

CSA Lab Assignment-2

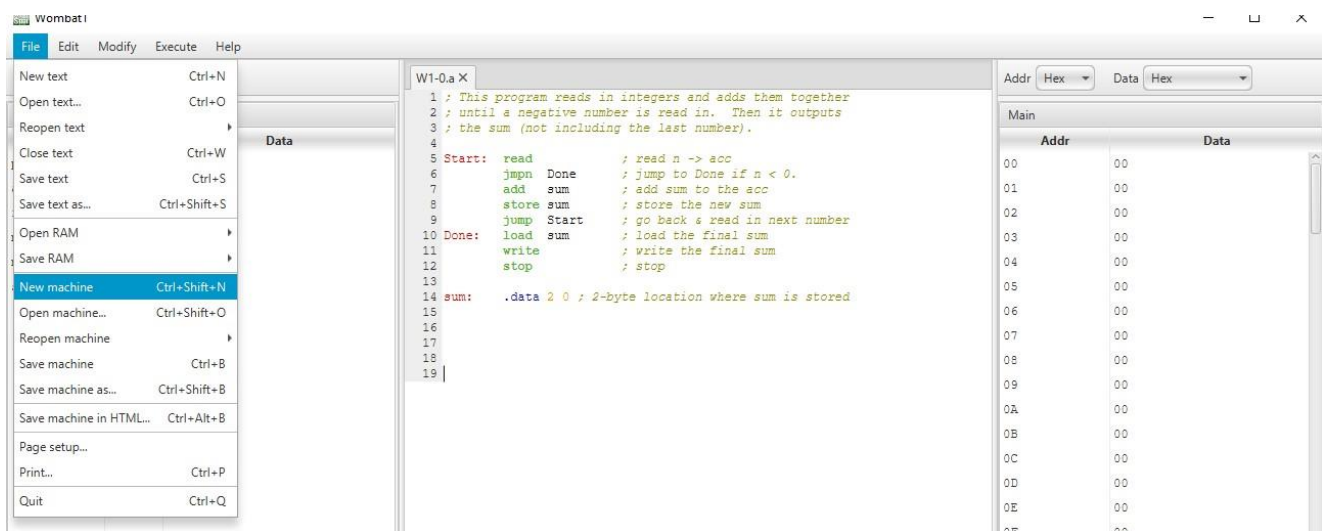
Harsh Bamotra AC-1216

Task- Adding two numbers

Step 1 : Creating a new machine and saving it.

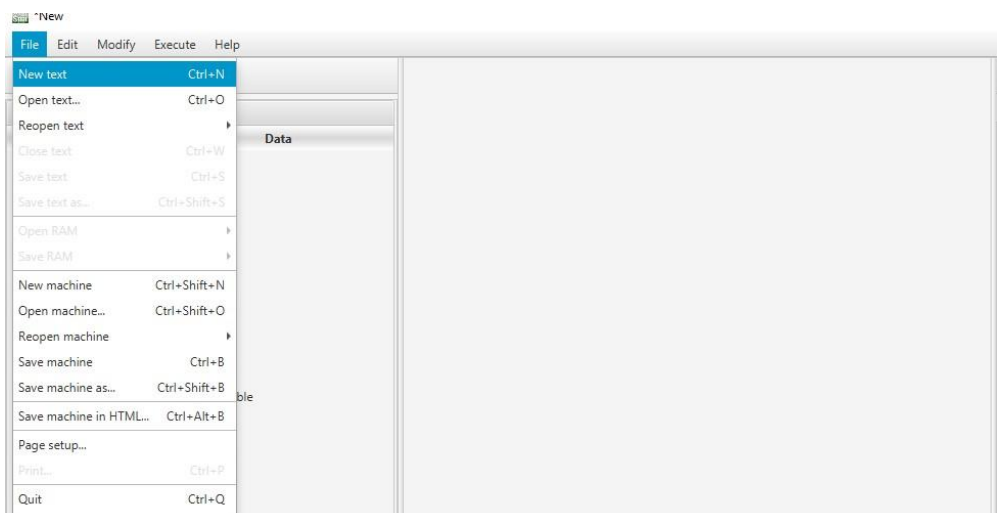
Click on the file option you will find it in the top of the application and then click on “New Machine” .

Then after creating a new machine save it with “.cpu” extension by clicking on the save file from the file.



Step 2: Creating text file to write commands

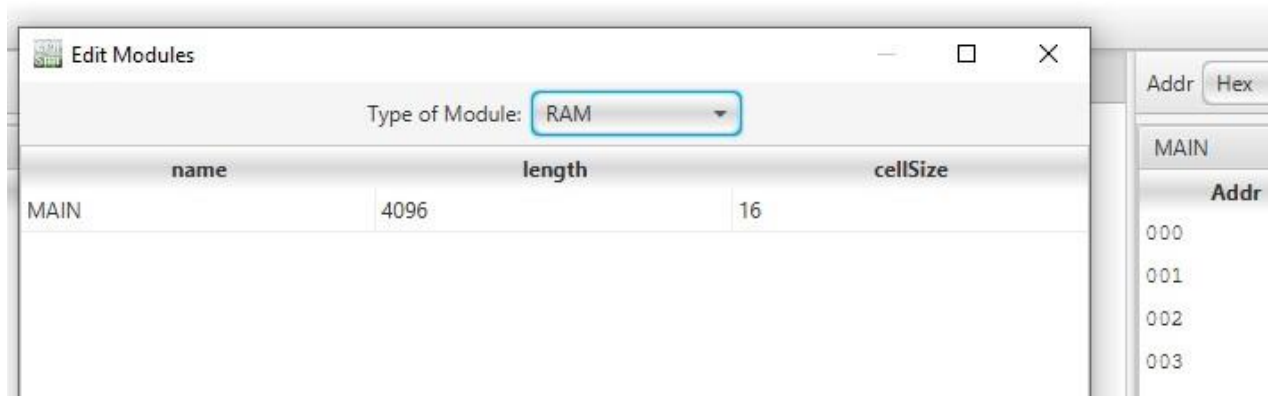
Click on the new text option from the file menu and create a new text and save it with “.a” extension .



Step 3: Creating Memory

To create memory go to **MODIFY → HARDWARE MODULES**

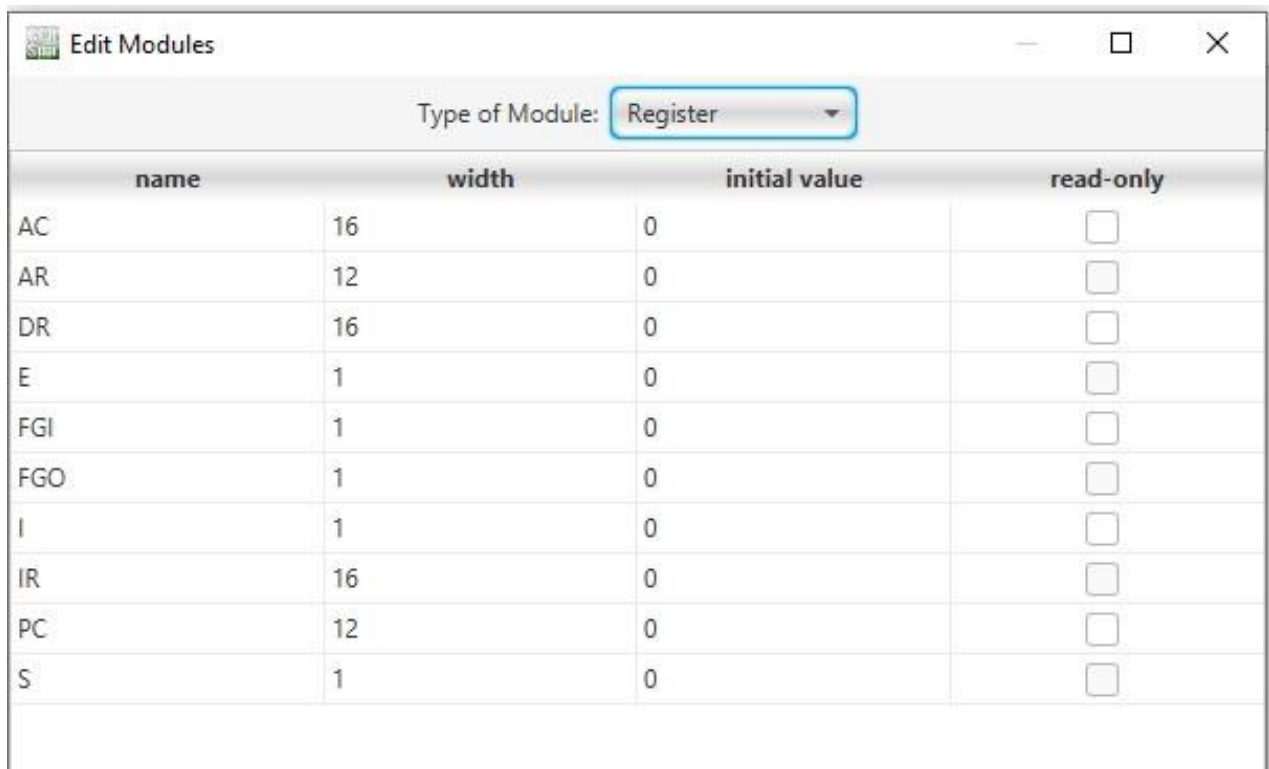
Now create MAIN memory in the RAM module.



Step 4: Creating Registers

To create registers go to **MODIFY → HARDWARE MODULES → REGISTER MODULE**

Now create the following Registers as shown in the picture below.



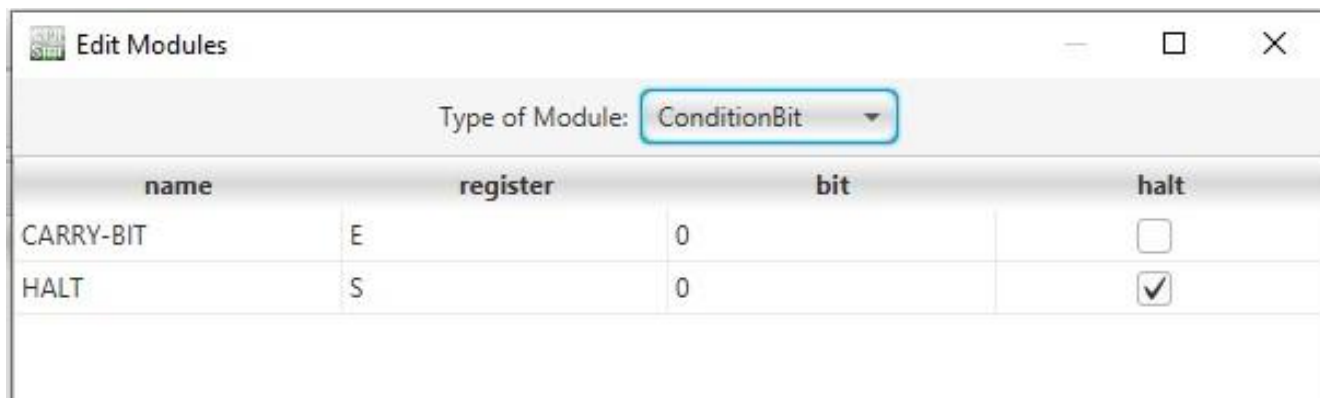
Functions of different registers:

- PC (Program counter)- It holds memory of the instructions.
- DR (Data register) -It holds memory operands .
- AR (Address register)-It holds address of the memory.
- AC (Accumulator)- It is the processor register
- IR (Instruction register)-It stores the instruction holds.
- S(Start stop flip flop)- It helps in starting and halting of the machine.
- FGI (Flag bit)- It is a control flip-flop and helps during the input.
- FGO(Flag bit)- It is also a control flip-flop and it helps during the output.
- I (Mod bit)- Specifies the addressing .
- E (Carry bit)- It holds the carry during the addition or subtractions.

Step 5: Creating Condition Bit

To create condition bit go to **MODIFY→ HARDWARE MODULE**

Now you have create the following Condition Bit in the Condition Bit module as shown in the pictures.



The screenshot shows a window titled 'Edit Modules' with a 'Type of Module:' dropdown set to 'ConditionBit'. Below this is a table with four columns: 'name', 'register', 'bit', and 'halt'.

name	register	bit	halt
CARRY-BIT	E	0	<input type="checkbox"/>
HALT	S	0	<input checked="" type="checkbox"/>

Function of Condition-Bit

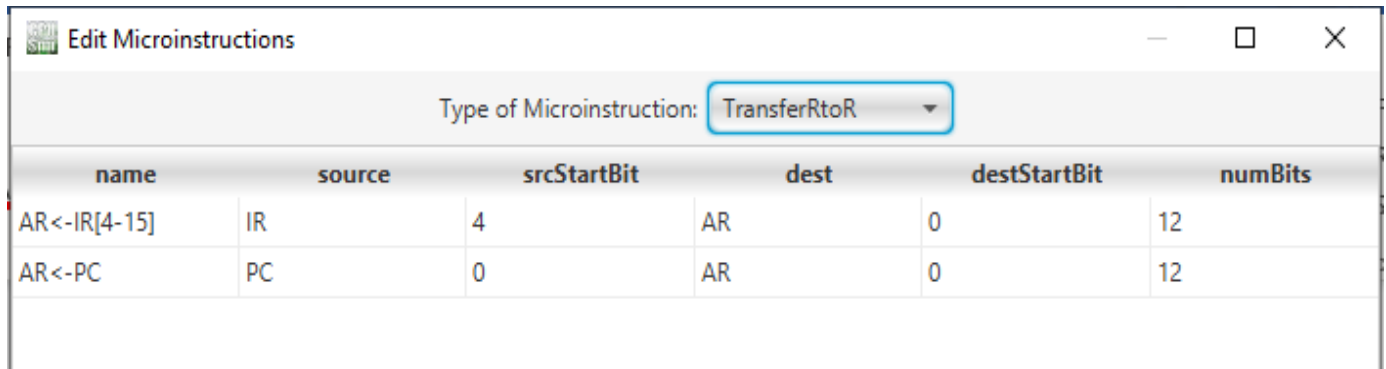
- CARRY-BIT- It stores the carry we get during the addition of the numbers.
- HLT- It halts the computer after the execution of the machine.

Step 6: Creating Microinstructions

Now to create Microinstructions go to **MODIFY→ MICROINSTRUCTIONS**

You have to create the following Microinstructions on your machine.

1. TransferRtoR



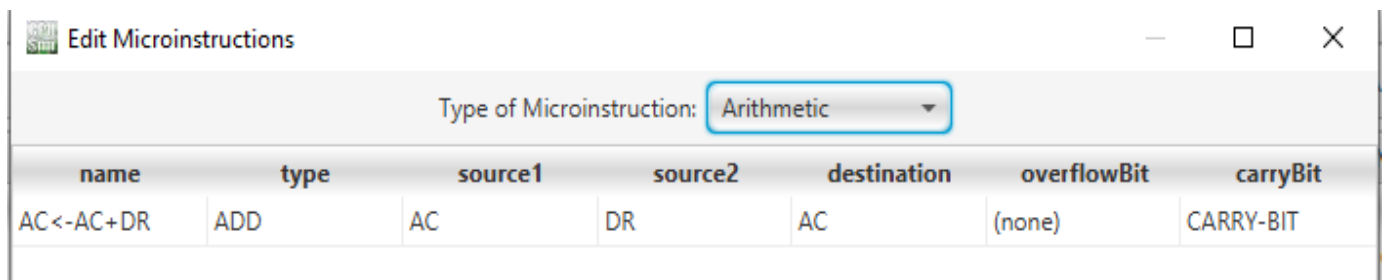
The screenshot shows a window titled 'Edit Microinstructions'. At the top, there is a dropdown menu labeled 'Type of Microinstruction:' with 'TransferRtoR' selected. Below this is a table with the following columns: name, source, srcStartBit, dest, destStartBit, and numBits. The table contains two rows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

name	source	srcStartBit	dest	destStartBit	numBits
AR<-IR[4-15]	IR	4	AR	0	12
AR<-PC	PC	0	AR	0	12

Functions of the instructions:

- **AR<-IR[4-15]**-It will transfer the data from IR to AR.
- **AR<-PC**- It will transfer the data from PC to AR.

2. Arithmetic



The screenshot shows a window titled 'Edit Microinstructions'. At the top, there is a dropdown menu labeled 'Type of Microinstruction:' with 'Arithmetic' selected. Below this is a table with the following columns: name, type, source1, source2, destination, overflowBit, and carryBit. The table contains one row of data.

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

Functions of the instructions:

- **AC<-AC+DR**- It will take the data from the DR and AC and then add them and finally store the result in AC.

3.Set

Edit Microinstructions				
Type of Microinstruction: Set				
name	register	start	numBits	value
FGI<-0	FGI	0	1	0
FGO<-0	FGO	0	1	0

Functions of the instructions:

- **FGI<-0** – It will flag that the input is ready .
- **FGO<-0** – It will flag that the output is ready for printing.

4.MemoryAccess

Edit Microinstructions				
Type of Microinstruction: MemoryAccess				
name	direction	memory	data	address
DR<-MAIN[AR]	read	MAIN	DR	AR
IR<-MAIN[AR]	read	MAIN	IR	AR
MAIN[AR]<-AC	write	MAIN	AC	AR

Functions of the instructions:

- **DR<-MAIN[AR]**- It will transfer the data from MAIN[AR] to DR .
- **IR<-MAIN[AR]** -It will transfer the data from MAIN[AR] to IR.
- **MAIN[AR]<-AC** - It will transfer the date from AC to MAIN[AR].

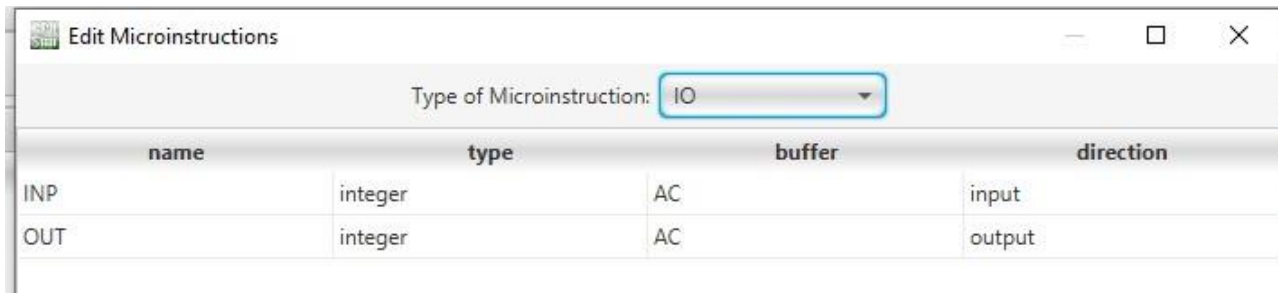
5. Increment

Edit Microinstructions				
Type of Microinstruction: Increment				
name	register	overflowBit	carryBit	delta
INCR-PC	PC	(none)	(none)	1

Functions of the instructions:

- INCR-PC – It will increment the PC .

6. IO



The screenshot shows a window titled "Edit Microinstructions". At the top, there is a dropdown menu labeled "Type of Microinstruction:" with "IO" selected. Below this is a table with four columns: "name", "type", "buffer", and "direction".

name	type	buffer	direction
INP	integer	AC	input
OUT	integer	AC	output

Functions of the instructions:

- INT- It will take the input from the user in the form of integer.
- OUT- It will take the output after addition of the numbers.

7. Decode



The screenshot shows a window titled "Edit Microinstructions". At the top, there is a dropdown menu labeled "Type of Microinstruction:" with "Decode" selected. Below this is a table with two columns: "name" and "ir".

name	ir
DECODE-IR	IR

Functions of the instructions:

- DECODE-IR – It will decode the IR.

8.SetCondBit



The screenshot shows a window titled "Edit Microinstructions". At the top, there is a dropdown menu labeled "Type of Microinstruction:" with "SetCondBit" selected. Below this is a table with three columns: "name", "bit", and "value".

name	bit	value
HLT-BIT	HALT	1

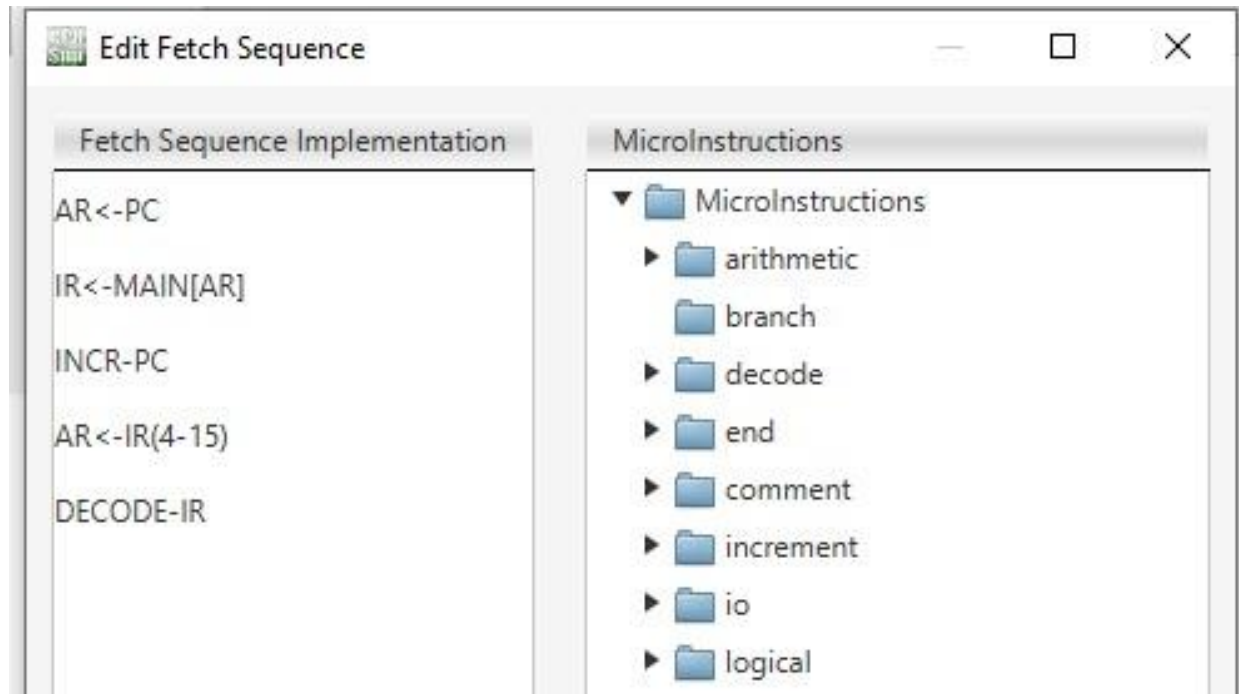
Functions of the instructions:

- HLT-BIT – It will set the HALT-BIT and will give it the value .

Step 7: Implementing Fetch Sequence

In order to implement the fetch sequence for our machine go to **MODIFY→FETCH SEQUENCE**.

Now create the following fetch sequence as shown in the figure.



Step 8: Creating Instructions

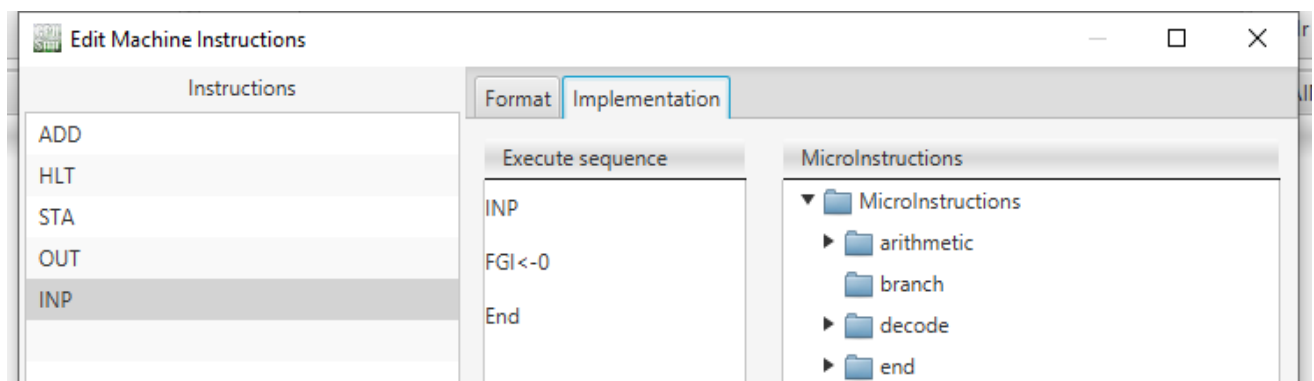
In order to create set instruction just go to **MODIFY→ MACHINE INSTUCTIONS** .

Now create the following instructions:-

1. INP (Register reference instruction)

Opcode-0xF800

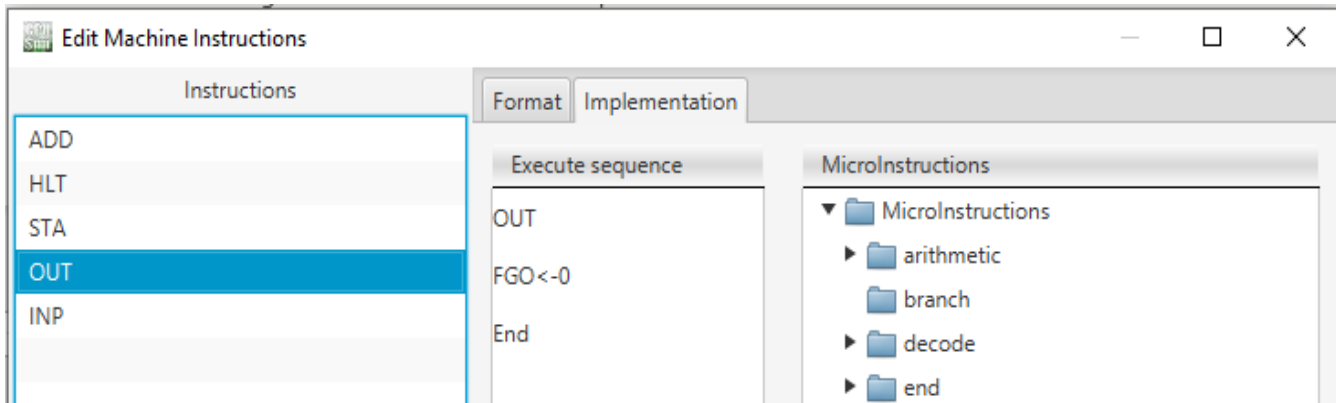
Field-REGISTER



2. OUT (Register reference instruction)

Opcode-0xF400

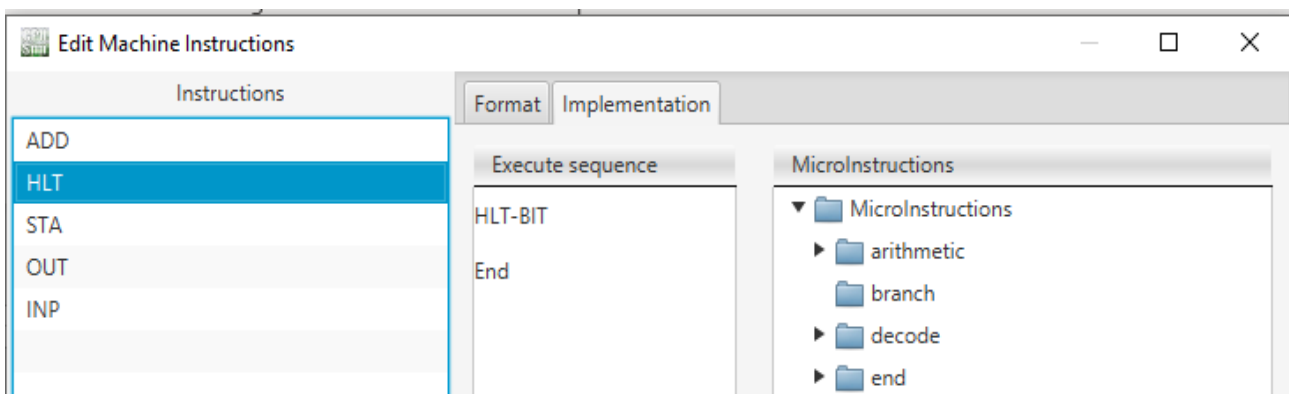
Field-REGISTER



3. HLT (Register reference instruction)

Opcode-0xE001

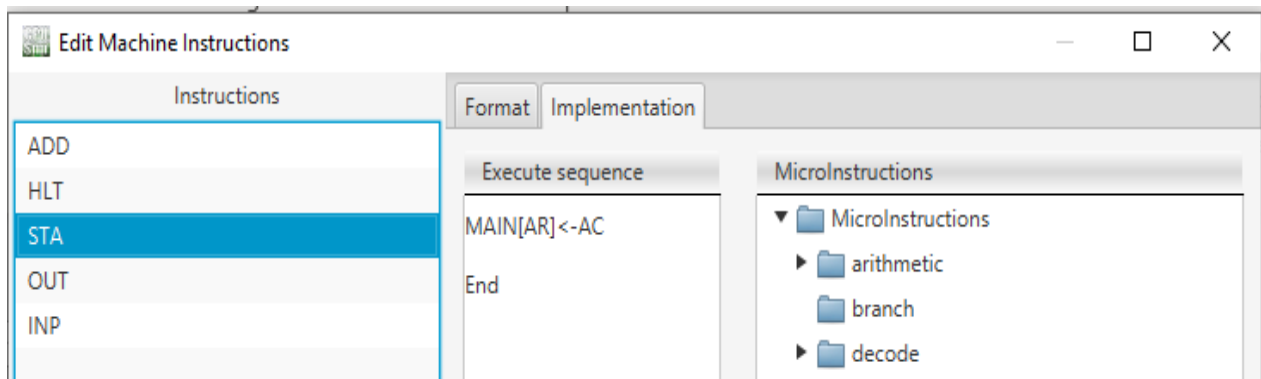
Field-REGISTER



4. STA (Memory reference instruction)

Opcode-0x6

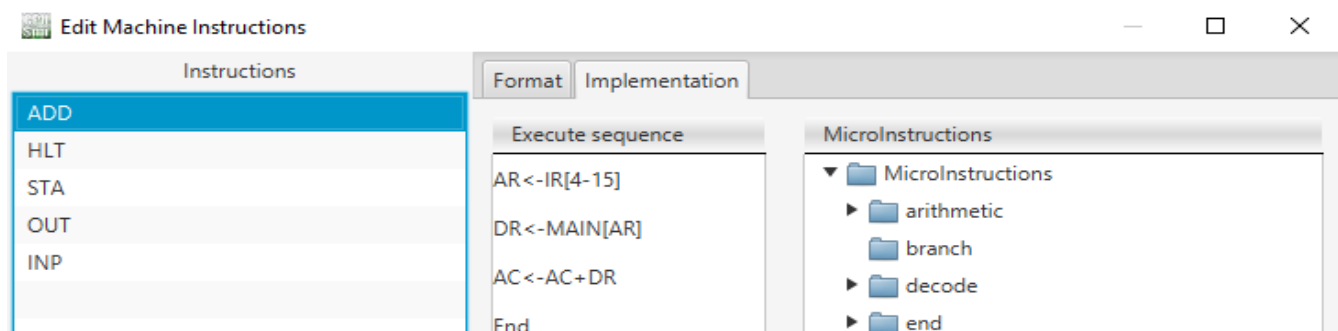
Field-OP , ADDR



5. ADD (Memory reference instruction)

Opcode-0x7

Field-OP , ADDR



Step 9: Writing Commands

Now just type the following commands in the text box we saved earlier as add.a as shown in the picture given below.

```
sum.a X
1 START: INP
2 STA NUM
3 INP
4 ADD NUM
5 OUT
6 HLT
7
8 NUM: .data 1 0
```

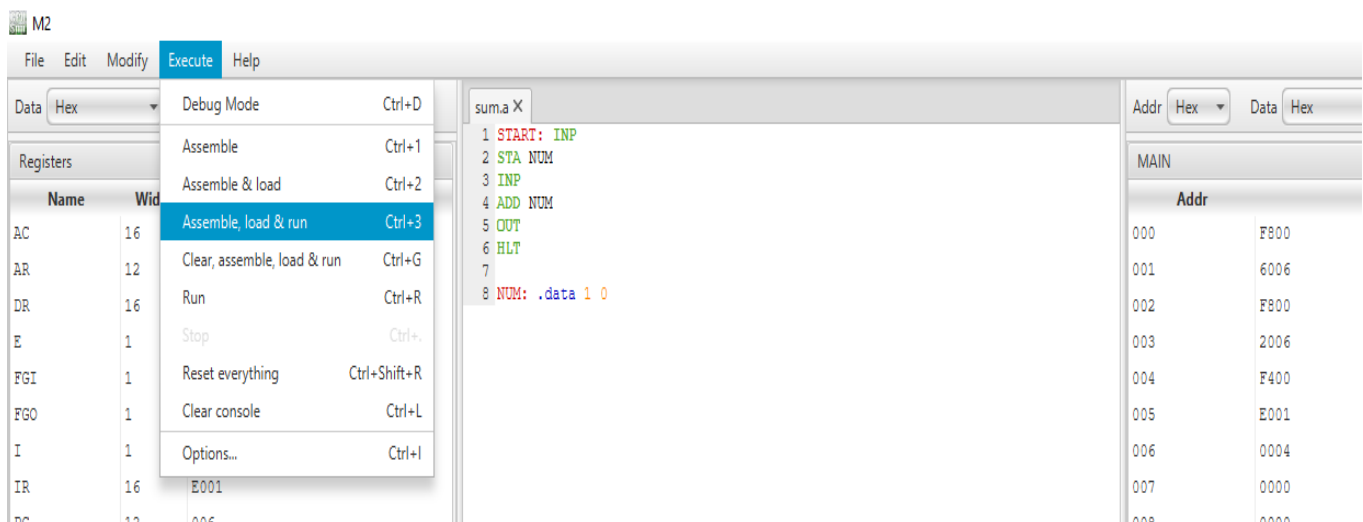
Explanation:-

Now as you can see in the above picture the commands we have written in the text box , so the **INP** commands are basically telling the machine that first you have to take input from the user and then the **ADD** command is used to add the integers we got from the user and **OUT** will store it as the output . After all this the **HLT** command is telling the machine to halt.

Step 10: Running the commands to perform the task

In order to run the commands go to **EXECUTE** and click on **RESET EVERYTHING**.

Now just select the **ASSEMBLE , LOAD , RUN** from the **EXECUTE** menu.



Now as soon you will click it you will see that the lower part of the window becomes yellow and now you just have to input the integers which you want to add .

```
EXECUTING...
```

```
Enter Inputs, the first of which must be an Integer: 50
```

```
Enter Inputs, the first of which must be an Integer: 100
```

Now as soon you will give the input and press enter you will see that the result is printed as shown in the picture below.

```
EXECUTING...
```

```
Enter Inputs, the first of which must be an Integer: 50
```

```
Enter Inputs, the first of which must be an Integer: 100
```

```
Output: 150
```

```
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HLT]
```