COLLEGE

(UNIVERSITY OF DELHI)

COMPUTER SYSTEM ARCHITECTURE

PRACTICAL FILE

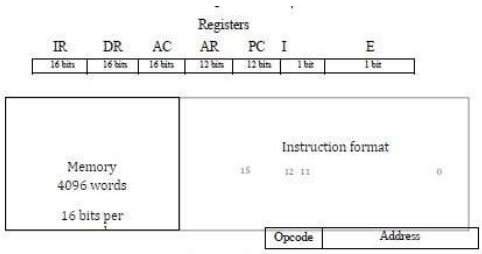
SUBMITTED BY-

NAME:- JATOTH DINESH
COURSE:- B.A. (P) (CA + ECO)
ROLL NO:- 23BAA009

SUBMITTED TO -

MS. SWEETY

INDEX

S.NO.	TOPIC	DATE	REMARKS
1	Create a machine based on the following architecture:  Basic Computer Instructions Memory Reference: Symbol, Hex; Register Reference: Symbol, Hex AND 0xxx (Direct Addressing) ADD 1xxx LDA 2xxx STA 3xxx BUN 4xxx BSA 5xxx ISZ 6xxx AND_I 8xxx (Indirect Addressing) ADD_I 9xxx LDA_I Axxx STA_I Bxxx BUN_I Cxxx BSA_I Dxxx ISZ_I Exxx CLA 7800 CLE 7400 CMA 7200 CME 7100 CIR 7080 CIL 7040 INC 7020 SPA 7010 SNA 7008 SZA 7004 SZE 7002 HLT 7001 INP F800 OUT F400 Design the register set, memory and the instruction set. Use this machine for the assignments of this section.		
2	Create a Fetch routine of the instruction cycle.		
3	Write an assembly program to simulate ADD operation on two user-entered numbers.		
4	Write an assembly program to simulate SUBTRACT operation on two userentered numbers.		

5	Write an assembly program to simulate the following logical operations on two user entered numbers. i. AND ii. OR iii. NOT iv. XOR v. NOR vi. NAND		
6	Write an assembly program for simulating following memory-reference instructions. i. ADD ii. LDA iii. STA iv. BUN v. ISZ		
7	Write an assembly language program to simulate the machine for following register reference instructions and determine the contents of AC, E, PC, AR and IR registers in decimal after the execution: i. CLA ii. CMA iii. CME iv. HLT		
8	Write an assembly language program to simulate the machine for following register reference instructions and determine the contents of AC, E, PC, AR and IR registers in decimal after the execution: i. INC ii. SPA iii. SNA iv. SZE		
9	Write an assembly language program to simulate the machine for following register reference instructions and determine the contents of AC, E, PC, AR and IR registers in decimal after the execution: i. CIR ii. CIL		

10	Write an assembly program that reads in integers and adds them together; until a negative non-zero number is read in. Then it outputs the sum (not including the last number).		
11	Write an assembly program that reads in integers and adds them together; until zero is read in. Then it outputs the sum.		

PRACTICAL 1

AIM: Creating a base machine

PROCEDURE :

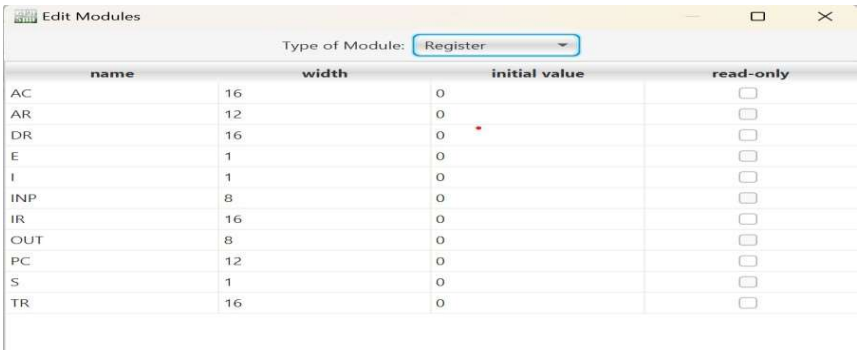
Use Stimulator- CPU Sim 4.0.11 for implementation

1.Open CPU Stimulator

2.Click on file and click on save machine and save the machine with .cpu extension

3. Go to Modify and click on the hardware module , then create the fields given below on the window pop-upped :

❼Types of modules -> Register



name	width	initial value	read-only
AC	16	0	☐
AR	12	0	☐
DR	16	0	☐
E	1	0	☐
I	1	0	☐
INP	8	0	☐
IR	16	0	☐
OUT	8	0	☐
PC	12	0	☐
S	1	0	☐
TR	16	0	☐

❼Types of modules -> Condition bit

name	register	bit	halt
CARRYBIT	E	0	☐
HALTBIT	S	0	☑

❼Types of modules -> RAM

Edit Modules

Type of Module: RAM

name	length	cellSize
MAIN	4096	16

4. Again go to Modify and click on Microinstruction then create the following field given below:

❼Types of Microinstruction -> TransferRtoR

Edit Microinstructions

Type of Microinstruction: TransferRtoR

name	source	srcStartBit	dest	destStartBit	numBits
AR<-IR(0-11)	IR	0	AR	0	12
AR<-PC	PC	0	AR	0	12

❼Types of Microinstruction -> Memory access

Edit Microinstructions

Type of Microinstruction: MemoryAccess

name	direction	memory	data	address
IR<-MAIN[AR]	read	MAIN	IR	AR
MAIN[AR]<-AC	write	MAIN	AC	AR

❼Types of Microinstruction -> Increment

Edit Microinstructions

Type of Microinstruction: Increment

name	register	overflowBit	carryBit	delta
PC-INCR	PC	(none)	(none)	1

❼Types of Microinstruction -> Decode

Edit Microinstructions

Type of Microinstruction: Decode

name	ir
DECODE-IR	IR

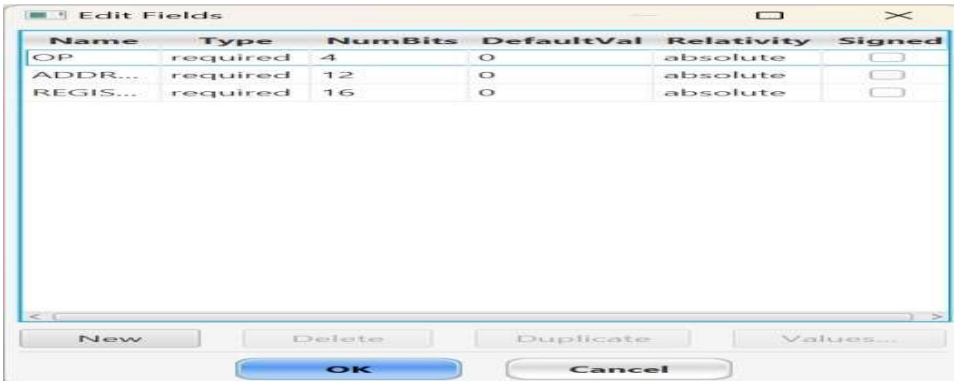
❼Types of Microinstruction -> SetConBit

Edit Microinstructions

Type of Microinstruction: SetCondBit

name	bit	value
CARRY	CARRYBIT	0
HALT	HALTBIT	1

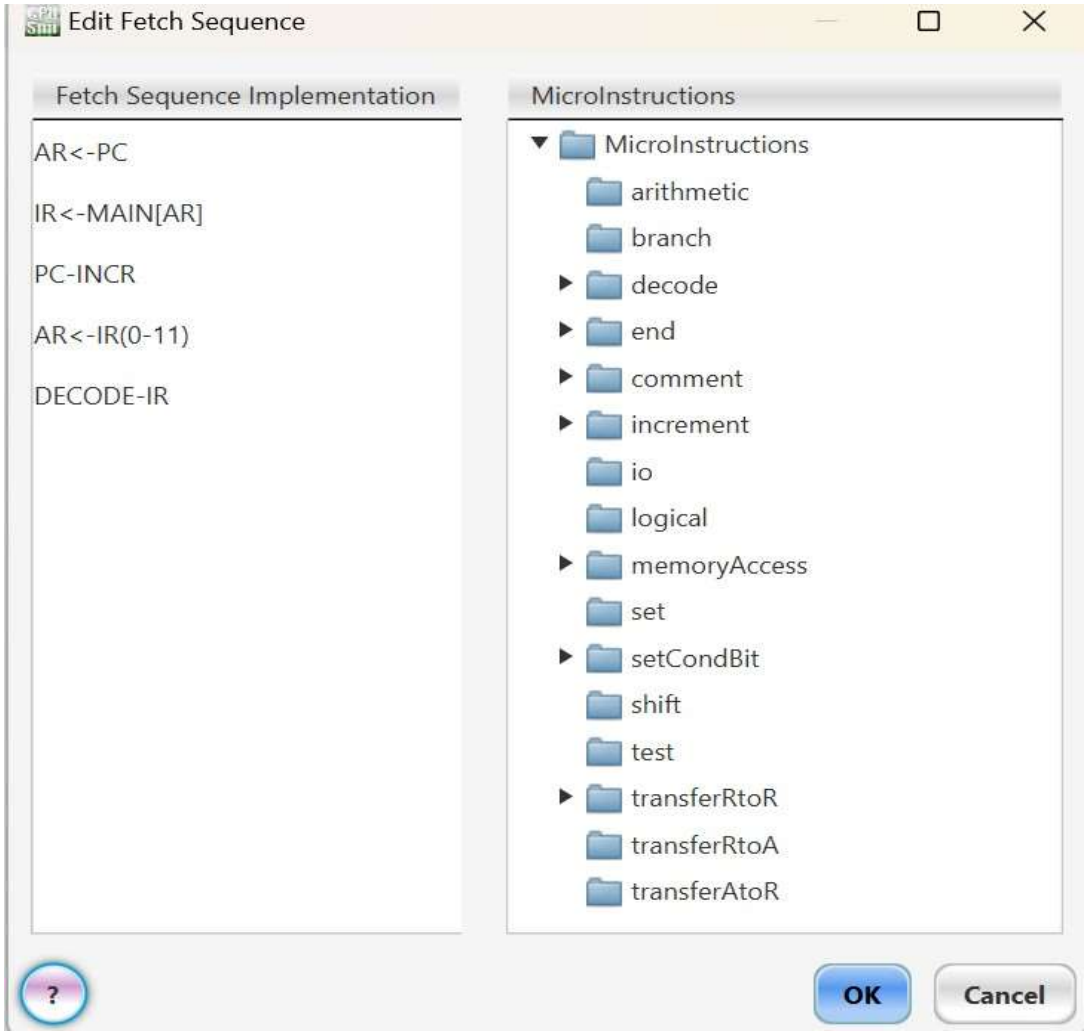
5. Go to Modify , click on machine instructions, then click on edit files option at the right bottom corner of the window…and create following new field given below :



PRACTICAL 2

AIM: To Implement a fetch Sequence <u>Procedure:</u>

Go to Modify and click on Fetch Sequence now drag the following Microinstructions given below from right side and drop it to left side or to fetch sequence implementation field:



PRACTICAL 3

AIM : Write an assembly program to simulate ADD operation on two user-entered numbers.

PROGRAM CODE :

addd.a

```
START:INP
STA NUM
INP
ADD NUM
OUT
HLT
NUM: .data 1 0
```

PROCEDURE :

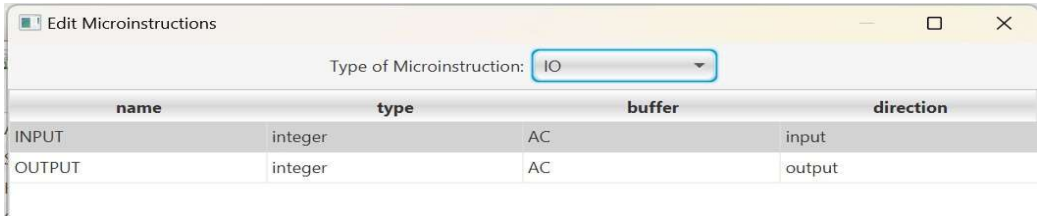
1. Go to Modify and click on Machine Instructions
2. Create five new instructions INP, OUT, HLT, STA, ADD with their respective format and implementation.

FORMATS :

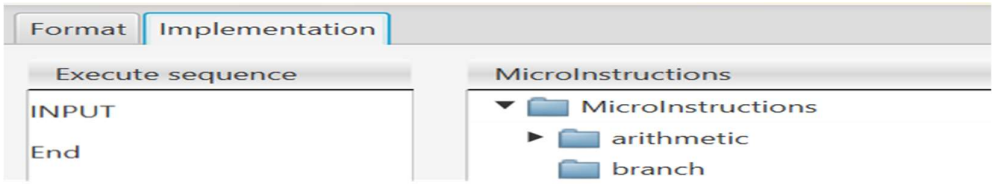
- For INP , set Opcode to 0xF800 and drag Register from the right side and drop it down to Opcode
- For OUT , set Opcode to 0Xf400 and drag Register from the right side and drop it down to Opcode.
- For HLT , set Opcode to 0x7001 and drag Register from the right side and drop it down to Opcode.
- For STA , set Opcode to 0x4 and drag OP and then Address from the right side and drop it down to Opcode.
- For ADD , set Opcode to 0x0 and drag OP and then Address from the right side and drop it down to Opcode.

IMPLEMETATIONS :

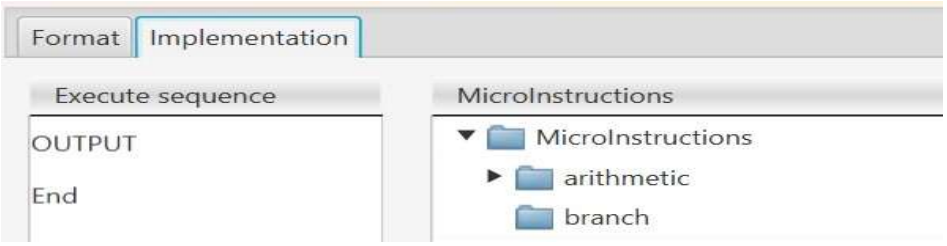
- For INP, double click on io and create new field



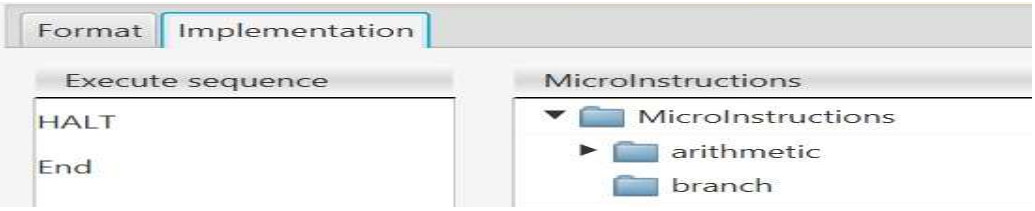
Then drag input files from io and drag to Execute Sequence column then drag end file from end



- For OUT , drag the output file from io and drop to Execute Sequence column then drag end file from end .



- For HLT , drag the file HALT from the SetConBit, then drag end file from end and drop them to Execute Sequence Column



- For STA, double click on Memory access and create new field MAIN[AR]<-AC. Then drag file from Memory access MAIN[AR]<-AC and end file from END and drop to Execute Sequence column.

Edit Microinstructions

Type of Microinstruction: MemoryAccess

name	direction	memory	data	address
DR<-MAIN[AR]	read	MAIN	DR	AR
IR<-MAIN[AR]	read	MAIN	IR	AR
MAIN[AR]<-AC	write	MAIN	AC	AR

- For ADD , double click on ARITHMETIC and create new field

Edit Microinstructions

Type of Microinstruction: Arithmetic

name	type	source1	source2	destination	overflowBit	carryBit
AC<-AC+DR	ADD	AC	DR	AC	(none)	CARRYBIT

Then drag AC<-AC+DR file from ARITHMETIC , then drag DR<-MAIN[AR] file from MEMORY ACCESS and end file from end and drop to Execute Sequence column

Format | Implementation

Execute sequence

DR<-MAIN[AR]

AC<-AC+DR

End

MicroInstructions

- MicroInstructions
  - arithmetic
  - branch
  - decode
  - end

- Press ctrl+S and save the text file with .a extension
- Go to Execute option , click on clear, load, assemble, and rum option for executing the program .

OUTPUT will be:

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 2
Enter Inputs, the first of which must be an Integer: 3
Output:  5
EXECUTION HALTED DUE TO AN EXCEPTION: No opcode matched the bits in the register: IR.
```

PRACTICAL 4

AIM : Write an assembly program to simulate SUBTRACT operation on two userentered numbers.

PROGRAM CODE :

SUBTRACTION.a

```
START:INP
STA NUM
INP
CMA
INC
ADD NUM
OUT
HLT
NUM: .data 1 0
```

PROCEDURE :

1. Go to Modify and click on Machine Instructions
2. Create seven new instructions INP, OUT, HLT, STA, ADD, INC, CMA with their respective format and implementation.

FORMATS :

- Same as practical 3
- For CMA, set Opcode to 0x7200 and drag register from the right side and drop it down to Opcode.
- For INC, set Opcode to 0x7020 and drag register from the right side and drop it down to Opcode.

IMPLEMETATIONS :

- Same as practical 3
- For CMA, double click on Logical and create new field AC<-AC'.

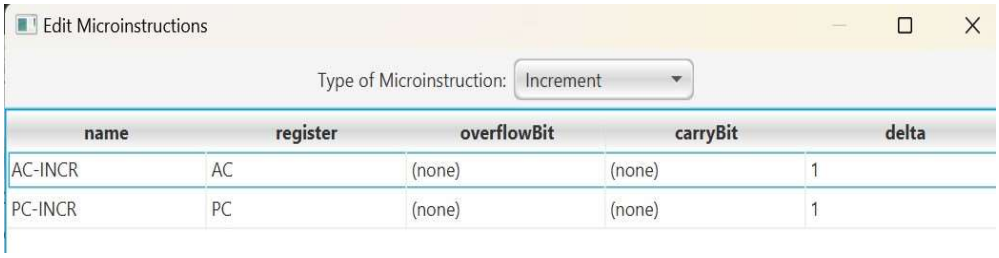
Edit Microinstructions

Type of Microinstruction: Logical

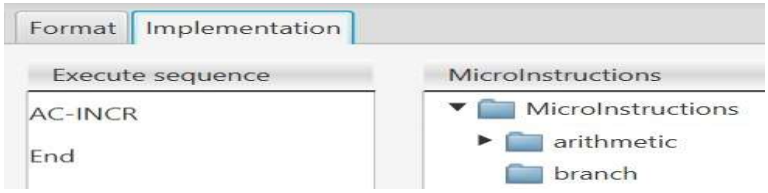
name	type	source1	source2	destination
AC<-AC'	NOT	AC	AC	AC

Then drag file AC<-AC' from Logical and end file from END and drop to Execute Sequence column.

- For INC , double click on Increment and create new field AC-INCR.



Then drag file AC-INCR from Increment and end file from END and drop to Execute Sequence column.



- Press ctrl+S and save the text file with .a extension
- Go to Execute option , click on clear, load, assemble, and rum option for executing the program .

OUTPUT will be:

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 9
Enter Inputs, the first of which must be an Integer: 8
Output:  1
EXECUTION HALTED DUE TO AN EXCEPTION: No opcode matched the bits in the register: IR.
```

PRACTICAL 5

AIM : write an assembly program to simulate the following logical operations on two user- entered numbers

(I) AND

PROGRAM CODE :

```
and.a
INP
STA NUM
INP
AND NUM
OUT
HLT
NUM: .data 1 0
```

PROCEDURE :

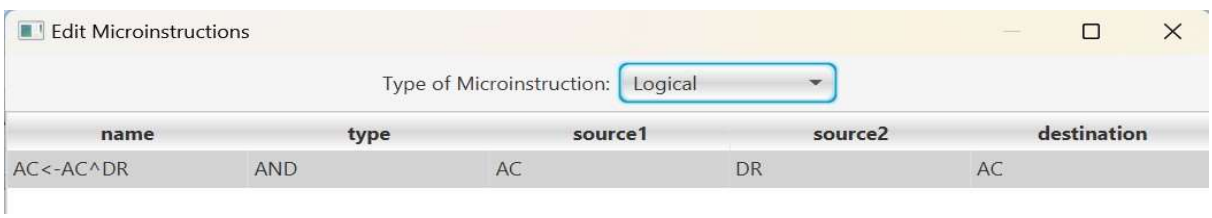
1. Go to Modify and click on Machine Instructions
2. Create five new instructions INP, OUT, HLT, STA, AND with their respective format and implementation.

FORMATS :

- Same as practical 3
- For AND , set Opcode to 0x0 and drag OP and then Address from the right side and drop it down to Opcode.

IMPLEMETATIONS :

- Same as practical 3
- For AND , double click on LOGICAL and create new field



Then drag AC<-AC^DR file from LOGICAL, then drag DR<-MAIN[AR] file from MEMORY ACCESS and end file from end and drop to Execute Sequence column

- Press ctrl+S and save the text file with .a extension
- Go to Execute option , click on clear, load, assemble, and run option for executing the program .

OUTPUT will be:

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 1
Enter Inputs, the first of which must be an Integer: 0
Output:  0
Enter Inputs, the first of which must be an Integer:
```

AIM : write an assembly program to simulate the following logical operations on two user- entered numbers

(II) OR

PROGRAM CODE :

OR.a

```
INP
STA NUM
INP
OR NUM
OUT
HLT
NUM: .data 1 0
```

PROCEDURE :

1. Go to Modify and click on Machine Instructions
2. Create five new instructions INP, OUT, HLT, STA, OR with their respective format and implementation.

FORMATS :

- Same as practical 3
- For OR , set Opcode to 0x0 and drag OP and then Address from the right side and drop it down to Opcode.

IMPLEMETATIONS :

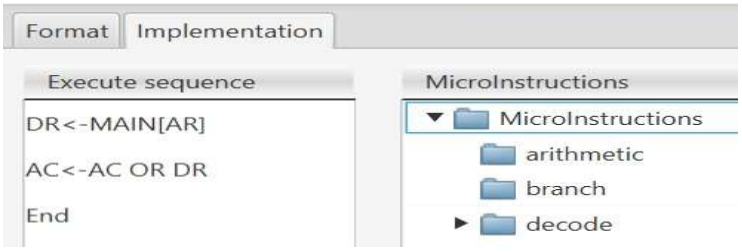
- Same as practical 3
- For OR , double click on LOGICAL and create new field

Edit Microinstructions

Type of Microinstruction: Logical

name	type	source1	source2	destination
AC<-AC OR DR	OR	AC	DR	AC
AC<-AC^DR	AND	AC	DR	AC

Then drag DR<-MAIN[AR] file from MEMORY ACCESS , then drag AC<-AC OR DR file from LOGICAL and end file from end and drop to Execute Sequence column



- Press ctrl+S and save the text file with .a extension
- Go to Execute option , click on clear, load, assemble, and run option for executing the program .

OUTPUT will be:

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 1
Enter Inputs, the first of which must be an Integer: 0
Output:  1
Enter Inputs, the first of which must be an Integer: 0
Enter Inputs, the first of which must be an Integer: 0
Output:  0
```

AIM : write an assembly program to simulate the following logical operations on two user- entered numbers

(III) NOT

PROGRAM CODE :

```
NOT.a
INP
NOT
OUT
HLT
```

PROCEDURE :

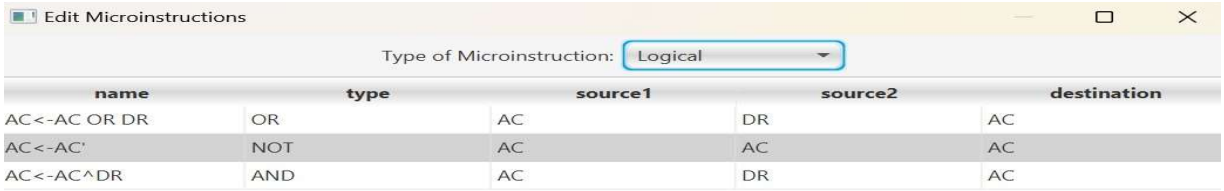
1. Go to Modify and click on Machine Instructions
2. Create four new instructions INP, OUT, HLT, NOT with their respective format and implementation.

FORMATS :

- Same as practical 3
- For NOT , set Opcode to 0x2 and drag register from the right side and drop it down to Opcode.

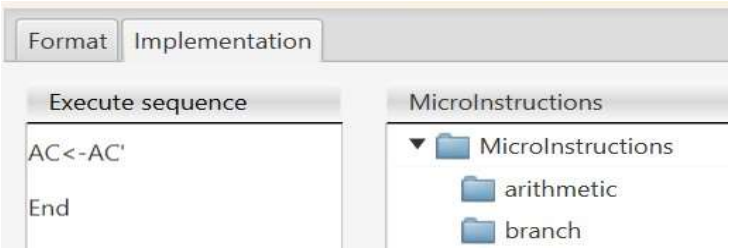
IMPLEMETATIONS :

- Same as practical 3
- For NOT , double click on LOGICAL and create new field



name	type	source1	source2	destination
AC<-AC OR DR	OR	AC	DR	AC
AC<-AC'	NOT	AC	AC	AC
AC<-AC^DR	AND	AC	DR	AC

Then drag AC<-AC' file from LOGICAL and end file from end and drop to Execute Sequence column



- Press ctrl+S and save the text file with .a extension
- Go to Execute option , click on clear, load, assemble, and run option for executing the program .

OUTPUT will be:

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 0
Output:  -1
EXECUTION HALTED DUE TO AN EXCEPTION: No opcode matched the bits in the register: IR.
```

AIM : write an assembly program to simulate the following logical operations on two user- entered numbers

(IV) XOR

PROGRAM CODE :

```
XOR.a
INP
STA NUM
INP
XOR NUM
OUT
HLT

NUM: .data 1 0
```

PROCEDURE :

1. Go to Modify and click on Machine Instructions
2. Create five new instructions INP, OUT, HLT,STA, XOR with their respective format and implementation.

FORMATS :

- Same as practical 3
- For XOR , set Opcode to 0x0 and drag OP and then Address from the right side and drop it down to Opcode.

IMPLEMETATIONS :

- Same as practical 3
- For XOR , double click on LOGICAL and create new field

Edit Microinstructions

Type of Microinstruction: Logical

name	type	source1	source2	destination
AC<-AC XOR DR	XOR	AC	DR	AC
AC<-AC^DR	AND	AC	DR	AC

-

Then drag DR<-MAIN[AR] file from MEMORY ACCESS , then drag AC<-AC XOR DR file from LOGICAL and end file from end and drop to Execute Sequence column

Format | Implementation

Execute sequence
DR<-MAIN[AR]
AC<-AC XOR DR
End

MicroInstructions
MicroInstructions
arithmetic
branch
decode

- Press ctrl+S and save the text file with .a extension
- Go to Execute option , click on clear, load, assemble, and run option for executing the program .

OUTPUT will be:

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 1
Enter Inputs, the first of which must be an Integer: 1
Output:  0
Enter Inputs, the first of which must be an Integer: 0
Enter Inputs, the first of which must be an Integer: 1
Output:  1
```

AIM : write an assembly program to simulate the following logical operations on two user- entered numbers

(V) NOR

PROGRAM CODE :

NOR.a

```
INP
STA NUM
INP
NOR NUM
OUT
HLT

NUM: .data 1 0

```

PROCEDURE :

1. Go to Modify and click on Machine Instructions
2. Create five new instructions INP, OUT, HLT, STA, NOR with their respective format and implementation.

FORMATS :

- Same as practical 3
- For NOR , set Opcode to 0x1 and drag OP and then Address from the right side and drop it down to Opcode.

IMPLEMETATIONS :

- Same as practical 3
- For NOR , double click on LOGICAL and create new field AC<-AC NOR DR.

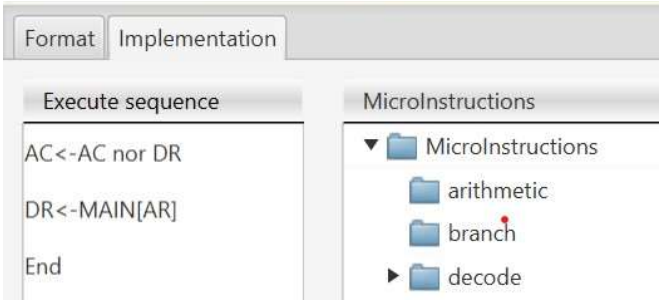
Edit Microinstructions

Type of Microinstruction: Logical

name	type	source1	source2	destination
AC<-AC nor DR	NOR	DR	AC	AC

Then drag AC<-AC NOR DR file from LOGICAL, then drag DR<-MAIN[AR] file from

MEMORY ACCESS , then and end file from end and drop to Execute Sequence column



- Press ctrl+S and save the text file with .a extension
- Go to Execute option , click on clear, load, assemble, and run option for executing the program .

OUTPUT will be:

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 1
Enter Inputs, the first of which must be an Integer: 0
Output:  -1
EXECUTION HALTED DUE TO AN EXCEPTION: No opcode matched the bits in the register: IR.
```

AIM : write an assembly program to simulate the following logical operations on two user- entered numbers

(VI) NAND

PROGRAM CODE :

NAND.a

```
INP
STA NUM
INP
NAND NUM
OUT
HLT

NUM: .data 1 0
```

PROCEDURE :

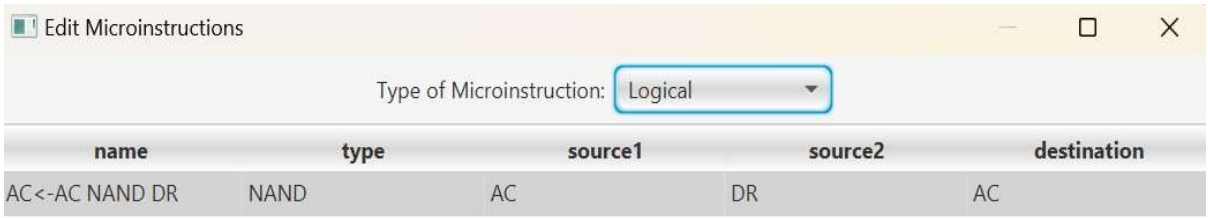
1. Go to Modify and click on Machine Instructions
2. Create five new instructions INP, OUT, HLT, STA, NAND with their respective format and implementation.

FORMATS :

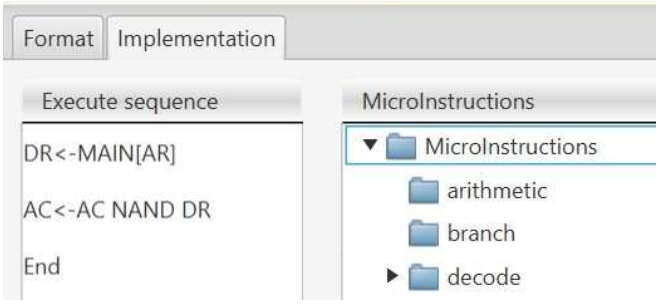
- Same as practical 3
- For NAND , set Opcode to 0x1 and drag OP and then Address from the right side and drop it down to Opcode.

IMPLEMETATIONS :

- Same as practical 3
- For NAND , double click on LOGICAL and create new field AC<-AC NAND DR.



Then drag DR<-MAIN[AR] file from MEMORY ACCESS , then drag AC<-AC NAND DR from LOGIC and at last drag end file from end and drop to Execute Sequence column



- Press ctrl+S and save the text file with .a extension
- Go to Execute option , click on clear, load, assemble, and run option for executing the program .

OUTPUT will be:

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 0
Enter Inputs, the first of which must be an Integer: 0
Output:  -1
EXECUTION HALTED DUE TO AN EXCEPTION: No opcode matched the bits in the register: IR.
```

PRACTICAL 6

AIM : Write an assembly program for simulating following memory-reference instructions

. (I) ADD

Same as practical 3

AIM : Write an assembly program for simulating following memory-reference instructions

. (II) LDA

PROGRAM CODE :

lda.a

```
INP
STA NUM
LDA NUM
OUT
HLT

NUM: .data 1 0
```

PROCEDURE :

1. Go to Modify and click on Machine Instructions
2. Create five new instructions INP, OUT, HLT, STA, LDA, with their respective format and implementation.

FORMATS :

- Same as practical 3
- For LDA , set Opcode to 0xA and drag OP and then Address from the right side and drop it down to Opcode.

IMPLEMETATIONS :

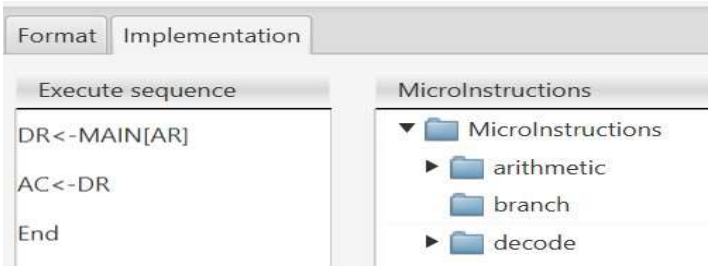
- Same as practical 3
- For LDA , double click on TransferRtoR and create new field AC<-DR.

Edit Microinstructions

Type of Microinstruction: TransferRtoR

name	source	srcStartBit	dest	destStartBit	numBits
AC<-DR	DR	0	AC	0	16
AR<-IR(0-11)	IR	0	AR	0	12
AR<-PC	PC	0	AR	0	12

Then drag the file DR<-MAIN[AR] from memory access and then drag PC<-AR from TransferRtoR and end file from end and drop to execute sequence column



- Press ctrl+S and save the text file with .a extension
- Go to Execute option , click on clear, load, assemble, and run option for executing the program .

OUTPUT will be:

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 6
Output:  6
EXECUTION HALTED DUE TO AN EXCEPTION: No opcode matched the bits in the register: IR.
```

Registers

Name	Width	Data
AC	16	6
AR	12	6
DR	16	6
E	1	0

AIM : Write an assembly program for simulating following memory-reference instructions

. (III) STA

PROGRAM CODE :

sta.a

```
INP
STA NUM
OUT
HLT
NUM: .data 1 0
```

PROCEDURE :

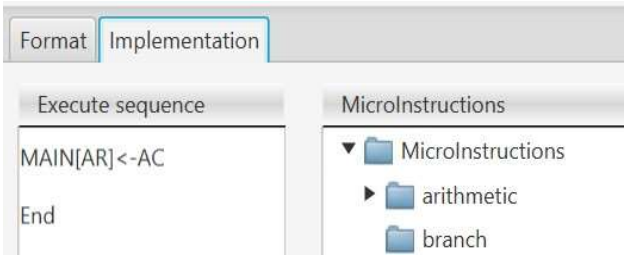
1. Go to Modify and click on Machine Instructions

2. Create four new instructions INP, OUT, HLT, STA, with their respective format and implementation.

FORMATS :

- Same as practical 3 **IMPLEMETATIONS :**

- Same as practical 3
- For STA , drag the file MAIN[AR]<-AC from memory access and end file from end and drop to execute sequence column



- Press ctrl+S and save the text file with .a extension
- Go to Execute option , click on clear, load, assemble, and run option for executing the program .

OUTPUT will be:

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 9
Output:  9
EXECUTION HALTED DUE TO AN EXCEPTION: No opcode matched the bits in the register: IR.
```

Addr Dec Data Dec

MAIN

Addr	Data
0	63488
1	16388
2	62464
3	28673
4	9
5	0
6	0

Data Dec

Registers

Name	Width	Data
AC	16	9
AR	12	9
DR	16	0
E	1	0
I	1	0
INP	8	0
IR	16	9
OUT	8	0
PC	12	5
S	1	-1
TR	16	0

AIM : Write an assembly program for simulating following memory-reference instructions

. (IV) BUN

PROGRAM CODE :

```
bun.a
1 INP
2 BUN K
3 INP
4 K: OUT
5 HLT
6
7
```

PROCEDURE :

1. Go to Modify and click on Machine Instructions
2. Create four new instructions INP, OUT, HLT, BUN, with their respective format and implementation.

FORMATS :

- Same as practical 3
- For BUN , set Opcode to 0x2 and drag OP and then Address from the right side and drop it down to Opcode.

IMPLEMETATIONS :

- Same as practical 3
- For BUN, double click on TransferRtoR and create a new field PC<-AR as follows:

Edit Microinstructions

Type of Microinstruction: TransferRtoR

name	source	srcStartBit	dest	destStartBit	numBits
AC<-DR	DR	0	AC	0	16
AR<-IR(0-11)	IR	0	AR	0	12
AR<-PC	PC	0	AR	0	12
PC<-AR	AR	0	PC	0	12

Then drag the file PC<-AR from TransferRtoR file and then drag end from end file and drop to execute sequence column.

Format | Implementation

Execute sequence

PC<-AR

End

MicroInstructions

MicroInstructions

arithmetic

branch

- Press ctrl+S and save the text file with .a extension
- Go to Execute option , click on clear, load, assemble, and run option for executing the program .

OUTPUT will be :

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 8
Output:  8
EXECUTION HALTED DUE TO AN EXCEPTION: No opcode matched the bits in the register: IR.
```

AIM : Write an assembly program for simulating following memory-reference instructions

. (V) ISZ

PROGRAM CODE :

```
ISZ.a
1 ISZ 008
2 OUT
3 HLT
4
```

PROCEDURE :

1. Go to Modify and click on Machine Instructions
2. Create four new instructions ISZ, OUT, HLT with their respective format and implementation.

FORMATS :

- Same as practical 3
- For ISZ , set Opcode to 0x6 and drag OP and then Address from the right side and drop it down to Opcode.

IMPLEMETATIONS :

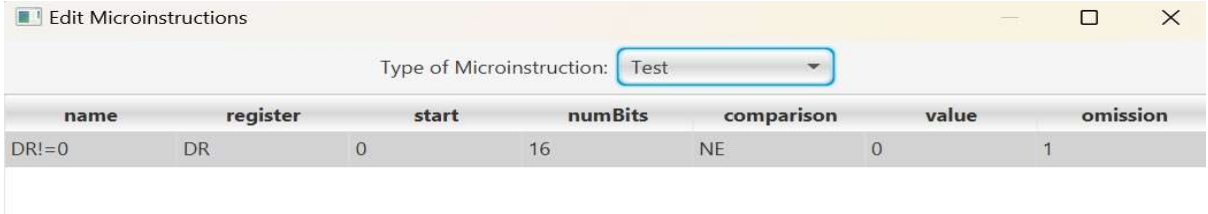
- Same as practical 3
- For ISZ, double click on INCREMENT and create a new field INCR-DR as follows:

Edit Microinstructions

Type of Microinstruction: Increment

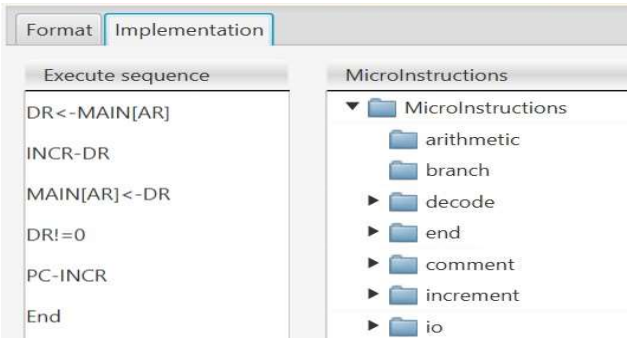
name	register	overflowBit	carryBit	delta
INCR-DR	DR	(none)	(none)	1
PC-INCR	PC	(none)	(none)	1

Then click on TEST to create a new field DR!=0 as follows:



name	register	start	numBits	comparison	value	omission
DR!=0	DR	0	16	NE	0	1

Then drag the file DR<-MAIN[AR] from memory access, drag INCR-DR file from increment , drag MAIN[AR]<-DR file from memory access, drag DR!=0 file from TEST, drag PC-INCR file from increment and then drag end from end file and drop to execute sequence column.



- Press ctrl+S and save the text file with .a extension
- Go to Execute option , click on clear, load, assemble, and run option for executing the program .

OUTPUT will be :

```
EXECUTING...
Output:  0
EXECUTION HALTED DUE TO AN EXCEPTION: No opcode matched the bits in the register: IR.
```

MAIN

Addr	Data
0	24584
1	62464
2	28673
3	0
4	0
5	0
6	0
7	0
8	1
9	0

PRACTICAL 7

AIM : Write an assembly language program to simulate the machine for following register reference instructions and determine the contents of AC, E, PC, AR and IR registers in decimal after the execution:

(I) CLA

PROGRAM CODE :

```
cla.a
1 CLA
2 HLT
3
```

PROCEDURE :

1. Go to Modify and click on Machine Instructions
2. Create four new instructions CLA, HLT with their respective format and implementation.

FORMATS :

- Same as practical 3
- For CLA , set Opcode to 0x7800 and drag register from the right side and drop it down to Opcode.

IMPLEMETATIONS :

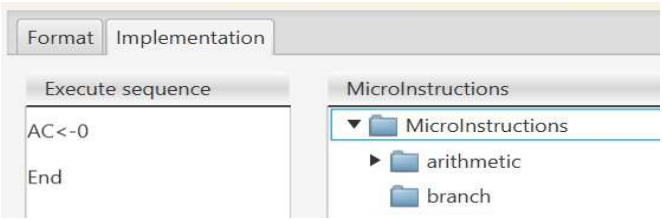
- Same as practical 3
- For CLA, double click on set and create new field AC<-0 as follows:

Edit Microinstructions

Type of Microinstruction: Set

name	register	start	numBits	value
AC<-0	AC	0	16	0

Then drag file AC<-0 from set file then drag drag end file from END and drop to Execute Sequence column.



- Press ctrl+S and save the text file with .a extension
- Go to Execute option , click on clear, load, assemble, and rum option for executing the program .

OUTPUT will be:

Data Dec

Registers

Name	Width	Data
AC	16	5
AR	12	0
DR	16	0
E	1	0

Data Dec

Registers

Name	Width	Data
AC	16	0
AR	12	0
DR	16	0
E	1	0

AIM : Write an assembly language program to simulate the machine for following register reference instructions and determine the contents of AC, E, PC, AR and IR registers in decimal after the execution:

(II) CMA

PROGRAM CODE :

cma.a

```
CMA
```

PROCEDURE :

1. Go to Modify and click on Machine Instructions

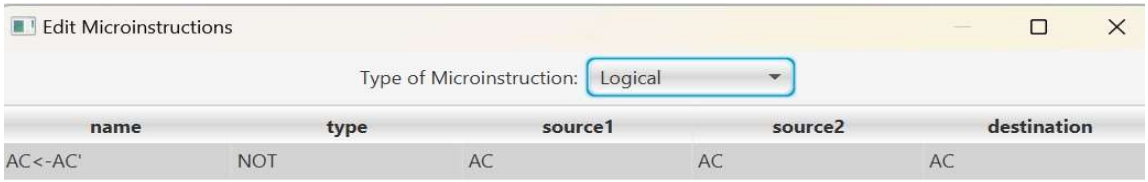
2. Create new instructions CMA with their respective format and implementation.

FORMATS :

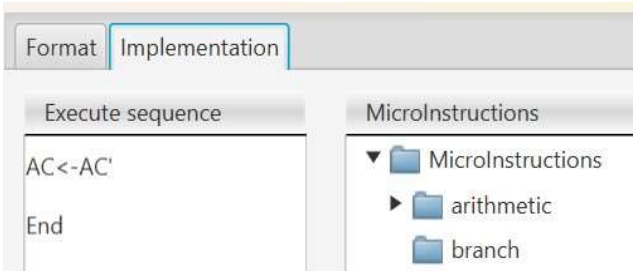
- For CMA , set Opcode to 0x7200 and drag register from the right side and drop it down to Opcode.

IMPLEMETATIONS :

- For CMA, double click on logical and create new field AC<-AC’ as follows:

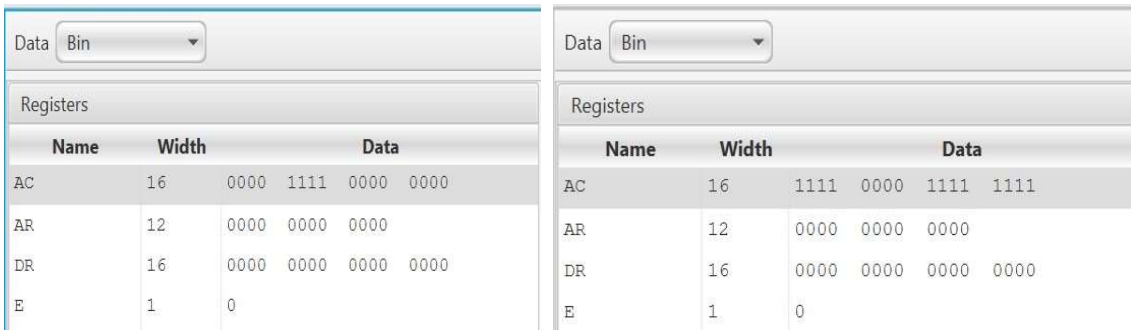


Then drag file AC<-AC' file from logical and at last drag end file from END and drop to Execute Sequence column.



- Press ctrl+S and save the text file with .a extension
- Go to Execute option , click on clear, load, assemble, and rum option for executing the program .

OUTPUT will be:



AIM : Write an assembly language program to simulate the machine for following register reference instructions and determine the contents of AC, E, PC, AR and IR registers in decimal after the execution:

(III) CME

PROGRAM CODE :

PROCEDURE :

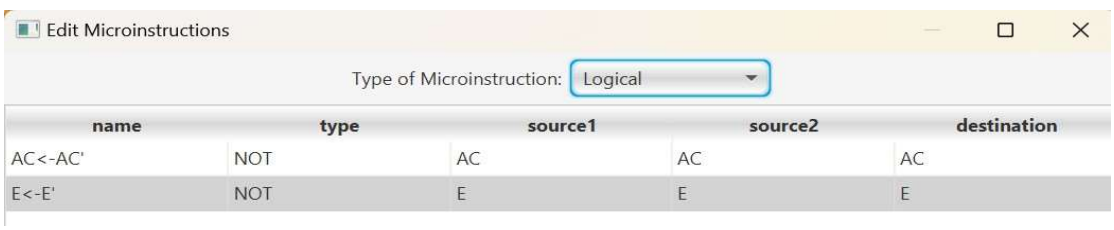
1. Go to Modify and click on Machine Instructions
2. Create four new instructions CME, HLT with their respective format and implementation.

FORMATS :

- Same as practical 3
- For CME , set Opcode to 0x7100 and drag register from the right side and drop it down to Opcode.

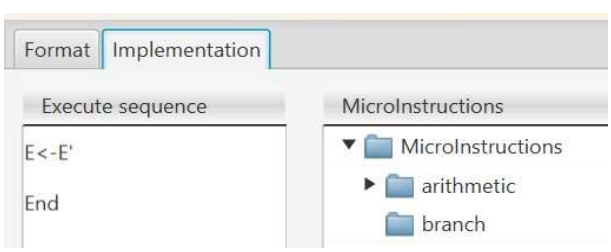
IMPLEMETATIONS :

- Same as practical 3
- For CLA, double click on logical and create new field E<-E' as follows:



name	type	source1	source2	destination
AC<-AC'	NOT	AC	AC	AC
E<-E'	NOT	E	E	E

Then drag file E<-E' from logic file then drag end file from END and drop to Execute Sequence column.



- Press ctrl+S and save the text file with .a extension
- Go to Execute option , click on clear, load, assemble, and rum option for executing the program .

OUTPUT will be:

Data Bin

Registers

Name	Width	Data			
AC	16	0000	0000	0000	0000
AR	12	0000	0000	0000	
DR	16	0000	0000	0000	0000
E	1	1			
I	1	0			
INP	8	0000	0000		

Data Bin

Registers

Name	Width	Data			
AC	16	0000	0000	0000	0000
AR	12	0000	0000	0000	
DR	16	0000	0000	0000	0000
E	1	0			
I	1	0			

AIM : Write an assembly language program to simulate the machine for following register reference instructions and determine the contents of AC, E, PC, AR and IR registers in decimal after the execution:

(IV) HLT

PROGRAM CODE :

HLT.A ×

```
1 HLT
```

PROCEDURE :

1. Go to Modify and click on Machine Instructions

2. Create new instructions HLT with their respective format and implementation.

FORMATS :

- For HLT , set Opcode to 0x7001 and drag register from the right side and drop it down to Opcode.

IMPLEMETATIONS :

- For HLT, drag file HALT from setCondBit then drag drag end file from END and drop to Execute Sequence column.

- Press ctrl+S and save the text file with .a extension
- Go to Execute option , click on clear, load, assemble, and rum option for executing the program .

OUTPUT will be:

```
EXECUTING...
EXECUTION HALTED DUE TO AN EXCEPTION: No opcode matched the bits in the register: IR.
```

PRACTICAL 8

AIM Write an assembly language program to simulate the machine for following register reference instructions and determine the contents of AC, E, PC, AR and IR registers in decimal after the execution:

(I) INC

PROGRAM CODE :

INC.a

```
INP
INC
OUT
HLT
```

PROCEDURE :

1. Go to Modify and click on Machine Instructions
2. Create four new instructions INP, OUT, HLT, INC with their respective format and implementation.

FORMATS :

- Same as practical 3
- For INC , set Opcode to 0x7020 and drag register from the right side and drop it down to Opcode.

IMPLEMETATIONS :

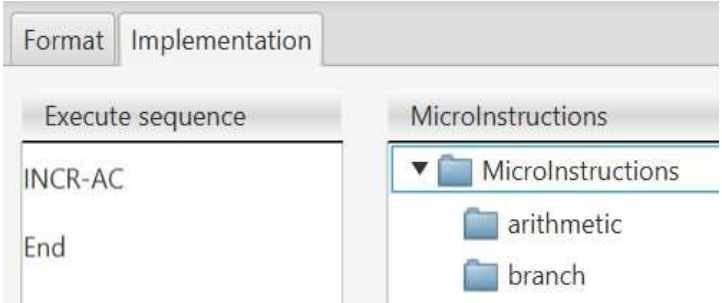
- Same as practical 3
- For INC, double click on INCREMENT and create new field INCR-AC.

Edit Microinstructions

Type of Microinstruction: Increment

name	register	overflowBit	carryBit	delta
INCR-AC	AC	(none)	(none)	1
INCR-DR	DR	(none)	(none)	1
PC-INCR	PC	(none)	(none)	1

Then drag file INCR-AC from INCREMENT , after that drag end file from END and drop to Execute Sequence column.



- Press ctrl+S and save the text file with .a extension
- Go to Execute option , click on clear, load, assemble, and rum option for executing the program .

OUTPUT will be:

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 4
Output:  5
EXECUTION HALTED DUE TO AN EXCEPTION: No opcode matched the bits in the register: IR.
```

AIM : Write an assembly language program to simulate the machine for following register reference instructions and determine the contents of AC, E, PC, AR and IR registers in decimal after the execution:

(II) SPA

PROGRAM CODE :

SPA.a

```
INP
SPA
OUT
HLT
```

PROCEDURE :

1. Go to Modify and click on Machine Instructions
2. Create four new instructions INP, OUT, HLT, SPA with their respective format and implementation.

FORMATS :

- Same as practical 3

- For SPA , set Opcode to 0x7010 and drag register from the right side and drop it down to Opcode.

IMPLEMETATIONS :

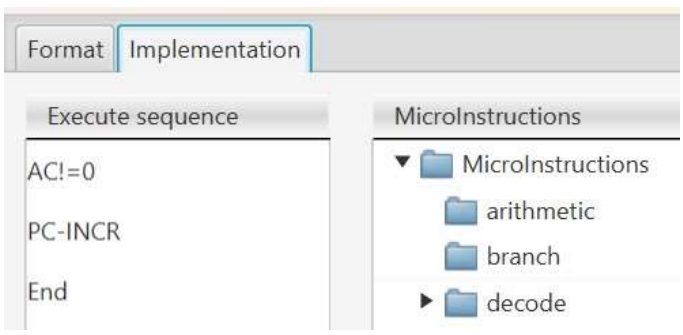
- Same as practical 3
- For SPA, double click on TEST and create new field AC!=0.

Edit Microinstructions

Type of Microinstruction: Test

name	register	start	numBits	comparison	value	omission
AC!=0	AC	0	1	NE	0	1
DR!=0	DR	0	16	NE	0	1

Then drag file AC!=0 from TEST , after that drag PC-INCR from INCREMENT file and at last drag end file from END and drop to Execute Sequence column.



- Press ctrl+S and save the text file with .a extension
- Go to Execute option , click on clear, load, assemble, and rum option for executing the program .

OUTPUT will be:

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: -1
Output:  -1
EXECUTION HALTED DUE TO AN EXCEPTION: No opcode matched the bits in the register: IR.
```

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 6
EXECUTION HALTED DUE TO AN EXCEPTION: No opcode matched the bits in the register: IR.
```

AIM : Write an assembly language program to simulate the machine for following register reference instructions and determine the contents of AC, E, PC, AR and IR registers in decimal after the execution:

(III) SNA

PROGRAM CODE :

SNA.a

```
INP
SNA
OUT
HLT
```

PROCEDURE :

1. Go to Modify and click on Machine Instructions
2. Create four new instructions INP, OUT, HLT, SNA with their respective format and implementation.

FORMATS :

- Same as practical 3
- For SNA , set Opcode to 0x7008 and drag register from the right side and drop it down to Opcode.

IMPLEMETATIONS :

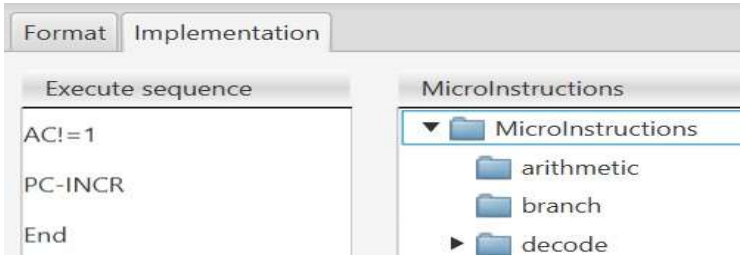
- Same as practical 3
- For SNA, double click on TEST and create new field AC!=1.

Edit Microinstructions

Type of Microinstruction: Test

name	register	start	numBits	comparison	value	omission
AC!=1	AC	0	1	NE	1	1

Then drag file AC!=1 from TEST , after that drag PC-INCR from INCREMENT file and at last drag end file from END and drop to Execute Sequence column.



- Press ctrl+S and save the text file with .a extension
- Go to Execute option , click on clear, load, assemble, and rum option for executing the program .

OUTPUT will be:

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: -3
EXECUTION HALTED DUE TO AN EXCEPTION: No opcode matched the bits in the register: IR.
```

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 8
Output:  8
EXECUTION HALTED DUE TO AN EXCEPTION: No opcode matched the bits in the register: IR.
```

AIM : Write an assembly language program to simulate the machine for following register reference instructions and determine the contents of AC, E, PC, AR and IR registers in decimal after the execution:

(IV) SZE

PROGRAM CODE :

sze.a

```
SZE
OUT
HLT
```

PROCEDURE :

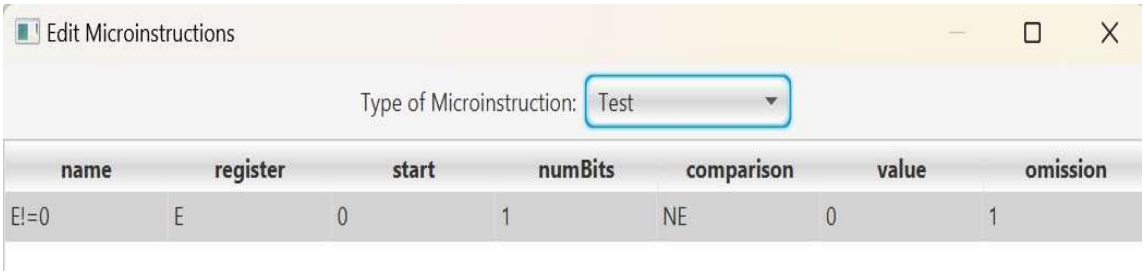
1. Go to Modify and click on Machine Instructions
2. Create three new instructions OUT, HLT, SZE with their respective format and implementation.

FORMATS :

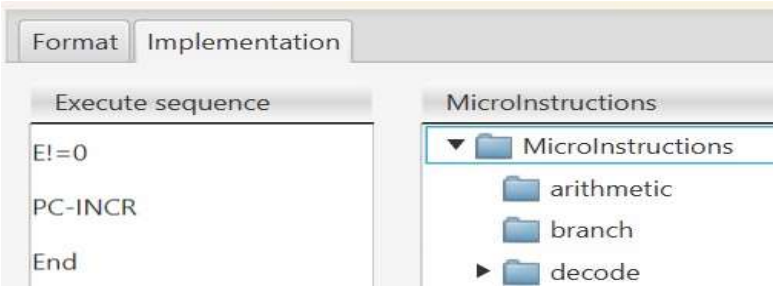
- Same as practical 3
- For SZE , set Opcode to 0x7002 and drag register from the right side and drop it down to Opcode.

IMPLEMETATIONS :

- Same as practical 3
- For SNA, double click on TEST and create new field E!=0.



Then drag file E!=0 from TEST , after that drag PC-INCR from INCREMENT file and at last drag end file from END and drop to Execute Sequence column.



- Press ctrl+S and save the text file with .a extension
- Go to Execute option , click on clear, load, assemble, and rum option for executing the program .

OUTPUT will be:

If E=0 then there is no output shown and execution is halted but if value of E !=0 then output 0 is shown .

(1) case if E=0

Name	Width	Data
AC	16	0
AR	12	0
DR	16	0
E	1	0
I	1	0
INP	8	0
IR	16	0
OUT	8	0
PC	12	4
S	1	-1
TR	16	0

```
EXECUTING...
EXECUTION HALTED DUE TO AN EXCEPTION: No opcode matched the bits in the register: IR.
```

(2) case if E!=0

Name	Width	Data
AC	16	0
AR	12	0
DR	16	0
E	1	1
I	1	0
INP	8	0
IR	16	0
OUT	8	0
PC	12	4
S	1	-1
TR	16	0

```
EXECUTING...
Output:  0
EXECUTION HALTED DUE TO AN EXCEPTION: No opcode matched the bits in the register: IR.
```

PRACTICAL 9

AIM : Write an assembly language program to simulate the machine for following register reference instructions and determine the contents of AC, E, PC, AR and IR registers in decimal after the execution:

(I) CIR

PROGRAM CODE :

CIR.a

```
INP
CIR
OUT
HLT
```

PROCEDURE :

1. Go to Modify and click on Machine Instructions
2. Create four new instructions INP, OUT, HLT, CIR with their respective format and implementation.

FORMATS :

- Same as practical 3
- For CIR , set Opcode to 0x7080 and drag register from the right side and drop it down to Opcode.

IMPLEMETATIONS :

- Same as practical 3
- For CIR, double click on transferRtoR and create new field E<-AC(15) and AC(0)<-E.

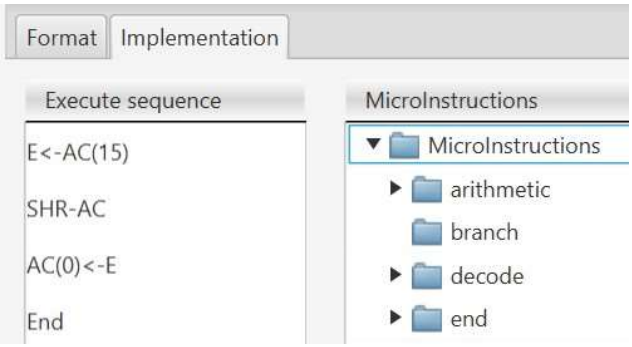
Edit Microinstructions

Type of Microinstruction: TransferRtoR

name	source	srcStartBit	dest	destStartBit	numBits
AC(0)<-E	E	0	AC	0	1
AR<-IR(0-11)	IR	0	AR	0	12
AR<-PC	PC	0	AR	0	12
E<-AC(15)	AC	15	E	0	1

Then click on Shift and create new field : SHR-AC.

Then drag file E<-AC(15) from transferRtoR , after that drag SHR-AC from shift file then drag AC(0)<-E from transferRtoR and at last drag end file from END and drop to Execute Sequence column.



- Press ctrl+S and save the text file with .a extension
- Go to Execute option , click on clear, load, assemble, and rum option for executing the program .

OUTPUT will be:

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 8
Output:  4
EXECUTION HALTED DUE TO AN EXCEPTION: No opcode matched the bits in the register: IR.
```

AIM : Write an assembly language program to simulate the machine for following register reference instructions and determine the contents of AC, E, PC, AR and IR registers in decimal after the execution:

(II) CIL

PROGRAM CODE :

```
CIL.a
INP
CIL
OUT
HLT
```

PROCEDURE :

1. Go to Modify and click on Machine Instructions
2. Create four new instructions INP, OUT, HLT, CIL with their respective format and implementation.

FORMATS :

- Same as practical 3
- For CIL , set Opcode to 0x7040 and drag register from the right side and drop it down to Opcode.

IMPLEMETATIONS :

- Same as practical 3
- For CIL, double click on transferRtoR and create new field E<-AC(0) and AC(15)<-E

Edit Microinstructions

Type of Microinstruction: TransferRtoR

name	source	srcStartBit	dest	destStartBit	numBits
AC(15)<-E	E	0	AC	15	0
AR<-IR(0-11)	IR	0	AR	0	12
AR<-PC	PC	0	AR	0	12
E<-AC(0)	AC	0	E	0	1

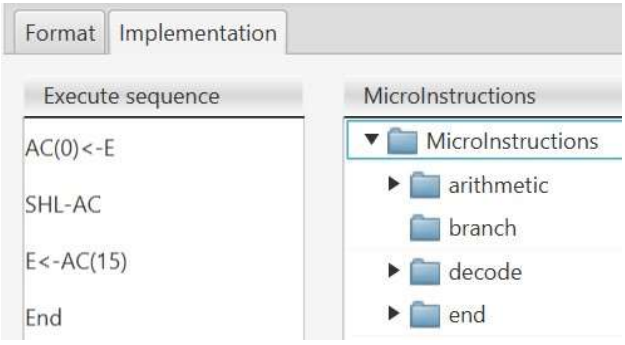
Then click on Shift and create new field : SHL-AC.

Edit Microinstructions

Type of Microinstruction: Shift

name	source	destination	type	direction	distance
SHL-AC	AC	AC	cyclic	left	1
SHR-AC	AC	AC	cyclic	right	1

Then drag file E<-AC(0) from transferRtoR , after that drag SHL-AC from shift file then drag AC(15)<-E from transferRtoR and at last drag end file from END and drop to Execute Sequence column.



- Press ctrl+S and save the text file with .a extension
- Go to Execute option , click on clear, load, assemble, and rum option for executing the program .

OUTPUT will be:

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 4
Output:  8
EXECUTION HALTED DUE TO AN EXCEPTION: No opcode matched the bits in the register: IR.
```

PRACTICAL 10

AIM : write an assembly program that reads in integers and adds them together; until a negative non-zero number is read in. Then it outputs the sum (not including the last number).

PROGRAM CODE :

SUMN.a

```
START: INP
JUMN DONE
ADD SUM
STA SUM
JUMP START
DONE: LDA SUM
OUT
HLT
SUM: .data 2 0
```

PROCEDURE :

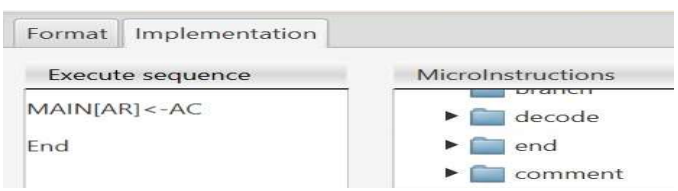
1. Go to Modify and click on Machine Instructions
2. Create eight new instructions INP, OUT, HLT, STA, ADD, JUMN, JUMP, LDA with their respective format and implementation.

FORMATS :

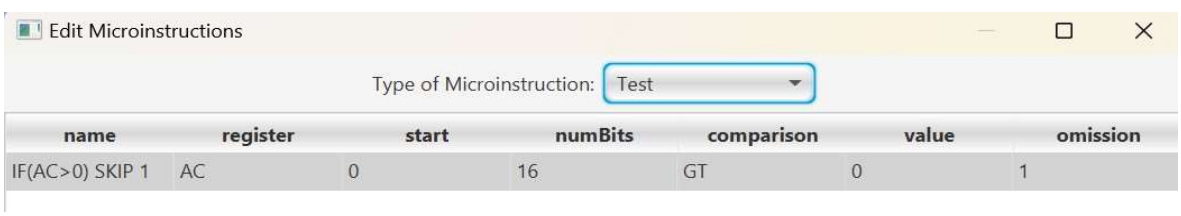
- Same as practical 3
- For LDA , set Opcode to 0xA and drag OP and then Address from the right side and drop it down to Opcode.
- For JUMP , set Opcode to 0x4 and drag OP and then Address from the right side and drop it down to Opcode.
- For JUMN , set Opcode to 0x0 and drag OP and then Address from the right side and drop it down to Opcode.

IMPLEMETATIONS :

- Same as practical 3
- For LDA , drag MAIN[AR]<-AC from memory access and end file from end and drop to execute sequence column.



- For JUMN , double click on TEST and create new field IF(AC>0) SKIP 1 .



name	register	start	numBits	comparison	value	omission
IF(AC>0) SKIP 1	AC	0	16	GT	0	1

Then double click on TransferRtoR and create new field PC<-AR.

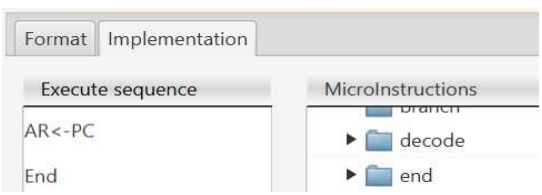
name	source	srcStartBit	dest	destStartBit	numBits
AR<-IR(0-11)	IR	0	AR	0	12
AR<-PC	PC	0	AR	0	12
PC<-AR	PC	0	AR	0	12

Then drag IF(AC>0) SKIP 1 from TEST, then drag the file PC<-AR from TransferRtoR and end file from end and drop to execute sequence column

- For JUMP , double click on TransferRtoR and create new field. AR<-PC.

name	source	srcStartBit	dest	destStartBit	numBits
AR<-IR(0-11)	IR	0	AR	0	12
AR<-PC	PC	0	AR	0	12
PC<-AR	PC	0	AR	0	12

Then drag the file AR<-PC from TransferRtoR and end file from end and drop to execute sequence column



- Press ctrl+S and save the text file with .a extension
- Go to Execute option , click on clear, load, assemble, and run option for executing the program .

OUTPUT will be:

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 9
Output:  9
Enter Inputs, the first of which must be an Integer: 8
Output:  17
Enter Inputs, the first of which must be an Integer: -17
Output:  0
Enter Inputs, the first of which must be an Integer: -2
```

```
Enter Inputs, the first of which must be an Integer: -2
Output:  -2
EXECUTION HALTED DUE TO AN EXCEPTION: No opcode matched the bits in the register: IR.
```

PRACTICAL 11

AIM : Write an assembly program that reads in integers and adds them together; until zero is read in. Then it outputs the sum.

PROGRAM CODE :

```
*111.a ×
START: INP
JMPZ DONE
ADD SUM
STA SUM
JUMP START
DONE: LDA SUM
OUT
HLT
SUM: .data 2 0
```

PROCEDURE :

1. Go to Modify and click on Machine Instructions
2. Create eight new instructions INP, OUT, HLT, STA, ADD, JMPZ, JUMP, LDA with their respective format and implementation.

FORMATS :

- Same as practical 10
- For JMPZ , set Opcode to 0x0 and drag OP and then Address from the right side and drop it down to Opcode.

IMPLEMETATIONS :

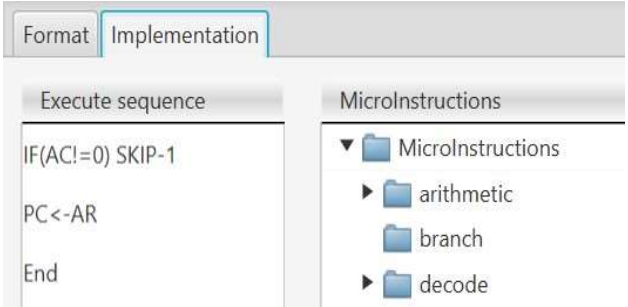
- Same as practical 10
- For JMPZ , double click on TEST and create new field IF(AC!=0) SKIP 1 .

Edit Microinstructions

Type of Microinstruction: Test

name	register	start	numBits	comparison	value	omission
IF(AC!=0) SKIP-1	AC	0	16	NE	0	1

Then drag IF(AC!=0) SKIP 1 from TEST, then drag the file PC<-AR from TransferRtoR and end file from end and drop to execute sequence column



- Press ctrl+S and save the text file with .a extension
- Go to Execute option , click on clear, load, assemble, and run option for executing the program .

OUTPUT will be:

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 9
Output:   9
Enter Inputs, the first of which must be an Integer: 8
Output:   17
Enter Inputs, the first of which must be an Integer: 0
Output:   17
Enter Inputs, the first of which must be an Integer:0
```

```
Enter Inputs, the first of which must be an Integer:0
EXECUTION HALTED DUE TO AN EXCEPTION: There are currently no predefined inputs from the user of type long.
```