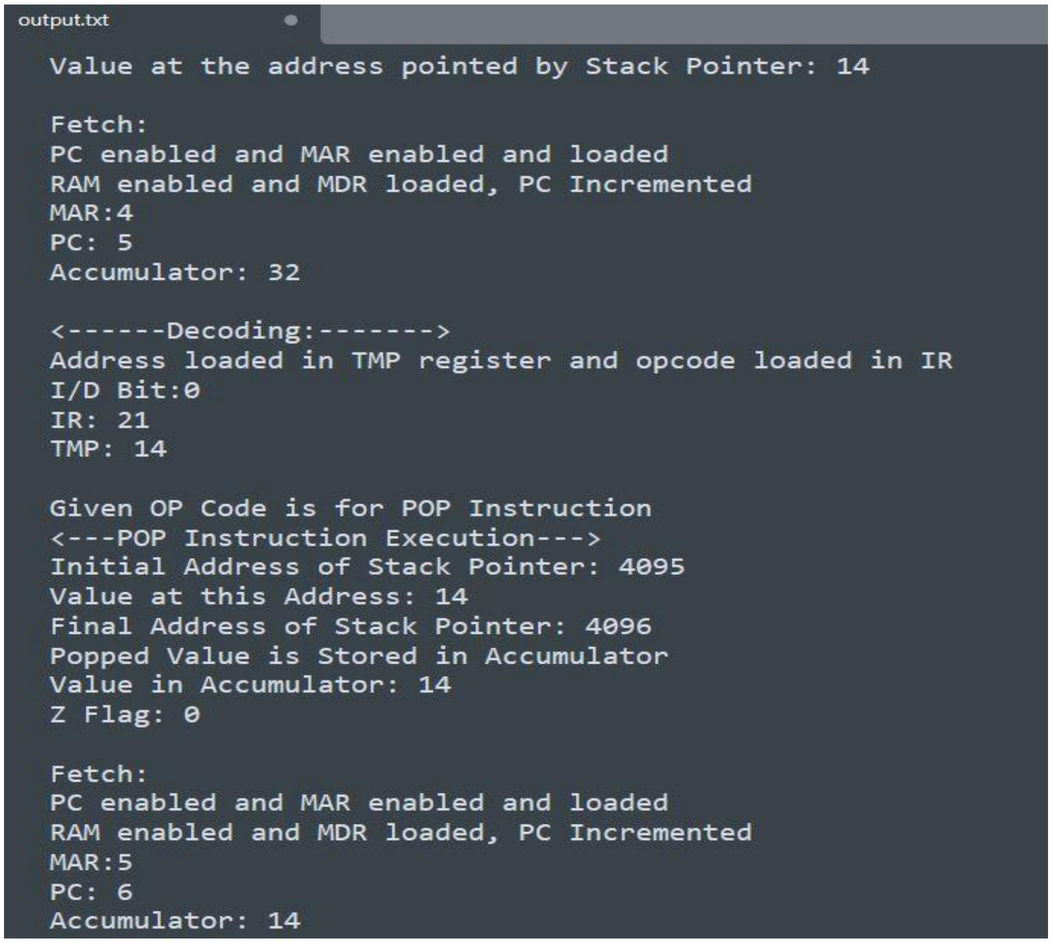
output.txt

```
<------Decoding:------->
Address loaded in TMP register and opcode loaded in IR
I/D Bit:1
IR: 0
TMP: 104

Given OP code is for MRI ADD Instruction
<-----MRI ADD Instruction Execution------>

Indirect ADD operation:
TMP enabled and MAR enabled and loaded
RAM enabled and address loaded into TMP register
TMP enabled and MAR enabled and loaded
RAM enabled and value loaded into TMP register
Accumulator value added to TMP register and result stored in Accumulator
MAR: 102
TMP: 7
Final Value of Accumulator: 16
Carry Flag: 0
Zero Flag: 0
Negative Flag: 0

Fetch:
PC enabled and MAR enabled and loaded
RAM enabled and MDR loaded, PC Incremented
MAR:2
PC: 3
Accumulator: 16
```

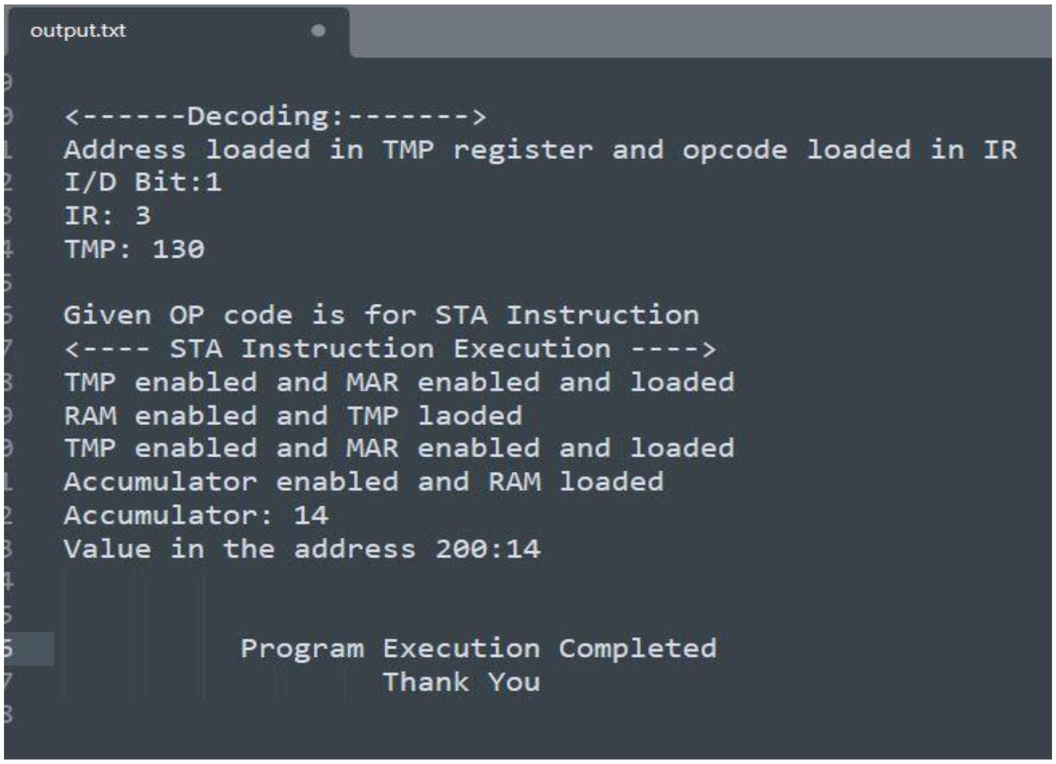


output.txt

```
<------Decoding:------->
Address loaded in TMP register and opcode loaded in IR
I/D Bit:0
IR: 20
TMP: 7

Given OP Code is for Immediate RLX Instruction
<---Immediate RLX Instruction Execution--->
Initial Value of Accumulator: 16
Initial Value of Carry Flag: 0
Final Value of Accumulator: 32
Final Value of Carry Flag: 0

Fetch:
PC enabled and MAR enabled and loaded
RAM enabled and MDR loaded, PC Incremented
MAR:3
PC: 4
Accumulator: 32

<------Decoding:------->
Address loaded in TMP register and opcode loaded in IR
I/D Bit:0
IR: 26
TMP: 14

Given OP Code is for Immediate PUSH Instruction
<---Immediate PUSH Instruction Execution--->
Initial Address of Stack Pointer: 4096
Final Address of Stack Pointer: 4095
```


output.txt

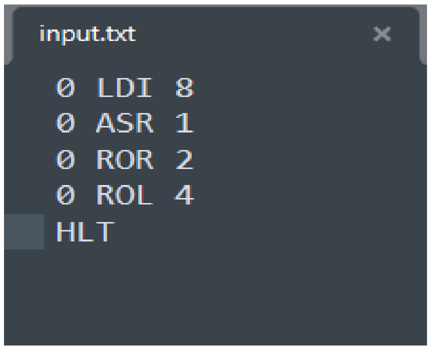
```
Value at the address pointed by Stack Pointer: 14

Fetch:
PC enabled and MAR enabled and loaded
RAM enabled and MDR loaded, PC Incremented
MAR:4
PC: 5
Accumulator: 32

<------Decoding:------->
Address loaded in TMP register and opcode loaded in IR
I/D Bit:0
IR: 21
TMP: 14

Given OP Code is for POP Instruction
<---POP Instruction Execution--->
Initial Address of Stack Pointer: 4095
Value at this Address: 14
Final Address of Stack Pointer: 4096
Popped Value is Stored in Accumulator
Value in Accumulator: 14
Z Flag: 0

Fetch:
PC enabled and MAR enabled and loaded
RAM enabled and MDR loaded, PC Incremented
MAR:5
PC: 6
Accumulator: 14
```


output.txt

```

<------Decoding:------->
Address loaded in TMP register and opcode loaded in IR
I/D Bit:1
IR: 3
TMP: 130

Given OP code is for STA Instruction
<---- STA Instruction Execution ---->
TMP enabled and MAR enabled and loaded
RAM enabled and TMP laoded
TMP enabled and MAR enabled and loaded
Accumulator enabled and RAM loaded
Accumulator: 14
Value in the address 200:14


          Program Execution Completed
                    Thank You
```

2.

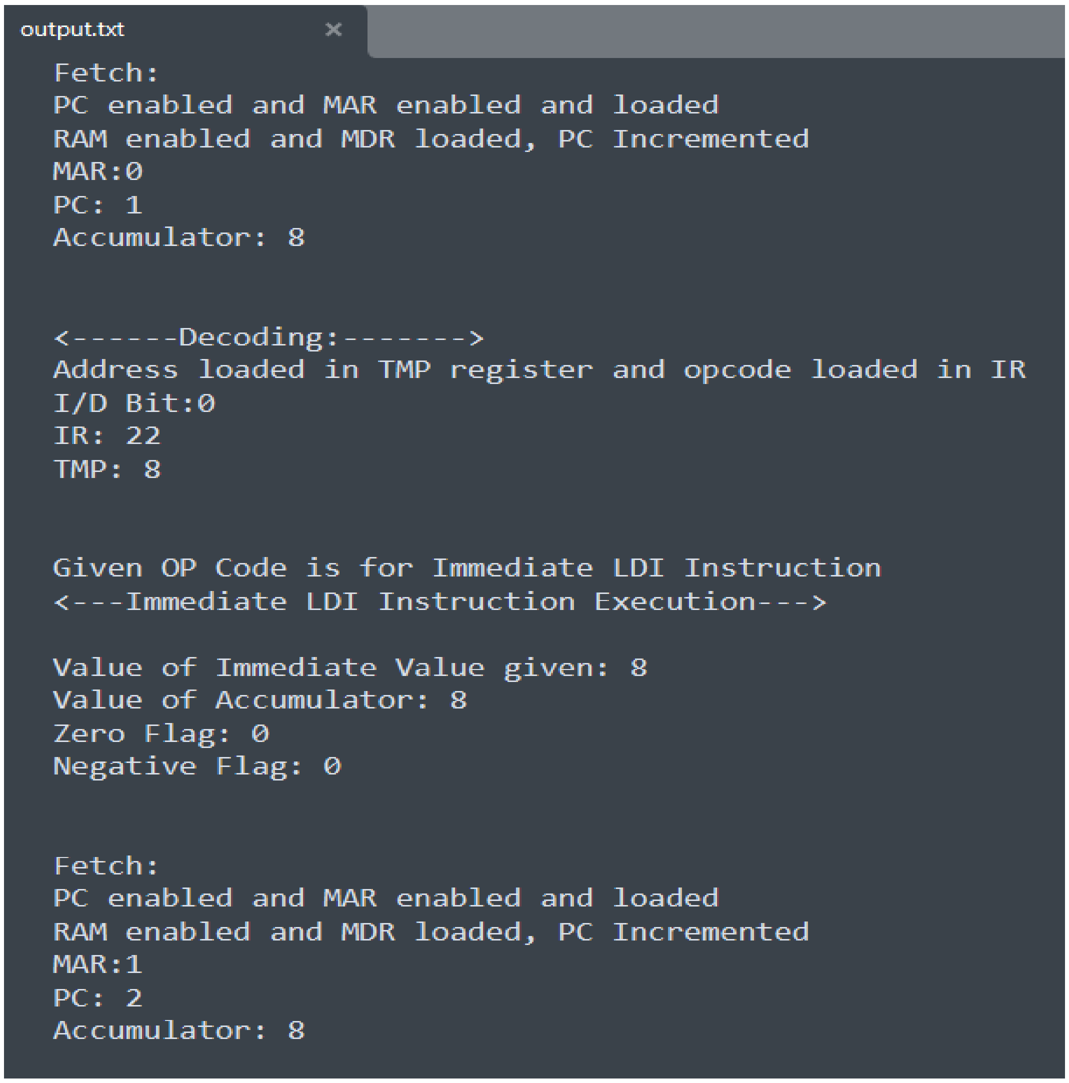
Input:



input.txt

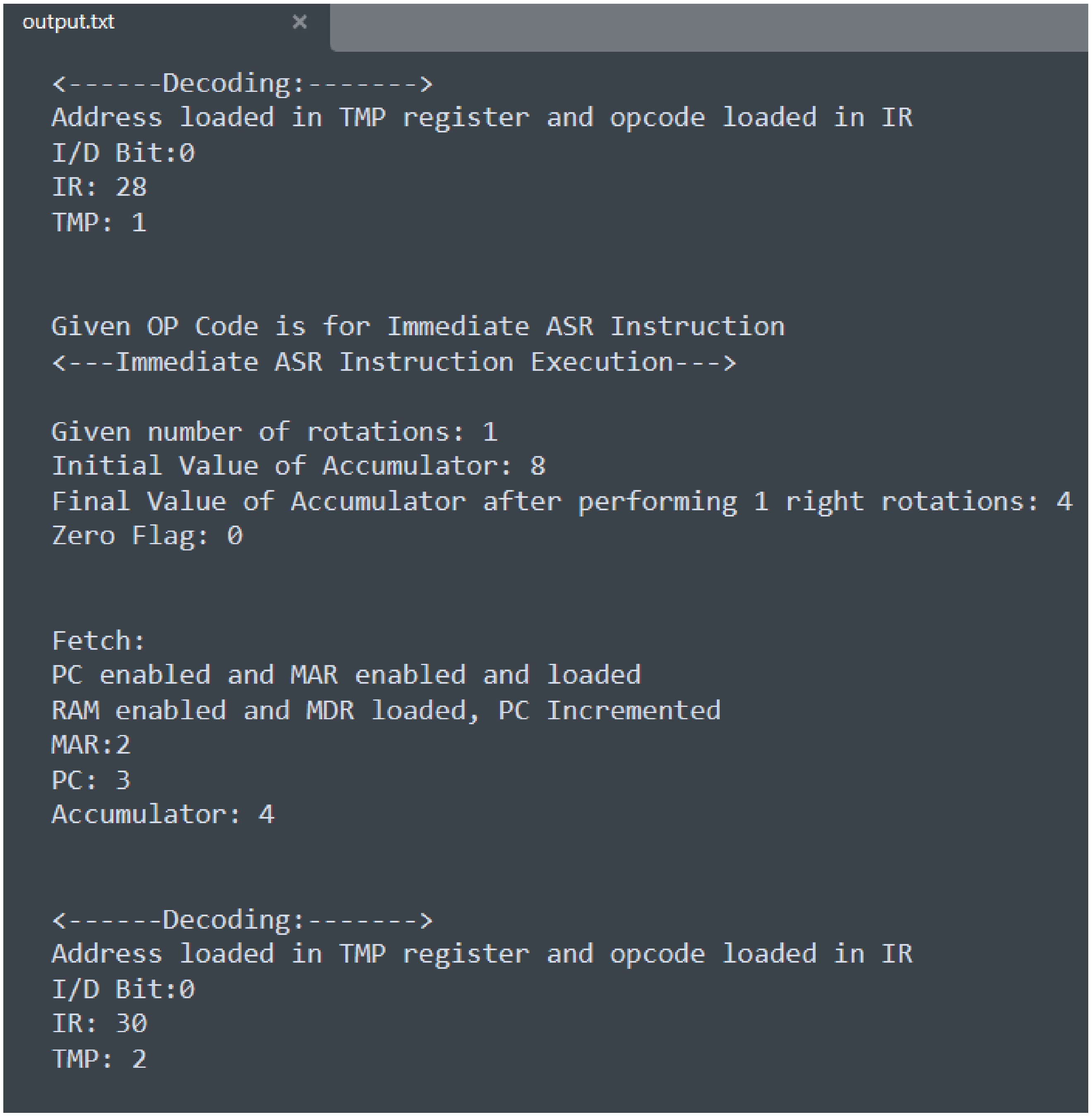
```
0 LDI 8
0 ASR 1
0 ROR 2
0 ROL 4
HLT
```

Output:

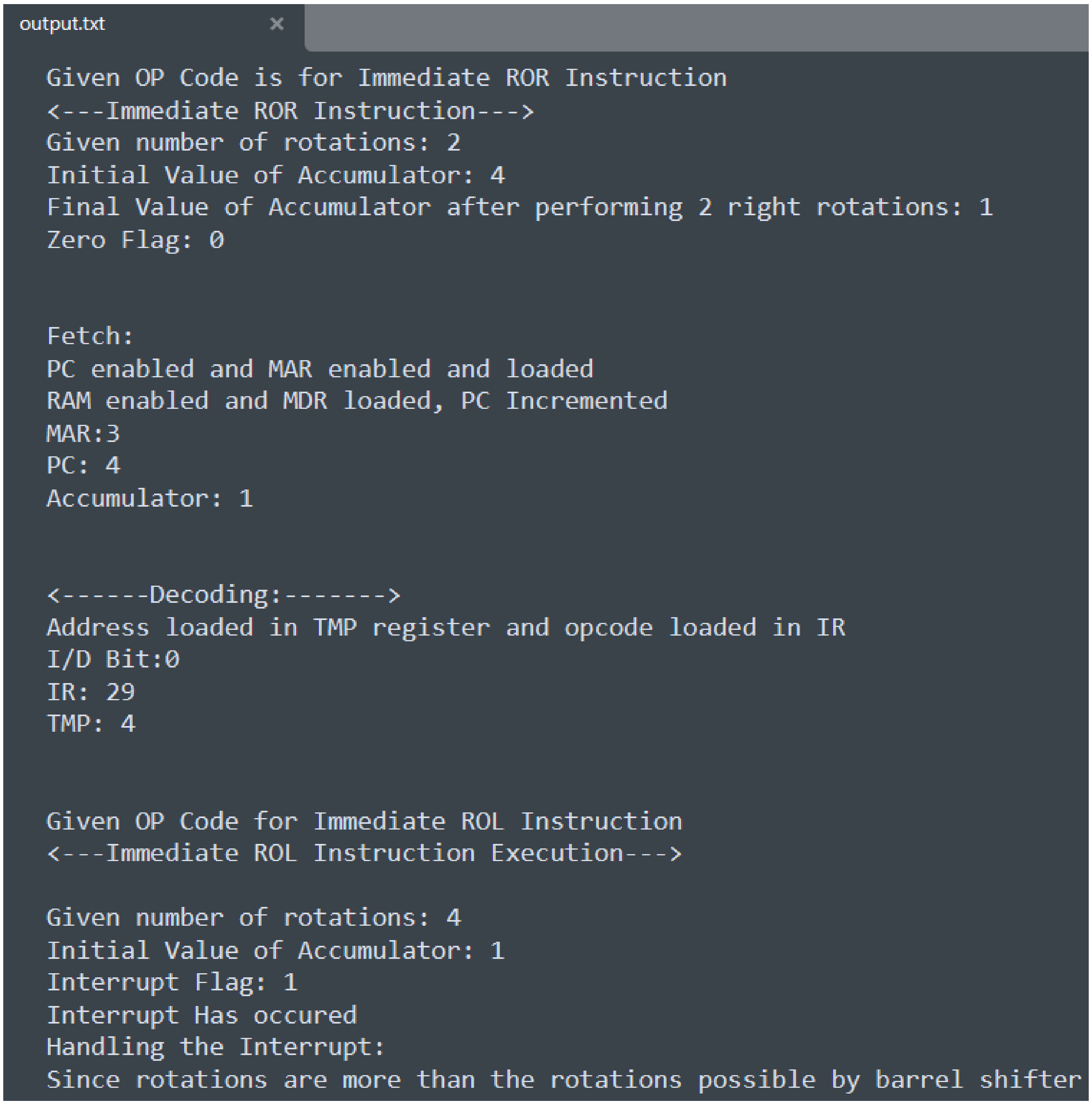


output.txt

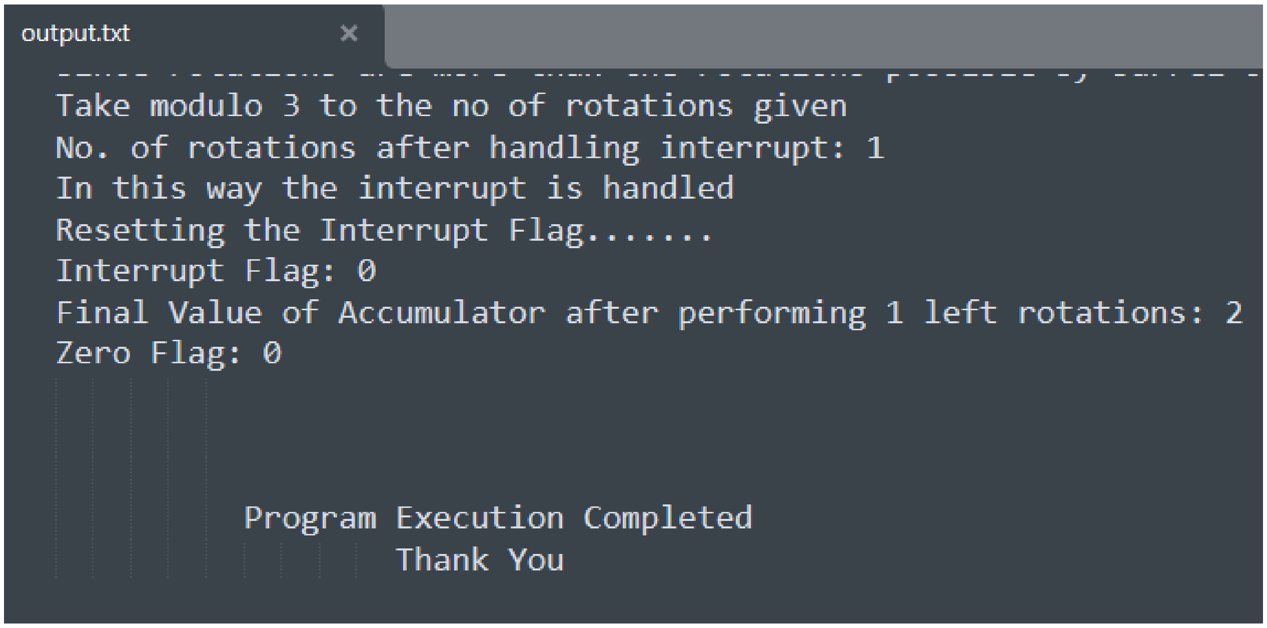
```
Fetch:
PC enabled and MAR enabled and loaded
RAM enabled and MDR loaded, PC Incremented
MAR:0
PC: 1
Accumulator: 8


<------Decoding:------->
Address loaded in TMP register and opcode loaded in IR
I/D Bit:0
IR: 22
TMP: 8


Given OP Code is for Immediate LDI Instruction
<---Immediate LDI Instruction Execution--->

Value of Immediate Value given: 8
Value of Accumulator: 8
Zero Flag: 0
Negative Flag: 0


Fetch:
PC enabled and MAR enabled and loaded
RAM enabled and MDR loaded, PC Incremented
MAR:1
PC: 2
Accumulator: 8
```


output.txt

```
<------Decoding:------->
Address loaded in TMP register and opcode loaded in IR
I/D Bit:0
IR: 28
TMP: 1


Given OP Code is for Immediate ASR Instruction
<---Immediate ASR Instruction Execution--->

Given number of rotations: 1
Initial Value of Accumulator: 8
Final Value of Accumulator after performing 1 right rotations: 4
Zero Flag: 0


Fetch:
PC enabled and MAR enabled and loaded
RAM enabled and MDR loaded, PC Incremented
MAR:2
PC: 3
Accumulator: 4


<------Decoding:------->
Address loaded in TMP register and opcode loaded in IR
I/D Bit:0
IR: 30
TMP: 2
```



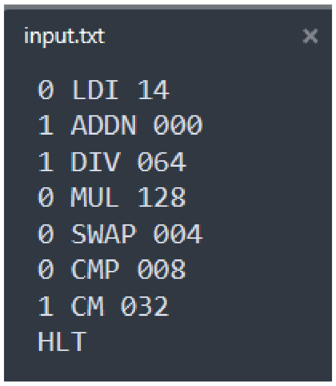
```
Given OP Code is for Immediate ROR Instruction
<---Immediate ROR Instruction--->
Given number of rotations: 2
Initial Value of Accumulator: 4
Final Value of Accumulator after performing 2 right rotations: 1
Zero Flag: 0


Fetch:
PC enabled and MAR enabled and loaded
RAM enabled and MDR loaded, PC Incremented
MAR:3
PC: 4
Accumulator: 1


<------Decoding:------->
Address loaded in TMP register and opcode loaded in IR
I/D Bit:0
IR: 29
TMP: 4


Given OP Code for Immediate ROL Instruction
<---Immediate ROL Instruction Execution--->

Given number of rotations: 4
Initial Value of Accumulator: 1
Interrupt Flag: 1
Interrupt Has occured
Handling the Interrupt:
Since rotations are more than the rotations possible by barrel shifter
```



```
[illegible]
Take modulo 3 to the no of rotations given
No. of rotations after handling interrupt: 1
In this way the interrupt is handled
Resetting the Interrupt Flag.......
Interrupt Flag: 0
Final Value of Accumulator after performing 1 left rotations: 2
Zero Flag: 0



          Program Execution Completed
                  Thank You
```

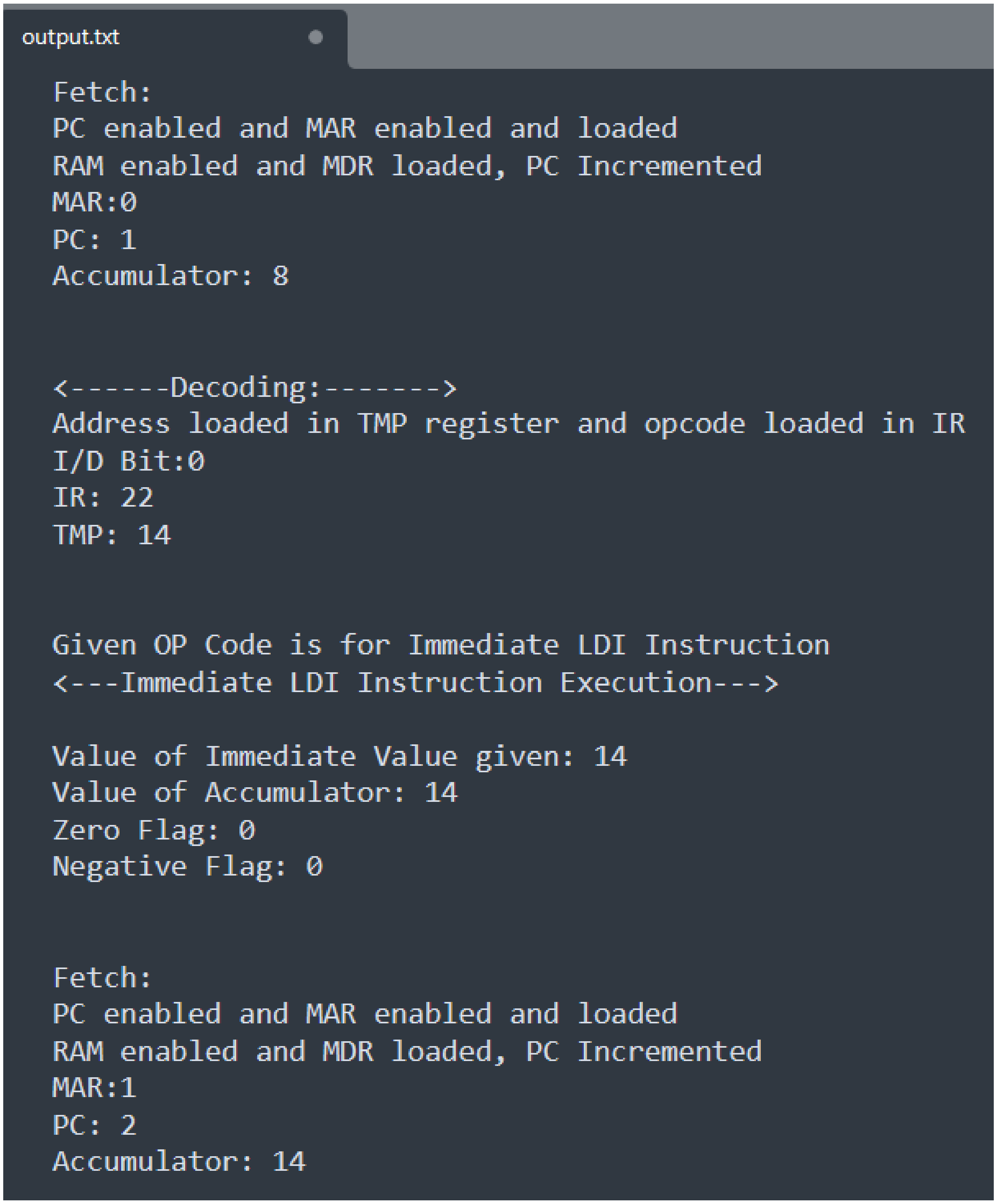
3.

Input:

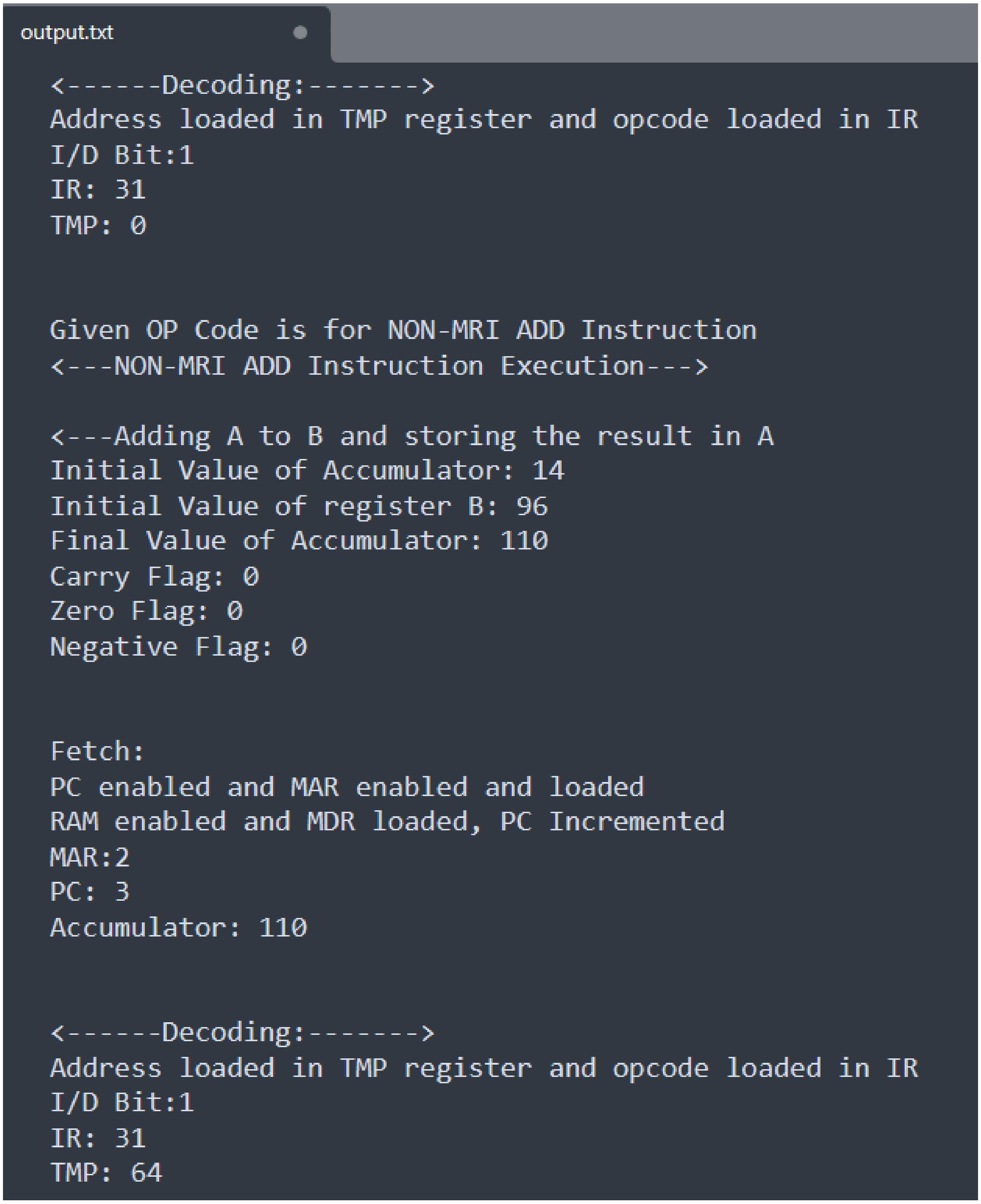


```
0 LDI 14
1 ADDN 000
1 DIV 064
0 MUL 128
0 SWAP 004
0 CMP 008
1 CM 032
HLT
```

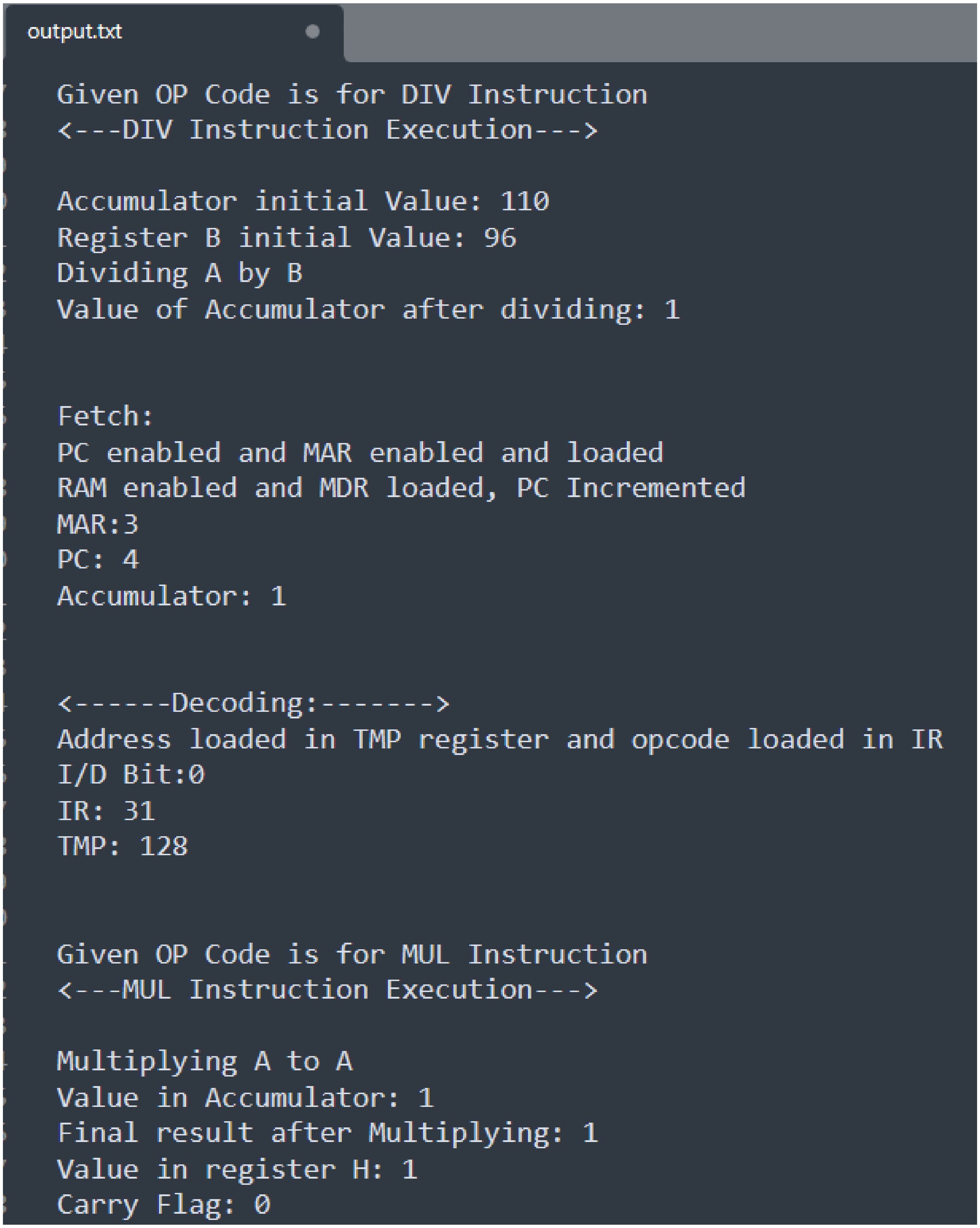
Output:


output.txt

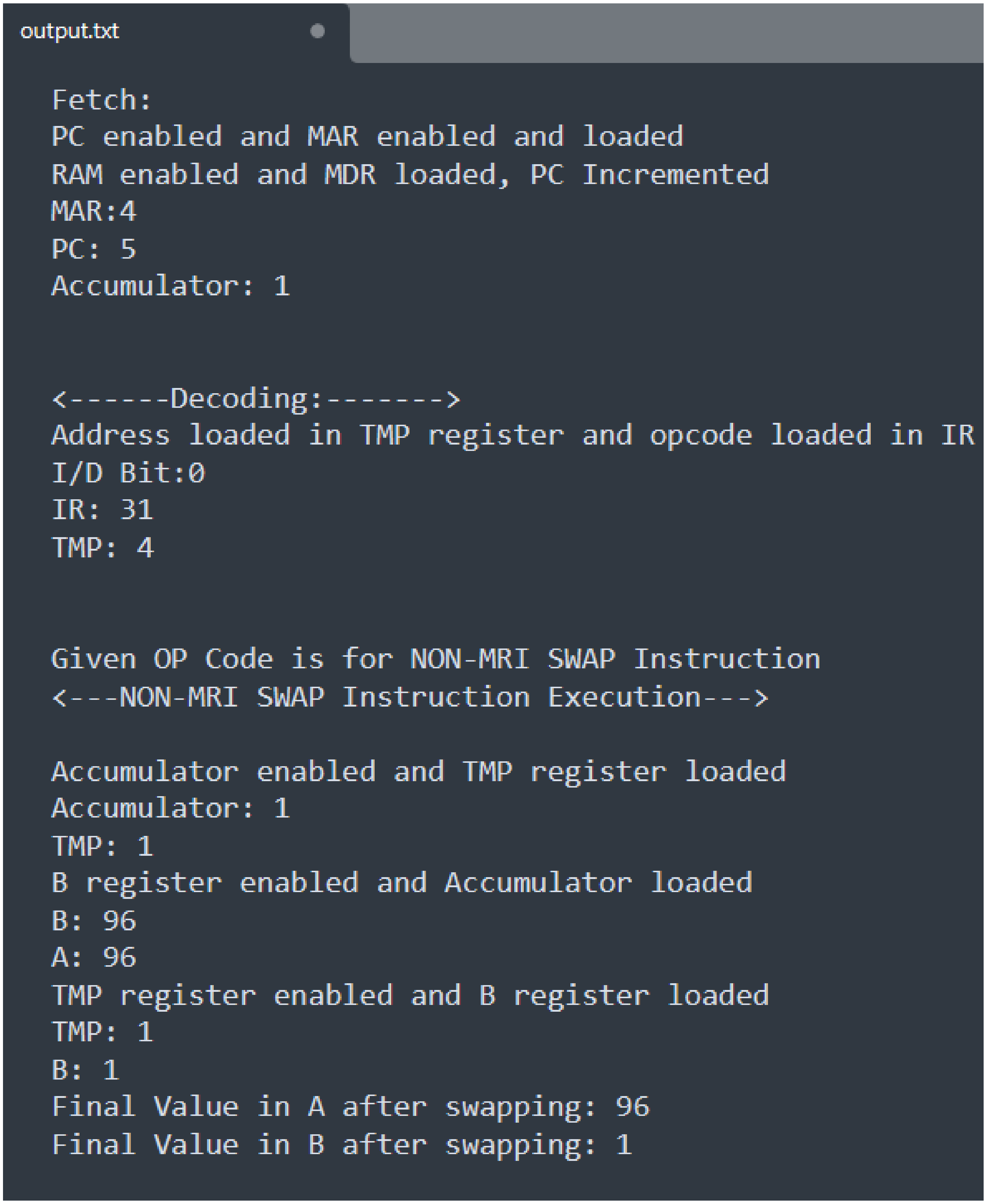
```
Fetch:
PC enabled and MAR enabled and loaded
RAM enabled and MDR loaded, PC Incremented
MAR:0
PC: 1
Accumulator: 8


<------Decoding:------->
Address loaded in TMP register and opcode loaded in IR
I/D Bit:0
IR: 22
TMP: 14


Given OP Code is for Immediate LDI Instruction
<---Immediate LDI Instruction Execution--->

Value of Immediate Value given: 14
Value of Accumulator: 14
Zero Flag: 0
Negative Flag: 0


Fetch:
PC enabled and MAR enabled and loaded
RAM enabled and MDR loaded, PC Incremented
MAR:1
PC: 2
Accumulator: 14
```


output.txt

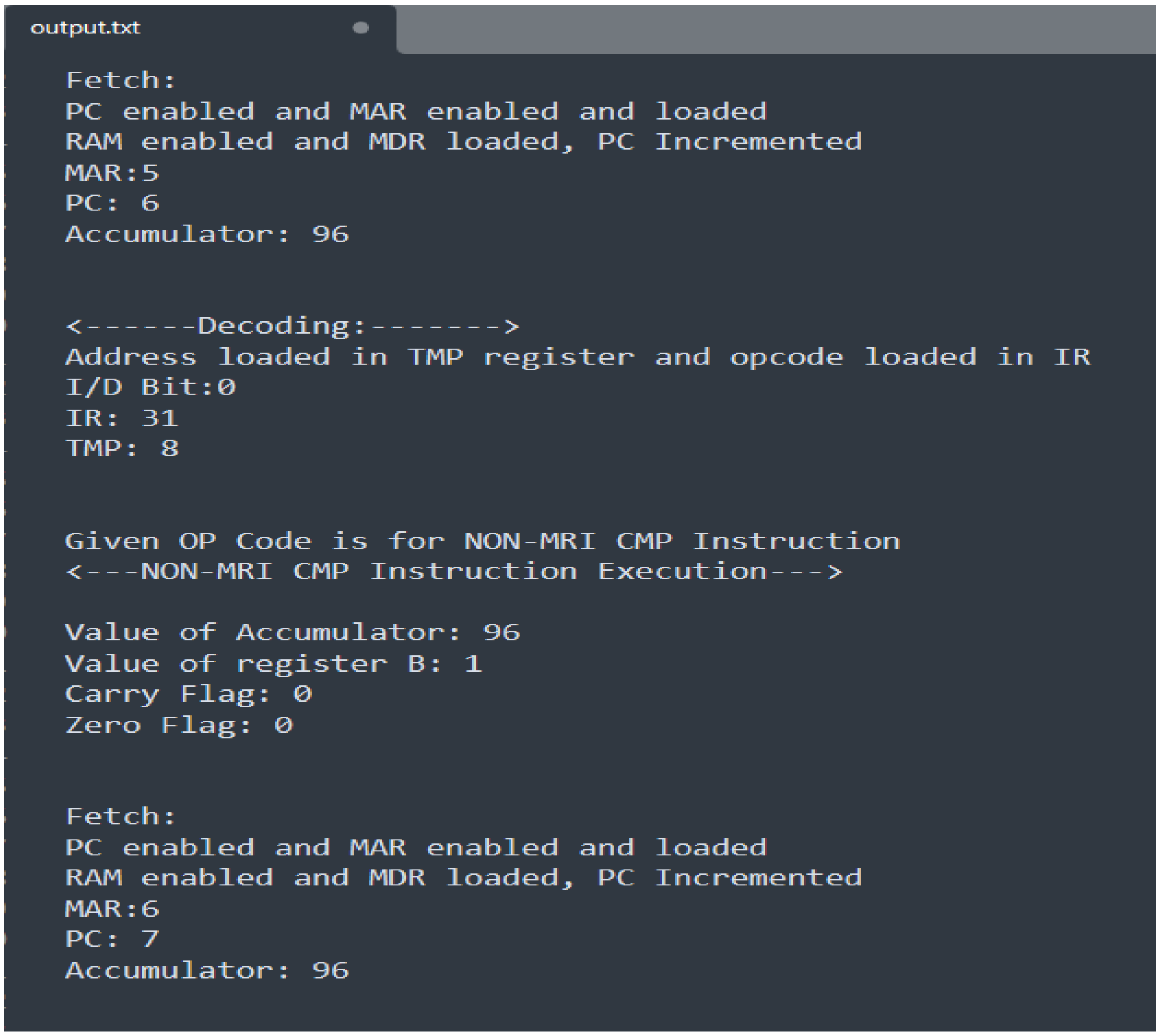
```
<------Decoding:------->
Address loaded in TMP register and opcode loaded in IR
I/D Bit:1
IR: 31
TMP: 0


Given OP Code is for NON-MRI ADD Instruction
<---NON-MRI ADD Instruction Execution--->

<---Adding A to B and storing the result in A
Initial Value of Accumulator: 14
Initial Value of register B: 96
Final Value of Accumulator: 110
Carry Flag: 0
Zero Flag: 0
Negative Flag: 0


Fetch:
PC enabled and MAR enabled and loaded
RAM enabled and MDR loaded, PC Incremented
MAR:2
PC: 3
Accumulator: 110


<------Decoding:------->
Address loaded in TMP register and opcode loaded in IR
I/D Bit:1
IR: 31
TMP: 64
```



```
Given OP Code is for DIV Instruction
<---DIV Instruction Execution--->

Accumulator initial Value: 110
Register B initial Value: 96
Dividing A by B
Value of Accumulator after dividing: 1


Fetch:
PC enabled and MAR enabled and loaded
RAM enabled and MDR loaded, PC Incremented
MAR:3
PC: 4
Accumulator: 1


<------Decoding:------->
Address loaded in TMP register and opcode loaded in IR
I/D Bit:0
IR: 31
TMP: 128


Given OP Code is for MUL Instruction
<---MUL Instruction Execution--->

Multiplying A to A
Value in Accumulator: 1
Final result after Multiplying: 1
Value in register H: 1
Carry Flag: 0
```

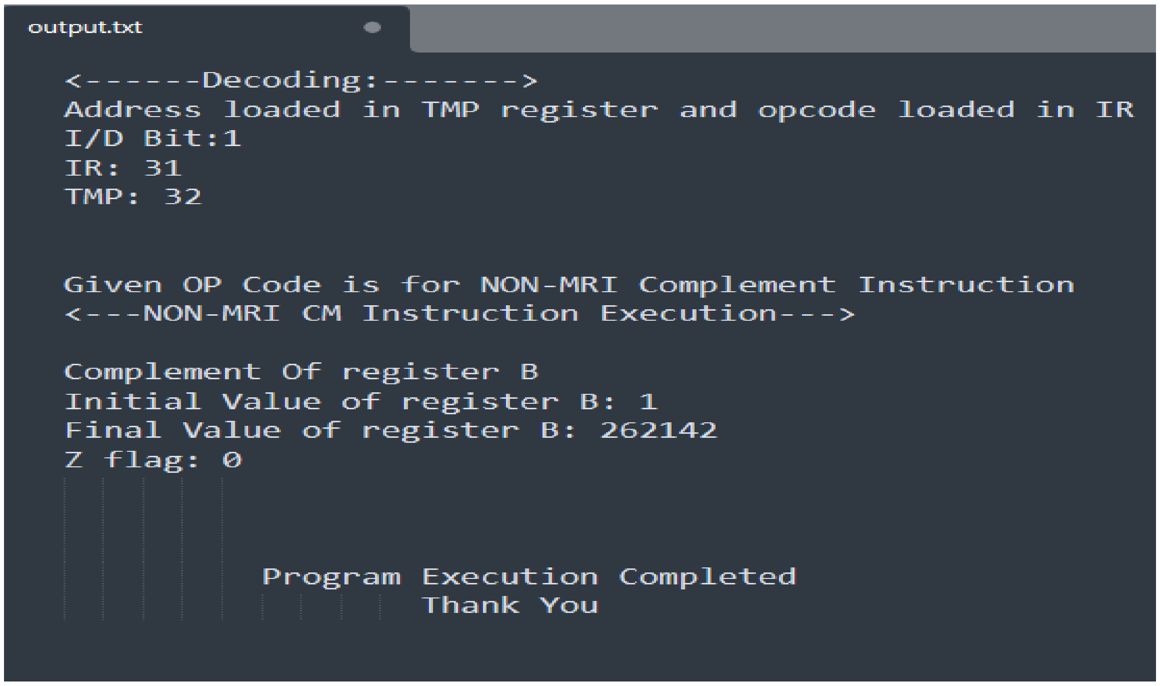


```
Fetch:
PC enabled and MAR enabled and loaded
RAM enabled and MDR loaded, PC Incremented
MAR:4
PC: 5
Accumulator: 1


<------Decoding:------->
Address loaded in TMP register and opcode loaded in IR
I/D Bit:0
IR: 31
TMP: 4


Given OP Code is for NON-MRI SWAP Instruction
<---NON-MRI SWAP Instruction Execution--->

Accumulator enabled and TMP register loaded
Accumulator: 1
TMP: 1
B register enabled and Accumulator loaded
B: 96
A: 96
TMP register enabled and B register loaded
TMP: 1
B: 1
Final Value in A after swapping: 96
Final Value in B after swapping: 1
```



output.txt

```
Fetch:
PC enabled and MAR enabled and loaded
RAM enabled and MDR loaded, PC Incremented
MAR:5
PC: 6
Accumulator: 96


<------Decoding:------->
Address loaded in TMP register and opcode loaded in IR
I/D Bit:0
IR: 31
TMP: 8


Given OP Code is for NON-MRI CMP Instruction
<---NON-MRI CMP Instruction Execution--->

Value of Accumulator: 96
Value of register B: 1
Carry Flag: 0
Zero Flag: 0


Fetch:
PC enabled and MAR enabled and loaded
RAM enabled and MDR loaded, PC Incremented
MAR:6
PC: 7
Accumulator: 96
```



output.txt

```
<------Decoding:------->
Address loaded in TMP register and opcode loaded in IR
I/D Bit:1
IR: 31
TMP: 32


Given OP Code is for NON-MRI Complement Instruction
<---NON-MRI CM Instruction Execution--->

Complement Of register B
Initial Value of register B: 1
Final Value of register B: 262142
Z flag: 0



          Program Execution Completed
                  Thank You
```

Non-MRI:

Non-MRI (Non-Memory referencing instructions) is the instruction set that do not refer the contents of the memory. These instructions are used to perform operations on registers or to define the functionality or behavior of the processor.

For Non-MRI: (OPCODE = 11111 (31))

I = 0 => Doing operation with A
I = 1 => Doing operation with B
I bit targets the register on which the function is to be performed.

➢ ADDN:

Used for the addition of values present in register A/B.

12-bit address: 0000 0000 0000
I = 0 => 0 ADDN 000 => A = A + A
I = 1 => 1 ADDN 000 => A = A + B

MICRO INSTRUCTIONS:

Case 1: A = A+A

T0: PC(E), MAR (L, E)
T1: RAM(E), MDR(L)

T2: PC(I), MDR(E), IR(L), I(L), TMP(L)
T3: ALU(L), A(E)
T4: ALU(E), A(L), R

Case 2: A = A+B

T0: PC(E) MAR (L, E)
T1:RAM(E) MDR(L)
T2: PC(I) MDR(E) IR(L) I(L) TMP(L) //Differentiated by the I value
T3: ALU(L), A(E)
T4: ALU(E), A(L), R

➢ SUBN:

Used for the subtraction of values present in register A/B.

12-bit address: 0000 0000 0001
I = 0 => 0 SUBN 001 => A = B- A
I = 1 => 1 SUBN 001 => A = A – B

MICRO INSTRUCTIONS:

Case 1: A = B - A

T0: PC(E), MAR (L, E)
T1: RAM(E), MDR(L)
T2: PC(I), MDR(E), R(L), I(L), TMP(L)
T3: ALU(L), A(E), B(E)
T4: ALU(E), A(L), R

Case 2: A = A - B

T0: PC(E), MAR (L, E)
T1: RAM(E), MDR(L)
T2: PC(I), MDR(E), IR(L), I(L), TMP(L)
T3: ALU(L), A(E), B(E)
T4: ALU(E), A(L), R

➢ MOV:

Used to move values form register A to register B or vice versa.

12-bit address: 0000 0000 0010
I = 0 => 0 MOV 002 => A <= B
I = 1 => 1 MOV 002 => B <= A

MICRO INSTRUCTIONS:

Case 1: B to A

T0: PC(E) MAR (L, E)
T1: RAM(E) MDR(L)
T2: PC(I) MDR(E) IR(L) I(L) TMP(L)
T3: B(E) A(L), R

Case:2 A to B

T0: PC(E), MAR (L, E)
T1: RAM(E), MDR(L)
T2: PC(I), MDR(E), IR(L), I(L), TMP(L)
T3: A(E), B(L), R

➢ SWAP:

Used to exchange the contents of register A and register B.

12-bit address: 0000 0000 0100

I = 0(1) => 0(1) SWAP 004 => A ⇔ B

MICRO INSTRUCTIONS:

T0: PC(E), MAR (E, L)
T1: RAM(E), MDR(L)
T2: PC(I), MDR(E), I(L), IR(L), TMP(L)
T3: A(E), TMP(L)
T4: B(E), A(L)
T5: TMP(E), B(L), R

➢ CMP:

Used to compare the values present in register A and register B.

12-bit address: 0000 0000 1000

Set Carry and Zero Flags to zero before running this instruction

I = 0(1) => 0(1) CMP 008 => A < B C = 1
A = B Z = 1

MICRO INSTRUCTIONS:

T0: PC(E), MAR (L, E)
T1: RAM(E), MDR(L)
T2: PC(I), MDR(E), IR(L), I(L), TMP(L)
T3: A(E), B(E), ALU(L)

T4: ALU(E), R

Automatically the flags are affected and using the flag value we decide whether greater or equal cases.

➢ CL:

Used to clear the values of register A and register B and assigning 0 to them.

12-bit address: 0000 0001 0000

I = 0   =>   0 CL 016   =>   A <= 0

I = 1   =>   1 CL 016   =>   B <= 0

MICRO INSTRUCTIONS:

Case 1: A <= 0

T0: PC(E), MAR (L, E)
T1: RAM(E), MDR(L)
T2: PC(I), MDR(E), IR(L), I(L), TMP(L)
T3: ALU(L), A(E)
T4: ALU(E), A(L), R

Case 2: B <= 0

T0: PC(E), MAR (L, E)
T1: RAM(E), MDR(L)
T2: PC(I), MDR(E), IR(L), I(L), TMP(L)
T3: ALU(L), B(E)
T4: ALU(E), B(L), R

➢ CM:

Used to compliment the value present in register A or register B.

12 bit address: 0000 0010 0000

I = 0 => 0 CM 032 => A <= ~A

I = 1 => 1 CM 032 => B <= ~B

MICRO INSTRUCTIONS:

Case 1: A <= ~A

T0: PC(E), MAR (L, E)
T1:RAM(E), MDR(L)
T2: PC(I), MDR(E), IR(L), I(L), TMP(L)
T3: ALU(L), A(E)
T4: ALU(E), A(L), R

Case 2: B <= ~B

T0: PC(E), MAR (L, E)
T1:RAM(E), MDR(L)
T2: PC(I), MDR(E), IR(L), I(L), TMP(L)
T3: ALU(L), B(E)
T4: ALU(E), B(L), R

➢ HLT:

Used to mark the end of the code and the seizure of clock.

MICRO INSTRUCTIONS:

T0: PC(E), MAR (L, E)
T1: RAM(E), MDR(L)
T2: PC(I) MDR(E), IR(L), I(L), TMP(L)

➢ MUL:

Used to Multiply the values present in register A/B and store the result in register H.

12-bit address: 0000 0100 0000

I=0 => 0 MUL 64 => H=A*A

I=1 => 1 MUL 64 => H=A*B

MICRO INSTRUCTIONS:

(I = 0) H <= A*A

T0: PC(E), MAR (E, L)
T1: RAM(E), MDR(L)
T2: MDR(E), IR(L), TMP(L), PC(I), I(L)
T3: A(E), ALU(L)
T4: ALU(E), H(L), <u>R</u>

(I = 1) H <= A*B
T0: PC(E), MAR (E, L)
T1: RAM(E), MDR(L)
T2: MDR(E), IR(L), TMP(L), PC(I), I(L)
T3: A(E), B(E), ALU(L)
T4: ALU(E), H(L) <u>R</u>

➢ DIV

Used to Multiply the values present in register A/B and store the result in register H.

12-bit address: 0000 1000 0000

I=0 => 0 DIV 128 => A=(B/A)

I=1 => 1 DIV 128 => A=(A/B)

Direct: 0 A<=(A/A)=1
Checking Flag :(Z==1) **(division by zero interrupt)**

MICRO INSTRUCTIONS:

I = 0 A=(B/A)
T0: PC(E), MAR (E, L)
T1: RAM(E), MDR(L)
T2: MDR(E), IR(L), TMP(L), PC(I), I(L)
//Checking for A: if Z==1 //division by zero interrupt//
T3: A(E), B(E), ALU(L)
T4: ALU(E), A(L), R //Check Flag//

I = 1 A<=A/B
T0: PC(E), MAR (E, L)
T1: RAM(E), MDR(L)
T2: MDR(E), IR(L), TMP(L), PC(I), I(L)
//Checking for B: Move A to TMP and B to A if Z==1 //division by zero interrupt//
T3: A(E), TMP(L)
T4: B(E), A(L) //Check Flag//
T5: TMP(E), A(L)
T6: A(E), B(E), ALU(L)
T7: ALU(E), A(L), R

MRI:

MRI (Memory referencing instructions) is the instruction set that assists the user to alter, move and perform operations on the values present at the memory locations.

This instruction set can be used for the given two addressing modes:-

I. Direct addressing
II. Indirect addressing

- Direct Addressing mode lets the user to access the memory location given with the instruction. This treats the value at the address as the value needed by the user.
- Indirect Addressing mode treats the value at the address provided by the user as an address. Processor then accesses this address location to provide the required value.

- **ADD:**

Instruction to add values present in RAM/Registers.

Direct:

MICRO INSTRUCTIONS:

T0: PC(E), MAR (L, E)

T1: RAM(E), MDR(L)
T2: PC(I), MDR(E), IR(L), TMP(L), I(L)
// IR<=MDR (12-16), TMP<=MDR (0-11)
T3: TMP(E), MAR (L, E)
T4: RAM(E), TMP(L)
//do the add operation and store it again back into the accumulator //
// A=A+TMP
T5: ALU(L), A(E), TMP(E)
T6: ALU(E), A(L), R

Indirect:

MICRO INSTRUCTIONS:

T0: PC(E), MAR (L, E)
T1: RAM(E), MDR(L)
T2: PC(I). MDR(E), IR(L), TMP(L) I(L)
T3: TMP(E) MAR (L, E)
T4: RAM(E) TMP(L)
T5: TMP(E), MAR (L, E)
T6: RAM(E) TMP(L)
T5: ALU(L), A(E), TMP(E)
T6: ALU(E), A(L), R
//do the add operation and store it again back into the accumulator
Note://This code can also be used for the direct only thing is that T5 and T6 are idle

- **SUB:**

Instruction to subtract values present in RAM/Registers.

Direct:

T0: PC(E), MAR (L, E)
T1: RAM(E), MDR(L)
T2: PC(I), MDR(E), IR(L), TMP(L), I(L)
T3: TMP(E), MAR (L, E)
T4: RAM(E), TMP(L)
T5: ALU(L), A(E)
T6: ALU(E), A(L), R
//do the Sub operation and store it again back into the accumulator

Indirect:

T0: PC(E), MAR (L, E)
T1: RAM(E), MDR(L)
T2: PC(I), MDR(E), IR(L), TMP(L), I(L)
T3: TMP(E), MAR (L, E)
T4: RAM(E), TMP(L)
T5: TMP(E), MAR (L, E)
T6: RAM(E), TMP(L)
T5: ALU(L), A(E)
T6: ALU(E), A(L) R
//do the sub operation and store it again back into the accumulator

➢ LDA:

Instruction to load register A with the value at the given address (Direct)

Direct:

T0: PC(E), MAR (E, L)
T1: RAM(E), MDR(L)
T2: MDR(E), IR(L), TMP(L), PC(I)

T3: TMP(E), MAR (E, L)
T4: RAM(E), A(L), R

ii. Indirect:

T0: PC(E), MAR (E, L)
T1: RAM(E), MDR(L)
T2: MDR(E), IR(L), TMP(L), PC(I)
T3: TMP(E), MAR (E, L)
T4: RAM(E), TMP(L)
T5: TMP(E), MAR (E, L)
T6: RAM(E), A(L) R

- **STA:**

Instruction to store the value from register A to the given memory location.

Direct:

T0: PC(E), MAR (E, L)
T1: RAM(E), MDR(L)
T2: MDR(E), IR(L), TMP(L), PC(I)
T3: TMP(E), MAR (E, L)
T4: RAM(L), A(E) R

Indirect:

T0: PC(E), MAR (E, L)
T1: RAM(E), MDR(L)
T2: MDR(E), IR(L), TMP(L), PC(I)
T3: TMP(E), MAR (E, L)

T4: RAM(E), TMP(L)
T5: TMP(E), MAR (E, L)
T6: RAM(L), A(E) R

IMMEDIATE :

Immediate instructions are the instruction set that do not alter the contents of the memory. These instructions are used to perform operations on registers or to define the functionality or behavior of the processor

➢ ROR:

T0: PC(E), MAR (L, E)
T1: RAM(E), MDR(L)
T2: MDR(E), IR(L), TMP(L), I(L), PC(I)

//input tmp (no of times of rotation) and Accumulator value to the barrel shifter and note the result in the accumulator again//

➢ ROL:

T0: PC(E), MAR (L, E)
T1: RAM(E), MDR(L)
T2: MDR(E), IR(L), TMP(L), I(L), PC(I)

//input tmp (no of times of rotation) and A value to the barrel shifter and note the result in the accumulator again//

➢ ASR:

T0: PC(E), MAR (L, E)
T1: RAM(E), MDR(L)
T2: PC(I), MDR(E), IR(L), TMP(L)

T3: TMP(E), ALU(L)
T4: ALU(E), A(L), R
// In ALU we use the 4X4 Barrel Shifter and do the operation in the arithmetic way by not changing the sign bit //

➢ RRX with carry

T0: PC(E) MAR (L, E)

T1:RAM(E), MDR(L)

T2: MDR(E), IR(L), TMP(L), I(L), PC(I)

//input carry, A and tmp value which has one time to rotate to the barrel shifter and note the answer in the accumulator //

➢ PUSH:

T0: PC(E), MAR (L, E)
T1: RAM(E), MDR(L)
T2: PC(I), MDR(E), IR(L), TMP(L)
T3: SP(D) //Using the Mechanism of the decrement before
T4: SP(E), MAR (L, E) //Stack Pointer always point to the highest filled location
T5: TMP(E), RAM(L)

➢ ADI:

T0: PC(E), MAR (L, E)
T1: RAM(E), MDR(L)
T2: PC(I), MDR(E), IR(L), TMP(L)
T3: TMP(E), A(E), ALU(L)
T4: ALU(E), A(L)
//The immediate value is given to the ALU along with the accumulator value the values are added and again stored in the accumulator//

➢ SBI:

T0: PC(E), MAR (L, E)
T1: RAM(E), MDR(L)

T2: PC(I), MDR(E), IR(L), TMP(L)
T3: TMP(E), A(E), ALU(L)
T4: ALU(E), A(L)

//The immediate value is given to the ALU along with the accumulator value the values are subtracted and again stored in the accumulator//

➢ RLX

T0: PC(E), MAR (L, E)

T1: RAM(E), MNR(L)

T2: MDR(E), IR(L), TMP(L), I(L), PC(I)

//input carry, A and tmp value which has how many times to rotate to the barrel shifter and note the answer in the accumulator //

➢ RLX With carry

T0: PC(E), MAR (L, E)
T1: RAM(E), MDR(L)
T2: MDR(E), IR(L), TMP(L), PC(I)
T3: TMP(E), A(E), ALU(L) //By default the maximum rotation is 1 time
T4: ALU(E), A(L), R

//input carry, A and tmp value which has how many times to rotate to the barrel shifter and note the answer in the accumulator //

➢ LDI

T0: PC(E), MAR (L, E)
T1: RAM(E), MDR(L)
T2: MDR(E), IR(L), TMP(L), PC(I)
T3: TMP(E) , A(L), R

➢ CPI

T0: PC(E), MAR (L, E)
T1: RAM(E), MDR(L)
T2: MDR(E), IR(L), TMP(L), PC(I)
T3: ALU(E), R

A < B	**C = 1**
A = B	**Z = 1**

➢ POP

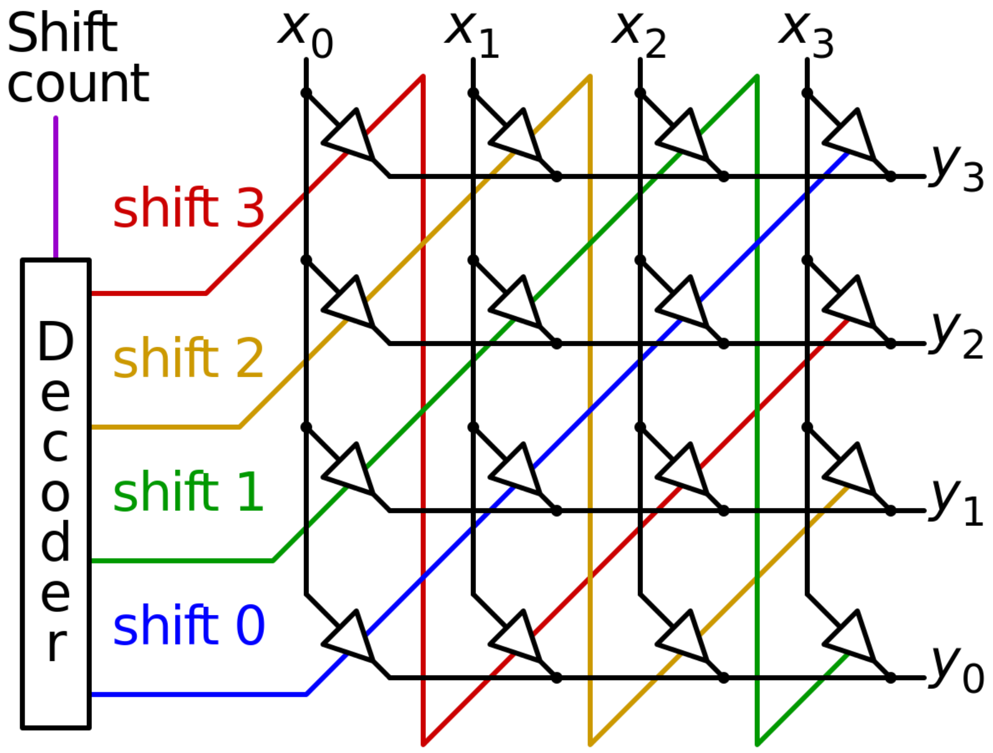
T0: PC(E), MAR (L, E)
T1: RAM(E), MDR(L)
T2: MDR(E), IR(L), TMP(L), PC(I)
T3: SP(E), A(L)
T4: SP(I), R

Barrel Shifter:

A Barrel shifter is a specialized digital electronic circuit with the purpose of shifting an entire data word by a specified number of bits by only using combinational logic, with no sequential logic used. The simplest way of achieving this is by using a series of multiplexers where one output is connected to the input of the next multiplexer in the chain, in a specific manner that depends on the amount of shift specified.

A better circuit with the pass transistors connected in group of four as shown in figure in which the control line requirements is decreased from 16 to 4 separate control lines S0 – S3. To derive a 2-to-4 decoder circuit, a particular control line might be chosen by encoding a 2-bit control field. The output of individual decoder would provide the suitable shift control line. 1024 pass transistors are required for a 32-bit data path to permit the desired shift operations. In only parallel shifts 32 control lines chosen by a 5-bit encoded control field are enough.

A barrel shifter is often used to shift and rotate n-bits in modern microprocessors, typically within a single clock cycle.



For example, take a four-bit barrel shifter, with inputs A, B, C and D. The shifter can cycle the order of the bits ABCD as DABC, CDAB, or BCDA; in this case, no bits are lost. That is, it can shift all of the outputs up to three positions to the right (and thus make any cyclic combination of A, B, C and D). The barrel shifter has a variety of applications, including being a useful component in microprocessors (alongside the ALU).

Interrupt Handling:

An interrupt is an event that alters the sequence in which the processor executes instructions. In our CONJOINT processor, we handle three types of interrupts.

➢ Address < Code Area:

The first 100 locations in the RAM are allocated to the code area. If the user tries to access any address between the addresses 0 - FF, we generate an interrupt i.e.; we set the interrupt flag **F=1.** We service the interrupt and after that we reset the flag **F=0.**

We handle this interrupt using the following methodology:
In this interrupt, we move on to next instruction by skipping the current instruction.

E.g.:
```
LDA 9
ADD 102
```
The address 9 is within the code area. So, we do not execute the instruction LDA and move on to the instruction ADD and execute it. In this way we handle the interrupt

➢ Division by Zero:

We cannot divide the any number by zero. So, whenever any divisor is zero, we generate an interrupt i.e.; we set the interrupt flag **F=1.** We service the interrupt and after that we reset the flag **F=0.**

We handle this interrupt using the following methodology:
Whenever the divisor is zero, we increment the divisor to 1 and perform the division operation. In this way we service the interrupt and continue our division operation.

- ## Rotations > 3:

  The Barrel shifter hardware that we use can rotate/shift up to maximum 3 shifts/rotations. If the user gives rotations greater than 3, we generate an interrupt i.e.; we set the interrupt flag **F=1.** We service the interrupt and after that we reset the flag **F=0.**

  We handle this interrupt using the following methodology:
  We do the modulo 3 operation to the number of rotations given by the user to bring the rotations/shift count within the range of Max. rotations/shift possible by the barrel shifter. Then we do the rotations/shifts. In this way we service the interrupt and return to main execution.

  E.g.: ROR 5

  5 modulo 3 = 2

  ROR 2

OPCODES:

MRI:

- ADD – 0
- SUB – 1
- LDA – 2
- STA – 3

NON-MRI:

- ADDN – 31 (0)
- SUBN – 31 (1)
- MOV – 31 (2)
- SWAP – 31 (4)
- CMP – 31 (8)
- CL – 31 (16)
- CM – 31 (32)
- DIV – 31 (64)
- MUL – 31 (128)
- HLT – 18

IMMEDIATE:

- ROR – 30
- ROL – 29
- ASR – 28
- RRX – 27
- PUSH – 26
- ADI – 25
- SBI – 24
- CPI – 23
- LDI – 22
- POP – 21
- RLX – 20

CONCLUSION:

Our Processor deals with instruction set of size 18-bit using RAM 4096X18bit. In one go we decode the single instruction as I bit, Opcode and 12-bit for address. The MSB is considered as I bit, which signifies the type of addressing whether direct or indirect while dealing with MRI instructions. In case of NON-MRI, we use the same I bit for handling different combinations of Registers. The next 5 bit is used for opcode which is used to identify the type of instruction we are executing. Then the last 12-bit is used for address handling while dealing with MRI instructions and in the case of NON-MRI instructions it signifies the type of instructions we are dealing and in the case of immediate these 12 bits take a 12-bit immediate value and we execute the instruction.

Types of Instructions we are dealing using this Processor:

- Memory Referencing Instructions (MRI)
- Non- Memory Referencing Instructions (NON-MRI)
- Immediate Instructions

Our Processor deals with three Interrupts:

- Address < Code Area
- Division by zero
- Rotation value >3

Flags dealt in the Processor are N,Z,C,F

N: To signify the negative integer value

Z: To signify the zero value of the accumulator

C: To signify the carry bit while dealing the arithmetic instructions.

F: To signify the interrupt.