# ATMA RAM SANATAN DHARMA COLLEGE

(UNIVERSITY OF DELHI)

## COMPUTER SYSTEM ARCHITECTURE

(PRACTICAL FILE)

***This practical report on CPU Sim was created under the valuable guidance and support of our esteemed professor*.**


**SUBMITTED BY**

NAME:**SUKET ROHILLA**

ROLL NO:**24/48069**


DATE:**27-12-2024**

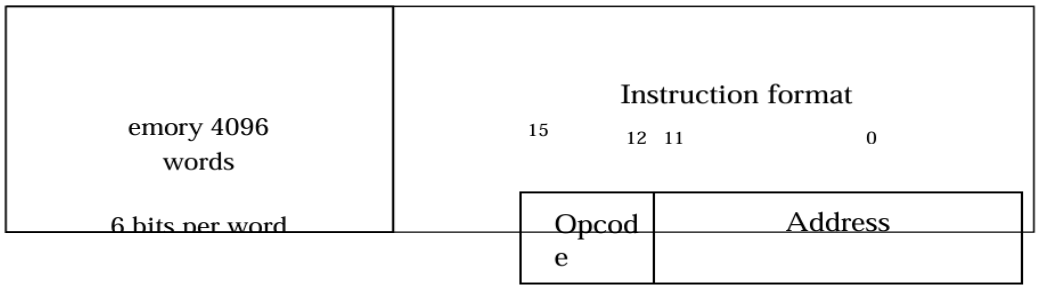
# Q1-Create a machine based on the following architecture

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

1. Create a machine based on the following architecture:

Registers

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




Basic Computer Instructions

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

**Design the register set, memory and the instruction set. Use this machine for the assignments of this section.**

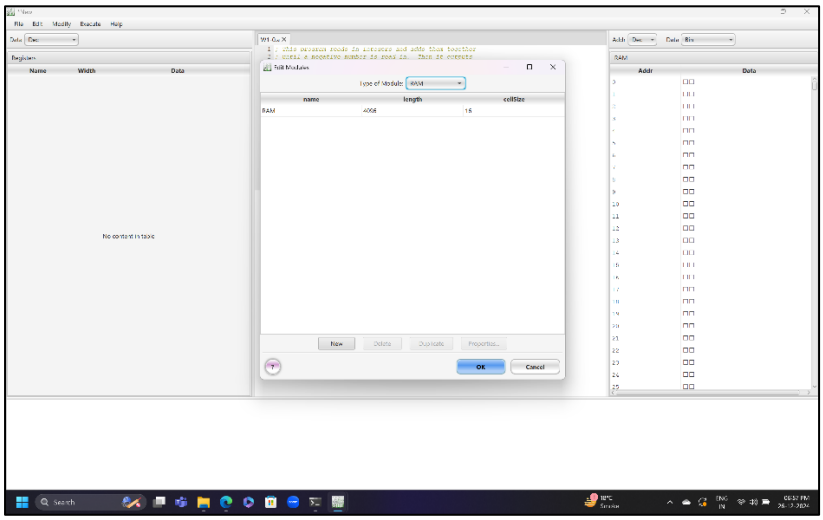
**ANSWER:**

- **CREATING RAM:**

**1. For creating a machine, First RAM should be created. For creating RAM, go to Modify and click Hardware Modules.**

**2. In hardware modules, click RAM from Type of module.**

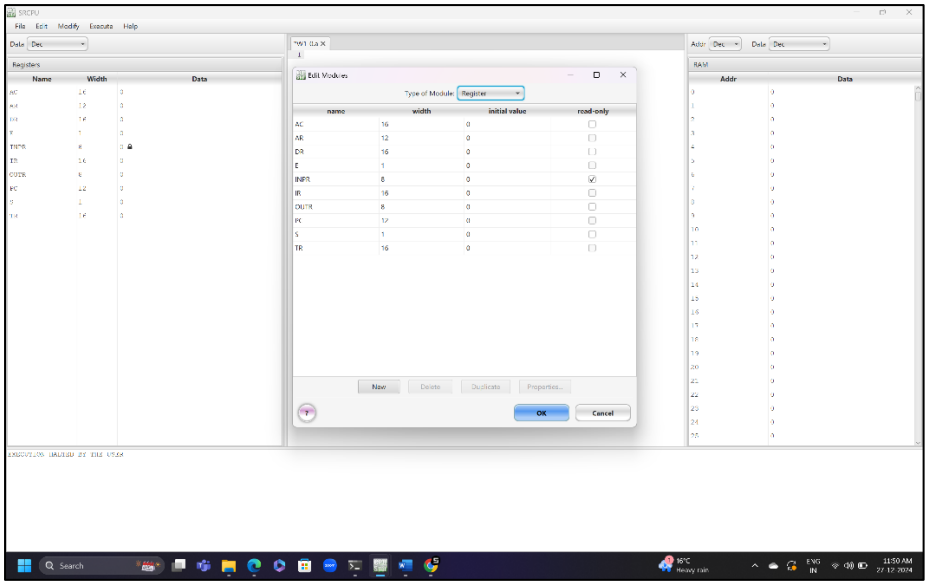
**3. Fill name, length and cell size of RAM.**



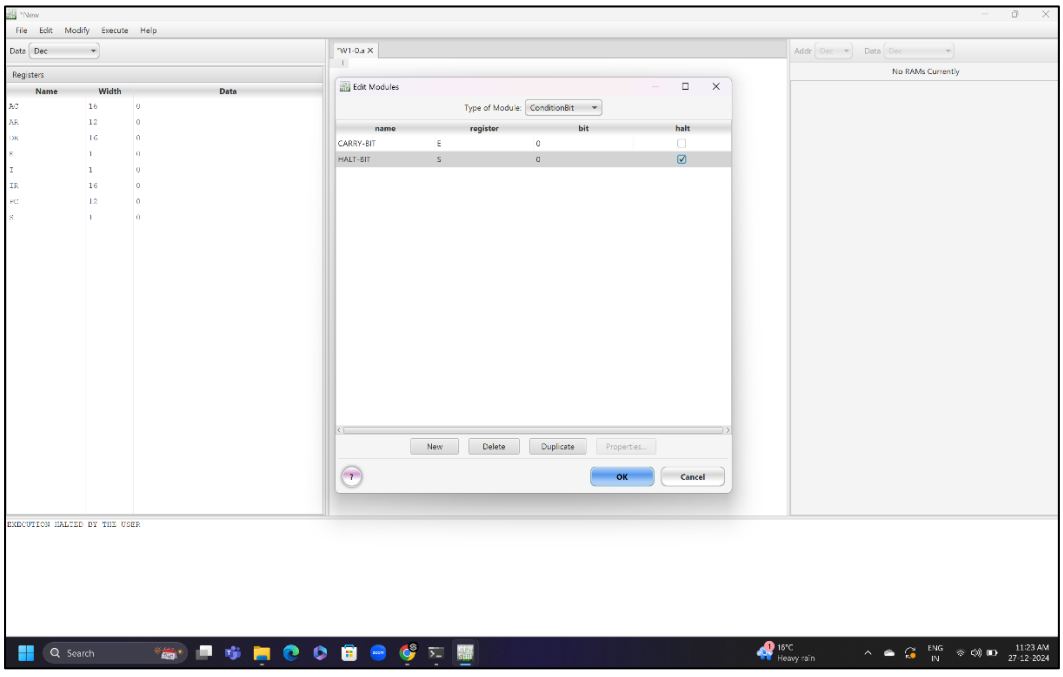
**•CREATING REGISTERS:**

**1. In hardware modules, select Registers as a type of module**

**2. Create various types of registers by writing name, width, initial value and apply a check mark if you need read only option**

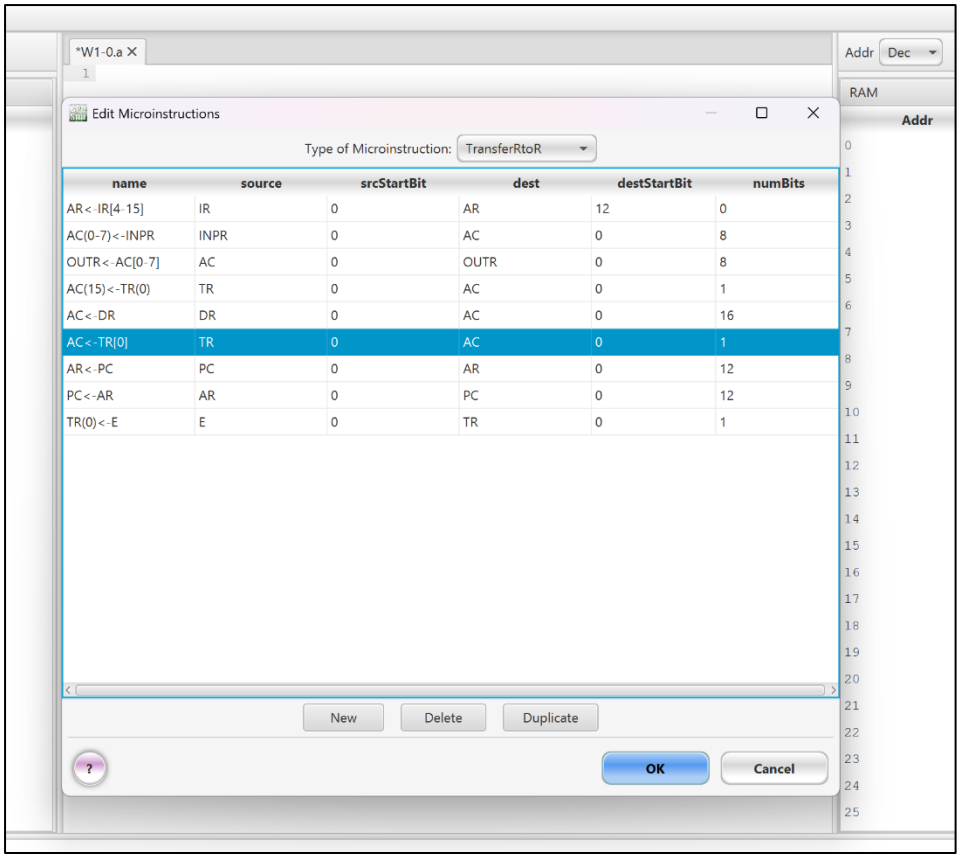


- **CREATING CONDITION BIT: Select Condition Bit from types of modules from the hardware modules and create carry out bits**

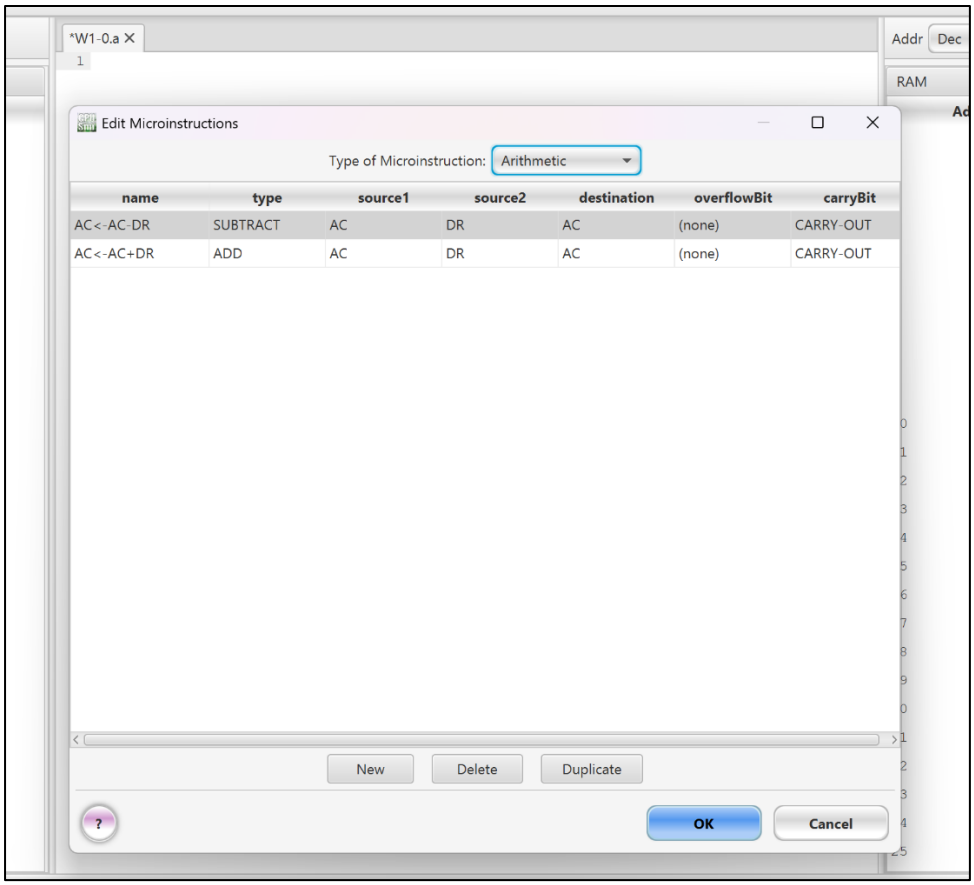



- **CREATING MICROINSTRUCTIONS: Go to Modify and select Microinstructions.**

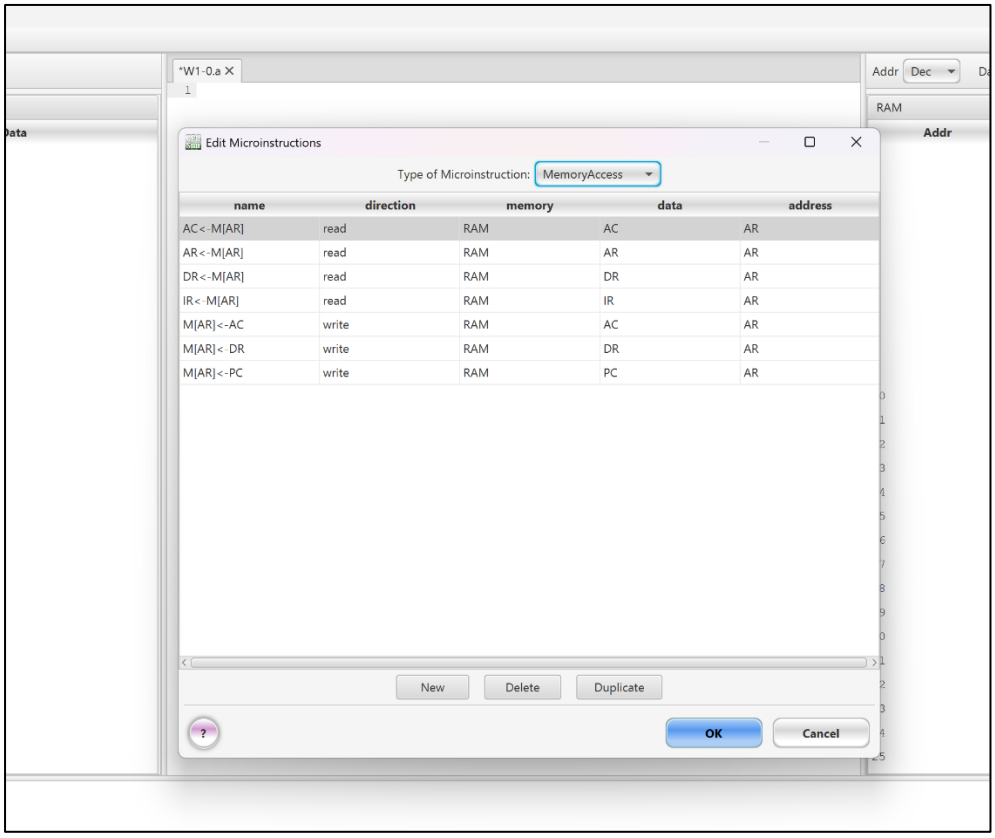
**1. TRANSFER R TO R**



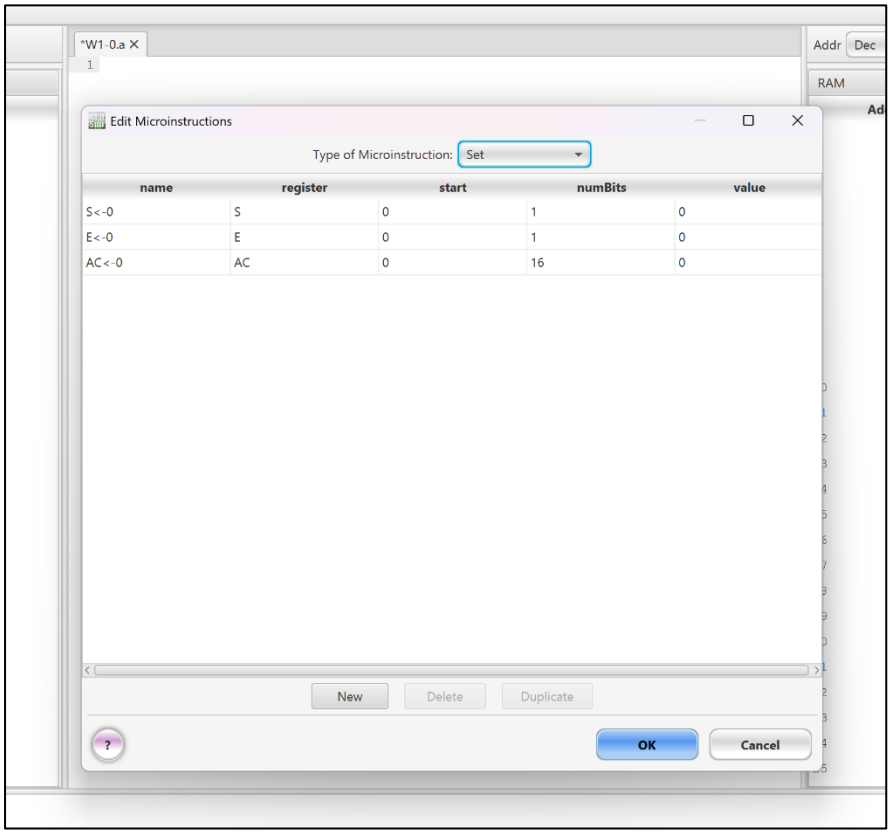

# 2. ARITHMETIC:



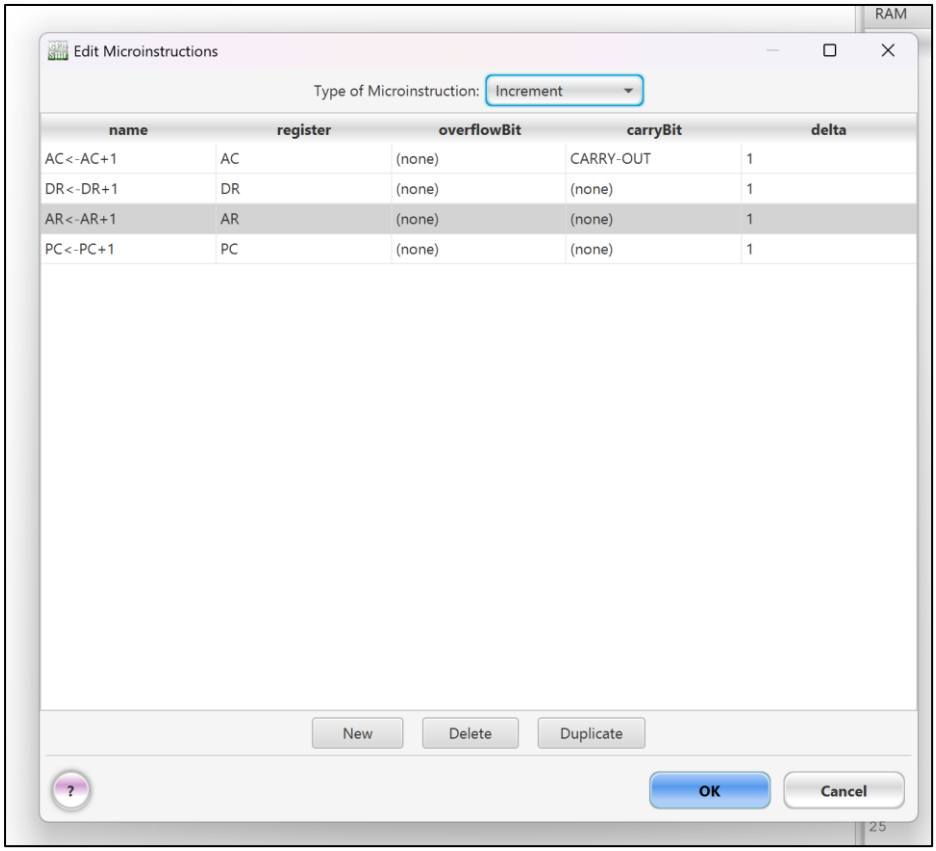
# 3. MEMORY ACCESS:



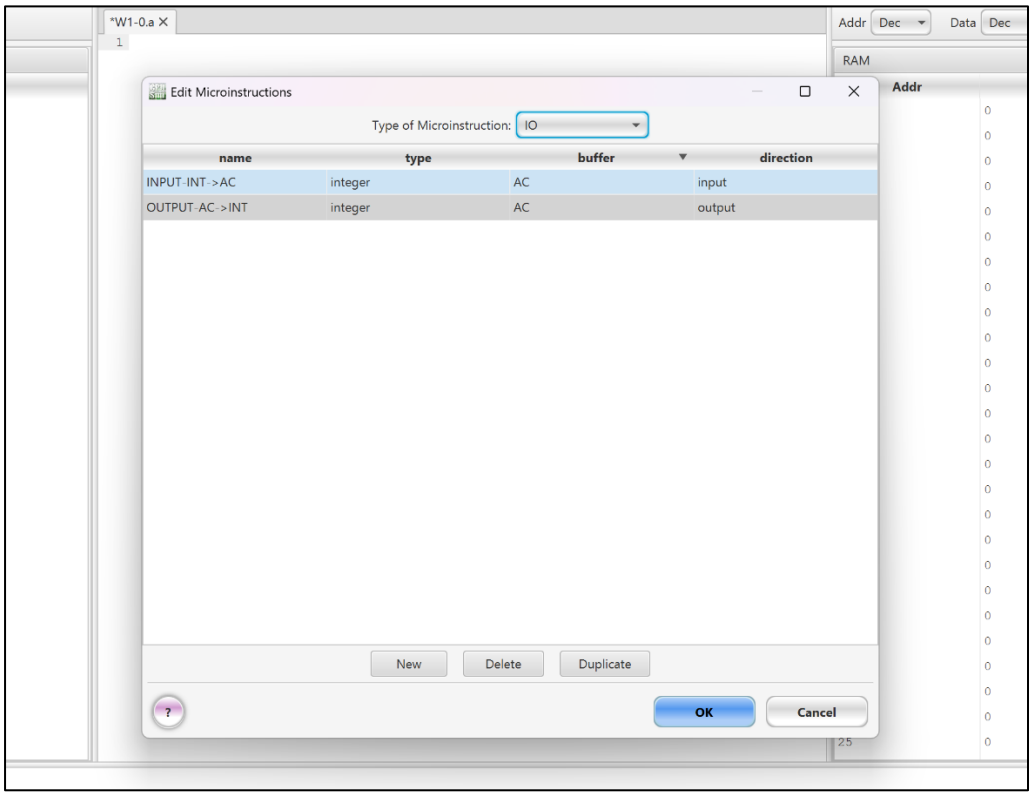

# 4. SET:



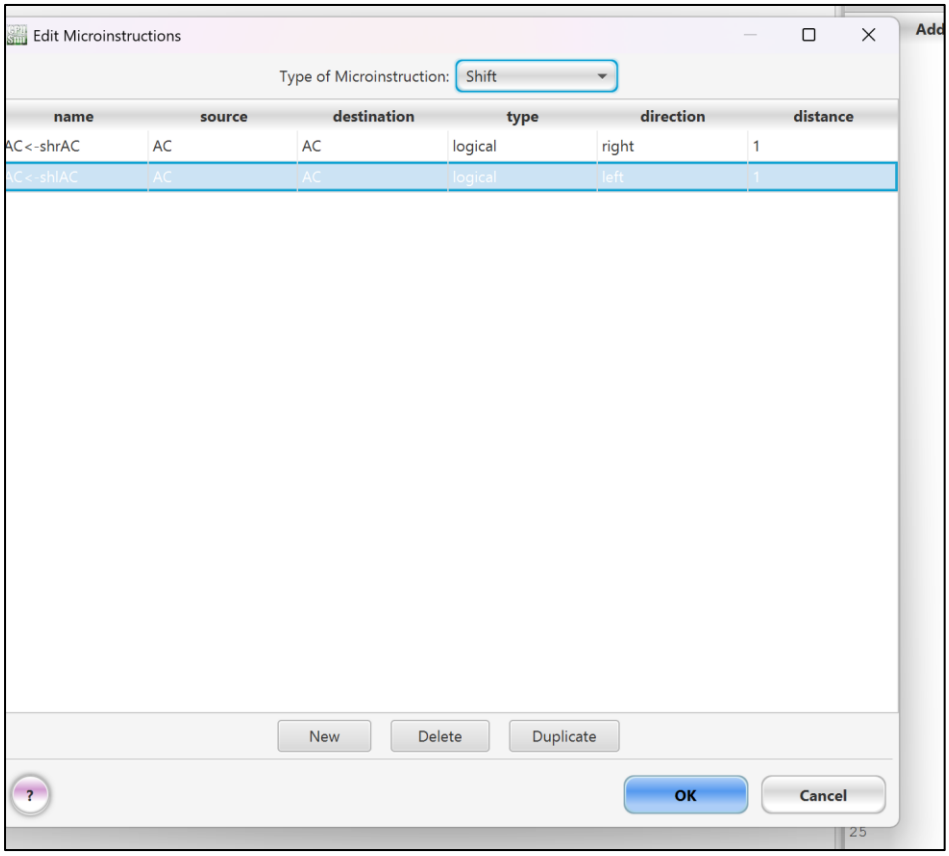
# 5. INCREMENT:



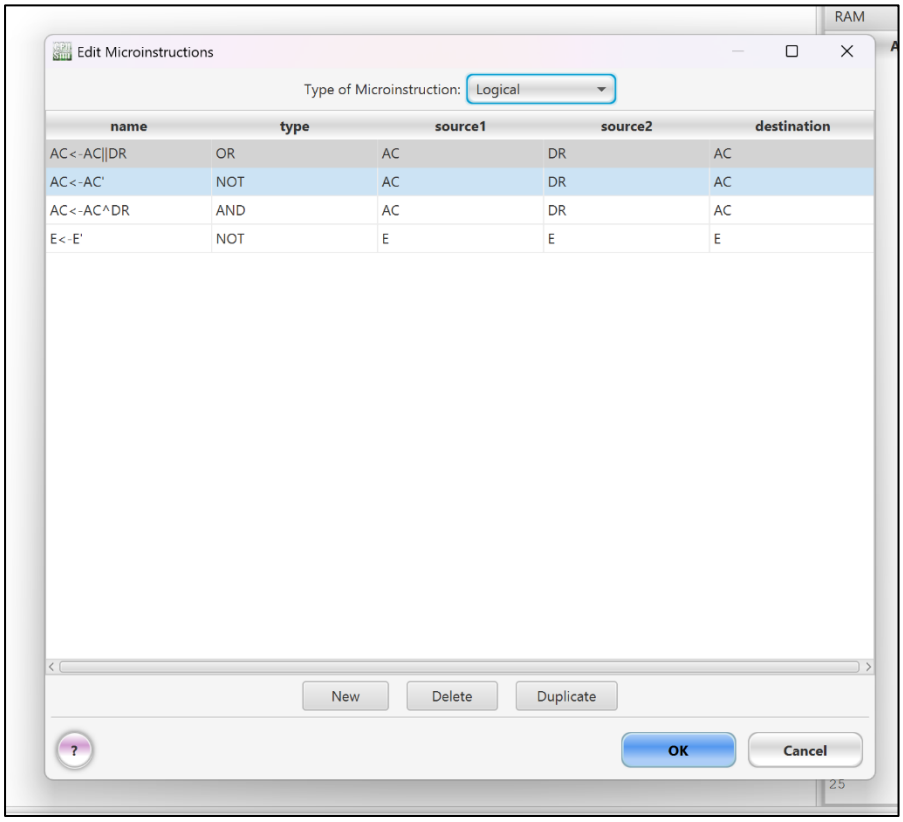

# 6. input-output:



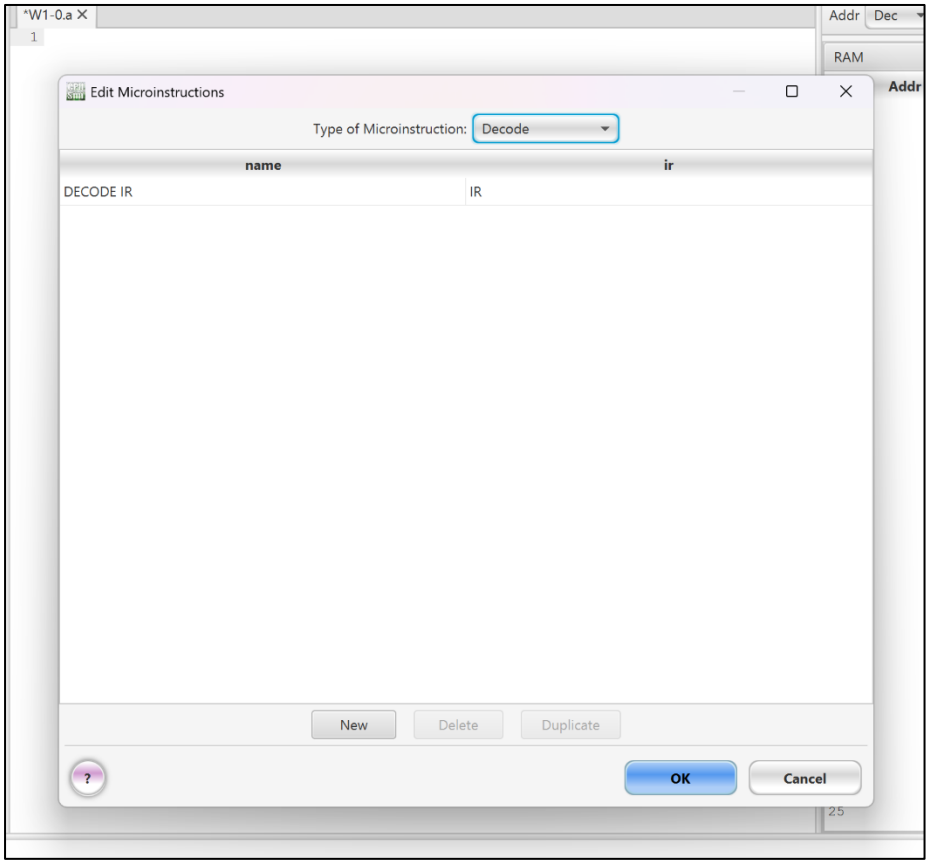
# 7. SHIFT:



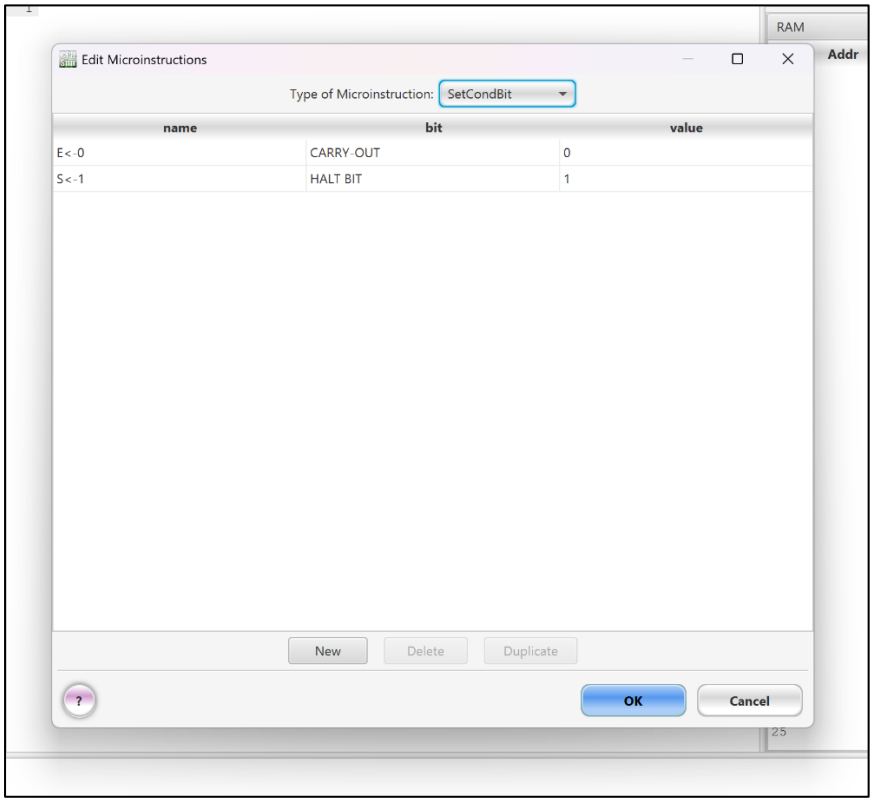

# 8. LOGICAL:



# 9. DECODE:




# 10. SET CONDITION BIT:

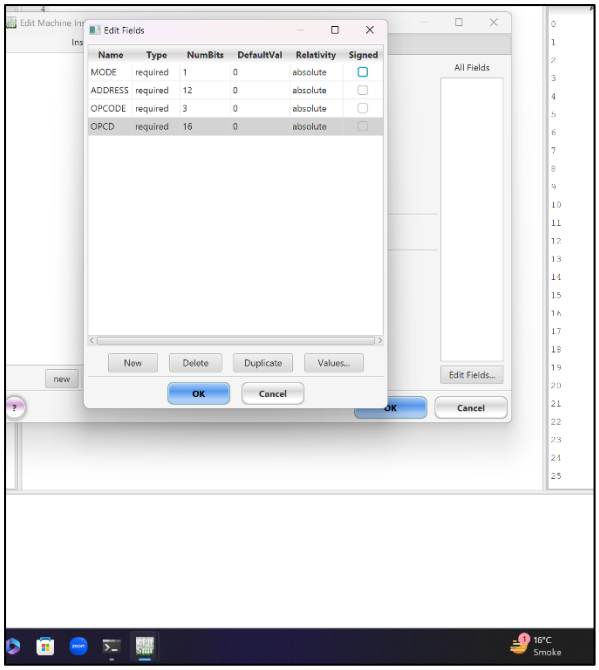


## CREATING MACHINE INSTRUCTIONS

**1.Go to modify and select Machine instructions.**

**2.Select Edit field from the right corner of The Machine Instructions.**

**3.Create all the fields of edit fields.**




## CREATING INSTRUCTIONS

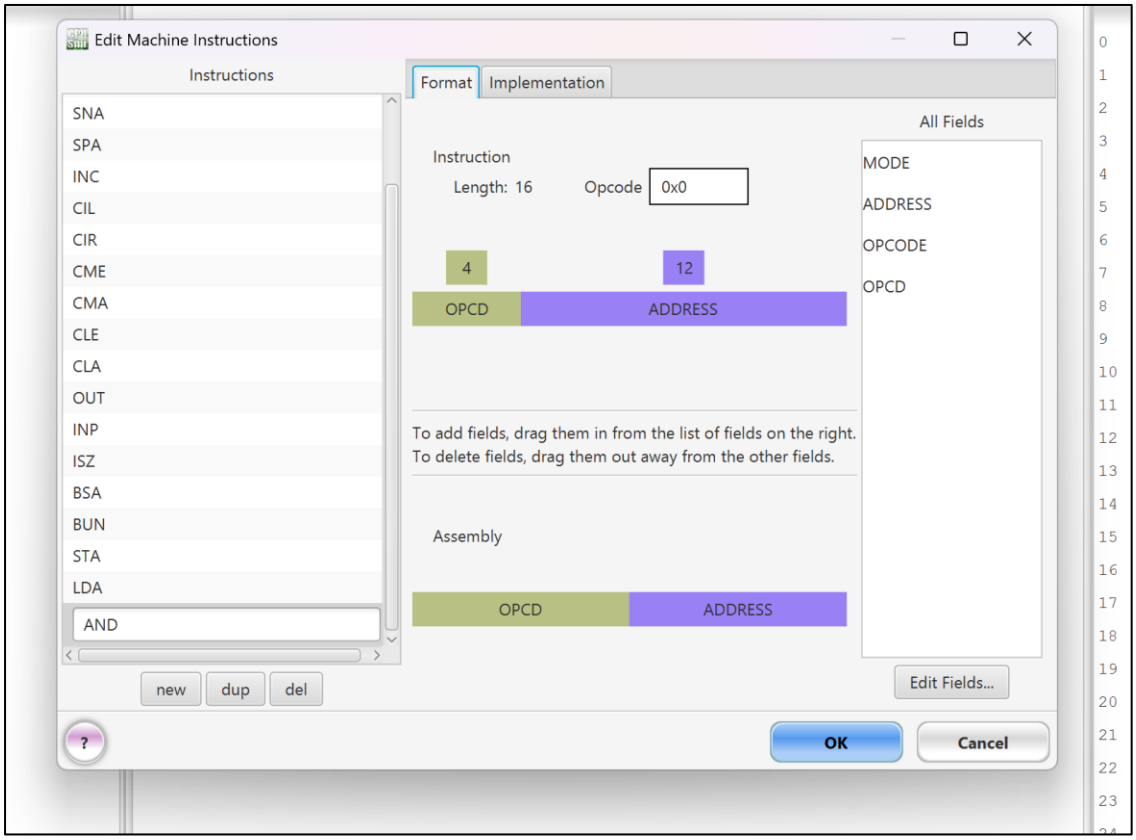
- **ADD:**

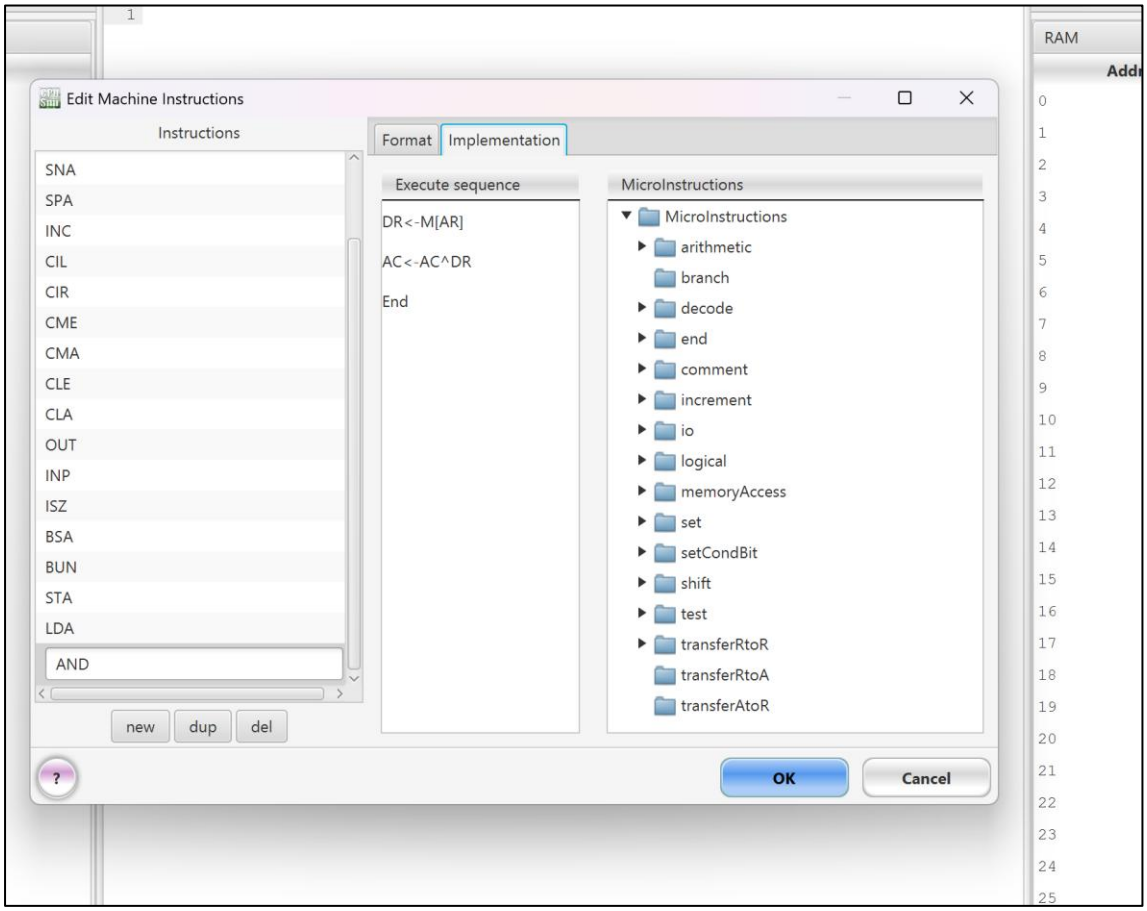
**1.Click on new and write the name of instruction**

**2.Give an opcode and drag required fields and put them under opcode.**

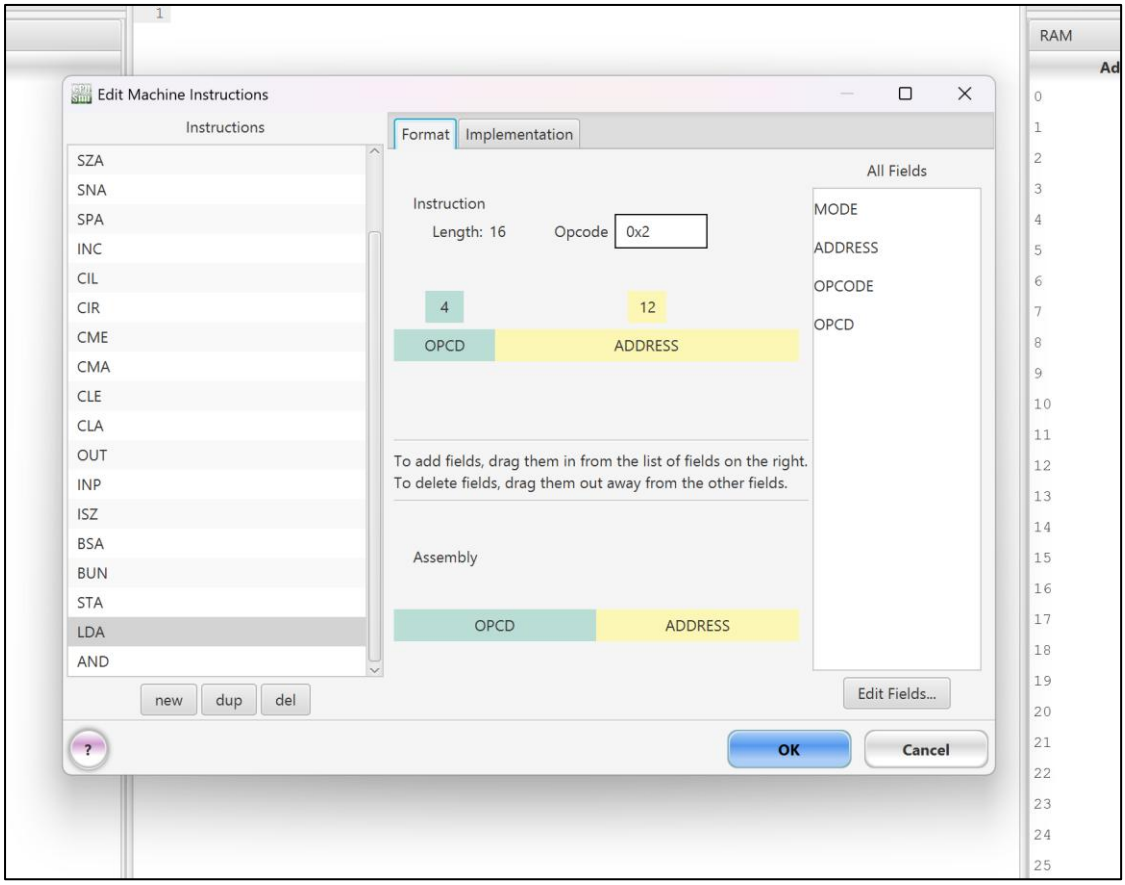
**3.Click on implementations and write required instructions by grabbing them from microinstructions.**

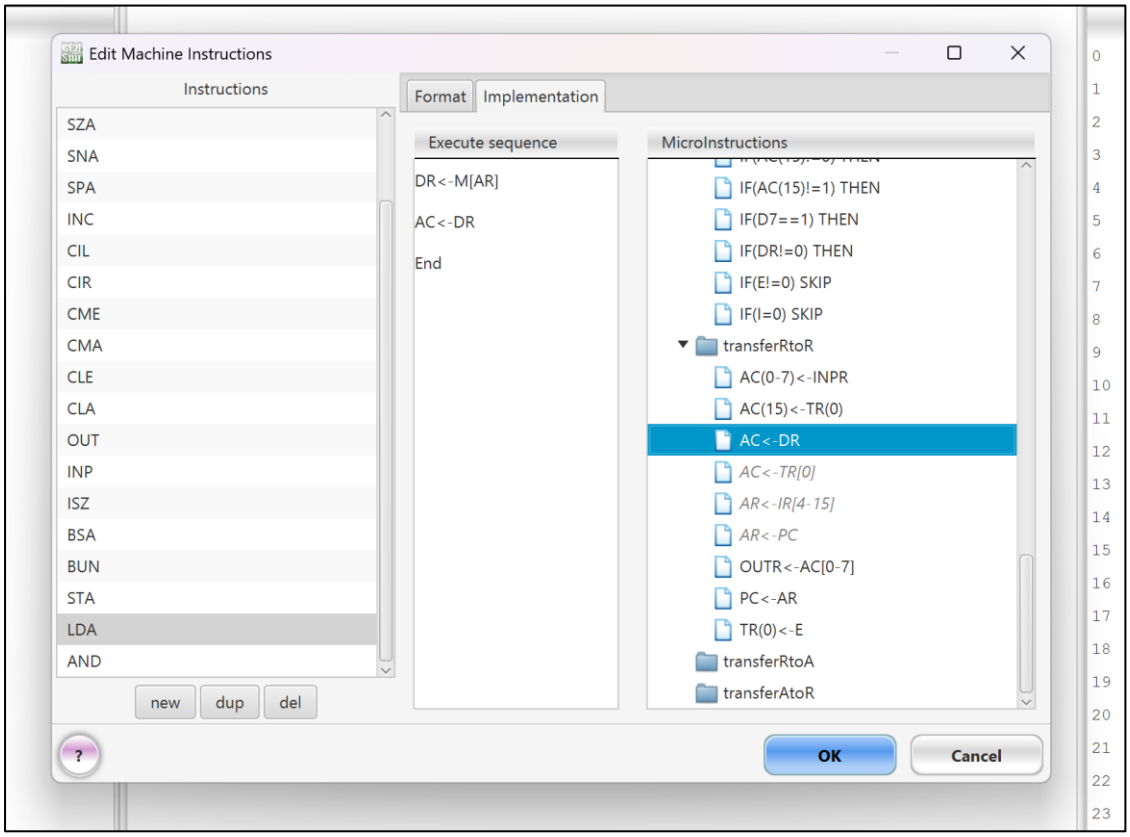
- **AND:**


Edit Machine Instructions
Instructions
SNA
SPA
INC
CIL
CIR
CME
CMA
CLE
CLA
OUT
INP
ISZ
BSA
BUN
STA
LDA
AND
new
dup
del
Format
Implementation
Instruction
Length: 16
Opcode
0x0
4
12
OPCD
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
OPCD
ADDRESS
All Fields
MODE
ADDRESS
OPCODE
OPCD
Edit Fields...
OK
Cancel

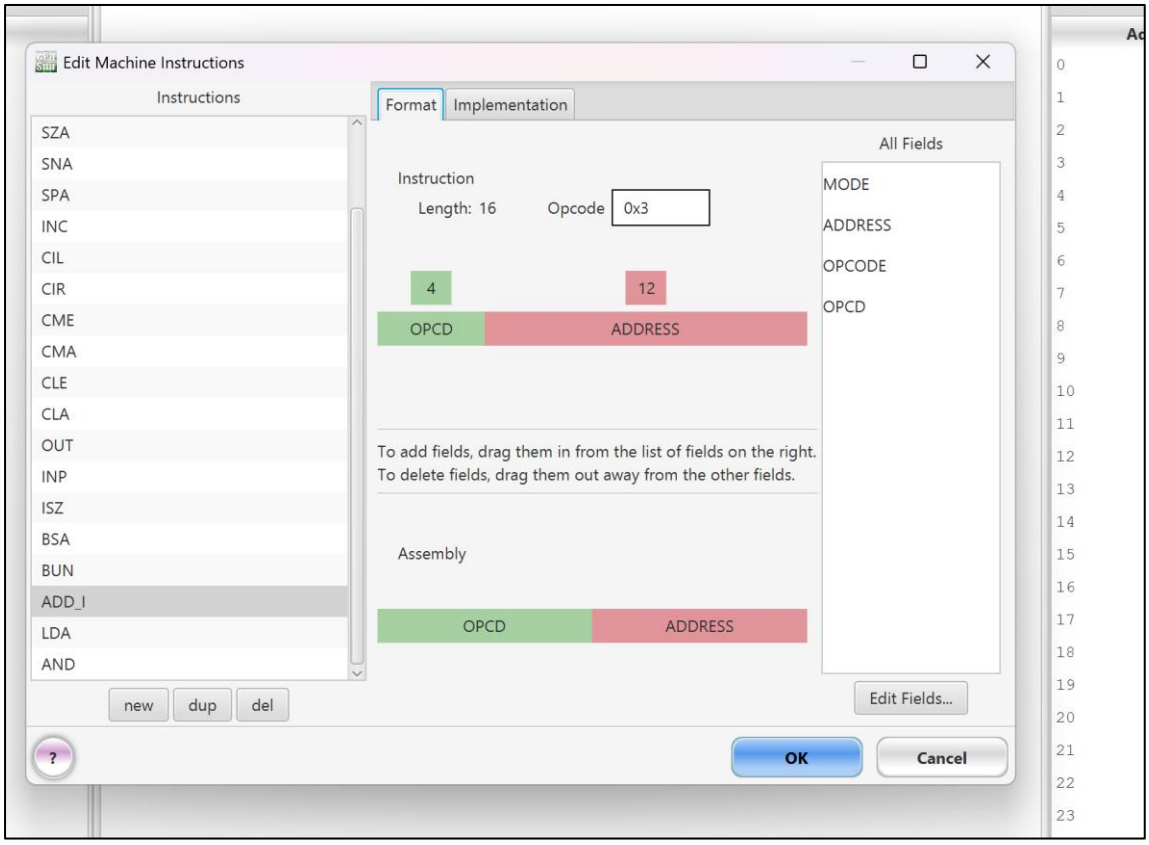

Edit Machine Instructions
Instructions
SNA
SPA
INC
CIL
CIR
CME
CMA
CLE
CLA
OUT
INP
ISZ
BSA
BUN
STA
LDA
AND
new
dup
del
Format
Implementation
Execute sequence
DR<-M[AR]
AC<-AC^DR
End
MicroInstructions
MicroInstructions
arithmetic
branch
decode
end
comment
increment
io
logical
memoryAccess
set
setCondBit
shift
test
transferRtoR
transferRtoA
transferAtoR
OK
Cancel
RAM

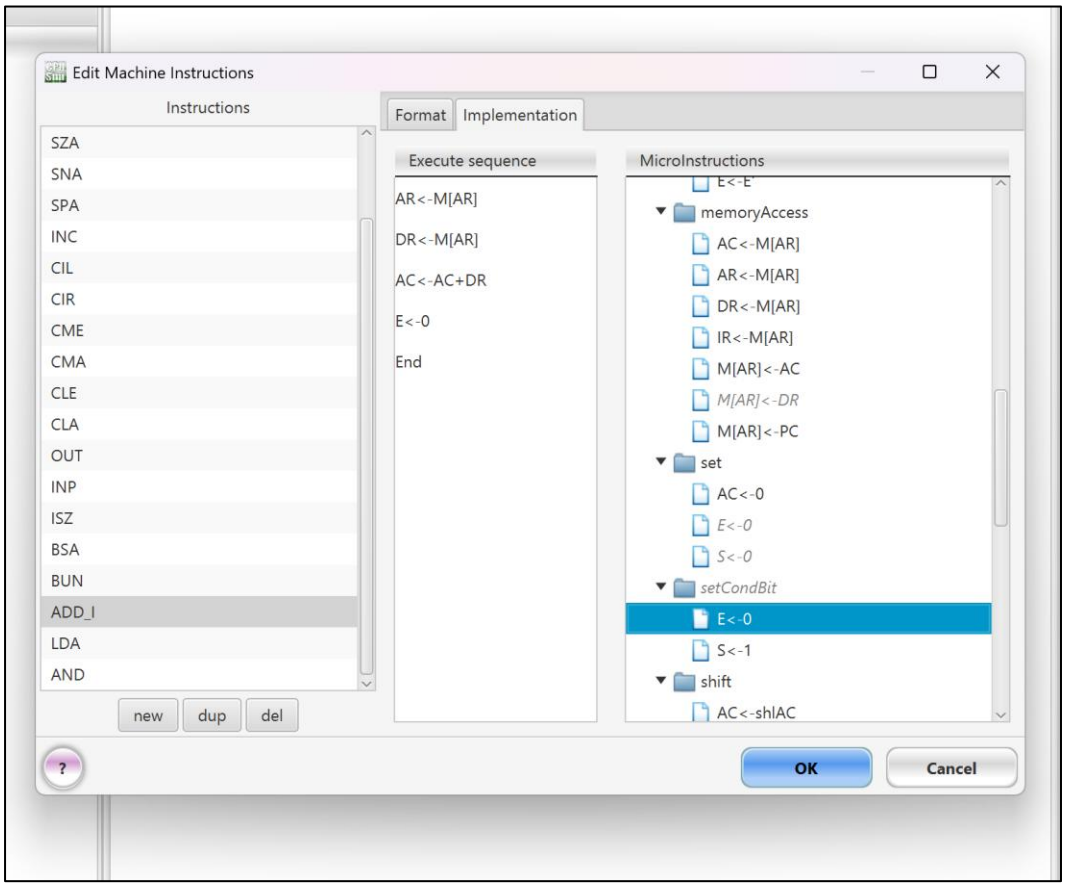
- **LDA:**


Edit Machine Instructions
Instructions
SZA
SNA
SPA
INC
CIL
CIR
CME
CMA
CLE
CLA
OUT
INP
ISZ
BSA
BUN
STA
LDA
AND
new
dup
del
Format
Implementation
Instruction
Length: 16
Opcode
0x2
4
12
OPCD
ADDRESS
All Fields
MODE
ADDRESS
OPCODE
OPCD
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
OPCD
ADDRESS
Edit Fields...
OK
Cancel
RAM

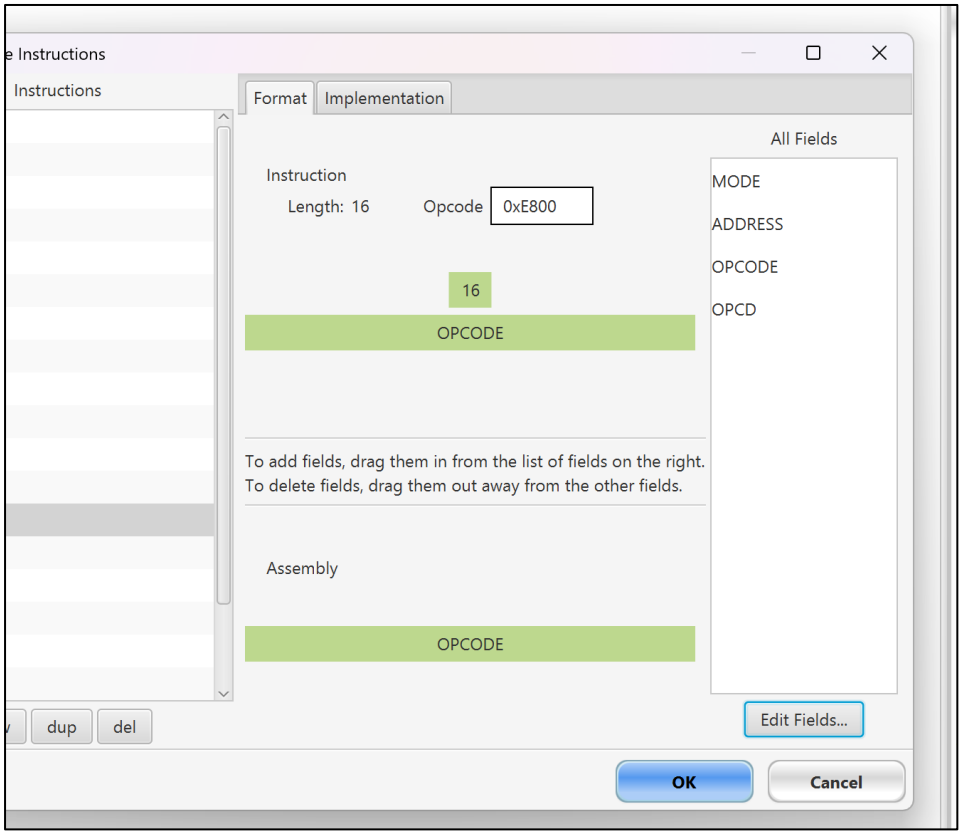

Edit Machine Instructions
Instructions
SZA
SNA
SPA
INC
CIL
CIR
CME
CMA
CLE
CLA
OUT
INP
ISZ
BSA
BUN
STA
LDA
AND
new
dup
del
Format
Implementation
Execute sequence
DR<-M[AR]
AC<-DR
End
MicroInstructions
IF(AC(15)!=1) THEN
IF(D7==1) THEN
IF(DR!=0) THEN
IF(E!=0) SKIP
IF(I=0) SKIP
transferRtoR
AC(0-7)<-INPR
AC(15)<-TR(0)
AC<-DR
AC<-TR[0]
AR<-IR[4-15]
AR<-PC
OUTR<-AC[0-7]
PC<-AR
TR(0)<-E
transferRtoA
transferAtoR
OK
Cancel

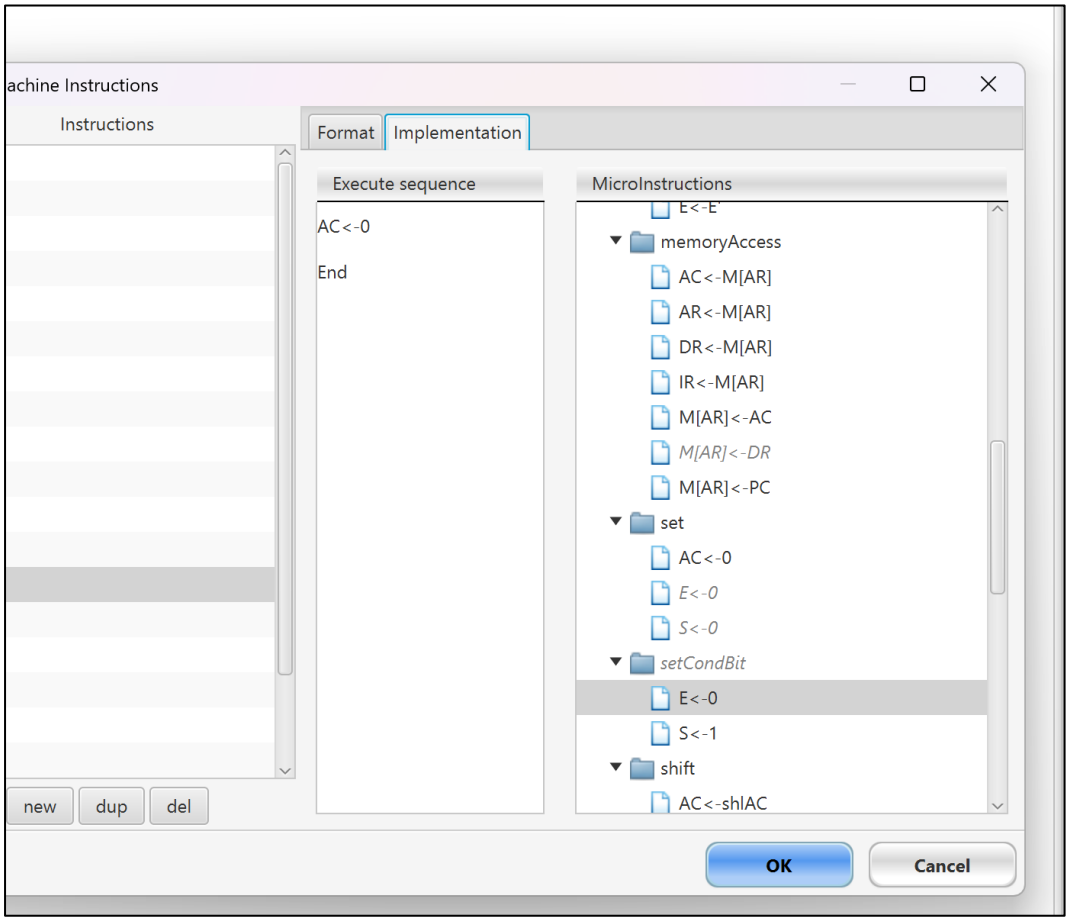
- **ADD_I (Indirect add):**



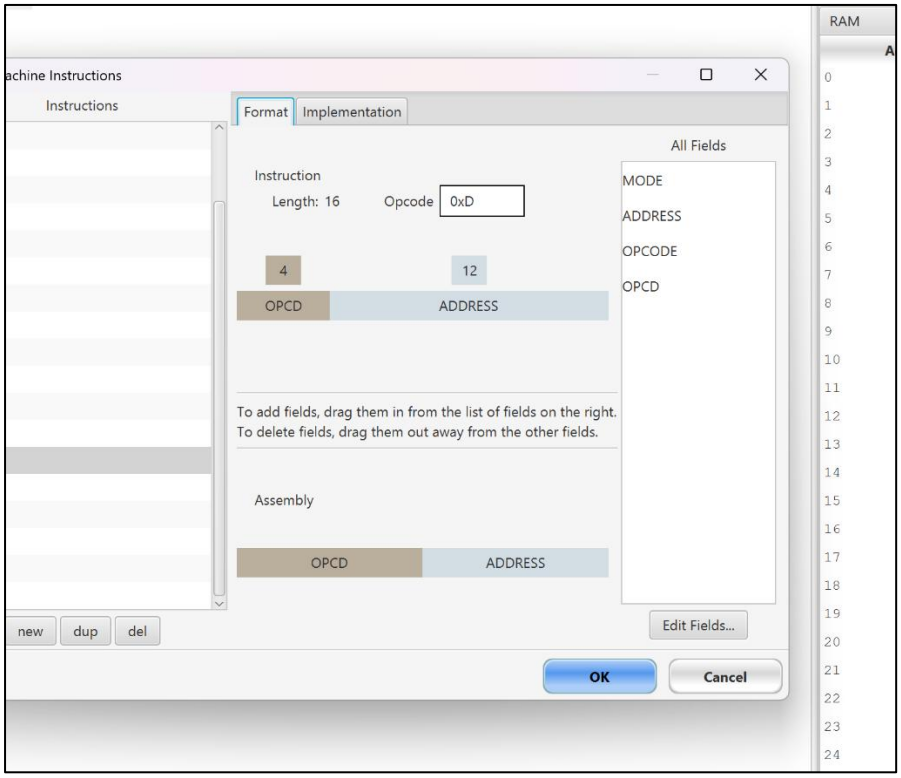



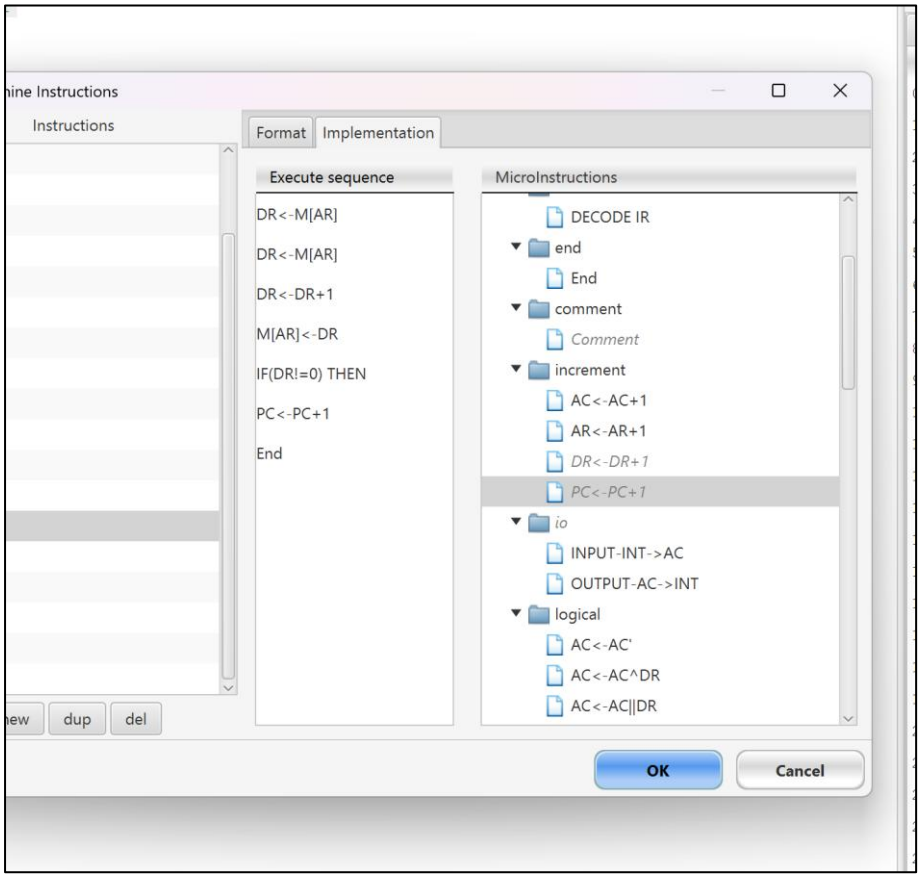
- **CLA:**


e Instructions
Instructions
Format
Implementation
All Fields
Instruction
Length: 16
Opcode
0xE800
MODE
ADDRESS
OPCODE
OPCD
16
OPCODE
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
OPCODE
dup
del
Edit Fields...
OK
Cancel

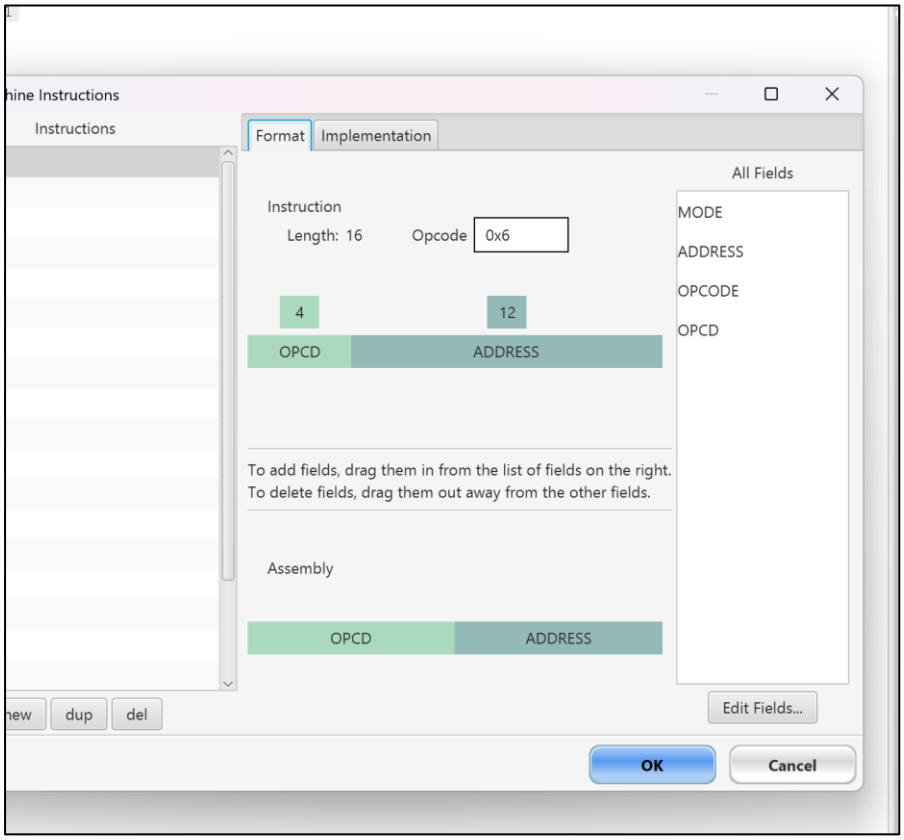

achine Instructions
Instructions
Format
Implementation
Execute sequence
AC<-0
End
MicroInstructions
memoryAccess
AC<-M[AR]
AR<-M[AR]
DR<-M[AR]
IR<-M[AR]
M[AR]<-AC
M[AR]<-DR
M[AR]<-PC
set
AC<-0
E<-0
S<-0
setCondBit
E<-0
S<-1
shift
AC<-shlAC
new
dup
del
OK
Cancel

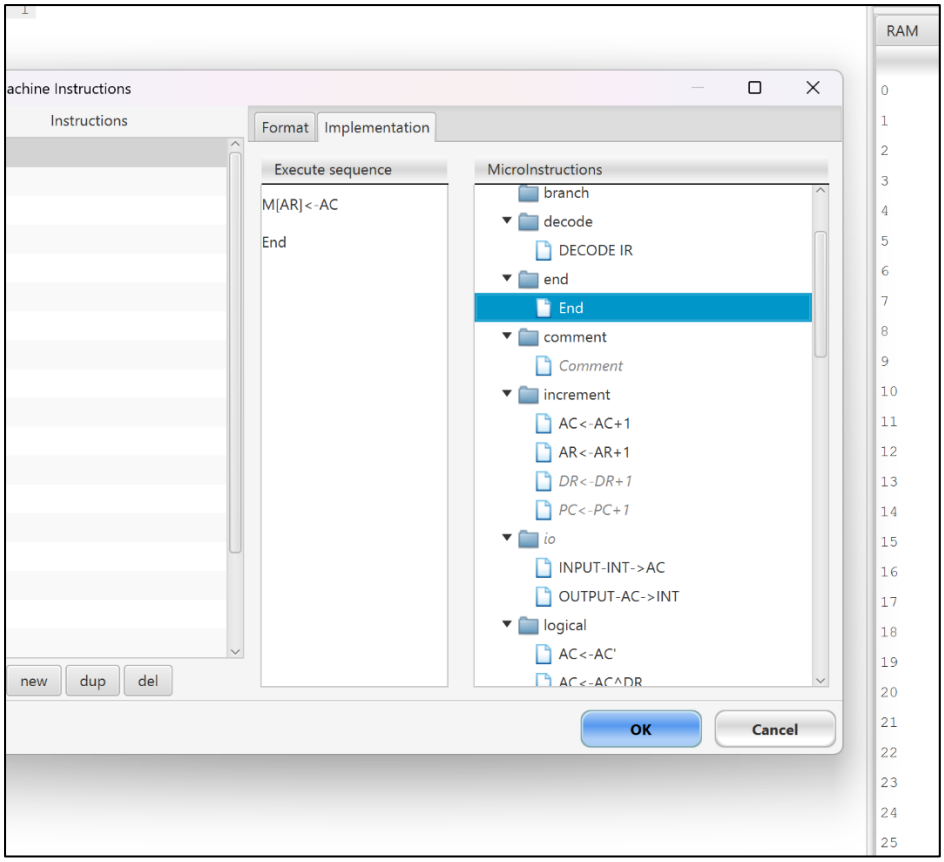
- **ISZ_I (Indirect ISZ):**



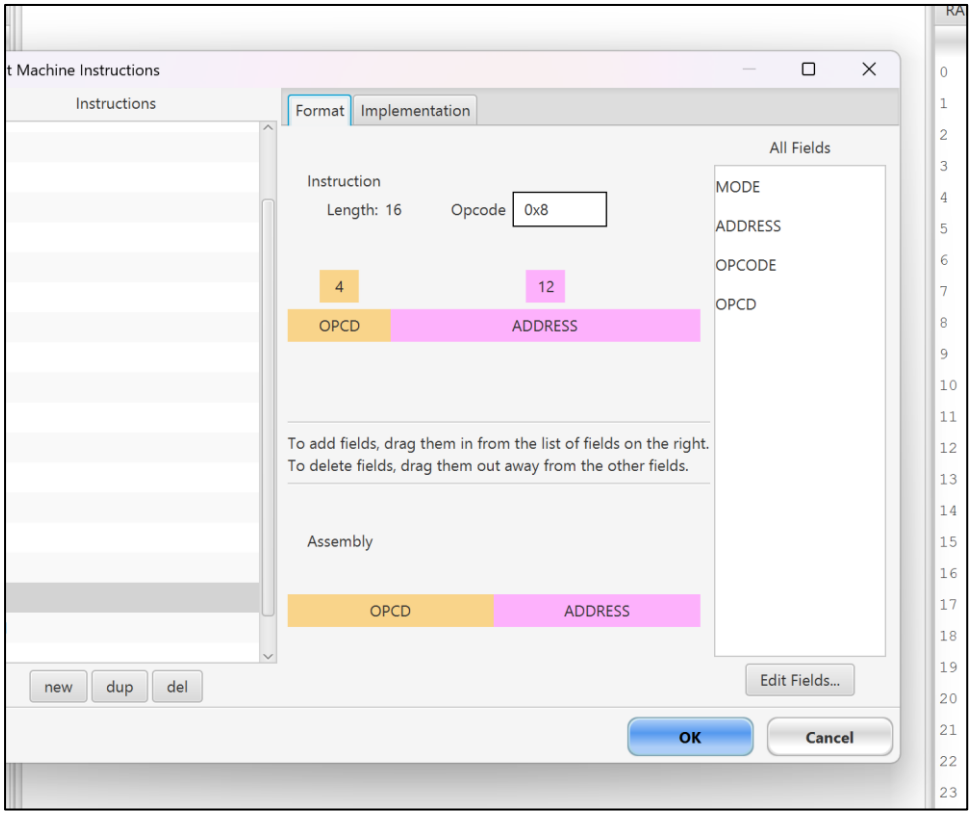



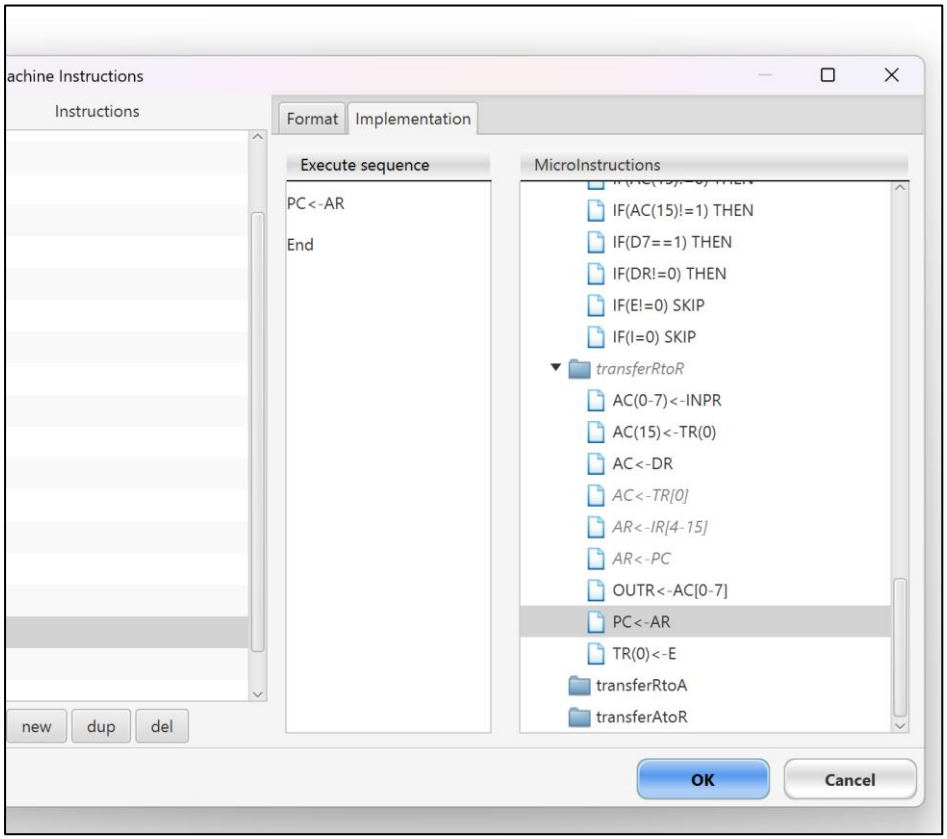
- **STA:**


hine Instructions
Instructions
Format
Implementation
All Fields
Instruction
Length: 16
Opcode
0x6
MODE
ADDRESS
OPCODE
OPCD
4
12
OPCD
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
OPCD
ADDRESS
new
dup
del
Edit Fields...
OK
Cancel

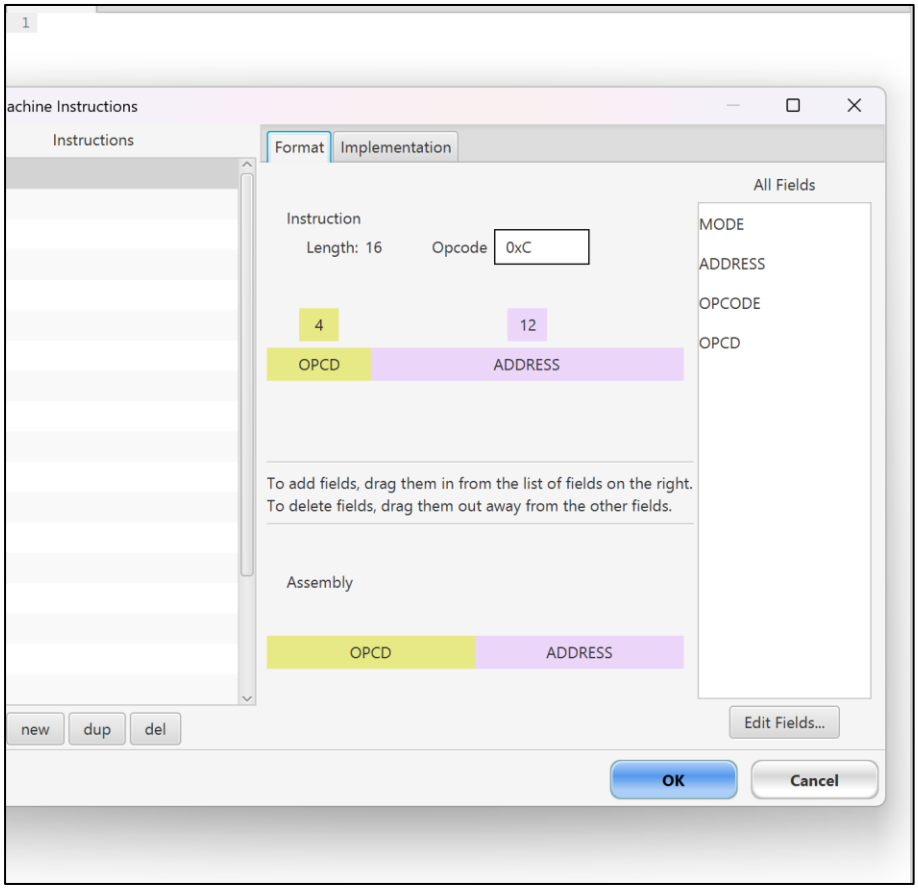

achine Instructions
Instructions
Format
Implementation
Execute sequence
M[AR]<-AC
End
MicroInstructions
branch
decode
DECODE IR
end
End
comment
Comment
increment
AC<-AC+1
AR<-AR+1
DR<-DR+1
PC<-PC+1
io
INPUT-INT->AC
OUTPUT-AC->INT
logical
AC<-AC'
new
dup
del
OK
Cancel
RAM

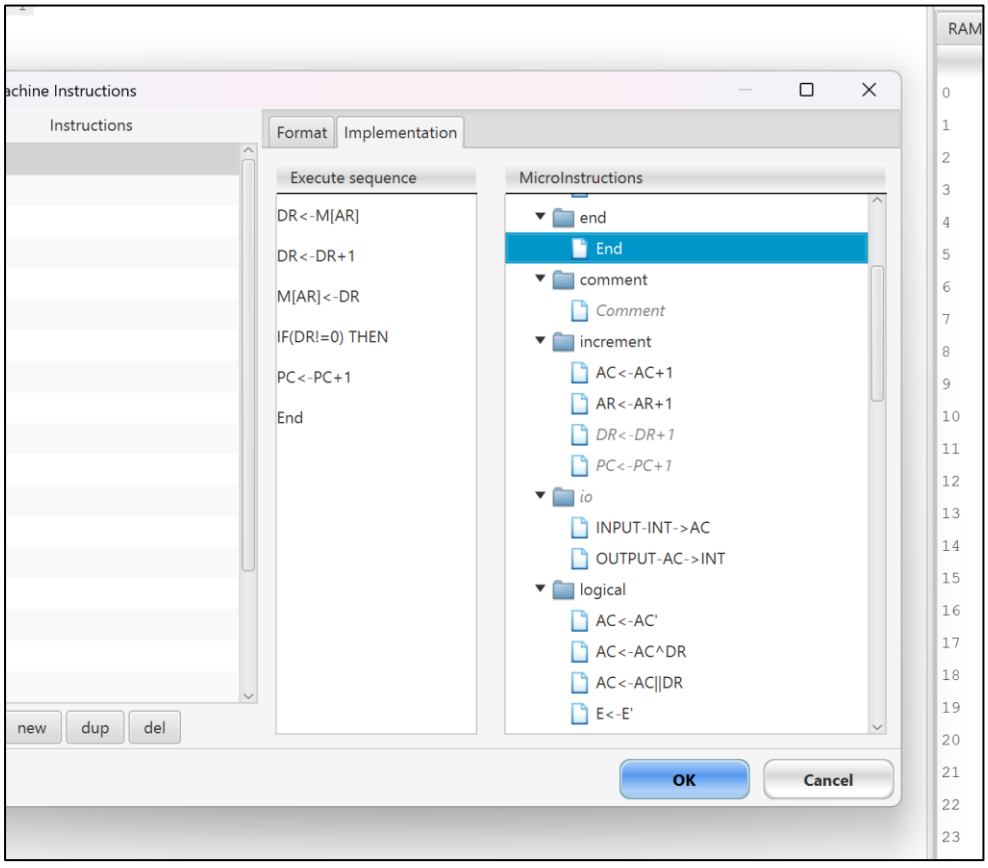
- **BUN:**


t Machine Instructions
Instructions
Format
Implementation
All Fields
Instruction
Length: 16
Opcode
0x8
MODE
ADDRESS
OPCODE
OPCD
4
12
OPCD
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
OPCD
ADDRESS
new
dup
del
Edit Fields...
OK
Cancel


achine Instructions
Instructions
Format
Implementation
Execute sequence
MicroInstructions
PC<-AR
End
IF(AC(15)!=1) THEN
IF(D7==1) THEN
IF(DR!=0) THEN
IF(E!=0) SKIP
IF(I=0) SKIP
transferRtoR
AC(0-7)<-INPR
AC(15)<-TR(0)
AC<-DR
AC<-TR[0]
AR<-IR[4-15]
AR<-PC
OUTR<-AC[0-7]
PC<-AR
TR(0)<-E
transferRtoA
transferAtoR
new
dup
del
OK
Cancel

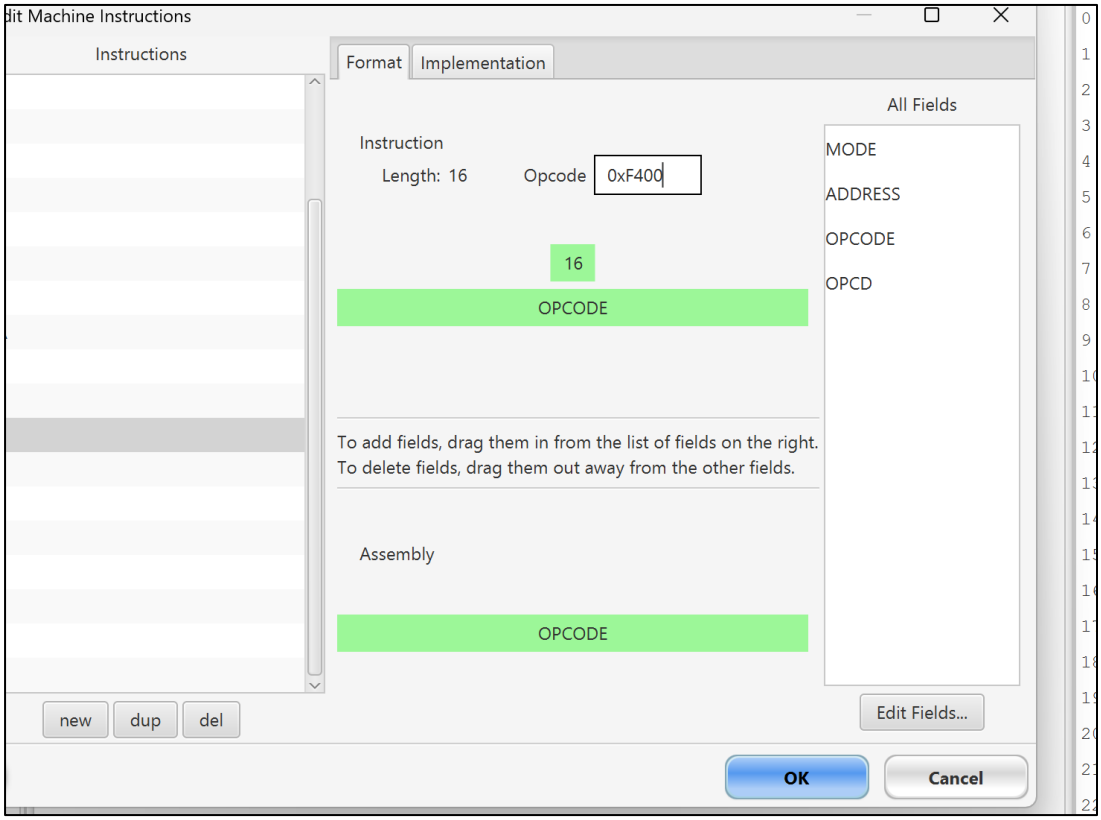
- **ISZ:**

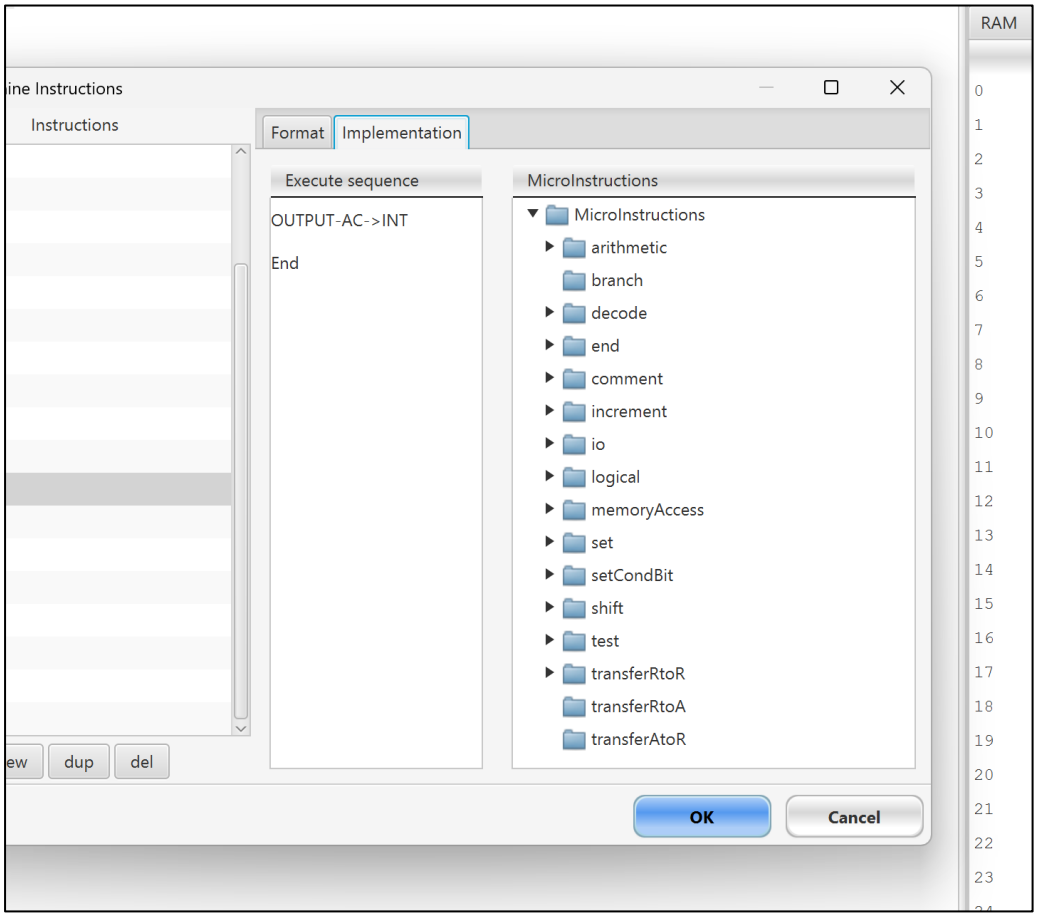

achine Instructions
Instructions
Format
Implementation
All Fields
Instruction
Length: 16
Opcode
0xC
MODE
ADDRESS
OPCODE
OPCD
4
12
OPCD
ADDRESS
To add fields, drag them in from the list of fields on the right.
To delete fields, drag them out away from the other fields.
Assembly
OPCD
ADDRESS
new
dup
del
Edit Fields...
OK
Cancel


achine Instructions
Instructions
Format
Implementation
Execute sequence
DR<-M[AR]
DR<-DR+1
M[AR]<-DR
IF(DR!=0) THEN
PC<-PC+1
End
MicroInstructions
end
End
comment
Comment
increment
AC<-AC+1
AR<-AR+1
DR<-DR+1
PC<-PC+1
io
INPUT-INT->AC
OUTPUT-AC->INT
logical
AC<-AC'
AC<-AC^DR
AC<-AC||DR
E<-E'
new
dup
del
OK
Cancel
RAM

- **AND_I (Indirect AND)**
- **LDA_I (Indirect LDA)**
- **STA_I (Indirect STA)**
- **BUN_I (Indirect BUN)**
- **CLE**
- **CMA**
- **CME**
- **CIR**
- **CIL**
- **INC**
- **SPA**
- **SNA**
- **SZA**
- **SZE**
- **HLT**
- **INP**

- **OUT**

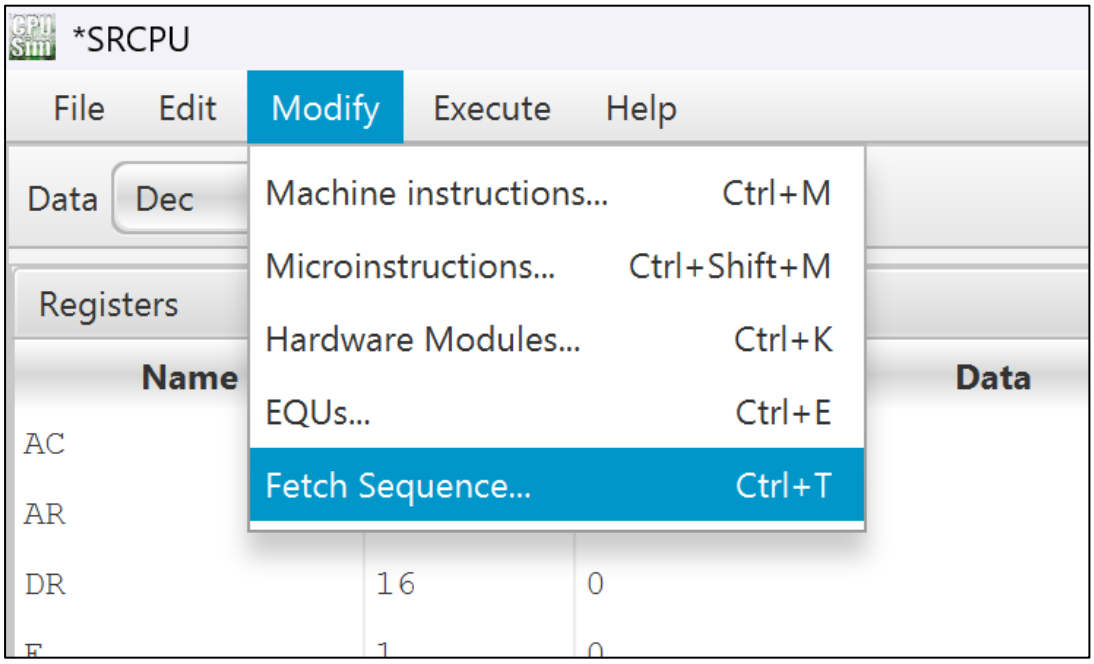

dit Machine Instructions
Instructions
Format
Implementation
All Fields
Instruction
Length: 16
Opcode
0xF400
MODE
ADDRESS
OPCODE
OPCD
16
OPCODE
To add fields, drag them in from the list of fields on the right.
To delete fields, drag them out away from the other fields.
Assembly
OPCODE
new
dup
del
Edit Fields...
OK
Cancel


RAM
ine Instructions
Instructions
Format
Implementation
Execute sequence
OUTPUT-AC->INT
End
MicroInstructions
MicroInstructions
arithmetic
branch
decode
end
comment
increment
io
logical
memoryAccess
set
setCondBit
shift
test
transferRtoR
transferRtoA
transferAtoR
ew
dup
del
OK
Cancel

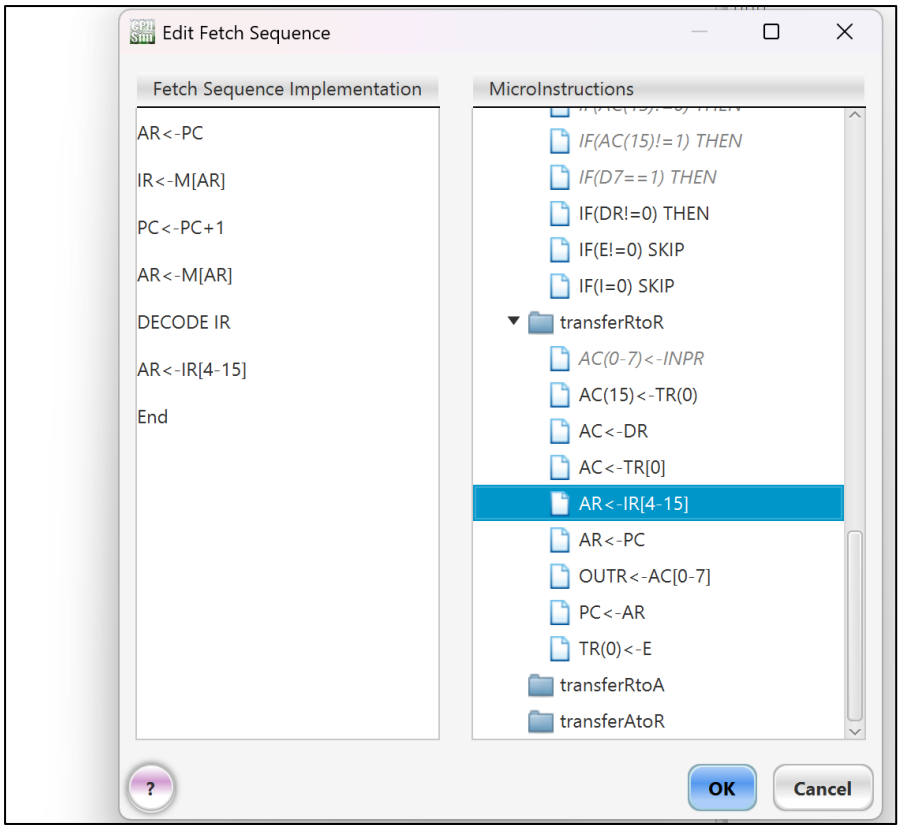
## Q2-Create a Fetch routine of the instruction cycle.

## ANSWER:

### 1. Go to modify and select fetch sequence




### 2. Grab required instructions from the microinstructions



**Q3-Write an assembly program to simulate ADD operation on two user-entered numbers.**

**ANSWER**

```
 1 INP
 2 STA A
 3 INP
 4 STA B
 5 LDA A
 6 ADD B
 7 OUT
 8 HLT
 9 A:  .data1[0]
10 B:  .data1[0]
```

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 3
Enter Inputs, the first of which must be an Integer: 2
Output:  5
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HALT-BIT]
```

**Q4- Write an assembly program to simulate SUBTRACT operation on two user-entered numbers**

**ANSWER**

```
 1 INP
 2 STA A
 3 INP
 4 STA B
 5 LDA A
 6 SUB B
 7 OUT
 8 HLT
 9 A:  .data1[0]
10 B:  .data1[0]
```

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 1
Enter Inputs, the first of which must be an Integer: 4
Output:  -3
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HALT-BIT]
```

## Q5-Write an assembly program to simulate the following logical operations on two user- entered numbers.

## ANSWER

### 1.AND

```
 1 INP
 2 STA A
 3 INP
 4 STA B
 5 LDA A
 6 AND B
 7 OUT
 8 HLT
 9 A: .data1[0]
10 B: .data1[0]
```

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 1
Enter Inputs, the first of which must be an Integer: 3
Output:  1
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HALT-BIT]
```

### 2.OR

```
 1 INP
 2 STA X
 3 INP
 4 STA Y
 5 LDA X
 6 OR Y
 7 OUT
 8 HLT
 9 X: .data1[0]
10 Y: .data1[0]
```

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 3
Enter Inputs, the first of which must be an Integer: 2
Output:  3
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HALT-BIT]
```

## 3.NOT

```
 1 INP
 2 STA A
 3 INP
 4 STA B
 5 LDA A
 6 NOT B
 7 OUT
 8 HLT
 9 A: .data1[0]
10 B: .data1[0]
```

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 1
Enter Inputs, the first of which must be an Integer: 6
Output:  -2
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HALT-BIT]
```

## 4.XOR

```
 1 INP
 2 STA A
 3 INP
 4 STA B
 5 LDA A
 6 XOR B
 7 OUT
 8 HLT
 9 A: .data1[0]
10 B: .data1[0]
```

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 4
Enter Inputs, the first of which must be an Integer: 3
Output:  7
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HALT-BIT]
```

## 5.NOR

```
 1 INP
 2 STA A
 3 INP
 4 STA B
 5 LDA A
 6 NOR B
 7 OUT
 8 HLT
 9 A: .data1[0]
10 B: .data1[0]
```

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 5
Enter Inputs, the first of which must be an Integer: 7
Output:  -8
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HALT-BIT]
```

## 6.NAND

```
 1 INP
 2 STA A
 3 INP
 4 STA B
 5 LDA A
 6 NAND B
 7 OUT
 8 HLT
 9 A: .data1[0]
10 B: .data1[0]
```

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 2
Enter Inputs, the first of which must be an Integer: 4
Output:  -1
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HALT-BIT]
```

## Q6-Write an assembly program for simulating following memory-reference instructions.

## ANSWER

## 1.ADD

```
 1 INP
 2 STA X
 3 INP
 4 STA Y
 5 LDA X
 6 ADD Y
 7 OUT
 8 HLT
 9 X: .data1[0]
10 Y: .data1[0]
```

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 2
Enter Inputs, the first of which must be an Integer: 4
Output:  6
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HALT-BIT]
```

## 2.LDA

```
 1 INP
 2 STA A
 3 INP
 4 STA B
 5 LDA A
 6 LDA B
 7 OUT
 8 HLT
 9 A: .data1[0]
10 B: .data1[0]
```

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 1
Enter Inputs, the first of which must be an Integer: 7
Output:  7
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HALT-BIT]
```

## 3.STA

```
 1 INP
 2 STA A
 3 INP
 4 STA B
 5 LDA A
 6 STA B
 7 OUT
 8 HLT
 9 A: .data1[0]
10 B: .data1[0]
```

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 2
Enter Inputs, the first of which must be an Integer: 3
Output:  2
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HALT-BIT]
```

## 4.BUN

```
1 INP
2 BUN K
3 INP
4 K: OUT
5 HLT
```

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 1
Output:  1
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HALT-BIT]
```

## 5.ISZ

```
1 INP
2 STA X
3 LDA X
4 ISZ X
5 OUT
6 HLT
7 X: .data1[0]
```

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: -1
EXECUTION HALTED DUE TO AN EXCEPTION: The step is out of range
at step 5 of ISZ.
```

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: -2
Output:  -2
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HALT-BIT]
```

**Q7-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:**

**ANSWER**

## 1.CLA

```
1 INP
2 CLA
3 OUT
4 HLT
```

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 1
Output:  0
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HALT-BIT]
```

## 2.CMA

```
1 INP
2 CMA
3 OUT
4 HLT
```

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 8
Output:  -9
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HALT-BIT]
```

## 3.CME

```
1 INP
2 CME
3 OUT
4 HLT
```

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 1
Output:  1
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HALT-BIT]
```

## 4.HLT

```
1 CMA
2 HLT
```

```
EXECUTING...
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HALT-BIT]
```

**Q8. 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:**

**ANSWER**

**1.INC**

```
1 INP
2 INC
3 OUT
4 HLT
```

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 2
Output:  3
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HALT-BIT]
```

**2.SPA**

```
1 INP
2 SPA
3 OUT
4 HLT
```

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 8
Output:  8
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HALT-BIT]
```

**3.SNA**

```
1 INP
2 SNA
3 OUT
4 HLT
```

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: -5
Output:  -5
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HALT-BIT]
```

**4.SZE**

```
1 INP
2 SZE
3 OUT
4 HLT
```

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 1
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HALT-BIT]
```

**Q9. 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:**

**ANSWER**

**1.CIR**

```
1 INP
2 CIR
3 OUT
4 HLT
```

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 2
Output:  1
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HALT-BIT]
```

## 2.CIL

```
1 INP
2 CIL
3 OUT
4 HLT
```

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 4
Output:  8
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HALT-BIT]
```

## Q10. 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).

## ANSWER

```
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:.data2[0]
```

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 6
Enter Inputs, the first of which must be an Integer: 7
Enter Inputs, the first of which must be an Integer: 8
Enter Inputs, the first of which must be an Integer: 0
Output:  21
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HALT-BIT]
```

## Q11. Write an assembly program that reads in integers and adds them together; until zero is read in. Then it outputs the sum.

## ANSWER

```
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:.data2[0]
```

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 6
Enter Inputs, the first of which must be an Integer: 7
Enter Inputs, the first of which must be an Integer: -1
Output:  13
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [HALT-BIT]
```