COMPUTER ARCHITECTURE
(ECT-312)

PROCESSOR PROJECT - EDITH

Submitted to:

Mr. Rakesh Bairathi

Submitted by:

Himanshu Omar	(2020UEC1341)
Gungun Singhal	(2020UEC1372)
Ashutosh Mishra	(2020UEC2016)
Khanra Akash Ananta	(2020UEC1901)
Guntuku Abhilash	(2020UEC2006)

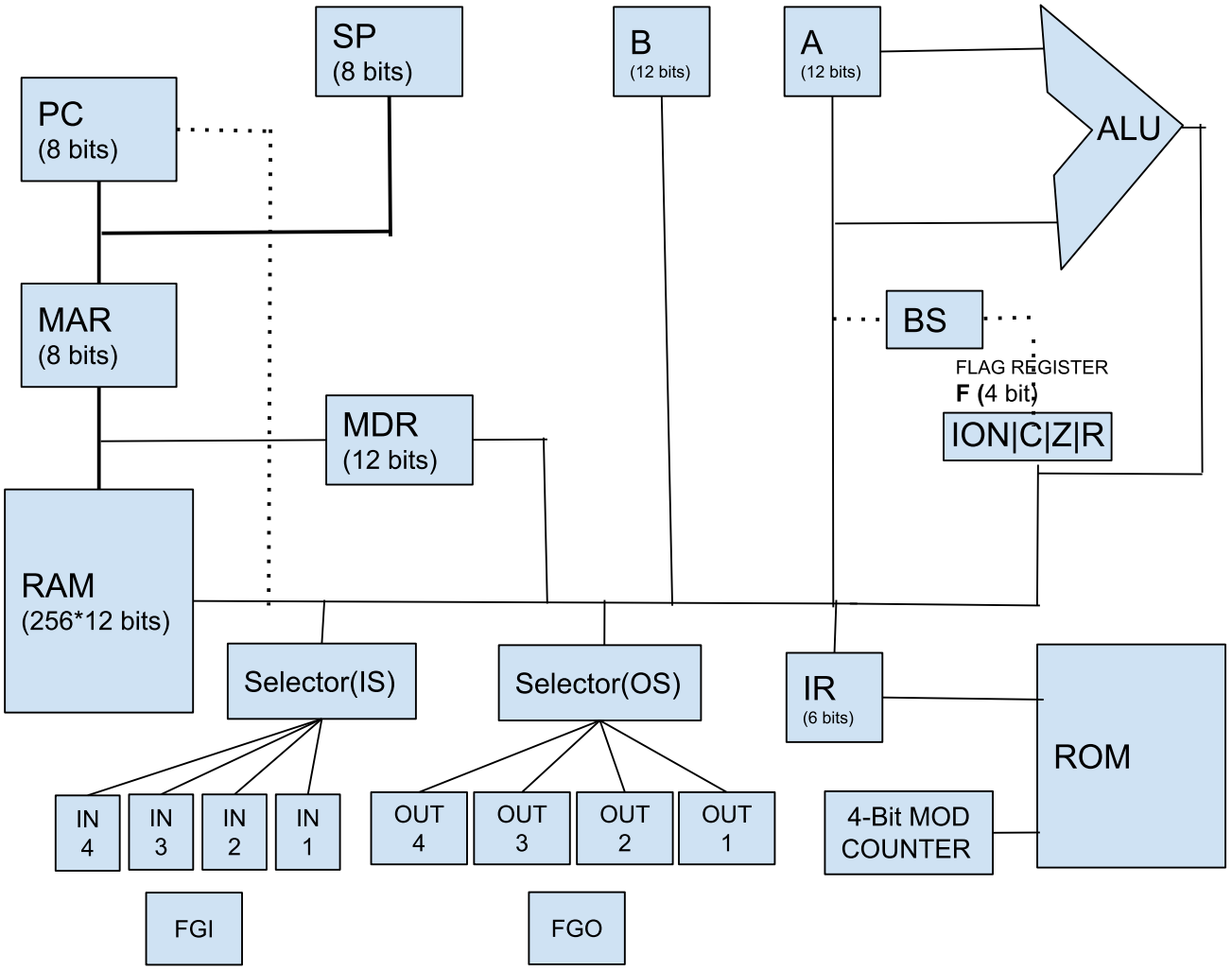
Malaviya National Institute of Technology

ELECTRONICS AND COMMUNICATION DEPARTMENT (2022-2023)

Table of Contents

Sr no.	Contents	Page no.
1.	Processor Architecture Layout	3
2.	Components Description	4
3.	Unique Features	7
4.	Instruction Set	8-9
	i.) MRI	8
	ii.) Non-MRI	9
5.	Micro-Instructions	10-21
	i.) MRI	10
	ii.) Non-MRI	15
6.	Sample Outputs	22
7.	Interrupt Handling	41
8.	Hardware Implementation	44
9.	Conclusion	46

Processor Architecture Layout



REGISTER DESCRIPTION

1. MEMORY DATA REGISTER (12-Bit) :

Instructions from RAM are fetched and transferred into this Register for further decoding and then sent to the IR register.

Format:

➢ <u>For MRI Instruction:</u>

1-Bit	3-Bit	8-Bit
I (Direct or Indirect)	MRI Opcode	Address

➢ <u>For non-MRI Instruction:</u>

1-Bit	3-Bit	4-Bit	4-Bit
I (Register or I/O based)	111	Non-MRI Opcode	Data/Don't care

2. FLAG REGISTER (4-Bit) :

Format:

4-Bit

ION	C	Z	R

- ***ION flag***:- ION is reset means processor will be handling interrupt, ION is set means processor will not be handling any interrupt.
- ***Carry flag:***- C is set and reset on every instruction based on Arithmetic and Logical Operations on Accumulator.
- ***Zero flag:***- Z is set and reset on instruction based on Arithmetic and Logical Operations on Accumulator when it becomes equal to zero.
- ***R flag:***- R is set when an interrupt has appeared and to be serviced, R is reset when the processor has finished handling that interrupt.

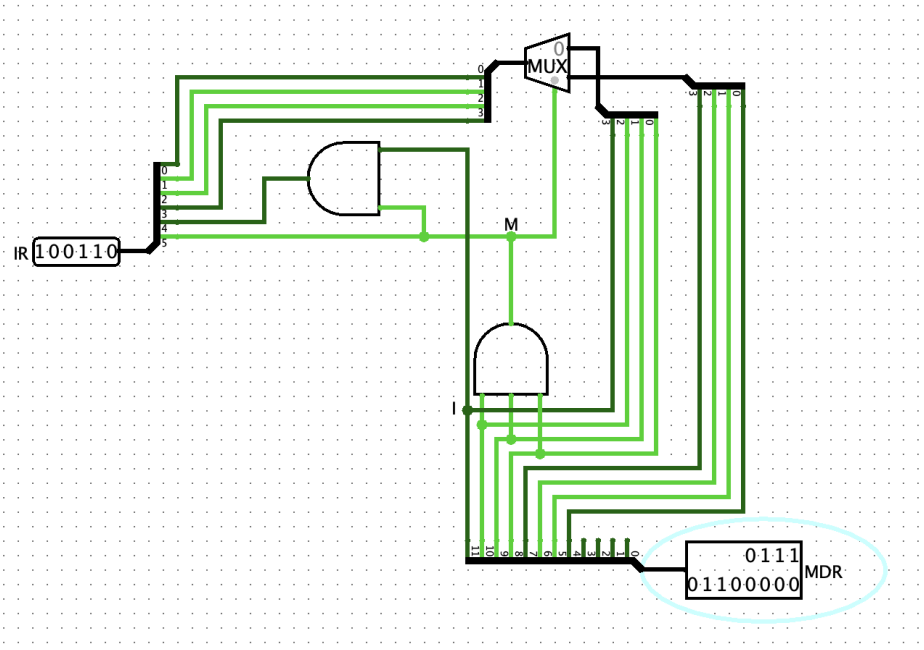
3. INSTRUCTION REGISTER (6-Bit) :

Format:

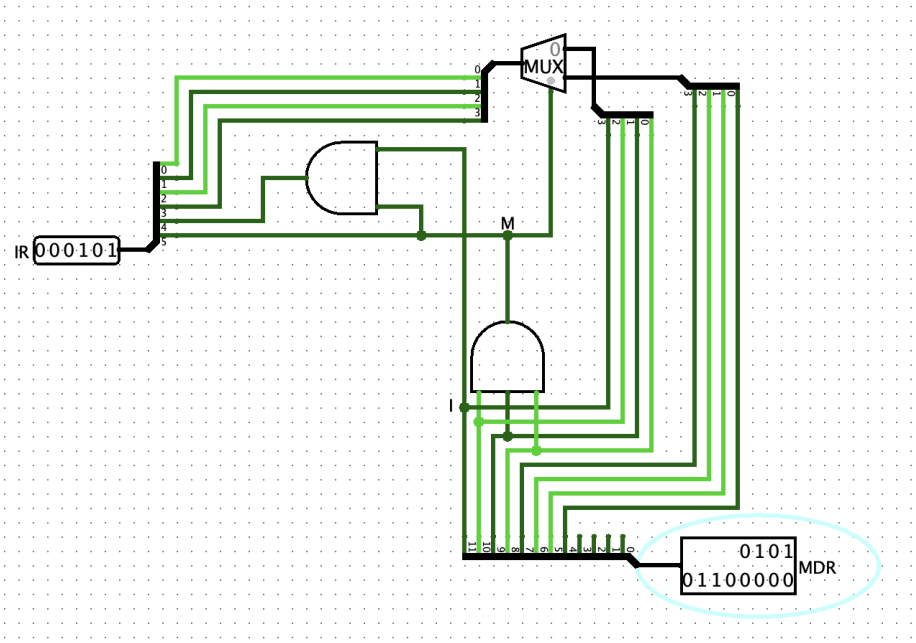
1-Bit	1-Bit	4-Bit
M (MRI/Non-MRI)	I/O (set for I/O operation)	Opcode(based on M)

- M=0 for MRI and M=1 for non-MRI
- I/O=0 for MRI and non-MRI register based instructions and I/O=MDR(11) for input-output based non-MRI instructions.
- For MRI, Opcode will be MDR(11-8) and for non-MRI, Opcode will be MDR(7-4)

Circuit Diagram:



Working Hardware : Decoding of non-MRI Instruction.



Working Hardware : Decoding of MRI Instruction

Most significant bit in IR is the “MODE” bit which will be set when there is a non-MRI instruction, i.e. when MDR(10-8) is ‘111’.
Next to MSB there is an I/O bit, which will be set for I/O operations only.

If MODE bit is set then IR(0-3) <- MDR(4-7)
else IR(0-3) <- MDR(8-11)

UNIQUE FEATURES OF EDITH PROCESSOR

1.Interrupt Handling:

Pressing a key “i” in between the execution of any operation will generate an interrupt.

_kbhit() function is implemented to read from the console. It checks the console for a recent keystroke. If the function returns a non-zero value, a keystroke is waiting in the buffer. The program can then call _getch() or _getche() to read the keystroke. By default, this function’s global state is scoped to the application.

More detail in Interrupt handling section.

2. Implementation of Barrel Shifter:

It is a specialised 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.

3. Multiple I/O ports:

EDITH has 4 input ports and 4 output ports. Each port can be selected only once at a time. Selection of ports is done by IS(Input Selector) and IO(Output Selector) respectively based on the last 4 bits(decoded bits) of instruction(Opcode) i.e. MDR(3-0).

4. Immediate Data Instruction:

In our Instruction Set, we kept Immediate Instruction in Non MRI, as we are not referring to memory(RAM) for Data. We give that data from Opcode itself. The last 4 bits of instruction(Opcode) i.e. MDR(3-0) will be immediately transferred to the target register.

Instruction Set

1. MRI Instruction

Format of 12-bit MRI Instruction:

I (1-Bit) (Direct or Indirect)	3-Bit MRI Opcode	8-Bit Address

Instruction Table:

MRI Instruction	Opcode	
	Direct (I=0)	Indirect (I=1)
LDA XX	0XX	8XX
STA XX	1XX	9XX
ADM XX	2XX	AXX
CALL XX	3XX	BXX
JMP XX	4XX	CXX
JC XX	5XX	DXX
JZ XX	6XX	EXX

'XX' - 8-Bit Address

2. non-MRI Instruction

Format of 12-bit non-MRI Instruction:

I (1-Bit) (Register or I/O based)	111	4-Bit Non-MRI Opcode	4-Bit Data/Don't care

Instruction Table:

Type	Non-MRI Instruction	Opcode
Arithmetic and Logical	ADD B	70_
	SUB B	71_
	CLA	72_
	CMA	73_
Register based Data Transfer	MOV A,B	74_
	MOV B,A	75_
Stack related	PUSH A	76_
	POP	77_
	RET	78_
* Immediate Data Transfer	MVI A,data	79X
	MVI F,data	7AX
Shift and Rotate related	LSC A,data	7BX
	RSC A,data	7CX
	RRX A,data	7DX
	ASR A,data	7EX
Halt	HLT	7F_
** I/O related (I/O=1)	IN S	F0X
	OUT S	F1X
	SKI S	F2X
	SKO S	F3X

'X' - 4-Bit Data

' _ ' - can be any value from 0 to F

** 'S' - 4-Bit Data to Select I/O Ports only take values(1,2,4,8)

MICRO INSTRUCTIONS

For MRI Instructions

LDA

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

Direct : instruction_format **(0XX)**

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) ,IR(L) , MAR(L,E) // MAR <- MDR(7-0), IR <- MDR(8-11)
T3: Idle
T4: Idle
T5: A(L) , RAM(E), Reset // A <- M[MAR]

Indirect: instruction_format **(8XX)**

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) , IR(L) , MAR(L,E) // MAR <- MDR(7-0), IR <- MDR(11-8)
T3: MDR(L) , RAM(E) // MDR <- M[MAR]
T4: MAR(L,E), MDR(E) // MAR <- MDR(7-0)
T5: A(L) , RAM(E), Reset // A <- M[MAR]

STA

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

Direct : instruction_format **(1XX)**

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) ,IR(L) , MAR(L,E) // MAR <- MDR(7-0), IR <- MDR(8-11)
T3: Idle

T4: Idle
T5: RAM(L) , A(E) , Reset // M[MAR] <- A

Indirect: instruction_format **(9XX)**

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) ,IR(L) , MAR(L,E) // MAR <- MDR(7-0), IR <- MDR(11-8)
T3: MDR(L) , RAM(E) // MDR <- M[MAR]
T4: MAR(L,E), MDR(E) // MAR <- MDR(7-0)
T5: RAM(L) , A(E) , Reset // M[MAR] <- A

ADM

Instruction to add AC to memory operand.

Direct : instruction_format **(2XX)**

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) ,IR(L) , MAR(L) // MAR <- MDR(7-0), IR <- MDR(8-11)
T3: Idle
T4: Idle
T5: MDR(L) , RAM(E) // MDR <- M[MAR]
T6: A(E,L) , MDR(E) , ALU(000) , Reset // A <- A + MDR

Indirect: instruction_format **(AXX)**

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) ,IR(L) , MAR(L,E) // MAR <- MDR(7-0), IR <- MDR(11-8)
T3: MDR(L) , RAM(E) // MDR <- M[MAR]
T4: MAR(L,E),MDR(E) // MAR <- MDR(7-0)
T5: MDR(L) , RAM(E) // MDR <- M[MAR]
T6: A(E,L) , MDR(E) , ALU(000) , Reset // A <- A + MDR

CALL

The CALL microoperation stores the return address in the stack and loads the PC with the given address.

Direct : instruction_format **(3XX)**

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E), IR(L), MAR(L,E) // MAR <- MDR(7-0), IR <- MDR(11-8)
T3: Idle
T4: Idle
T5: SP(D)
T6: SP(E), MAR(E,L) // MAR <- SP
T7: RAM(L), PC(E) // M[MAR] <- PC
T8: PC(L), MDR(E), Reset // PC <- MDR(7-0)

Indirect: instruction_format **(BXX)**

T0 : PC(E) , MAR(E,L)
T1 : PC(I), RAM(E), MDR(L)
T2: MDR(E), IR(L), MAR(L,E) // MAR <- MDR(7-0), IR <- MDR(11-8)
T3: MDR(L), RAM(E) // MDR <- M[MAR]
T4: Idle
T5: SP(D)
T6: SP(E), MAR(E,L) // MAR <- SP
T7: RAM(L), PC(E) // M[MAR] <- PC
T8: PC(L), MDR(E) // PC <- MDR(7-0)

JMP

The program sequence is transferred to the memory location specified by the particular level given in the operand.

Direct : instruction_format **(4XX)**

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) ,IR(L), MAR(L,E) // IR <- MDR(8-11)

T3: Idle
T4: Idle
T5: PC(L) , MDR(E), Reset // PC <- MDR(7-0)

Indirect: instruction_format **(CXX)**

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) , IR(L) , MAR(L,E) // MAR <- MDR(7-0), IR <- MDR(11-8)
T3: MDR(L) , RAM(E) // MDR <- M[MAR]
T4: Idle
T5: PC(L) , MDR(E), Reset // PC <- MDR(7-0)

JC

The program sequence is transferred to a particular level or a 12-bit address if C=1 (or carry is 1)

Direct : instruction_format **(5XX)**

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) ,IR(L), MAR(L,E) // IR <- MDR(8-11)
T3: Idle
T4: Idle
T5: PC(L) , MDR(E), Reset // if C=1, PC <- MDR(7-0)
else Reset

Indirect: instruction_format **(DXX)**

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) , IR(L) , MAR(L,E) // MAR <- MDR(7-0), IR <-MDR(11-8)
T3: MDR(L) , RAM(E) // MDR <- M[MAR]
T4: Idle
T5: PC(L) , MDR(E), Reset // if C=1, PC <- MDR(7-0)
else Reset

JZ

The program sequence is transferred to a particular level or a 12-bit address if Z=1 (or zero flag is 1)

Direct : instruction_format **(6XX)**

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) ,IR(L) // IR <- MDR(8-11)
T3: Idle
T4: Idle
T5: PC(L) , MDR(E), Reset // if Z=1, PC <- MDR(7-0)
else Reset

Indirect: instruction_format **(EXX)**

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) , IR(L) , MAR(L,E) // MAR <- MDR(7-0), IR <- MDR(11-8)
T3: MDR(L) , RAM(E) // MDR <- M[MAR]
T4: Idle
T5: PC(L) , MDR(E), Reset // if Z=1, PC <- MDR(7-0)
else Reset

For non-MRI Instructions

ADD B : instruction_format **(70_)**

Instruction to add values present in RAM/Registers.

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) ,IR(L) // IR <- MDR(4-7)
T3: Idle
T4: Idle
T5: A(E,L) , B(E) , ALU(000) , Reset

SUB B : instruction_format **(71_)**

Instruction to subtract values present in RAM/Registers.

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) ,IR(L) // IR <- MDR(4-7)
T3: Idle
T4: Idle
T5: A(E,L) , B(E) , ALU(001) , Reset

CLA : instruction_format **(72_)**

This instruction specifies to clear the accumulator.

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) ,IR(L) // IR <- MDR(4-7)
T3: Idle
T4: Idle
T5: A(E,L) , ALU(002) , Reset

CMA : instruction_format **(73_)**

This instruction specifies to complement the value present in the accumulator.

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) ,IR(L) // IR <- MDR(4-7)
T3: Idle
T4: Idle
T5: A(E,L) , ALU(003) , Reset

MOV A,B : instruction_format **(74_)**

This instruction specifies to move the data present in register B to register A.

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) ,IR(L) // IR <- MDR(4-7)
T3: Idle
T4: Idle
T5: B(E), A(L) , Reset // A <- B

MOV B,A : instruction_format **(75_)**

This instruction specifies to move the data present in register A to register B.

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) ,IR(L) // IR <- MDR(4-7)
T3: Idle
T4: Idle
T5: B(L), A(E) , Reset // B <- A

PUSH : instruction_format (76_)

This instruction is used to push the data from A to the address pointed by SP in RAM.

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) ,IR(L) // IR <-MDR(4-7)
T3: Idle
T4: Idle
T5: SP(D)
T6: SP(E) , MAR(E,L) // MAR <- SP
T7: RAM(L) , A(E) , Reset // M[MAR] <- A

POP : instruction_format (77_)

This instruction is used to pop the data from the address pointed by SP in RAM.

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) ,IR(L) // IR <-MDR(4-7)
T3: Idle
T4: Idle
T5: SP(E) , MAR(E,L) // MAR <- SP
T6: RAM(E) , A(L) // A <- M[MAR]
T7: SP(I) , Reset

RET : instruction_format (78_)
Retrieves PC from the address pointed by SP in RAM.
T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) ,IR(L) // IR <-MDR(4-7)
T3: Idle
T4: Idle
T5: SP(E) , MAR(E,L) // MAR <- SP
T6: RAM(E) , PC(L) // PC <- M[MAR]
T7: SP(I) , Reset

MVI A , data : instruction_format **(79X)**

This instruction specifies to move the data immediately in A.

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) ,IR(L) // IR <-MDR(4-7)
T3: Idle
T4: Idle
T5: MDR(E), A(L) , Reset // A <- MDR(0-3)

MVI F , data : instruction_format **(7AX)**

This instruction specifies to move the data immediately in F.

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) ,IR(L) // IR <-MDR(4-7)
T3: Idle
T4: Idle
T5: MDR(E), F(L) , Reset // F <- MDR(0-3)

LSC A , data : instruction_format **(7BX)**

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) ,IR(L) // IR <-MDR(4-7)
T3: Idle
T4: Idle
T5: MDR(E), BS(L) // BS <- MDR(0-3)
T6: A(E,L), BS(E), Reset

RSC A , data : instruction_format (7CX)

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) ,IR(L) // IR <-MDR(4-7)
T3: Idle
T4: Idle
T5: MDR(E), BS(L) // BS <- MDR(0-3)
T6: A(E,L), BS(E), Reset

RRX A , data : instruction_format (7DX)

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

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) ,IR(L) // IR <-MDR(4-7)
T3: Idle
T4: Idle
T5: MDR(E), BS(L) // BS <- MDR(0-3)
T6: A(E,L), BS(E), Reset

ASR A , data : instruction_format (7EX)

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) ,IR(L) // IR <-MDR(4-7)
T3: Idle
T4: Idle
T5: MDR(E), BS(L) // BS <- MDR(0-3)
T6: A(E,L), BS(E), Reset

HLT : instruction_format (7F_)

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) ,IR(L) // IR <-MDR(4-7)
T3: Idle
T4: Idle
T5: disable SG , Reset // SG - sequence generator

IN S : instruction_format (F0X)

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) ,IR(L) // IR <-MDR(4-7)
T3: Idle
T4: Idle
T5: MDR(E), IS(L) // IS <- MDR(0-3)
T6: A(L), IN(E), FGI(0) , Reset // A <- IN, FGI <- 0,
(IN Port selected by IS)

OUT S : instruction_format (F1X)

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) ,IR(L) // IR <-MDR(4-7)
T3: Idle
T4: Idle
T5: MDR(E), OS(L) // OS <- MDR(0-3)
T6: A(E), OUT(L), FGO(0), Reset // OUT <- A, FGO <- 0,
(OUT Port selected by OS)

SKI S : instruction_format **(F2X)**

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) ,IR(L) // IR <-MDR(4-7)
T3: Idle
T4: Idle
T5: MDR(E), IS(L) // IS <- MDR(0-3)
T6: PC(I), Reset // if FGI=1,PC<-PC+1, (FGI is selected by IS)
else Reset

SKO S : instruction_format **(F3X)**

T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)
T2: MDR(E) ,IR(L) // IR <-MDR(4-7)
T3: Idle
T4: Idle
T5: MDR(E), OS(L) // OS <- MDR(0-3)
T6: PC(I), Reset // if FGO=1,PC<-PC+1,(FGO is selected by OS)
else Reset

Sample Outputs

1. CALL 0x53

Input File:-

Number of Instructions :-
MVI A,7
LSC 8
STA 0X53
MVI A,7
LSC 8
MOV B,A
MVI A,8
LSC 4
ADD B
STA 0X54
CALL 0X53
HLT

input.txt

```
12
797
7B8
153
797
7B8
750
798
7B4
700
154
353
```

Output File:-

output.txt

```
~Fetch:
T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)

PC=01 IR=00 MDR=797 MAR=00 SP=FF A=700 B=000
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

~Decode:
This is non MRI Instruction
T2:  MDR(E) ,IR(L)         // IR <-MDR(4-7)

PC=01 IR=29 MDR=797 MAR=00 SP=FF A=700 B=000
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

~Execution:

* * * * * * * * MVI A,7 * * * * * * * *

T3: Idle
T4: Idle
T5: MDR(E) ,A(L), RESET

PC=01 IR=29 MDR=797 MAR=00 SP=FF A=007 B=000
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000
```

```
--- NEXT INSTRUCTION ---

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

PC=02 IR=29 MDR=7B8 MAR=01 SP=FF A=007 B=000
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

~Decode:
This is non MRI Instruction
T2:  MDR(E) ,IR(L)         // IR <-MDR(4-7)

PC=02 IR=2B MDR=7B8 MAR=01 SP=FF A=007 B=000
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

~Execution:

* * * * * * * * LSC A,8 * * * * * * * *
T6:   A(E,L), BS(E), Reset

PC=02 IR=2B MDR=7B8 MAR=01 SP=FF A=700 B=000
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000
```

```
--- NEXT INSTRUCTION ---

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

PC=03 IR=2B MDR=153 MAR=02 SP=FF A=700 B=000
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

~Decode:
This is MRI Instruction
T2:  MDR(E) ,IR(L) , MAR(L,E)        // MAR <- MDR(7-0), IR <-MDR(8-11)

PC=03 IR=01 MDR=153 MAR=53 SP=FF A=700 B=000
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

~Execution:

 * * * * * * * * Direct STA 53 * * * * * * * *

T3: Idle
T4: Idle
T5:  RAM(L) , A(E), Reset

Accumulaor Content--700 is loaded in Memory location 0X53

PC=03 IR=01 MDR=153 MAR=53 SP=FF A=700 B=000
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000
```

```
--- NEXT INSTRUCTION ---

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

PC=04 IR=01 MDR=797 MAR=03 SP=FF A=700 B=000
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

~Decode:
This is non MRI Instruction
T2:  MDR(E) ,IR(L)         // IR <-MDR(4-7)

PC=04 IR=29 MDR=797 MAR=03 SP=FF A=700 B=000
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

~Execution:

* * * * * * * * MVI A,7 * * * * * * * *

T3: Idle
T4: Idle
T5: MDR(E) ,A(L), RESET

PC=04 IR=29 MDR=797 MAR=03 SP=FF A=007 B=000
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000
```

```
--- NEXT INSTRUCTION ---

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

PC=05 IR=29 MDR=7B8 MAR=04 SP=FF A=007 B=000
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

~Decode:
This is non MRI Instruction
T2:  MDR(E) ,IR(L)         // IR <-MDR(4-7)

PC=05 IR=2B MDR=7B8 MAR=04 SP=FF A=007 B=000
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

~Execution:

* * * * * * * * LSC A,8 * * * * * * * *
T6:   A(E,L), BS(E), Reset

PC=05 IR=2B MDR=7B8 MAR=04 SP=FF A=700 B=000
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000
```

```
--- NEXT INSTRUCTION ---

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

PC=06 IR=2B MDR=750 MAR=05 SP=FF A=700 B=000
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

~Decode:
This is non MRI Instruction
T2:  MDR(E) ,IR(L)         // IR <-MDR(4-7)

PC=06 IR=25 MDR=750 MAR=05 SP=FF A=700 B=000
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

~Execution:

* * * * * * * * MOV B,A * * * * * * * *

T3: Idle
T4: Idle
T5:   B(L) , A(E) , Reset

PC=06 IR=25 MDR=750 MAR=05 SP=FF A=700 B=700
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000
```

```
--- NEXT INSTRUCTION ---

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

PC=07 IR=25 MDR=798 MAR=06 SP=FF A=700 B=700
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

~Decode:
This is non MRI Instruction
T2:  MDR(E) ,IR(L)         // IR <-MDR(4-7)

PC=07 IR=29 MDR=798 MAR=06 SP=FF A=700 B=700
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

~Execution:

* * * * * * * * MVI A,8 * * * * * * * *

T3: Idle
T4: Idle
T5: MDR(E) ,A(L), RESET

PC=07 IR=29 MDR=798 MAR=06 SP=FF A=008 B=700
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000
```

```
--- NEXT INSTRUCTION ---

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

PC=08 IR=29 MDR=7B4 MAR=07 SP=FF A=008 B=700
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

~Decode:
This is non MRI Instruction
T2:  MDR(E) ,IR(L)         // IR <-MDR(4-7)

PC=08 IR=2B MDR=7B4 MAR=07 SP=FF A=008 B=700
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

~Execution:

* * * * * * * * LSC A,4 * * * * * * * *
T6:   A(E,L), BS(E), Reset

PC=08 IR=2B MDR=7B4 MAR=07 SP=FF A=080 B=700
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000
```

```
--- NEXT INSTRUCTION ---

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

PC=09 IR=2B MDR=700 MAR=08 SP=FF A=080 B=700
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

~Decode:
This is non MRI Instruction
T2:  MDR(E) ,IR(L)         // IR <-MDR(4-7)

PC=09 IR=20 MDR=700 MAR=08 SP=FF A=080 B=700
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

~Execution:

* * * * * * * * ADD B * * * * * * * *

T3: Idle
T4: Idle
T5:  A(E,L) , B(E) , ALU(000) , Reset

PC=09 IR=20 MDR=700 MAR=08 SP=FF A=780 B=700
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000
```

```
--- NEXT INSTRUCTION ---

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

PC=0A IR=20 MDR=154 MAR=09 SP=FF A=780 B=700
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

~Decode:
This is MRI Instruction
T2:  MDR(E) ,IR(L) , MAR(L,E)        // MAR <- MDR(7-0), IR <-MDR(8-11)

PC=0A IR=01 MDR=154 MAR=54 SP=FF A=780 B=700
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

~Execution:

 * * * * * * * * Direct STA 54 * * * * * * * *

T3: Idle
T4: Idle
T5:  RAM(L) , A(E), Reset

Accumulaor Content--780 is loaded in Memory location 0X54

PC=0A IR=01 MDR=154 MAR=54 SP=FF A=780 B=700
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000
```

```
--- NEXT INSTRUCTION ---

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

PC=0B IR=01 MDR=353 MAR=0A SP=FF A=780 B=700
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

~Decode:
This is MRI Instruction
T2:  MDR(E) ,IR(L) , MAR(L,E)        // MAR <- MDR(7-0), IR <-MDR(8-11)

PC=0B IR=03 MDR=353 MAR=53 SP=FF A=780 B=700
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

~Execution:

 * * * * * * * * Direct CALL 53 * * * * * * * *

T3:  Idle
T4:  Idle
T5: SP(D)
T6: SP(E), MAR(E,L)
T7:  RAM(L), PC(E)
T8: PC(L),MDR(E), RESET

PC=53 IR=03 MDR=353 MAR=FE SP=FE A=780 B=700
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000
```

```
--- NEXT INSTRUCTION ---

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

PC=54 IR=03 MDR=700 MAR=53 SP=FE A=780 B=700
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

~Decode:
This is non MRI Instruction
T2:  MDR(E) ,IR(L)         // IR <-MDR(4-7)

PC=54 IR=20 MDR=700 MAR=53 SP=FE A=780 B=700
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

~Execution:

* * * * * * * * ADD B * * * * * * * *

T3: Idle
T4: Idle
T5:  A(E,L) , B(E) , ALU(000) , Reset

PC=54 IR=20 MDR=700 MAR=53 SP=FE A=E80 B=700
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000
```

```
--- NEXT INSTRUCTION ---

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

PC=55 IR=20 MDR=780 MAR=54 SP=FE A=E80 B=700
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

~Decode:
This is non MRI Instruction
T2:  MDR(E) ,IR(L)         // IR <-MDR(4-7)

PC=55 IR=28 MDR=780 MAR=54 SP=FE A=E80 B=700
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

~Execution:

* * * * * * * * RET * * * * * * * *

T3: Idle
T4: Idle
T5: SP(E) ,MAR(E,L)
T6: RAM(E) , PC(L)
T7: SP(I) ,RESET

PC=0B IR=28 MDR=780 MAR=FE SP=FF A=E80 B=700
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000
```

```
--- NEXT INSTRUCTION ---

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

PC=0C IR=28 MDR=7FF MAR=0B SP=FF A=E80 B=700
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

~Decode:
This is non MRI Instruction
T2:  MDR(E) ,IR(L)         // IR <-MDR(4-7)

PC=0C IR=2F MDR=7FF MAR=0B SP=FF A=E80 B=700
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

 * * * * * * * * HALT has occured * * * * * * * *

PC=0C IR=2F MDR=7FF MAR=0B SP=FF A=E80 B=700
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000
```

2. ASR 3

Input File:-

Number of Instructions :- 2
ASR 3
HLT

input.txt

```
2
7E3
7FF
```

Output File:-

output.txt

```
~Fetch:
T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)

PC=01 IR=00 MDR=7E3 MAR=00 A=700 SP=FF B=781
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

~Decode:
This is non MRI Instruction
T2:  MDR(E) ,IR(L)         // IR <-MDR(4-7)

PC=01 IR=2E MDR=7E3 MAR=00 A=700 SP=FF B=781
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

~Execution:

* * * * * * * * ASC A,3 * * * * * * * *

PC=01 IR=2E MDR=7E3 MAR=00 A=0E0 SP=FF B=781
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

T6:   A(E,L), BS(E), Reset
```

```
-- NEXT INSTRUCTION --

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

PC=02 IR=2E MDR=7FF MAR=01 A=0E0 SP=FF B=781
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

~Decode:
This is non MRI Instruction
T2:  MDR(E) ,IR(L)         // IR <-MDR(4-7)

PC=02 IR=2F MDR=7FF MAR=01 A=0E0 SP=FF B=781
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

 * * * * * * * * HALT has occured * * * * * * * *

PC=02 IR=2F MDR=7FF MAR=01 A=0E0 SP=FF B=781
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000
```

3. SKI 2

Input File:-

Number of Instructions :- 4
SKI 2
ADD B
ASR 3
HLT

Output File:-

output.txt

```
~Fetch:
T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)

PC=01 IR=00 MDR=F22 MAR=00 SP=FF A=E52 B=000
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

~Decode:
This is non MRI Instruction
T2:  MDR(E) ,IR(L)         // IR <-MDR(4-7)

PC=01 IR=32 MDR=F22 MAR=00 SP=FF A=E52 B=000
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

~Execution:

* * * * * * * * SKI 2 * * * * * * * *

T6:   PC(I), Reset

PC=02 IR=32 MDR=F22 MAR=00 SP=FF A=E52 B=000
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

--- NEXT INSTRUCTION ---

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

PC=03 IR=32 MDR=7E3 MAR=02 SP=FF A=E52 B=000
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

~Decode:
This is non MRI Instruction
T2:  MDR(E) ,IR(L)         // IR <-MDR(4-7)

PC=03 IR=2E MDR=7E3 MAR=02 SP=FF A=E52 B=000
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

~Execution:

* * * * * * * * ASR A,3 * * * * * * * *

PC=03 IR=2E MDR=7E3 MAR=02 SP=FF A=FCA B=000
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

T6:   A(E,L), BS(E), Reset
```

```
--- NEXT INSTRUCTION ---

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

PC=04 IR=2E MDR=7FF MAR=03 SP=FF A=FCA B=000
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

~Decode:
This is non MRI Instruction
T2:  MDR(E) ,IR(L)         // IR <-MDR(4-7)

PC=04 IR=2F MDR=7FF MAR=03 SP=FF A=FCA B=000
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

 * * * * * * * * HALT has occured * * * * * * * *

PC=04 IR=2F MDR=7FF MAR=03 SP=FF A=FCA B=000
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000
```

Interrupt Handling

It is implemented by kbhit() whenever any key is pressed it checks whether the key is “i”. If the key pressed is “i” then an interrupt occurs and sets the R flag. It will jump to Interrupt handling Subroutine and the return address is saved in the stack.

Interrupt handling Subroutine is present from 0xF0 location in RAM and at last location of subroutine has a JMP instruction which takes the PC to the main program.

After interrupt handling, the PC takes the return address from the stack and resets the R flag.

There is an ION flag which signifies whether a processor can take an interrupt or not.

ION flag = 0 indicates that the processor can take and handle Interrupt.

ION flag = 0 indicates that the processor won’t take and handle any Interrupt.

Example:

Input File:-

Number of Instructions :- 3
ADD B
SUB B
HLT

```
no of instructions:
3
700
710
7FF
```

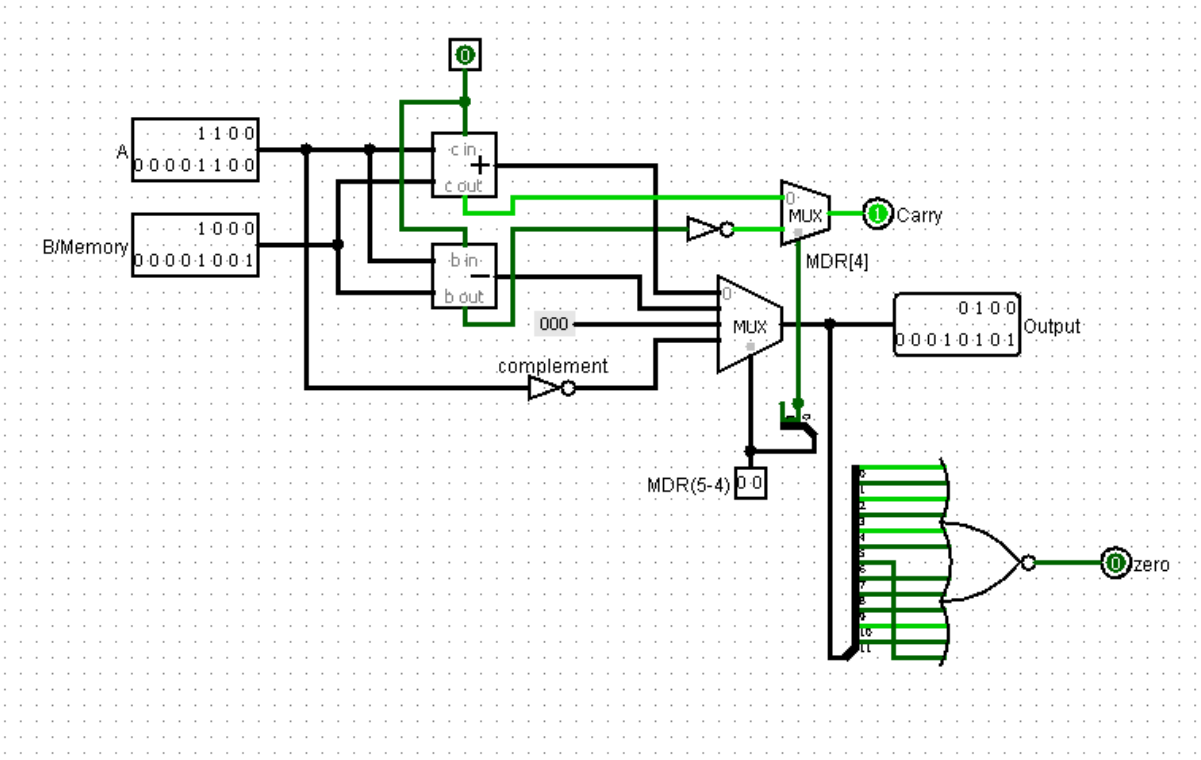
Output File:-

```
~Fetch:
T0 : PC(E) , MAR(E,L)
T1 : PC(I) , RAM(E) , MDR(L)

PC=01 IR=00 MDR=700 MAR=00 SP=FF A=700 B=000
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

~Decode:
This is non MRI Instruction
T2:  MDR(E) ,IR(L)         // IR <-MDR(4-7)

PC=01 IR=20 MDR=700 MAR=00 SP=FF A=700 B=000
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

~Execution:

* * * * * * * * ADD B * * * * * * * *

T3: Idle
T4: Idle
T5:  A(E,L) , B(E) , ALU(000) , Reset

PC=01 IR=20 MDR=700 MAR=00 SP=FF A=700 B=000
(ION=0 C=0 Z=0 R=1) FLG=1
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

* * * * * * * *Interrupt Has Occured* * * * * * * *
 PC jump to location 0xF0 where Interrupt handling subroutine is written and Stack has the return address and R flag is set.

PC=F0 IR=20 MDR=700 MAR=FE SP=FE A=700 B=000
(ION=0 C=0 Z=0 R=1) FLG=1
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000
```

```
.....................Interrupt Handling...................
.
.
.
.
.
.
.
.
* * * * * * * *Interrupt has been serviced* * * * * * * *

PC=F1 IR=20 MDR=700 MAR=FE SP=FE A=700 B=000
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

* * * * * * * *PC will jump to main program where the interrupt has occured* * * * * * * *

PC=01 IR=20 MDR=700 MAR=FE SP=FF A=700 B=000
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000


--- NEXT INSTRUCTION ---

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

PC=02 IR=20 MDR=710 MAR=01 SP=FF A=700 B=000
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000
```

```
~Decode:
This is non MRI Instruction
T2:  MDR(E) ,IR(L)         // IR <-MDR(4-7)

PC=02 IR=21 MDR=710 MAR=01 SP=FF A=700 B=000
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

~Execution:


PC=03 IR=2F MDR=7FF MAR=02 SP=FF A=0 B=000
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000

 * * * * * * * * HALT has occured * * * * * * * *

PC=03 IR=2F MDR=7FF MAR=02 SP=FF A=0 B=000
(ION=0 C=0 Z=0 R=0) FLG=0
IN1=005 IN2=000 IN3=000 IN4=000 -- FGI=0110
OUT1=000 OUT2=000 OUT3=000 OUT4=000 -- FGO=0000
```

Hardware implementation of ALU



Based on values of MDR[5] and MDR[4], result is selected among 4 different operations.

00 - ADD

01 - SUB

10 - CLA

11 - CMA

And correspondingly zero and carry flags will be updated.

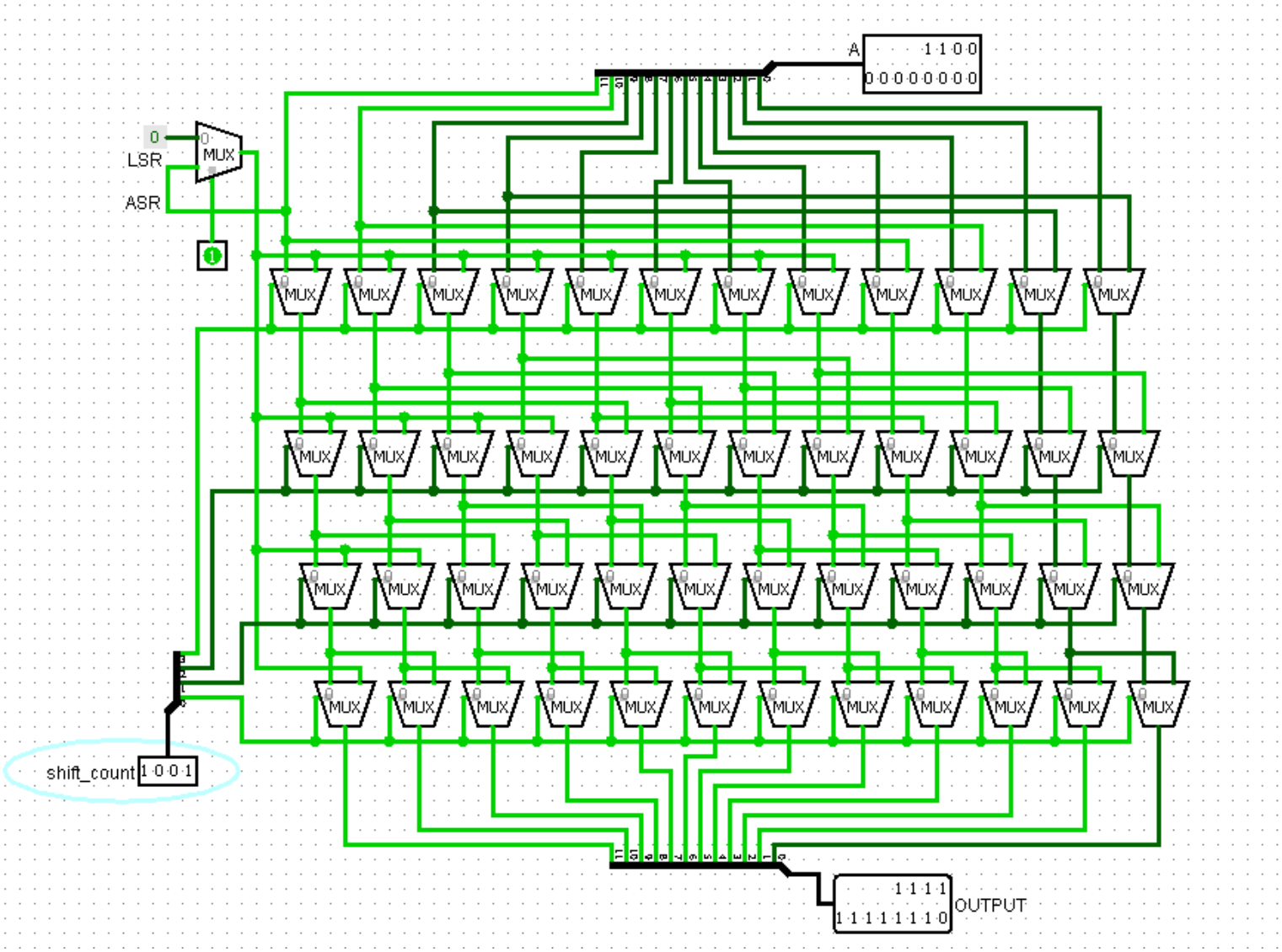
Updating Carry Flag :

In addition, based on the values of A and (B or from memory) if calculation happens to generate a carry then carry flag will be set i.e. 1 otherwise 0.
In case of Subtraction, we are using a hardware which computes subtraction as:
A SUB B = A - B + C - 1 , where C is carry flag
If C == 1 (set) , simply A - B
else if C == 0 , then A - B - 1 and hence use of NOT gate in b/w is justified.

Updating Zero Flag :

The NOR gate outputs a 1 only if all input bits are 0. If any input bit is a 1, the NOR's output is a 0.

Hardware implementation of ASR in Barrel Shifter



If A[11] = 1 i.e. value in accumulator is negative and hence shifted bits are ‘1’.
If A[11] = 0 i.e. value in accumulator is positive and hence shifted bits will be ‘0’.

Barrel shifter is able to complete the shift in a single clock cycle, giving it a great advantage over a simple shifter which can shift n bits in n clock cycles.

Conclusion

EDITH deals with instruction set of size 12-bit using RAM 256X12 bits. It decodes a macro-instruction as I bit, Opcode(3-bits) and 8-bits for address. MSB is considered as I bit, which signifies the type of addressing whether direct or indirect while dealing with MRI instructions or whether register type or I/O type instructions in case of NON-MRI instructions.

MRI : Next to I bit, 3 bits are used for opcode which is used to identify the type of instruction we want to execute in a program, followed by last 8-bits that are used for address handling.

NON-MRI : Next to I bit, 3 bits are always '111', further 4 bits are identifying as opcode for non-MRI instructions, it signifies the type of instruction to be executed, and last 4 bits represents data in case of MVI instruction, shift count in case of shift operations, port selector in case of I/O operation, and can be any value otherwise.

Flags dealt in the Processor are ION, C , Z , R
ION: To signify whether a processor can take an interrupt or not.
C: To signify the carry bit while dealing the arithmetic instructions.
Z: To signify the zero value of the accumulator
R: To signify an interrupt.

EDITH also takes care of an interrupt (**CONDITION : when ION = 0**) that might be generated while executing a program. It first completes the execution of the particular macro-operation it is performing and then handles interrupt and further it continues to execute the program where it has left off.