

**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***

Qno.	Practical Questions	Page No.
1	Create Machine	5
2	Create Fetch Routine of the instruction cycle.	16
3	Assembly program for Add operation	20
4	Write assembly program for Subtract operation.	21
5	Program for logical operation input both numbers by user.	22
6	Write assembly program for Multiply operation.	27
7	Write program for Memory reference instructions. (ADD, LDA, STA, BUN, ISZ)	28
8	Write program for Register reference instructions. (CLA, CMA, CME, HLT)	30
9	Write program for Register reference instructions. (INC, SPA, SNA, SZE)	33
10	Write program for Register reference instructions. (CIR, CIL)	37

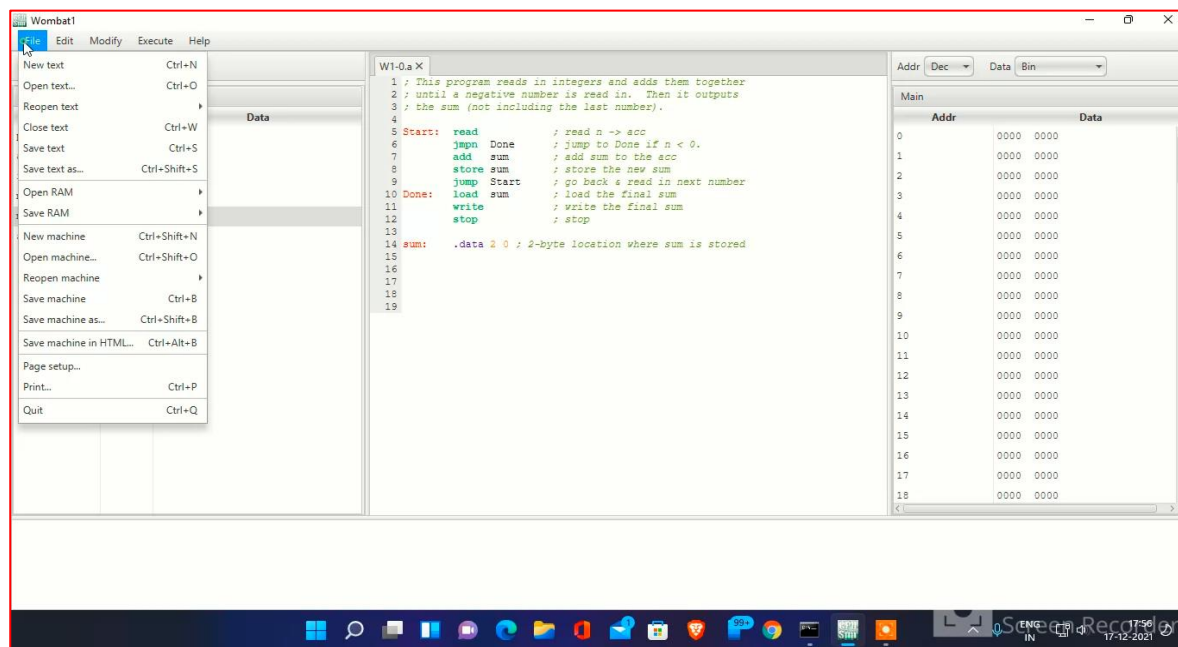
11	Write a program that reads two integers and adds them together until a negative no read.	39
12	Write program that reads and add two integers until 0 is read.	41
13	Create a new machine with indirect bit. And a write a program to add two no with initializing memory from 022.	42
14	Determine the value 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 ARCHITECTURE

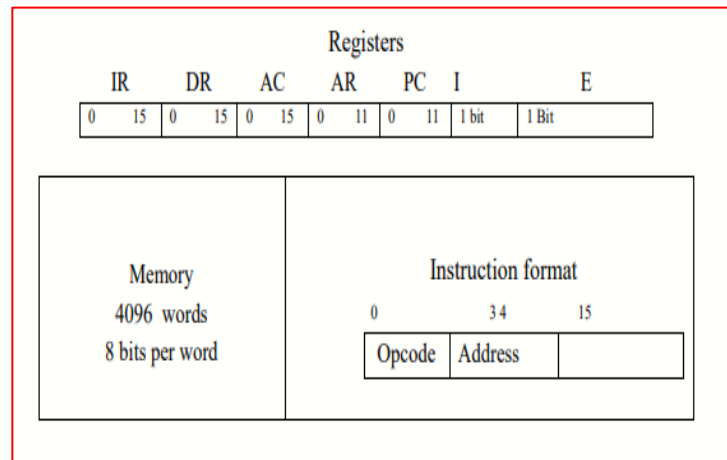
## CPU SIMULATOR

### 1. Main Page of CPU SIM

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

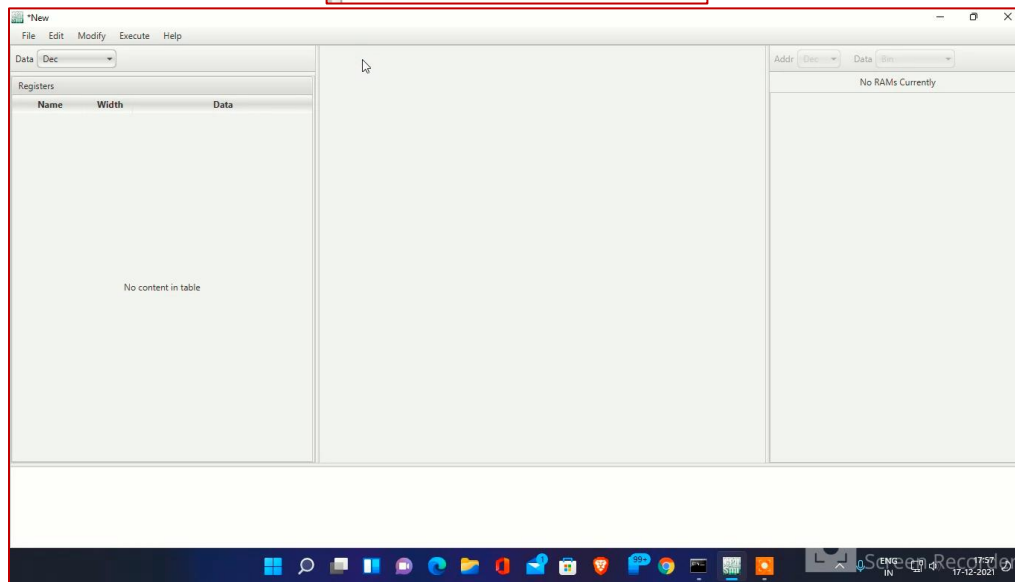
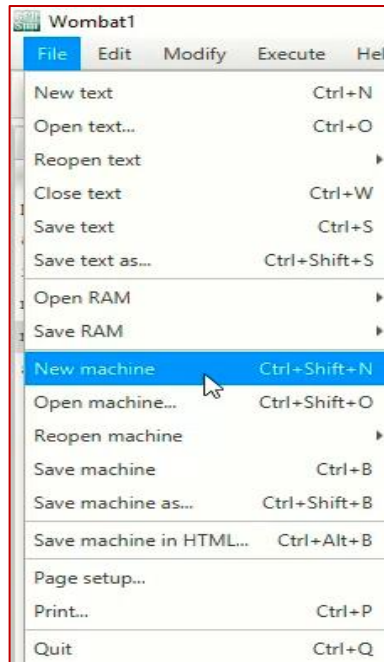


**Question 1. Create a machine based on the following architecture:**



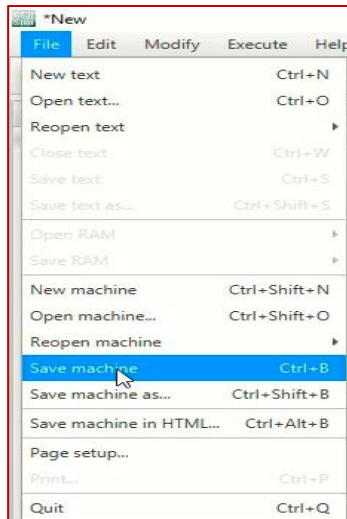
## 2. Create new machine

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

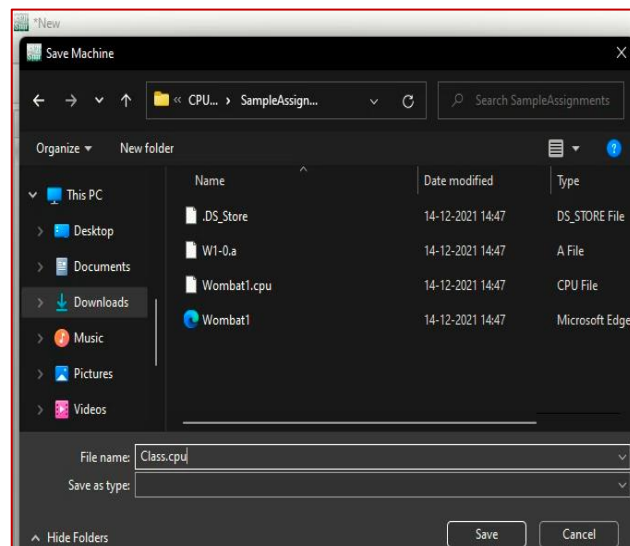


### 3. Save the machine.

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**

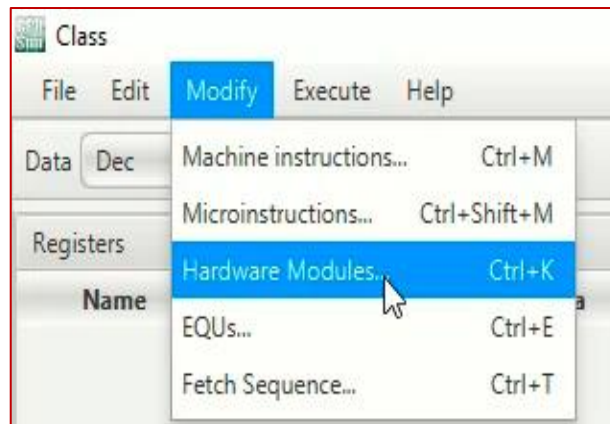


Now machine show its name as Class and after every work it show \* sign with name that means the work on machine not save so we have to save our work everytime.



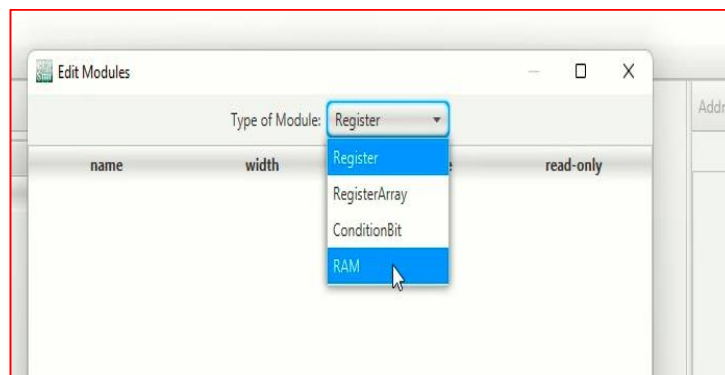
## 4. Using Hardware Modules

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



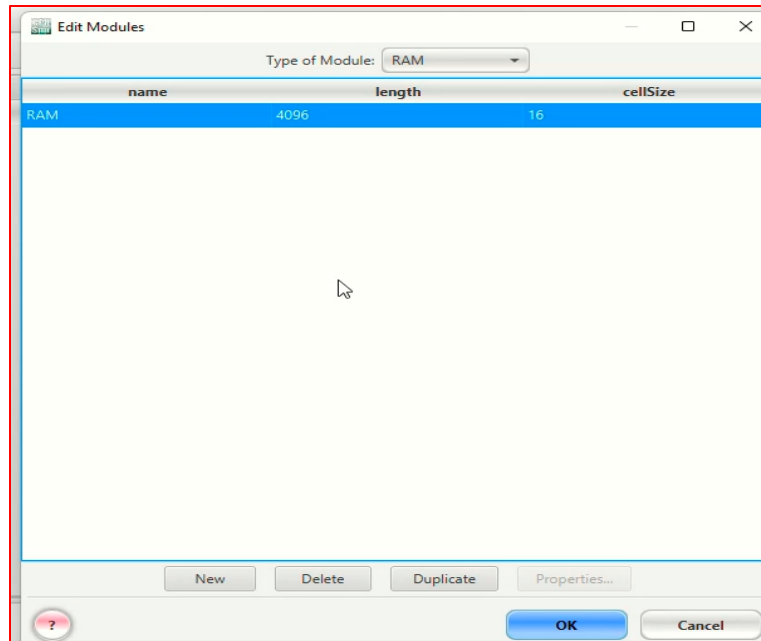
### A) Make a memory for your machine

Then select “Type of Module” as RAM...



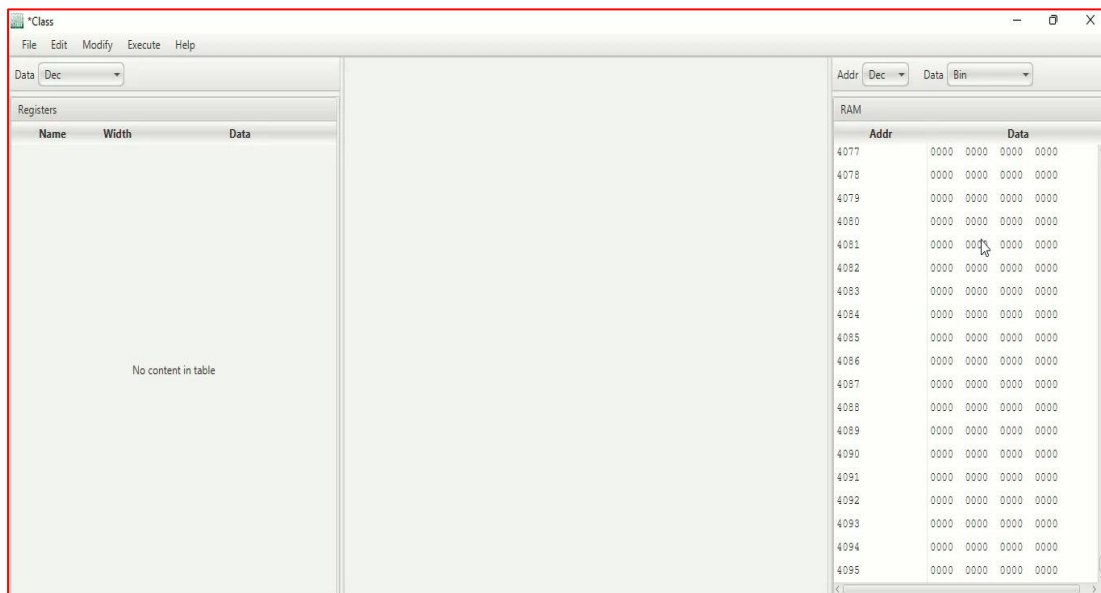
Click on new to make new memory and Name to your memory like RAM , select length of memory that you want like 4096 and put cell size as 16.





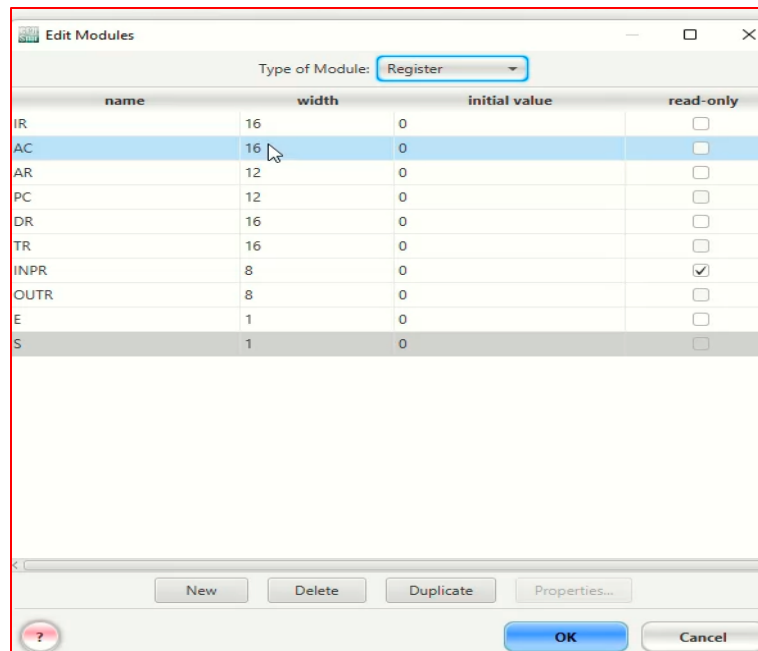
Select “OK” to save changes.

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



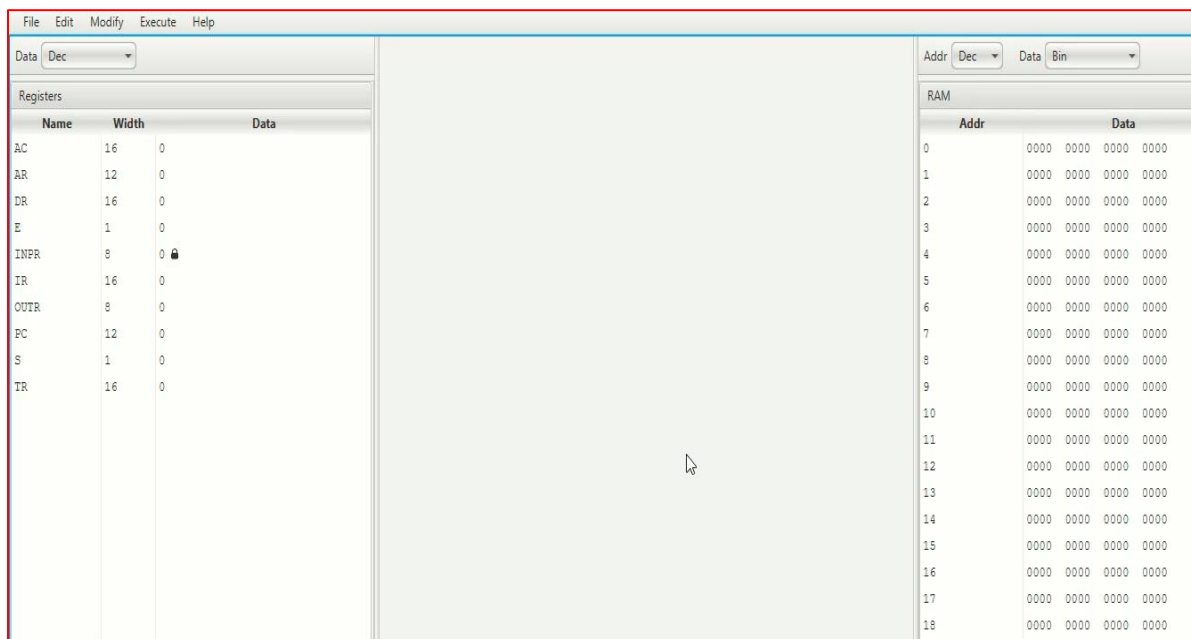
## B) Create registers for your machine.

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.



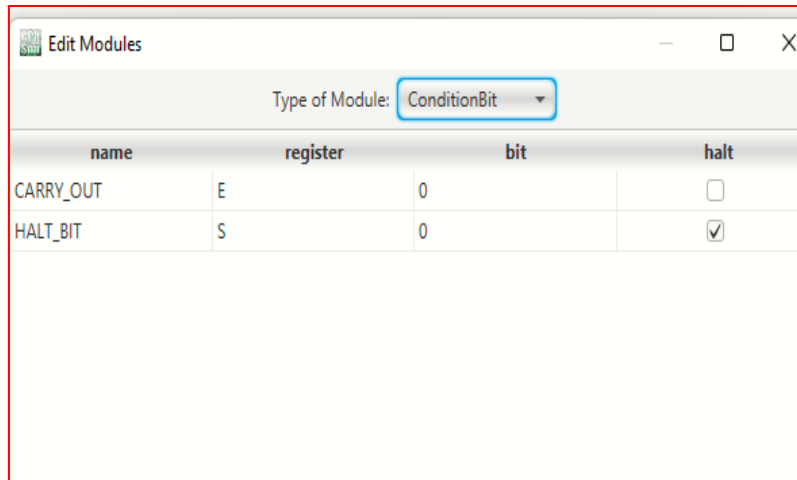
Click OK to save changes...

Now your Registers represents on left side of your machine look like given figure.



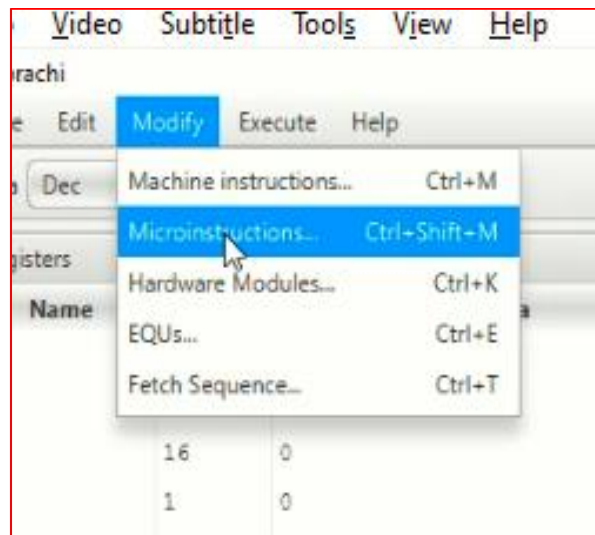
### C) Create condition bit.

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



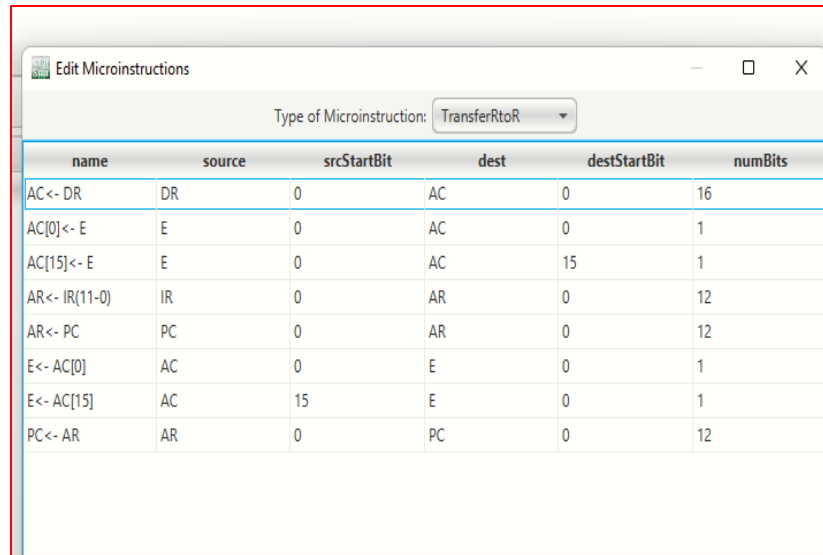
## 5. Using microinstructions

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



### A) Writing instructions for transferring instruction from register to register

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



The screenshot shows a window titled "Edit Microinstructions". Below the title bar, there is a dropdown menu labeled "Type of Microinstruction:" with "TransferRtoR" selected. Below this is a table with the following columns: name, source, srcStartBit, dest, destStartBit, and numBits. The table contains the following rows:

name	source	srcStartBit	dest	destStartBit	numBits
AC <- DR	DR	0	AC	0	16
AC[0] <- E	E	0	AC	0	1
AC[15] <- E	E	0	AC	15	1
AR <- IR(11-0)	IR	0	AR	0	12
AR <- PC	PC	0	AR	0	12
E <- AC[0]	AC	0	E	0	1
E <- AC[15]	AC	15	E	0	1
PC <- AR	AR	0	PC	0	12

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....

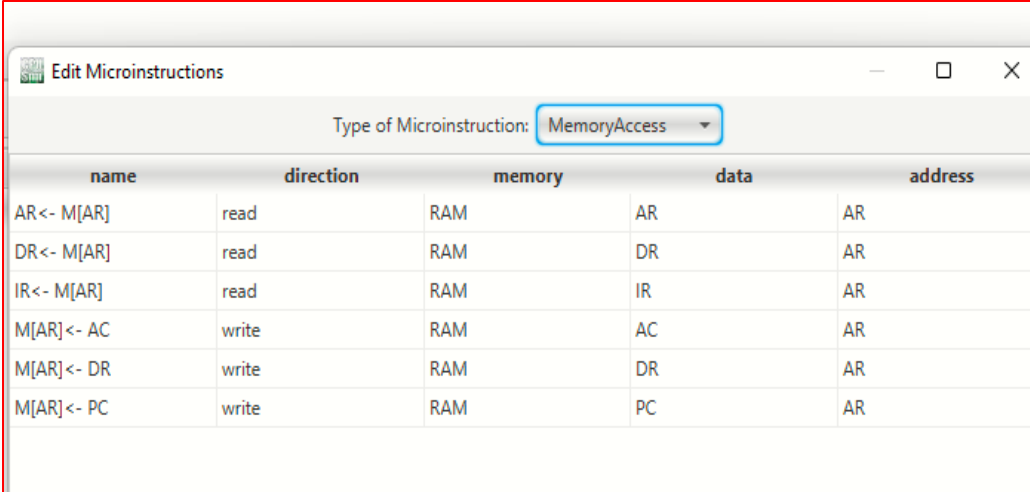
And destStartBit is till which bit is ending....

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...

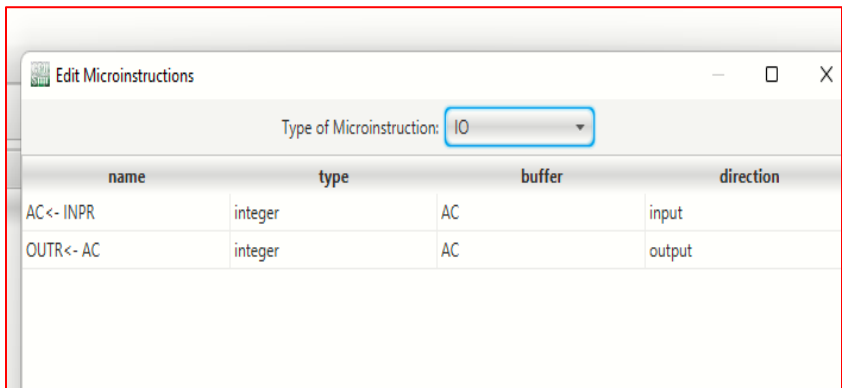


The screenshot shows a window titled "Edit Microinstructions". At the top, there is a dropdown menu labeled "Type of Microinstruction:" with "MemoryAccess" selected. Below this is a table with five columns: name, direction, memory, data, and address. The table contains six rows of microinstructions.

name	direction	memory	data	address
AR <- M[AR]	read	RAM	AR	AR
DR <- M[AR]	read	RAM	DR	AR
IR <- M[AR]	read	RAM	IR	AR
M[AR] <- AC	write	RAM	AC	AR
M[AR] <- DR	write	RAM	DR	AR
M[AR] <- PC	write	RAM	PC	AR

### C) Writing microinstructions for input output registers.

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



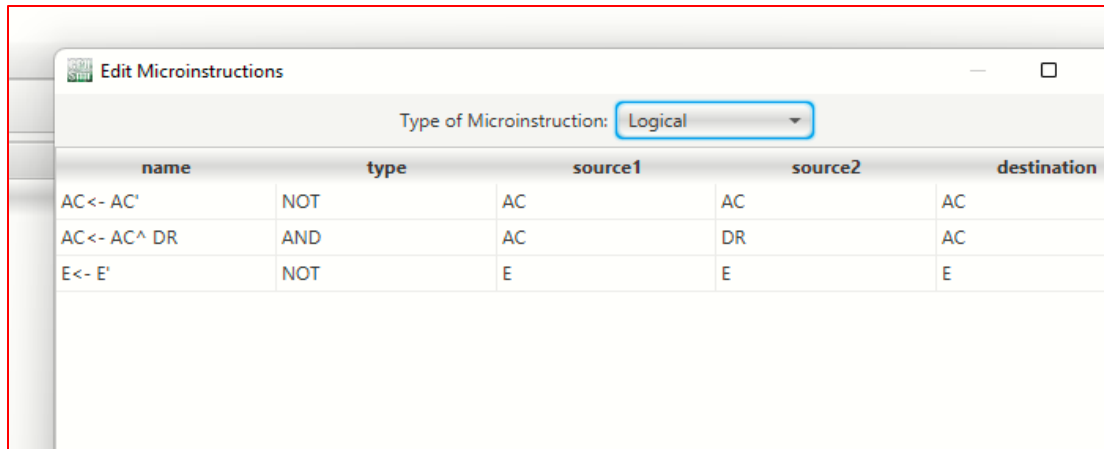
The screenshot shows the same "Edit Microinstructions" window, but now the dropdown menu is set to "IO". The table below has four columns: name, type, buffer, and direction. It contains two rows of microinstructions.

name	type	buffer	direction
AC <- INPR	integer	AC	input
OUTR <- AC	integer	AC	output

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

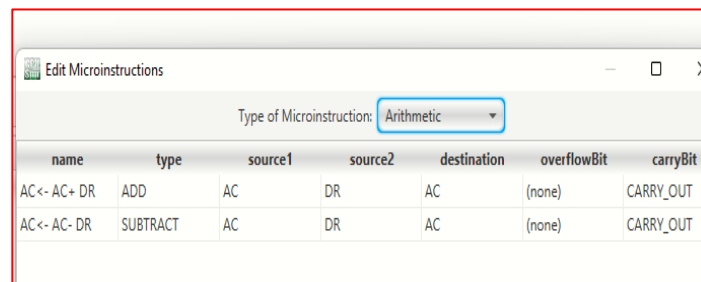
### D) Writing microinstructions to apply logical expressions (AND, NOT etc.) with the data in registers and flip-flops.

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



## E) Writing microinstructions to do arithmetic operations on registers.

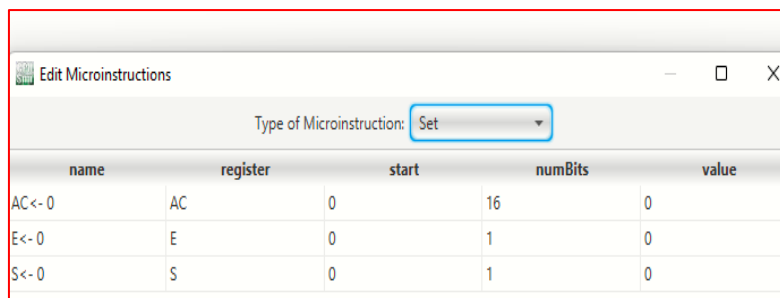
Go to Type of microinstructions----> choose Arithmetic.



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.

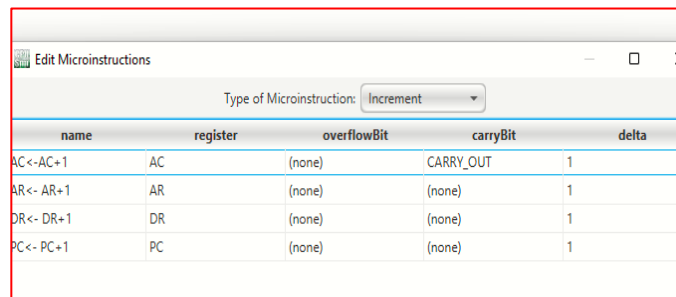
## F) Write set microinstructions.

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



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

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

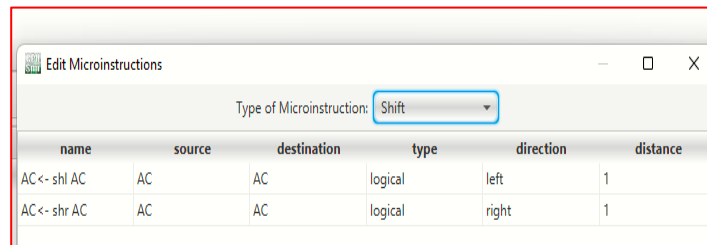


The screenshot shows the 'Edit Microinstructions' window. The 'Type of Microinstruction' dropdown is set to 'Increment'. Below the dropdown is a table with the following data:

name	register	overflowBit	carryBit	delta
AC <- AC + 1	AC	(none)	CARRY_OUT	1
AR <- AR + 1	AR	(none)	(none)	1
DR <- DR + 1	DR	(none)	(none)	1
PC <- PC + 1	PC	(none)	(none)	1

**H) Writing microinstruction to shift left or right.**

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

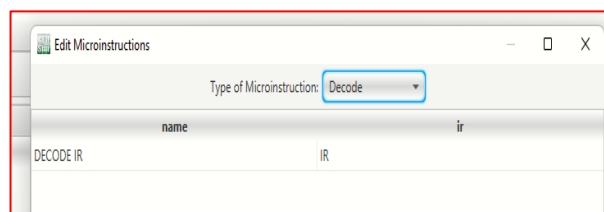


The screenshot shows the 'Edit Microinstructions' window. The 'Type of Microinstruction' dropdown is set to 'Shift'. Below the dropdown is a table with the following data:

name	source	destination	type	direction	distance
AC <- shl AC	AC	AC	logical	left	1
AC <- shr AC	AC	AC	logical	right	1

**I) Writing microinstruction to decode by instruction register.**

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

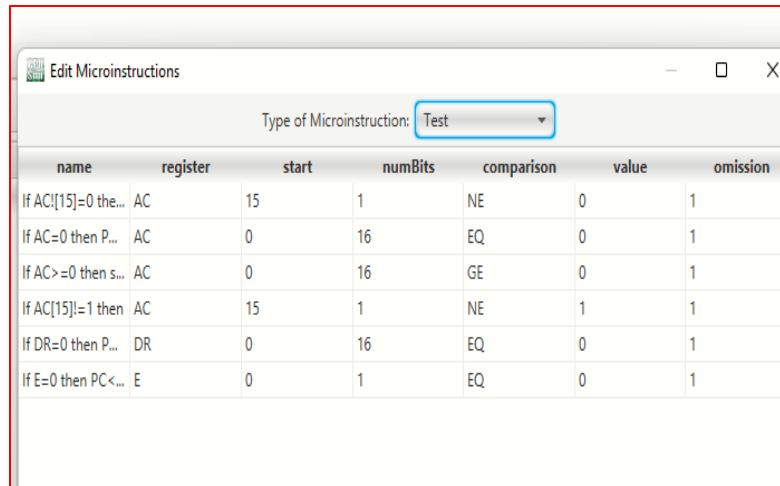


The screenshot shows the 'Edit Microinstructions' window. The 'Type of Microinstruction' dropdown is set to 'Decode'. Below the dropdown is a table with the following data:

name	ir
DECODE IR	IR

**J) Writing microinstructions to test instructions.**

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

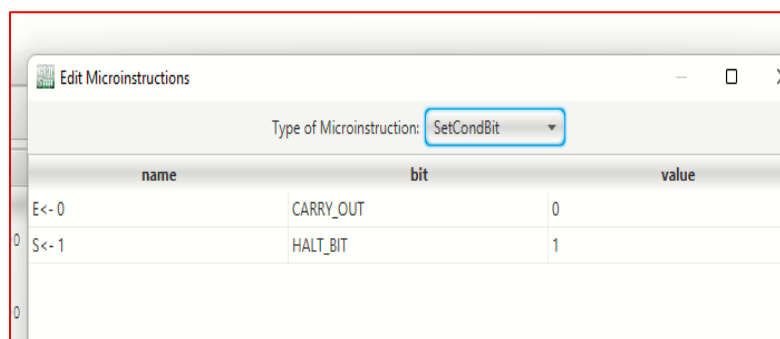


Edit Microinstructions  
 Type of Microinstruction: Test

name	register	start	numBits	comparison	value	omission
If AC[15]=0 then...	AC	15	1	NE	0	1
If AC=0 then P...	AC	0	16	EQ	0	1
If AC>=0 then s...	AC	0	16	GE	0	1
If AC[15]!=1 then	AC	15	1	NE	1	1
If DR=0 then P...	DR	0	16	EQ	0	1
If E=0 then PC<...	E	0	1	EQ	0	1

## K) Writing microinstructions to setConditionBit.

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

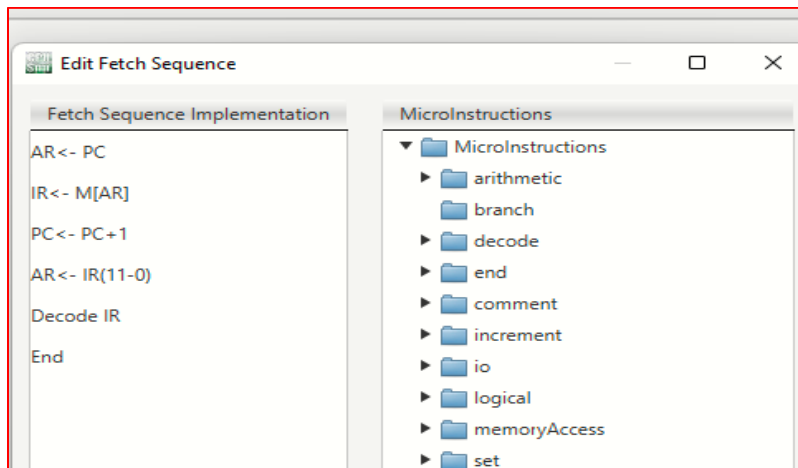


Edit Microinstructions  
 Type of Microinstruction: SetCondBit

name	bit	value
E<- 0	CARRY_OUT	0
S<- 1	HALT_BIT	1



## Question 2. 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

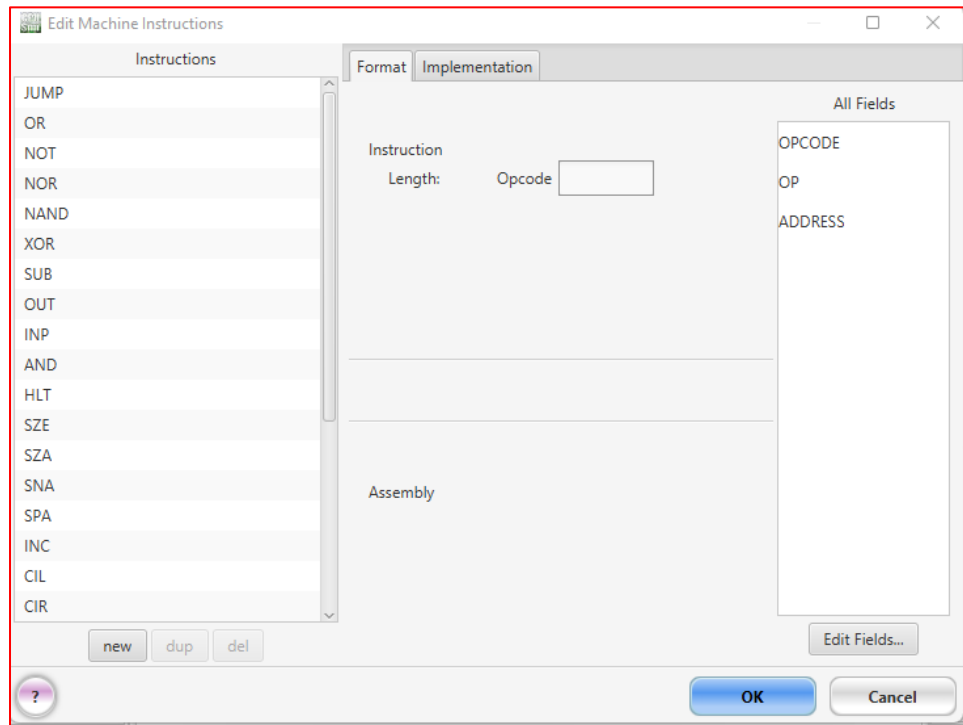
T2: D0, ..., D7<--- Decode IR (12-14), AR<--- IR(0-11), I<--- IR(15)

\*\*\*\*\*

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.

Name	Type	NumBits	DefaultVal	Relativity	Signed
OPCO...	required	16	0	absolute	<input type="checkbox"/>
OP	required	4	0	absolute	<input type="checkbox"/>
ADDR...	required	12	0	absolute	<input type="checkbox"/>

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

Format
Implementation

Instruction
Length: 16
Opcode
0x2

4

12

OP
ADDRESS

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

OP
ADDRESS

**For all memory reference instructions, the format is same.**

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

Format
Implementation

Execute sequence

DR<- M[AR]  
AC<- AC+DR  
End

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

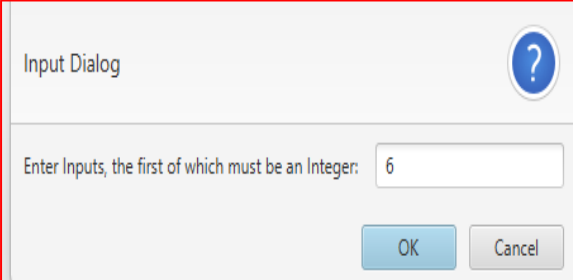
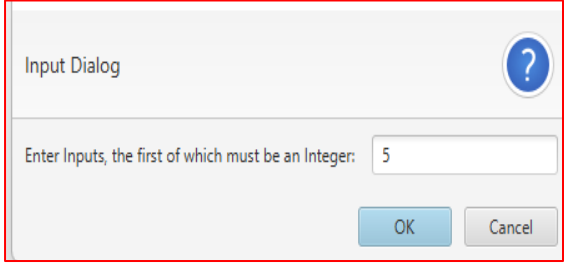
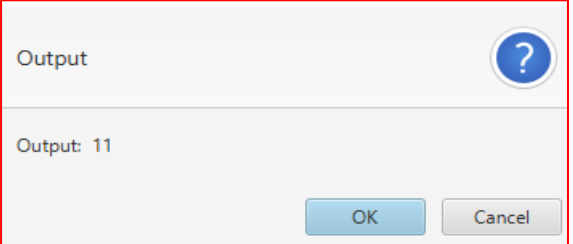
### Basic Computer Instructions

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.**

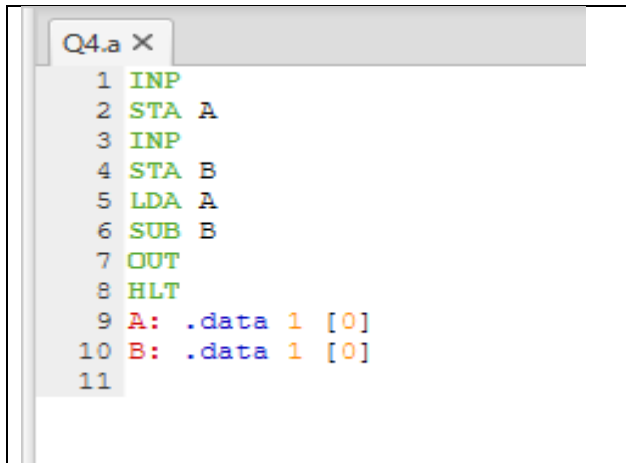
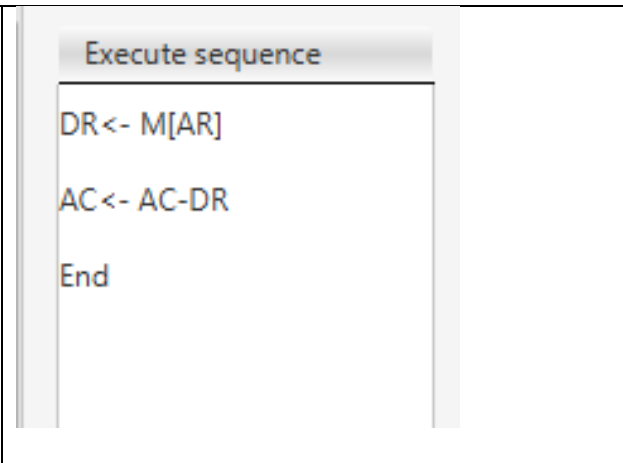
```
q3.a X
1 INP
2 STA A
3 INP
4 STA B
5 LDA A
6 ADD B
7 OUT
8 HLT
9 A: .data 1 [0]
10 B: .data 1 [0]
11
```

**RESULT:**

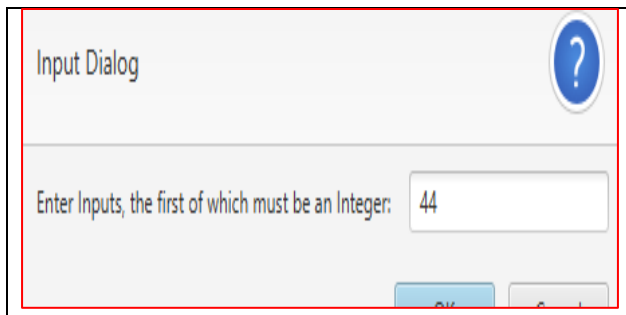
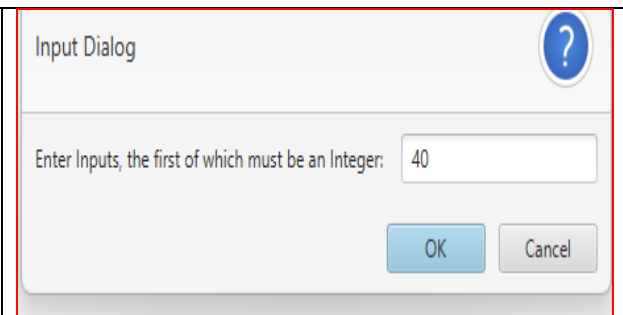
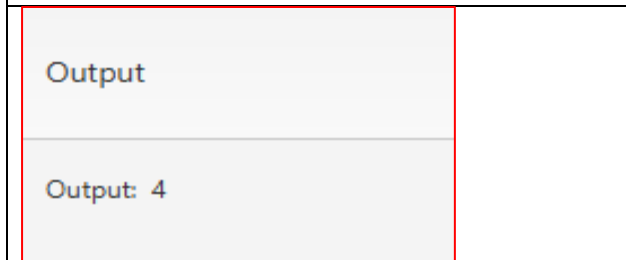
	
	<b>Input 1= 5 Input 2= 6 Output= 11</b>

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

**Ans)**

	
<b>CODE</b>	<b>IMPLEMENTATION</b>

**RESULT:**

	
	<b><u>INPUT 1: 44</u></b> <b><u>INPUT 2: 40</u></b> <b><u>OUTPUT: 4</u></b>

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

**(a)AND**

<pre> 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 </pre>	<pre> DR&lt;- M[AR]  AC&lt;- AC^ DR  End </pre>
CODE	IMPLEMENTATION

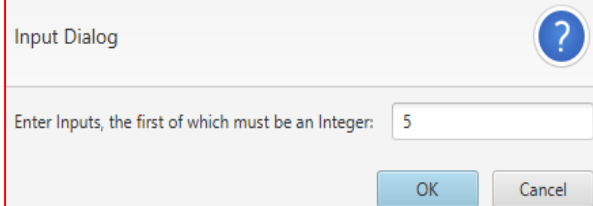
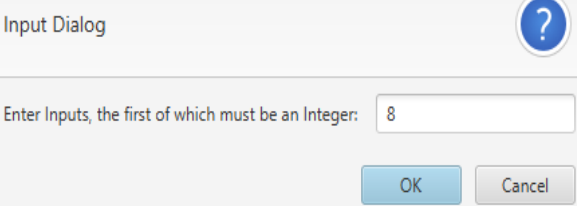
**RESULT:**

<p>Input Dialog</p> <p>Enter Inputs, the first of which must be an Integer: <input type="text" value="4"/></p>	<p>Input Dialog</p> <p>Enter Inputs, the first of which must be an Integer: <input type="text" value="6"/></p>
<p>Output</p> <p>Output: 4</p>	<p><b>INPUT 1: 4</b>  <b>INPUT 2: 6</b>  <b>OUTPUT: 4</b></p>

**(b)OR**

<pre> 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 </pre>	<pre> DR&lt;- M[AR]  AC&lt;- AC or DR  End </pre>
CODE	IMPLEMENTATION

RESULT:

 <p>Input Dialog</p> <p>Enter Inputs, the first of which must be an Integer: <input type="text" value="5"/></p> <p>OK Cancel</p>	 <p>Input Dialog</p> <p>Enter Inputs, the first of which must be an Integer: <input type="text" value="8"/></p> <p>OK Cancel</p>
<p>Output</p> <hr/> <p>Output: 13</p>	<p><b>INPUT 1: 5</b>  <b>INPUT 2: 8</b>  <b>OUTPUT: 13</b></p>

(c)NOT

<pre> 1 INP 2 STA A 3 LDA A 4 NOT A 5 OUT 6 HLT 7 A: .data 1[4] 8 </pre>	<pre> DR&lt;- M[AR]  AC&lt;- AC'  End </pre>
CODE	IMPLEMENTATION

RESULT:



<div>Input Dialog</div> <div>Enter Inputs, the first of which must be an Integer: <input type="text" value="5"/></div>	<div>Output</div> <div>Output: -6</div>
--	---

## (d)XOR

<pre> 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 </pre>	<pre> DR&lt;- M[AR]  AC&lt;- AC xor DR  End </pre>
<b>CODE</b>	<b>IMPLEMENTATION</b>

### RESULT:

<div>Input Dialog</div> <div>Enter Inputs, the first of which must be an Integer: <input type="text" value="7"/></div>	<div>Input Dialog</div> <div>Enter Inputs, the first of which must be an Integer: <input type="text" value="8"/></div>
<div>Output</div> <div>Output: 15</div>	<b>INPUT 1: 7</b> <b>INPUT 2: 8</b> <b>OUTPUT: 15</b>

## (e)NOR

<pre> 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 </pre>	<pre> DR&lt;- M[AR]  AC&lt;- AC nor DR  End </pre>
--	--

CODE	IMPLEMENTATION
------	----------------

RESULT:

<p>Input Dialog</p> <p>Enter Inputs, the first of which must be an Integer: 9</p>	<p>Input Dialog</p> <p>Enter Inputs, the first of which must be an Integer: 5</p>
<p>Output</p> <p>Output: -14</p>	<p><b>INPUT 1: 9</b>  <b>INPUT 2: 5</b>  <b>OUTPUT: -14</b></p>

(f)NAND

<pre> 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] 11 </pre>	<p>Format Implementation</p> <p>Execute sequence</p> <pre> DR&lt;- M[AR] AC&lt;- AC nand DR End </pre>
CODE	IMPLEMENTATION

RESULT:

<p>Input Dialog</p> <p>Enter Inputs, the first of which must be an Integer: 2</p>	<p>Input Dialog</p> <p>Enter Inputs, the first of which must be an Integer: 4</p>
---	---

<div data-bbox="215 191 634 331"><p>Output</p></div> <div data-bbox="215 331 634 491"><p>Output: 0</p></div>		<p><b>INPUT 1: 2</b> <b>INPUT 2: 4</b> <b>OUTPUT: 0</b></p>
--	--	---

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

<div><div>q6.a X</div><div><div>1 INP</div><div>2 STA A</div><div>3 INP</div><div>4 STA B</div><div>5 LDA A</div><div>6 MUL B</div><div>7 OUT</div><div>8 HLT</div><div>9 A: .data 1[4]</div><div>10 B: .data 1[6]</div><div>11</div></div></div>	<div><div>Execute sequence</div><div><div>DR&lt;- M[AR]</div><div>AC&lt;- AC*DR</div><div>End</div></div></div>
CODE	IMPLEMENTATION

**RESULT:**

<div>Input Dialog</div> <div>Enter Inputs, the first of which must be an Integer: <input type="text" value="9"/></div>	<div>Input Dialog</div> <div>Enter Inputs, the first of which must be an Integer: <input type="text" value="8"/></div>
<div>Output</div> <div>Output: 72</div>	<p><b>INPUT 1: 9</b>  <b>INPUT 2: 8</b>  <b>OUTPUT: 72</b></p>

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

**(a)ADD and (b)LDA**

<pre>Q7(A)(B).a X 1 LDA A 2 ADD B 3 HLT 4 A: .data 1[5] 5 B: .data 1[4] 6</pre>	CODE	<pre>Execute sequence DR&lt;- M[AR] AC&lt;- AC+DR End</pre>	ADD
<pre>Execute sequence DR&lt;- M[AR] AC&lt;- DR End</pre>	LOAD	IMPLEMENTATION	

**RESULT:**

<pre>A: .data 1[5] B: .data 1[4]</pre>		<table border="1"> <thead> <tr> <th>Name</th><th>Width</th><th>Data</th></tr> </thead> <tbody> <tr> <td>AC</td><td>16</td><td>0009</td></tr> </tbody> </table>	Name	Width	Data	AC	16	0009
Name	Width	Data						
AC	16	0009						

**(c)STA**

<pre>Q7(A)(B).a X  Q7.a X  q7(E).a X 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</pre>		<pre>Execute sequence M[AR]&lt;- AC End</pre>	
CODE		IMPLEMENTATION	

## (d)BUN

<pre> Q7(A)(B).a ×  Q7.a ×  c 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 </pre>	<div>Execute sequence</div> <pre> PC&lt;- AR End </pre>
<b>CODE</b>	<b>IMPLEMENTATION</b>

RESULT:

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

Name	Width	Data
AC	16	0004

## (e)ISZ

<pre> Q7(A)(B).a ×  Q7.a ×  c 1 LDA A 2 ISZ A 3 HLT 4 A: .data 1[6] 5 </pre>	<div>Execute sequence</div> <pre> DR&lt;- M[AR] DR&lt;- DR+1 M[AR]&lt;- DR If DR=0 then PC&lt;- PC+1 End </pre>
<b>CODE</b>	<b>IMPLEMENTATION</b>

RESULT:

A: .data 1[6]	<table><tr><th>Name</th><th>Width</th><th>Data</th></tr><tr><td>AC</td><td>16</td><td>0006</td></tr></table>	Name	Width	Data	AC	16	0006
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**

<div> <div>q8(A).a ×</div> <div>q8.a ×</div> <div> 1 LDA A  2 CLA  3 HLT  4 A: .data 1[5]  5 </div> </div>	<div>code</div>	<div>Execute sequence</div> <div>AC&lt;- 0</div> <div>End</div>	<div>implementation</div>
--	-----------------	---	---------------------------

**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**

<div style="border: 1px solid red; padding: 5px;"> <pre> 1 LDA A 2 CMA 3 HLT 4 A: .data 1[5] 5 </pre> </div> <p style="color: red; font-weight: bold; margin-top: 10px;">code</p>	<div style="border: 1px solid red; padding: 5px;"> <p>Execute sequence</p> <pre> AC&lt;- AC' End </pre> </div> <p style="color: red; font-weight: bold; margin-top: 10px;">implementation</p>
---	---

**RESULT:**

<table border="1" style="width: 100%;"> <thead> <tr> <th>Name</th> <th>Width</th> <th>Data</th> </tr> </thead> <tbody> <tr> <td>AC</td> <td>16</td> <td>0000 0000 0000 0101</td> </tr> </tbody> </table> <p style="color: red; font-weight: bold; margin-top: 10px;">After loading</p>	Name	Width	Data	AC	16	0000 0000 0000 0101	<table border="1" style="width: 100%;"> <thead> <tr> <th colspan="3">Registers</th> </tr> <tr> <th>Name</th> <th>Width</th> <th>Data</th> </tr> </thead> <tbody> <tr> <td>AC</td> <td>16</td> <td>1111 1111 1111 1010</td> </tr> </tbody> </table> <p style="color: red; font-weight: bold; margin-top: 10px;">After CMA</p>	Registers			Name	Width	Data	AC	16	1111 1111 1111 1010
Name	Width	Data														
AC	16	0000 0000 0000 0101														
Registers																
Name	Width	Data														
AC	16	1111 1111 1111 1010														

**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**

<div style="border: 1px solid red; padding: 5px;"> <pre> 1 LDA A 2 CME 3 HLT 4 A: .data 1[5] 5 </pre> </div> <p style="color: red; font-weight: bold; margin-top: 10px;">code</p>	<div style="border: 1px solid red; padding: 5px;"> <p>Execute sequence</p> <pre> S&lt;- 1 End </pre> </div> <p style="color: red; font-weight: bold; margin-top: 10px;">HLT</p>
---	---



<div>Execute sequence</div> <div>E&lt;- E'</div> <div>End</div>	CME	Implementation
---	-----	----------------

RESULT:

E	1	0
After loading		

E	1	1
After CME		

Now, all registers are containing value are:

File	Edit	Modify	Execute	Help
Data	Dec			
Registers				
Name	Width			
AC	16	5		
AR	12	1		
DR	16	5		
E	1	-1		
INPR	8	0		
IR	16	-8191		
OUTR	8	0		
PC	12	3		
S	1	-1		
TR	16	0		

**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**

<div>Q9(A).a × q9.a ×</div> <pre> 1 LDA A 2 INC 3 HLT 4 A: .data 1[4] 5 </pre> <p><b>code</b></p>	<div>Execute sequence</div> <pre> AC&lt;- AC+1 End </pre> <p><b>implementation</b></p>
---	--

**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**

<div> <div>q9.a X Q9(C).a X</div> <pre> 1 LDA A 2 SPA 3 CMA 4 HLT 5 A: .data 1[4] 6 </pre> </div> <div>code</div>	<div> <div>Execute sequence</div> <pre> If AC![15]=0 then PC&lt;- PC+1 End </pre> </div> <div>implementation</div>
---	--

RESULT:


<table border="1"> <thead> <tr> <th>Name</th> <th>Width</th> <th>Data</th> </tr> </thead> <tbody> <tr> <td>AC</td> <td>16</td> <td>0000 0000 0000 0100</td> </tr> </tbody> </table> <div>After loading</div>	Name	Width	Data	AC	16	0000 0000 0000 0100	<table border="1"> <thead> <tr> <th>Name</th> <th>Width</th> <th>Data</th> </tr> </thead> <tbody> <tr> <td>AC</td> <td>16</td> <td>0000 0000 0000 0100</td> </tr> </tbody> </table> <div>After performing SPA</div>	Name	Width	Data	AC	16	0000 0000 0000 0100
Name	Width	Data											
AC	16	0000 0000 0000 0100											
Name	Width	Data											
AC	16	0000 0000 0000 0100											

Now, all registers are containing value are:

File Edit Modify Execute Help

Data Dec

Registers

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

<div> <div>Q9(C).a X Q9(D).a X</div> <pre> 1 LDA A 2 SNA 3 CMA 4 HLT 5 A: .data 1[4] 6 </pre> </div> <div>code</div>	<div> <div>Execute sequence</div> <pre> If AC[15]!=1 then End </pre> </div> <div>implementation</div>
--	---

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:

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

(d)SZE

<pre> Q9(D).a X 1 LDA A 2 SZE 3 CMA 4 HLT 5 A: .data 1[4] 6 </pre> <p><b>code</b></p>	<pre> Execute sequence If E=0 then PC&lt;- PC+1 PC&lt;- PC+1 End </pre> <p><b>implementation</b></p>
---	--

**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**

<div style="border: 1px solid gray; padding: 5px;"> <div style="display: flex; justify-content: space-between;"> <span>Q10.a X</span> <span>Q10(B).a X</span> </div> <pre> 1 LDA A 2 CIR 3 HLT 4 A: .data 1[7] 5 </pre> </div> <p style="color: red; text-align: center; margin-top: 10px;"><b>code</b></p>	<div style="border: 1px solid gray; padding: 5px;"> <div style="background-color: #f0f0f0; padding: 2px; text-align: center; border-bottom: 1px solid gray;">Execute sequence</div> <pre> E&lt;- AC[0] AC&lt;- shr AC AC[15]&lt;- E End </pre> </div> <p style="color: red; text-align: center; margin-top: 10px;"><b>implementation</b></p>
---	--

**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:

Registers		
Name	Width	Data
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

```

Execute sequence

```

AC <- shl AC
AC[0] <- E
E <- AC[15]
End

```

Code
implementation

**RESULT:**

Registers

Name	Width	Data
AC	16	0000 0000 0000 0111

Registers

Name	Width	Data
AC	16	0000 0000 0000 1110

After loading
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).

<pre> q11.a X 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         </pre>	<p>code</p>	<p>Execute sequence</p> <p>If AC &gt;= 0 then skp</p> <p>PC &lt;- AR</p> <p>End</p> <p><b>Implementation of JUMP</b></p>
--	-------------	--

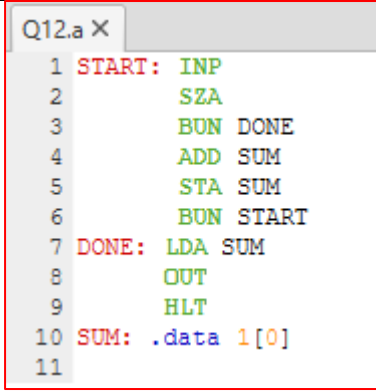
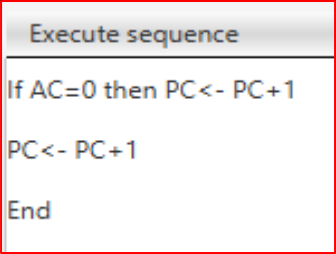
**RESULT:**

<p>Input Dialog</p> <p>Enter Inputs, the first of which must be an Integer: 5</p> <p><b>First input gives 5</b></p>	<p>If AC &gt;= 0 then skp</p> <p>PC &lt;- AR</p> <p><b>Because of jump statement it found AC &gt;= 0 so it skips and move to line 3 ADD SUM</b></p>																														
<table border="1"> <tr><td>AC</td><td>16</td><td>0005</td></tr> <tr><td>AR</td><td>12</td><td>005</td></tr> <tr><td>DR</td><td>16</td><td>0000</td></tr> <tr><td>E</td><td>1</td><td>0</td></tr> <tr><td>INPR</td><td>8</td><td>00</td></tr> <tr><td>IR</td><td>16</td><td>3005</td></tr> <tr><td>OUTR</td><td>8</td><td>00</td></tr> <tr><td>PC</td><td>12</td><td>002</td></tr> <tr><td>S</td><td>1</td><td>0</td></tr> <tr><td>TR</td><td>16</td><td>0000</td></tr> </table> <p><b>As jump condition true so</b></p>	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	<p>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.</p> <p>Output</p> <p>Output: 0</p>
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																													

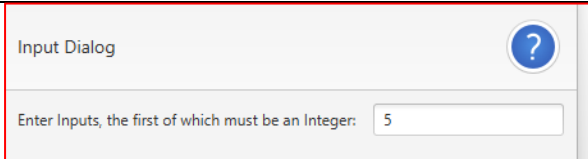
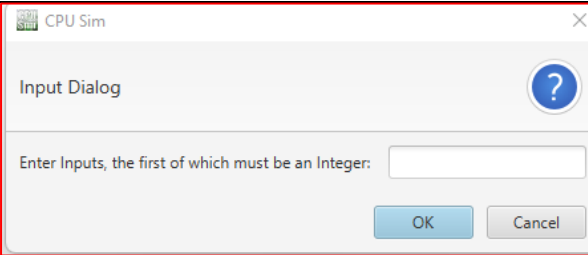
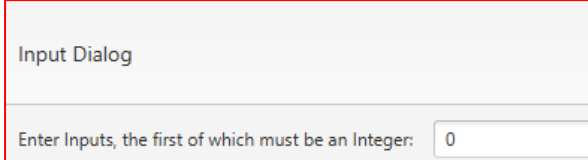



<b>AC =5+0=5 and BUN statement restart from START again.</b>	<b>And 0 output get.</b>
--	--------------------------

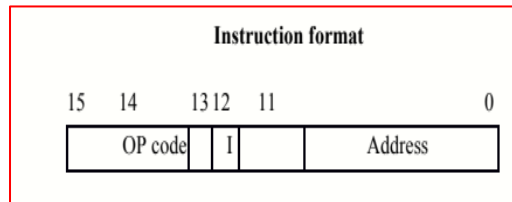
**Question 12.** Write an assembly program that reads in integers and adds them together; until zero is read in. Then it outputs the sum.

 <pre> 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 </pre>	code	 <p><b>Implementation of SZA</b></p>
---	------	--

**RESULT:**

 <p><b>First time take input 5 so it performs BUN to START as SZA condition not satisfied.</b></p>	 <p><b>So due to this, go on START again it asks for input again.</b></p>
 <p><b>Now we give input as 0. SZA condition satisfied. So BUN DONE performed and move to DONE column</b></p>	 <p><b>SUM Load and give output as 5.</b></p>

**Question 13.** Create a machine for the following instruction format:



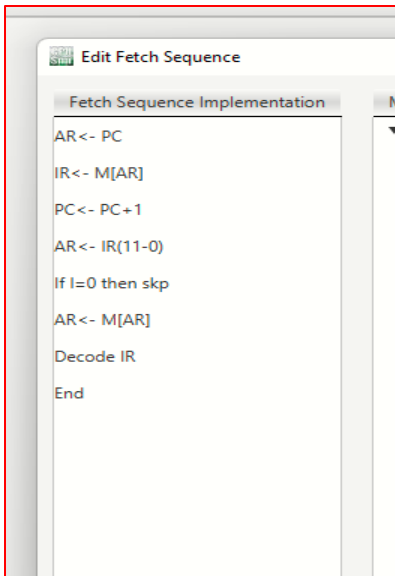
Write an assembly program to simulate the machine for addition of two numbers with direct address part=082. The instruction to be stored at address 022 in RAM, initialize the memory word with any decimal value at address 082. Determine the content of AC, DR, AR, PC and IR in decimal after the execution.

**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:

Type of Microinstruction: <span style="border: 1px solid blue; padding: 2px;">Test</span>						
name	register	start	numBits	comparison	value	omission
If I=0 then skip	IR	12	1	EQ	0	1

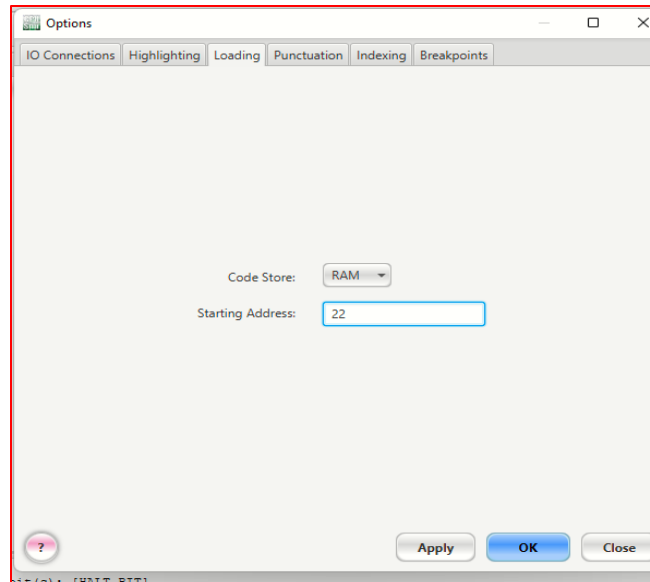
If I=0 then skip in IR register of start bit 12 and number of bits are 1 bit with omission value 1.



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

<p><b>Format of ADD</b></p>	<p><b>Implementation</b></p>
-----------------------------	------------------------------

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

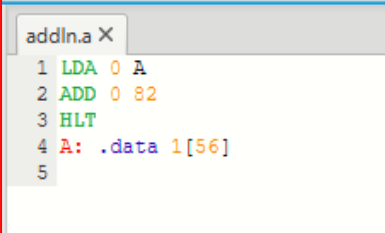


Now our memory starts from 022 address means that now our instruction is to be stored at memory address 022.

Addr	Dec	Data	Dec
RAM			
Addr		Data	
20		0	
21		0	
22		32793	
23		16466	
24		57345	
25		56	
26		0	
27		0	
28		0	

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.

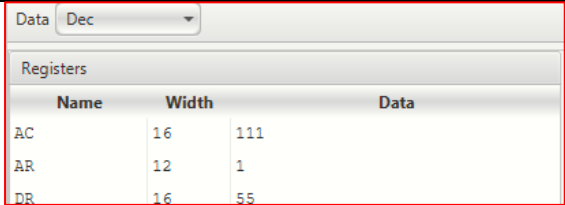
E	1	0
INPR	8	0
IR	16	0
OUTR	8	0
PC	12	22
S	1	0
TR	16	0



```

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

```



**Output= 111**

Code

## 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.

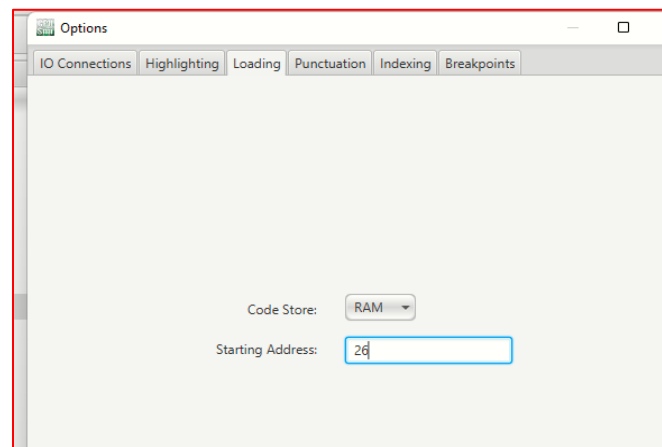
Now, other registers contain values after execution of program:

Data <span>Dec</span>		
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

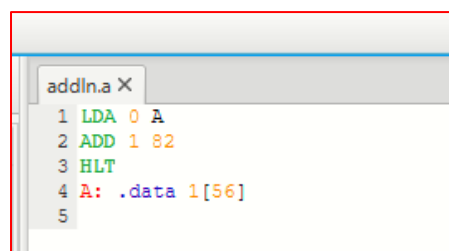
**Question 14.** Simulate the machine for the memory-reference instruction referred in above question with I=1 (Indirect Address) and address part= 082. The instruction to be stored at address 026 in RAM. Initialize the memory word at address 082 with the value 298. Initialize the memory word at address 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.



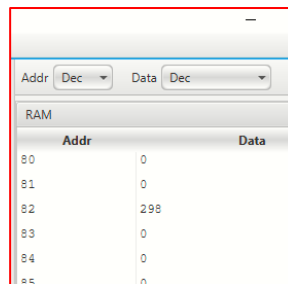
**Writing the Code:**

A screenshot of an assembly code editor window titled 'addln.a X'. The code is as follows:

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

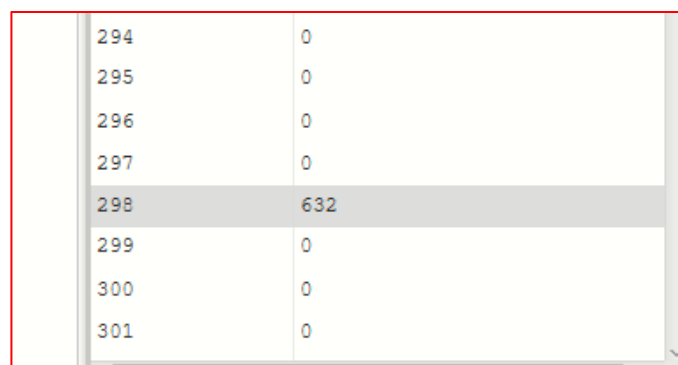
**Understanding the code:**

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.



Addr	Data
80	0
81	0
82	298
83	0
84	0
85	0

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



294	0
295	0
296	0
297	0
298	632
299	0
300	0
301	0

And effective address contain operand 632.

**Output:  $56+632= 688$ .**



Name	Width	Data
AC	16	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)**

```

q15.a X
1 JUMP 1 INDIRECT
2 LDA 0 A
3 ADD 0 B
4 HLT
5 INDIRECT: LDA 1 A
6           ADD 1 B
7           HLT
8 A: .data 1[16]
9 B: .data 1[10]
10

```

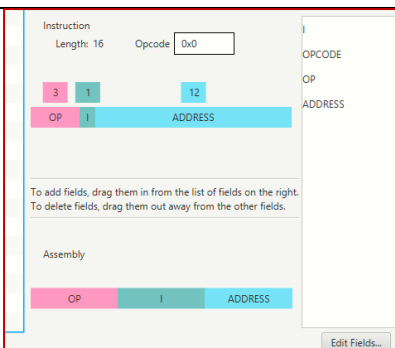
**Code For I=1**

```

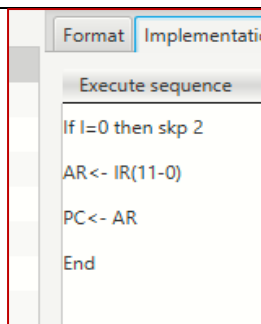
q15.a X
1 JUMP 0 INDIRECT
2 LDA 0 A
3 ADD 0 B
4 HLT
5 INDIRECT: LDA 1 A
6           ADD 1 B
7           HLT
8 A: .data 1[16]
9 B: .data 1[10]
10

```

**For I=0**



**Format of Jump.**



**Implementation of Jump.**

Name	Width
AC	16

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

Name	Width
AC	16

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 1<sup>st</sup> 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.