Name: Prachi Aggarwal

College Roll no: 21570015

Semester: Ist

Course: Bsc(H) Computer Science

Unique Paper Code: 32341102

Paper Name: Computer System Architecture

Practical Name: CPU Simulator

Submitted to: Ms. Neha Singh

Submitted by: *Prachi*

INDEX

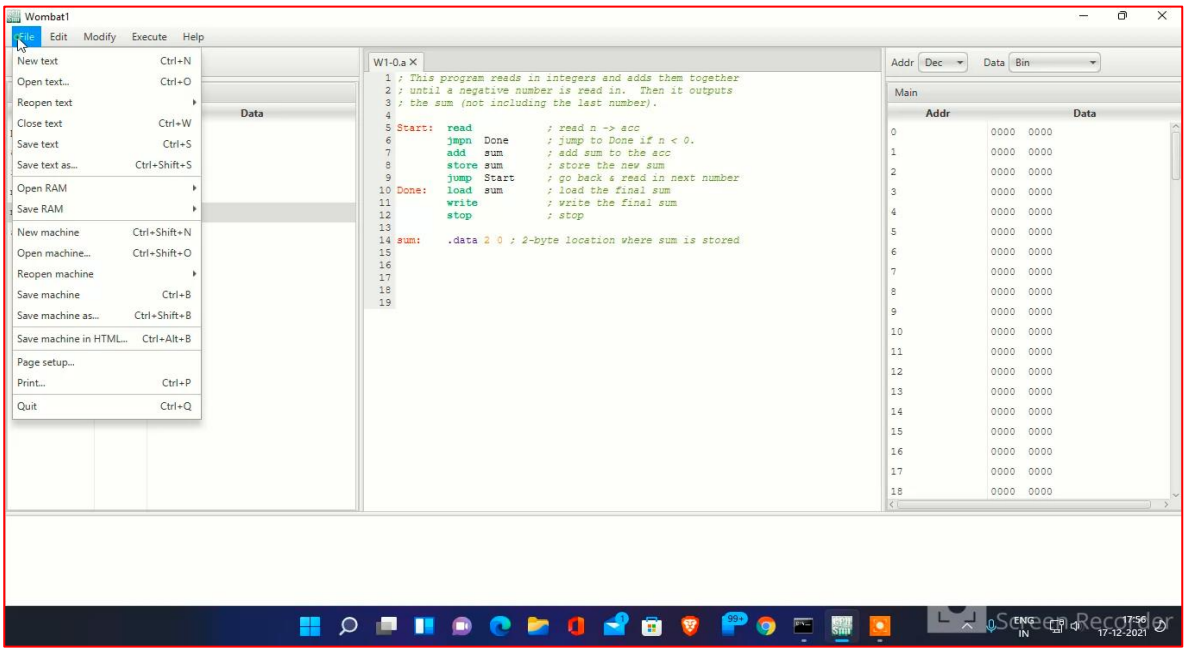
11	Write a program that reads two integers and adds them together until a negative no read.	39
12	Write program that reads and add two integers until 0 is read.	41
13	Create a new machine with indirect bit. And a write a program to add two no with initializing memory from 022.	42
14	Determine the value of all registers after initialize RAM from 026 and initialize a memory word with 082 contain value 298 address with operand 632.	46
15	Write a program to check the I bit to determine the addressing mode and then jump accordingly.	50

PRACTICAL COMPUTER SYSTEM ARCHIETECTURE

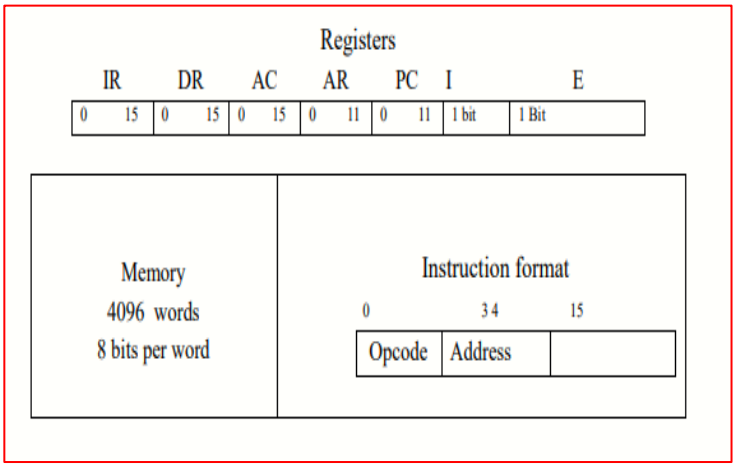
CPU SIMULATOR

1. Main Page of CPU SIM

(Use Simulator – CPU Sim 3.6.9 or any higher version for the implementation)

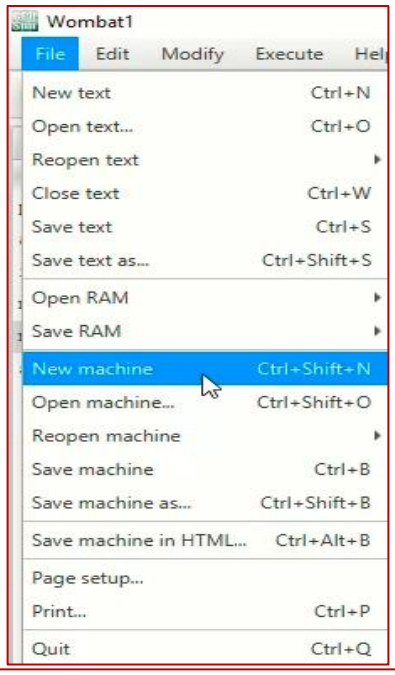


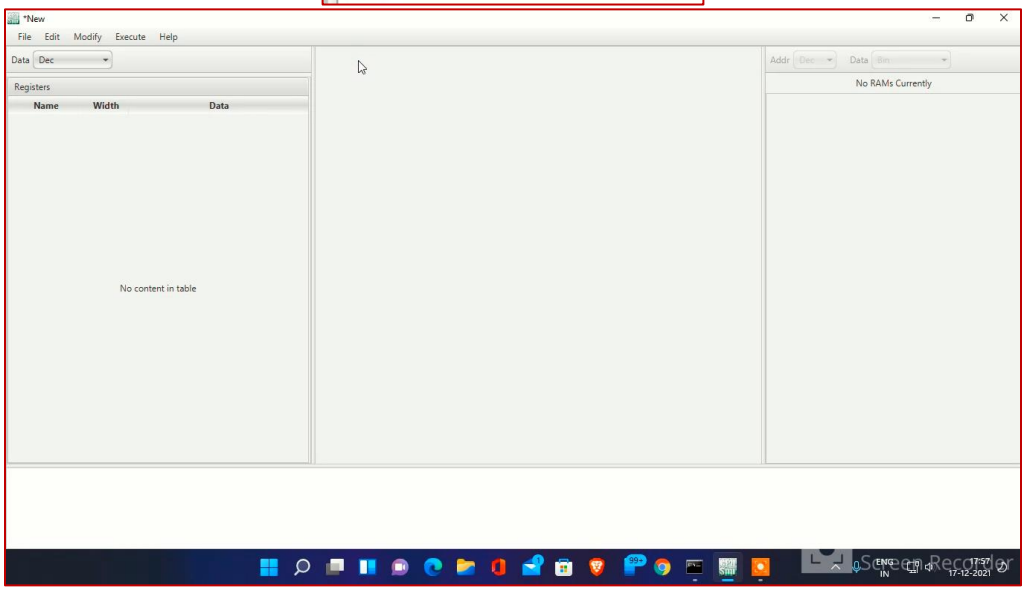
Question 1. **Create a machine based on the following architecture:**



2. Create new machine

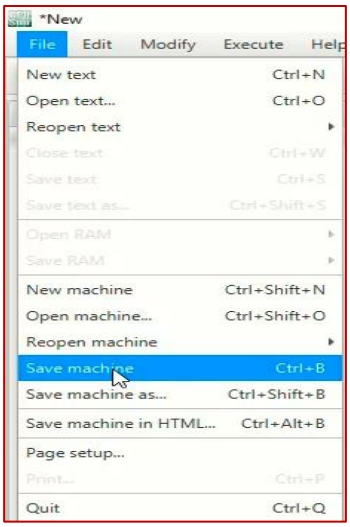
Go on File option----> click on new machine



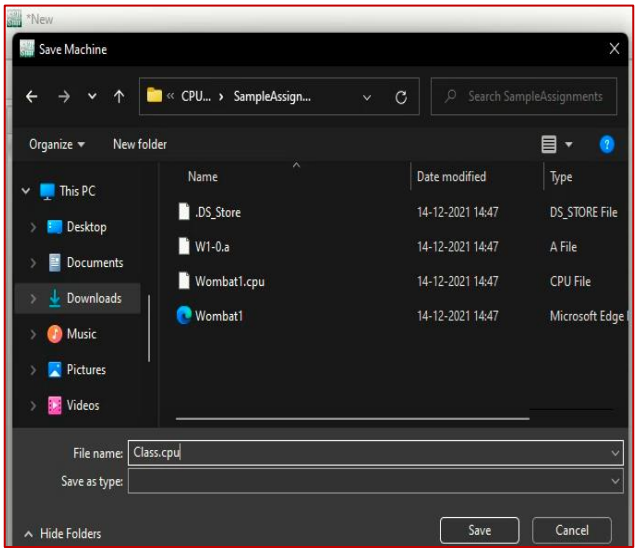


3. Save the machine.

Again, go to File option---> then click save machine



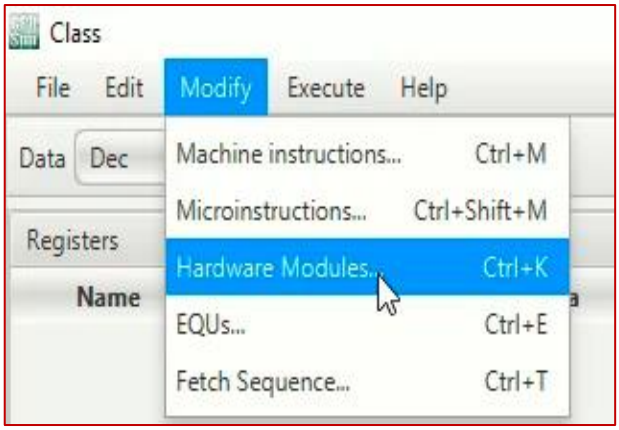
Then choose your Directory where you want to store and store your machine with extension .cpp



Now machine show its name as Class and after every work it show * sign with name that means the work on machine not save so we have to save our work everytime.

4. Using Hardware Modules

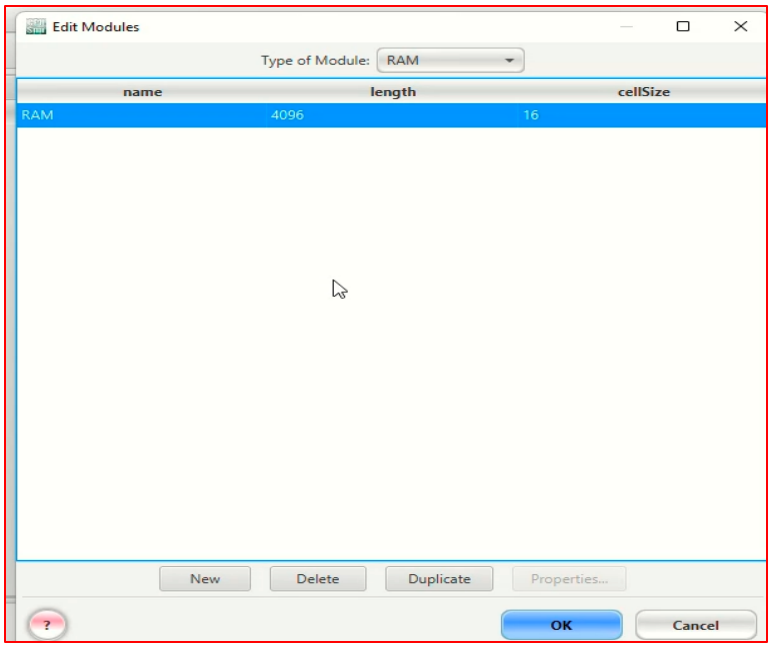
Go to modify option ---> then click Hardware modules.



A) Make a memory for your machine

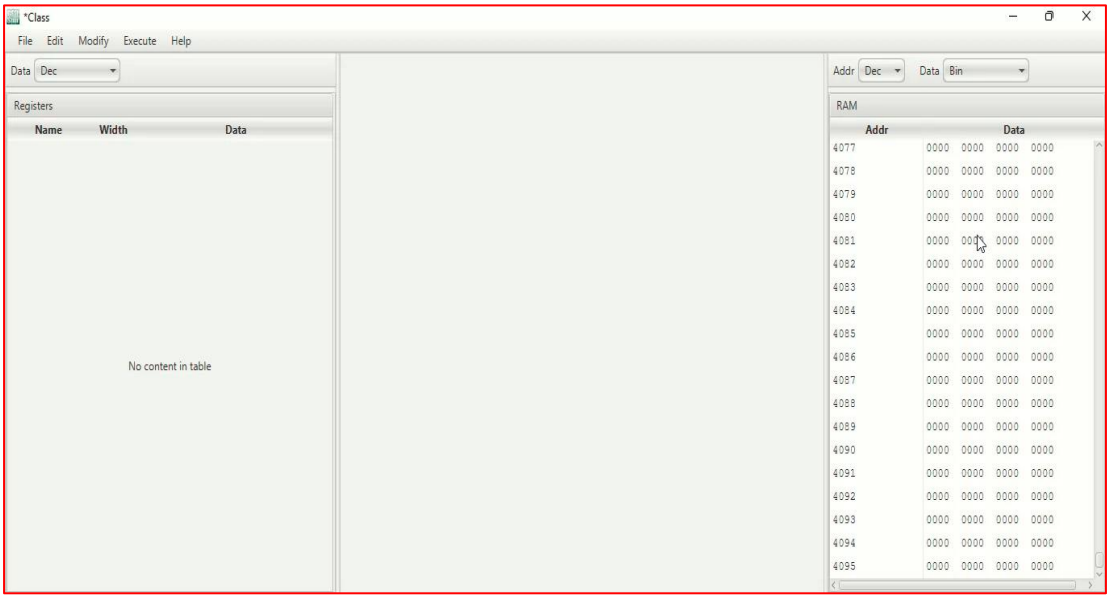
Then select "Type of Module" as RAM...

Click on new to make new memory and Name to your memory like RAM , select length of memory that you want like 4096 and put cell size as 16.



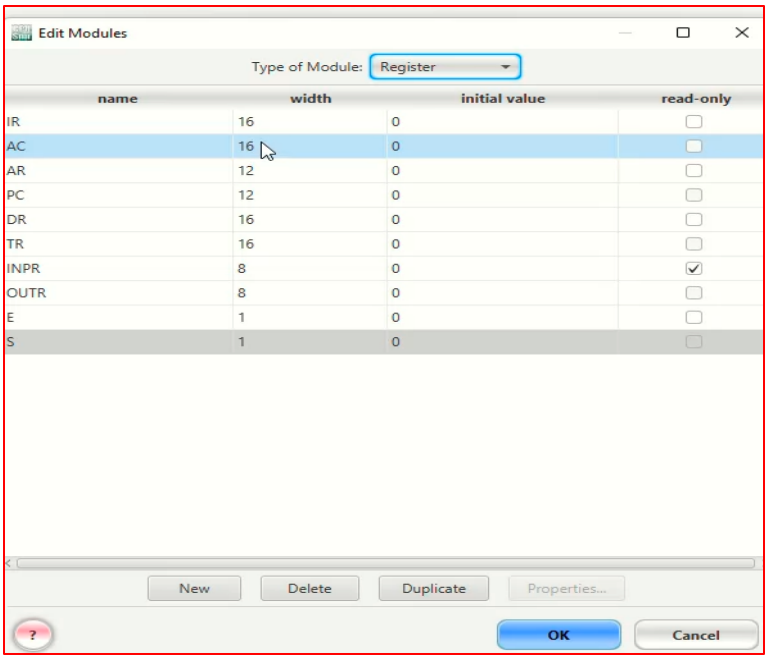
Select "OK" to save changes.

Now your memory represents on <u>right side</u> of your machine look like given figure. (It is showing memory from 0 to 4095 bits i.e., total 4096 bits).



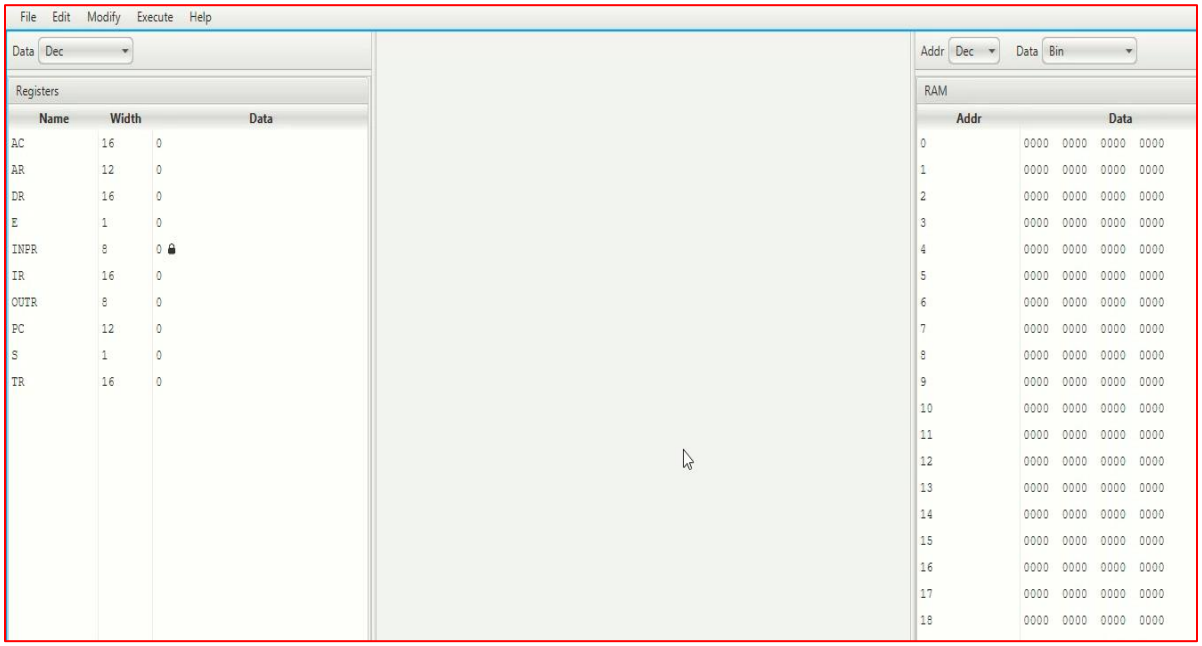
B) Create registers for your machine.

Same process as creating memory, go to Modify ---->Hardware Modules------> Select Type of Modules as "Registers" -----> click on new every time during creating a new register.



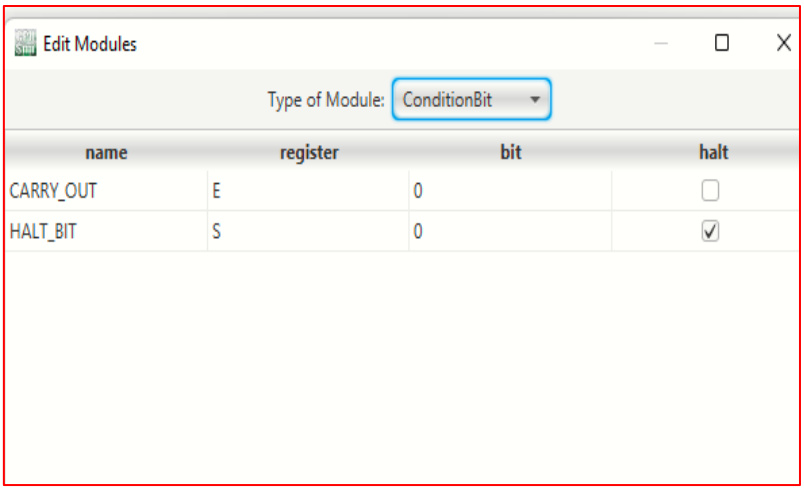
Click OK to save changes...

Now your Registers represents on <u>left side</u> of your machine look like given figure.



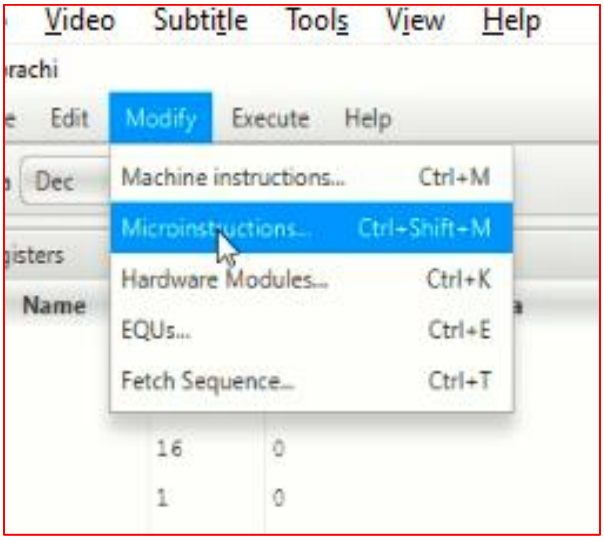
C) Create condition bit.

Modify ---->Hardware Modules------> Select Type of Modules as "Condition Bit" -----> click on new every time during creating a new condition bit.



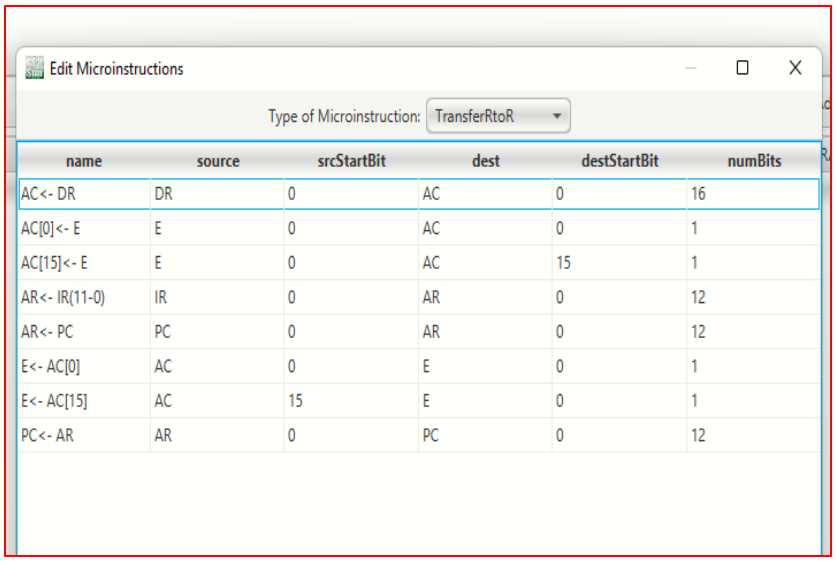
5. Using microinstructions

Go to modify option ---> then click Microinstructions.



A) Writing instructions for transferring instruction from register to register

Go to Type of microinstructions----> choose TransferRtoR



Here source means from which register the information is transferring.....

Dest means to which register we transferring....

SrcStartBit means from which bit it is starting....

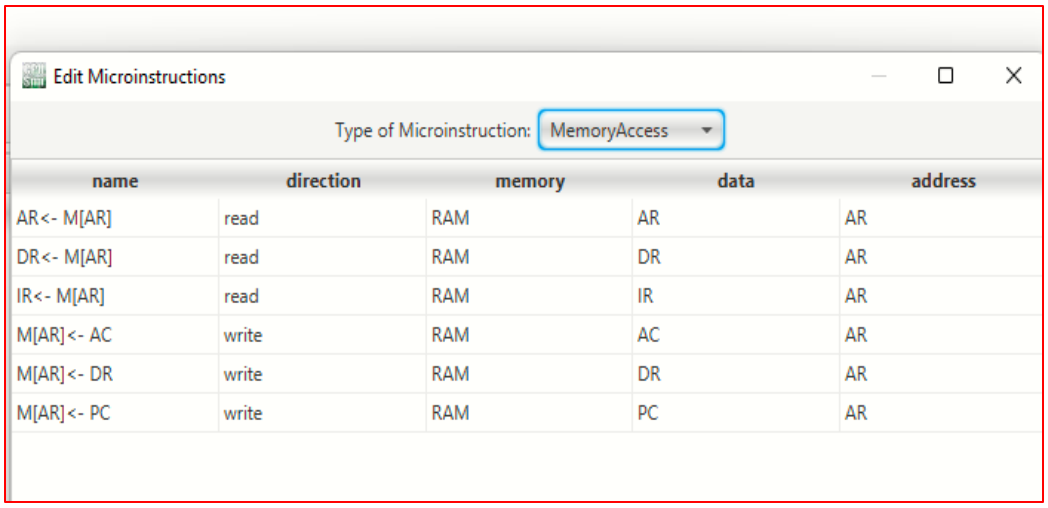
And destStartBit is till which bit is ending....

NumBit means that the number of bits occupying by source register.....

B) Writing instructions to access our memory of machine

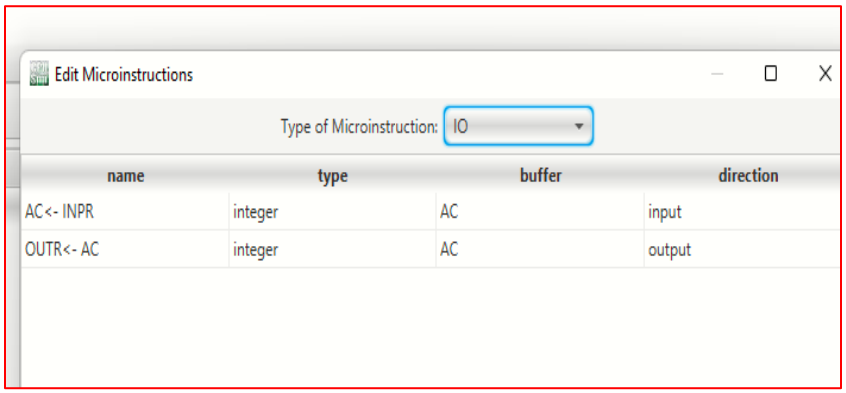
Go to Type of microinstructions----> choose MemoryAccess

The table given below are the instructions help to access the memory name RAM...



C) Writing microinstructions for input output registers.

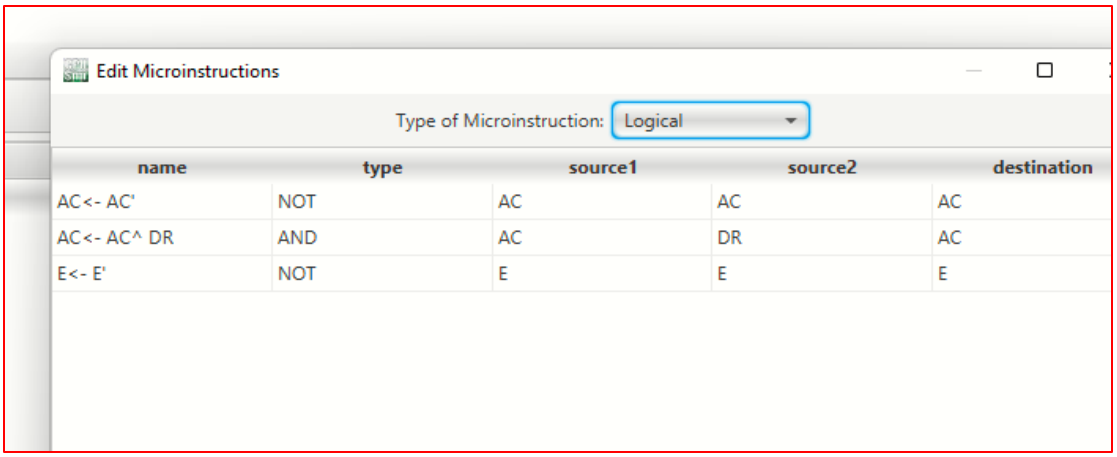
Go to Type of microinstructions----> choose IO



Write these above instructions to make our input output registers working.

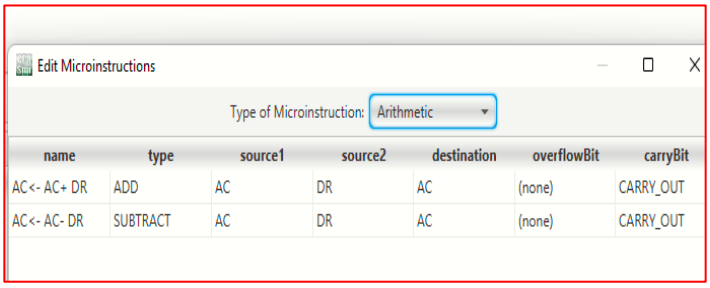
D) Writing microinstructions to apply logical expressions (AND, NOT etc.) with the data in registers and flip-flops.

Go to Type of microinstructions----> choose Logical



E) Writing microinstructions to do arithmetic operations on registers.

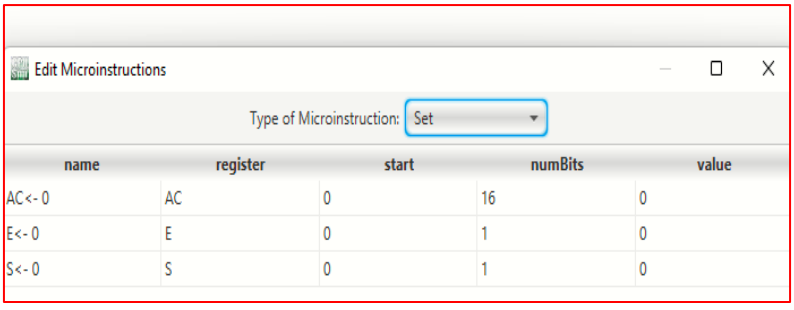
Go to Type of microinstructions----> choose Arithmetic.



First instruction for ADD and second one for SUBTRACT... Every Time when we do these operations then a carry bit remains that's why we use condition Bit as carry out in both cases.

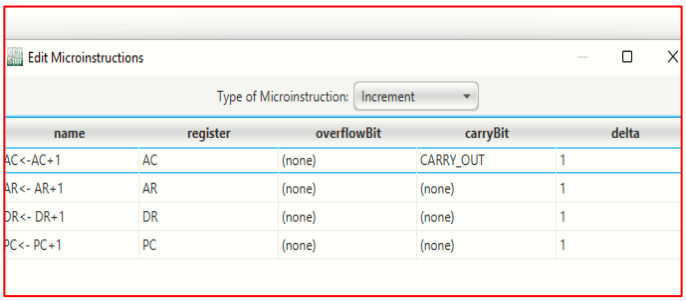
F) Write set microinstructions.

Go to Type of microinstructions----> choose Set.



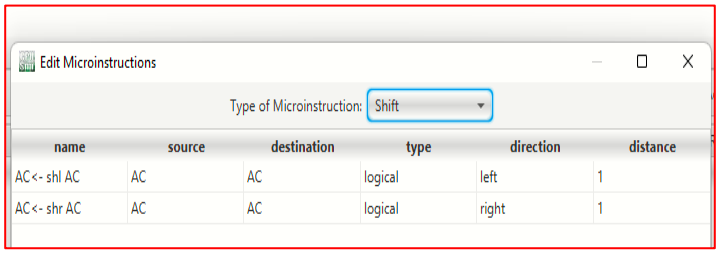
G) Writing microinstruction to increment value in registers by 1.

Go to Type of microinstructions----> choose Increment.



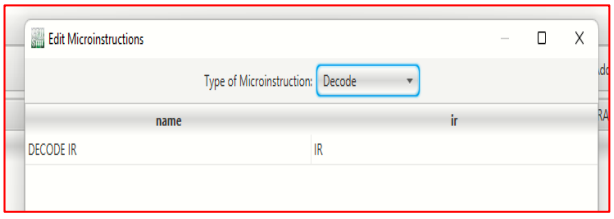
H) Writing microinstruction to shift left or right.

Go to Type of microinstructions----> choose Shift.



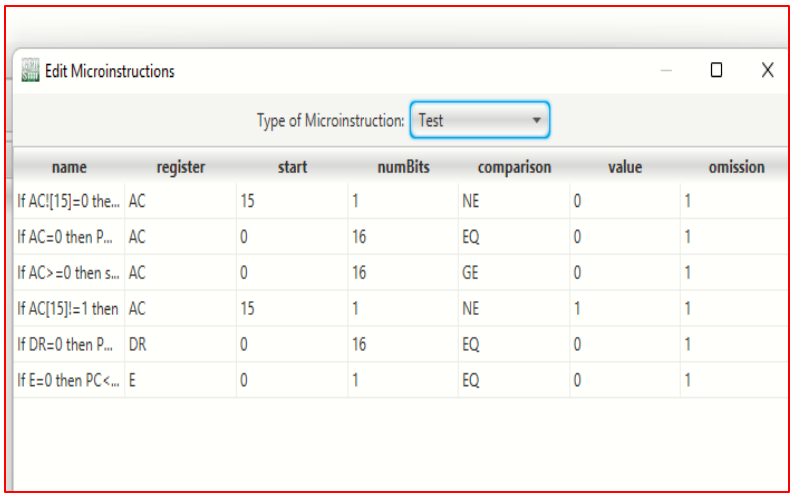
I) Writing microinstruction to decode by instruction register.

Go to Type of microinstructions----> choose Decode.



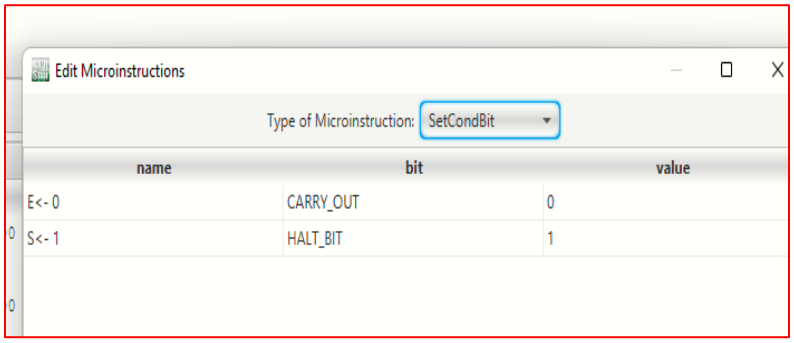
J) Writing microinstructions to test instructions.

Go to Type of microinstructions----> choose Test.

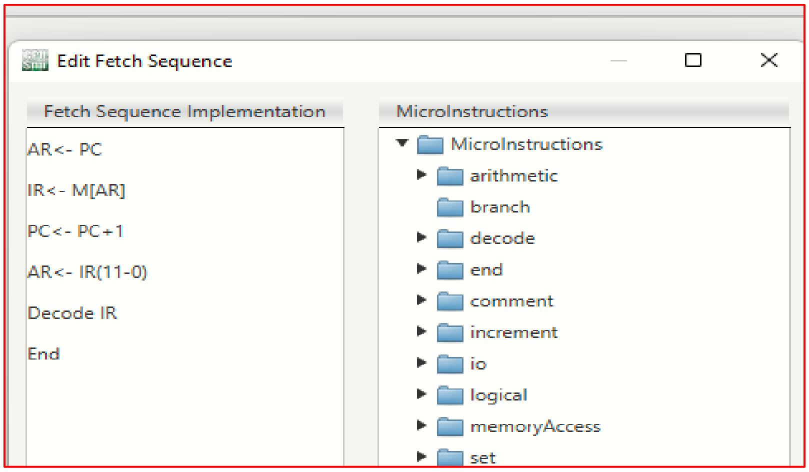


K) Writing microinstructions to setConditionBit.

Go to Type of microinstructions----> choose SetCondBit.



<u>Question 2.</u> **Create a Fetch routine of the instruction cycle.**



So, here is the fetch instructions firstly the program counter PC is loaded with the address of the first instruction in the program. The sequence counter Sc is cleared to 0, providing a decoded timing signal T0. After each clock pulse, SC is incremented by one, so that the timing signals go through a sequence T0, T1, T2, T3 and so on. The microoperations for the fetch and decode phases can be specified by the following register transfer statements.

T0: AR<--- PC

T1: IR<--- M[AR], PC<--- PC+1

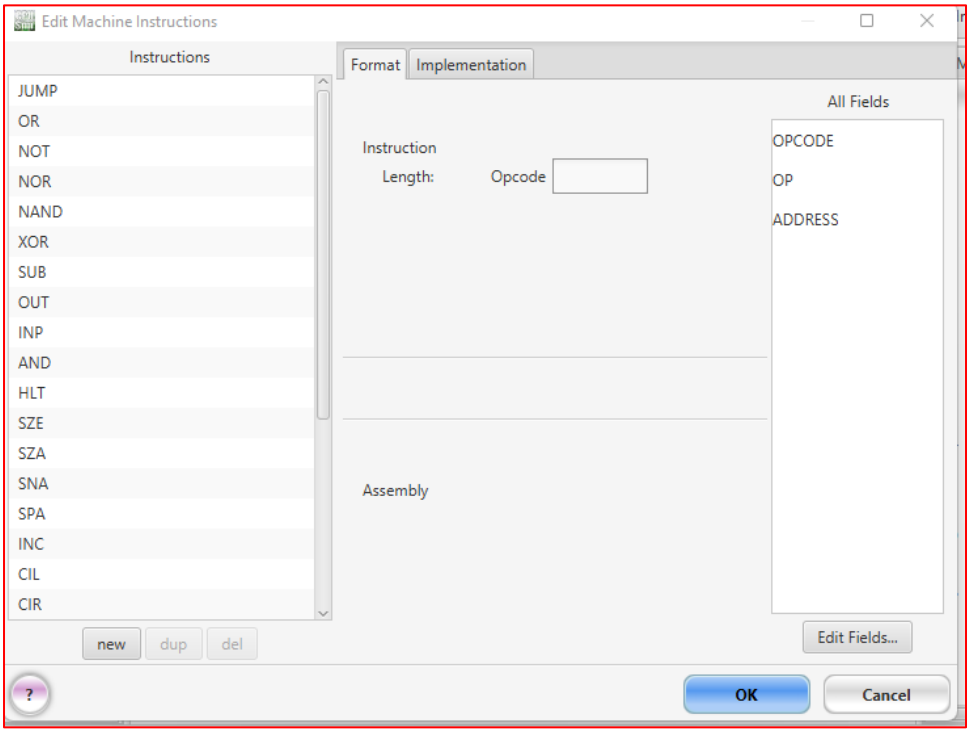
T2: D0,….,D7<--- Decode IR (12-14), AR<--- IR(0-11), I<--- IR(15)

****

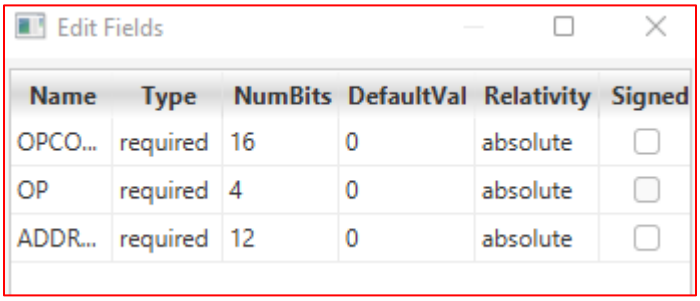
Now, it's time to create a structure of our machine instructions.

- **Modify ----> machine instructions**

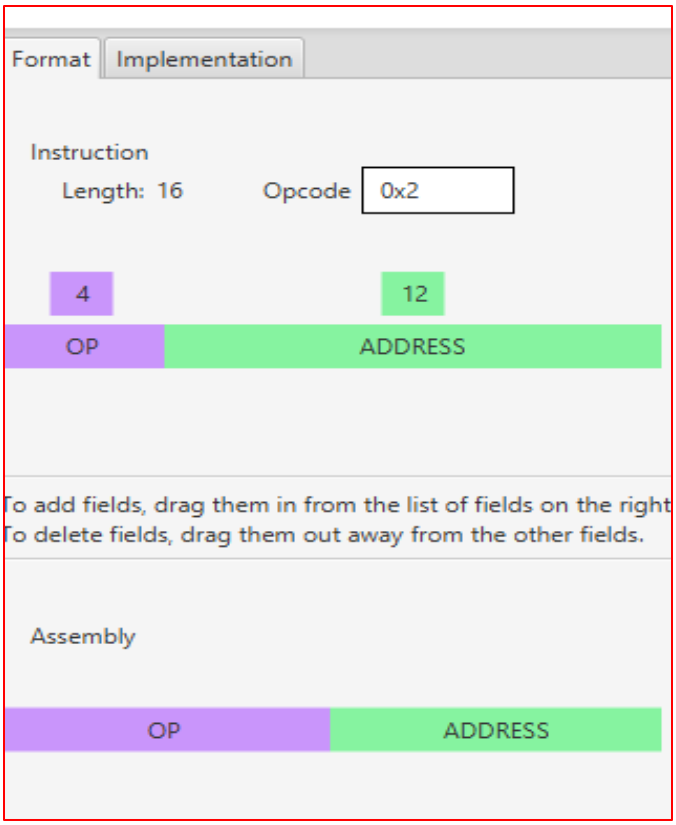
On left side of the screen of machine instructions, here is column of INSTRUCTIONS we click on new and make all instructions needed



While on right side of the screen, here is column of All Fields in which we make 4-bit construct of opcode 12-bit construct of address and 16-bit opcode construct for reference register instructions.

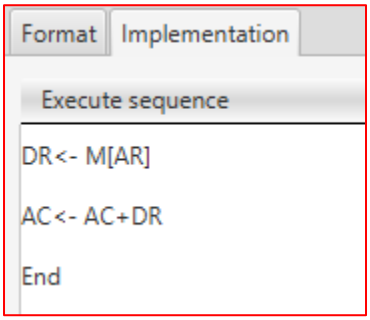


FOR EG: ADD is a memory reference instruction and we using format of 4-bit opcode and 12-bits address of opcode 2xxx.



For all memory reference instructions, the format is same.

Now, put all implementation for e.g.: ADD instruction after formatting it.



All the opcodes for all instructions are in the below table:

Basic Computer Instructions

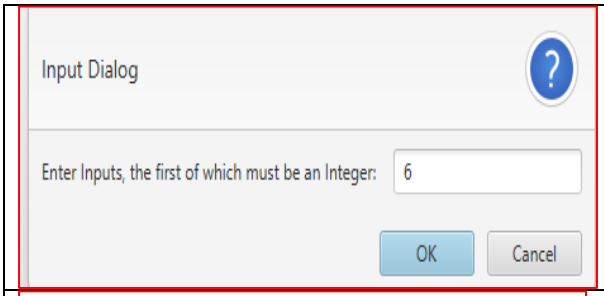
Memory Reference			Register Reference		
Symbol	Hex		Symbol	Hex	
AND	0xxx	Direct Addressing	CLA	E800	
ADD	2xxx		CLE	E400	
LDA	4xxx		CMA	E200	
STA	6xxx		CME	E100	
BUN	8xxx		CIR	E080	
			CIL	E040	
ISZ	Cxxx		INC	E020	
AND_I	1xxx	Indirect Addressing	SPA	E010	
ADD_I	3xxx		SNA	E008	
LDA_I	5xxx		SZA	E004	
STA_I	7xxx		SZE	E002	
BUN_I	9xxx		HLT	E001	
ISZ_I	Dxxx				

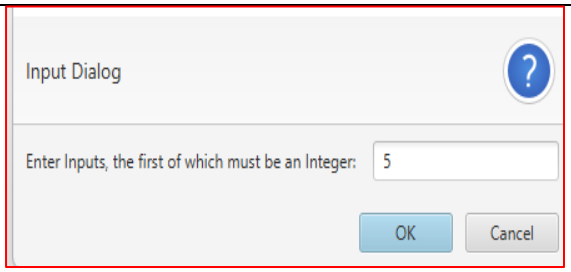
Question 3. Write an assembly program to simulate ADD operation on two user-entered numbers.

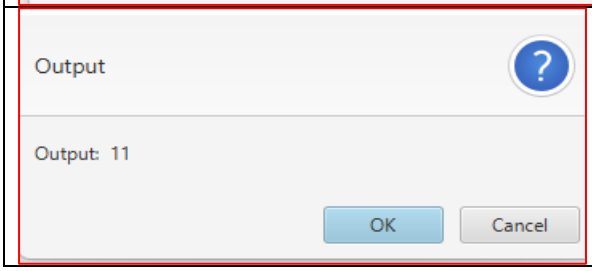
q3.a

```
1  INP
2  STA A
3  INP
4  STA B
5  LDA A
6  ADD B
7  OUT
8  HLT
9  A: .data 1 [0]
10 B: .data 1 [0]
11
```

RESULT:







Input 1= 5
Input 2= 6
Output= 11

Question 4. Write an assembly program to simulate SUBTRACT operation on two user-entered numbers.

Ans)

Q4.a

```
1  INP
2  STA A
3  INP
4  STA B
5  LDA A
6  SUB B
7  OUT
8  HLT
9  A:  .data 1 [0]
10 B:  .data 1 [0]
11
```

CODE

Execute sequence

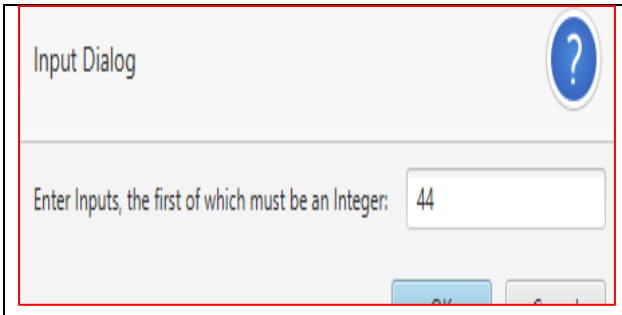
DR<- M[AR]

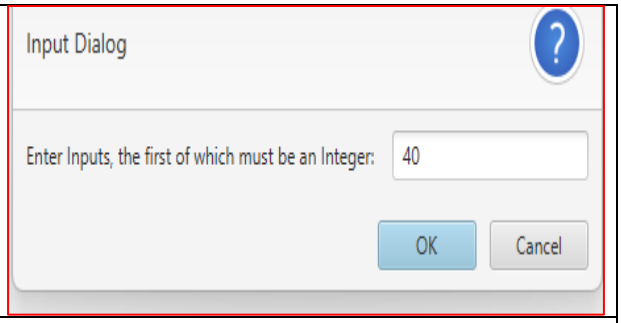
AC<- AC-DR

End

IMPLEMENTATION

RESULT:





INPUT 1: 44
INPUT 2: 40
OUTPUT: 4

Question 5. Write an assembly program to simulate the following logical operations on two user entered numbers.

(a)AND

```
 1 INP
 2 STA A
 3 INP
 4 STA B
 5 LDA A
 6 AND B
 7 OUT
 8 HLT
 9 A:  .data 1[4]
10 B:  .data 1[6]
11
```

CODE

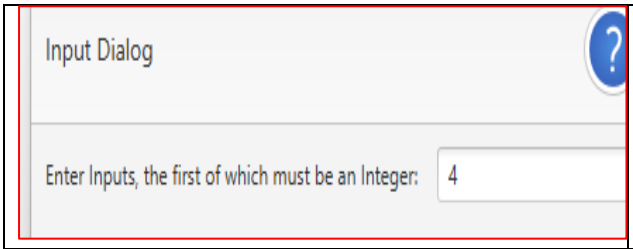
DR<- M[AR]

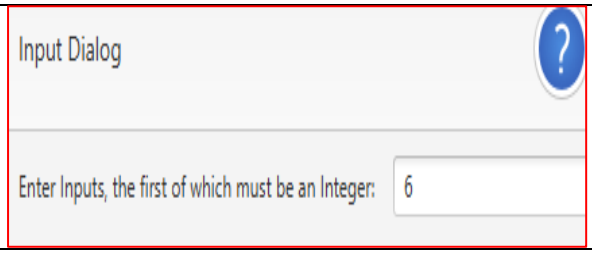
AC<- AC^ DR

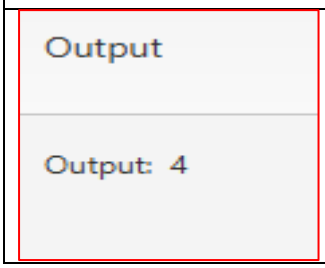
End

IMPLEMENTATION

RESULT:







INPUT 1: 4
INPUT 2: 6
OUTPUT: 4

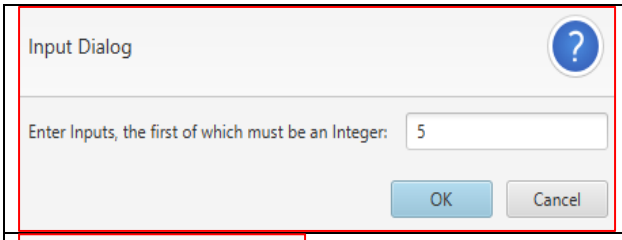
(b)OR

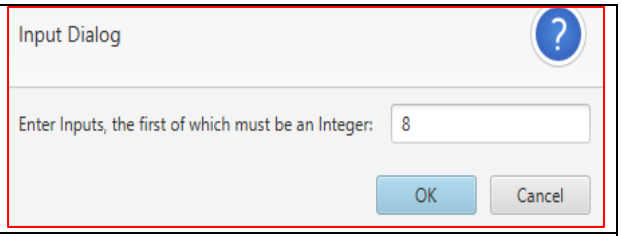
```
1  INP
2  STA A
3  INP
4  STA B
5  LDA A
6  OR B
7  OUT
8  HLT
9  A:  .data 1[5]
10 B:  .data 1[3]
11
```

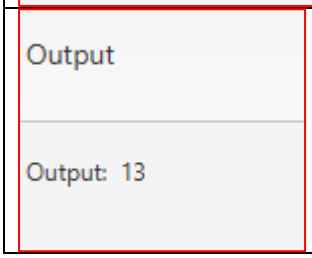
```
DR<- M[AR]

AC<- AC or DR

End
```

CODE | **IMPLEMENTATION**

RESULT:







INPUT 1: 5
INPUT 2: 8
OUTPUT: 13

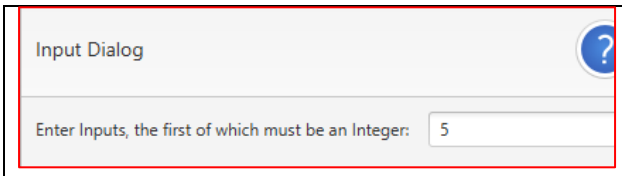
(c)NOT

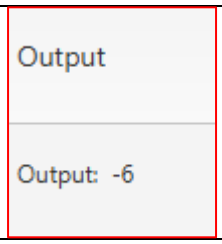
```
1  INP
2  STA A
3  LDA A
4  NOT A
5  OUT
6  HLT
7  A:  .data 1[4]
8
```

```
DR<- M[AR]

AC<- AC'

End
```

CODE | **IMPLEMENTATION**

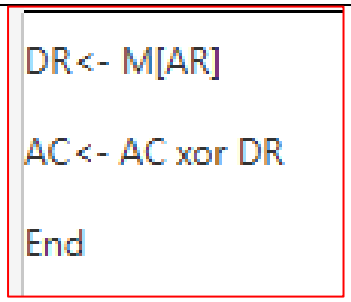
RESULT:





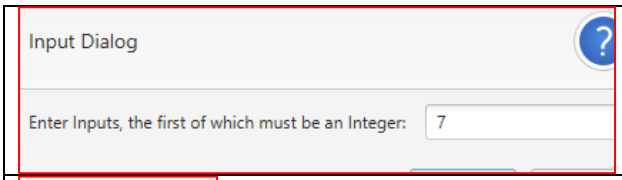
(d)XOR

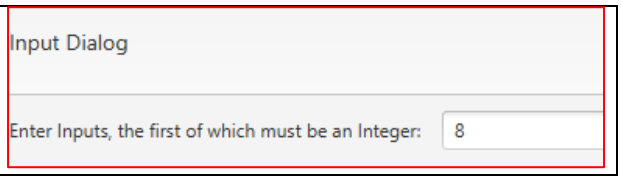
```
1  INP
2  STA A
3  INP
4  STA B
5  LDA A
6  XOR B
7  OUT
8  HLT
9  A: .data 1 [2]
10 B: .data 1 [2]
11
```

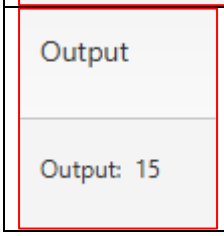


CODE | **IMPLEMENTATION**

RESULT:



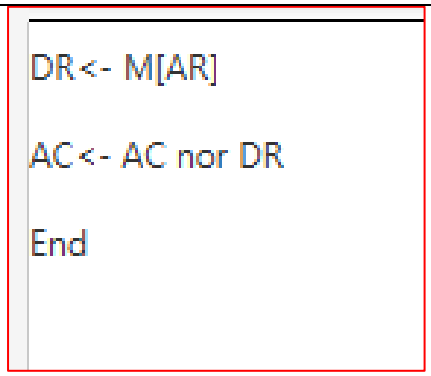




INPUT 1: 7
INPUT 2: 8
OUTPUT: 15

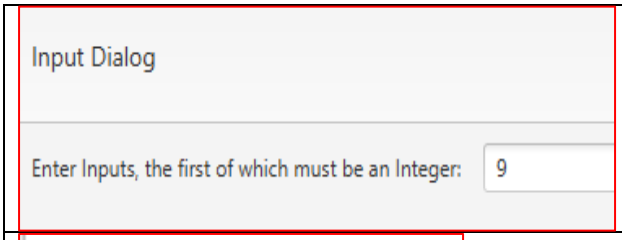
(e)NOR

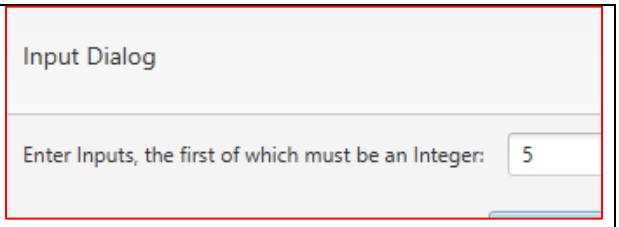
```
1  INP
2  STA A
3  INP
4  STA B
5  LDA A
6  NOR B
7  OUT
8  HLT
9  A: .data 1[5]
10 B: .data 1[4]
11
```



CODE	IMPLEMENTATION

RESULT:

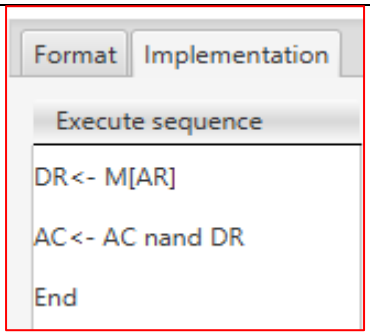




INPUT 1: 9
INPUT 2: 5
OUTPUT: -14

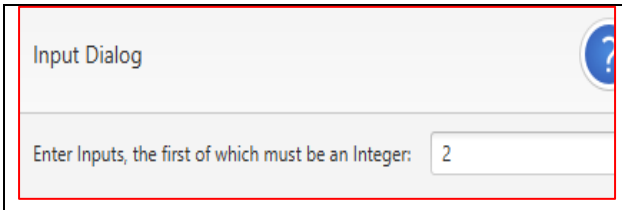
(f)NAND

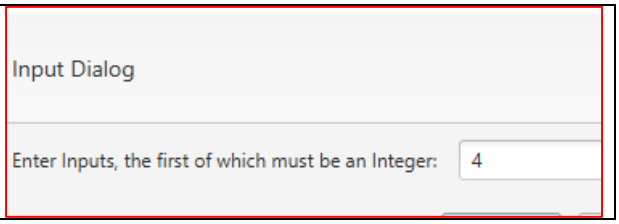
```
 1 INP
 2 STA A
 3 INP
 4 STA B
 5 LDA A
 6 NAND B
 7 OUT
 8 HLT
 9 A: .data 1[2]
10 B: .data 1[4]
```



CODE	IMPLEMENTATION

RESULT:





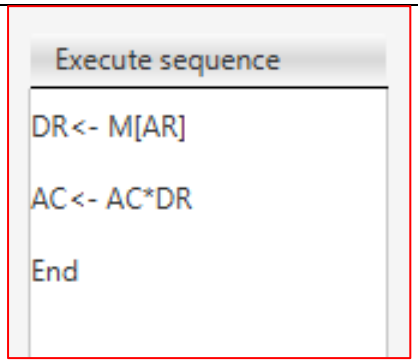
		**INPUT 1: 2** **INPUT 2: 4** **OUTPUT: 0**

Question 6. Write an assembly program to simulate MULTIPLY operation on two user-entered numbers.

q6.a

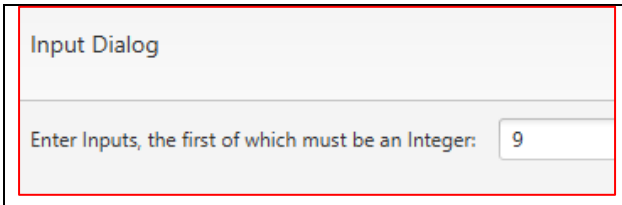
```
1  INP
2  STA A
3  INP
4  STA B
5  LDA A
6  MUL B
7  OUT
8  HLT
9  A: .data 1[4]
10 B: .data 1[6]
11
```

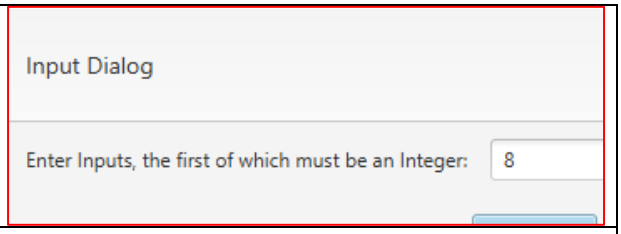
CODE



IMPLEMENTATION

RESULT:





INPUT 1: 9
INPUT 2: 8
OUTPUT: 72

Question 7. Write an assembly program for simulating following memory-reference instructions.

(a)ADD and (b)LDA

Q7(A)(B).a

```
1  LDA A
2  ADD B
3  HLT
4  A:  .data 1[5]
5  B:  .data 1[4]
6
```

CODE

Execute sequence

DR<- M[AR]

AC<- AC+DR

End

ADD

Execute sequence

DR<- M[AR]

AC<- DR

End

LOAD

IMPLEMENTATION

RESULT:

```
A:  .data 1[5]
B:  .data 1[4]
```

Name	Width	Data
AC	16	0009

(c)STA

Q7(A)(B).a | Q7.a | q7(E).a

```
1  INP
2  STA A
3  INP
4  STA B
5  LDA A
6  ADD B
7  OUT
8  HLT
9  A: .data 1 [0]
10 B: .data 1 [0]
11
```

CODE

Execute sequence

M[AR]<- AC

End

IMPLEMENTATION

(d)BUN

Q7(A)(B).a | Q7.a

```
1 LDA Q
2 BUN R
3 ADD S
4 R: AND T
5 HLT
6 Q: .data 1[6]
7 S: .data 1[8]
8 T: .data 1[4]
9
```

Execute sequence

PC<- AR

End

CODE | **IMPLEMENTATION**

RESULT:

```
6 Q: .data 1[6]
7 S: .data 1[8]
8 T: .data 1[4]
9
```

Name	Width	Data
AC	16	0004

(e)ISZ

Q7(A)(B).a | Q7.a

```
1 LDA A
2 ISZ A
3 HLT
4 A: .data 1[6]
5
```

Execute sequence

DR<- M[AR]

DR<- DR+1

M[AR]<- DR

If DR=0 then PC<- PC+1

End

CODE | **IMPLEMENTATION**

RESULT:

```
A: .data 1[6]
```

Name	Width	Data
AC	16	0006

Question 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:

(a)CLA

q8(A).a | q8.a

```
1 LDA A
2 CLA
3 HLT
4 A: .data 1[5]
5
```

code

Execute sequence

AC<- 0

End

implementation

RESULT:

Name	Width	
AC	16	0000

Now, all registers are containing value are:

File Edit Modify Execute Help

Data Dec

Registers

Name	Width	Data
AC	16	0
AR	12	1
DR	16	5
E	1	0
INPR	8	0
IR	16	-8191
OUTR	8	0
PC	12	3
S	1	-1
TR	16	0

(b)CMA

q8(A).a | q8.a | Q8

```
1 LDA A
2 CMA
3 HLT
4 A: .data 1[5]
5
```

code

Execute sequence

AC<- AC'

End

implementation

RESULT:

Name	Width	Data
AC	16	0000 0000 0000 0101

After loading

Registers

Name	Width	Data
AC	16	1111 1111 1111 1010

After CMA

Now, all registers are containing value are:

File Edit Modify Execute Help

Data Dec

Registers

Name	Width	Data
AC	16	-6
AR	12	1
DR	16	5
E	1	0
INPR	8	0
IR	16	-8191
OUTR	8	0
PC	12	3
S	1	-1
TR	16	0

(c)CME and (d)HLT

```
1 LDA A
2 CME
3 HLT
4 A: .data 1[5]
5
```

code

Execute sequence

S<- 1

End

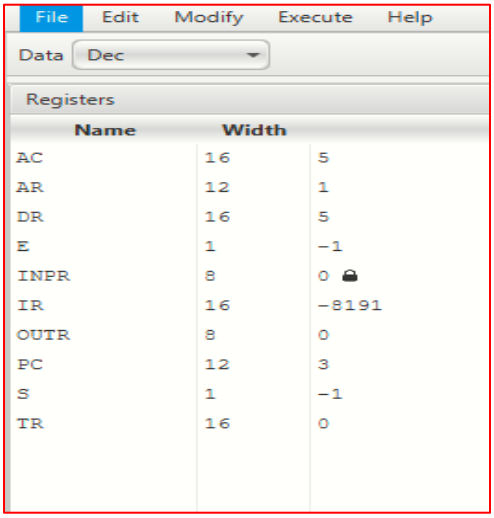
HLT

Execute sequence E<- E' End **CME**	**Implementation**

RESULT:

E 1 0 **After loading**	E 1 1 **After CME**

Now, all registers are containing value are:



Question 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:

(A)INC

Q9(A).a × q9.a ×

```
1 LDA A
2 INC
3 HLT
4 A: .data 1[4]
5
```

code

Execute sequence

AC<- AC+1

End

implementation

RESULT:

Name	Width	Data
AC	16	0000 0000 0000 0100

After loading

Name	Width	Data
AC	16	0000 0000 0000 0101

After increment

Now, all registers are containing value are:

Data Dec

Registers

Name	Width	D
AC	16	5
AR	12	1
DR	16	4
E	1	0
INPR	8	0
IR	16	-8191
OUTR	8	0
PC	12	3
S	1	-1
TR	16	0

(B)SPA

q9.a | Q9(C).a

```
1 LDA A
2 SPA
3 CMA
4 HLT
5 A: .data 1[4]
6
```

code

Execute sequence

If AC![15]=0 then

PC<- PC+1

End

implementation

RESULT:

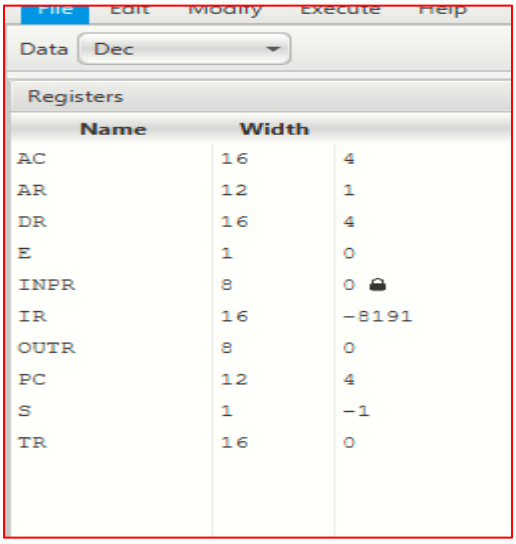
Name	Width	Data
AC	16	0000 0000 0000 0100

After loading

Name	Width	Data
AC	16	0000 0000 0000 0100

After performing SPA

Now, all registers are containing value are:



Name	Width	
AC	16	4
AR	12	1
DR	16	4
E	1	0
INPR	8	0
IR	16	-8191
OUTR	8	0
PC	12	4
S	1	-1
TR	16	0

(c)SNA

Q9(C).a | Q9(D).a

```
1 LDA A
2 SNA
3 CMA
4 HLT
5 A: .data 1[4]
6
```

code

Execute sequence

If AC[15]!=1 then

End

implementation

RESULT:

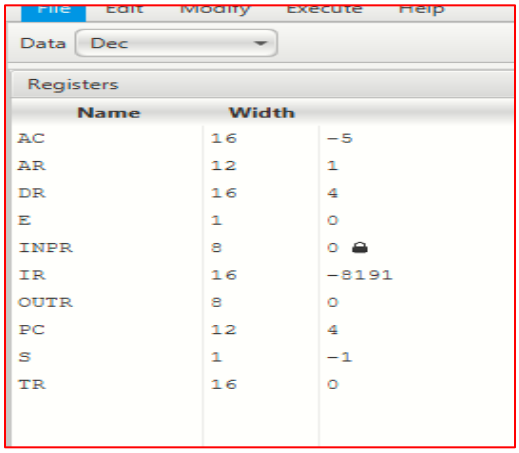
Name	Width	Data
AC	16	0000 0000 0000 0100

After loading

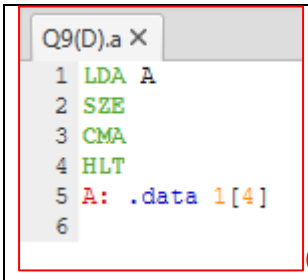
Name	Width	Data
AC	16	1111 1111 1111 1011

After performing SNA

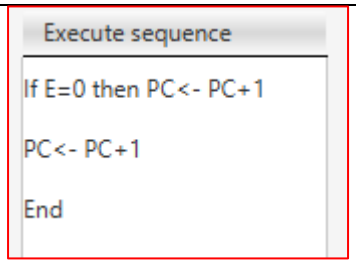
Now, all registers are containing value are:



(d)SZE



code



implementation

RESULT:

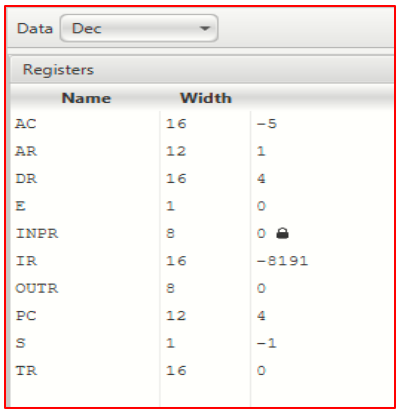
Name	Width	Data
AC	16	0000 0000 0000 0100

After loading

Name	Width	Data
AC	16	1111 1111 1111 1011

After performing SZE

Now, all registers are containing value are:


Data
Dec
Registers
Name
Width
AC 16 -5
AR 12 1
DR 16 4
E 1 0
INPR 8 0
IR 16 -8191
OUTR 8 0
PC 12 4
S 1 -1
TR 16 0

Question 10. **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:**

(a)CIR

Q10.a | Q10(B).a

```
1 LDA A
2 CIR
3 HLT
4 A: .data 1[7]
5
```

code

Execute sequence

E<- AC[0]

AC<- shr AC

AC[15]<- E

End

implementation

RESULT:

Name	Width	Data
AC	16	0000 0000 0000 0111

After loading

Name	Width	Data
AC	16	1000 0000 0000 0011

After CIR

Now, all registers are containing value are:

Data Dec

Registers

Name	Width	D
AC	16	-32765
AR	12	1
DR	16	7
E	1	-1
INPR	8	0
IR	16	-8191
OUTR	8	0
PC	12	3
S	1	-1
TR	16	0

(b)CIL

Q10(B).a

```
1 LDA A
2 CIL
3 HLT
4 A: .data 1[7]
5
```

Code

Execute sequence

```
AC<- shl AC

AC[0]<- E

E<- AC[15]

End
```

implementation

RESULT:

Registers

Name	Width	Data
AC	16	0000 0000 0000 0111

After loading

Registers

Name	Width	Data
AC	16	0000 0000 0000 1110

After CIL

Now, all registers are containing value are:

File Edit Modify Execute Help

Data Dec

Registers

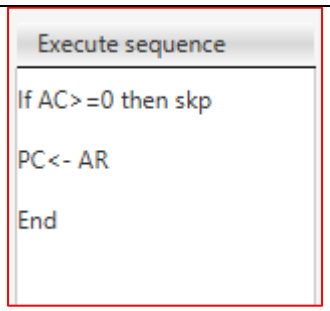
Name	Width	Data
AC	16	14
AR	12	1
DR	16	7
E	1	0
INPR	8	0
IR	16	-8191
OUTR	8	0
PC	12	3
S	1	-1
TR	16	0

Question 11. **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).**

q11.a

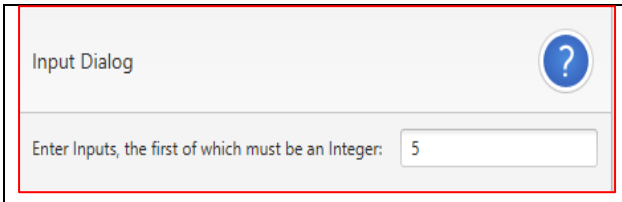
```
1  START: INP
2         JUMP DONE
3         ADD SUM
4         STA SUM
5         BUN START
6  DONE:  LDA SUM
7         OUT
8         HLT
9  SUM:  .data 1[0]
10
```

code

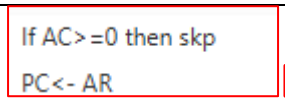


Implementation of JUMP

RESULT:



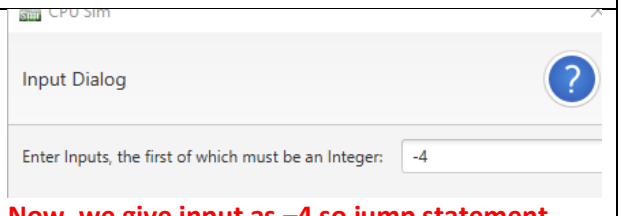
First input gives 5



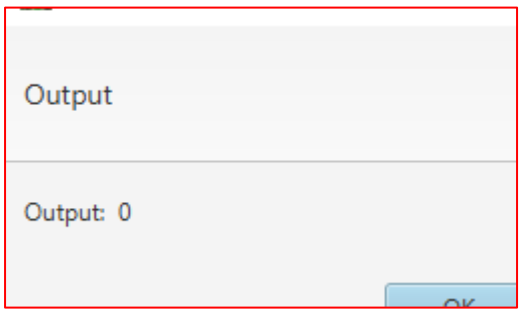
Because of jump statement it found AC >=0 so it skips and move to line 3 ADD SUM

AC	16	0005
AR	12	005
DR	16	0000
E	1	0
INPR	8	00
IR	16	3005
OUTR	8	00
PC	12	002
S	1	0
TR	16	0000

As jump condition true so



Now, we give input as –4 so jump statement would not skip and it will jump to line 6 of DONE column. Where SUM is loaded.



AC =5+0=5 and BUN statement restart from START again.	**And 0 output get.**

Question 12. **Write an assembly program that reads in integers and adds them together; until zero is read in. Then it outputs the sum.**

Q12.a

```
1  START:  INP
2          SZA
3          BUN DONE
4          ADD SUM
5          STA SUM
6          BUN START
7  DONE:   LDA SUM
8          OUT
9          HLT
10 SUM:  .data 1[0]
11
```

code

Execute sequence

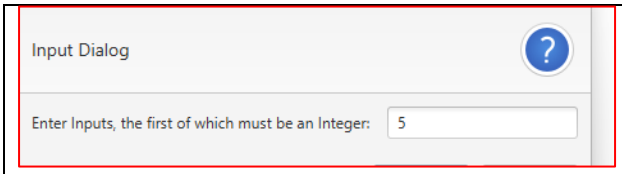
If AC=0 then PC<- PC+1

PC<- PC+1

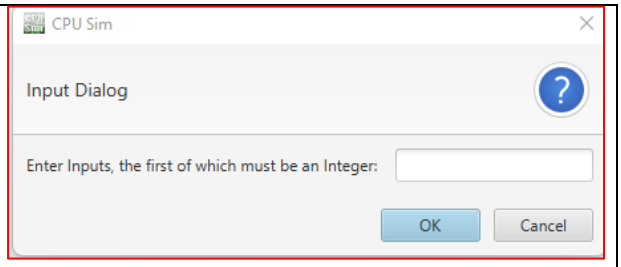
End

Implementation of SZA

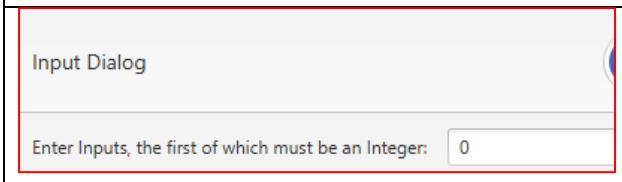
RESULT:



First time take input 5 so it performs BUN to START as SZA condition not satisfied.



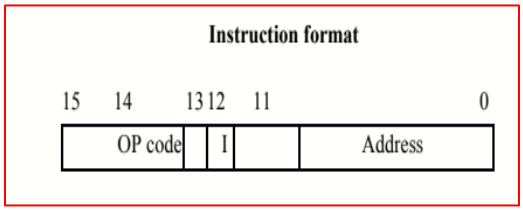
So due to this, go on START again it asks for input again.



Now we give input as 0. SZA condition satisfied. So BUN DONE performed and move to DONE column

SUM Load and give output as 5.

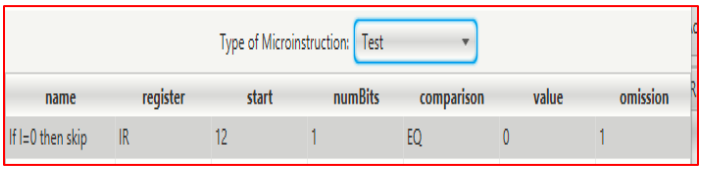
<u>Question 13.</u> Create a machine for the following instruction format:



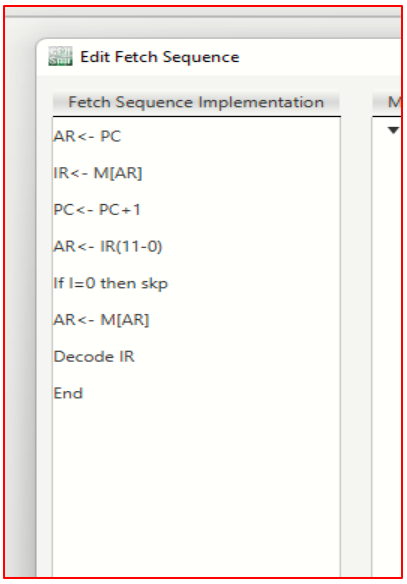
Write an assembly program to simulate the machine for addition of two numbers with direct address part=082. The instruction to be stored at address 022 in RAM, initialize the memory word with any decimal value at address 082. Determine the content of AC, DR, AR, PC and IR in decimal after the execution.

Ans) Now we creating another machine in as same in question 1 and 2 just we are adding a mode bit in our instruction format. And changing some implementation in fetch sequence.

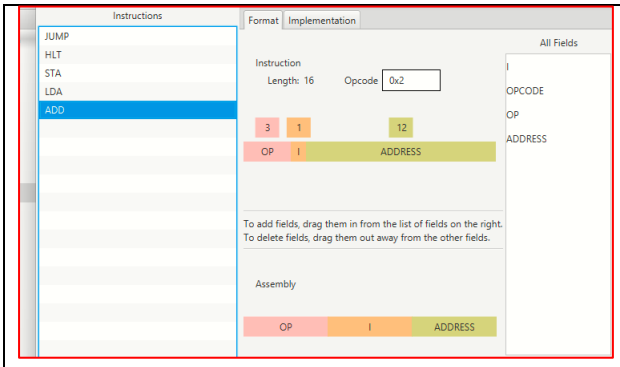
Making a test instruction:



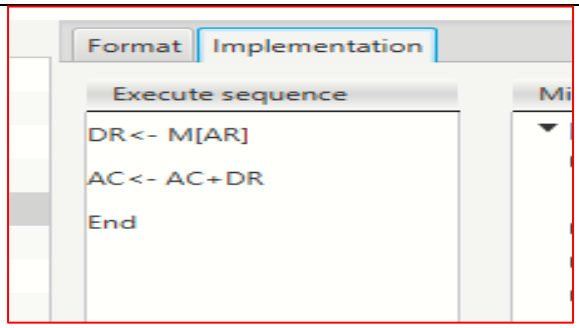
If I=0 then skip in IR register of start bit 12 and number of bits are 1 bit with omission value 1.



Now, during fetching of instruction every time it check the mode bit is direct or indirect and according to that execute the instruction.

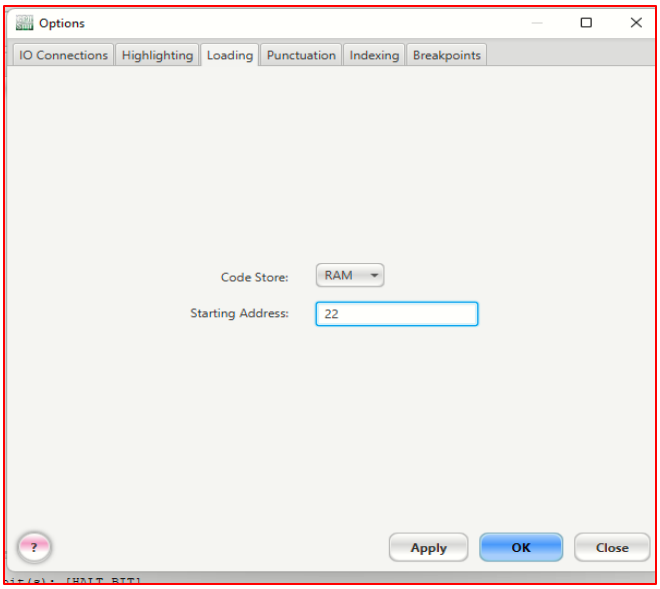


Format of ADD

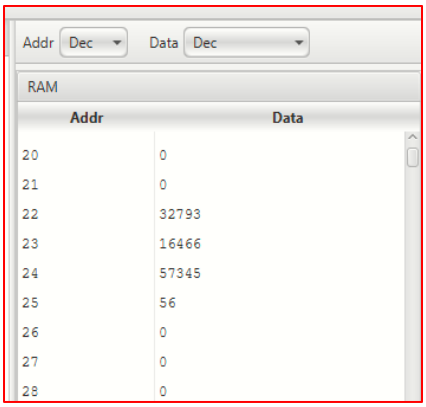


Implementation

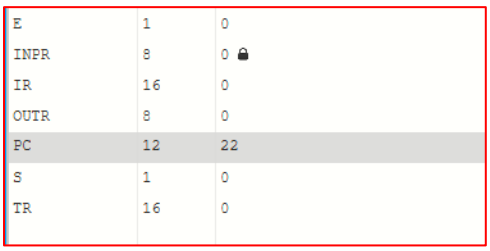
Go to Execute----> then Options ----> loading ----> change starting address from 0 to 22 ----> Apply----> OK.



Now our memory starts from 022 address means that now our instruction is to be stored at memory address 022.



If we want to that our instruction executes according to address 22, we have to save PC with value 22 so that next instruction will be executed as 23 addresses.



addIn.a

```
1 LDA 0 A
2 ADD 0 82
3 HLT
4 A: .data 1[56]
5
```

Code

Data Dec

Registers

Name	Width	Data
AC	16	111
AR	12	1
DR	16	55

Output= 111

Understanding the code:

Firstly, load the value of A into Ac which is 56.

Now the instruction ADD 0 82 is working like that 0 bit represent direct address and at address 82 we will get our second operand in memory.

77	0
78	0
79	0
80	0
81	0
82	55
83	0
84	0
85	0

And then ADD operation perform between 56 and 55 which gives 111 as output stored in AC.

Now, other registers contain values after execution of program:

Data Dec

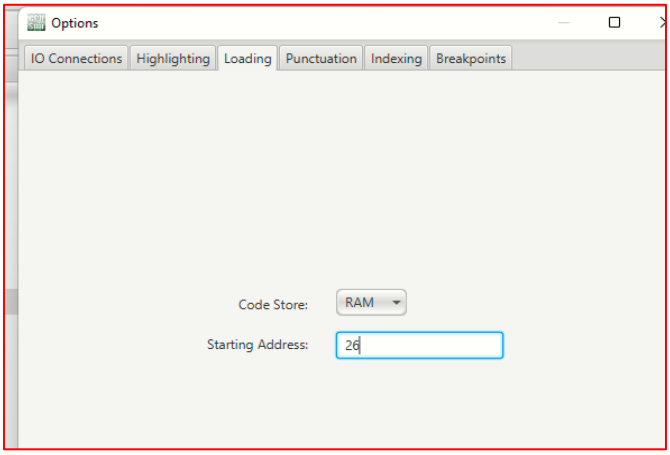
Registers

Name	Width	
AC	16	111
AR	12	1
DR	16	55
E	1	0
INPR	8	0
IR	16	-8191
OUTR	8	0
PC	12	25
S	1	-1
TR	16	0

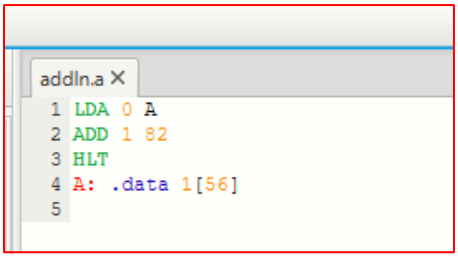
Question 14. **Simulate the machine for the memory-reference instruction referred in above question with I=1 (Indirect Address) and address part= 082. The instruction to be stored at address 026 in RAM. Initialize the memory word at address 082 with the value 298. Initialize the memory word at address 298 with operand 632 and AC with 937. Determine the contents of AC, DR, PC, AR and IR in decimal after the execution.**

Ans)

Go to Execute----> then Options ----> loading ----> change starting address from 0 to 26 ----> Apply----> OK.

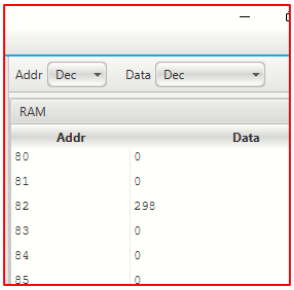


Writing the Code:

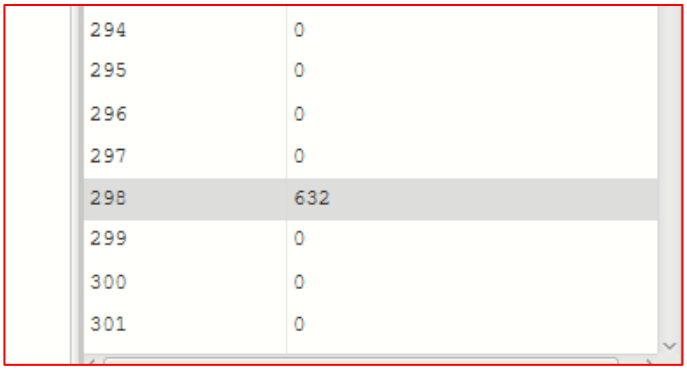


Understanding the code:

First, we load A in AC then the statement ADD 1 82 means at address 82 there is an address of second operand because of mod bit I=1.

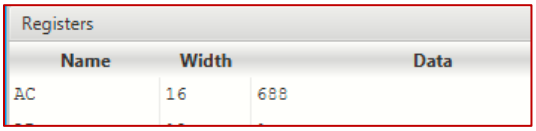


According to the question we put address (298) of operand at address 82.

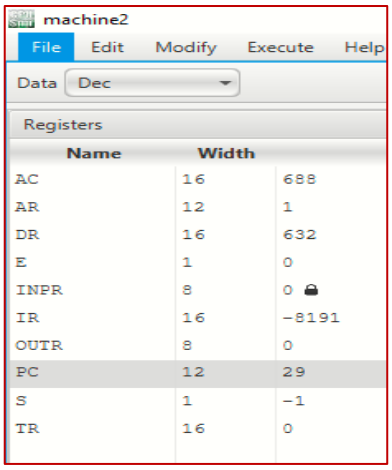


And effective address contain operand 632.

Output: 56+632= 688.

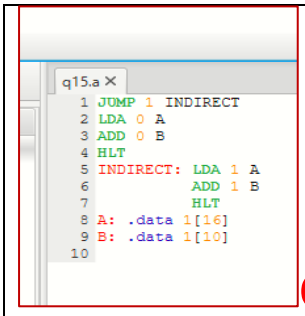


Now, other registers contain values after execution of program:

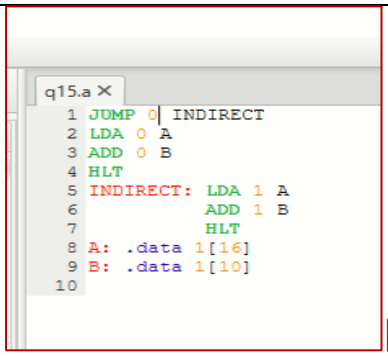

machine2
File
Edit
Modify
Execute
Help
Data
Dec
Registers
Name
Width
AC 16 688
AR 12 1
DR 16 632
E 1 0
INPR 8 0
IR 16 -8191
OUTR 8 0
PC 12 29
S 1 -1
TR 16 0

Question 15. **The instruction format contains 3 bits of opcode, 12 bits for address and 1 bit for addressing mode There are only two addressing modes, I=0 is direct addressing and I=1 is indirect addressing. Write an assembly program to check the I bit to determine the addressing mode and then jump accordingly.**

Ans)

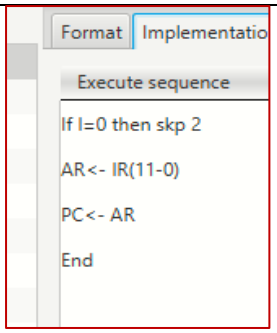


Code For I=1



For I=0

Format of Jump.



Implementation of Jump.

Output for I=1 is 15+12=27.

Output for I=0 is 16+10=26.

Explanation:

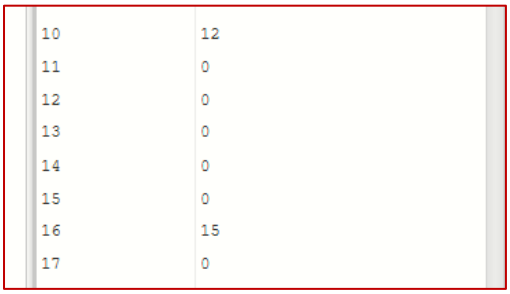
(I) For I=0

In jump statement when we get I=0 then we skip it's all microinstructions of jump and move to next instruction that is addition of A (16) and B (10) in direct addressing mode and we do not go to INDIRECT field on line no.5 in code. So simply it adds A+B means 16+10.

(II) For I=1

So, we don't get I=0 here in 1st line of code so it will not skip the microinstructions of the jump statement and because of it PC makes it move to line 5 of INDIRECT addition.

As A contains 16 so it acts as address rather than operand and its same as with B contains 10 so it acts as address rather than operand.



10	12
11	0
12	0
13	0
14	0
15	0
16	15
17	0

So it adds 12+15 rather than 10+16 so we get 27 as output.