CSA PRACTICAL FILE

RAMANUJAN COLLEGE

DSC 02: COMPUTER SYSTEM ARCHITECTURE

SEMESTER-1

(2025-26)

Submitted By:-

Name: **Gaohar Imran**

College Roll No. : **25570022**

University Roll No. : **25020570040**

Course: **B.Sc. (Hons) Computer Science**

Submitted To:-

Dr. Kamlesh Kumar Raghuvanshi

Department of Computer Science

Acknowledgement

I would like to take this opportunity to acknowledge everyone who has helped us in every stage of this project.

I am deeply indebted to my computer system architecture professor, **Dr Kamlesh Kumar Raghuvanshi** for his guidance and suggestions in completing this project. The completion of this project was possible under his guidance and support.

I am also very thankful to my parents and my friends who have boosted me up morally with their continuous support.

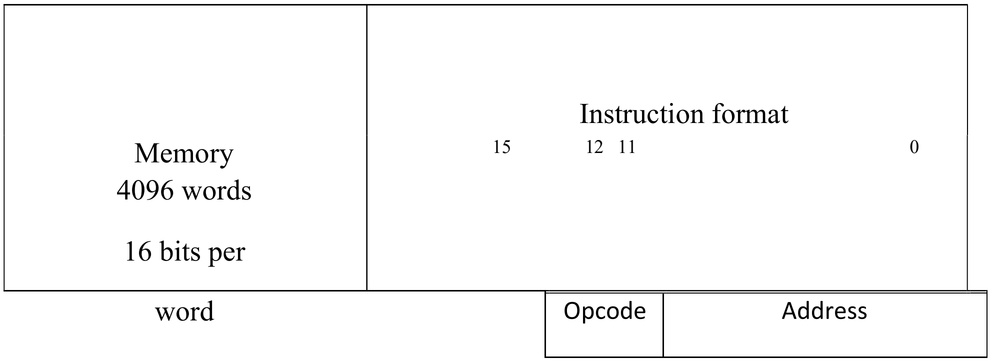
At last but not least, I am very thankful to God almighty for showering his blessings upon me.

INDEX

Q.1. Create a machine based on the following architecture:

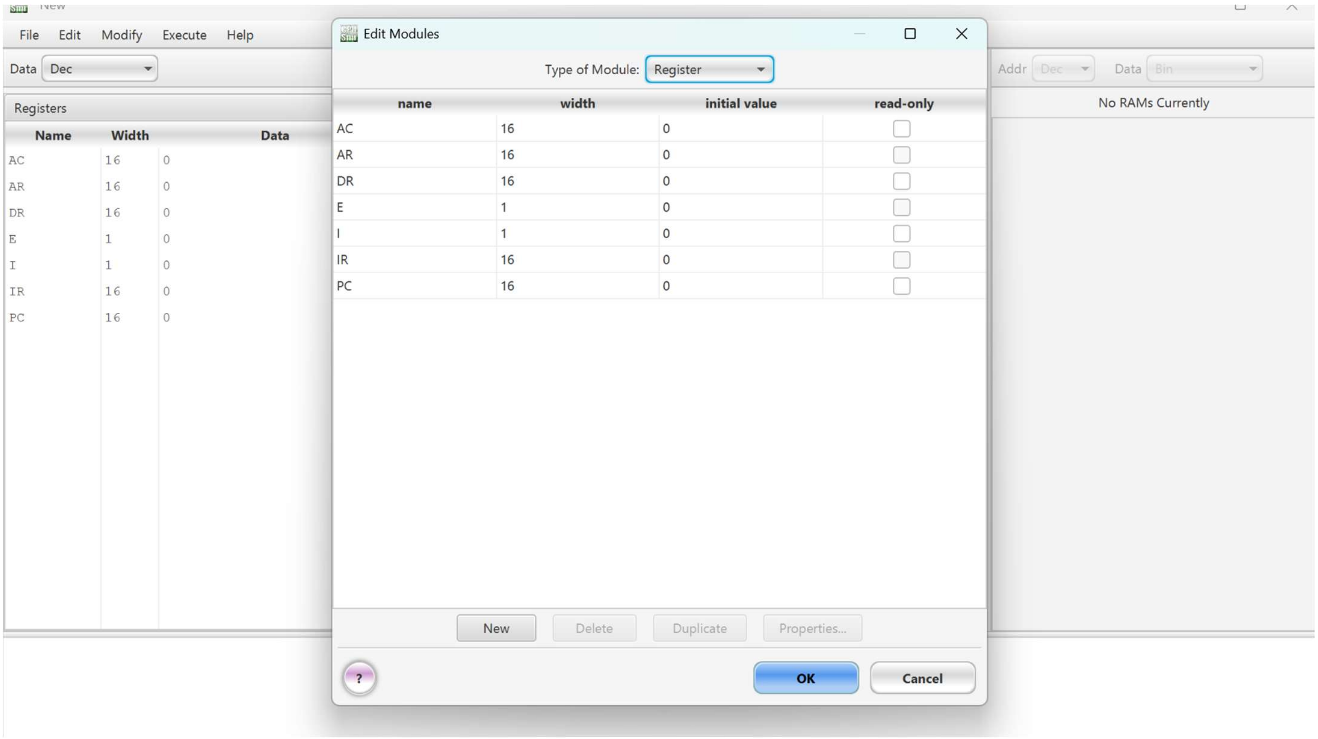
Registers

IR	DR	AC	AR	PC	I	E
16 bits	16 bits	16 bits	12 bits	12 bits	1 bit	1 bit



Basic Computer Instructions

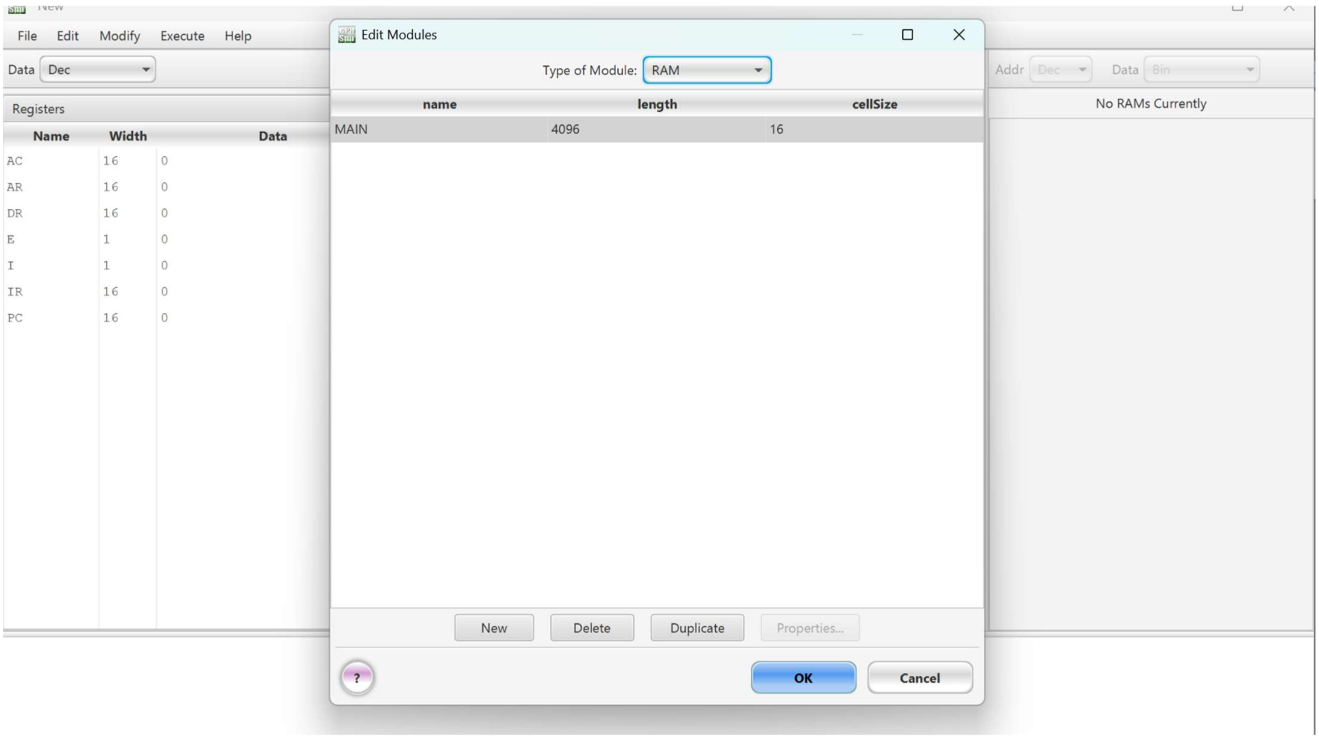
Memory Reference			Register Reference	
Symbol	Hex		Symbol	Hex
AND	0xxx	Direct Addressing	CLA	7800
ADD	1xxx		CLE	7400
LDA	2xxx		CMA	7200
STA	3xxx		CME	7100
BUN	4xxx		CIR	7080
BSA	5xxx		CIL	7040
ISZ	6xxx		INC	7020
AND_I	8xxx	Indirect Addressing	SPA	7010
ADD_I	9xxx		SNA	7008
LDA_I	Axxx		SZA	7004
STA_I	Bxxx		SZE	7002
BUN_I	Cxxx		HLT	7001
BSA_I	Dxxx		INP	F800
ISZ_I	Exxx		OUT	F400



Registers

Name	Width	Data
AC	16	0
AR	16	0
DR	16	0
E	1	0
I	1	0
IR	16	0
PC	16	0

name	width	initial value	read-only
AC	16	0	□
AR	16	0	□
DR	16	0	□
E	1	0	□
I	1	0	□
IR	16	0	□
PC	16	0	□

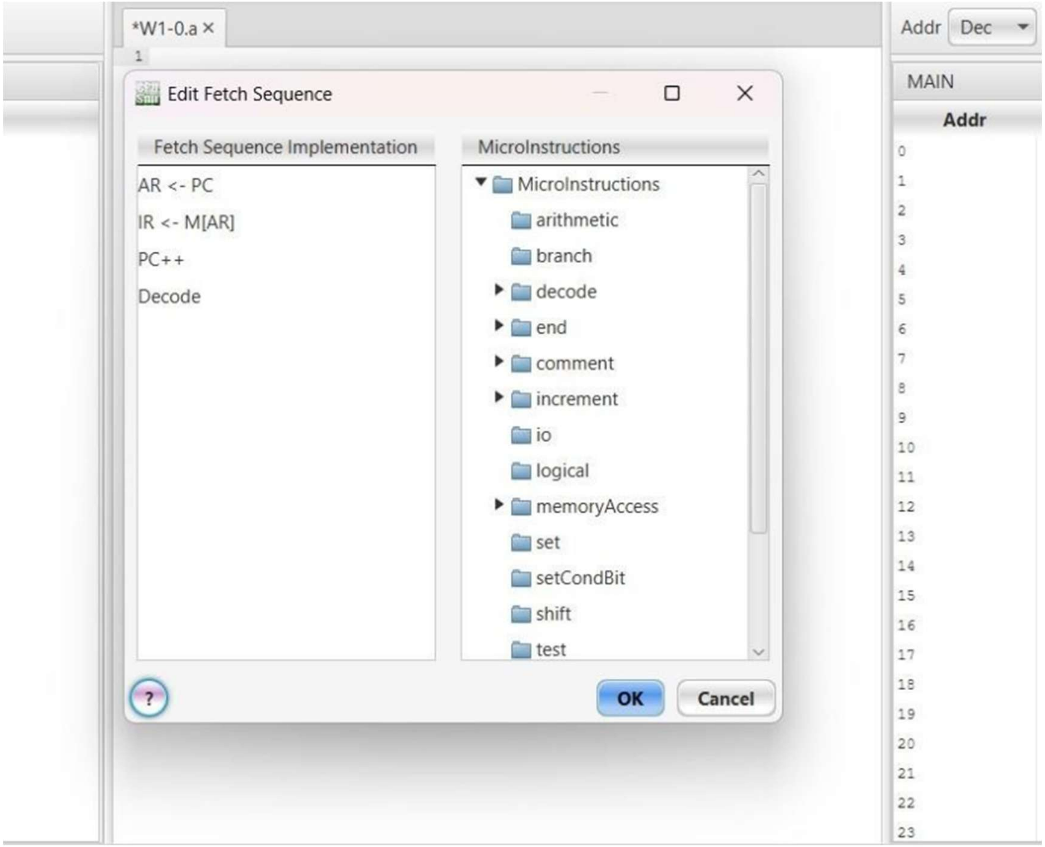


Registers

Name	Width	Data
AC	16	0
AR	16	0
DR	16	0
E	1	0
I	1	0
IR	16	0
PC	16	0

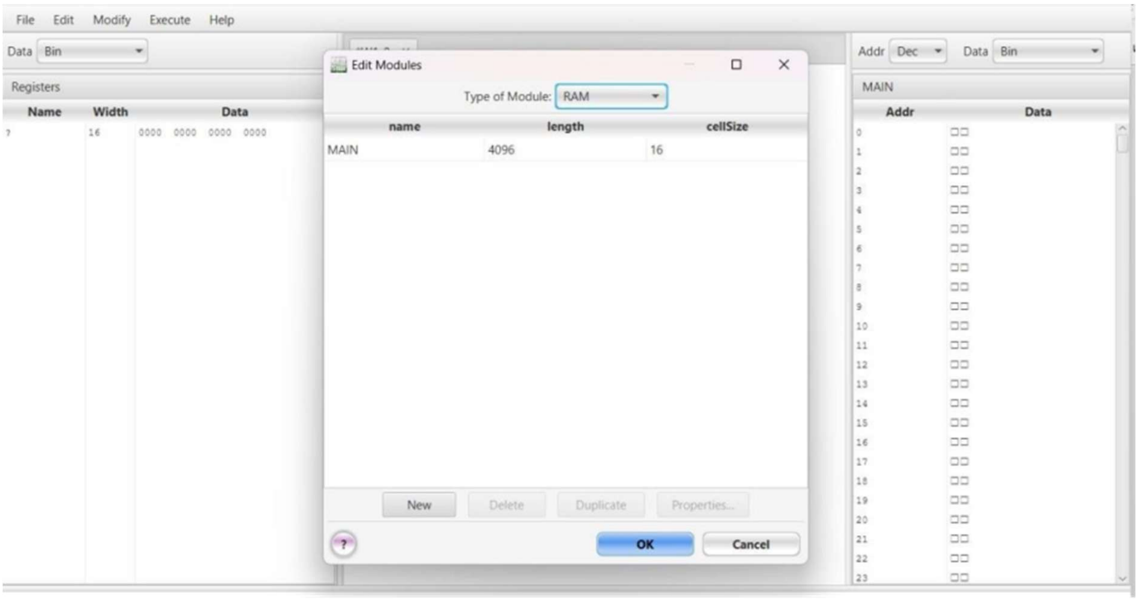
name	length	cellSize
MAIN	4096	16

Q2. Creating a FETCH Routine for the Instruction Cycle.



Edit Fetch Sequence

Fetch Sequence Implementation	MicroInstructions
AR <- PC	▼ MicroInstructions
IR <- M[AR]	arithmetic
PC++	branch
Decode	▶ decode
	▶ end
	▶ comment
	▶ increment
	io
	logical
	▶ memoryAccess
	set
	setCondBit
	shift
	test



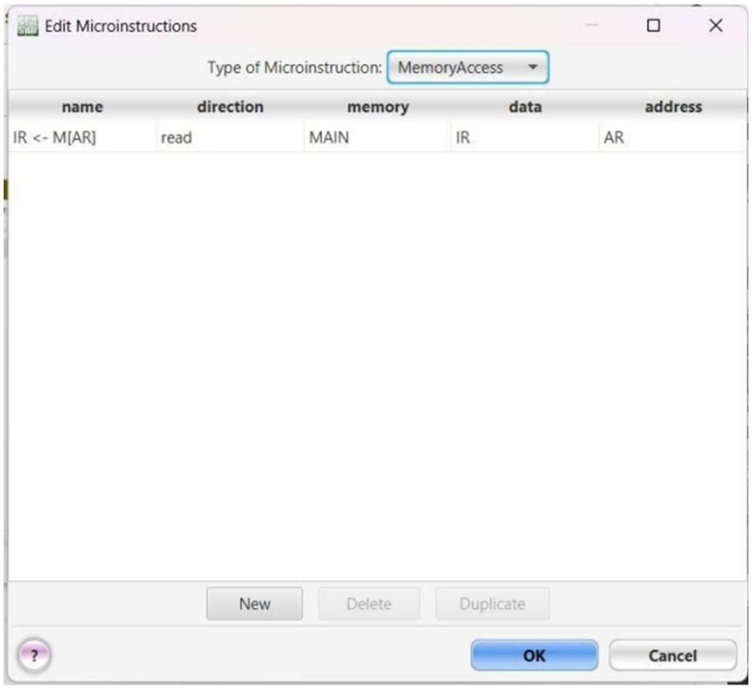
Registers

Name	Width	Data
?	16	0000 0000 0000 0000

Edit Modules

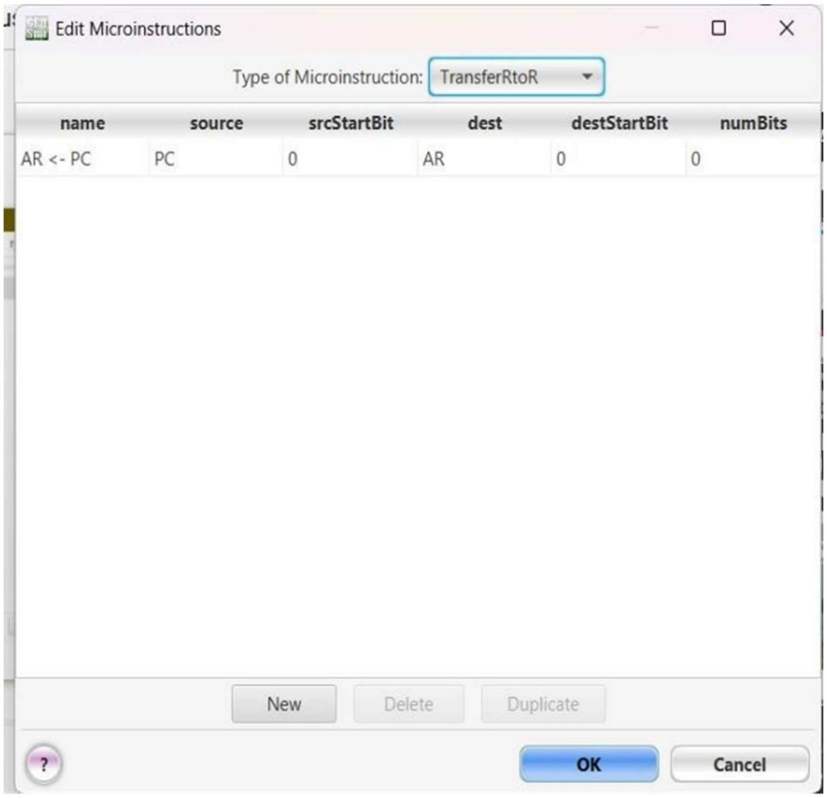
Type of Module: RAM

name	length	cellSize
MAIN	4096	16



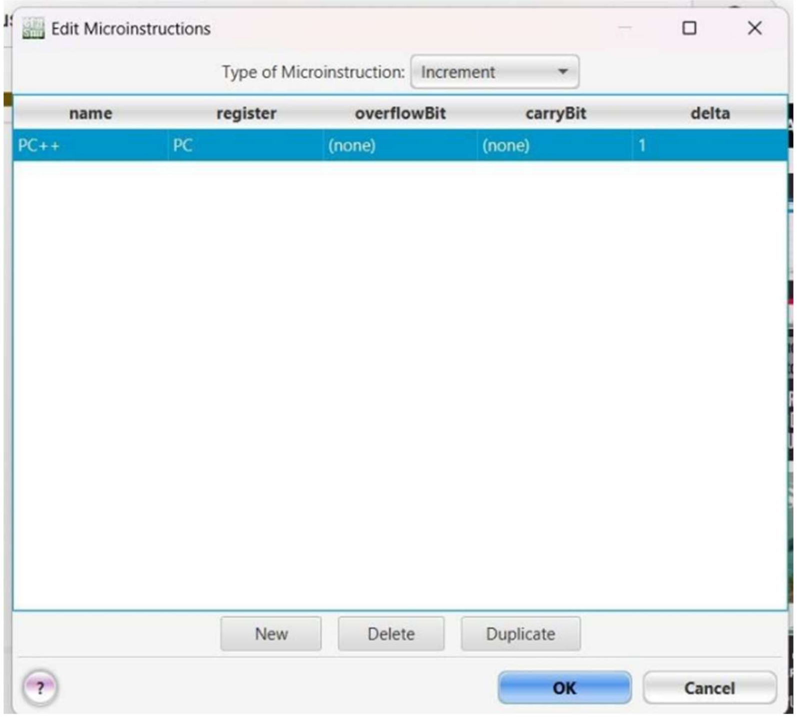
Type of Microinstruction: MemoryAccess

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



Type of Microinstruction: TransferRtoR

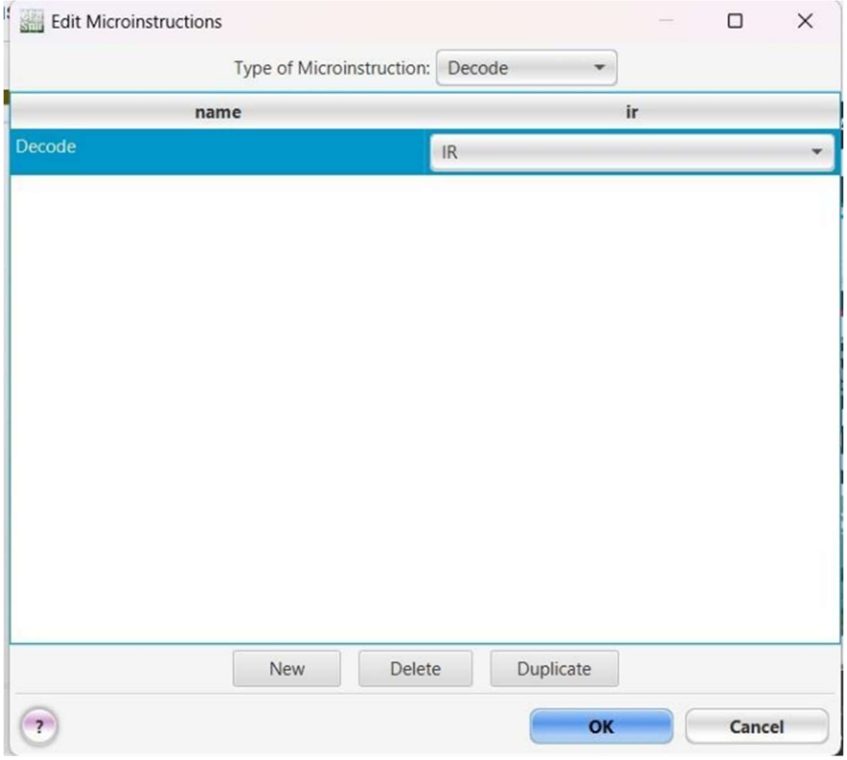
name	source	srcStartBit	dest	destStartBit	numBits
AR <- PC	PC	0	AR	0	0



Edit Microinstructions

Type of Microinstruction: Increment

name	register	overflowBit	carryBit	delta
PC++	PC	(none)	(none)	1

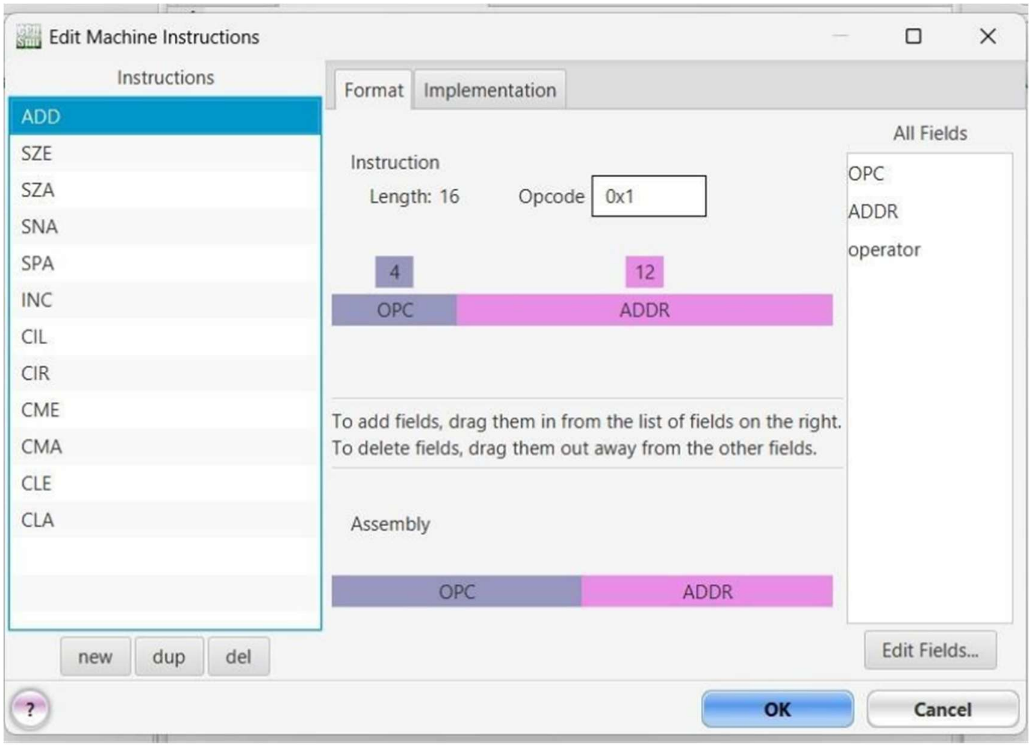


Edit Microinstructions

Type of Microinstruction: Decode

name	ir
Decode	IR

Q. 3. Write an Assembly Program to simulate ADD operation on two user entered numbers.



Edit Machine Instructions

Instructions

- ADD
- SZE
- SZA
- SNA
- SPA
- INC
- CIL
- CIR
- CME
- CMA
- CLE
- CLA

Instruction

Length: 16   Opcode 0x1

4	12
OPC	ADDR

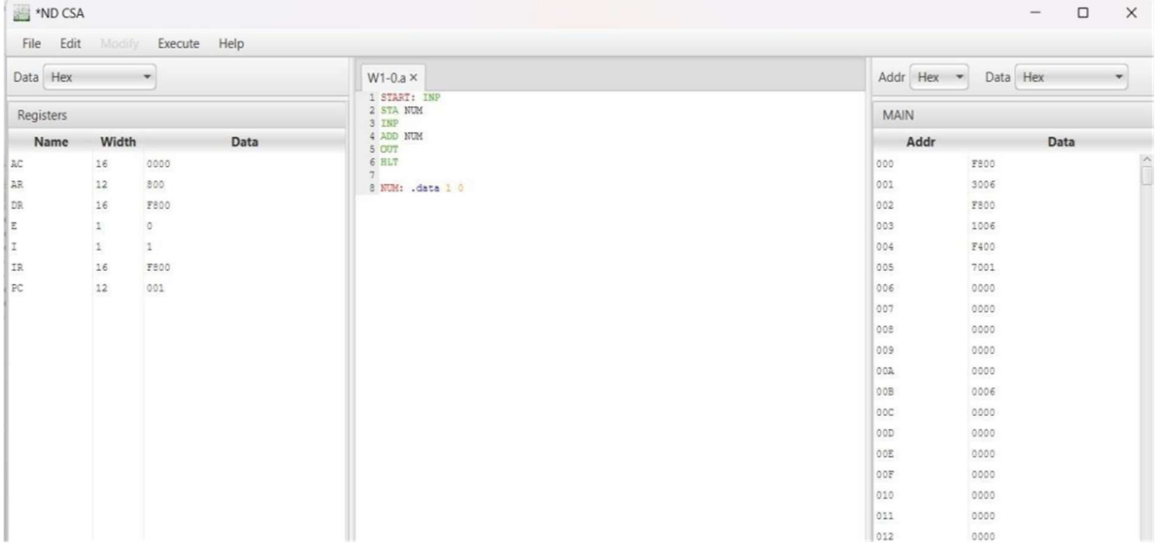
To add fields, drag them in from the list of fields on the right.
To delete fields, drag them out away from the other fields.

Assembly

OPC	ADDR

All Fields

- OPC
- ADDR
- operator



Data: Hex

```
START: INP
STA NUM
INP
ADD NUM
OUT
HLT

NUM: .data 1 0
```

Registers

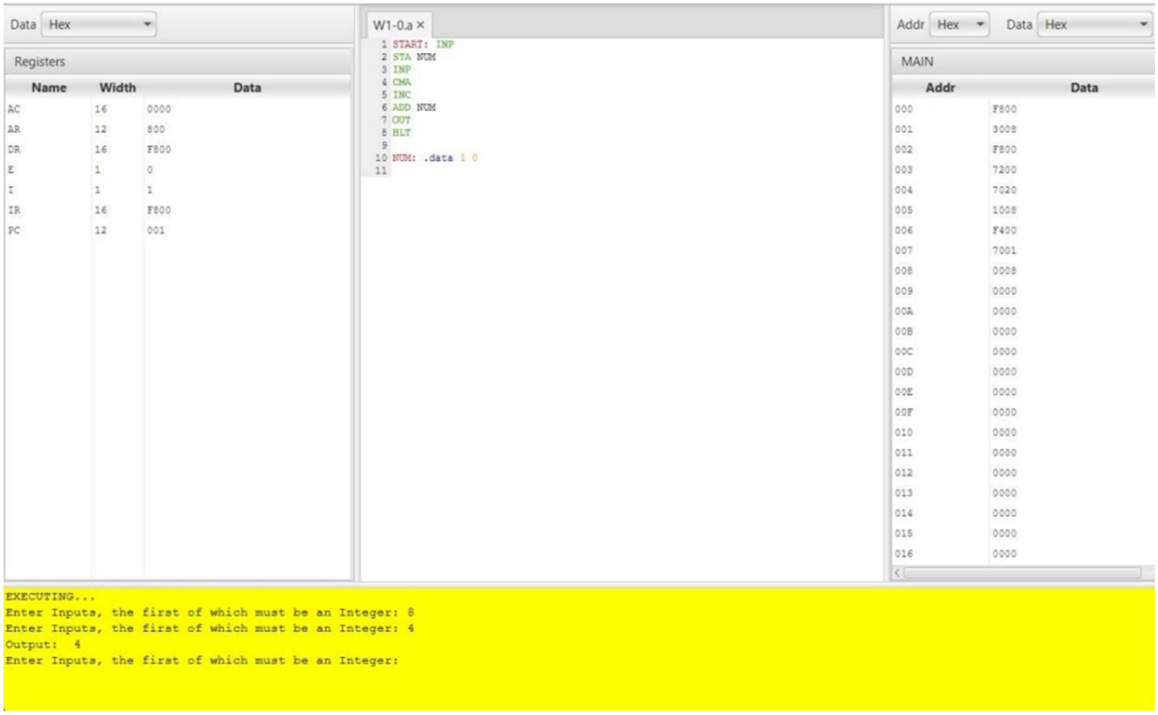
Name	Width	Data
AC	16	0000
AR	12	800
DR	16	F800
E	1	0
I	1	1
IR	16	F800
PC	12	001

Addr: Hex   Data: Hex

MAIN

Addr	Data
000	F800
001	3006
002	F800
003	1006
004	F400
005	7001
006	0000
007	0000
008	0000
009	0000
00A	0000
00B	0006
00C	0000
00D	0000
00E	0000
00F	0000
010	0000
011	0000
012	0000

Q. 4. Write an Assembly Program to simulate SUBTRACT operation on two user entered numbers.



Data: Hex

Registers

Name	Width	Data
AC	16	0000
AR	12	800
DR	16	F800
E	1	0
I	1	1
IR	16	F800
PC	12	001

W1-0.a

```
START: INP
STA NUM
INP
CMA
INC
ADD NUM
OUT
HLT

NUM: .data 1 0

```

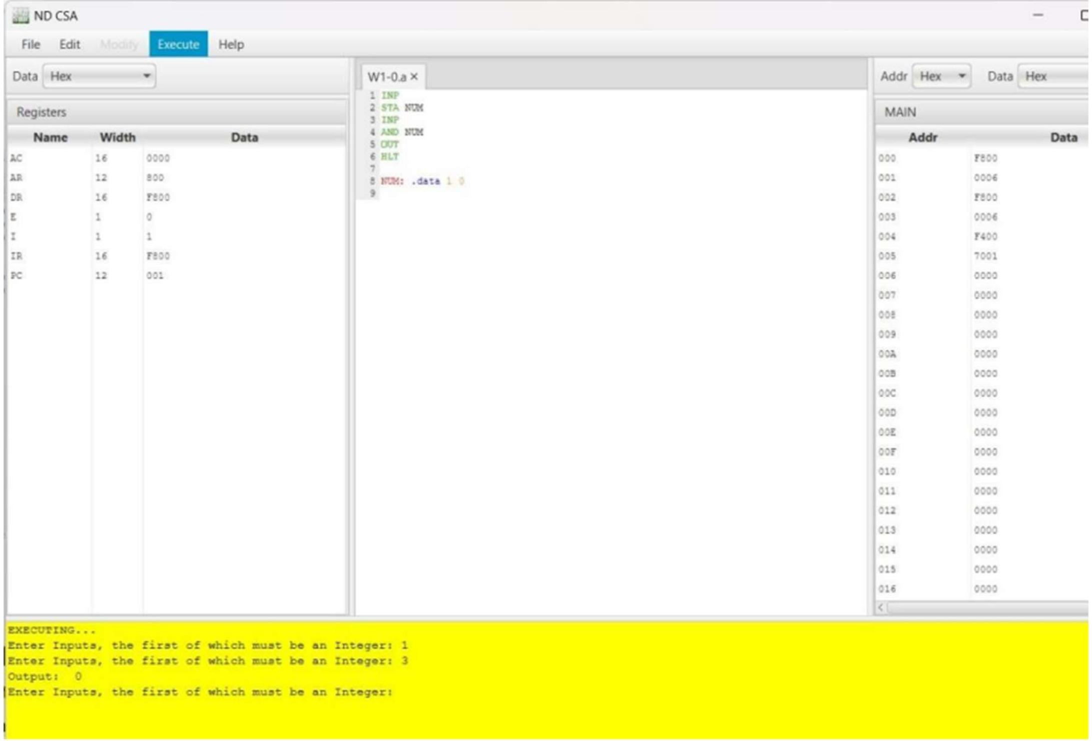
Addr: Hex Data: Hex

MAIN

Addr	Data
000	F800
001	3008
002	F800
003	7200
004	7020
005	1008
006	F400
007	7001
008	0008
009	0000
00A	0000
00B	0000
00C	0000
00D	0000
00E	0000
00F	0000
010	0000
011	0000
012	0000
013	0000
014	0000
015	0000
016	0000

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 8
Enter Inputs, the first of which must be an Integer: 4
Output:  4
Enter Inputs, the first of which must be an Integer:
```

Q. 5. Write an Assembly Program to simulate following logical operations : AND, OR, NOR, NAND, XOR.

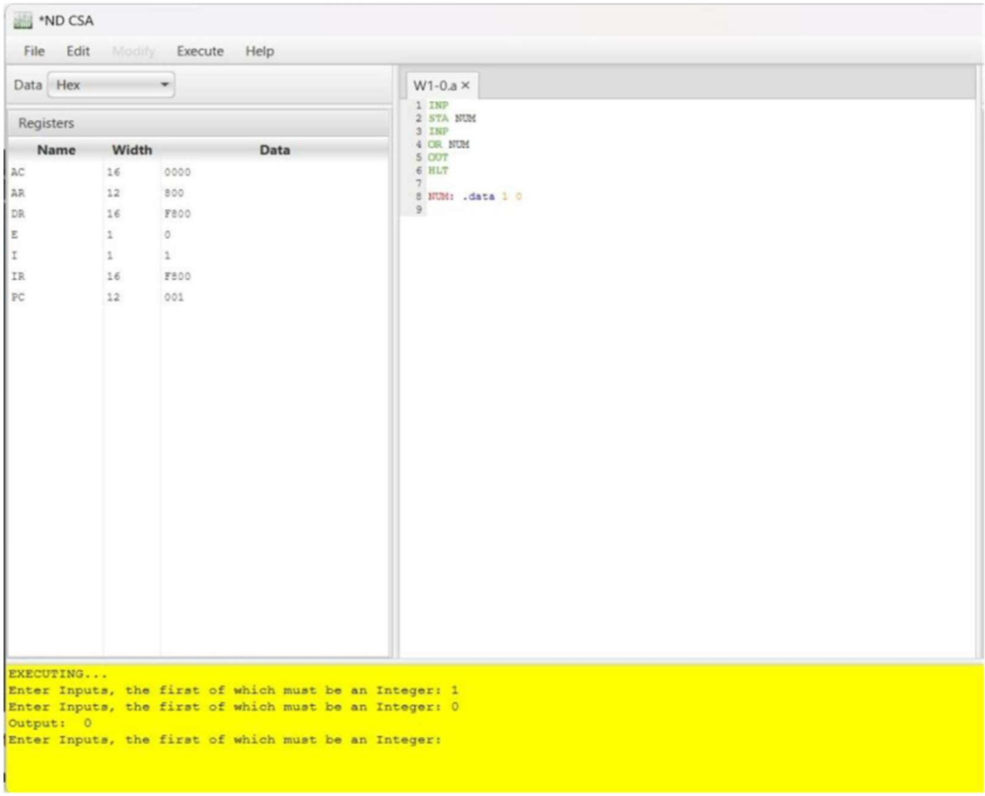


Name	Width	Data
AC	16	0000
AR	12	800
DR	16	F800
E	1	0
I	1	1
IR	16	F800
PC	12	001

```
INP
STA NUM
INP
AND NUM
OUT
HLT

NUM: .data 1 0

```

Addr	Data
000	F800
001	0006
002	F800
003	0006
004	F400
005	7001
006	0000
007	0000
008	0000
009	0000
00A	0000
00B	0000
00C	0000
00D	0000
00E	0000
00F	0000
010	0000
011	0000
012	0000
013	0000
014	0000
015	0000
016	0000

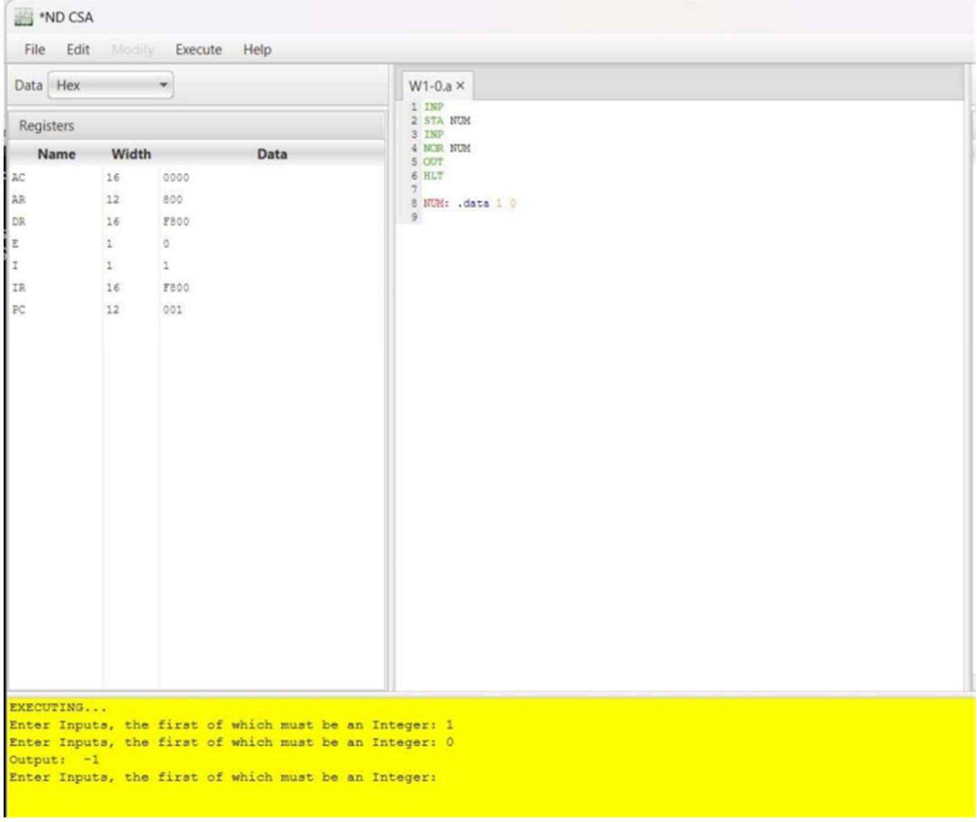
```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 1
Enter Inputs, the first of which must be an Integer: 3
Output:  0
Enter Inputs, the first of which must be an Integer:
```



Name	Width	Data
AC	16	0000
AR	12	800
DR	16	F800
E	1	0
I	1	1
IR	16	F800
PC	12	001

```
INP
STA NUM
INP
OR NUM
OUT
HLT

NUM: .data 1 0

```

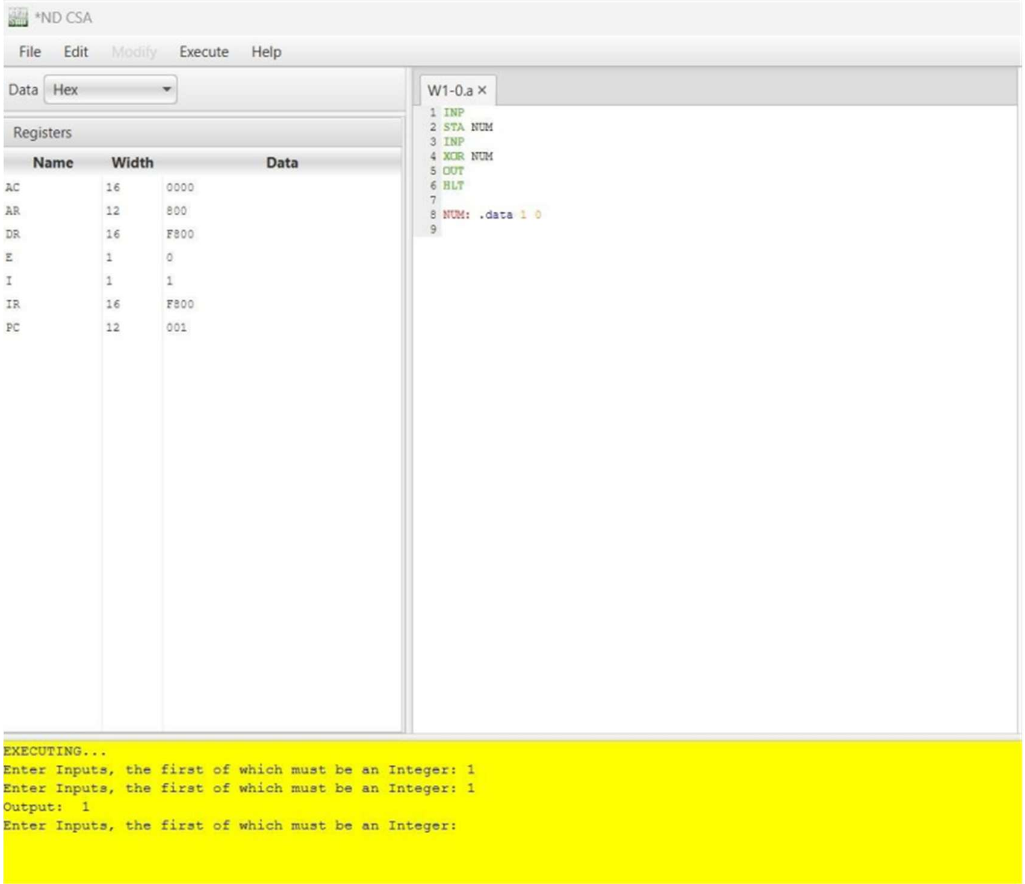
```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 1
Enter Inputs, the first of which must be an Integer: 0
Output:  0
Enter Inputs, the first of which must be an Integer:
```



Name	Width	Data
AC	16	0000
AR	12	800
DR	16	F800
E	1	0
I	1	1
IR	16	F800
PC	12	001

```
INP
STA NUM
INP
NOR NUM
OUT
HLT

NUM: .data 1 0
```

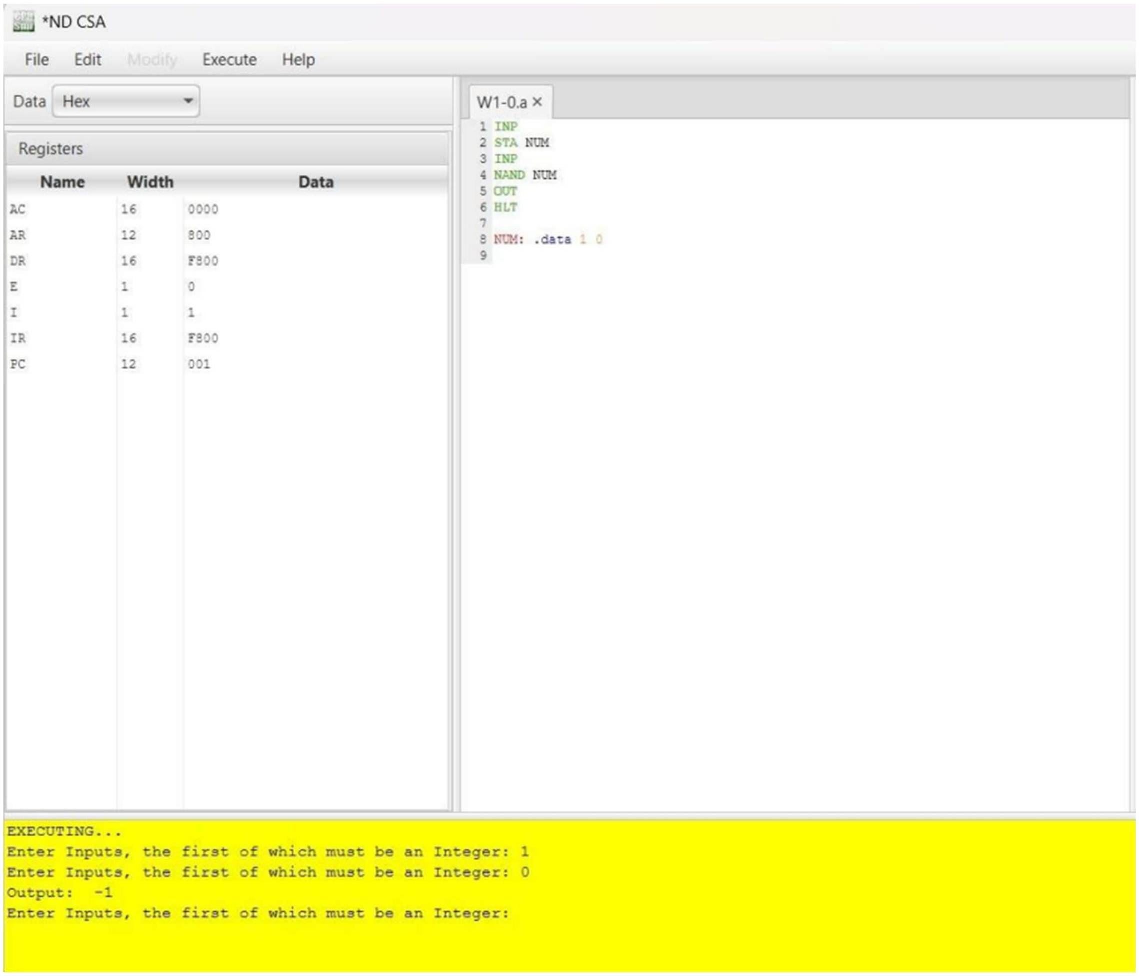
```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 1
Enter Inputs, the first of which must be an Integer: 0
Output:  -1
Enter Inputs, the first of which must be an Integer:
```



Name	Width	Data
AC	16	0000
AR	12	800
DR	16	F800
E	1	0
I	1	1
IR	16	F800
PC	12	001

```
INP
STA NUM
INP
XOR NUM
OUT
HLT

NUM: .data 1 0
```

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

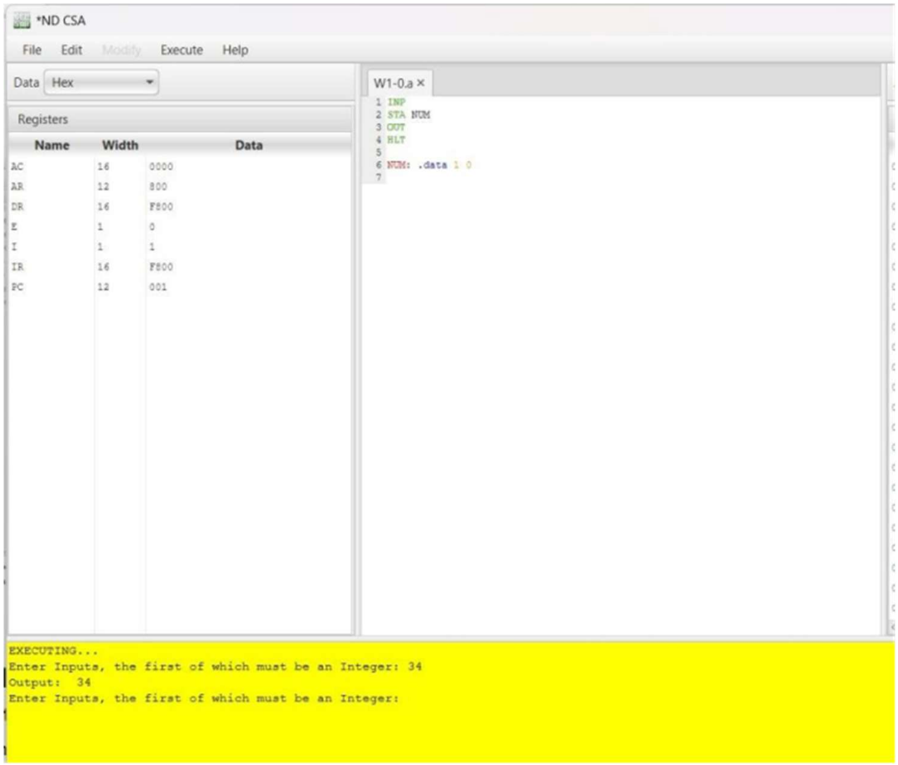


Name	Width	Data
AC	16	0000
AR	12	800
DR	16	F800
E	1	0
I	1	1
IR	16	F800
PC	12	001

```
INP
STA NUM
INP
NAND NUM
OUT
HLT

NUM: .data 1 0

```

```
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:
```

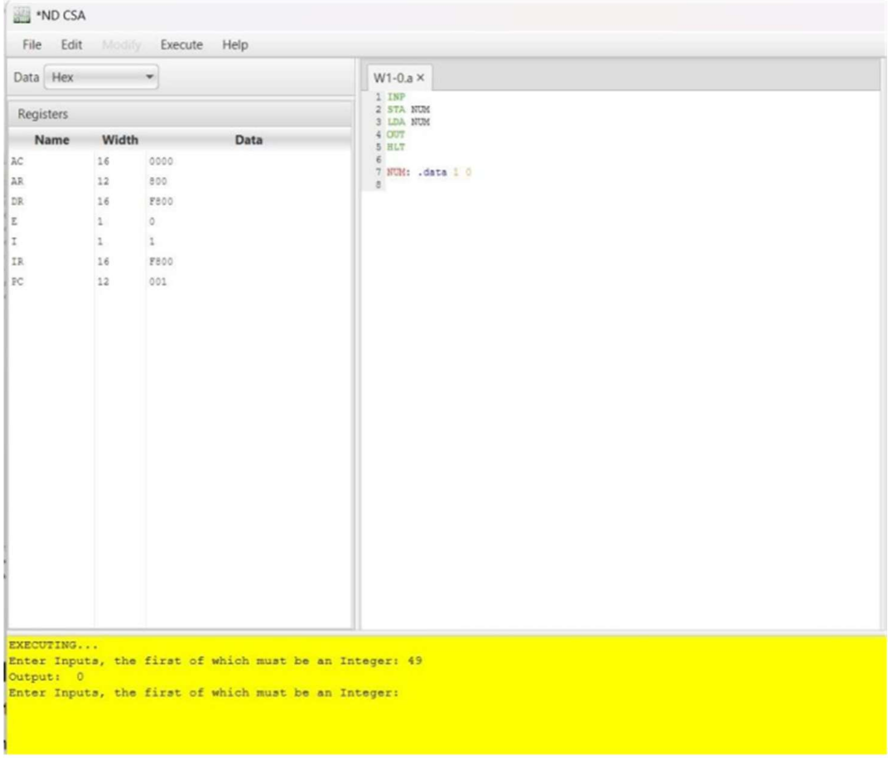
Q6. Write an Assembly program to simulate following memory reference instructions : LDA, STA, BUN, ISZ.



Name	Width	Data
AC	16	0000
AR	12	800
DR	16	F800
E	1	0
I	1	1
IR	16	F800
PC	12	001

```
INP
STA NUM
OUT
HLT

NUM: .data 1 0

```

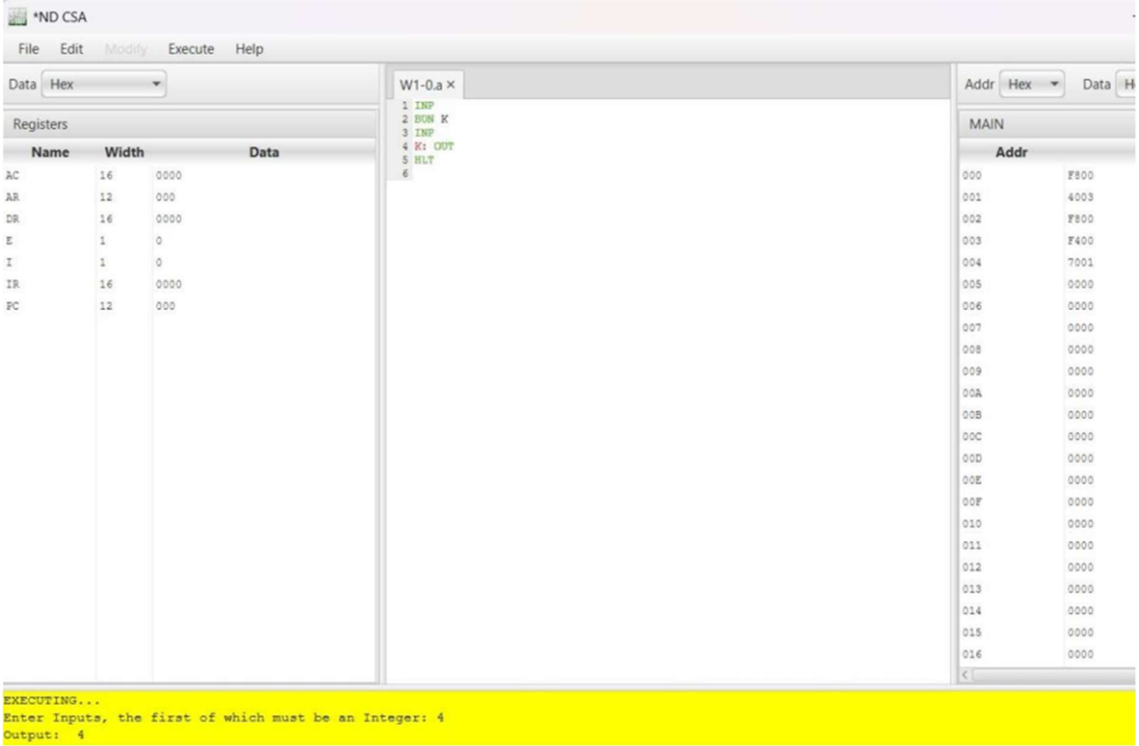
```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 34
Output:  34
Enter Inputs, the first of which must be an Integer:
```



Name	Width	Data
AC	16	0000
AR	12	800
DR	16	F800
E	1	0
I	1	1
IR	16	F800
PC	12	001

```
INP
STA NUM
LDA NUM
OUT
HLT

NUM: .data 1 0

```

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 49
Output:  0
Enter Inputs, the first of which must be an Integer:
```

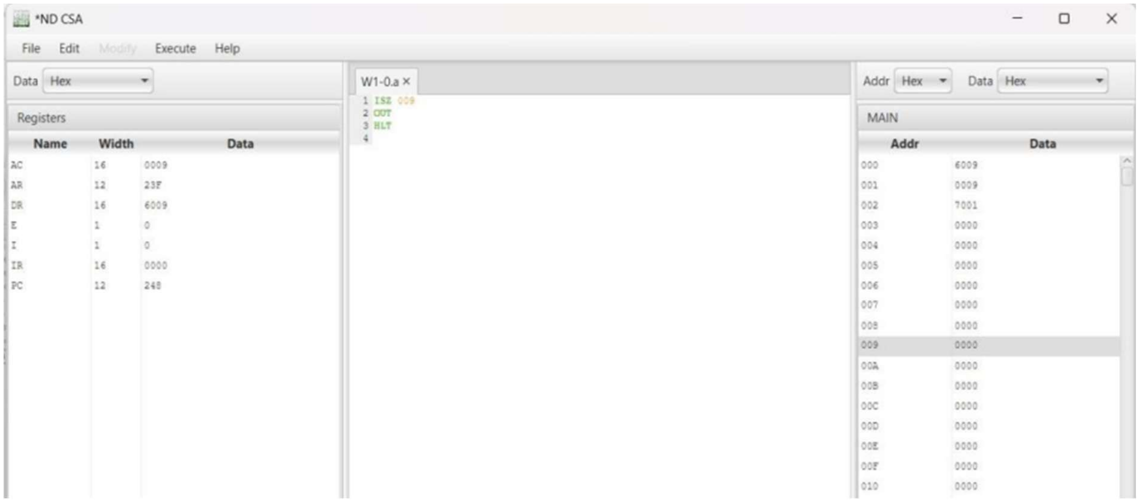


Name	Width	Data
AC	16	0000
AR	12	000
DR	16	0000
E	1	0
I	1	0
IR	16	0000
PC	12	000

```
INP
BUN K
INP
K: OUT
HLT
```

Addr	
000	F800
001	4003
002	F800
003	F400
004	7001
005	0000
006	0000
007	0000
008	0000
009	0000
00A	0000
00B	0000
00C	0000
00D	0000
00E	0000
00F	0000
010	0000
011	0000
012	0000
013	0000
014	0000
015	0000
016	0000

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 4
Output:  4
```

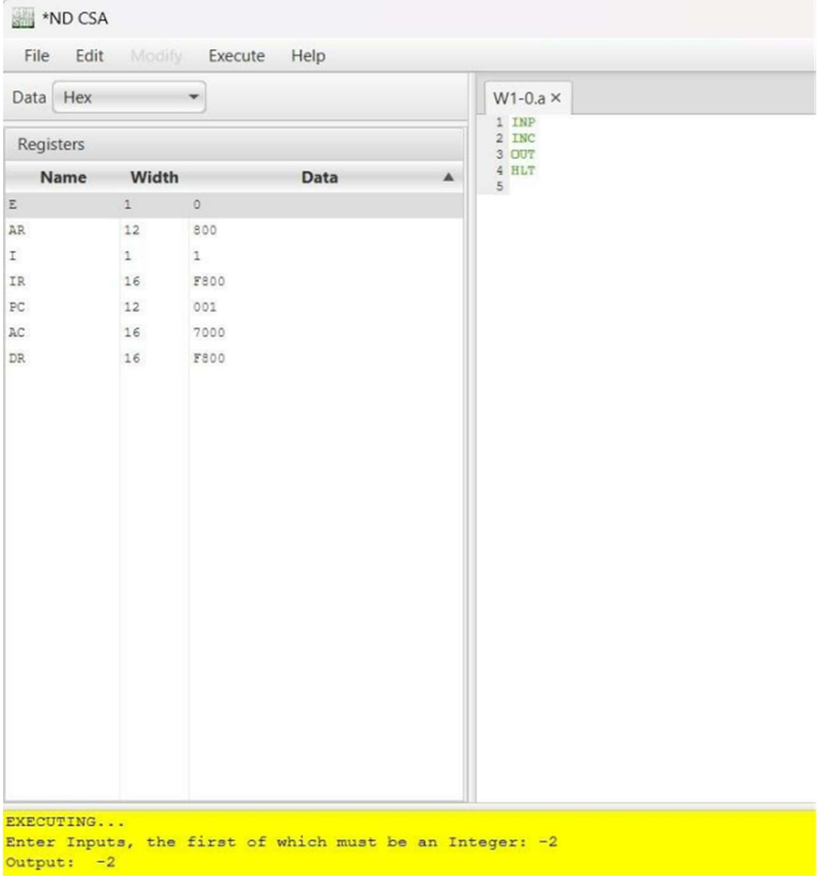


Name	Width	Data
AC	16	0009
AR	12	23F
DR	16	6009
E	1	0
I	1	0
IR	16	0000
PC	12	248

```
ISZ 009
OUT
HLT
```

Addr	Data
000	6009
001	0009
002	7001
003	0000
004	0000
005	0000
006	0000
007	0000
008	0000
009	0000
00A	0000
00B	0000
00C	0000
00D	0000
00E	0000
00F	0000
010	0000

Q7. Write an Assembly Program to simulate following register instructions : INC, SPA, SNA, SZE.

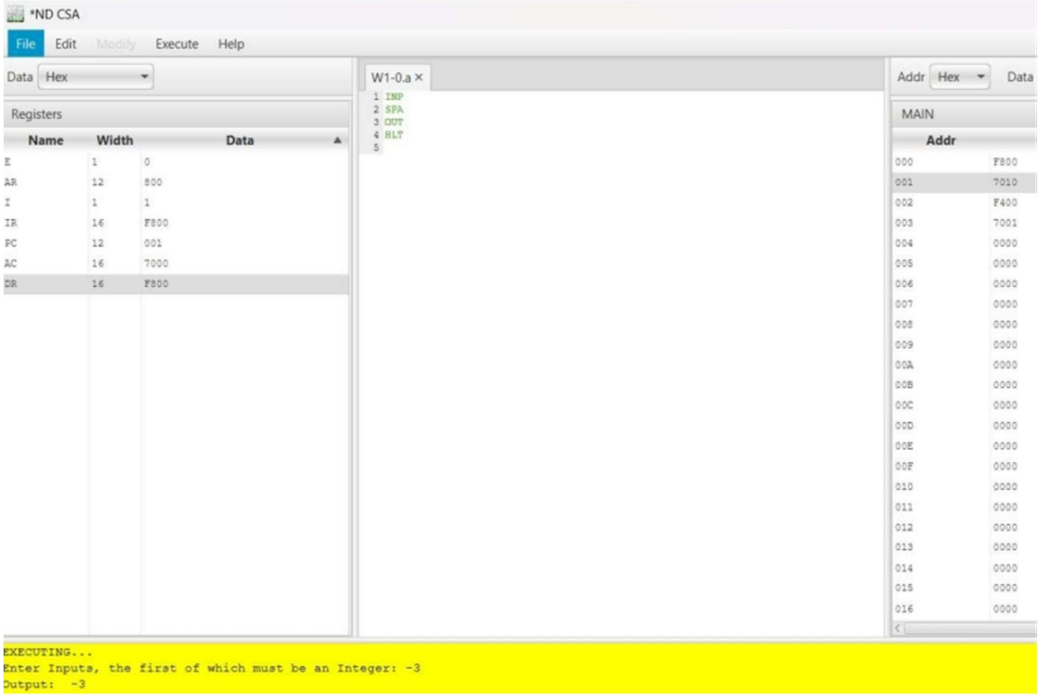


Data: Hex

```
INP
INC
OUT
HLT
```

Name	Width	Data
E	1	0
AR	12	800
I	1	1
IR	16	F800
PC	12	001
AC	16	7000
DR	16	F800

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



Data: Hex

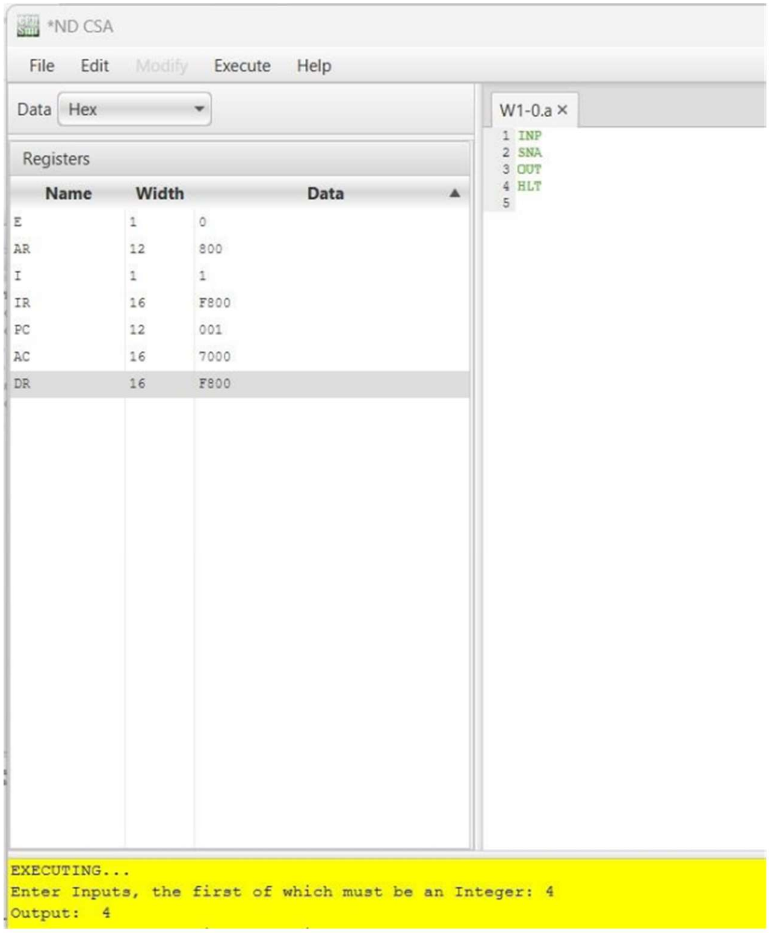
```
INP
SPA
OUT
HLT
```

Name	Width	Data
E	1	0
AR	12	800
I	1	1
IR	16	F800
PC	12	001
AC	16	7000
DR	16	F800

Addr: Hex

Addr	Data
000	F800
001	7010
002	F400
003	7001
004	0000
005	0000
006	0000
007	0000
008	0000
009	0000
00A	0000
00B	0000
00C	0000
00D	0000
00E	0000
00F	0000
010	0000
011	0000
012	0000
013	0000
014	0000
015	0000
016	0000

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: -3
Output:  -3
```



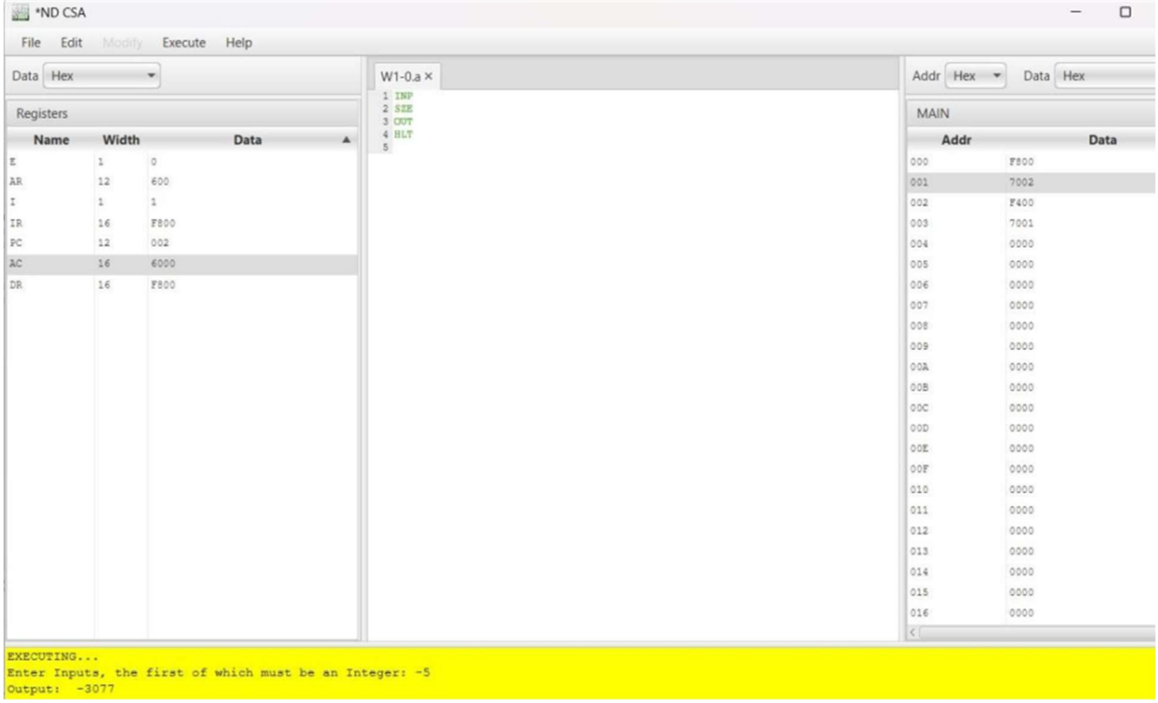
Data: Hex

Registers

Name	Width	Data
E	1	0
AR	12	800
I	1	1
IR	16	F800
PC	12	001
AC	16	7000
DR	16	F800

```
INP
SNA
OUT
HLT

```

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 4
Output:  4
```



Data: Hex

Registers

Name	Width	Data
E	1	0
AR	12	600
I	1	1
IR	16	F800
PC	12	002
AC	16	6000
DR	16	F800

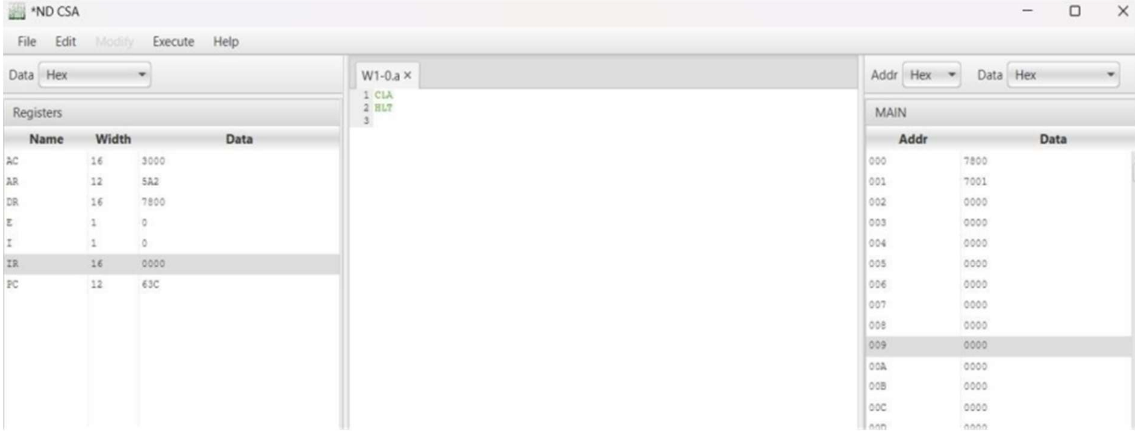
```
INP
SZE
OUT
HLT

```

Addr: Hex Data: Hex

Addr	Data
000	F800
001	7002
002	F400
003	7001
004	0000
005	0000
006	0000
007	0000
008	0000
009	0000
00A	0000
00B	0000
00C	0000
00D	0000
00E	0000
00F	0000
010	0000
011	0000
012	0000
013	0000
014	0000
015	0000
016	0000

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: -5
Output:  -3077
```

Q8. Write an Assembly Program to simulate following register instructions : CLA, CMA, CME, HLT.



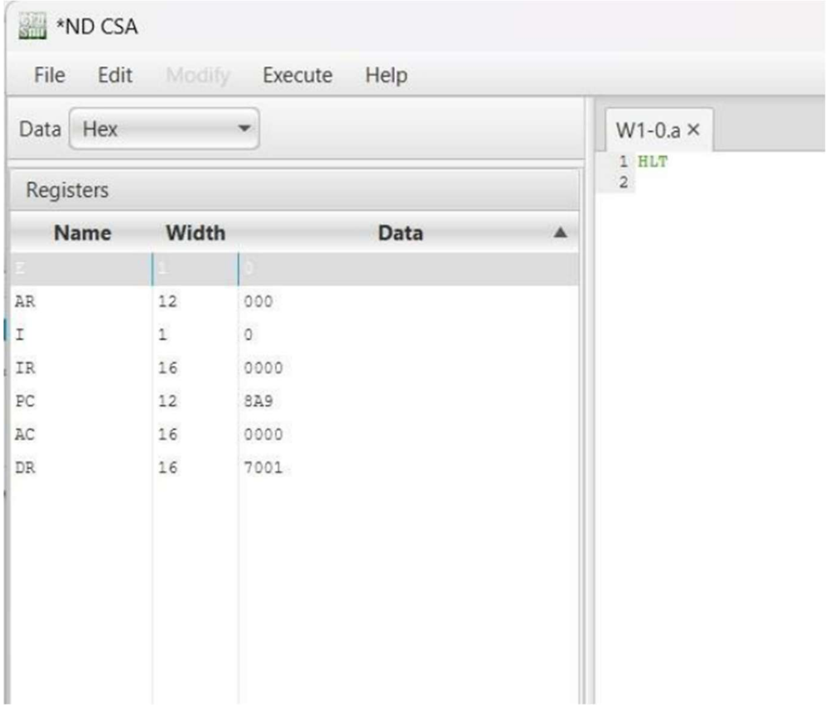
```
CLA
HLT
```

Registers

Name	Width	Data
AC	16	3000
AR	12	5A2
DR	16	7800
E	1	0
I	1	0
IR	16	0000
PC	12	63C

MAIN

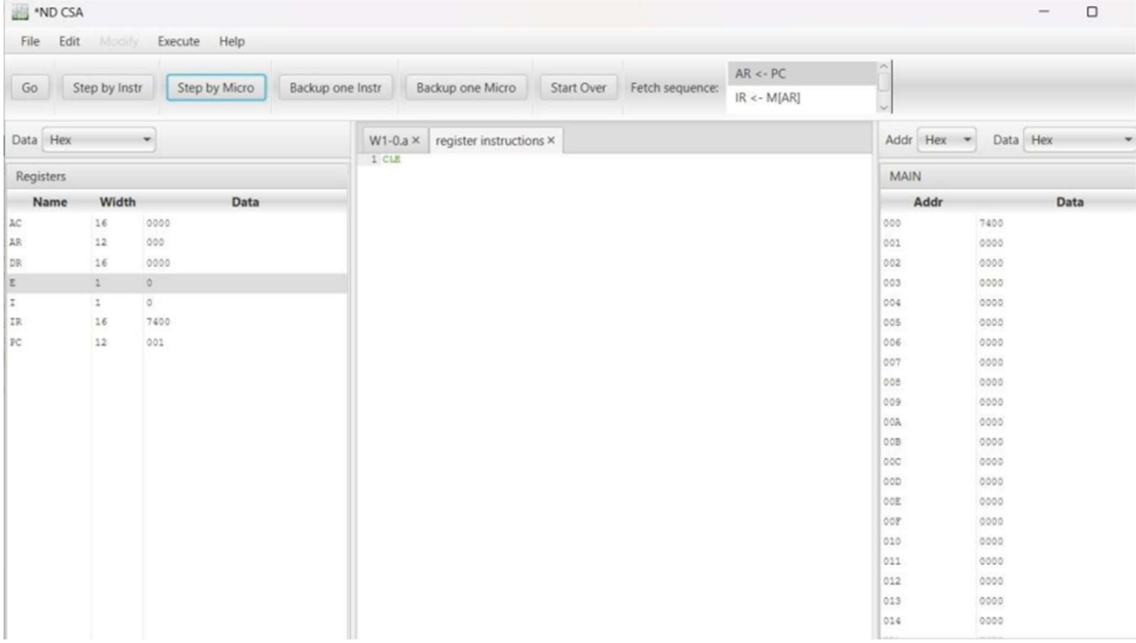
Addr	Data
000	7800
001	7001
002	0000
003	0000
004	0000
005	0000
006	0000
007	0000
008	0000
009	0000
00A	0000
00B	0000
00C	0000
00D	0000



```
HLT
```

Registers

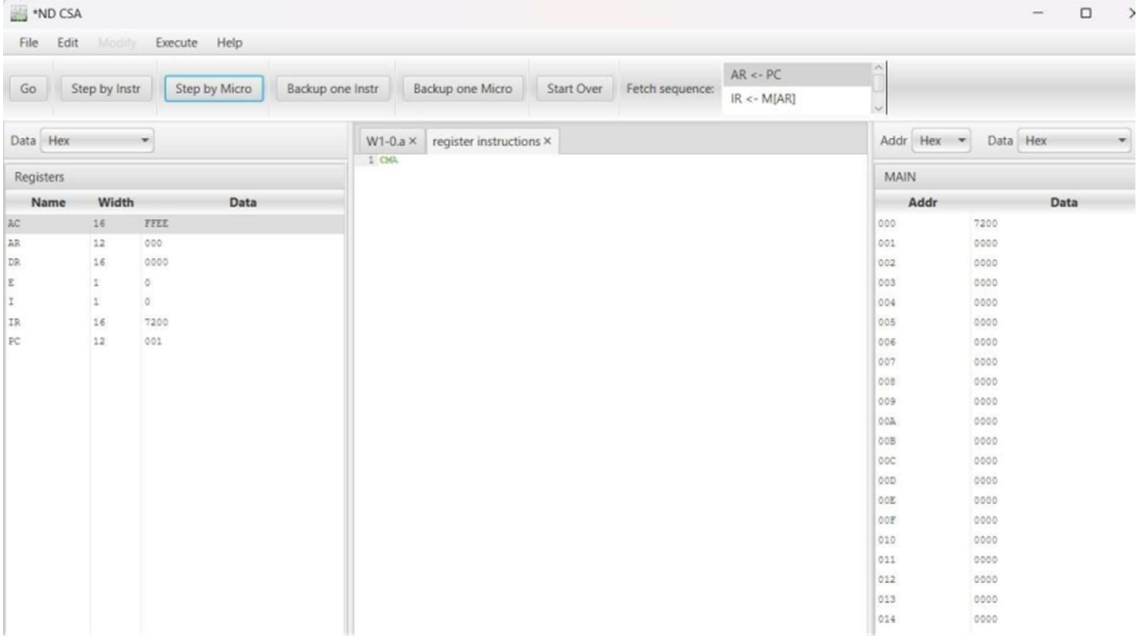
Name	Width	Data
E	1	0
AR	12	000
I	1	0
IR	16	0000
PC	12	8A9
AC	16	0000
DR	16	7001



```
1 CLE
```

Name	Width	Data
AC	16	0000
AR	12	000
DR	16	0000
E	1	0
I	1	0
IR	16	7400
PC	12	001

Addr	Data
000	7400
001	0000
002	0000
003	0000
004	0000
005	0000
006	0000
007	0000
008	0000
009	0000
00A	0000
00B	0000
00C	0000
00D	0000
00E	0000
00F	0000
010	0000
011	0000
012	0000
013	0000
014	0000

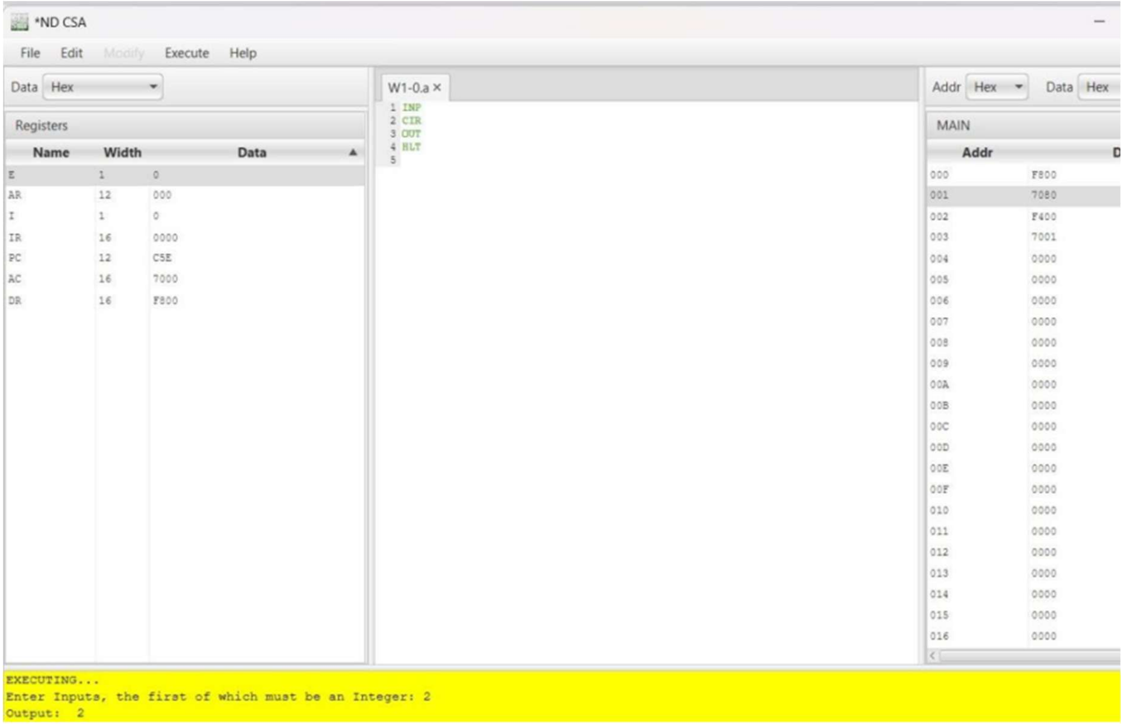


```
1 CMA
```

Name	Width	Data
AC	16	FFEE
AR	12	000
DR	16	0000
E	1	0
I	1	0
IR	16	7200
PC	12	001

Addr	Data
000	7200
001	0000
002	0000
003	0000
004	0000
005	0000
006	0000
007	0000
008	0000
009	0000
00A	0000
00B	0000
00C	0000
00D	0000
00E	0000
00F	0000
010	0000
011	0000
012	0000
013	0000
014	0000

Q. 9. Write an Assembly Program to simulate following register instructions : CIR, CIL.

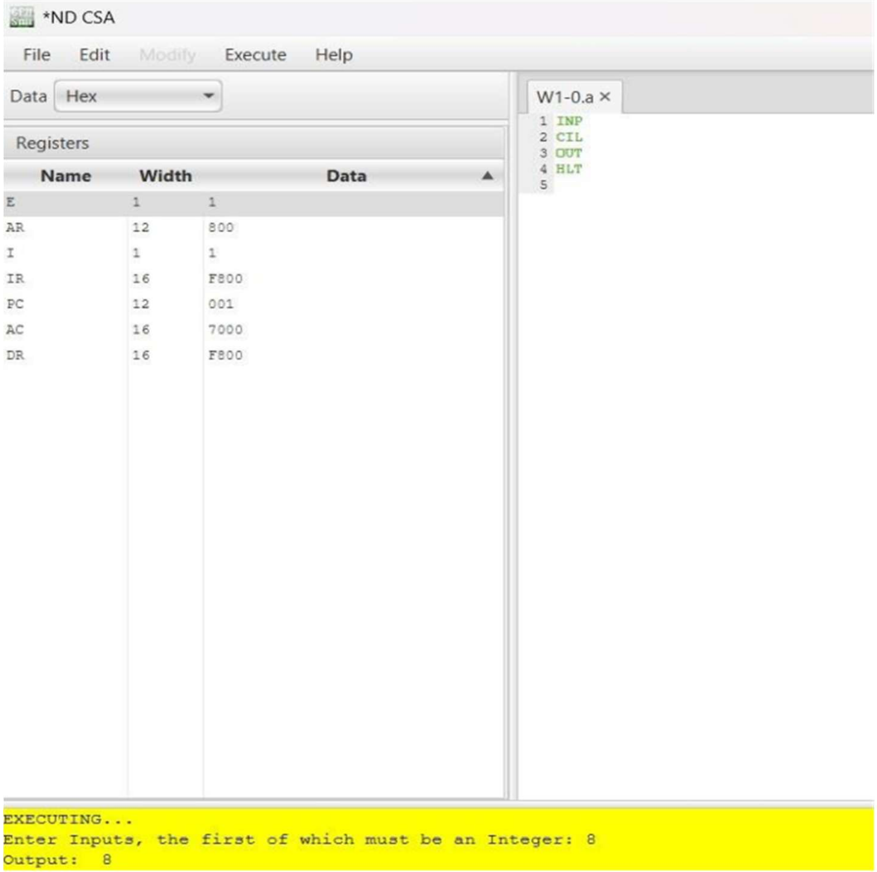


```
INP
CIR
OUT
HLT
```

Name	Width	Data
E	1	0
AR	12	000
I	1	0
IR	16	0000
PC	12	C5E
AC	16	7000
DR	16	F800

Addr	D
000	F800
001	7080
002	F400
003	7001
004	0000
005	0000
006	0000
007	0000
008	0000
009	0000
00A	0000
00B	0000
00C	0000
00D	0000
00E	0000
00F	0000
010	0000
011	0000
012	0000
013	0000
014	0000
015	0000
016	0000

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 2
Output:  2
```

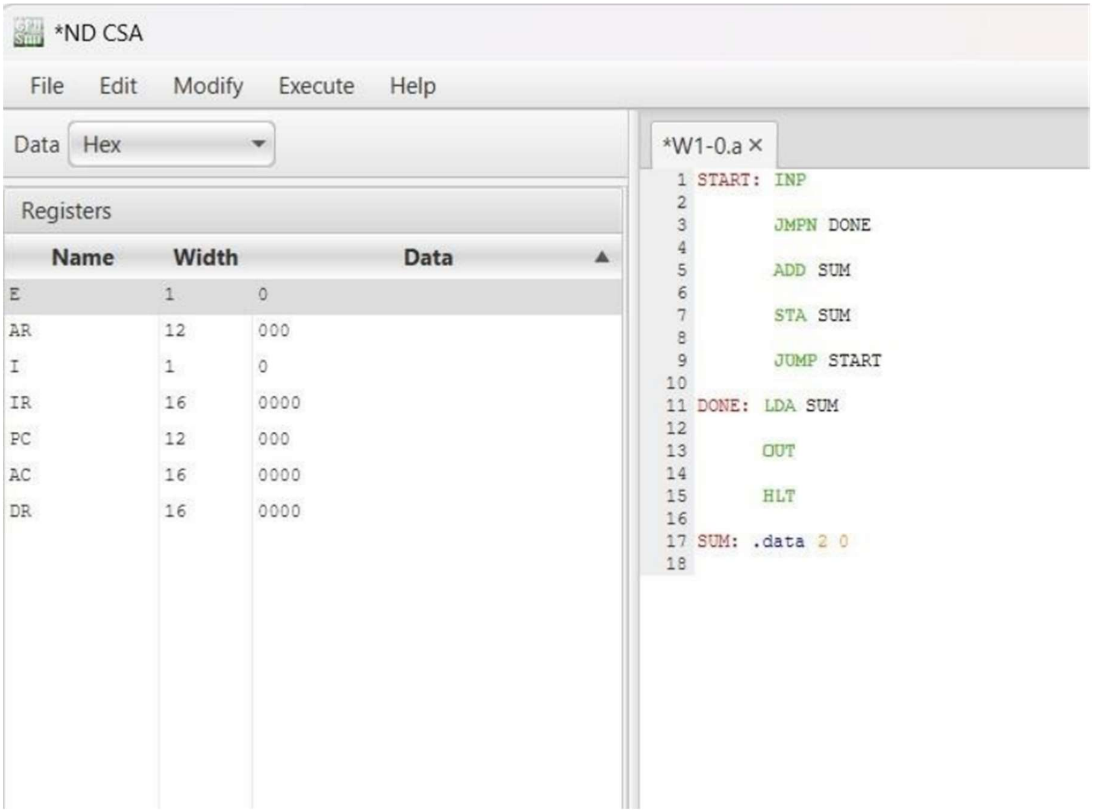


```
INP
CIL
OUT
HLT
```

Name	Width	Data
E	1	1
AR	12	800
I	1	1
IR	16	F800
PC	12	001
AC	16	7000
DR	16	F800

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

Q. 10. Write an Assembly Program that reads in integers and adds them together ; until a negative non zero number is read in.

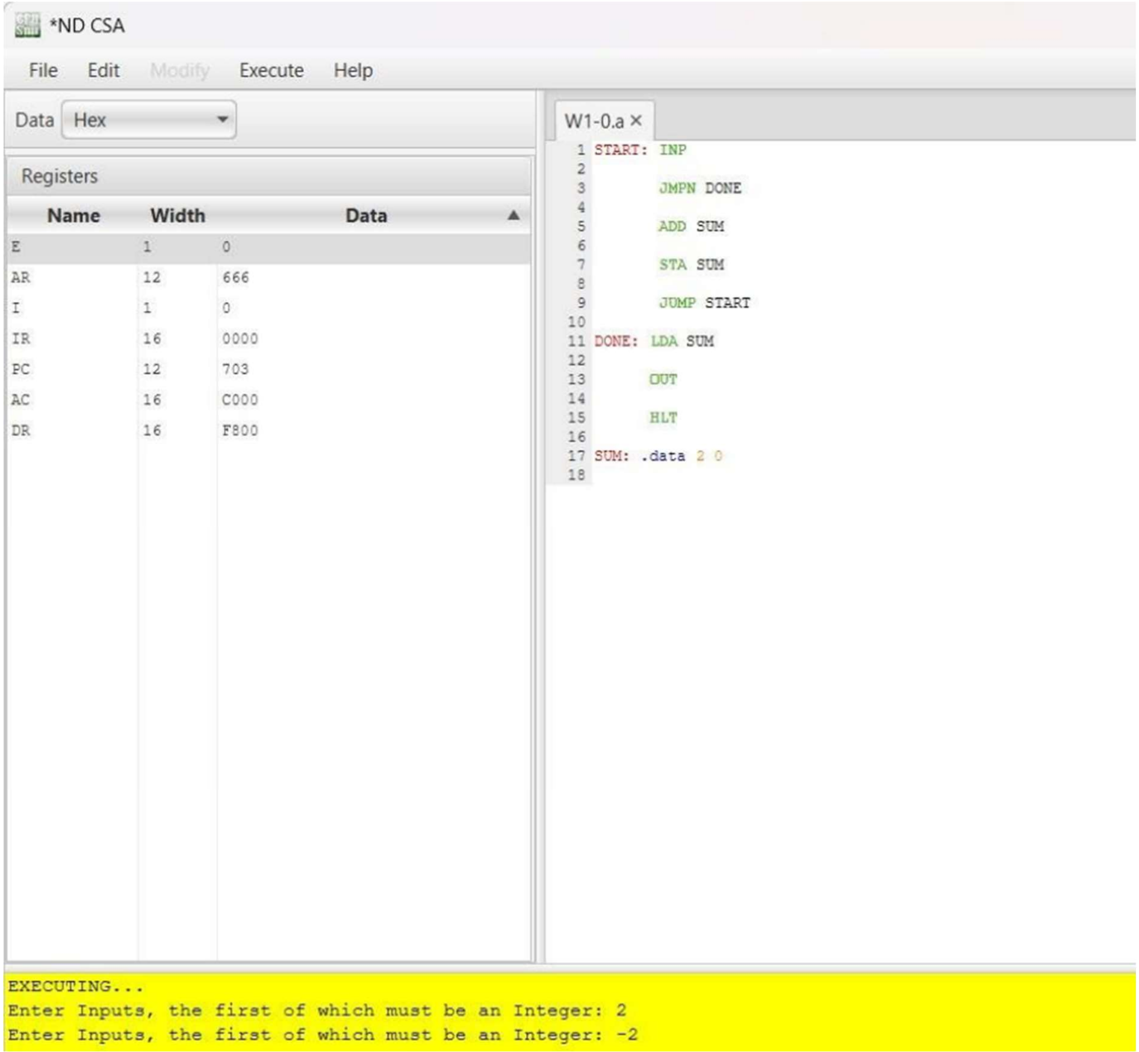


Data Hex

Name	Width	Data
E	1	0
AR	12	000
I	1	0
IR	16	0000
PC	12	000
AC	16	0000
DR	16	0000

```
START: INP

       JMPN DONE

       ADD SUM

       STA SUM

       JUMP START

DONE: LDA SUM

      OUT

      HLT

SUM: .data 2 0

```

Q. 11. Write an Assembly Program that reads in integers and adds them together ; until zero is read in.



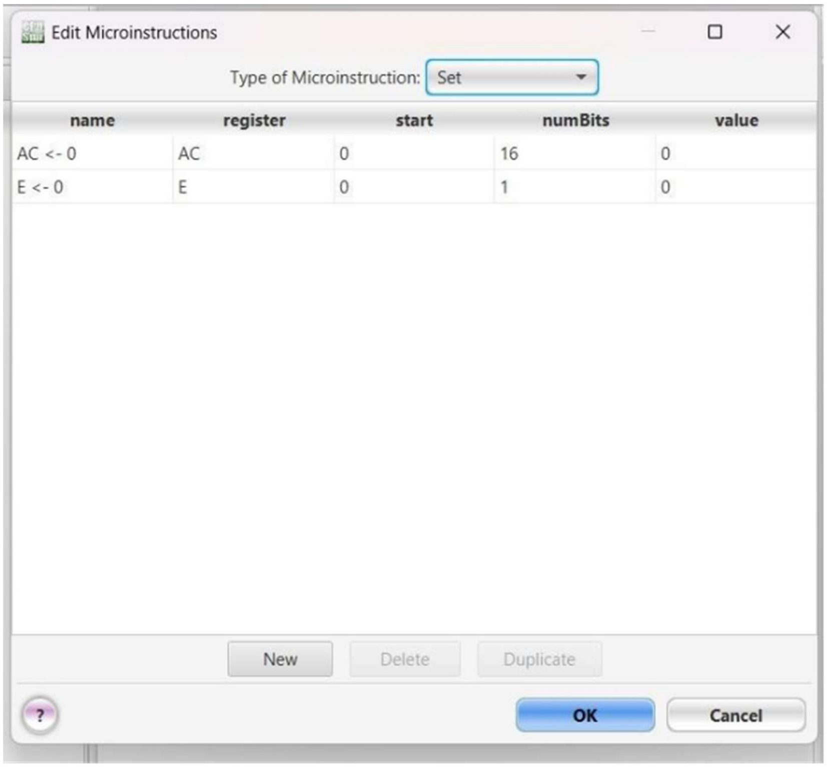
Data: Hex

Name	Width	Data
E	1	0
AR	12	666
I	1	0
IR	16	0000
PC	12	703
AC	16	C000
DR	16	F800

```
START: INP

       JMPN DONE

       ADD SUM

       STA SUM

       JUMP START

DONE: LDA SUM

      OUT

      HLT

SUM: .data 2 0

```

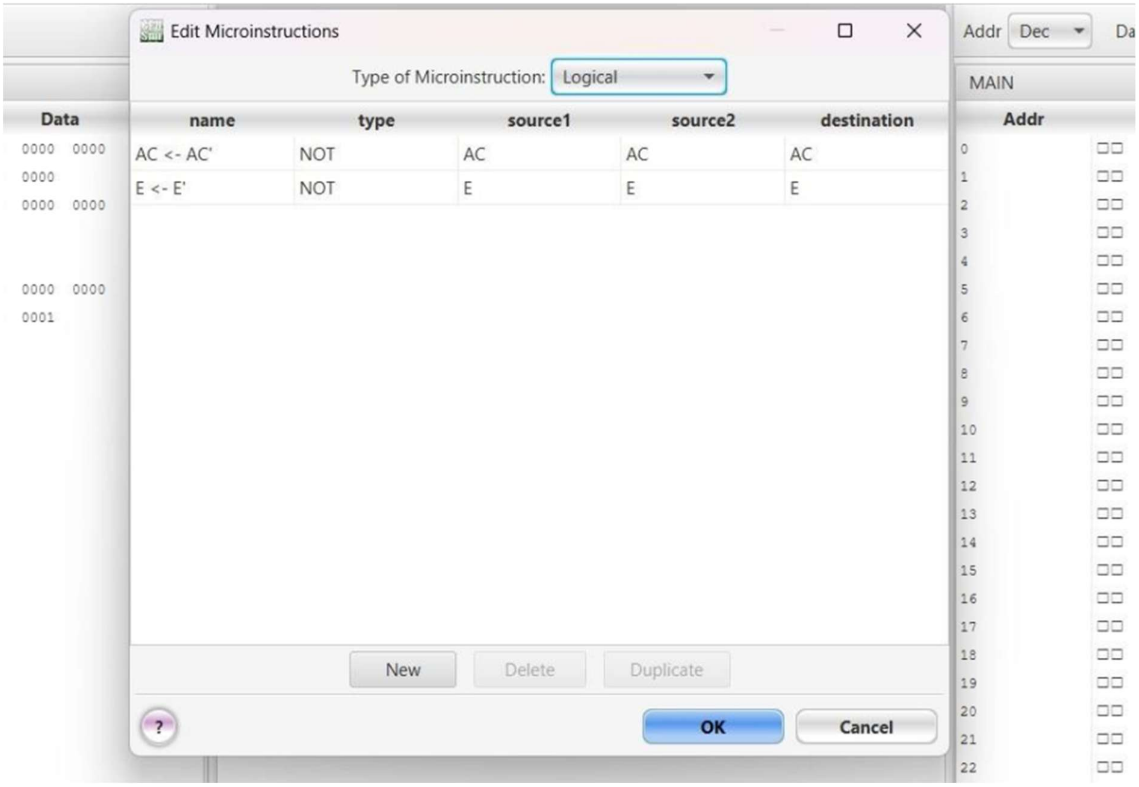
```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 2
Enter Inputs, the first of which must be an Integer: -2
```

IMPORTANT MICRO INSTRUCTIONS :



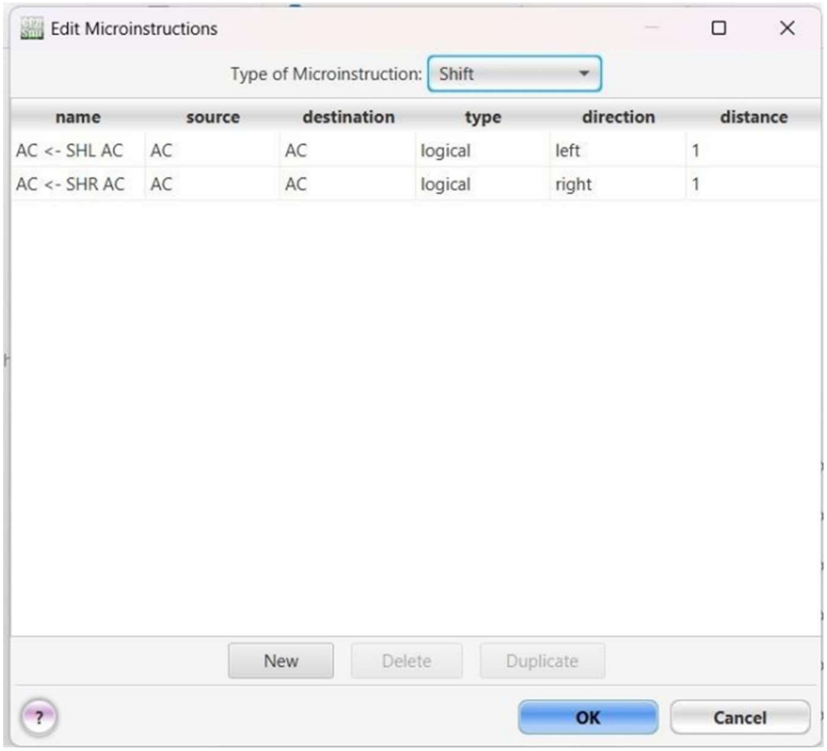
Type of Microinstruction: Set

name	register	start	numBits	value
AC <- 0	AC	0	16	0
E <- 0	E	0	1	0



Type of Microinstruction: Logical

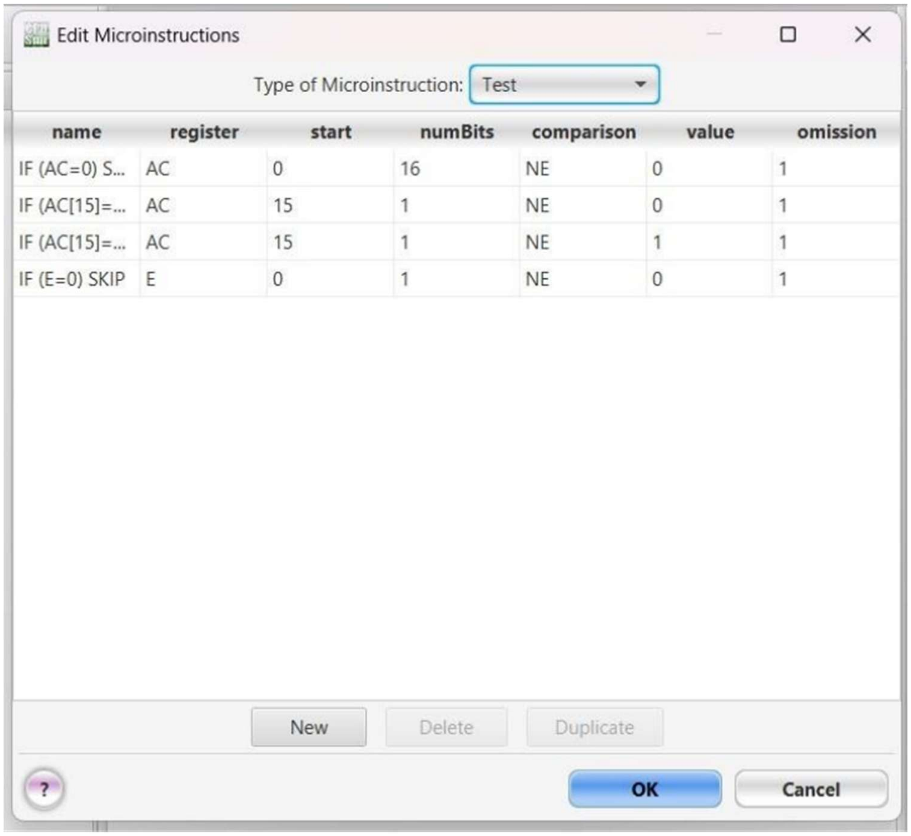
name	type	source1	source2	destination
AC <- AC'	NOT	AC	AC	AC
E <- E'	NOT	E	E	E



Edit Microinstructions

Type of Microinstruction: Shift

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



Edit Microinstructions

Type of Microinstruction: Test

name	register	start	numBits	comparison	value	omission
IF (AC=0) S...	AC	0	16	NE	0	1
IF (AC[15]=...	AC	15	1	NE	0	1
IF (AC[15]=...	AC	15	1	NE	1	1
IF (E=0) SKIP	E	0	1	NE	0	1