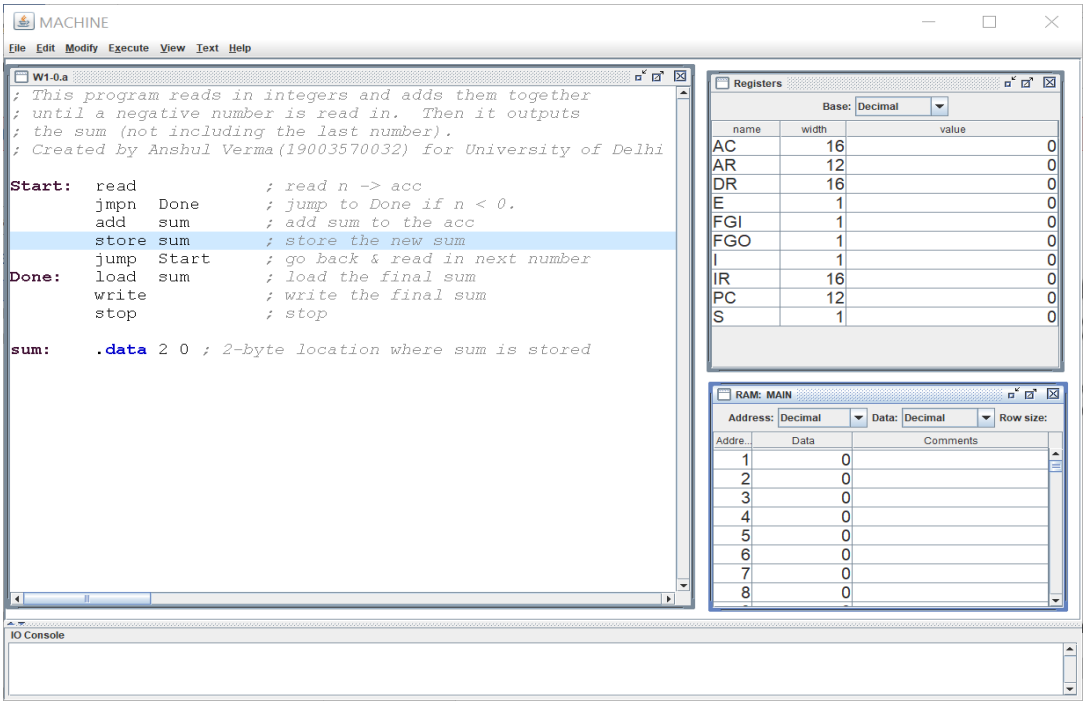
Atma Ram Sanatan Dharma College
University of Delhi



Computer System Architecture

Practical File for Paper Code 32341102

Submitted By

Anshul Verma
College Roll No. - 19/78065
University Roll No. - 19003570032
BSc (Hons) Computer Science

Submitted To

Ms Uma Ojha
Department of Computer Science

INDEX

PRACTICAL 1

Dated 03rd September 2019

Objective

Create a machine based on the given architecture and design the register set, memory and instruction formats for the machine.

Register Set

IR	DR	AC	AR	PC	FGI	FGO	S	I	E
0 15	0 15	0 15	011	011	1 Bit	1 Bit	1 Bit	1 bit	1 Bit

Memory and Instruction Format

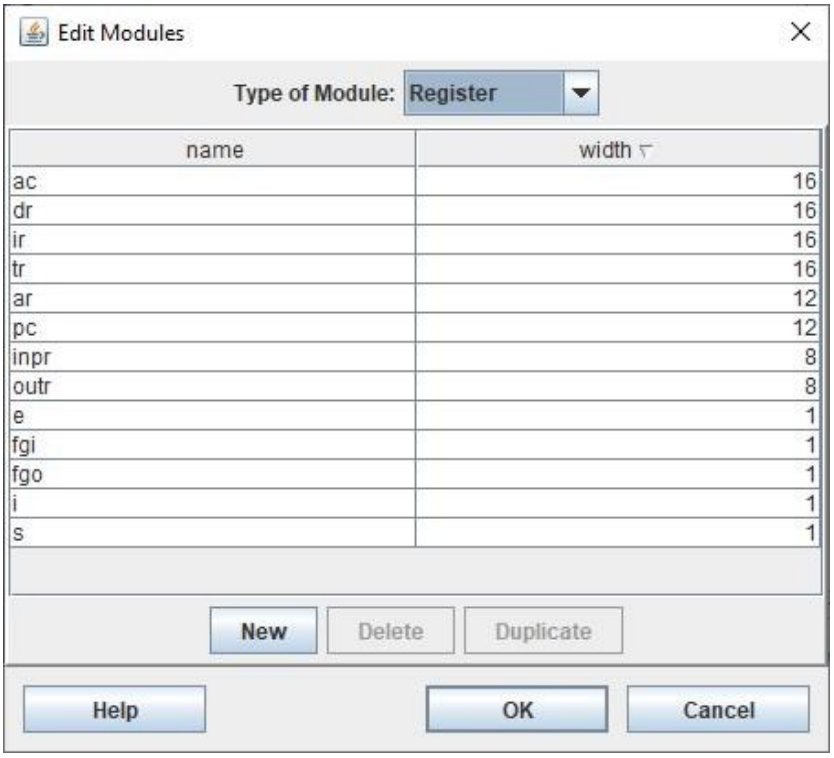
Memory	Instruction format
4096 words 16 bits per word	0 3 4 15 Opcode | Address

Basic Computer Instructions

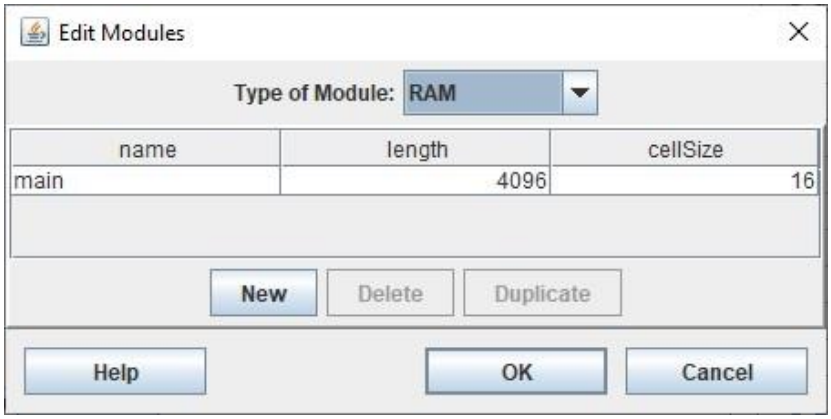
	Hexadecimal code		
Symbol	$I = 0$	$I = 1$	Description
AND	0xxx	8xxx	AND memory word to *AC*
ADD	1xxx	9xxx	Add memory word to *AC*
LDA	2xxx	Axxx	Load memory word to *AC*
STA	3xxx	Bxxx	Store content of *AC* in memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
CLA	7800		Clear *AC*
CLE	7400		Clear *E*
CMA	7200		Complement *AC*
CME	7100		Complement *E*
CIR	7080		Circulate right *AC* and *E*
CIL	7040		Circulate left *AC* and *E*
INC	7020		Increment *AC*
SPA	7010		Skip next instruction if *AC* positive
SNA	7008		Skip next instruction if *AC* negative
SZA	7004		Skip next instruction if *AC* zero
SZE	7002		Skip next instruction if *E* is 0
HLT	7001		Halt computer
INP	F800		Input character to *AC*
OUT	F400		Output character from *AC*
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag
ION	F080		Interrupt on
IOF	F040		Interrupt off

Observation

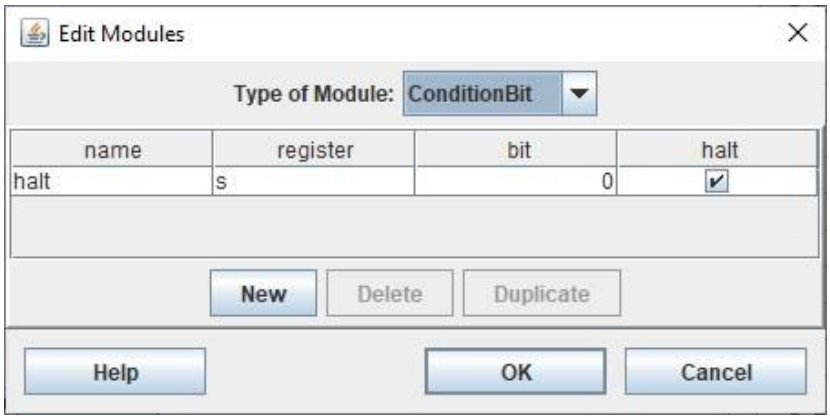
Registers



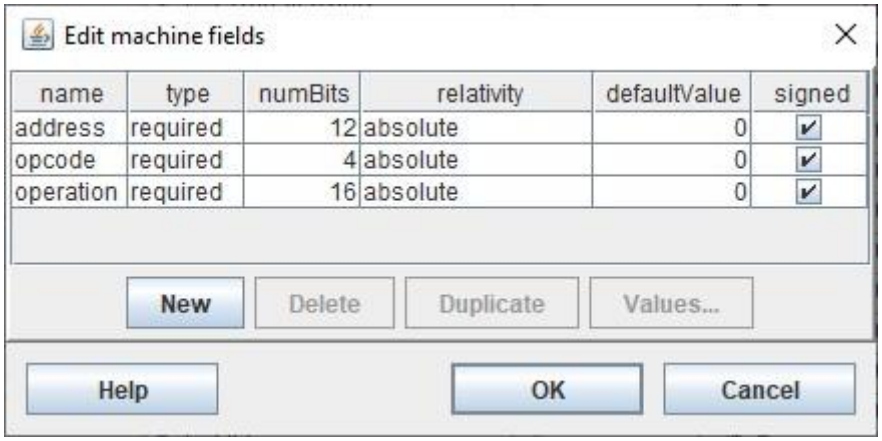
Memory



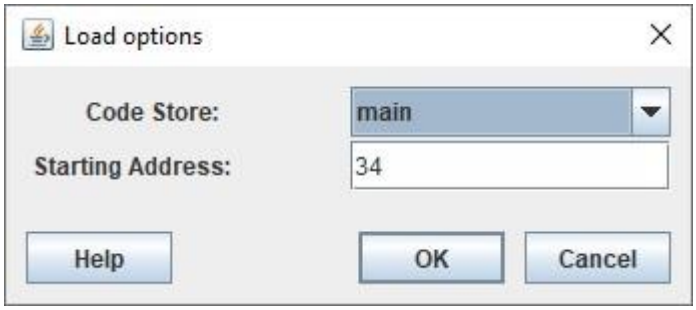
Condition Bits



Instruction Formats



Loading Point



PRACTICAL 2

Dated 03rd September 2019

Objective

Create the micro-operations and associate with instructions as given in the list of basic computer instructions. Design the instruction set for the machine keeping in mind that bits are left indexed in CPUSim.

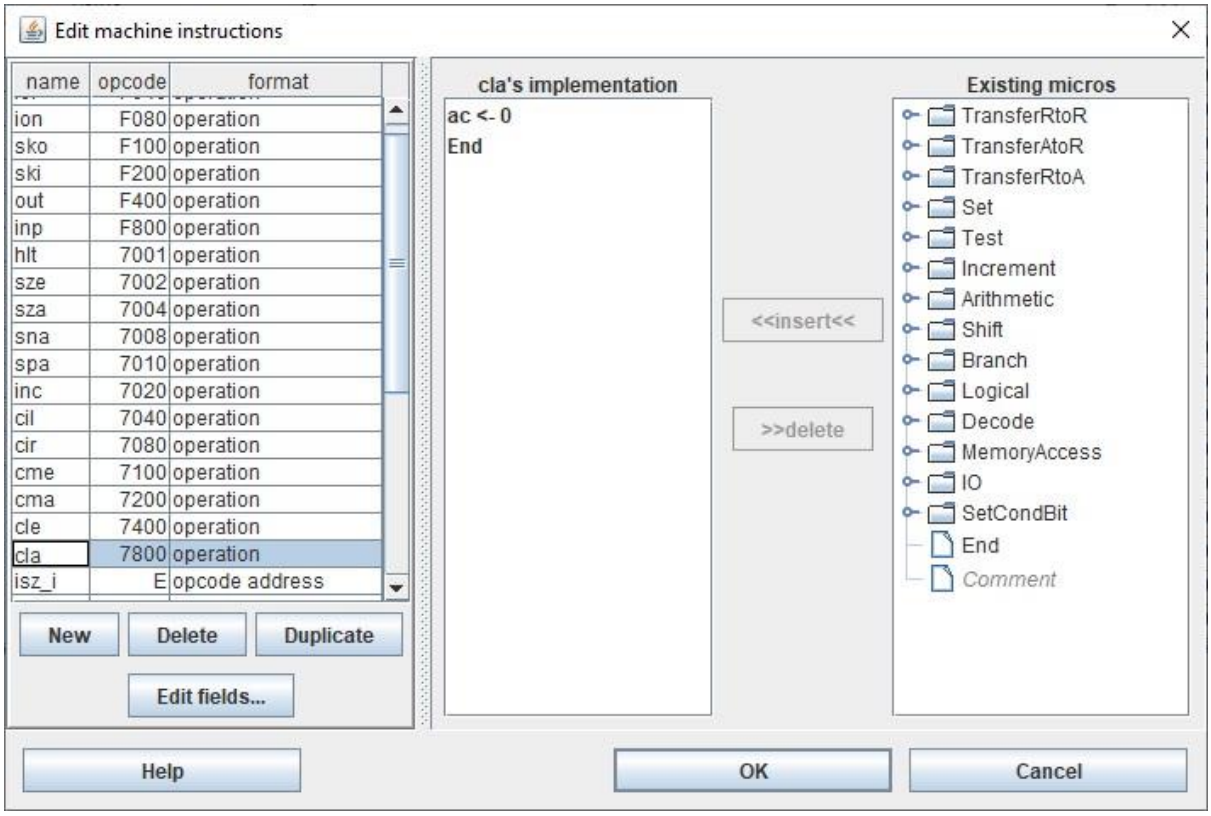
Observation

Register Reference Instructions

$D_7I'T_3 = r$ (common to all register-reference instructions)
$IR(i) = B_i$ [bit in $IR(0–11)$ that specifies the operation]

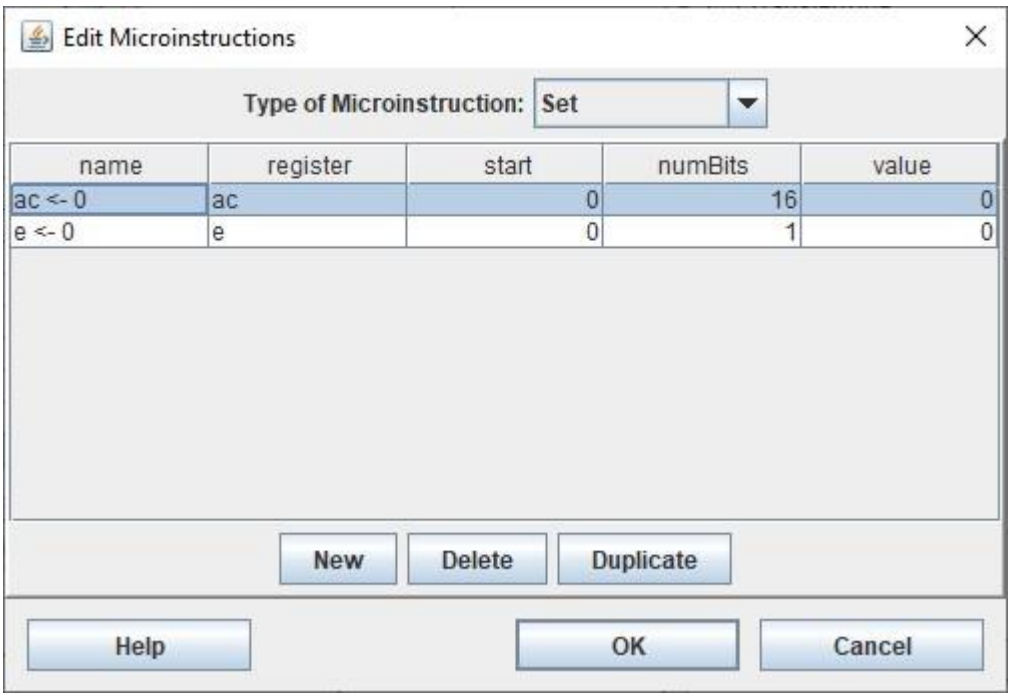
	r:	$SC \leftarrow 0$	Clear SC
CLA	rB_{11}:	$AC \leftarrow 0$	Clear AC
CLE	rB_{10}:	$E \leftarrow 0$	Clear E
CMA	rB_9:	$AC \leftarrow \overline{AC}$	Complement AC
CME	rB_8:	$E \leftarrow \overline{E}$	Complement E
CIR	rB_7:	$AC \leftarrow \text{shr } AC, AC(15) \leftarrow E, E \leftarrow AC(0)$	Circulate right
CIL	rB_6:	$AC \leftarrow \text{shl } AC, AC(0) \leftarrow E, E \leftarrow AC(15)$	Circulate left
INC	rB_5:	$AC \leftarrow AC + 1$	Increment AC
SPA	rB_4:	If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$	Skip if positive
SNA	rB_3:	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$	Skip if negative
SZA	rB_2:	If $(AC = 0)$ then $PC \leftarrow PC + 1)$	Skip if AC zero
SZE	rB_1:	If $(E = 0)$ then $(PC \leftarrow PC + 1)$	Skip if E zero
HLT	rB_0:	$S \leftarrow 0$ (S is a start–stop flip-flop)	Halt computer

CLA – Clear AC



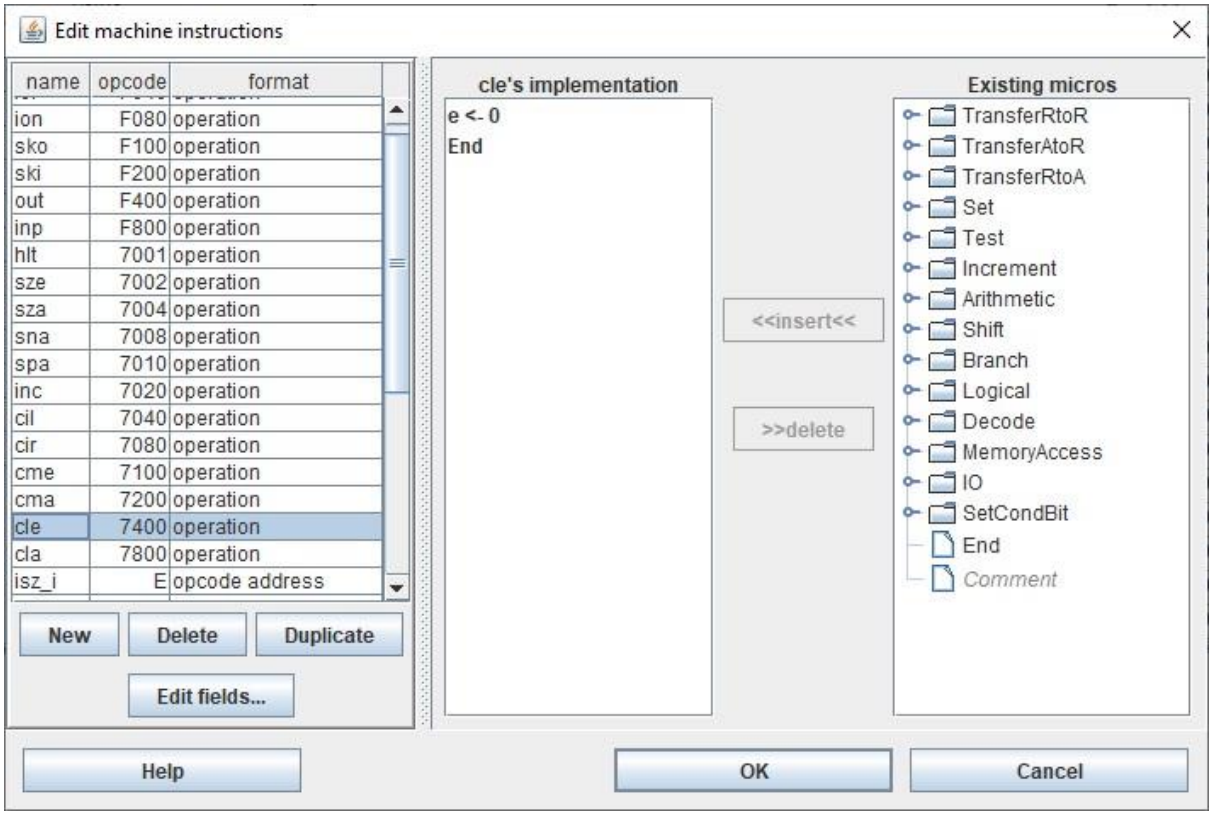
Implementation of CLA

- **AC <- 0**



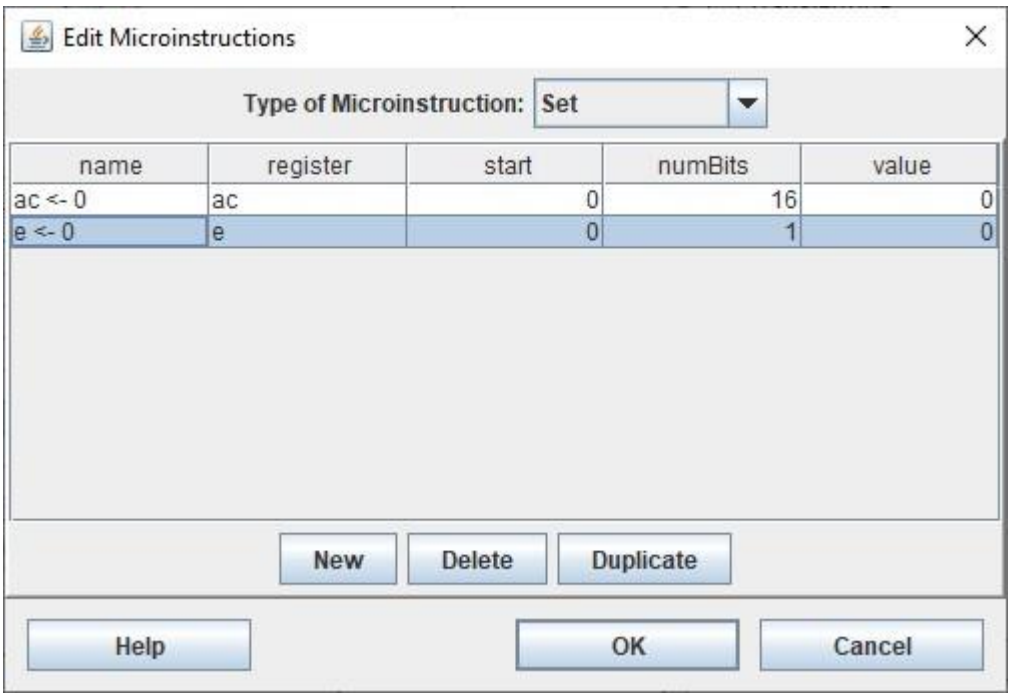
- **SC <- 0**

CLE – Clear E



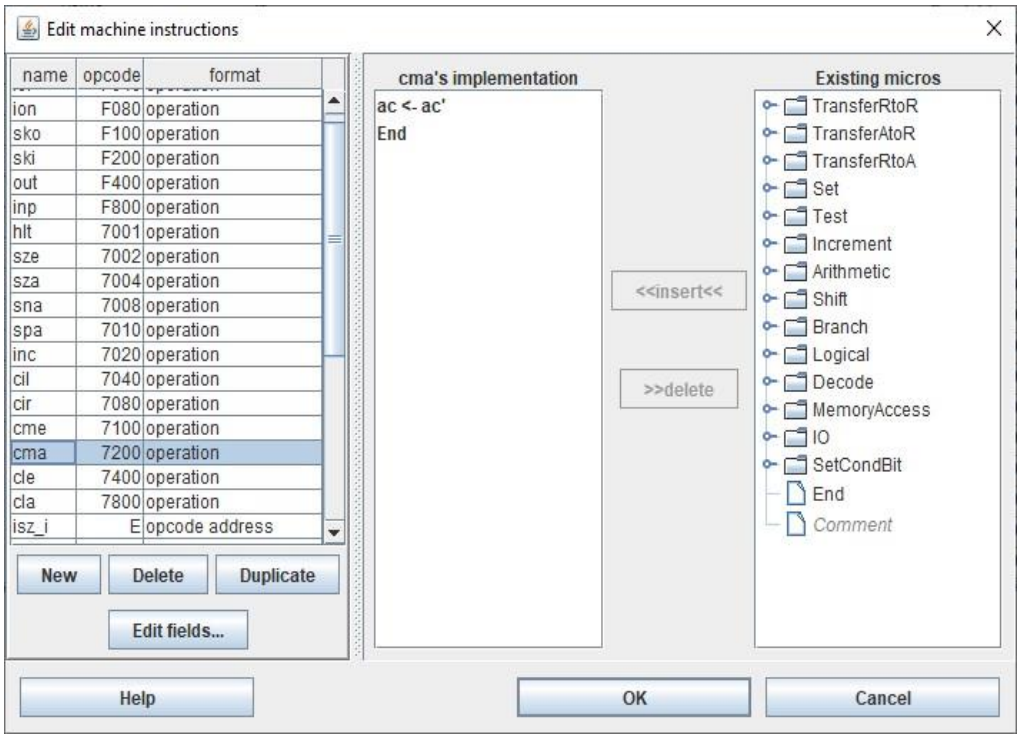
Implementation of CLE

- **E <- 0**



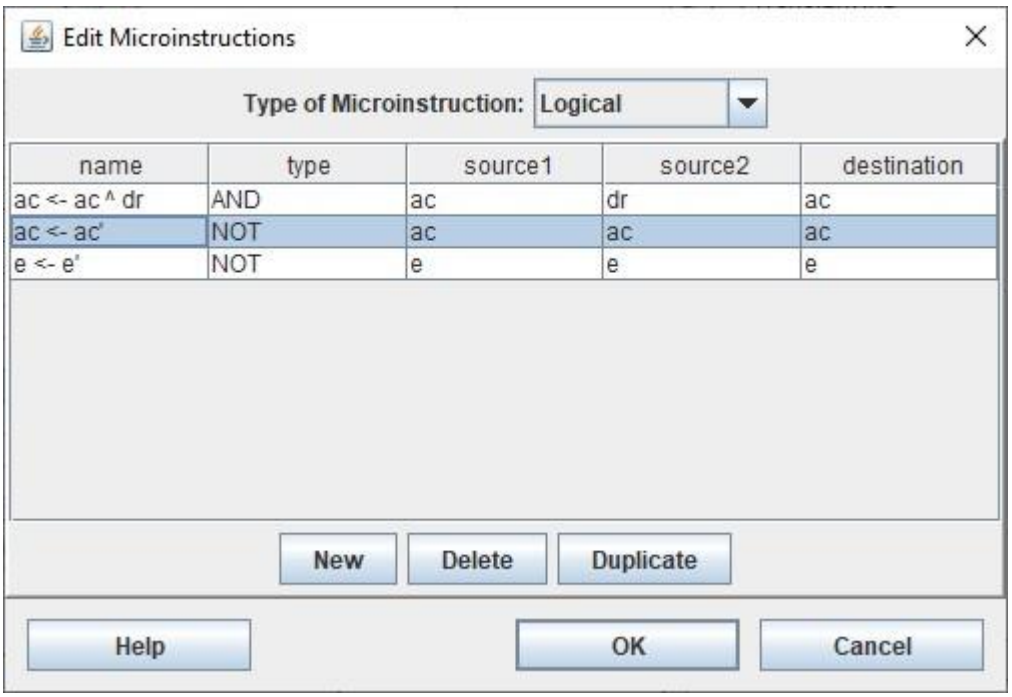
- **SC <- 0**

CMA – Complement AC



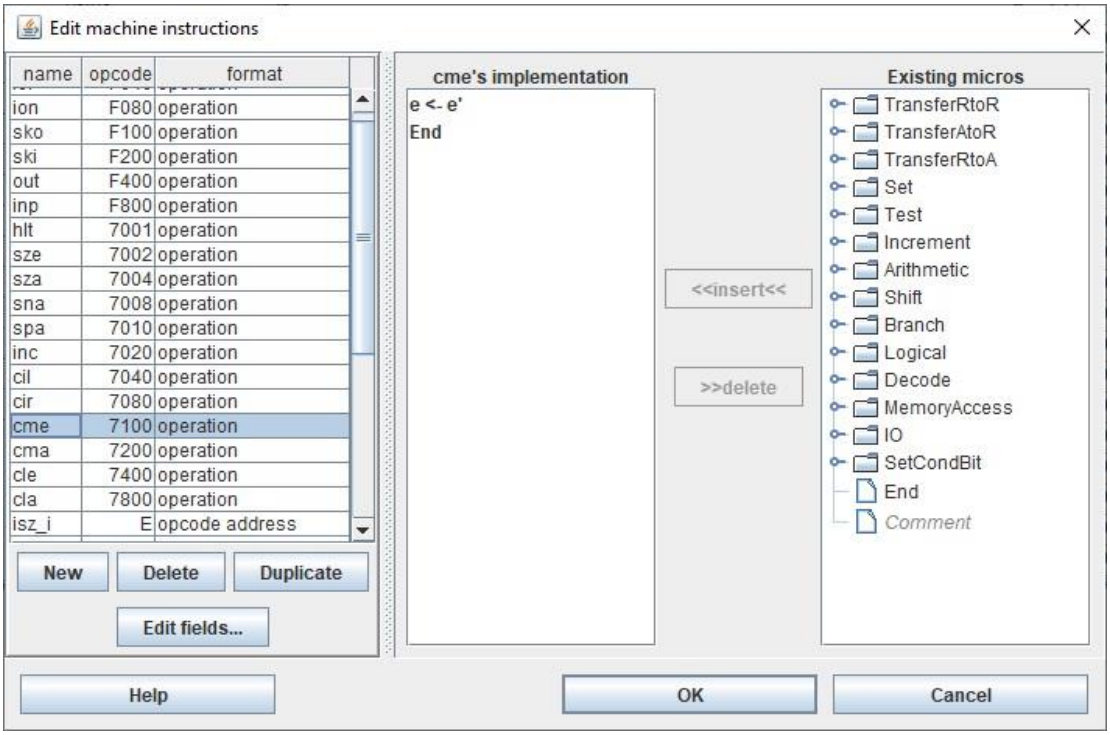
Implementation of CMA

- **AC <- AC ’**



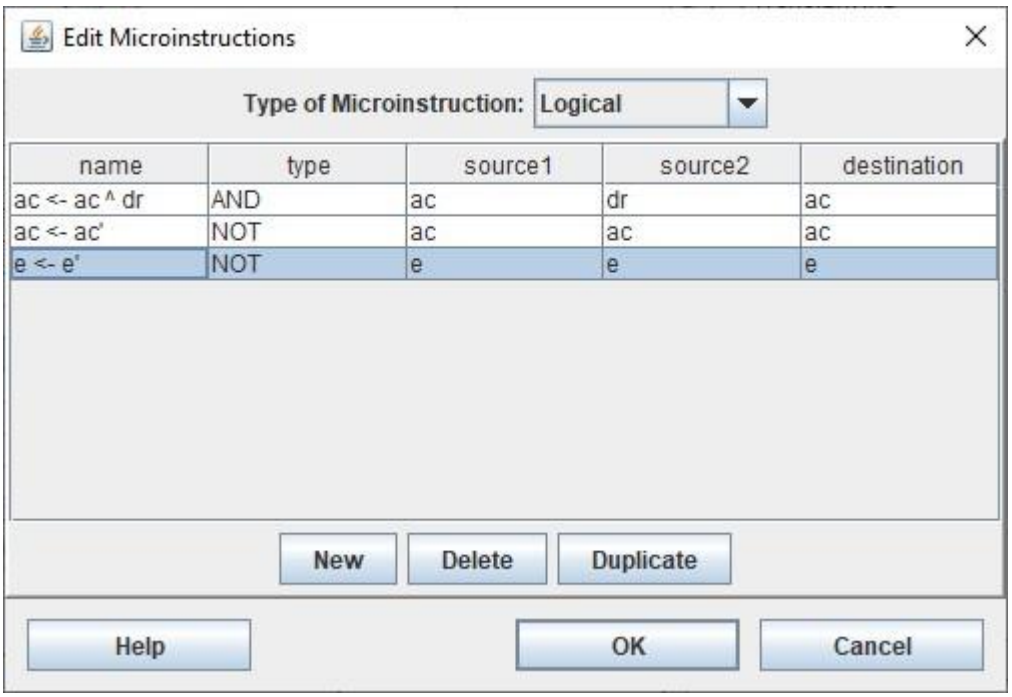
- **SC <- 0**

CME – Complement E



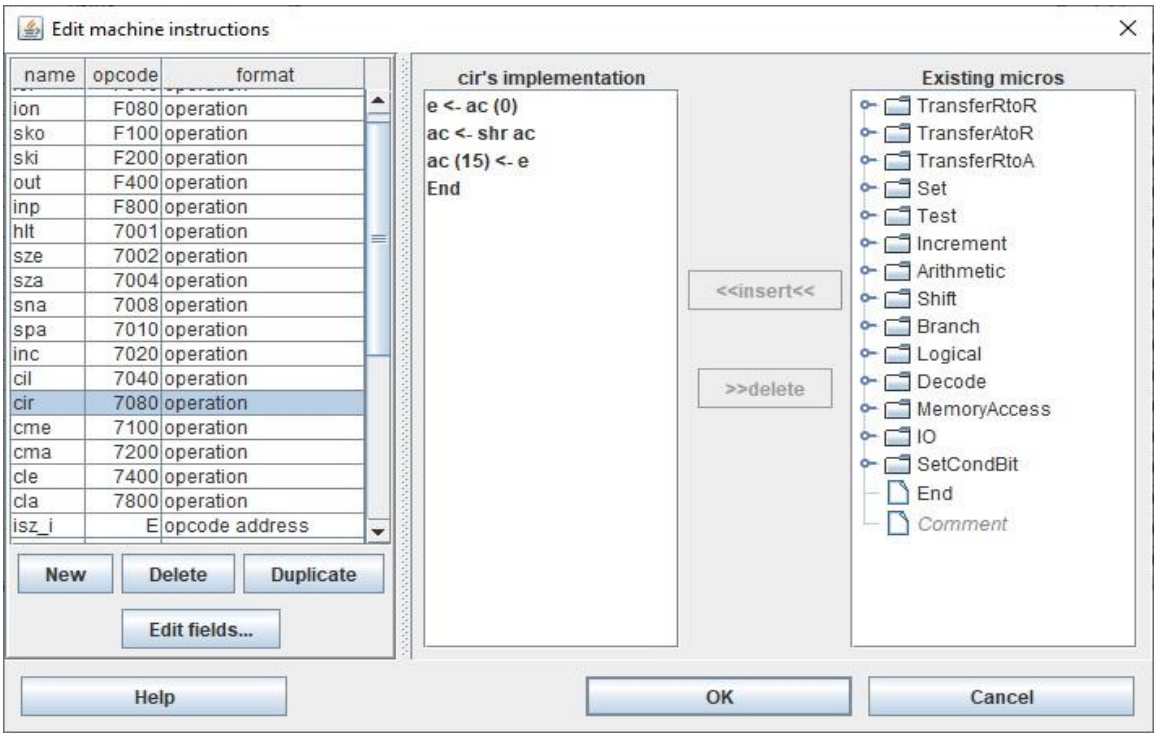
Implementation of CME

- **E <- E ‘**



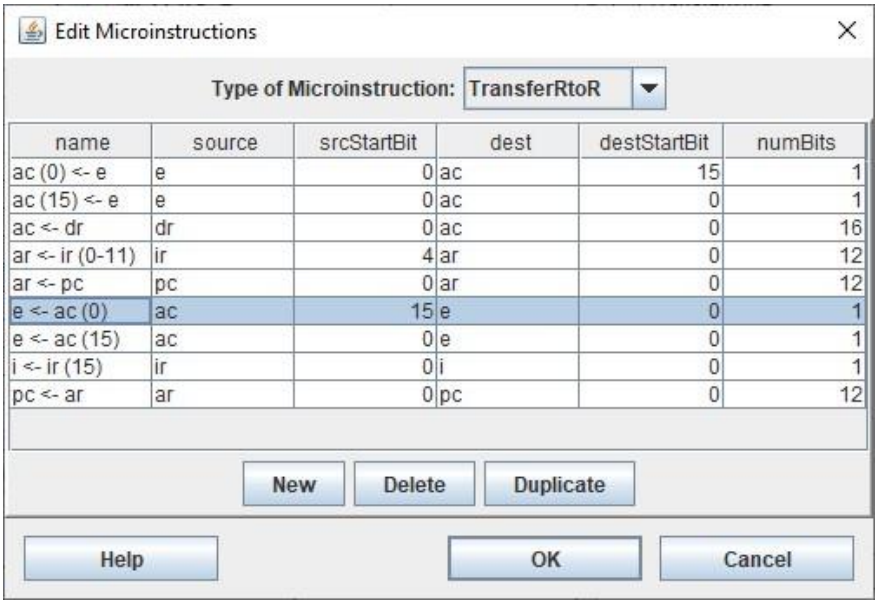
- **SC <- 0**

CIR – Circulate Right

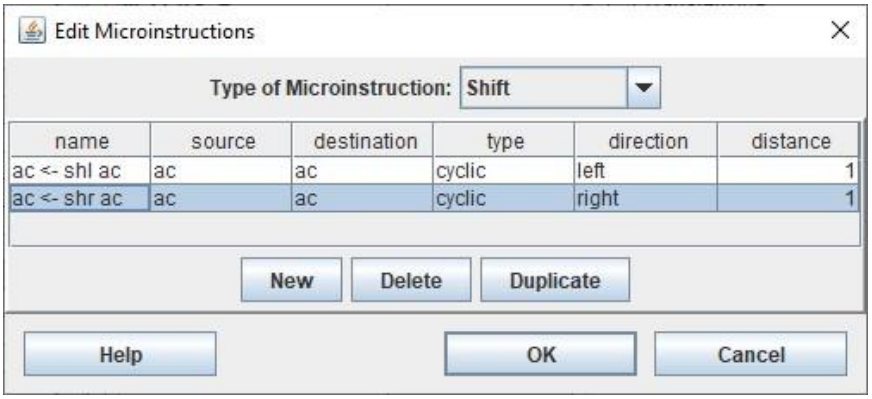


Implementation of CIR

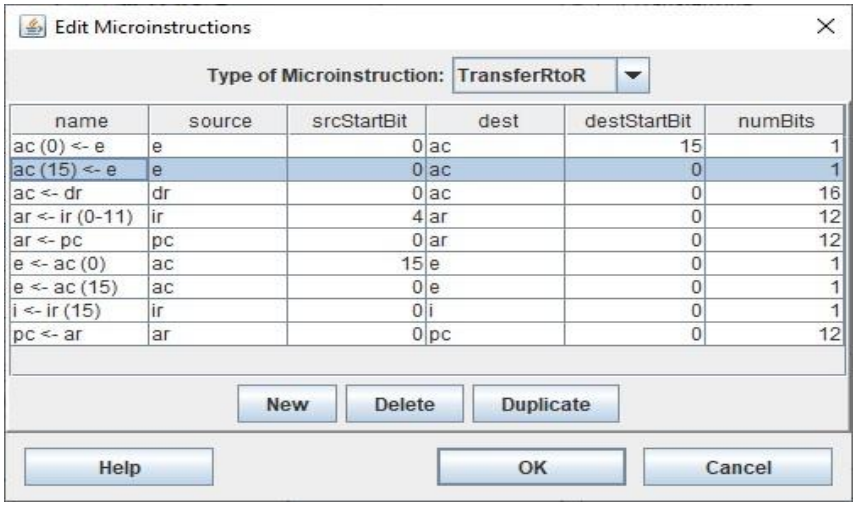
- **E <- AC (0)**



- **AC <- shr AC**

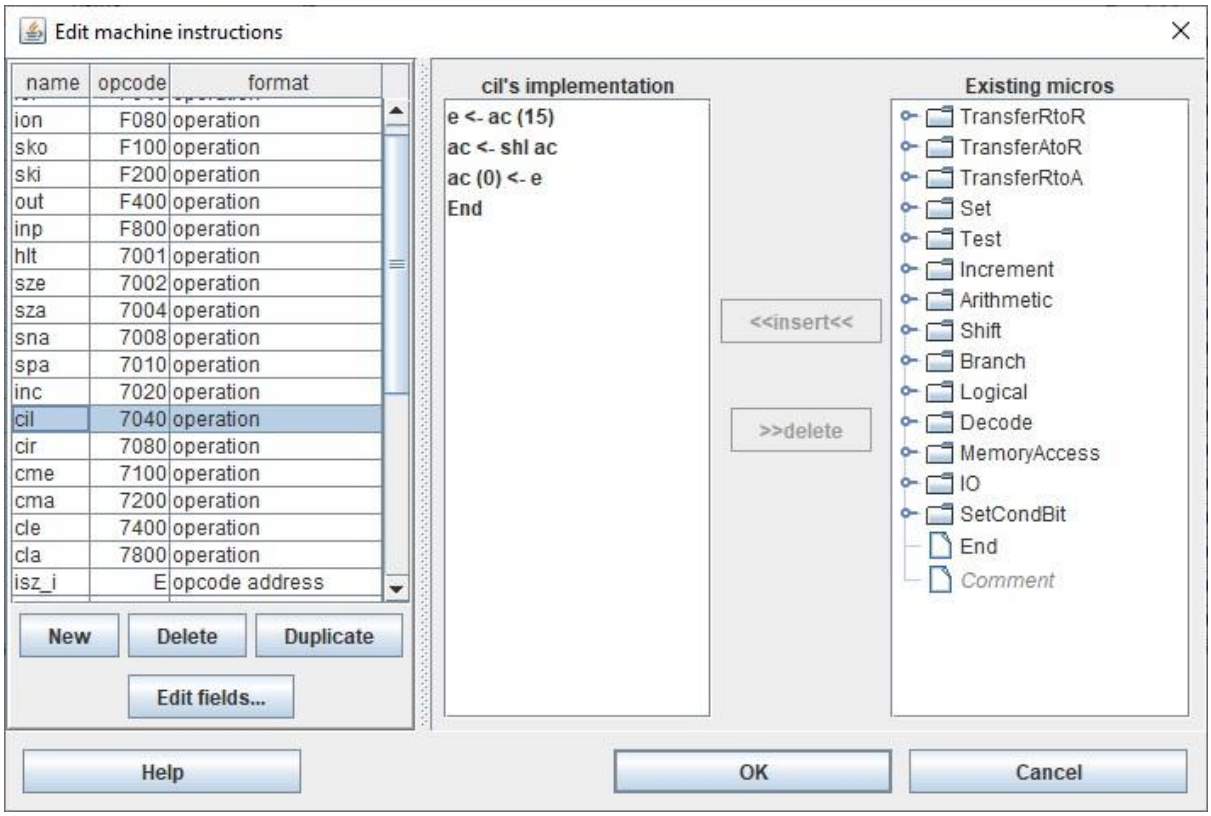


- **AC (15) <- 0**



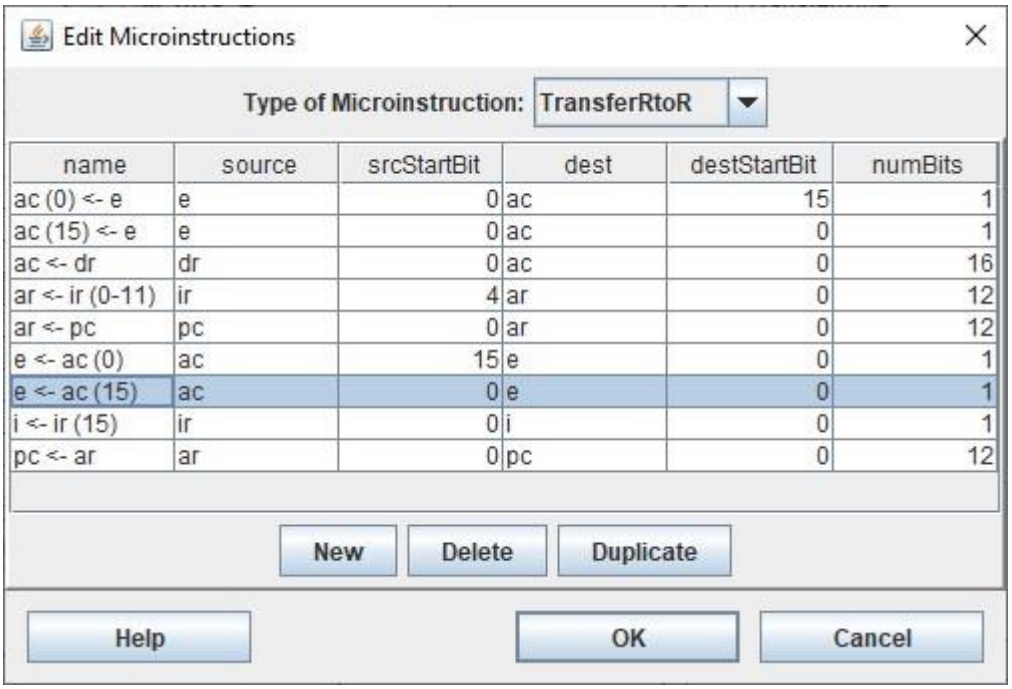
- **SC <- 0**

CIL – Circulate Left

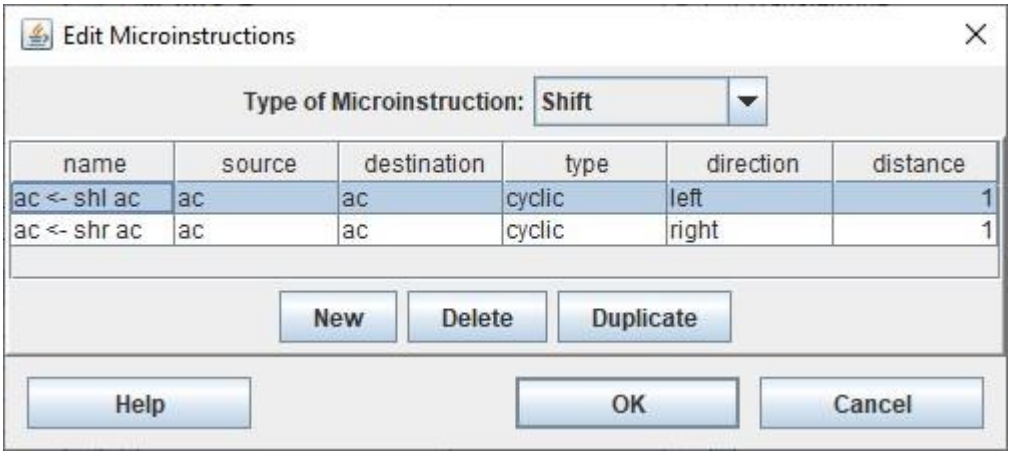


Implementation of CIL

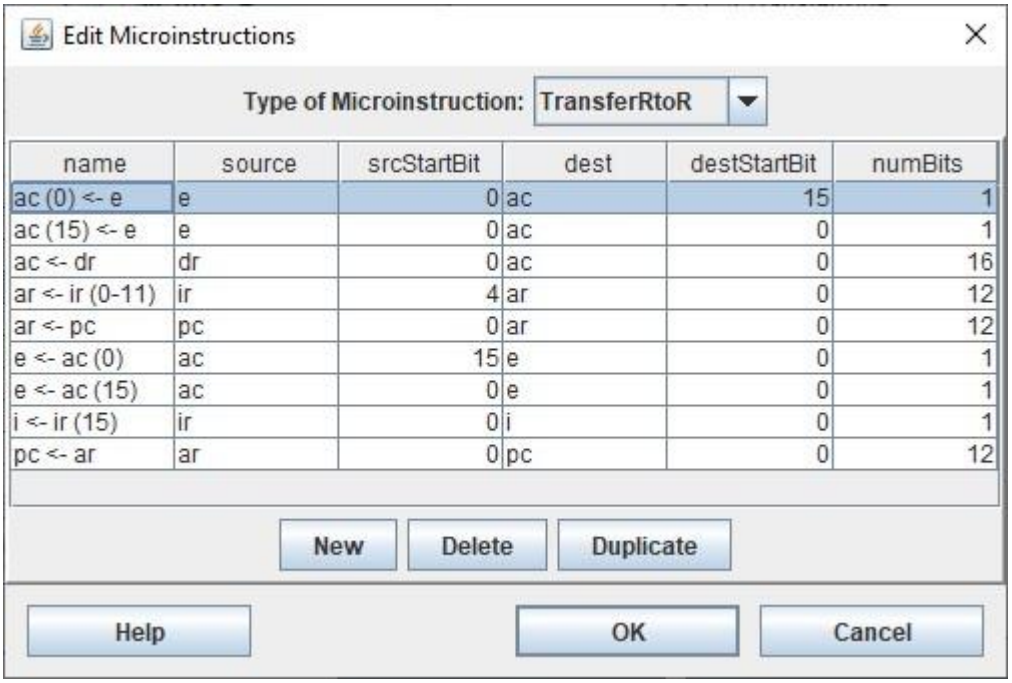
- **E <- AC (15)**



- **AC <- shl AC**

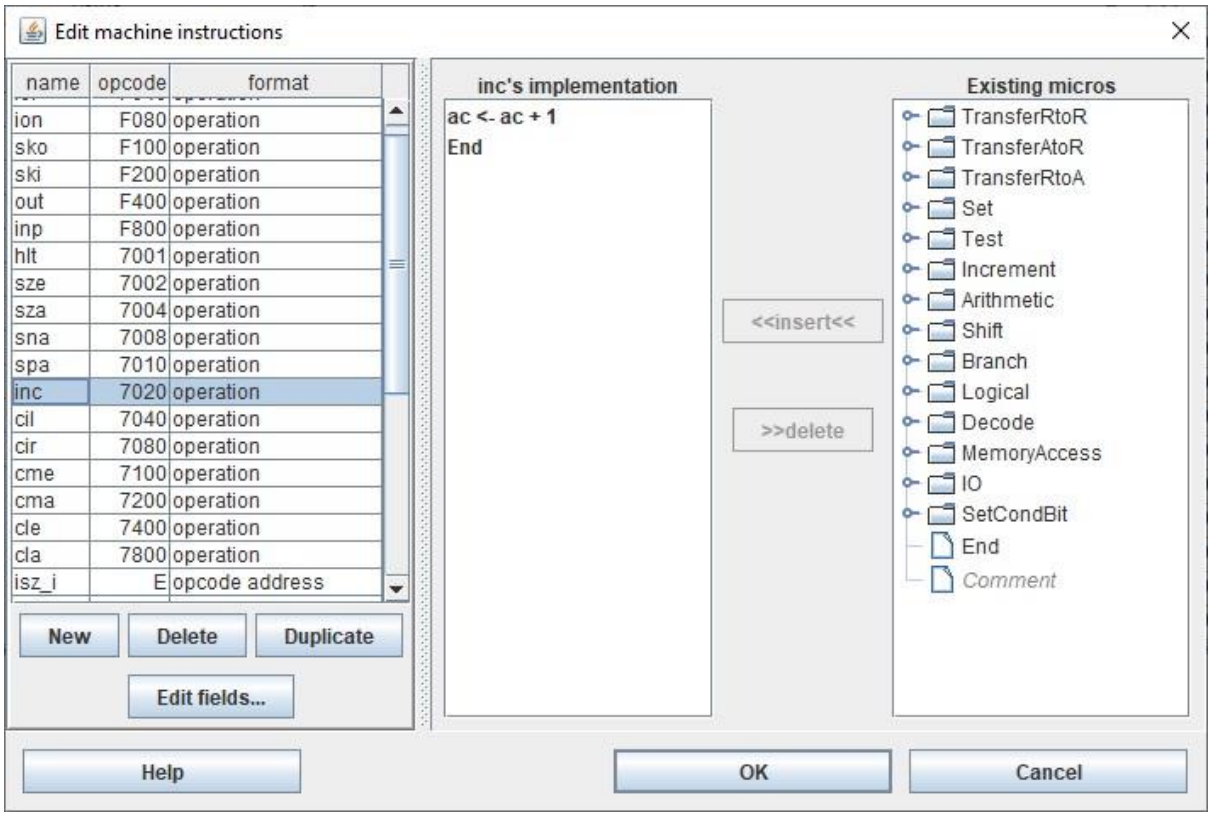

Edit Microinstructions
Type of Microinstruction: Shift
name source destination type direction distance
ac <- shl ac ac ac cyclic left 1
ac <- shr ac ac ac cyclic right 1
New Delete Duplicate
Help OK Cancel

- **AC (0) <- E**


Edit Microinstructions
Type of Microinstruction: TransferRtoR
name source srcStartBit dest destStartBit numBits
ac (0) <- e e 0 ac 15 1
ac (15) <- e e 0 ac 0 1
ac <- dr dr 0 ac 0 16
ar <- ir (0-11) ir 4 ar 0 12
ar <- pc pc 0 ar 0 12
e <- ac (0) ac 15 e 0 1
e <- ac (15) ac 0 e 0 1
i <- ir (15) ir 0 i 0 1
pc <- ar ar 0 pc 0 12
New Delete Duplicate
Help OK Cancel

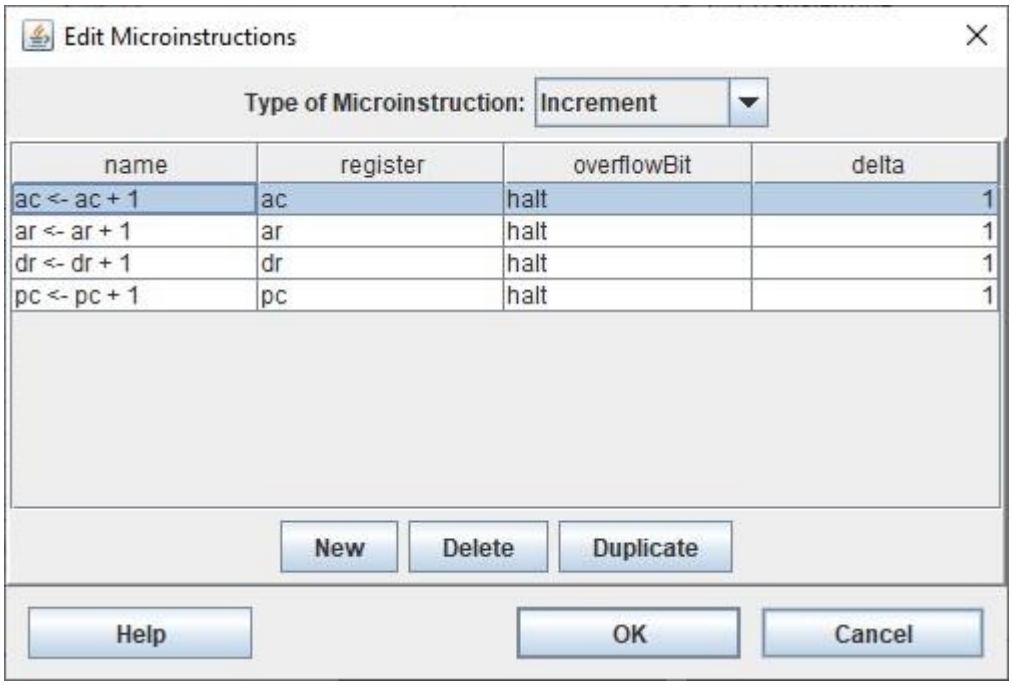
- **SC <- 0**

INC – Increment AC



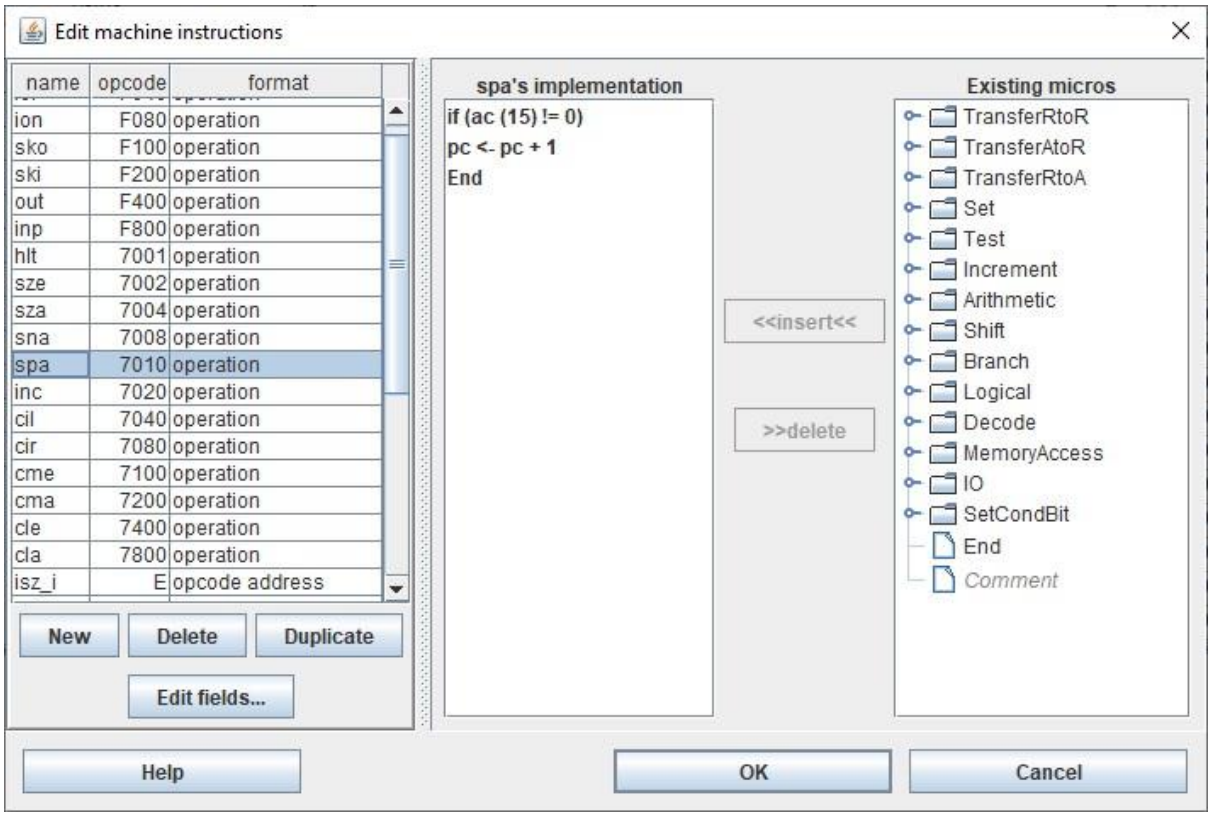
Implementation of INC

- **AC <- AC + 1**



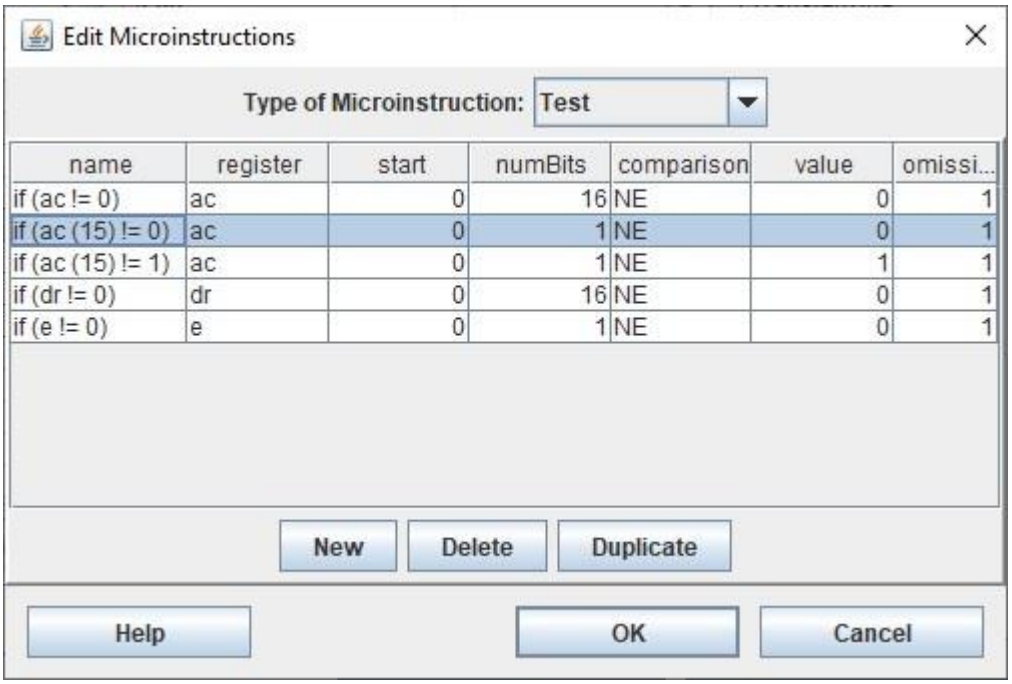
- **SC <- 0**

SPA – Skip If Positive

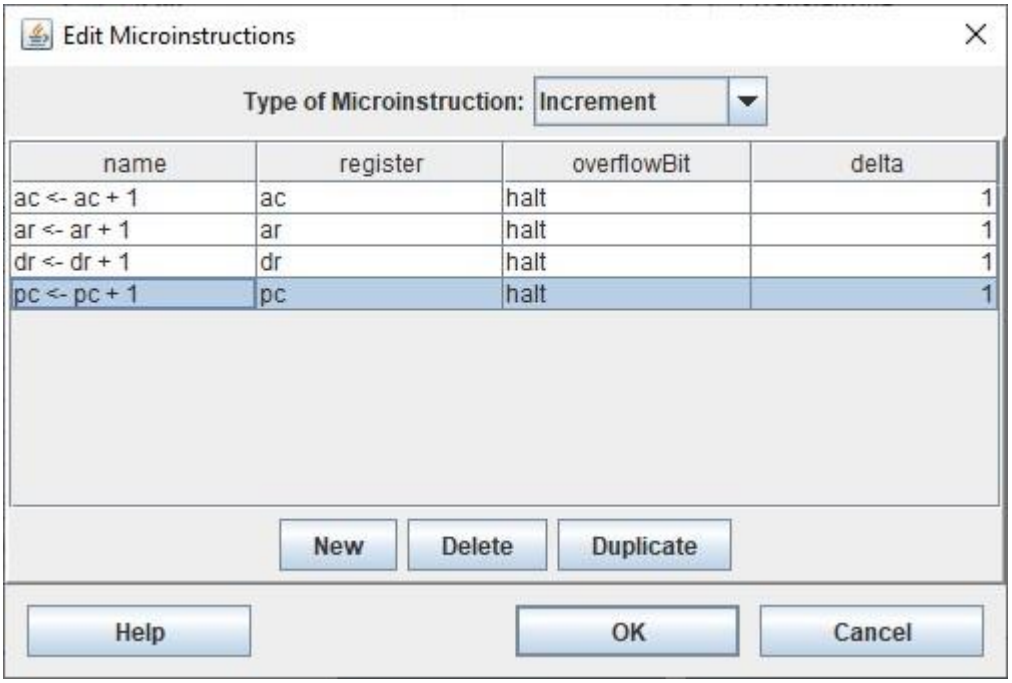


Implementation of SPA

- **<u>If AC (15) != 0</u>** (CPUSim skips a micro-operation when this is true)

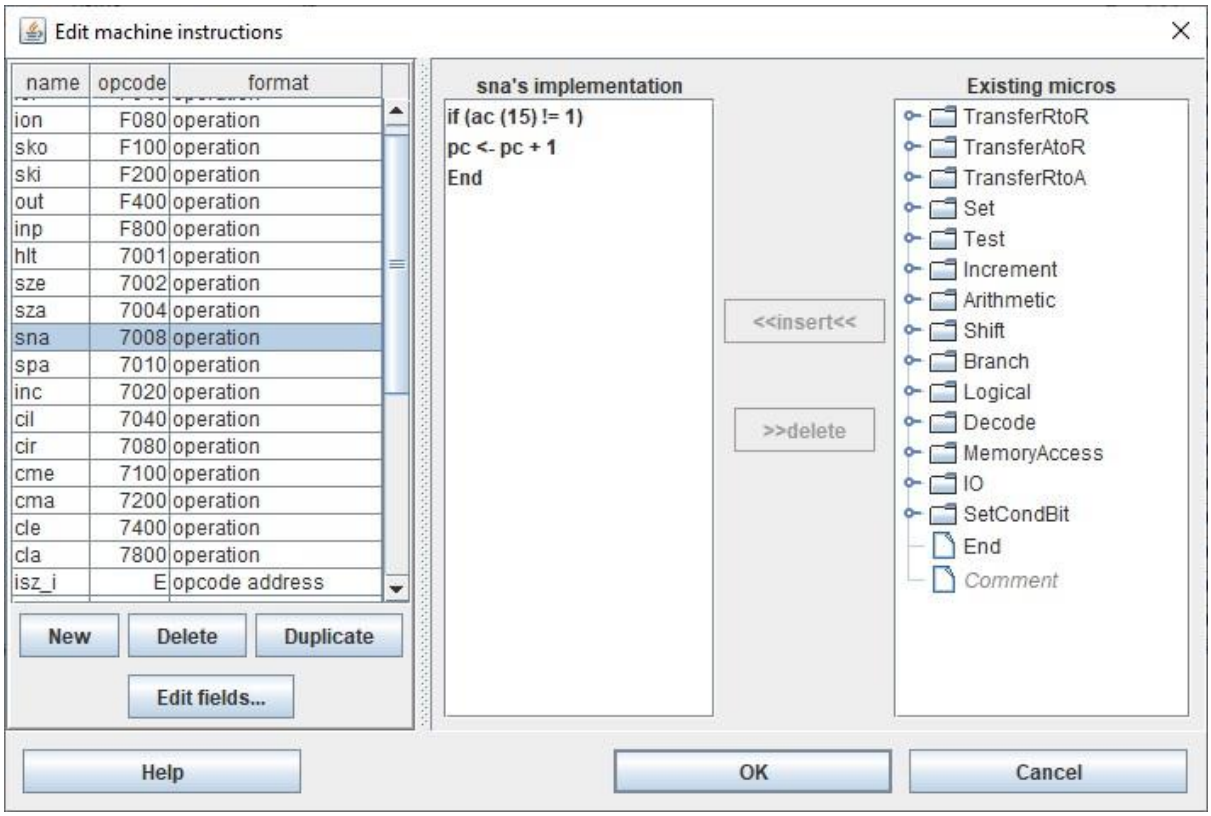


- **<u>PC <- PC + 1</u>**



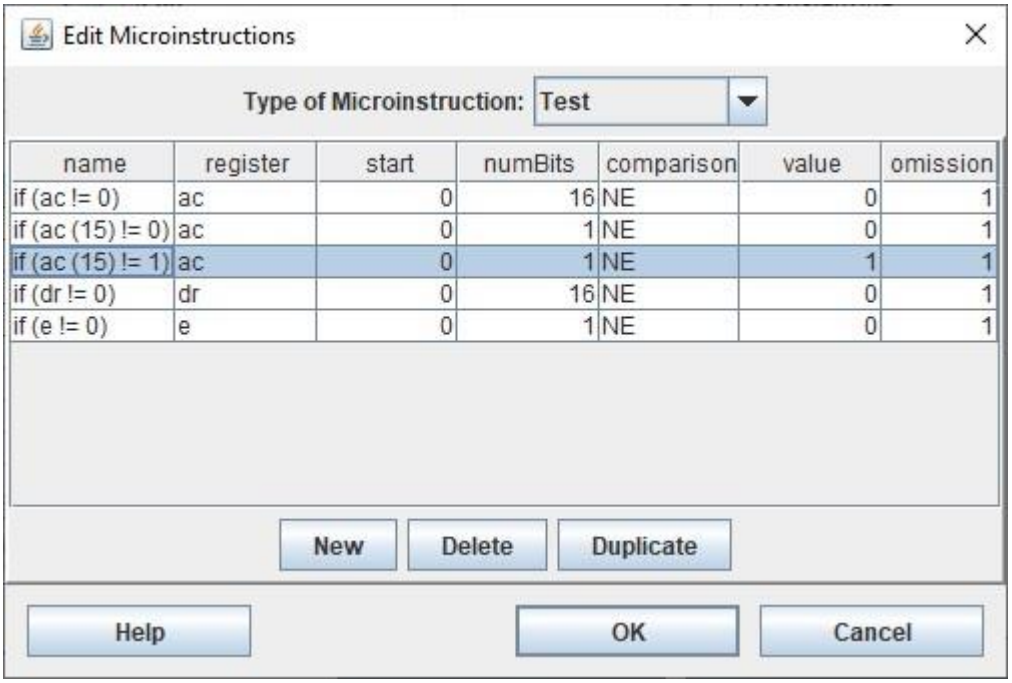
- **SC <- 0**

SNA – Skip If Negative

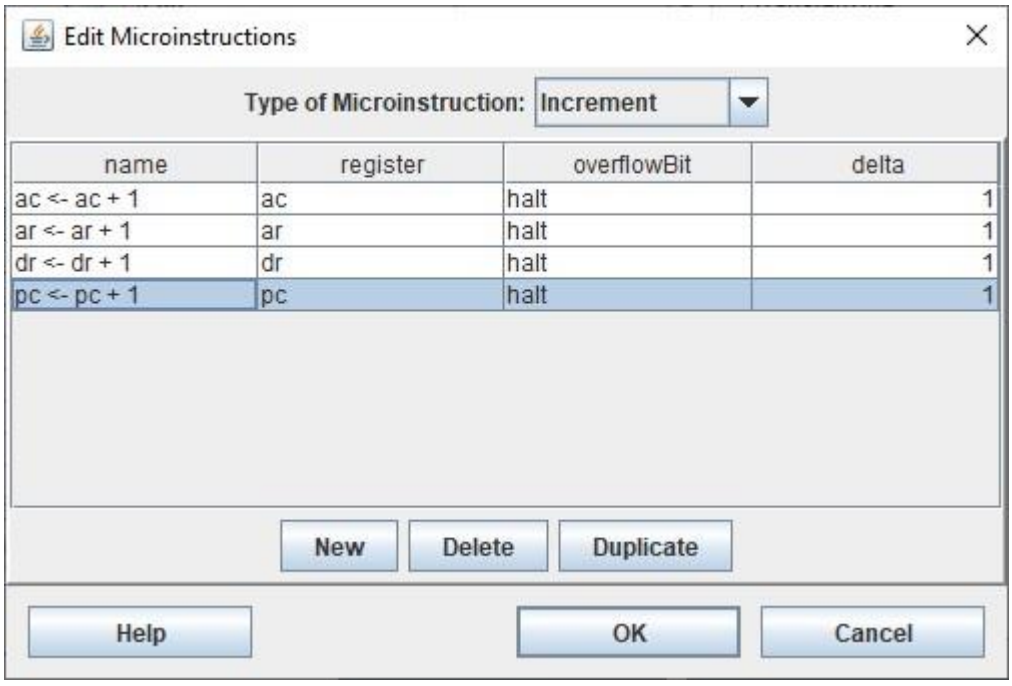


Implementation of SNA

- **If AC (15) != 1** (CPUSim skips a micro-operation when this is true)

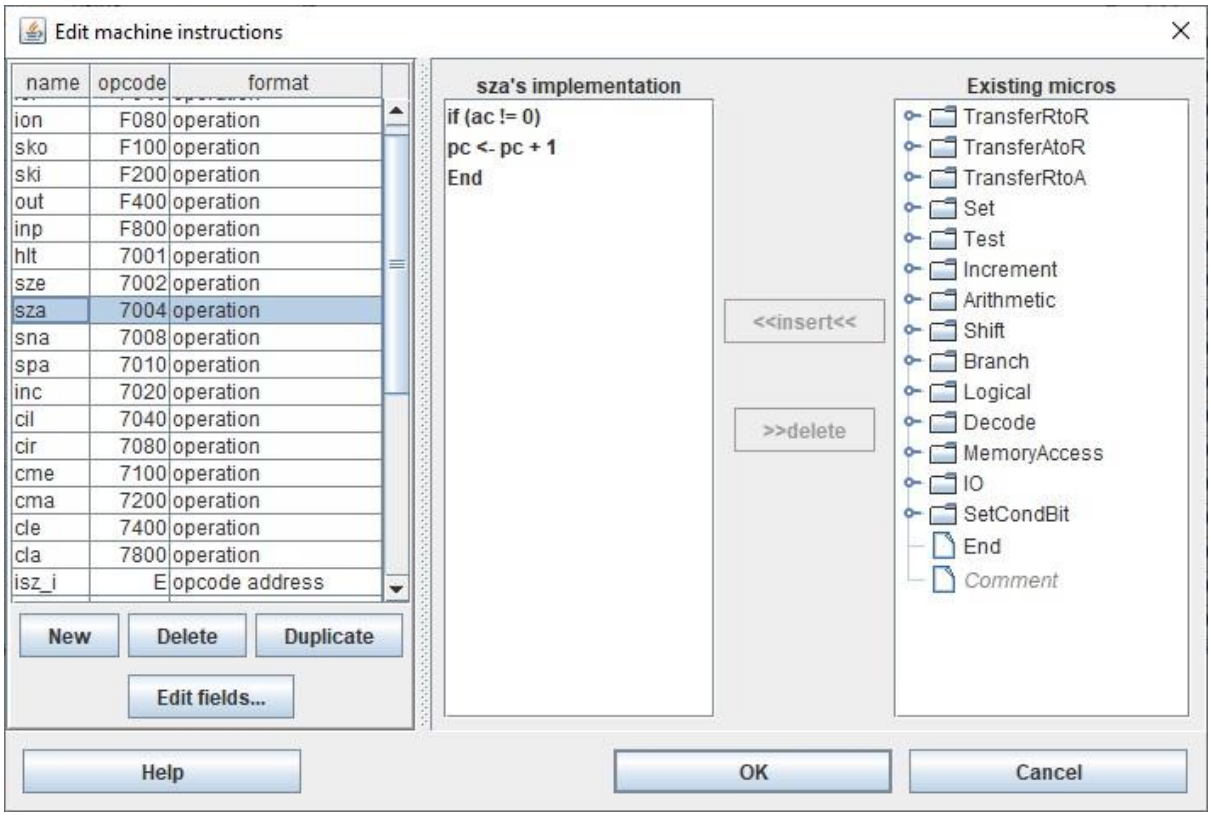

Edit Microinstructions
Type of Microinstruction: Test
name register start numBits comparison value omission
if (ac != 0) ac 0 16 NE 0 1
if (ac (15) != 0) ac 0 1 NE 0 1
if (ac (15) != 1) ac 0 1 NE 1 1
if (dr != 0) dr 0 16 NE 0 1
if (e != 0) e 0 1 NE 0 1
New
Delete
Duplicate
Help
OK
Cancel

- **PC <- PC + 1**


Edit Microinstructions
Type of Microinstruction: Increment
name register overflowBit delta
ac <- ac + 1 ac halt 1
ar <- ar + 1 ar halt 1
dr <- dr + 1 dr halt 1
pc <- pc + 1 pc halt 1
New
Delete
Duplicate
Help
OK
Cancel

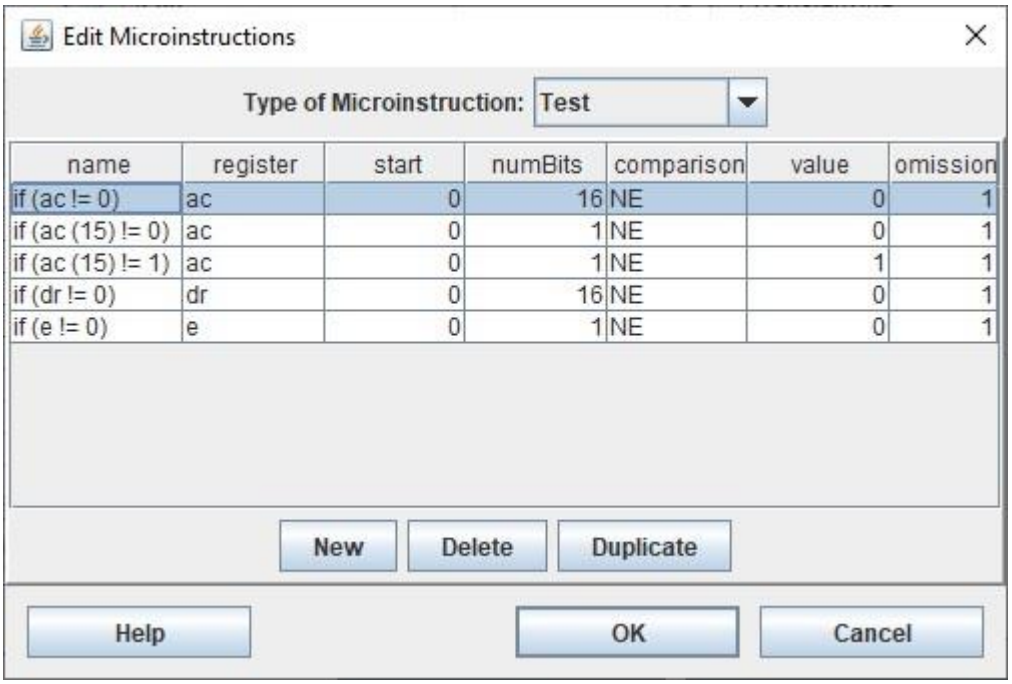
- **SC <- 0**

SZA – Skip If AC Zero

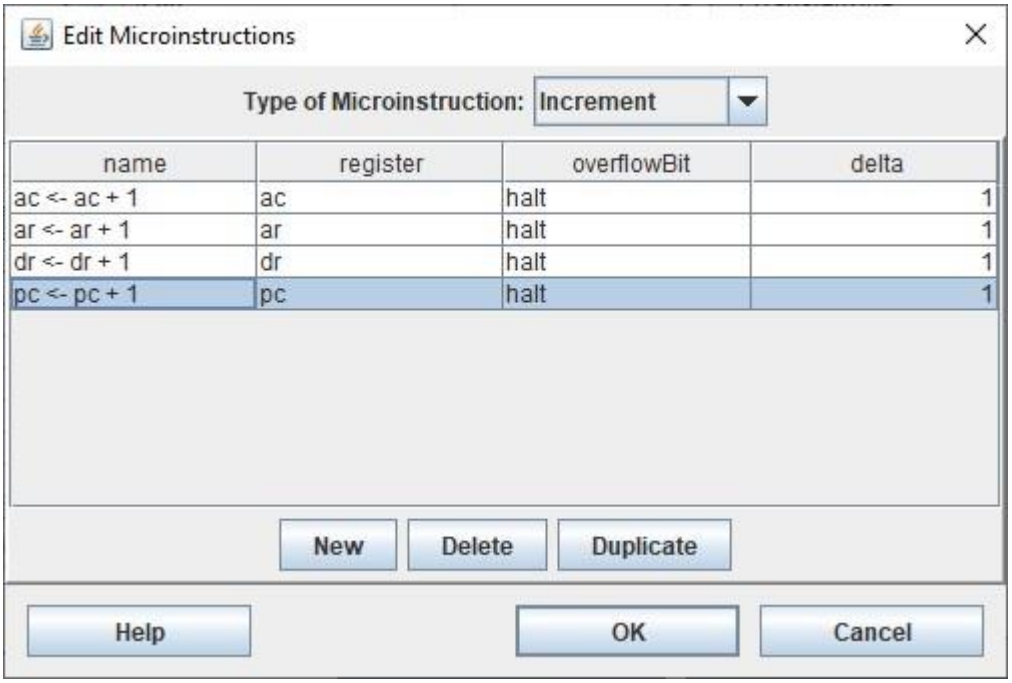


Implementation of SZA

- **<u>If AC != 0</u>** (CPUSim skips a micro-operation when this is true)

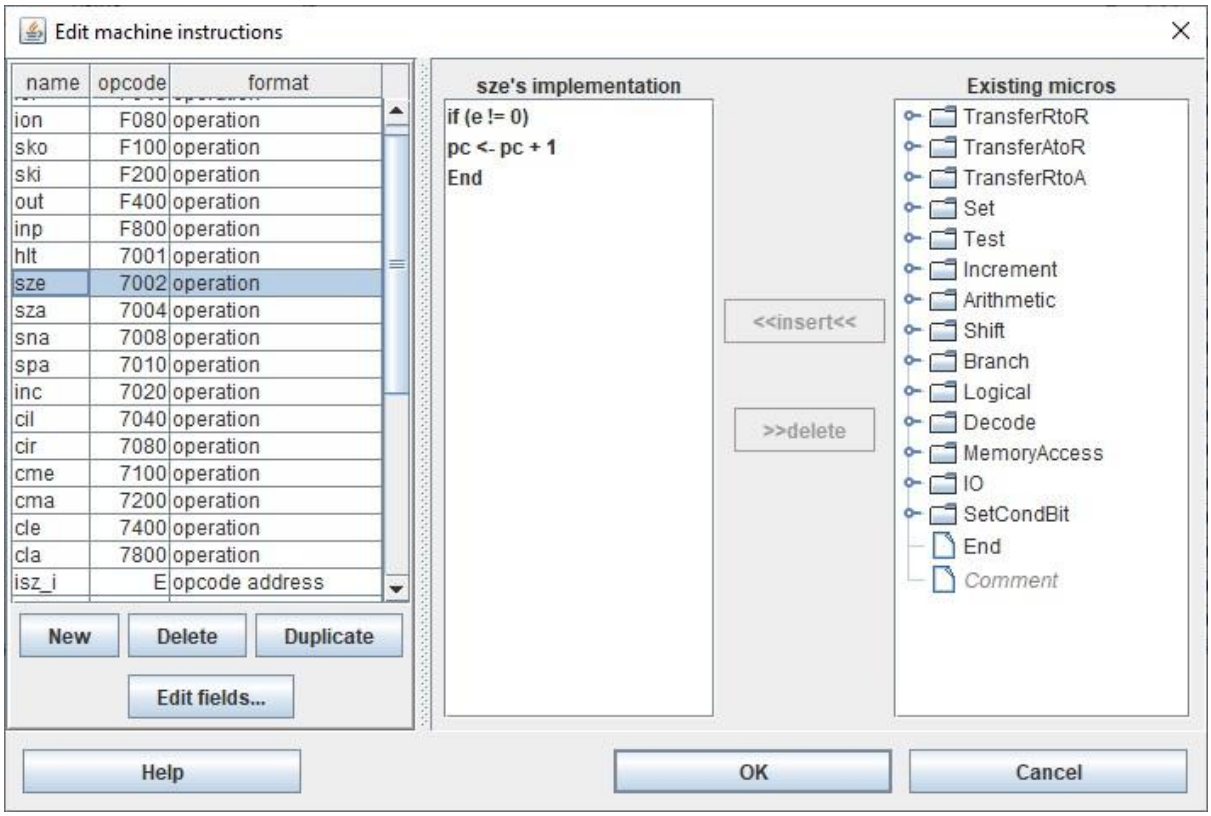


- **<u>PC <- PC + 1</u>**



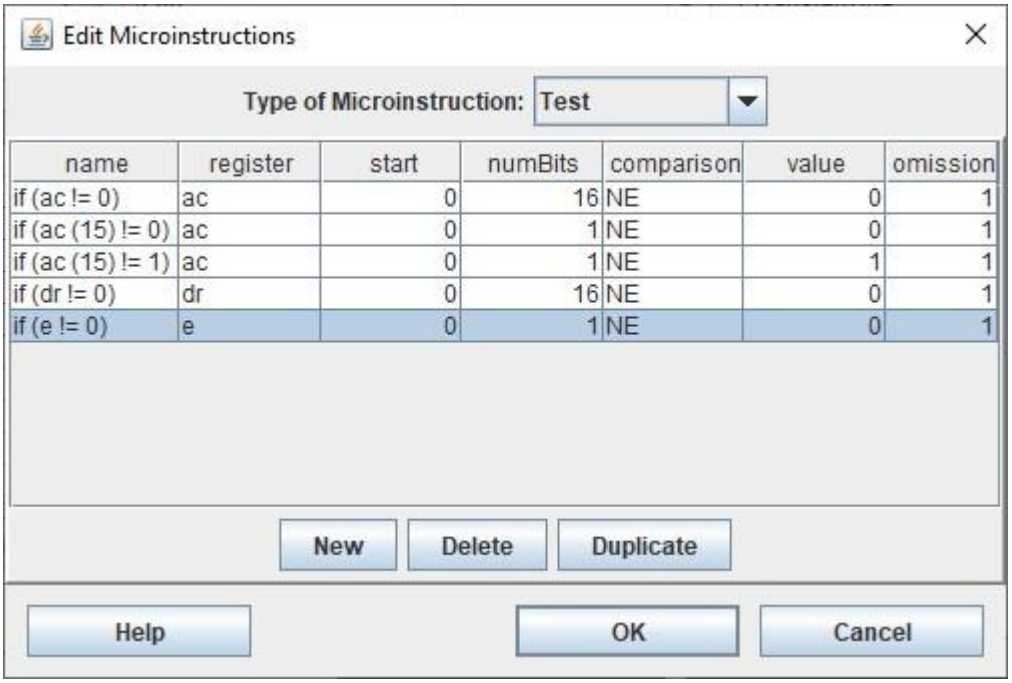
- **SC <- 0**

SZE – Skip If E Zero

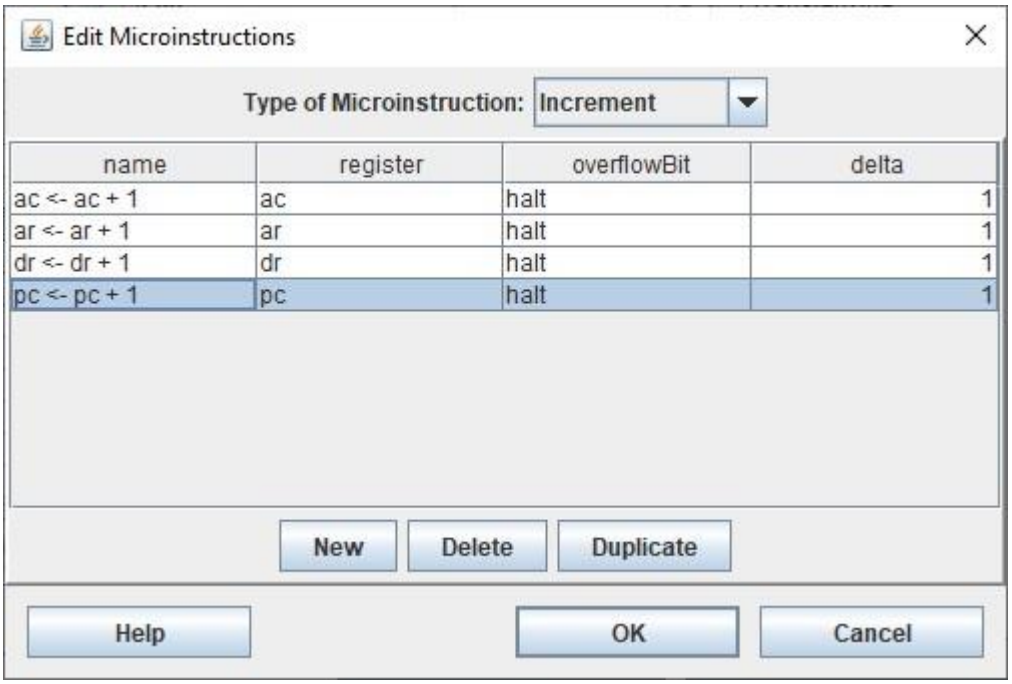


Implementation of SZE

- **If E != 0** (CPUSim skips a micro-operation when this is true)

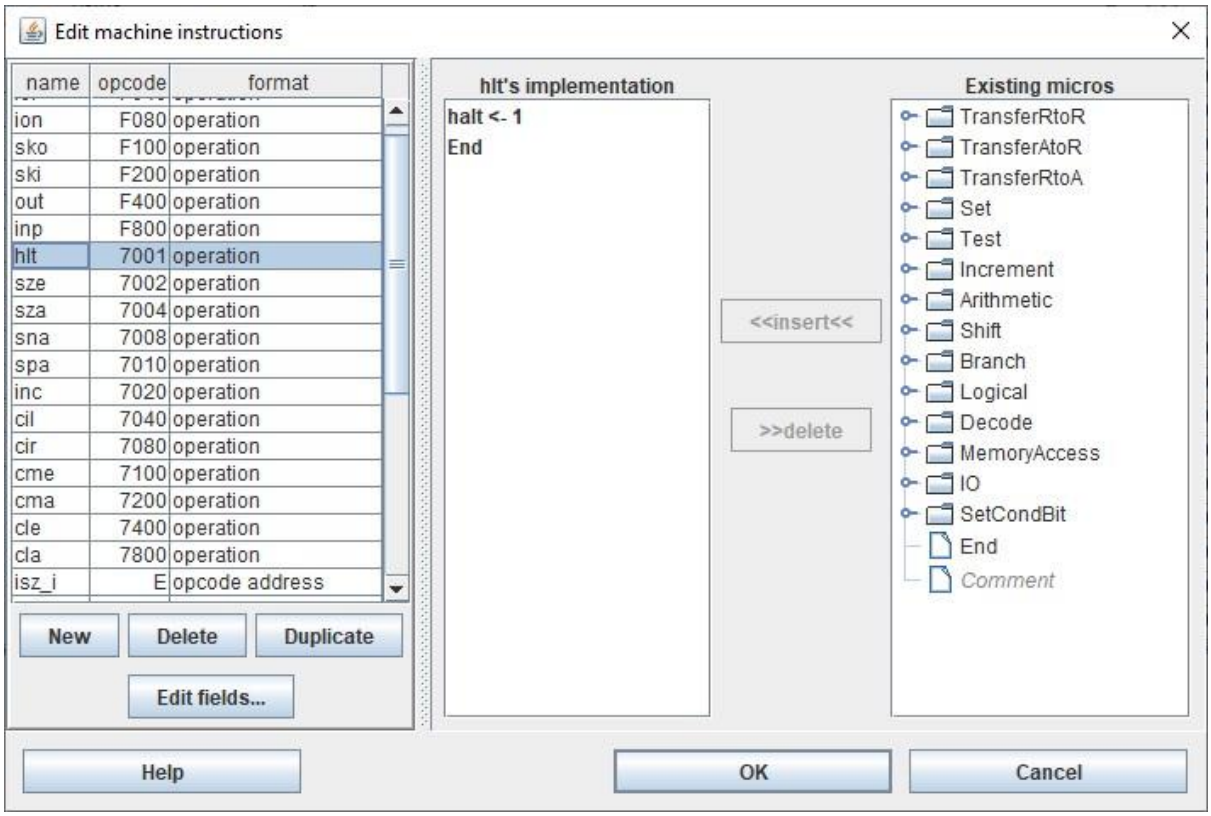

Edit Microinstructions
Type of Microinstruction: Test
name register start numBits comparison value omission
if (ac != 0) ac 0 16 NE 0 1
if (ac (15) != 0) ac 0 1 NE 0 1
if (ac (15) != 1) ac 0 1 NE 1 1
if (dr != 0) dr 0 16 NE 0 1
if (e != 0) e 0 1 NE 0 1
New
Delete
Duplicate
Help
OK
Cancel

- **PC <- PC + 1**


Edit Microinstructions
Type of Microinstruction: Increment
name register overflowBit delta
ac <- ac + 1 ac halt 1
ar <- ar + 1 ar halt 1
dr <- dr + 1 dr halt 1
pc <- pc + 1 pc halt 1
New
Delete
Duplicate
Help
OK
Cancel

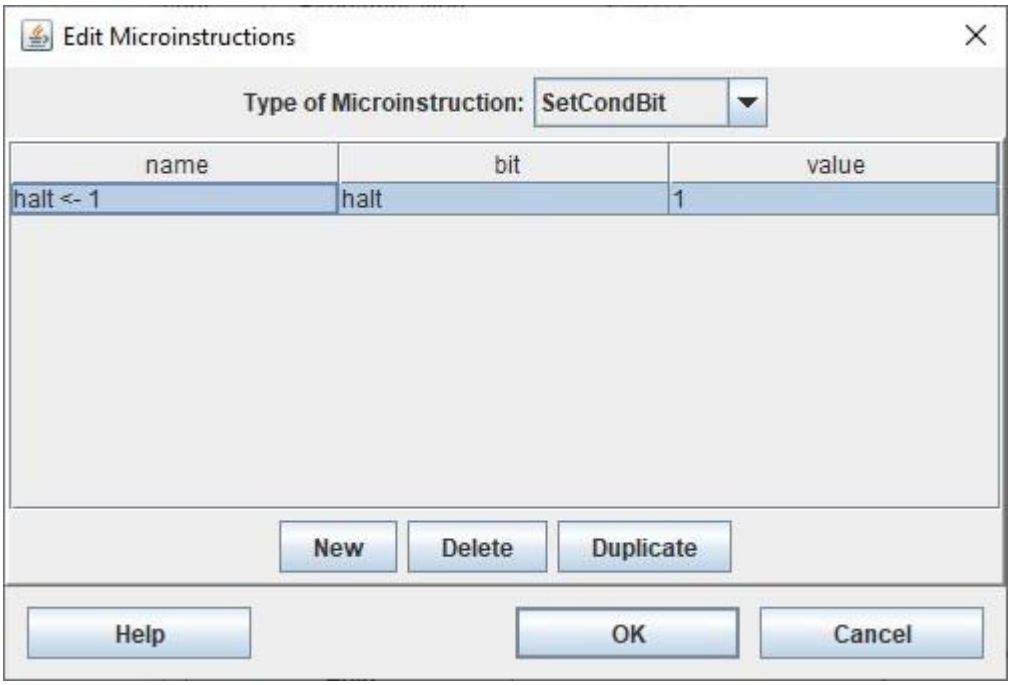
- **SC <- 0**

HLT – Halt Computer



Implementation of HLT

- **HALT <-1 (S <- 1)**



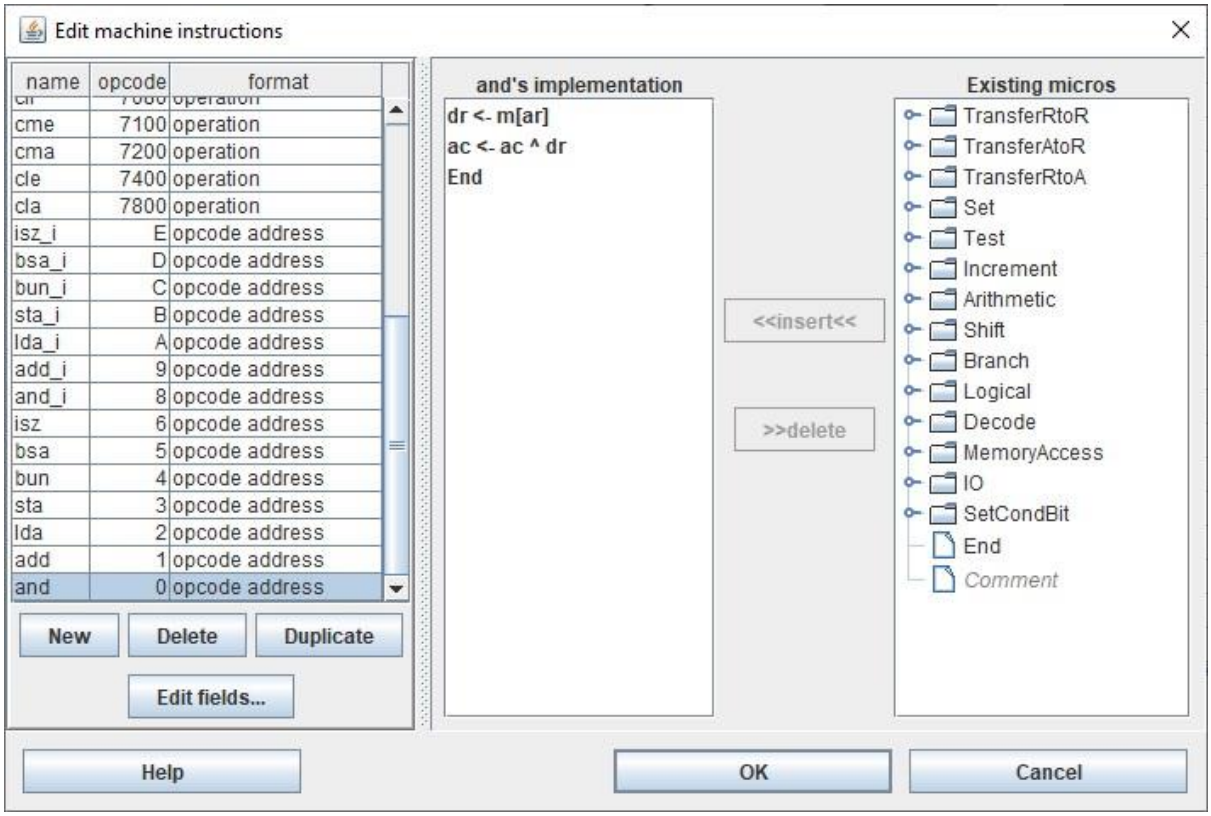
- **SC <- 0**

Memory Reference Instructions

Symbol	Operation decoder	Symbolic description
AND	D_0	$AC \leftarrow AC \wedge M[AR]$
ADD	D_1	$AC \leftarrow AC + M[AR], \quad E \leftarrow C_{out}$
LDA	D_2	$AC \leftarrow M[AR]$
STA	D_3	$M[AR] \leftarrow AC$
BUN	D_4	$PC \leftarrow AR$
BSA	D_5	$M[AR] \leftarrow PC, \quad PC \leftarrow AR + 1$
ISZ	D_6	$M[AR] \leftarrow M[AR] + 1,$ If $M[AR] + 1 = 0$ then $PC \leftarrow PC + 1$

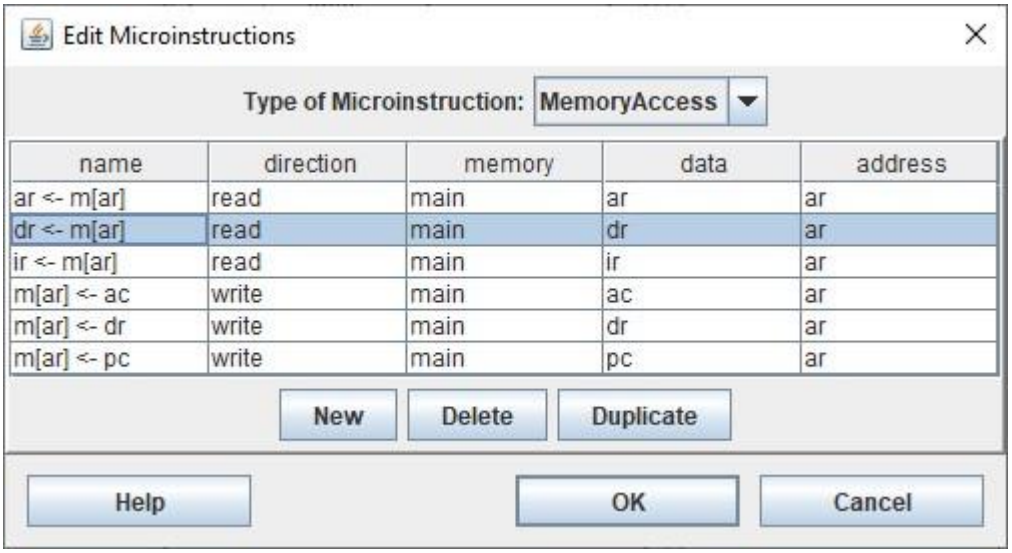
AND to AC

AND – Direct Addressing Mode

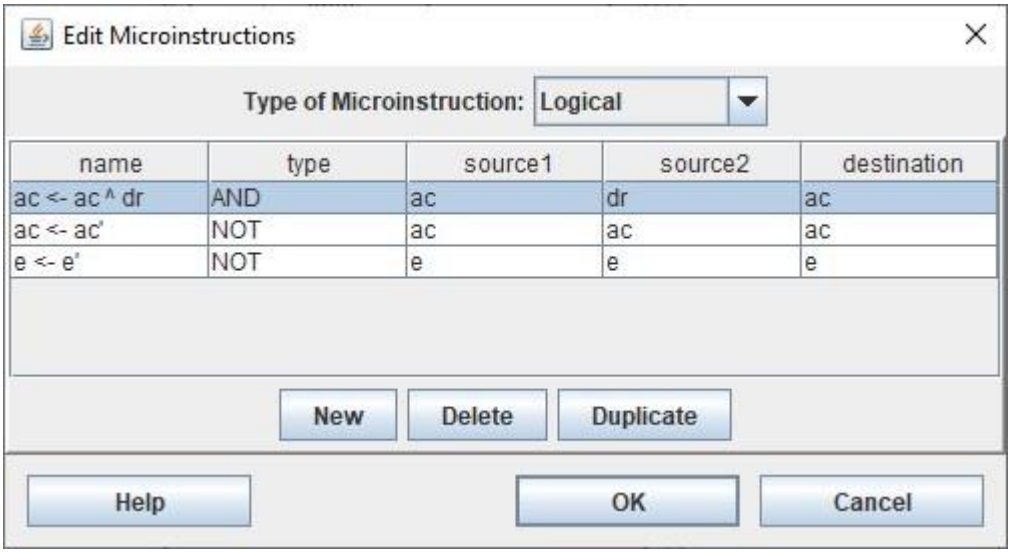


Implementation of AND

- **DR <- M[AR]**

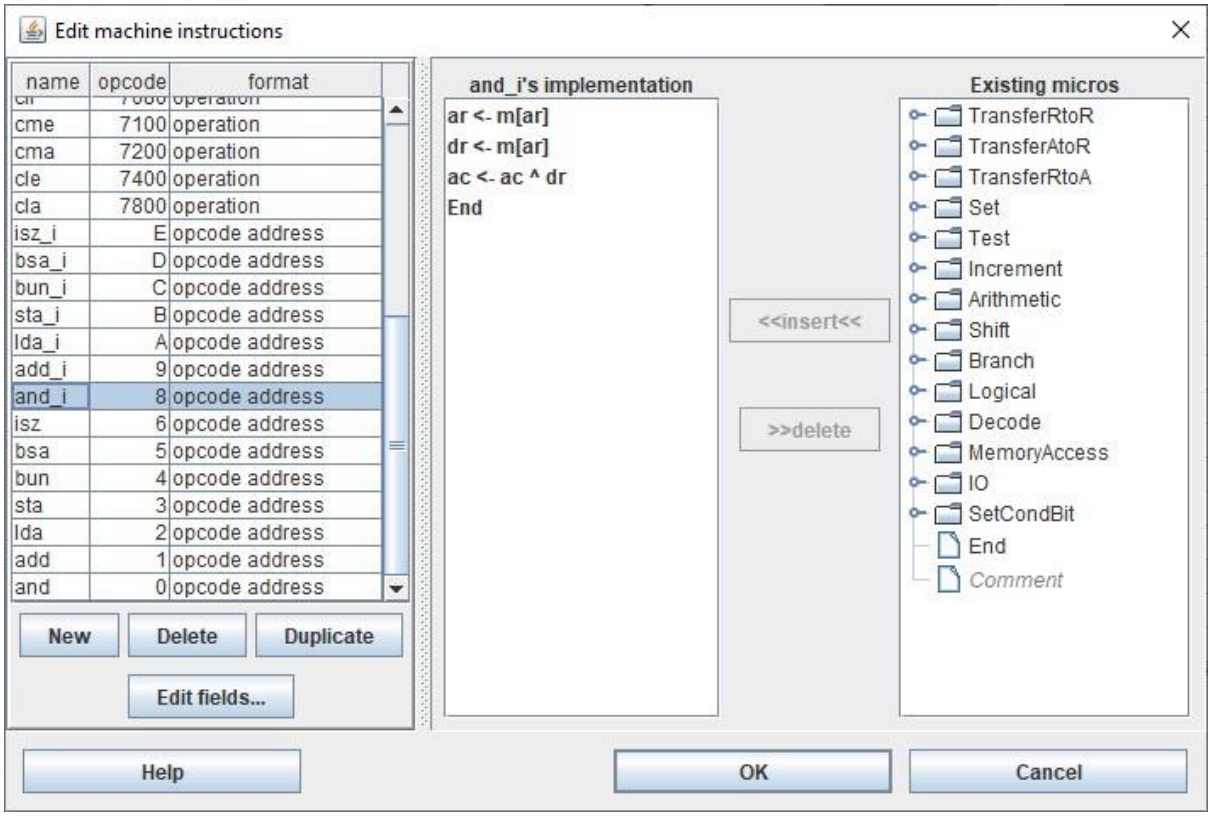


- **AC <- AC ^ DR**



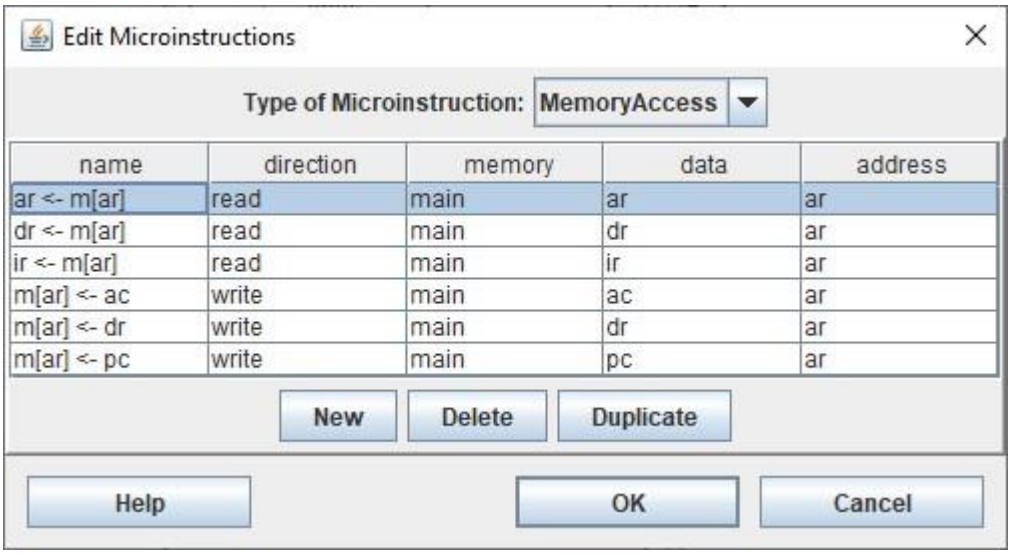
- **SC <- 0**

AND_I – Indirect Addressing Mode

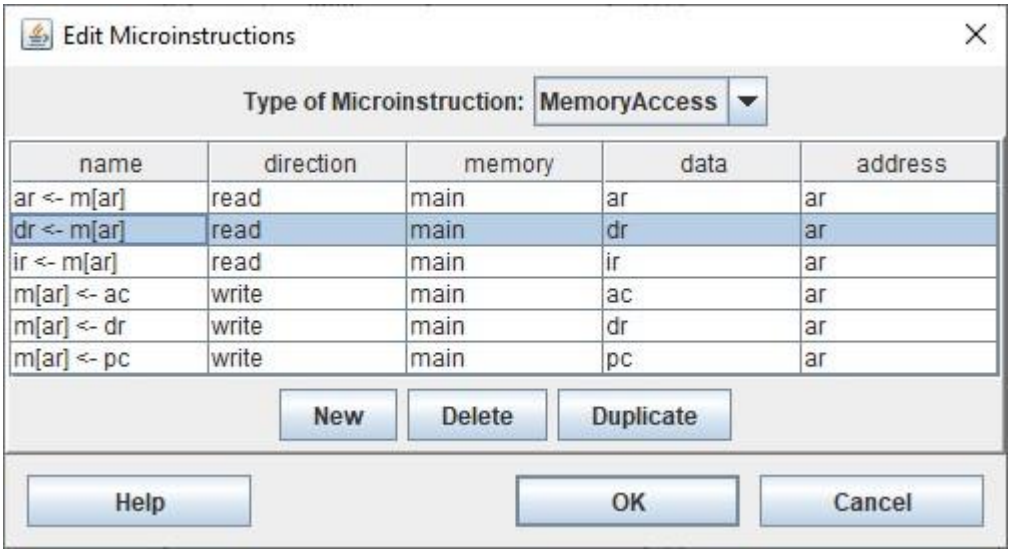


Implementation of AND_I

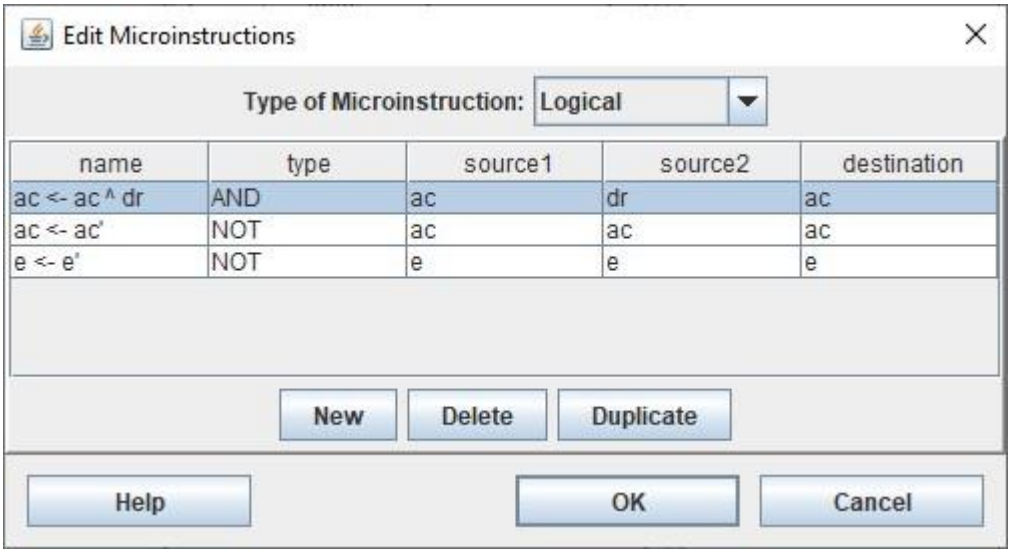
- **AR <- M[AR]**



- **DR <- M[AR]**


Edit Microinstructions
Type of Microinstruction: MemoryAccess
name direction memory data address
ar <- m[ar] read main ar ar
dr <- m[ar] read main dr ar
ir <- m[ar] read main ir ar
m[ar] <- ac write main ac ar
m[ar] <- dr write main dr ar
m[ar] <- pc write main pc ar
New
Delete
Duplicate
Help
OK
Cancel

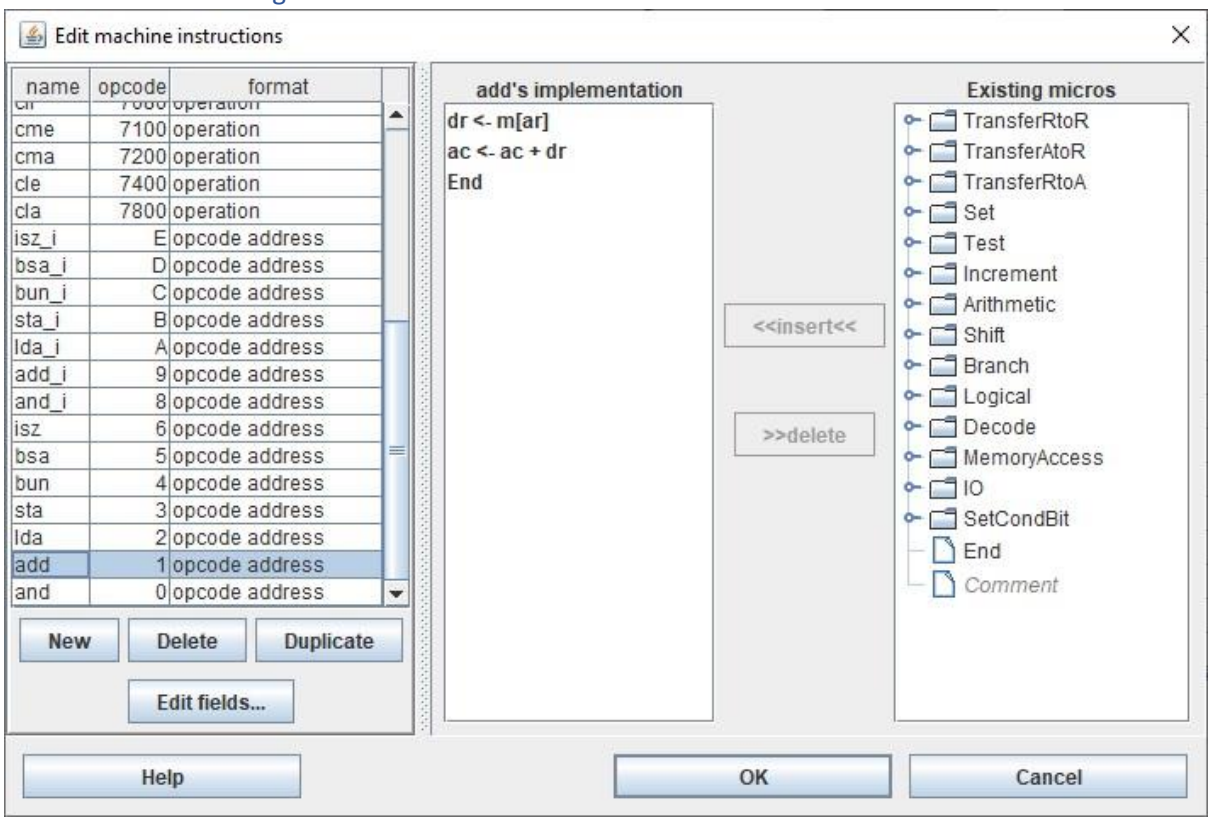
- **AC <- AC ^ DR**


Edit Microinstructions
Type of Microinstruction: Logical
name type source1 source2 destination
ac <- ac ^ dr AND ac dr ac
ac <- ac' NOT ac ac ac
e <- e' NOT e e e
New
Delete
Duplicate
Help
OK
Cancel

- **SC <- 0**

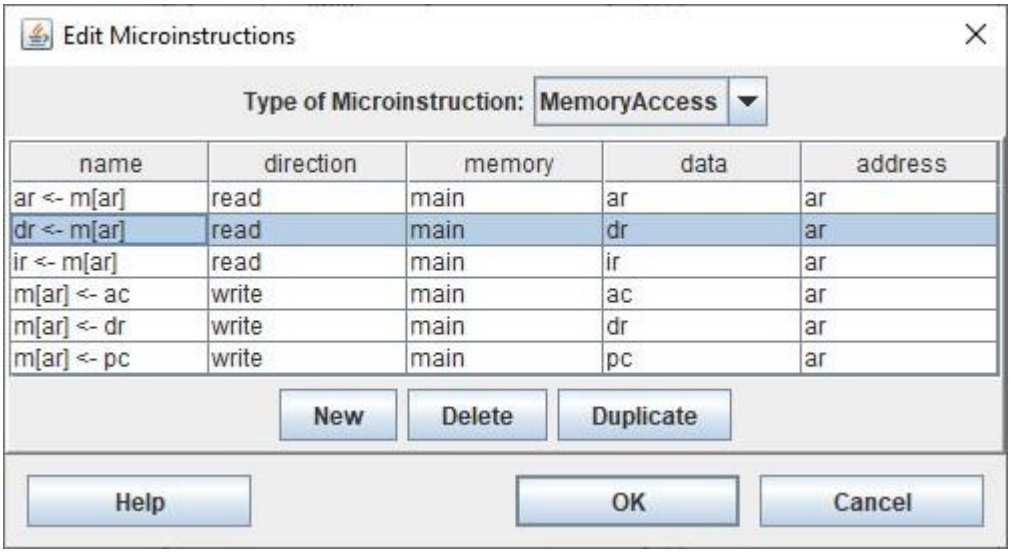
ADD to AC

ADD – Direct Addressing Mode

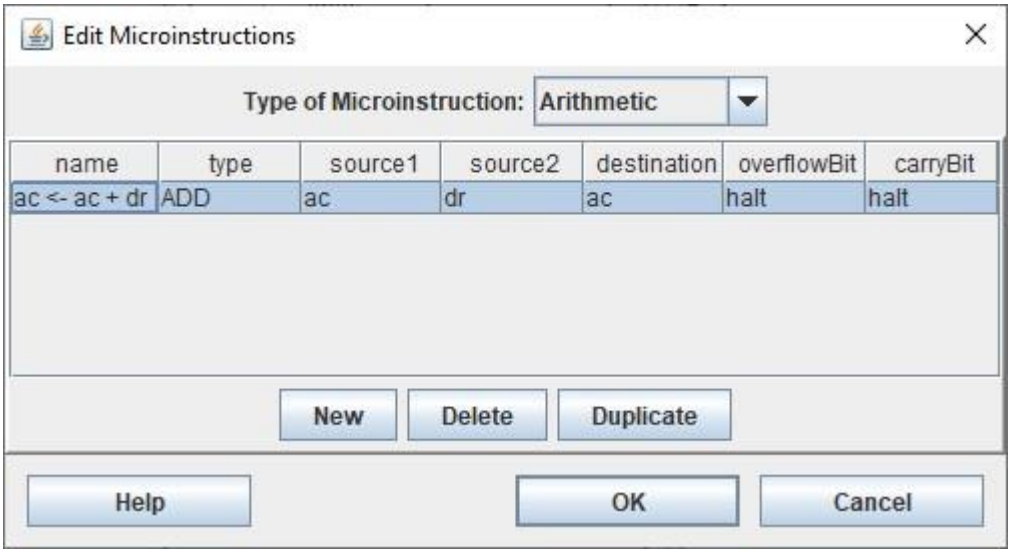


Implementation of ADD

- **DR <- M[AR]**

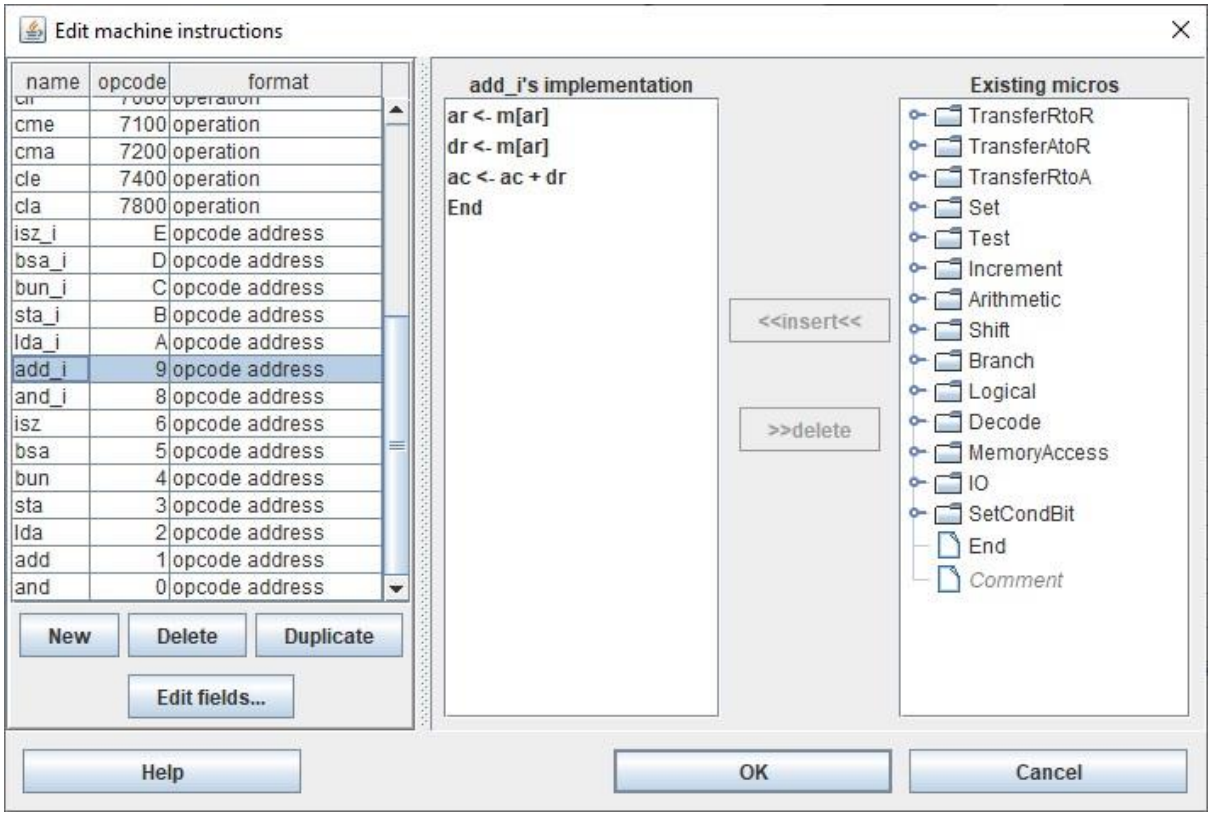


- **AC <- AC + DR**



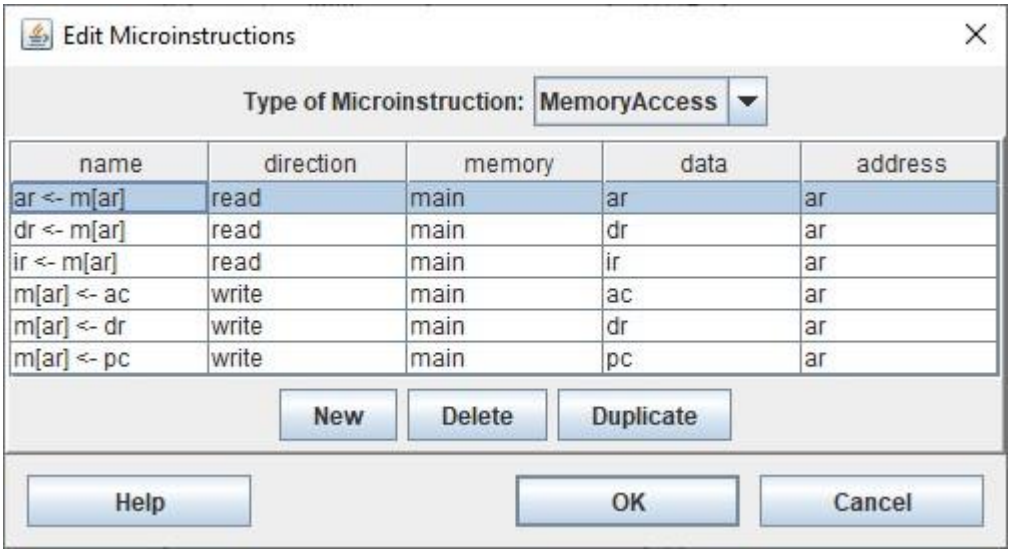
- **SC <- 0**

ADD_I – Indirect Addressing Mode

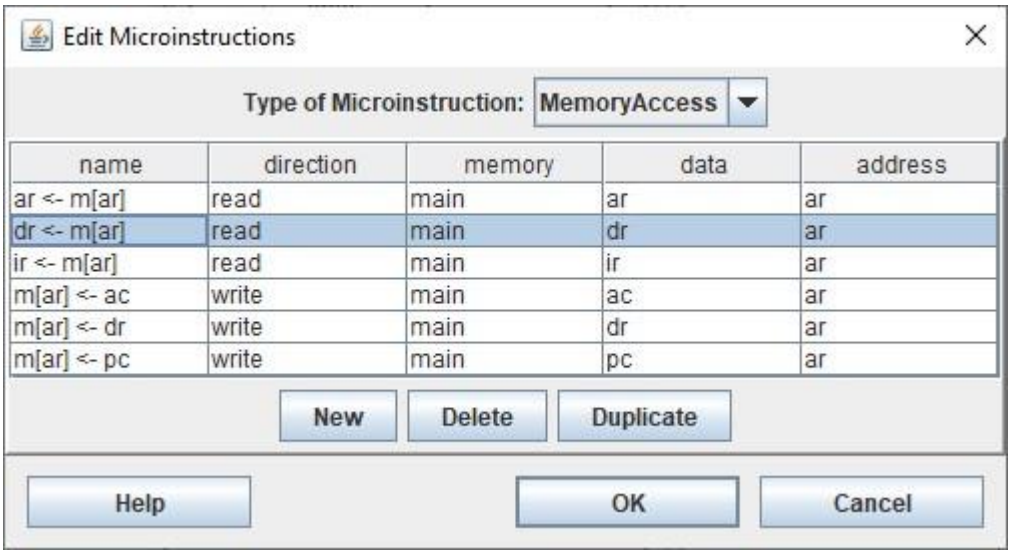


Implementation of ADD_I

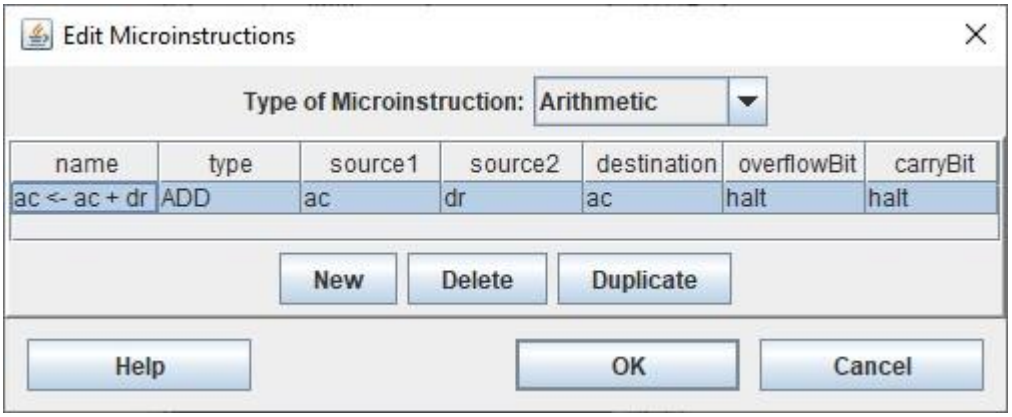
- **AR <- M[AR]**



- **DR <- M[AR]**



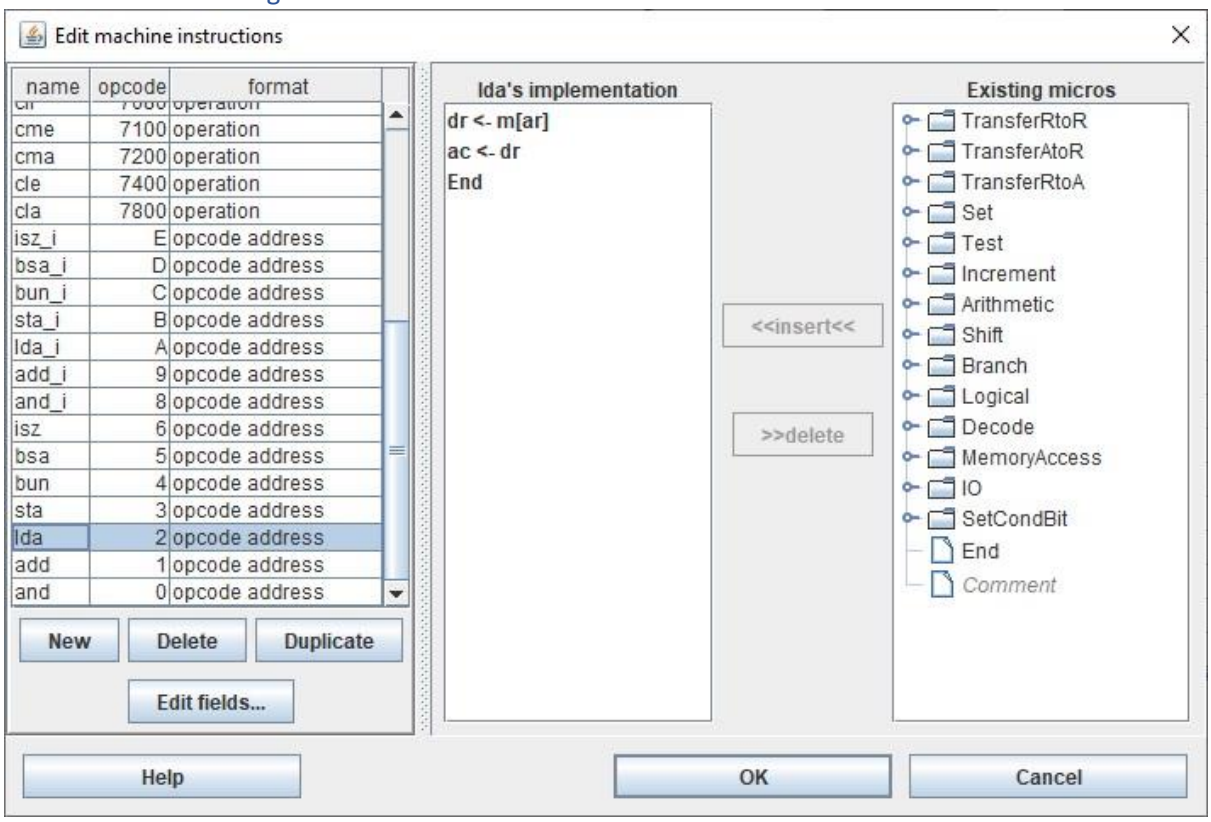
- **AC <- AC + DR**



- **SC <- 0**

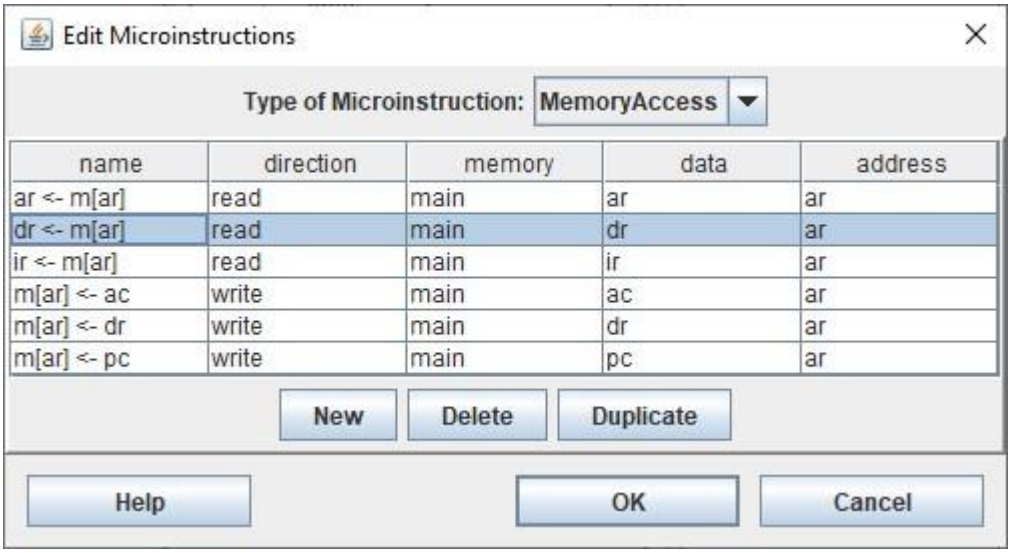
LOAD to AC

LDA – Direct Addressing Mode

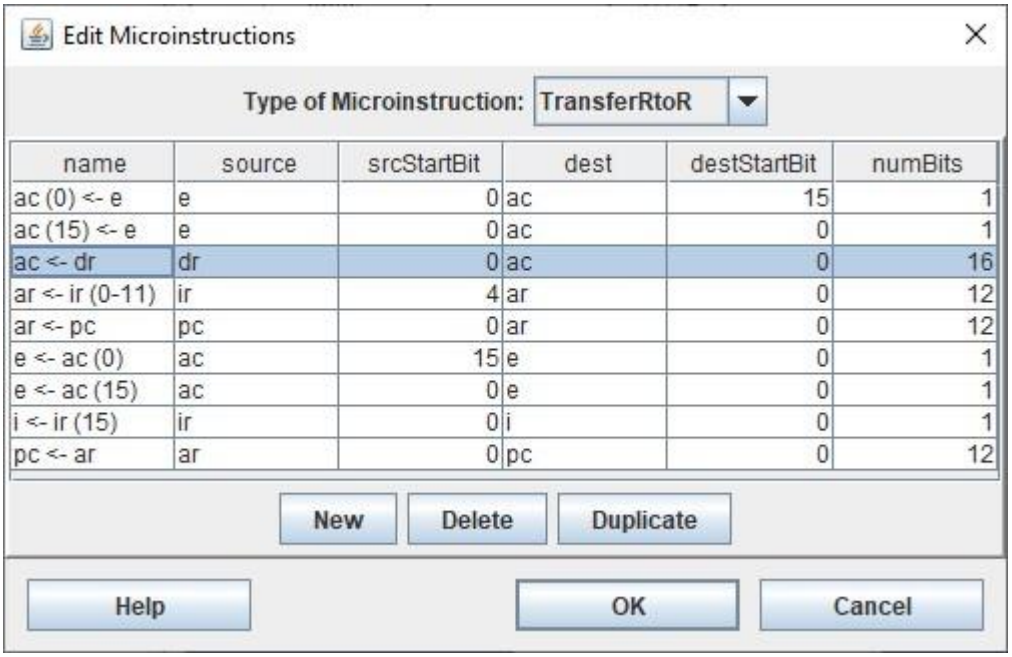


Implementation of LDA

- **DR <- M[AR]**

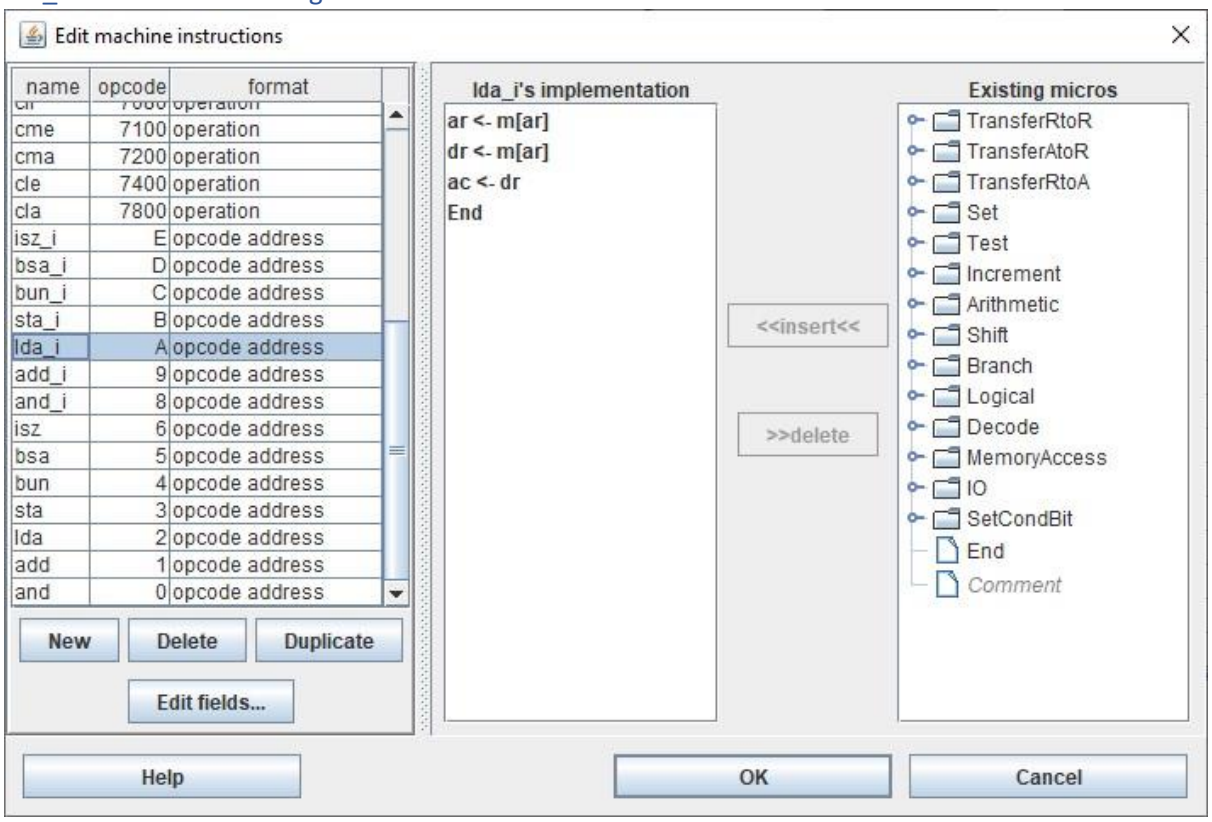


- **AC <- DR**



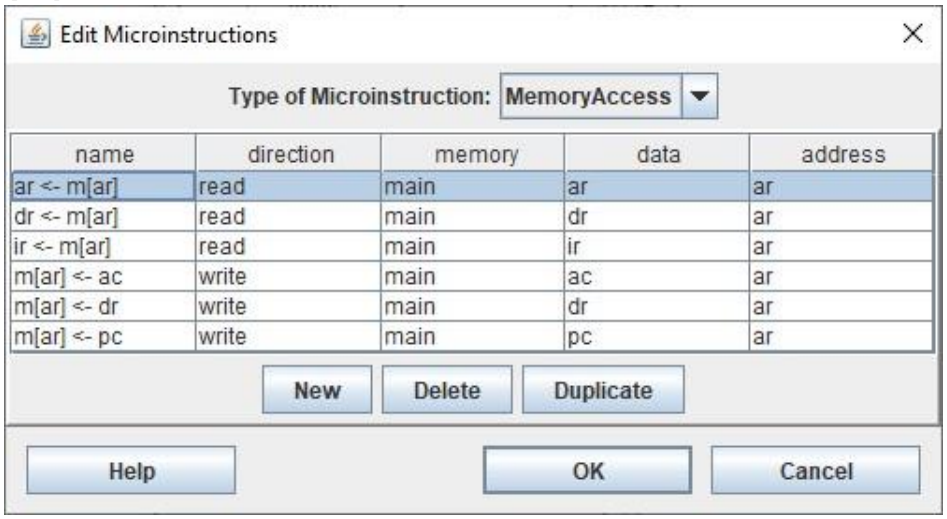
- **SC <- 0**

LDA_I – Indirect Addressing Mode

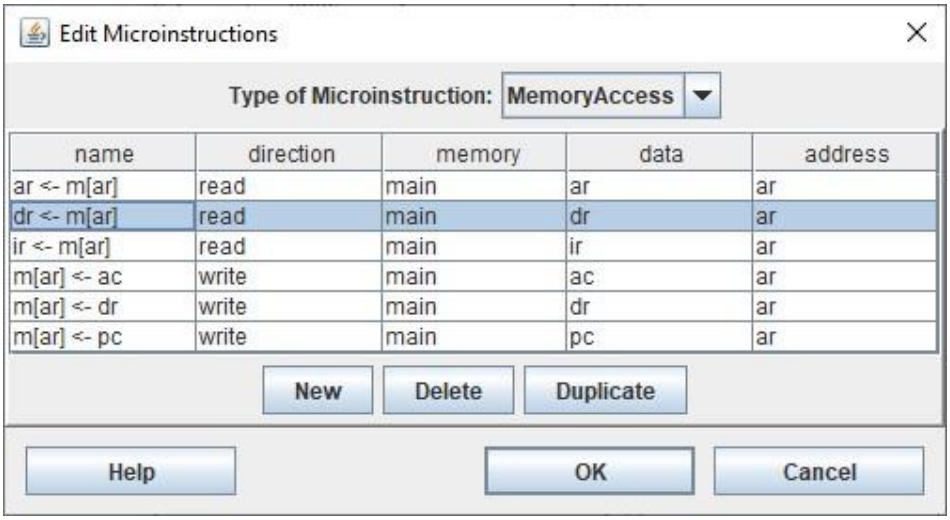


Implementation of LDA_I

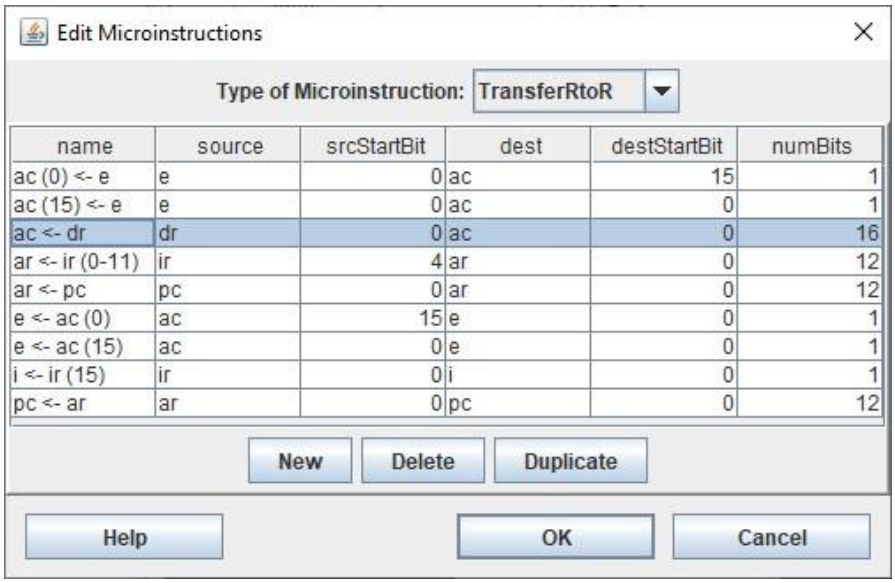
- **AR <- M[AR]**



- **DR <- M[AR]**



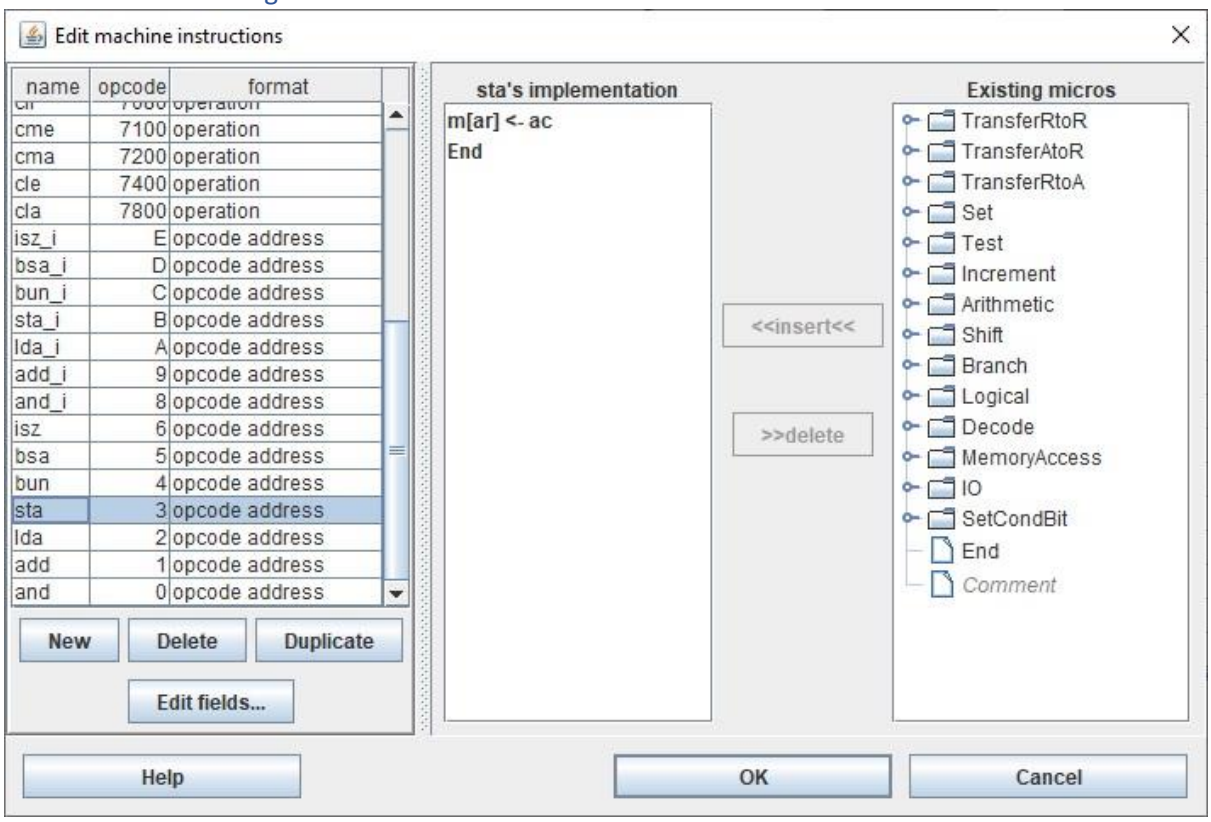
- **AC <- DR**



- **SC <- 0**

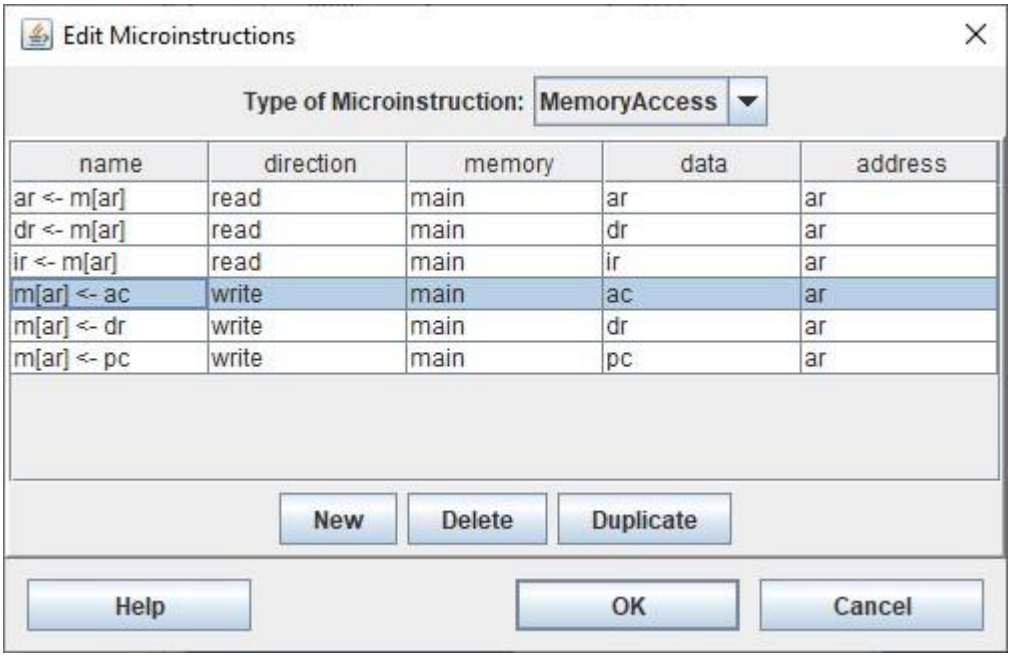
STORE AC

STA – Direct Addressing Mode



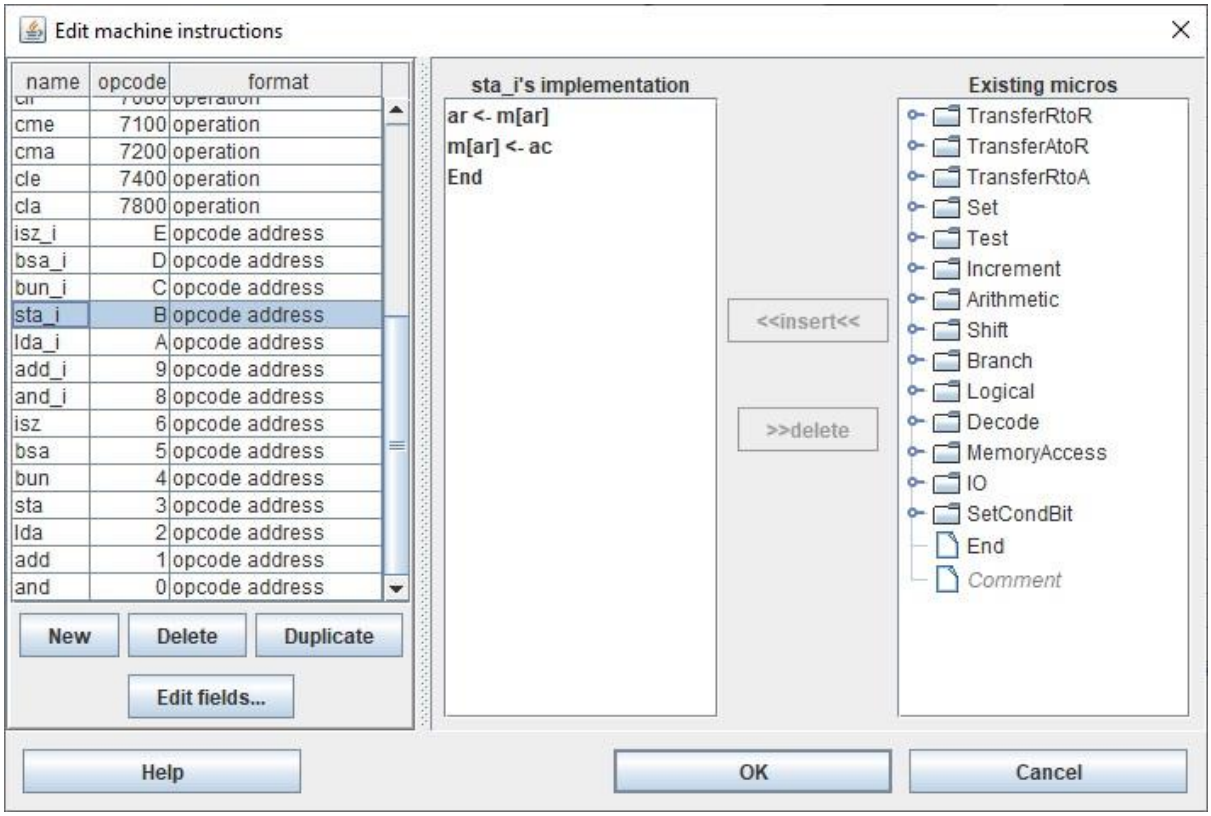
Implementation of STA

- **M[AR] <- AC**



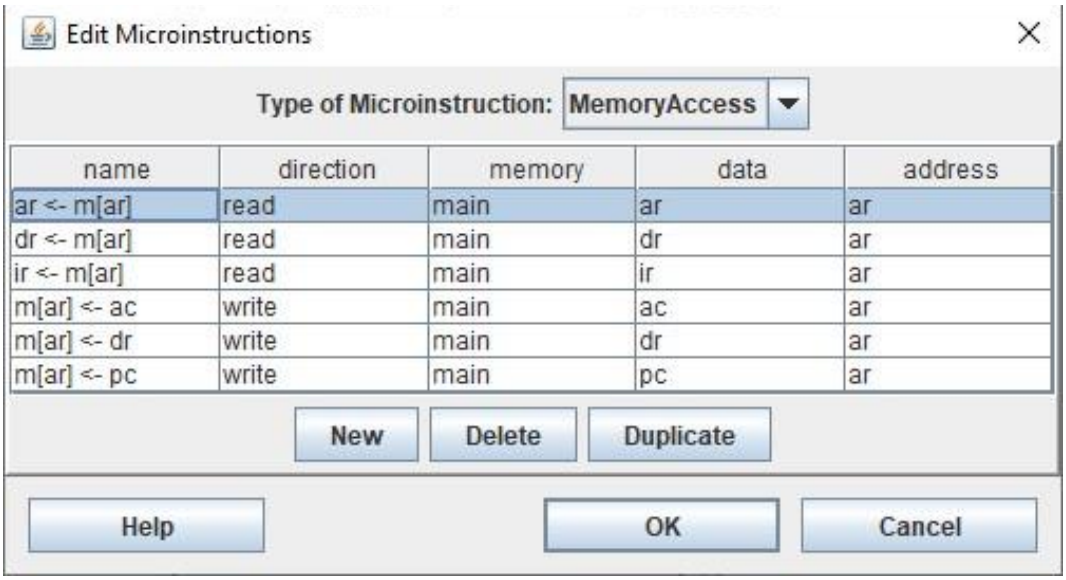
- **SC <- 0**

STA_I – Indirect Addressing Mode

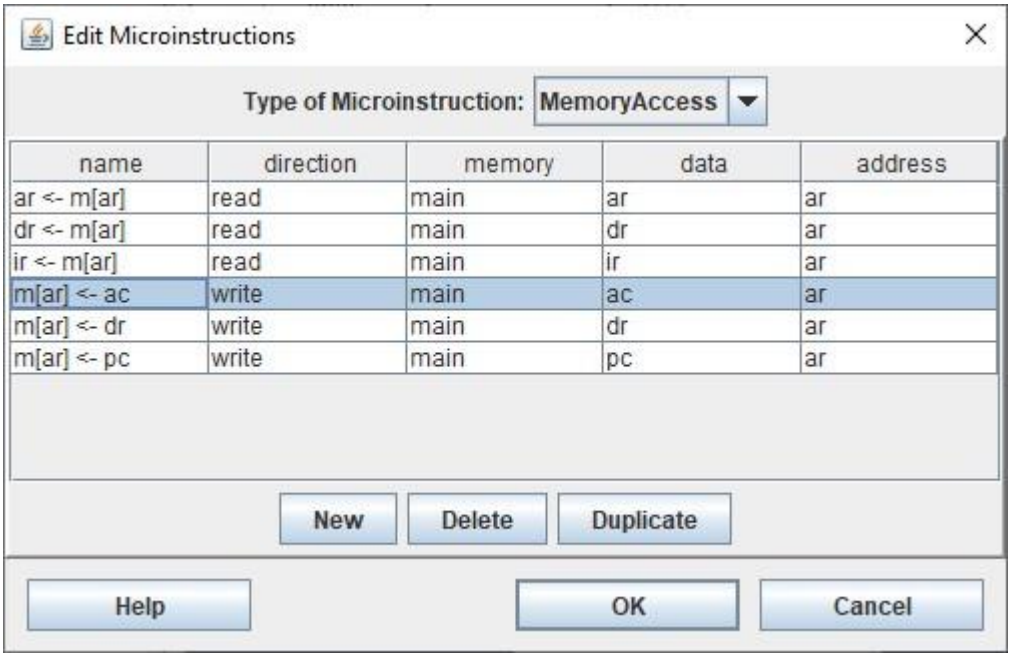


Implementation of STA_I

- **AR <- M[AR]**



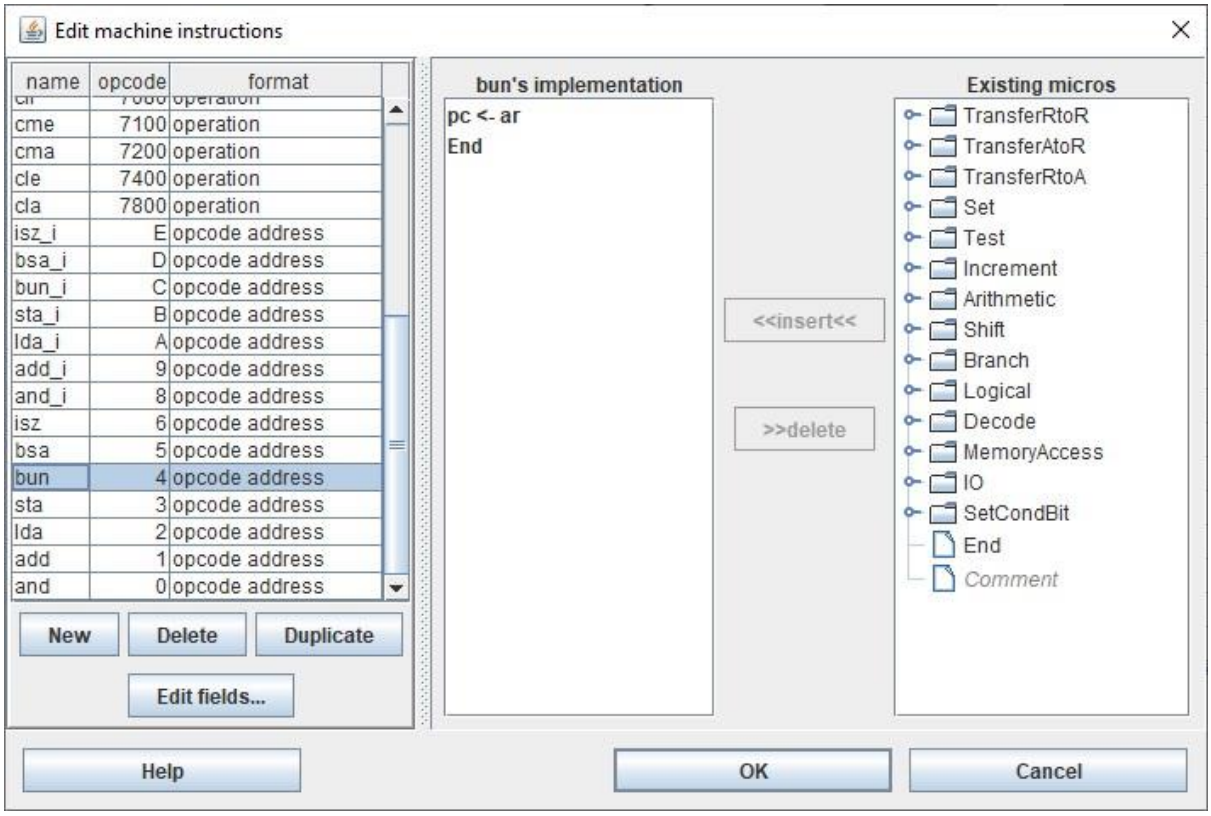
- **M[AR] <- AC**



- **SC <- 0**

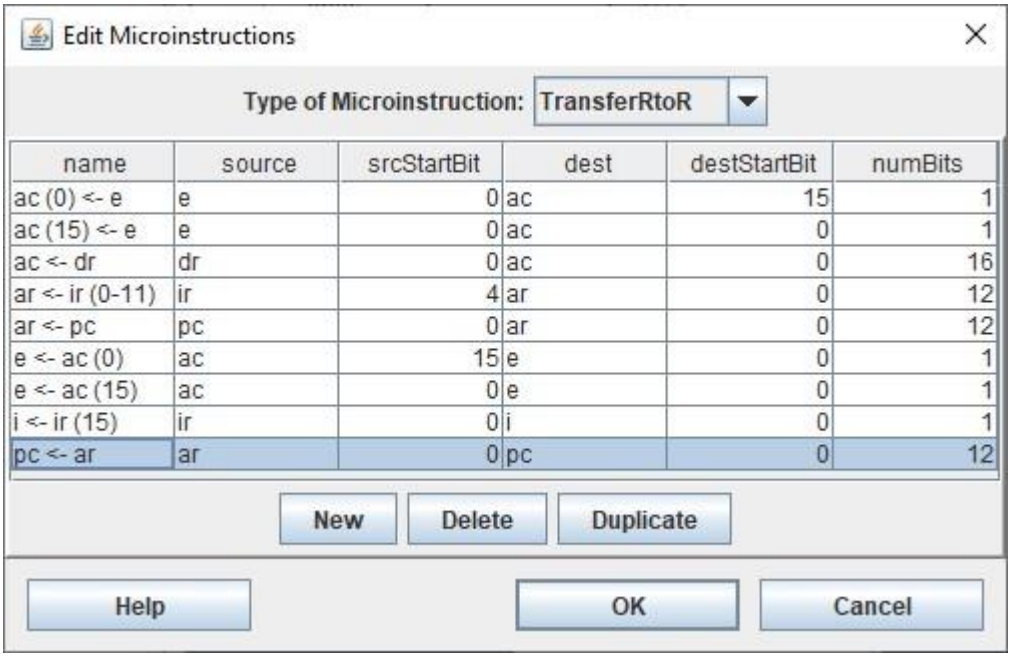
Branch UNconditionally

BUN – Direct Addressing Mode



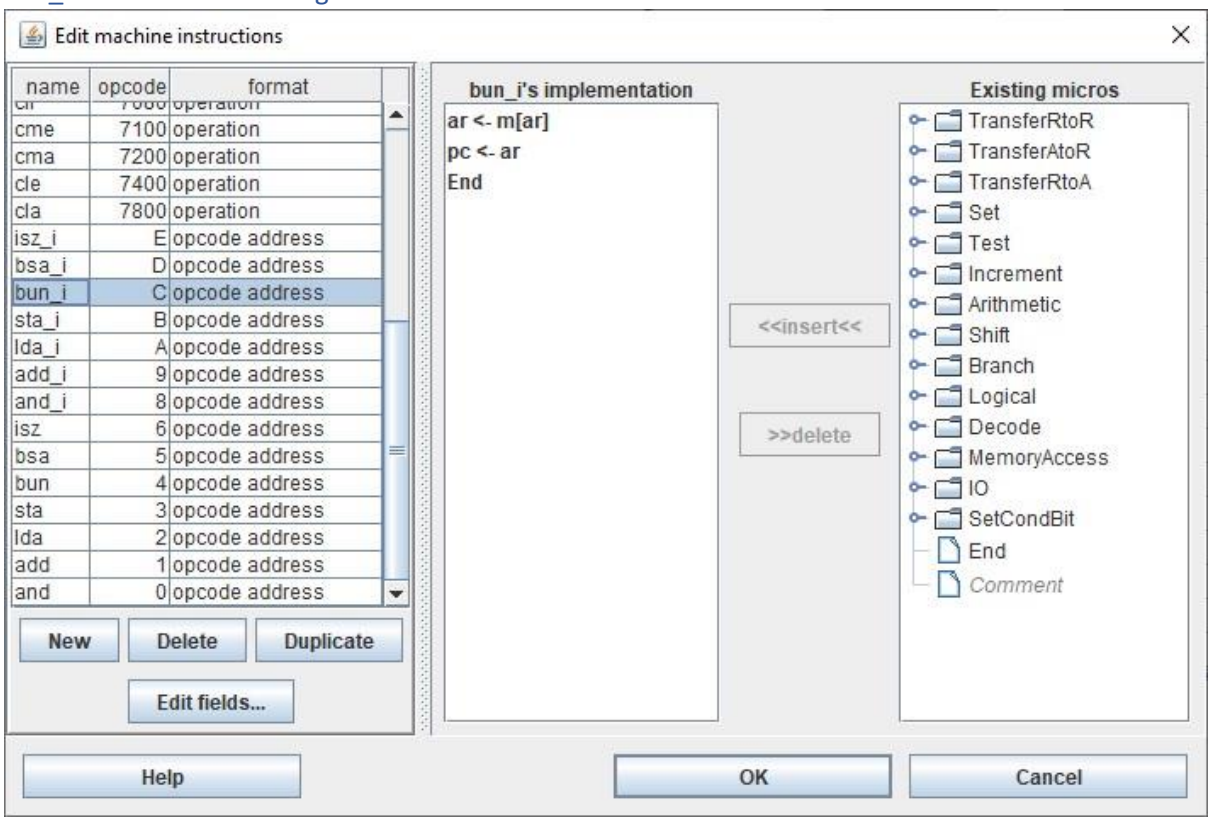
Implementation of BUN

- **PC <- AR**



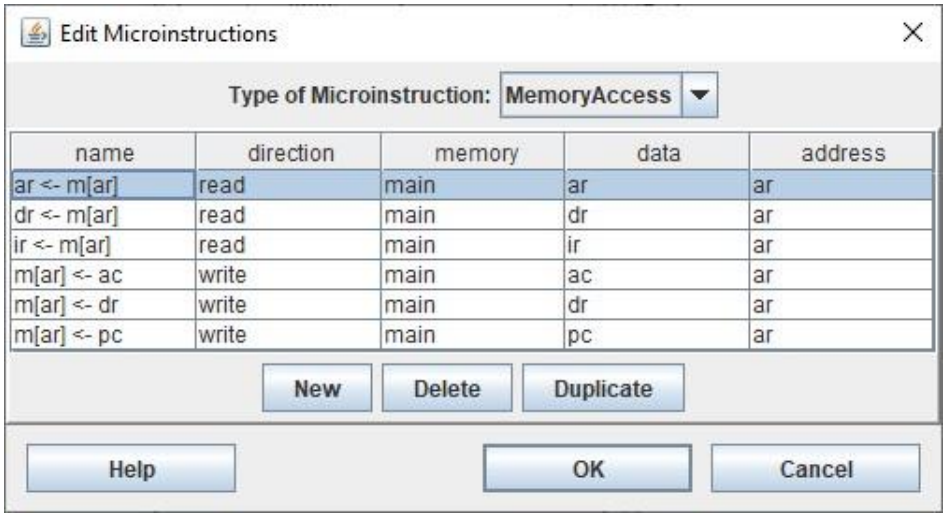
- **SC <- 0**

BUN_I – Indirect Addressing Mode

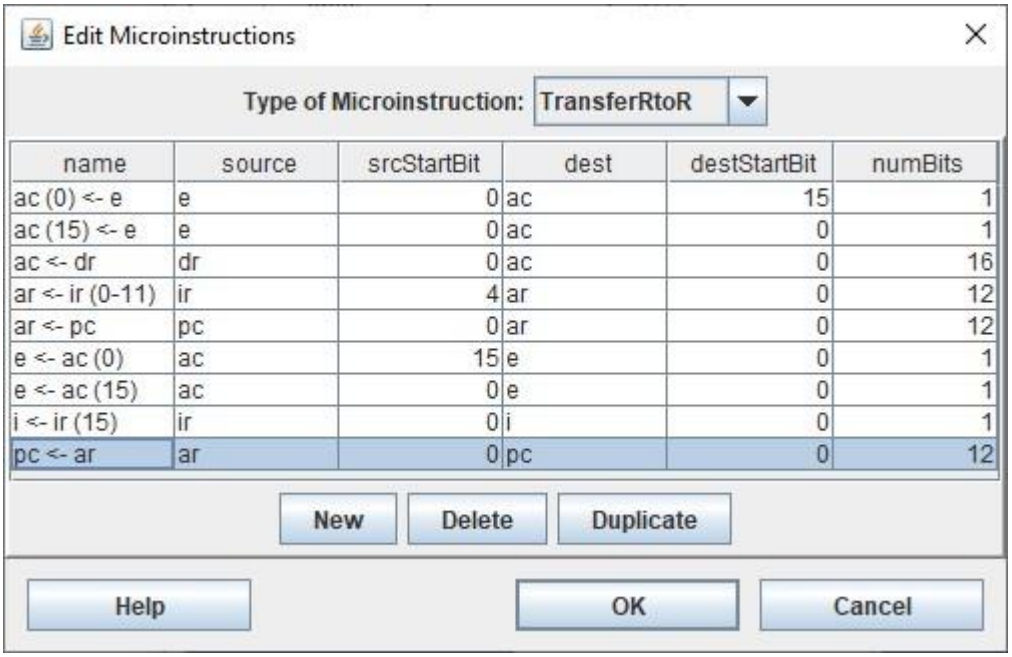


Implementation of BUN_I

- **AR <- M[AR]**


Edit Microinstructions
Type of Microinstruction: MemoryAccess
name direction memory data address
ar <- m[ar] read main ar ar
dr <- m[ar] read main dr ar
ir <- m[ar] read main ir ar
m[ar] <- ac write main ac ar
m[ar] <- dr write main dr ar
m[ar] <- pc write main pc ar
New
Delete
Duplicate
Help
OK
Cancel

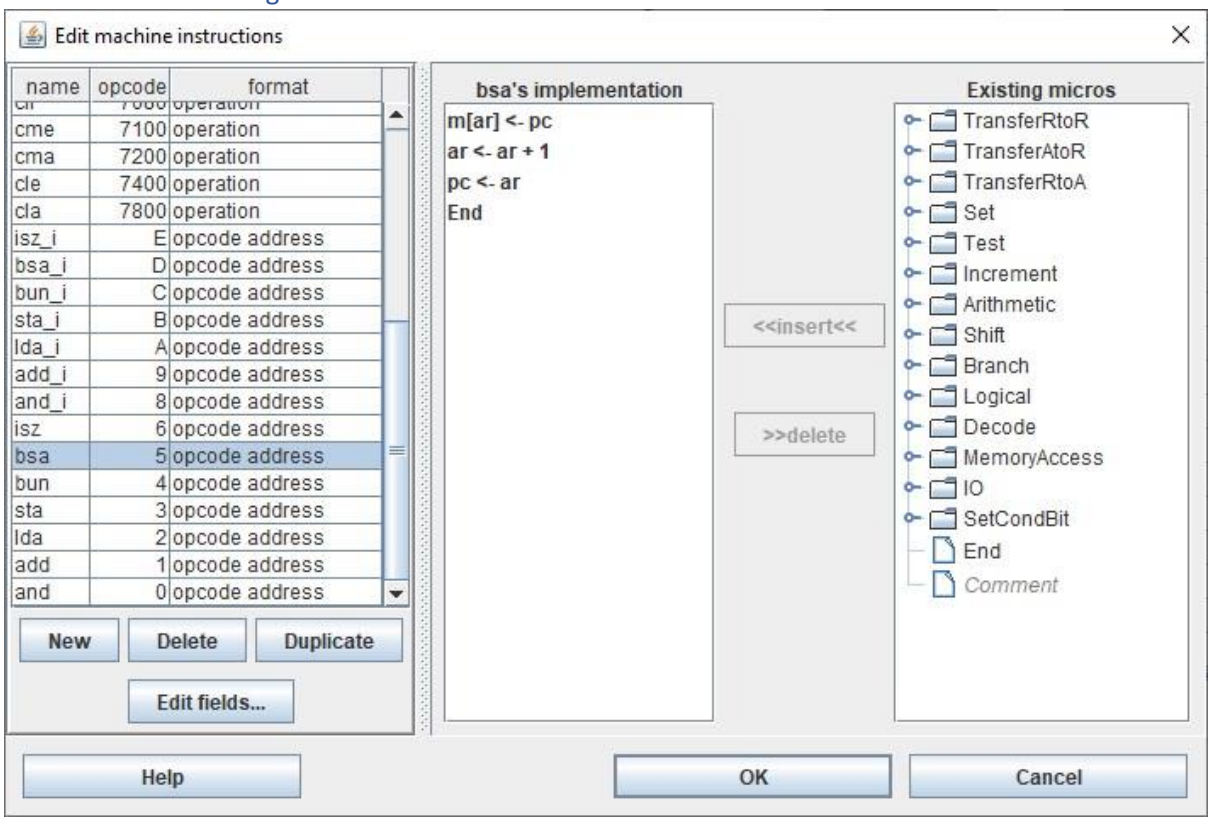
- **PC <- AR**


Edit Microinstructions
Type of Microinstruction: TransferRtoR
name source srcStartBit dest destStartBit numBits
ac (0) <- e e 0 ac 15 1
ac (15) <- e e 0 ac 0 1
ac <- dr dr 0 ac 0 16
ar <- ir (0-11) ir 4 ar 0 12
ar <- pc pc 0 ar 0 12
e <- ac (0) ac 15 e 0 1
e <- ac (15) ac 0 e 0 1
i <- ir (15) ir 0 i 0 1
pc <- ar ar 0 pc 0 12
New
Delete
Duplicate
Help
OK
Cancel

- **SC <- 0**

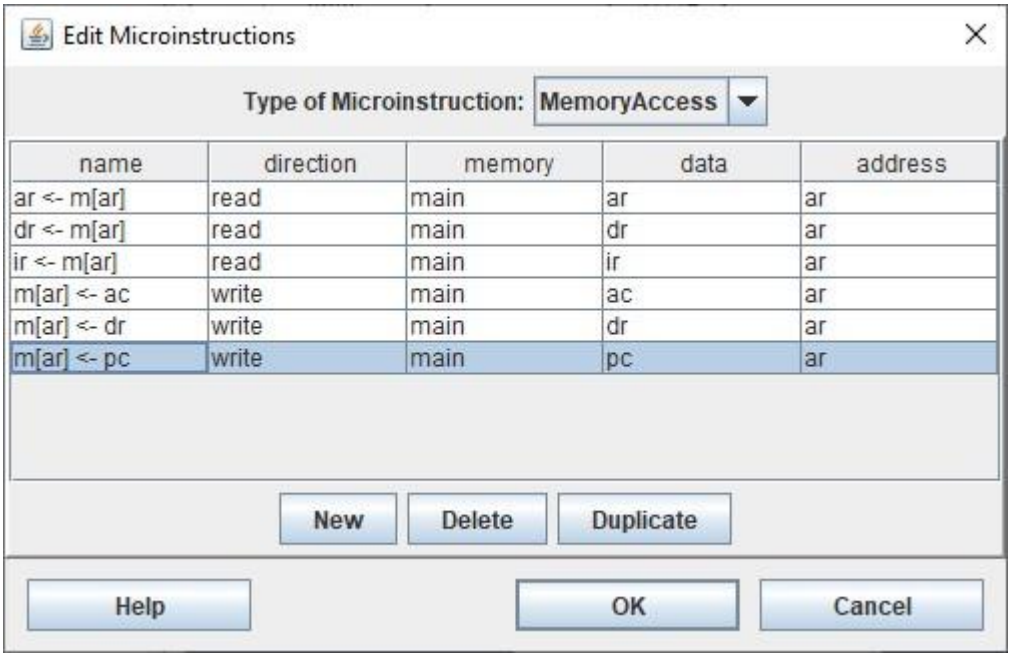
Branch and Save return Address

BSA – Direct Addressing Mode

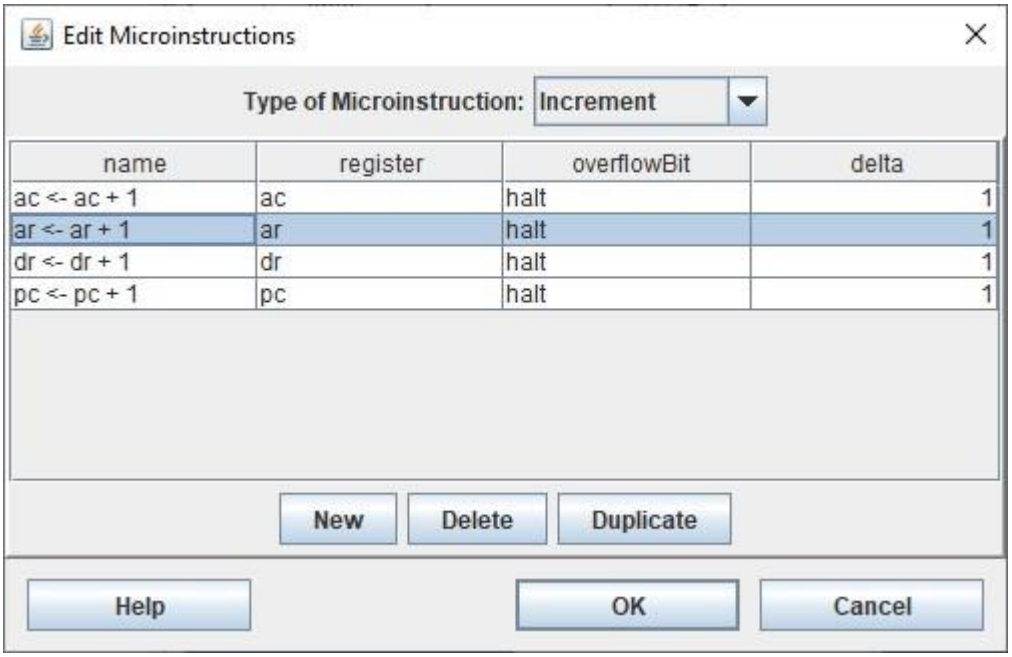


Implementation of BSA

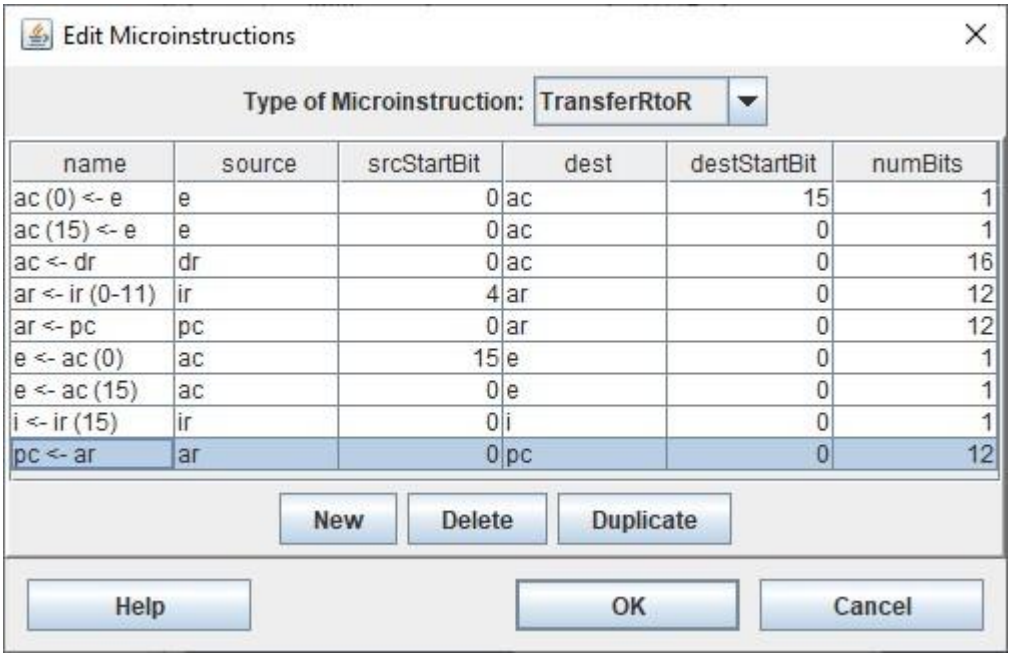
- **M[AR] <- PC**



- **AR <- AR + 1**

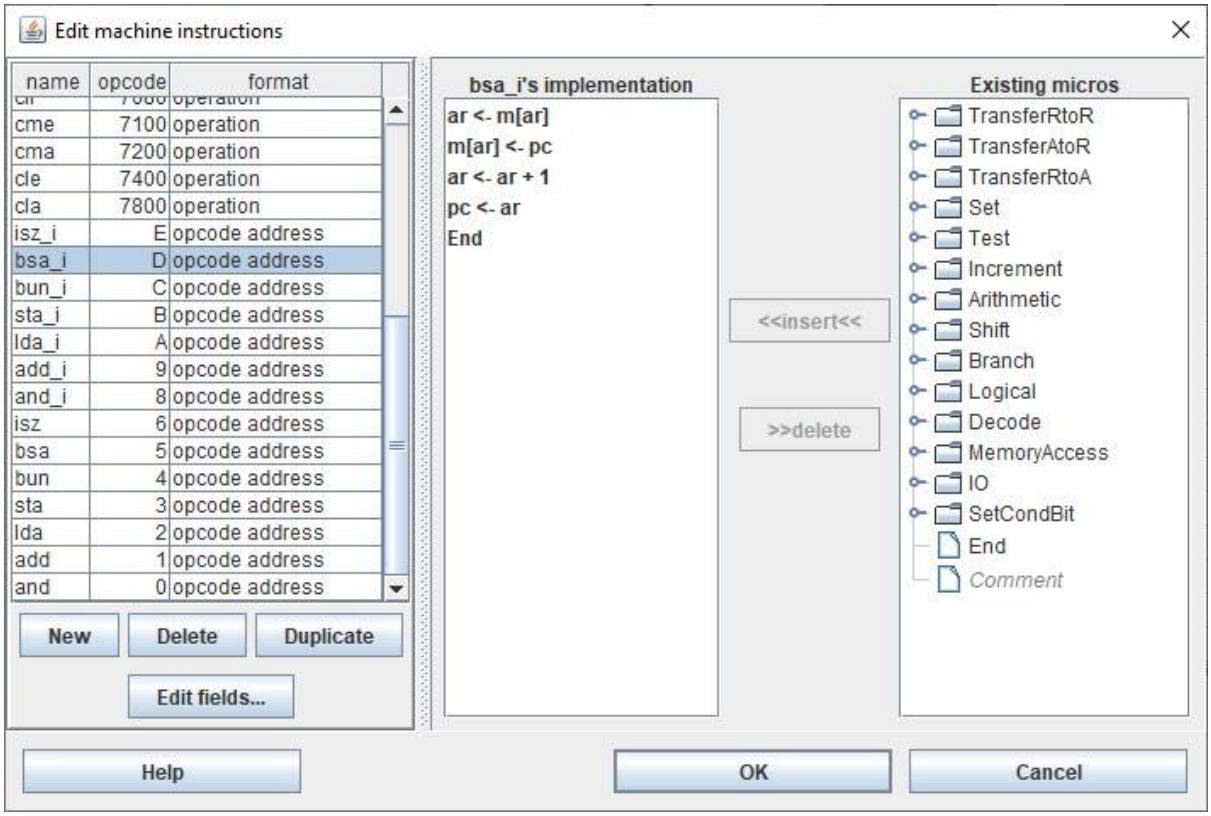

Edit Microinstructions
Type of Microinstruction: Increment
name | register | overflowBit | delta
ac <- ac + 1 | ac | halt | 1
ar <- ar + 1 | ar | halt | 1
dr <- dr + 1 | dr | halt | 1
pc <- pc + 1 | pc | halt | 1
New
Delete
Duplicate
Help
OK
Cancel

- **PC <- AR**


Edit Microinstructions
Type of Microinstruction: TransferRtoR
name | source | srcStartBit | dest | destStartBit | numBits
ac (0) <- e | e | 0 | ac | 15 | 1
ac (15) <- e | e | 0 | ac | 0 | 1
ac <- dr | dr | 0 | ac | 0 | 16
ar <- ir (0-11) | ir | 4 | ar | 0 | 12
ar <- pc | pc | 0 | ar | 0 | 12
e <- ac (0) | ac | 15 | e | 0 | 1
e <- ac (15) | ac | 0 | e | 0 | 1
i <- ir (15) | ir | 0 | i | 0 | 1
pc <- ar | ar | 0 | pc | 0 | 12
New
Delete
Duplicate
Help
OK
Cancel

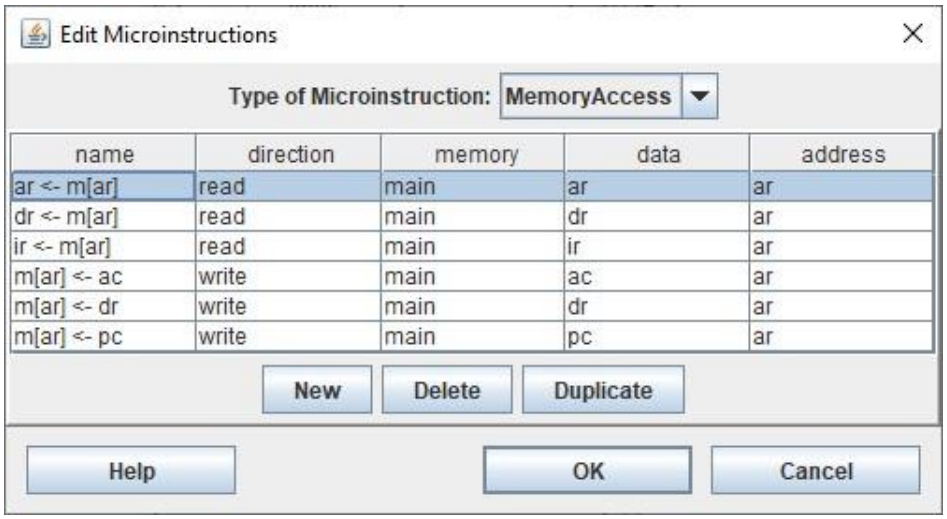
- **SC <- 0**

BSA_I – Indirect Addressing Mode

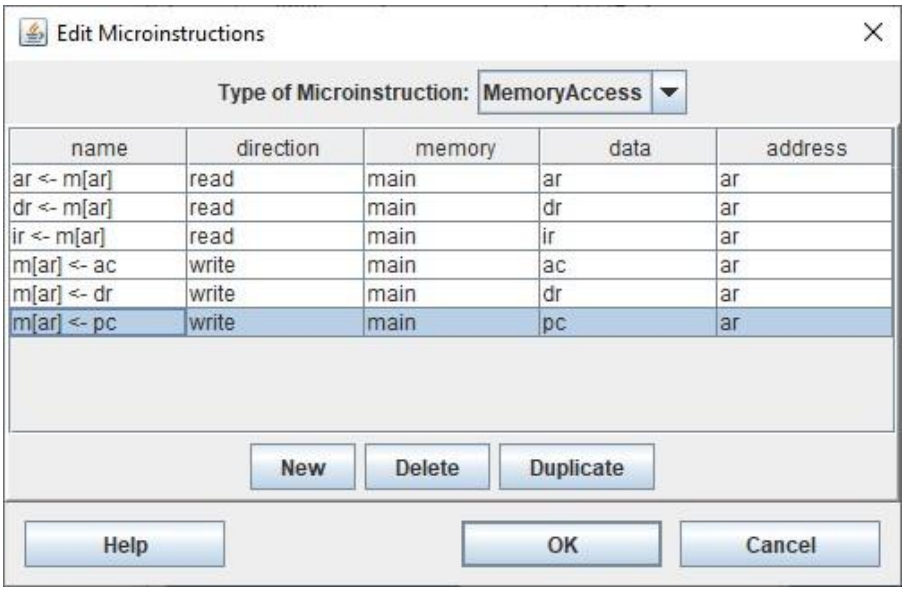


Implementation of BSA

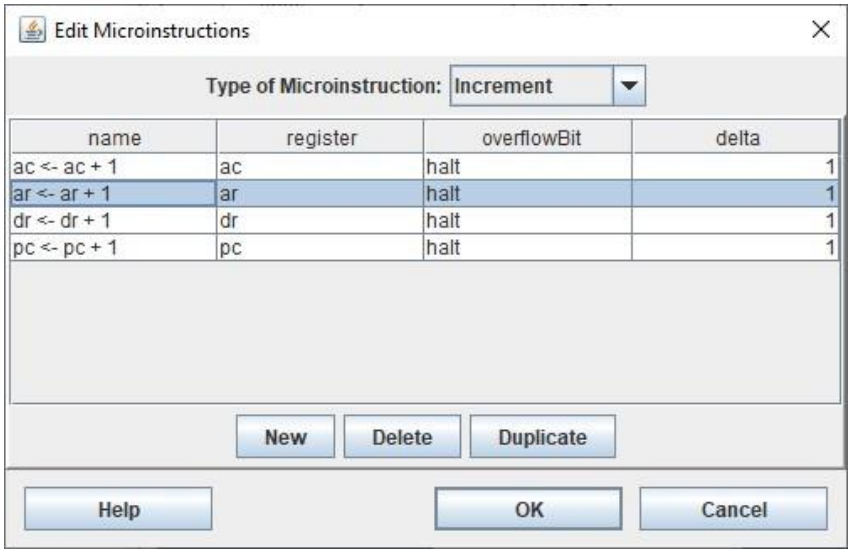
- **<u>AR <- M[AR]</u>**



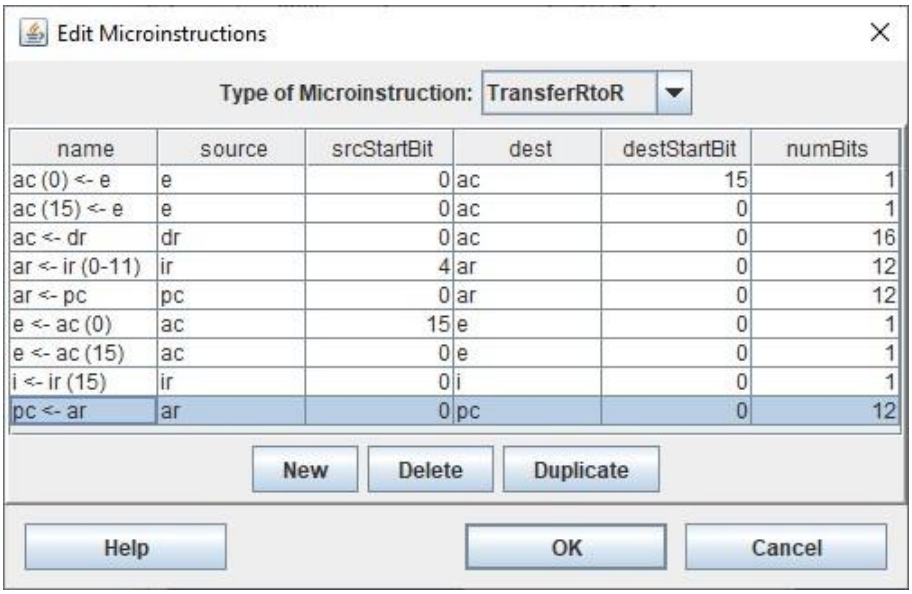
- **M[AR] <- PC**



- **AR <- AR + 1**



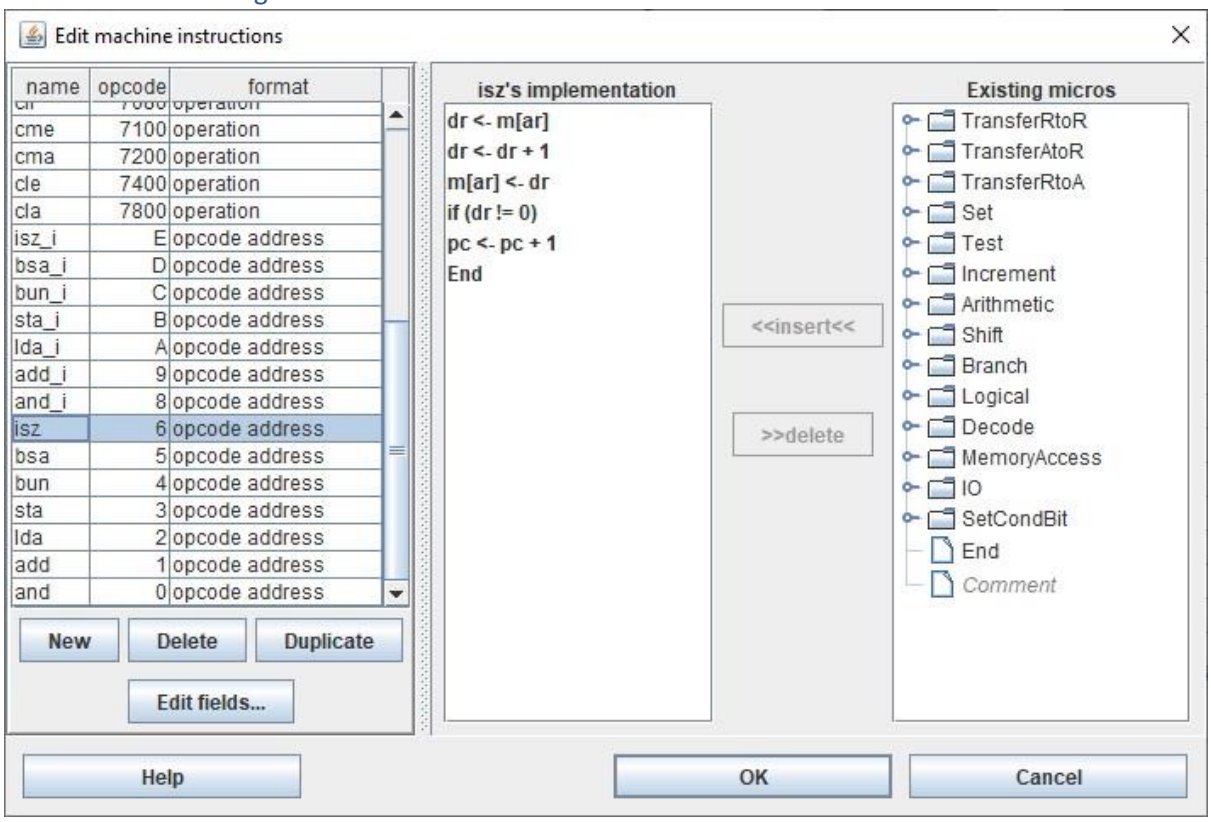
- **PC <- AR**



- **SC <- 0**

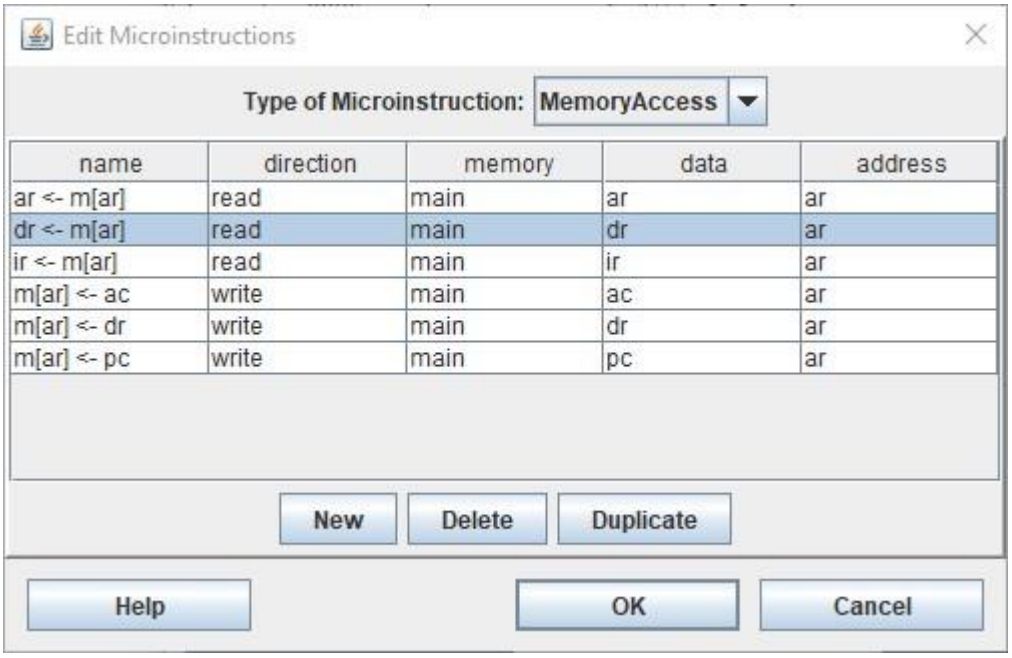
Increment and Skip If Zero

ISZ – Direct Addressing Mode

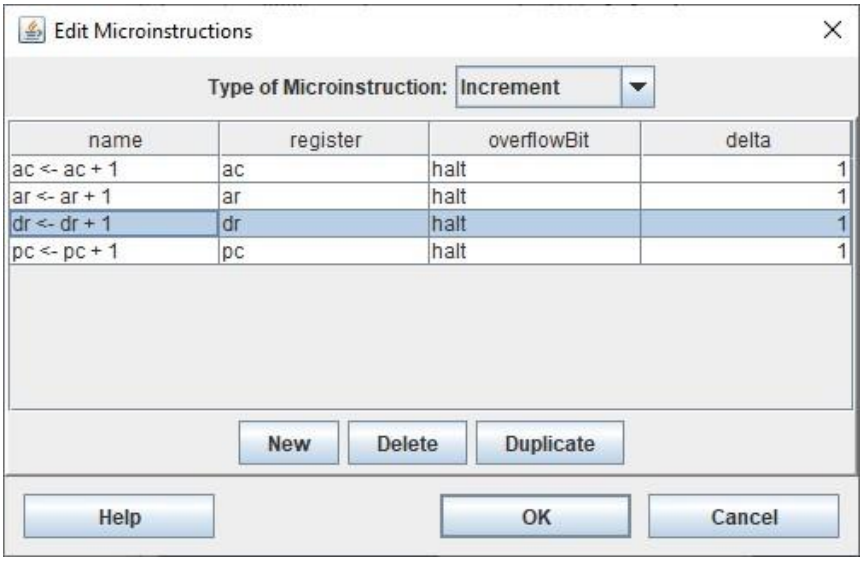


Implementation of ISZ

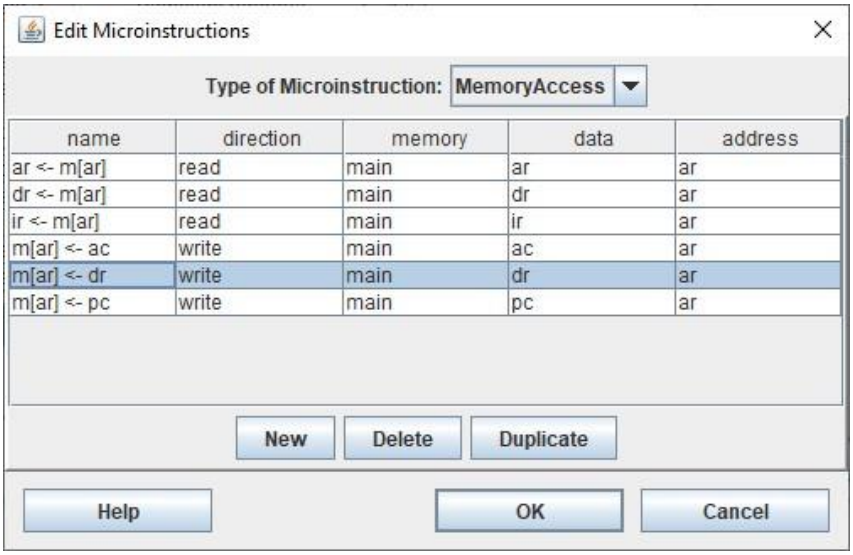
- **DR <- M[AR]**



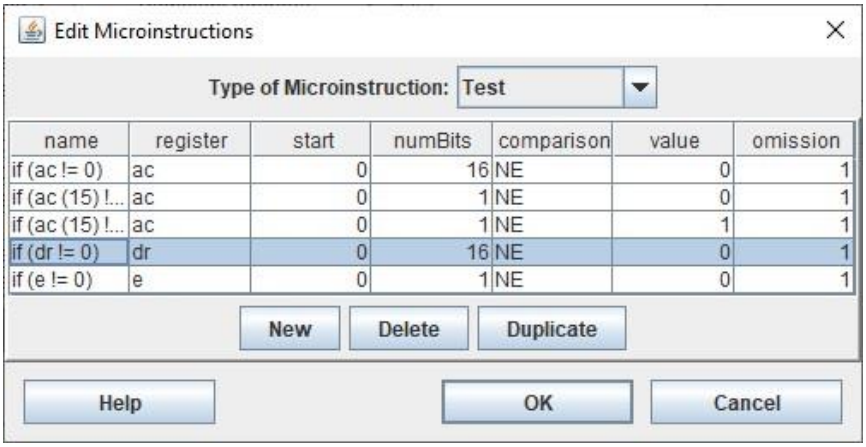
- **DR <- DR + 1**



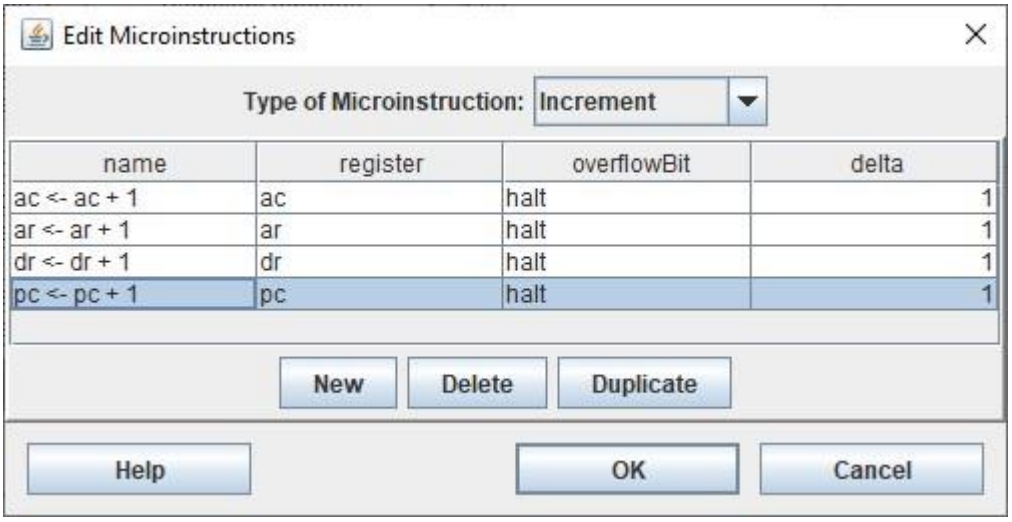
- **M[AR] <- DR**



- **If DR != 0** (CPUSim skips a micro-operation when this is true)

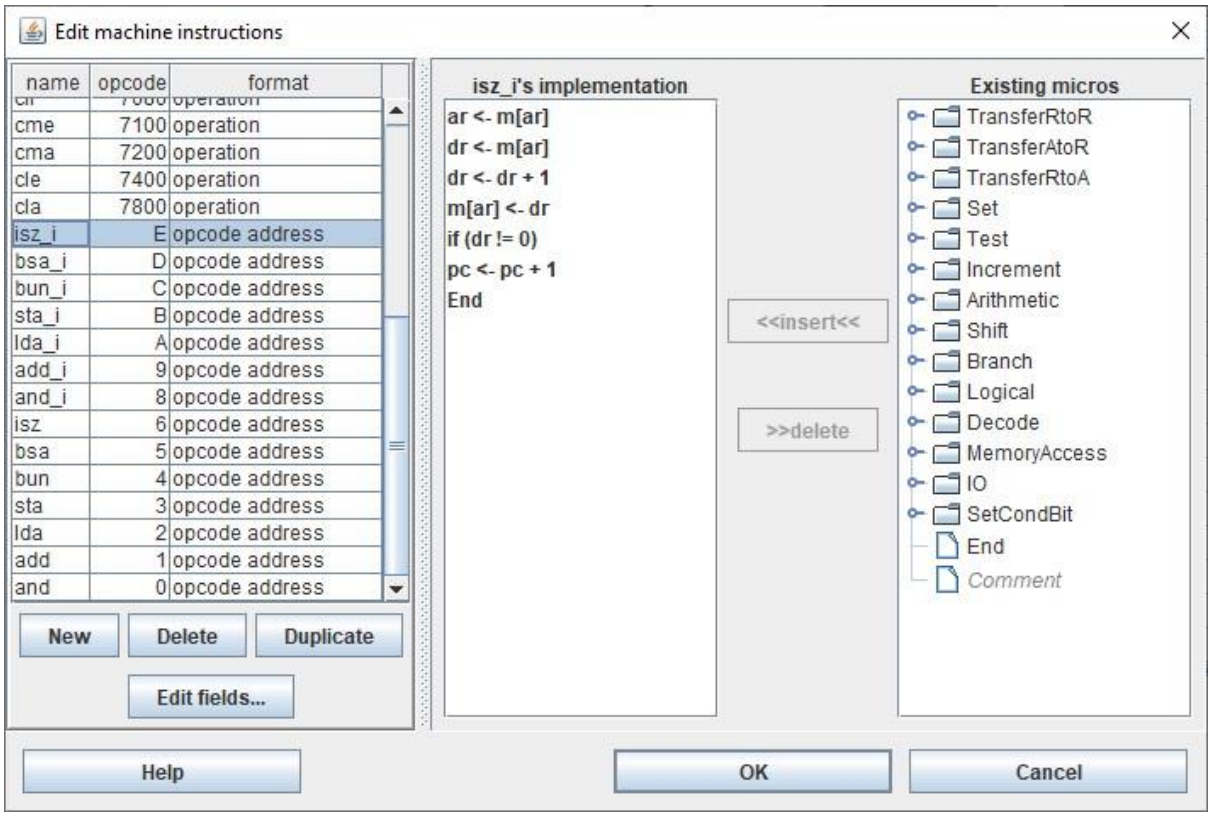


PC <- PC + 1



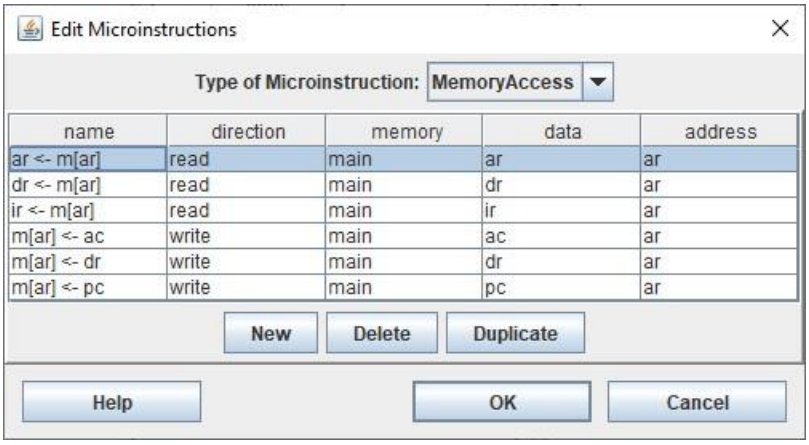
- **SC <- 0**

ISZ_I – Indirect Addressing Mode

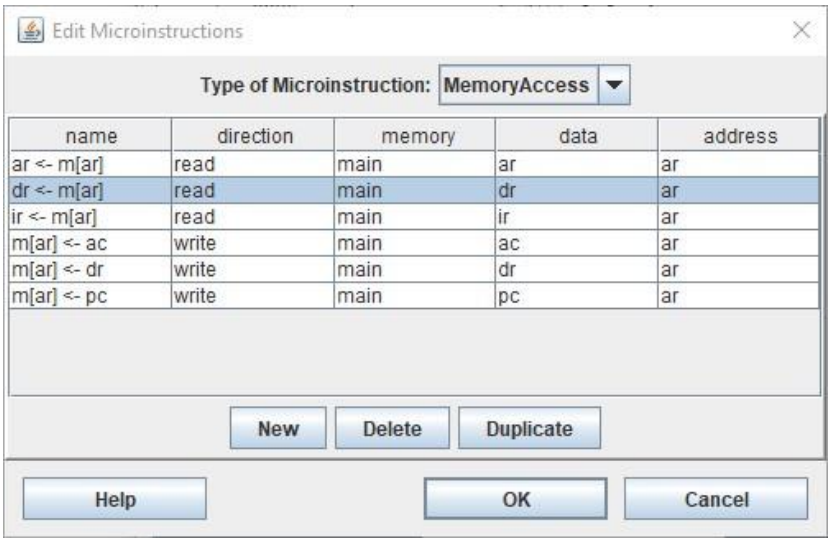


Implementation of ISZ_I

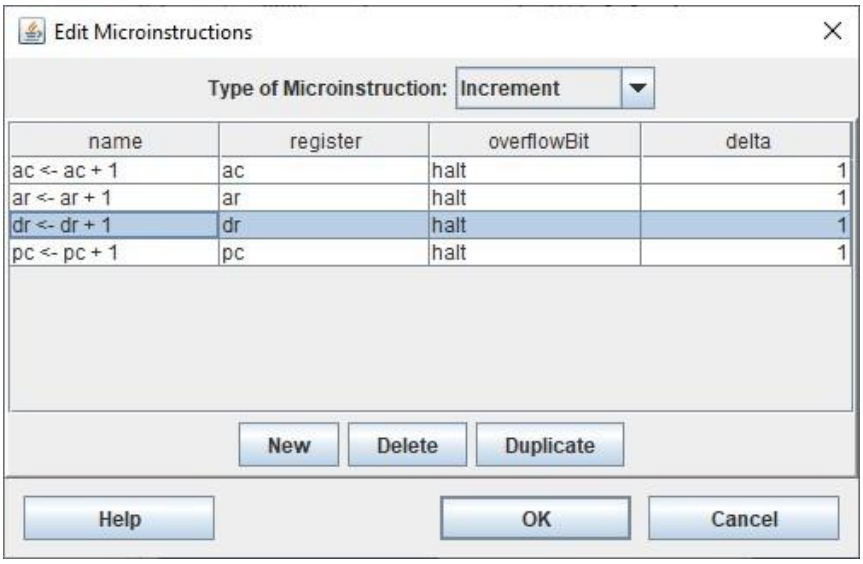
- **AR <- M[AR]**


Edit Microinstructions
Type of Microinstruction: MemoryAccess
name direction memory data address
ar <- m[ar] read main ar ar
dr <- m[ar] read main dr ar
ir <- m[ar] read main ir ar
m[ar] <- ac write main ac ar
m[ar] <- dr write main dr ar
m[ar] <- pc write main pc ar
New Delete Duplicate
Help OK Cancel

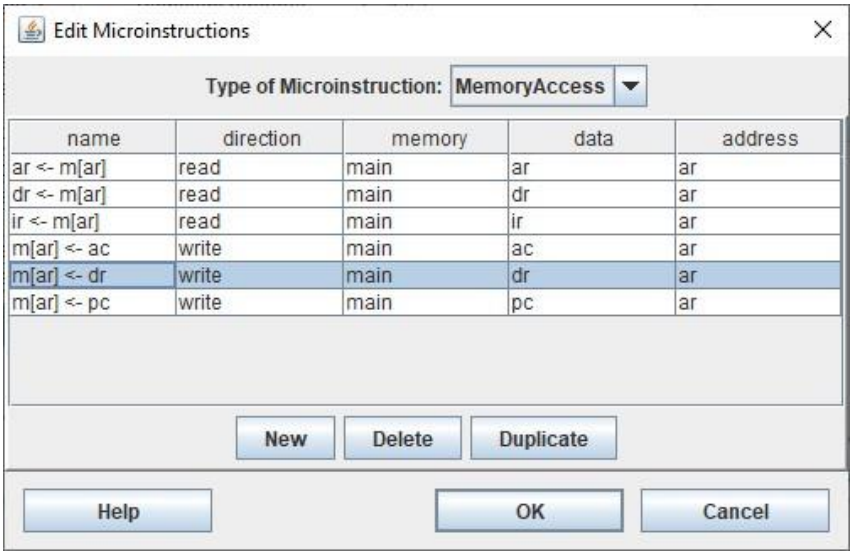
- **DR <- M[AR]**


Edit Microinstructions
Type of Microinstruction: MemoryAccess
name direction memory data address
ar <- m[ar] read main ar ar
dr <- m[ar] read main dr ar
ir <- m[ar] read main ir ar
m[ar] <- ac write main ac ar
m[ar] <- dr write main dr ar
m[ar] <- pc write main pc ar
New Delete Duplicate
Help OK Cancel

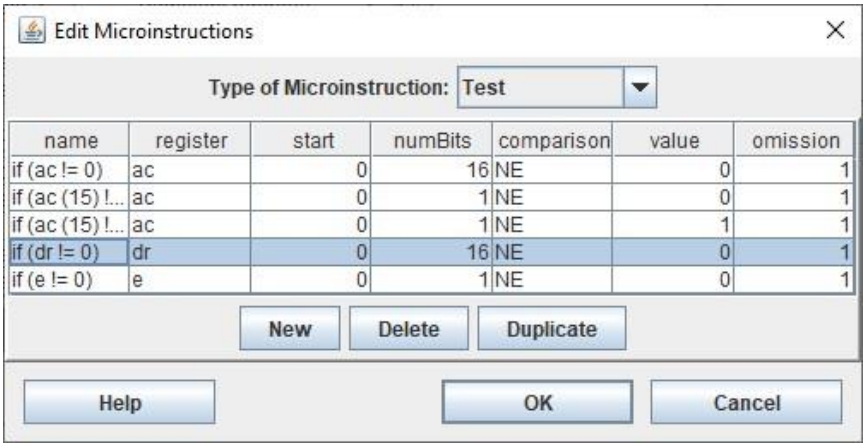
- **DR <- DR + 1**


Edit Microinstructions
Type of Microinstruction: Increment
name register overflowBit delta
ac <- ac + 1 ac halt 1
ar <- ar + 1 ar halt 1
dr <- dr + 1 dr halt 1
pc <- pc + 1 pc halt 1
New Delete Duplicate
Help OK Cancel

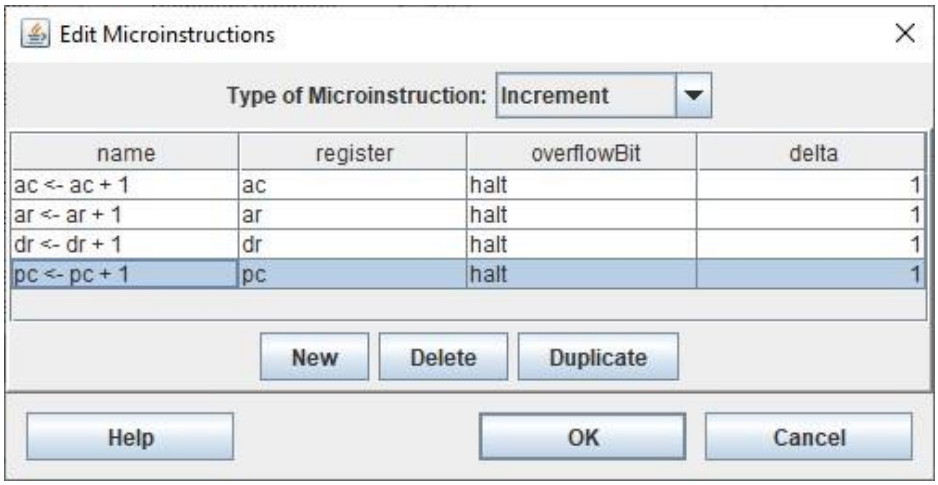
M[AR] <- DR



- **If DR != 0** (CPUSim skips a micro-operation when this is true)



- **PC <- PC + 1**



- **SC <- 0**

PRACTICAL 3

Dated 10th September 2019

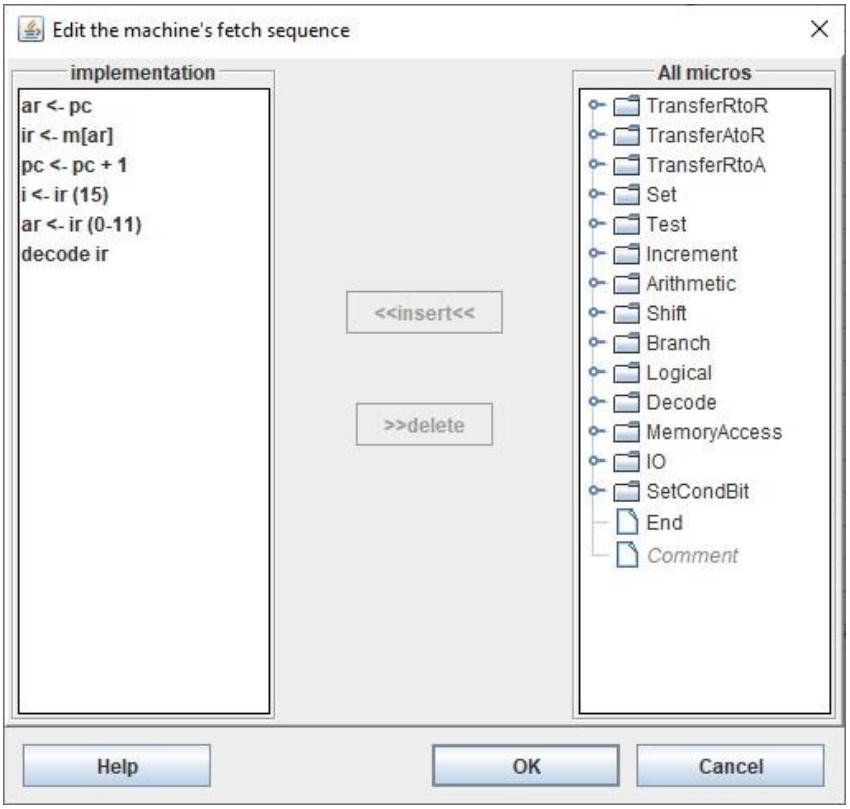
Objective

Create the fetch routine for the basic computer as designed in the previous practical.

T_0: $AR \leftarrow PC$
T_1: $IR \leftarrow M[AR]$, $PC \leftarrow PC + 1$
T_2: $D_0, \ldots, D_7 \leftarrow$ Decode $IR(12\text{–}14)$, $AR \leftarrow IR(0\text{–}11)$, $I \leftarrow IR(15)$

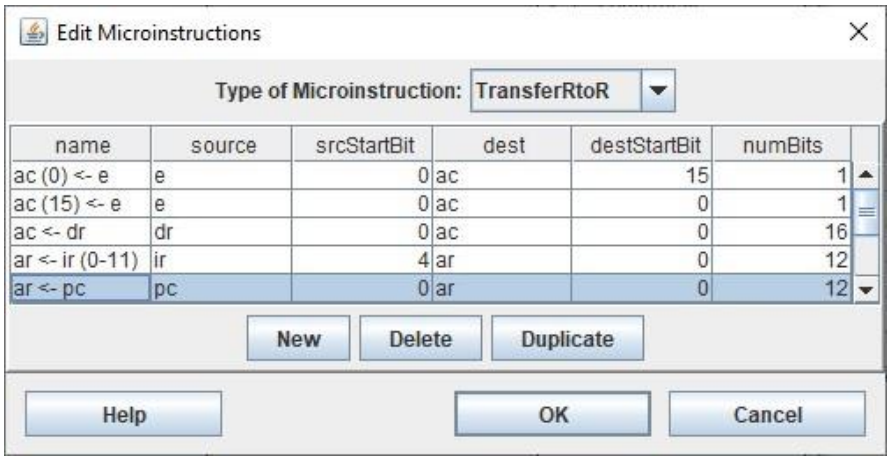
Observation

Fetch Sequence

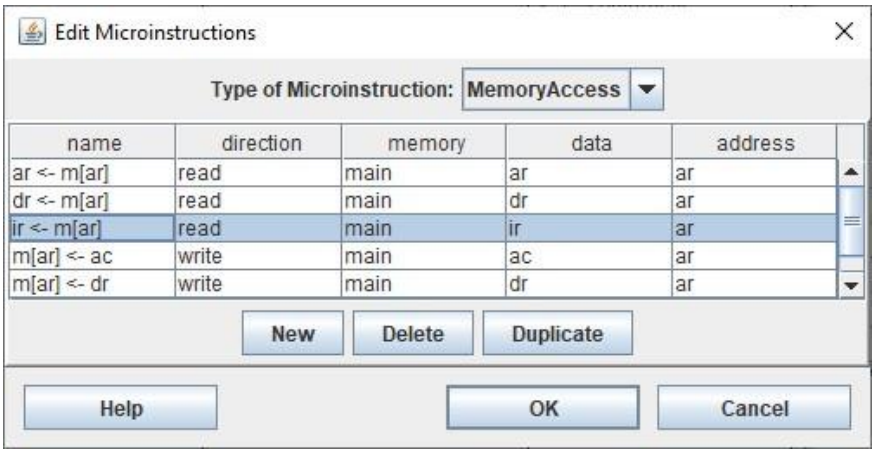


Implementation of the Fetch Sequence

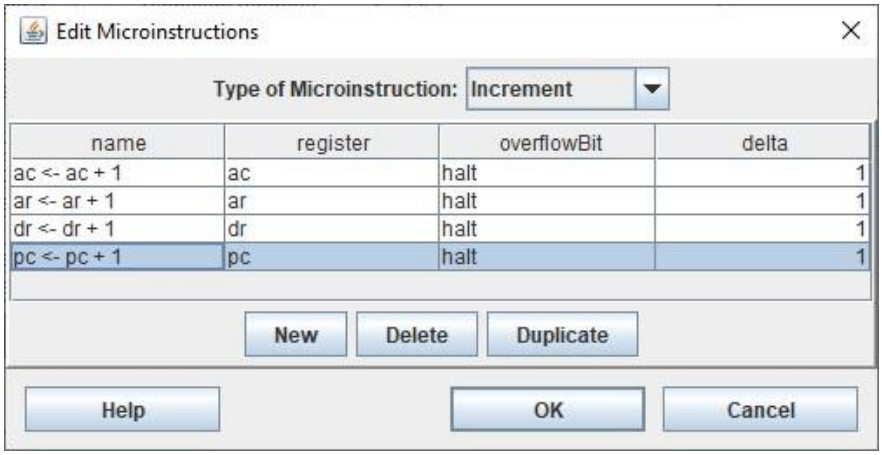
- **AR <- PC**



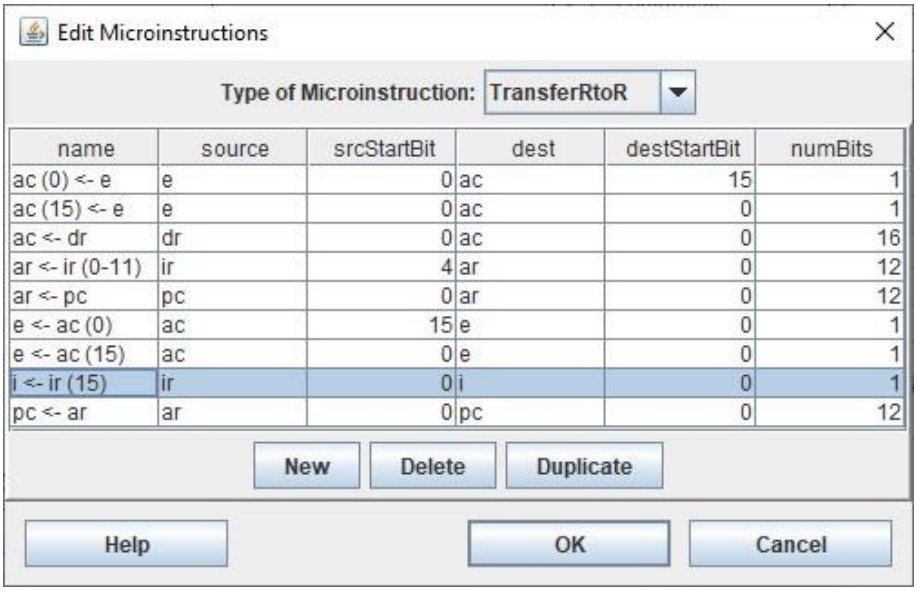
IR <- M[AR]



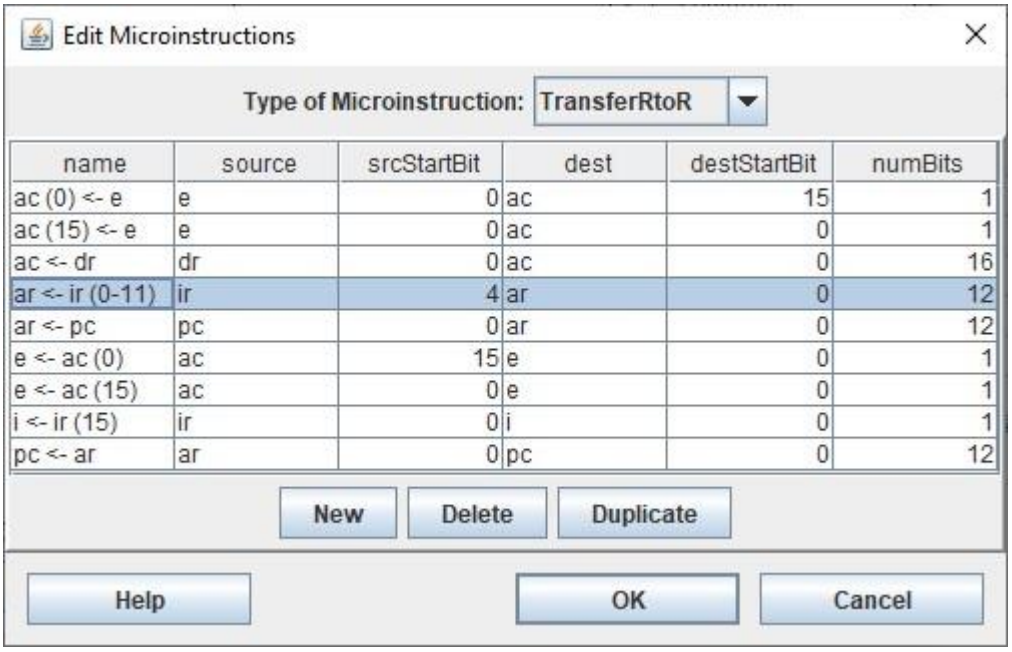
- **PC <- PC + 1**



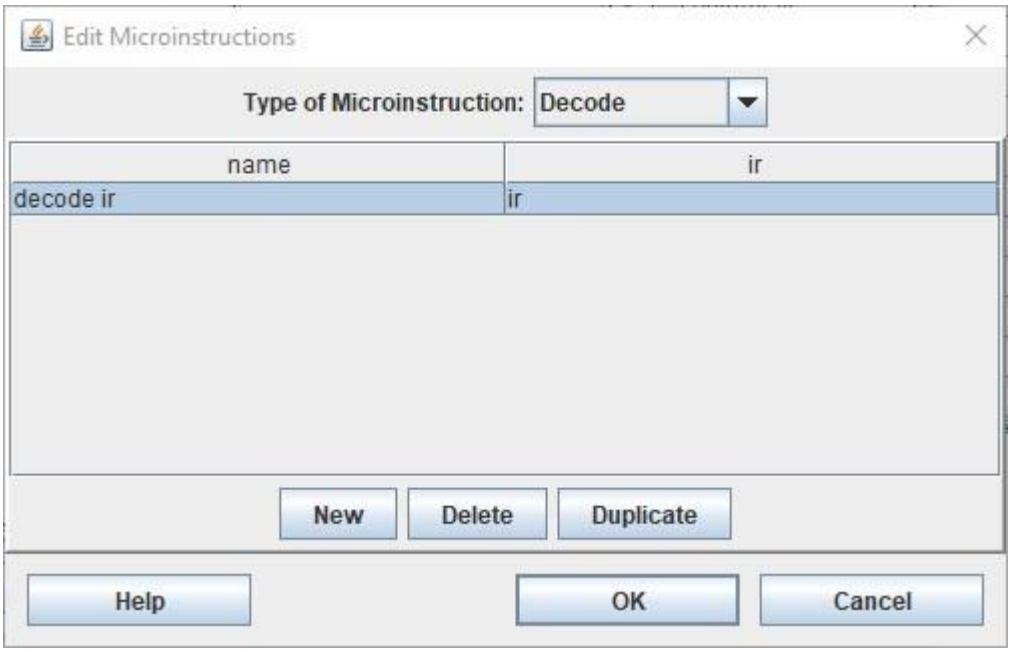
- **I <-IR (15)**



- **AR <- IR (0-11)**



- **Decode IR**



PRACTICAL 4

Dated 24th September 2019

Objective

Simulate the machine to determine the contents of AC, E, PC, AR, and IR registers in hexadecimal after the execution of each of the following register reference instructions:

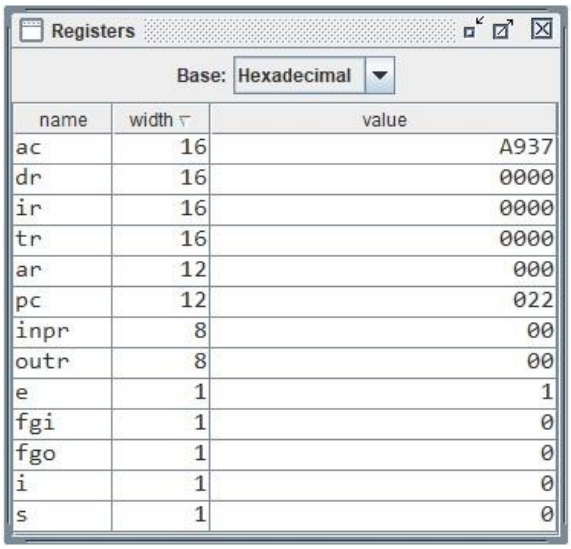
(a) CLA (e) CIR (i) SNA
(b) CLE (f) CIL (j) SZA
(c) CMA (g) INC (k) SZE
(d) CME (h) SPA (l) HLT

Initialize the contents of AC to $(A937)_{16}$, that of PC to $(022)_{16}$ and E to 1.

Observation

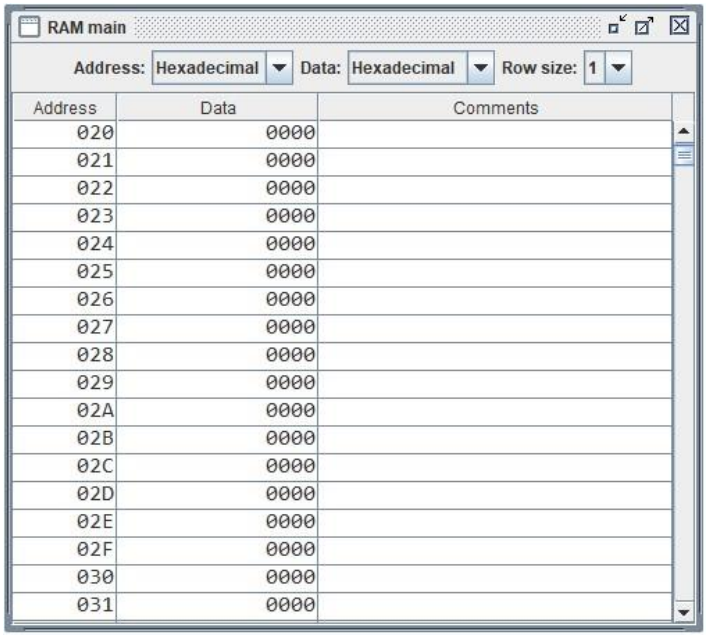
Before Loading

Registers



name	width	value
ac	16	A937
dr	16	0000
ir	16	0000
tr	16	0000
ar	12	000
pc	12	022
inpr	8	00
outr	8	00
e	1	1
fgi	1	0
fgo	1	0
i	1	0
s	1	0

Memory



Address	Data	Comments
020	0000	
021	0000	
022	0000	
023	0000	
024	0000	
025	0000	
026	0000	
027	0000	
028	0000	
029	0000	
02A	0000	
02B	0000	
02C	0000	
02D	0000	
02E	0000	
02F	0000	
030	0000	
031	0000	

CLA

```
cla
```

RAM main

Address: Hexadecimal Data: Hexadecimal Row size: 1

Address	Data	Comments
020	0000	
021	0000	
022	7800	cla
023	0000	
024	0000	

Registers After Execution

Registers

Base: Hexadecimal

name	width	value
ac	16	0000
dr	16	0000
ir	16	7800
tr	16	0000
ar	12	800
pc	12	023
inpr	8	00
outr	8	00
e	1	1
fgi	1	0
fgo	1	0
i	1	0
s	1	0

CLE

Assembly Program

```
cle
```

Memory After Loading

RAM main

Address: Hexadecimal Data: Hexadecimal Row size: 1

Break	Address	Data	Comments
☐	020	0000	
☐	021	0000	
☐	022	7400	cle
☐	023	0000	
☐	024	0000	

Registers

Base: Hexadecimal

name	width	value
ac	16	A937
dr	16	0000
ir	16	7400
tr	16	0000
ar	12	400
pc	12	023
inpr	8	00
outr	8	00
e	1	0
fgi	1	0
fgo	1	0
i	1	0
s	1	0

CMA

Assembly Program

```
cma
```

Memory After Loading

RAM main

Address: Hexadecimal Data: Hexadecimal Row size: 1

Break	Address	Data	Comments
	020	0000	
	021	0000	
	022	7200	cma
	023	0000	
	024	0000	

Registers After Execution

Registers

Base: Hexadecimal

name	width	value
ac	16	56C8
dr	16	0000
ir	16	7200
tr	16	0000
ar	12	200
pc	12	023
inpr	8	00
outr	8	00
e	1	1
fgi	1	0
fgo	1	0
i	1	0
s	1	0

CME

```
cme
```

RAM main

Address: Hexadecimal Data: Hexadecimal Row size: 1

Break	Address	Data	Comments
	020	0000	
	021	0000	
	022	7100	cme
	023	0000	
	024	0000	

Registers After Execution

Registers

Base: Hexadecimal

name	width	value
ac	16	A937
dr	16	0000
ir	16	7100
tr	16	0000
ar	12	100
pc	12	023
inpr	8	00
outr	8	00
e	1	0
fgi	1	0
fgo	1	0
i	1	0
s	1	0

CIR

Assembly Program

```
cir
```

Memory After Loading

RAM main

Address: Hexadecimal Data: Hexadecimal Row size: 1

Break	Address	Data	Comments
	020	0000	
	021	0000	
	022	7080	cir
	023	0000	
	024	0000	

Registers

Base: Hexadecimal

name	width	value
ac	16	D49B
dr	16	0000
ir	16	7080
tr	16	0000
ar	12	080
pc	12	023
inpr	8	00
outr	8	00
e	1	1
fgi	1	0
fgo	1	0
i	1	0
s	1	0

CIL

Assembly Program

```
cil
```

Memory After Loading

RAM main

Address: Hexadecimal Data: Hexadecimal Row size: 1

Break	Address	Data	Comments
	020	0000	
	021	0000	
	022	7040	cil
	023	0000	
	024	0000	

Registers After Execution

Registers

Base: Hexadecimal

name	width	value
ac	16	526F
dr	16	0000
ir	16	7040
tr	16	0000
ar	12	040
pc	12	023
inpr	8	00
outr	8	00
e	1	1
fgi	1	0
fgo	1	0
i	1	0
s	1	0

INC

```
inc
```

RAM main

Address: Hexadecimal Data: Hexadecimal Row size: 1

Break	Address	Data	Comments
	020	0000	
	021	0000	
	022	7020	inc
	023	0000	
	024	0000	

Registers After Execution

Registers

Base: Hexadecimal

name	width	value
ac	16	A938
dr	16	0000
ir	16	7020
tr	16	0000
ar	12	020
pc	12	023
inpr	8	00
outr	8	00
e	1	1
fgi	1	0
fgo	1	0
i	1	0
s	1	0

SPA

Assembly Program

```
spa
```

Memory After Loading

RAM main

Address: Hexadecimal Data: Hexadecimal Row size: 1

Break	Address	Data	Comments
	020	0000	
	021	0000	
	022	7010	spa
	023	0000	
	024	0000	

Registers

Base: Hexadecimal

name	width	value
ac	16	A937
dr	16	0000
ir	16	7010
tr	16	0000
ar	12	010
pc	12	023
inpr	8	00
outr	8	00
e	1	1
fgi	1	0
fgo	1	0
i	1	0
s	1	0

SNA

Assembly Program

```
sna
```

Memory After Loading

RAM main

Address: Hexadecimal Data: Hexadecimal Row size: 1

Break	Address	Data	Comments
	020	0000	
	021	0000	
	022	7008	sna
	023	0000	
	024	0000	

Registers After Execution

Registers

Base: Hexadecimal

name	width	value
ac	16	A937
dr	16	0000
ir	16	7008
tr	16	0000
ar	12	008
pc	12	024
inpr	8	00
outr	8	00
e	1	1
fgi	1	0
fgo	1	0
i	1	0
s	1	0

SZA

```
sza
```

RAM main

Address: Hexadecimal Data: Hexadecimal Row size: 1

Break	Address	Data	Comments
	020	0000	
	021	0000	
	022	7004	sza
	023	0000	
	024	0000	

Registers After Execution

Registers

Base: Hexadecimal

name	width	value
ac	16	A937
dr	16	0000
ir	16	7004
tr	16	0000
ar	12	004
pc	12	023
inpr	8	00
outr	8	00
e	1	1
fgi	1	0
fgo	1	0
i	1	0
s	1	0

SZE

Assembly Program

```
sze
```

Memory After Loading

RAM main

Address: Hexadecimal Data: Hexadecimal Row size: 1

Break	Address	Data	Comments
	020	0000	
	021	0000	
	022	7002	sze
	023	0000	
	024	0000	

Registers After Execution

Registers

Base: Hexadecimal

name	width	value
ac	16	A937
dr	16	0000
ir	16	7002
tr	16	0000
ar	12	002
pc	12	023
inpr	8	00
outr	8	00
e	1	1
fgi	1	0
fgo	1	0
i	1	0
s	1	0

HLT

Assembly Program

```
hlt
```

Memory After Loading

RAM main

Address: Hexadecimal Data: Hexadecimal Row size: 1

Break	Address	Data	Comments
	020	0000	
	021	0000	
	022	7001	hlt
	023	0000	
	024	0000	

Registers After Execution

Registers

Base: Hexadecimal

name	width	value
ac	16	A937
dr	16	0000
ir	16	7001
tr	16	0000
ar	12	001
pc	12	023
inpr	8	00
outr	8	00
e	1	1
fgi	1	0
fgo	1	0
i	1	0
s	1	1


Execution Halted
The following halt condition bits are set: halt
OK

PRACTICAL 5

Dated 24th September 2019

Objective

Simulate the machine for the following memory-reference instructions with I = 0 and address part = 082. The instruction is to be stored at address 022 in RAM. Initialize the memory word at address 082 with the operand B8F2 and AC with A937. Determine the contents of AC, E, PC, AR, and IR registers in hexadecimal after the execution.

(a) ADD
(b) AND
(c) LDA
(d) STA
(e) BUN
(f) BSA
(g) ISZ

Observation

Before Loading

Registers

Registers

Base: Hexadecimal

name	width	value
ac	16	A937
dr	16	0000
ir	16	0000
tr	16	0000
ar	12	000
pc	12	022
inpr	8	00
outr	8	00
e	1	1
fgi	1	0
fgo	1	0
i	1	0
s	1	0

Memory

RAM main

Address: Hexadecimal Data: Hexadecimal Row size: 1

Address	Data	Comments
021	0000	
022	0000	
023	0000	
024	0000	
025	0000	

RAM main

Address: Hexadecimal Data: Hexadecimal Row size: 1

Address	Data	Comments
080	0000	
081	0000	
082	B8F2	
083	0000	
084	0000	

ADD

Assembly Program

add 0x082

Memory After Loading

RAM main

Address: Hexadecimal Data: Hexadecimal Row size: 1

Address	Data	Comments
020	0000	
021	0000	
022	1082	add 0x082
023	0000	
024	0000	

Registers After Execution

Registers

Base: Hexadecimal

name	width	value
ac	16	6229
dr	16	B8F2
ir	16	1082
tr	16	0000
ar	12	082
pc	12	023
inpr	8	00
outr	8	00
e	1	1
fgi	1	0
fgo	1	0
i	1	0
s	1	1

AND

```
and 0x082
```

Memory After Loading

RAM main

Address: Hexadecimal Data: Hexadecimal Row size: 1

Break	Address	Data	Comments
	020	0000	
	021	0000	
	022	0082	and 0x082
	023	0000	
	024	0000	

Registers After Execution

Registers

Base: Hexadecimal

name	width	value
ac	16	A832
dr	16	B8F2
ir	16	0082
tr	16	0000
ar	12	082
pc	12	023
inpr	8	00
outr	8	00
e	1	1
fgi	1	0
fgo	1	0
i	1	0
s	1	0

LDA

Assembly Program

```
lda 0x082
```

Memory After Loading

RAM main

Address: Hexadecimal Data: Hexadecimal Row size: 1

Address	Data	Comments
020	0000	
021	0000	
022	2082	lda 0x082
023	0000	
024	0000	

Registers After Execution

Registers

Base: Hexadecimal

name	width	value
ac	16	B8F2
dr	16	B8F2
ir	16	2082
tr	16	0000
ar	12	082
pc	12	023
inpr	8	00
outr	8	00
e	1	1
fgi	1	0
fgo	1	0
i	1	0
s	1	0

STA

```
sta 0x082
```

Memory After Loading

RAM main

Address: Hexadecimal Data: Hexadecimal Row size: 1

Break	Address	Data	Comments
	020	0000	
	021	0000	
	022	3082	sta 0x082
	023	0000	
	024	0000	

Registers After Execution

Registers

Base: Hexadecimal

name	width	value
ac	16	A937
dr	16	0000
ir	16	3082
tr	16	0000
ar	12	082
pc	12	023
inpr	8	00
outr	8	00
e	1	1
fgi	1	0
fgo	1	0
i	1	0
s	1	0

Memory After Execution

RAM main

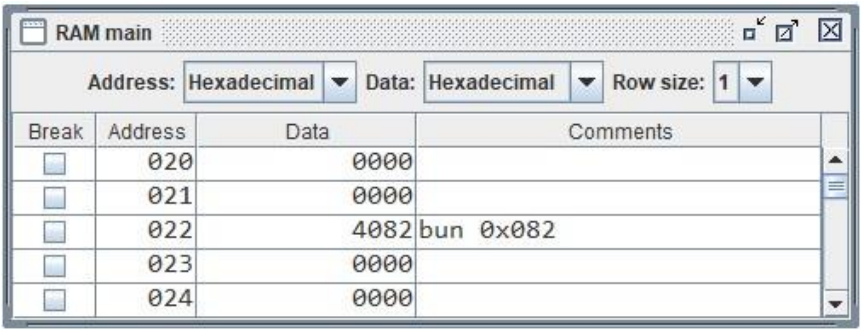
Address: Hexadecimal Data: Hexadecimal Row size: 1

Break	Address	Data	Comments
	080	0000	
	081	0000	
	082	A937	
	083	0000	
	084	0000	

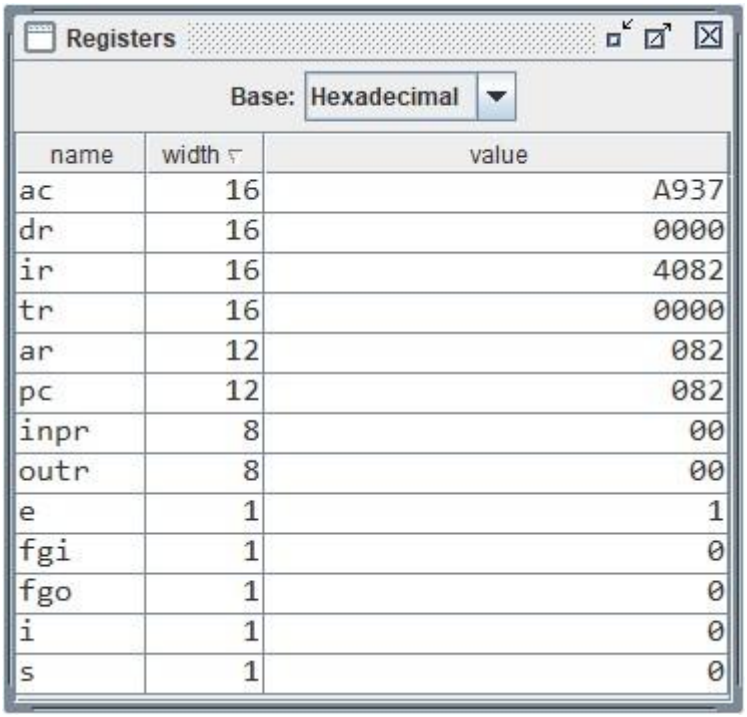
BUN

```
bun 0x082
```

Memory After Loading



Registers After Execution



BSA

```
bsa 0x082
```

Memory After Loading

RAM main

Address: Hexadecimal Data: Hexadecimal Row size: 1

Break	Address	Data	Comments
	020	0000	
	021	0000	
	022	5082	bsa 0x082
	023	0000	
	024	0000	

Registers After Execution

Registers

Base: Hexadecimal

name	width	value
ac	16	A937
dr	16	0000
ir	16	5082
tr	16	0000
ar	12	083
pc	12	083
inpr	8	00
outr	8	00
e	1	1
fgi	1	0
fgo	1	0
i	1	0
s	1	0

Memory After Execution

RAM main

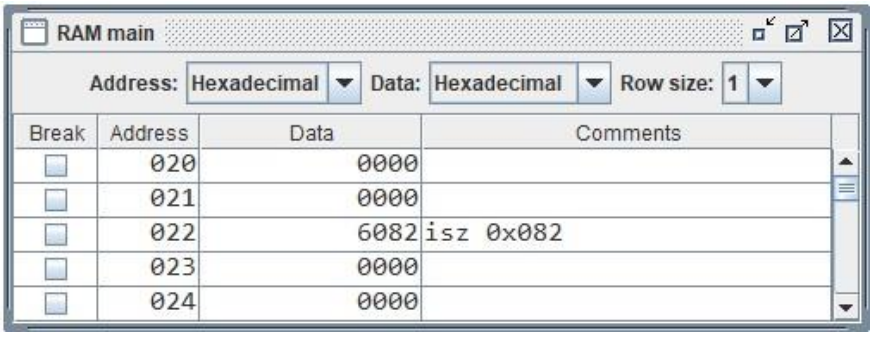
Address: Hexadecimal Data: Hexadecimal Row size: 1

Break	Address	Data	Comments
	080	0000	
	081	0000	
	082	0023	
	083	0000	
	084	0000	

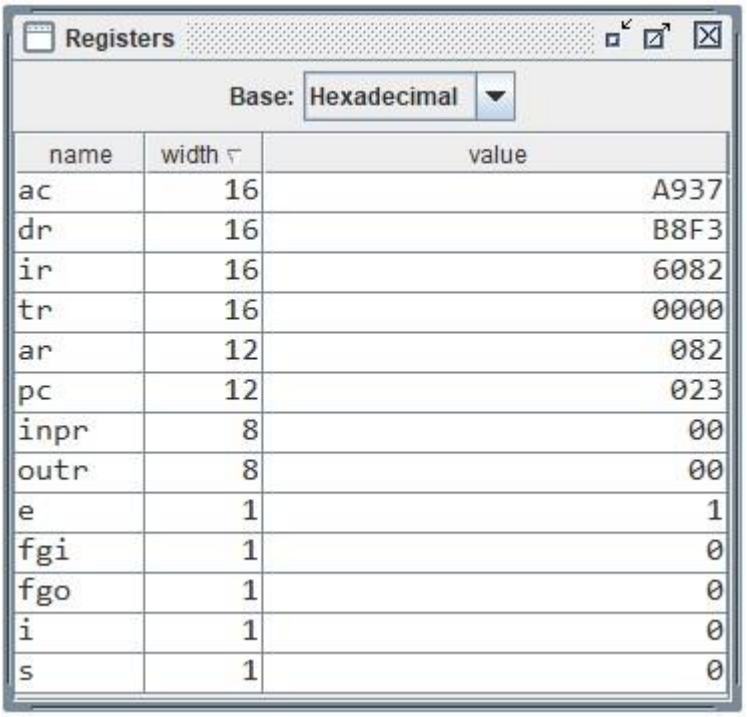
ISZ

```
isz 0x082
```

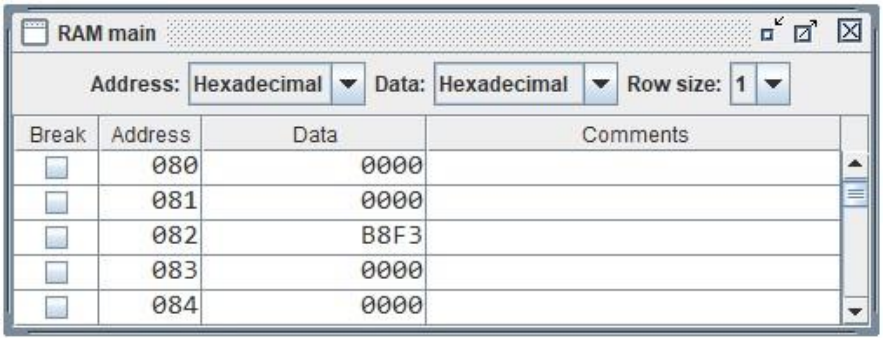
Memory After Loading



Registers After Execution



Memory After Execution



PRACTICAL 6

Dated 24th September 2019

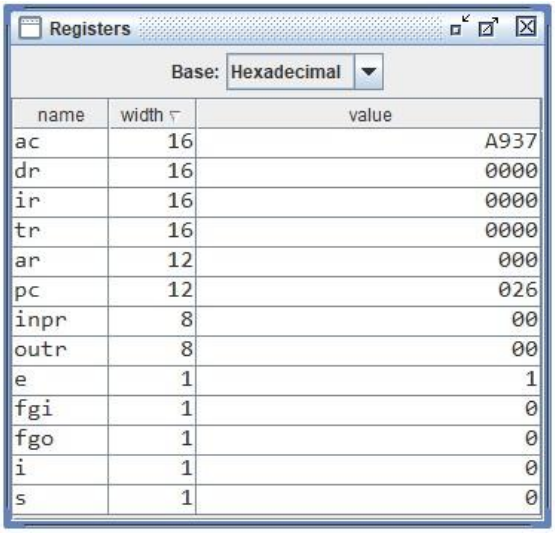
Objective

Simulate the machine for the memory-reference instructions referred in above question with I = 1 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 B8F2 and AC with A937. Determine the contents of AC, DR, PC, AR and IR in hexadecimal after the execution.

Observation

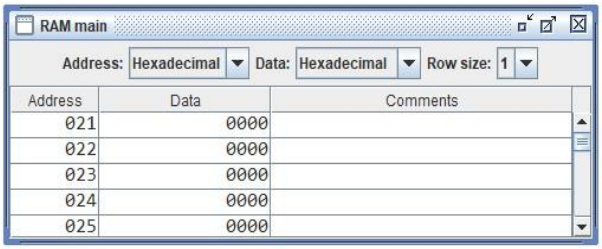
Before Loading

Registers

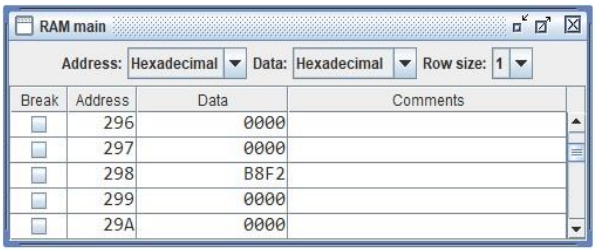


name	width	value
ac	16	A937
dr	16	0000
ir	16	0000
tr	16	0000
ar	12	000
pc	12	026
inpr	8	00
outr	8	00
e	1	1
fgi	1	0
fgo	1	0
i	1	0
s	1	0

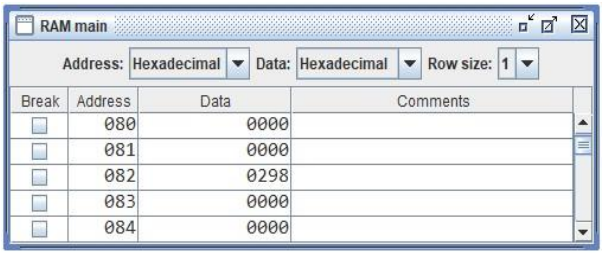
Memory



Address	Data	Comments
021	0000	
022	0000	
023	0000	
024	0000	
025	0000	



Break	Address	Data	Comments
	296	0000	
	297	0000	
	298	B8F2	
	299	0000	
	29A	0000	

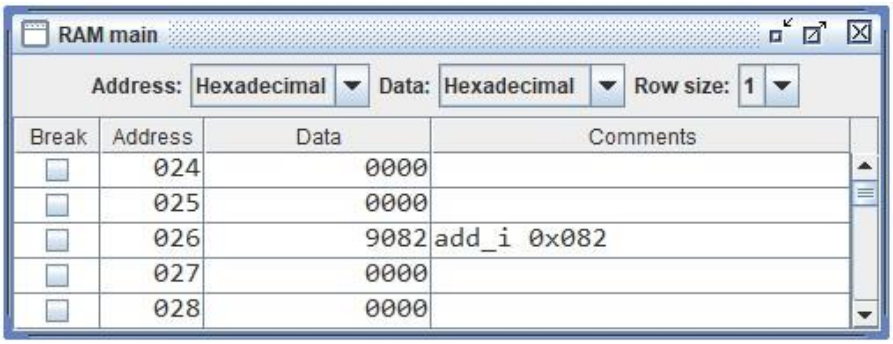


Break	Address	Data	Comments
	080	0000	
	081	0000	
	082	0298	
	083	0000	
	084	0000	

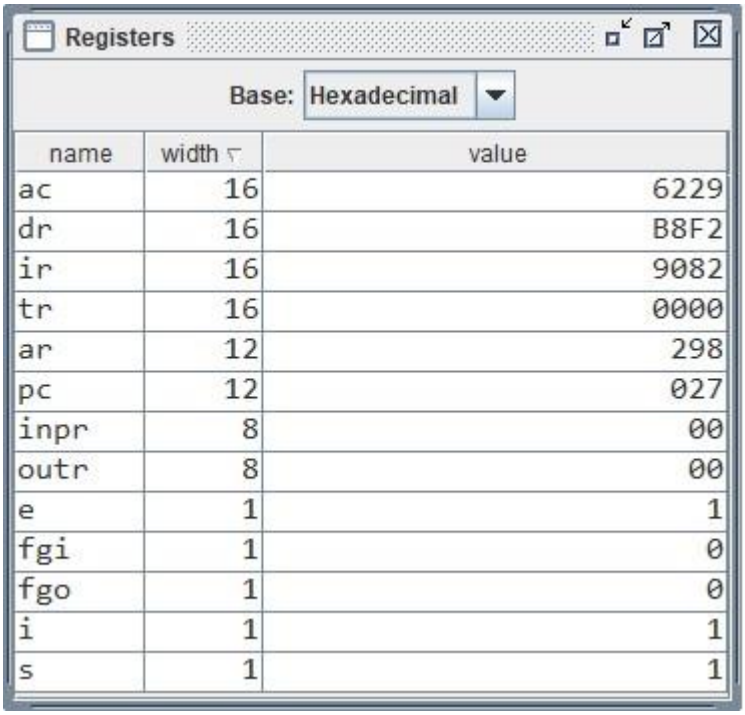
ADD

```
add_i 0x082
```

Memory After Loading



Registers After Execution



AND

Assembly Program

```
and_i 0x082
```

Memory After Loading

RAM main

Address: Hexadecimal Data: Hexadecimal Row size: 1

Break	Address	Data	Comments
	024	0000	
	025	0000	
	026	8082	and_i 0x082
	027	0000	
	028	0000	

Registers After Execution

Registers

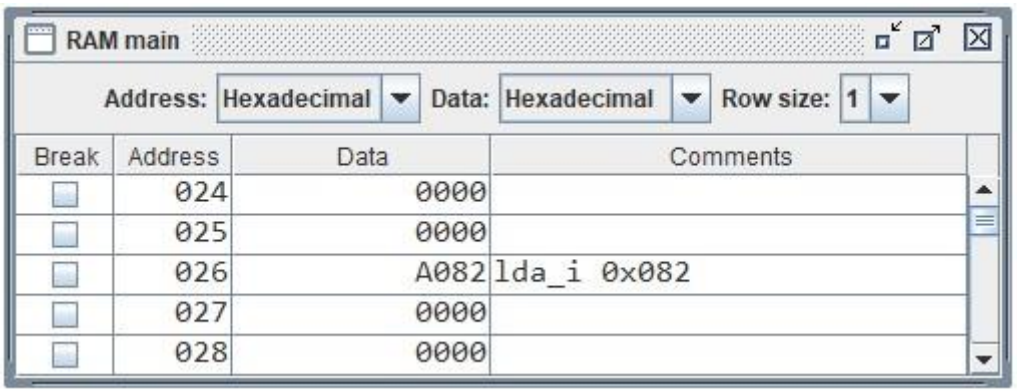
Base: Hexadecimal

name	width	value
ac	16	A832
dr	16	B8F2
ir	16	8082
tr	16	0000
ar	12	298
pc	12	027
inpr	8	00
outr	8	00
e	1	1
fgi	1	0
fgo	1	0
i	1	1
s	1	0

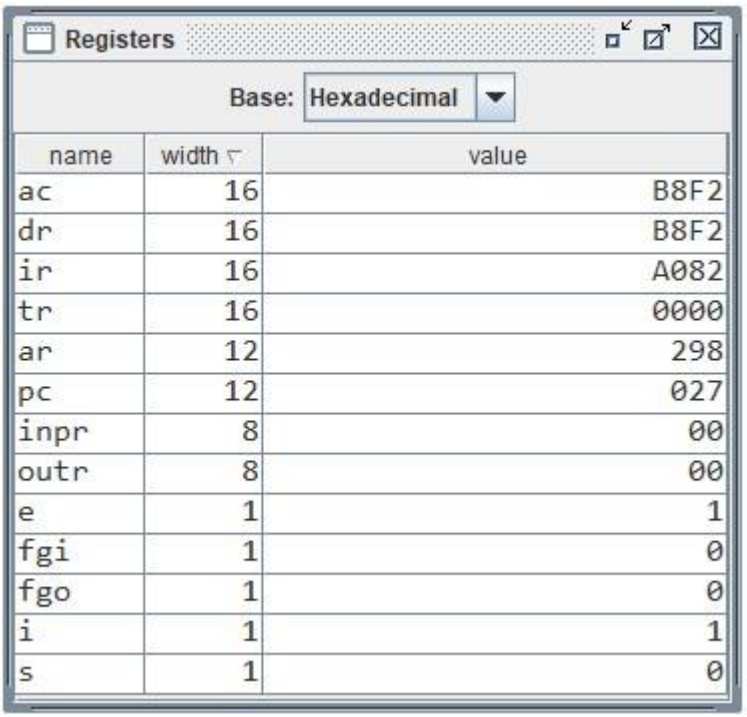
LDA

```
lda_i 0x082
```

Memory After Loading



Registers After Execution



STA

```
sta_i 0x082
```

Memory After Loading

RAM main

Address: Hexadecimal Data: Hexadecimal Row size: 1

Break	Address	Data	Comments
	024	0000	
	025	0000	
	026	B082	sta_i 0x082
	027	0000	
	028	0000	

Registers After Execution

Registers

Base: Hexadecimal

name	width	value
ac	16	A937
dr	16	0000
ir	16	B082
tr	16	0000
ar	12	298
pc	12	027
inpr	8	00
outr	8	00
e	1	1
fgi	1	0
fgo	1	0
i	1	1
s	1	0

Memory After Execution

RAM main

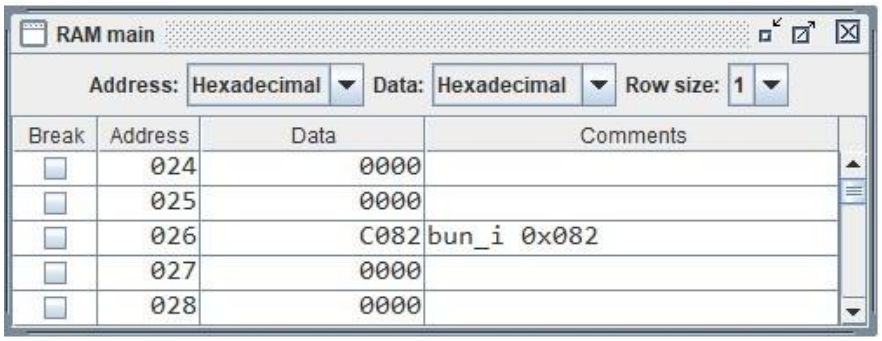
Address: Hexadecimal Data: Hexadecimal Row size: 1

Break	Address	Data	Comments
	296	0000	
	297	0000	
	298	A937	
	299	0000	
	29A	0000	

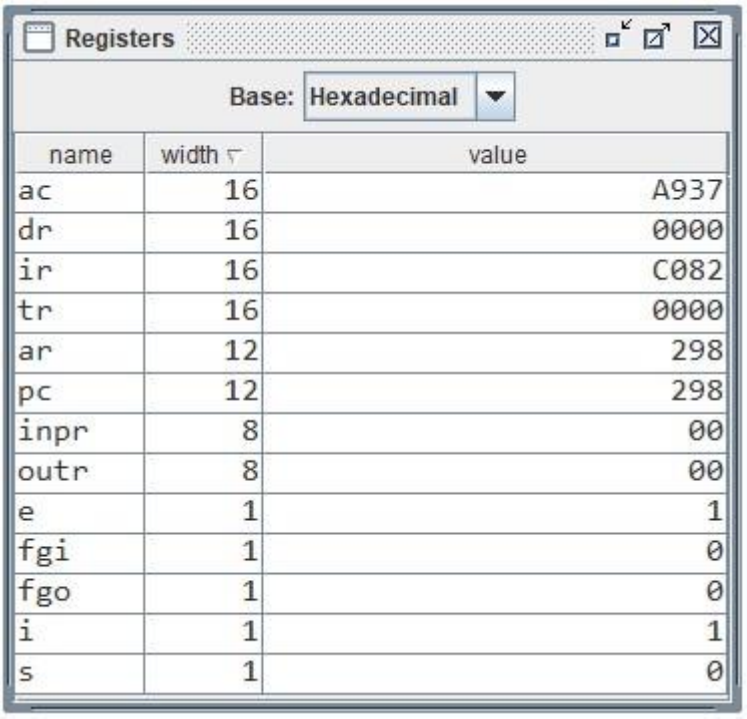
BUN

```
bun_i 0x082
```

Memory After Loading



Registers After Execution



name	width	value
ac	16	A937
dr	16	0000
ir	16	C082
tr	16	0000
ar	12	298
pc	12	298
inpr	8	00
outr	8	00
e	1	1
fgi	1	0
fgo	1	0
i	1	1
s	1	0

BSA

```
bsa_i 0x082
```

Memory After Loading

RAM main

Address: Hexadecimal Data: Hexadecimal Row size: 1

Break	Address	Data	Comments
	024	0000	
	025	0000	
	026	D082	bsa_i 0x082
	027	0000	
	028	0000	

Registers After Execution

Registers

Base: Hexadecimal

name	width	value
ac	16	A937
dr	16	0000
ir	16	D082
tr	16	0000
ar	12	299
pc	12	299
inpr	8	00
outr	8	00
e	1	1
fgi	1	0
fgo	1	0
i	1	1
s	1	0

Memory After Execution

RAM main

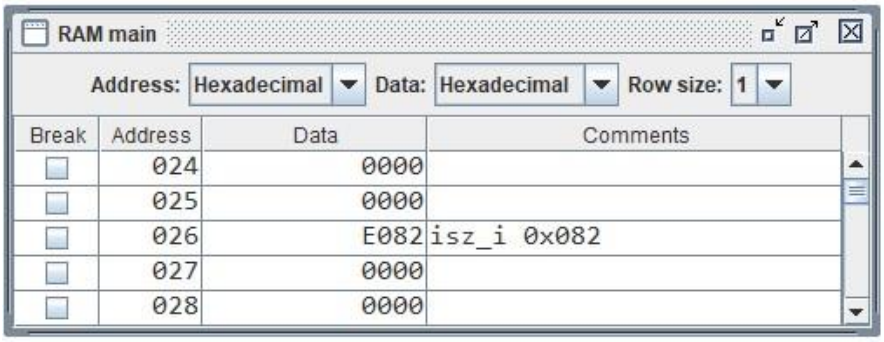
Address: Hexadecimal Data: Hexadecimal Row size: 1

Break	Address	Data	Comments
	296	0000	
	297	0000	
	298	0027	
	299	0000	
	29A	0000	

ISZ

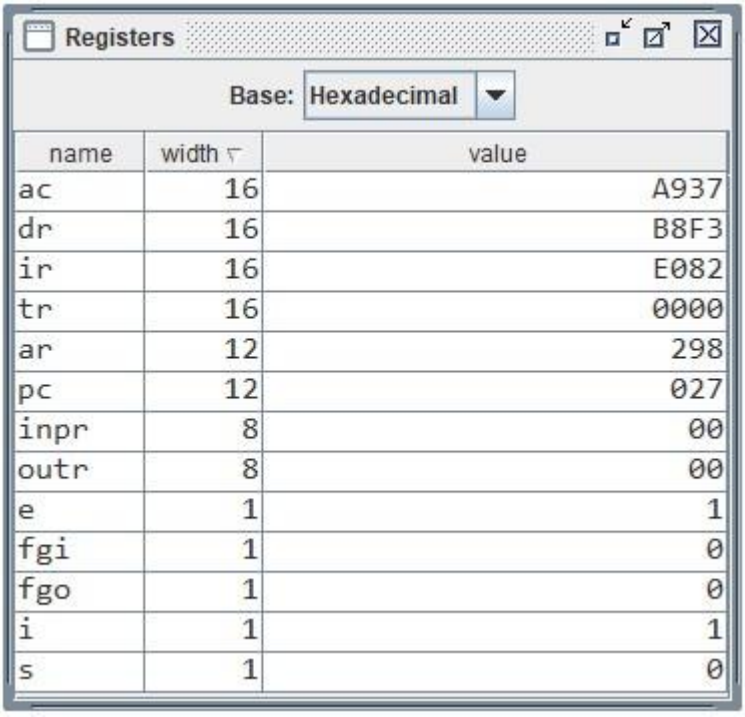
```
isz_i 0x082
```

Memory After Loading



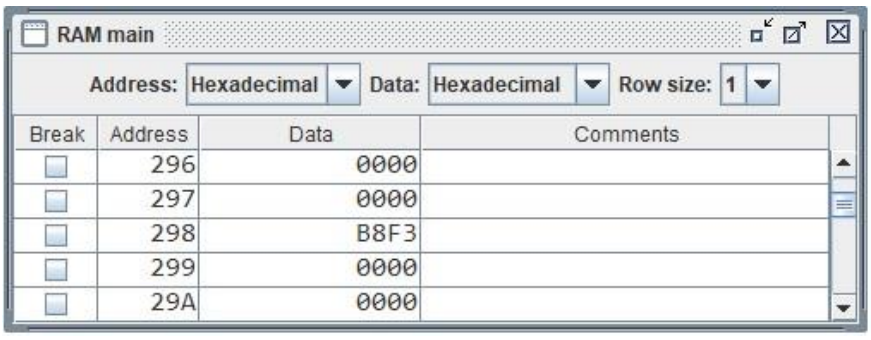
Break	Address	Data	Comments
	024	0000	
	025	0000	
	026	E082	isz_i 0x082
	027	0000	
	028	0000	

Registers After Execution



name	width	value
ac	16	A937
dr	16	B8F3
ir	16	E082
tr	16	0000
ar	12	298
pc	12	027
inpr	8	00
outr	8	00
e	1	1
fgi	1	0
fgo	1	0
i	1	1
s	1	0

Memory After Execution



Break	Address	Data	Comments
	296	0000	
	297	0000	
	298	B8F3	
	299	0000	
	29A	0000	

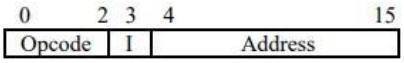
PRACTICAL 7

Dated 01st October 2019

Objective

Modify the machine created in Practical 1 according to the following instruction format:

Instruction format



(a) The instruction format contains a 3-bit opcode, a 1-bit addressing mode and a 12-bit address. There are only two addressing modes, I = 0 (direct addressing) and I = 1 (indirect addressing).

(b) Create a new register I of 1 bit.

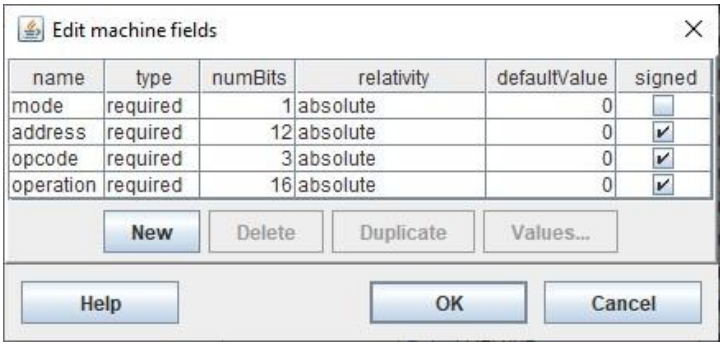
(c) Create two new microinstructions as follows:

i. Check the opcode of instruction to determine type of instruction (MemoryReference/Register-Reference/Input-Output) and then jump accordingly.

ii. Check the I bit to determine the addressing mode and then jump accordingly.

Observation

Machine Fields

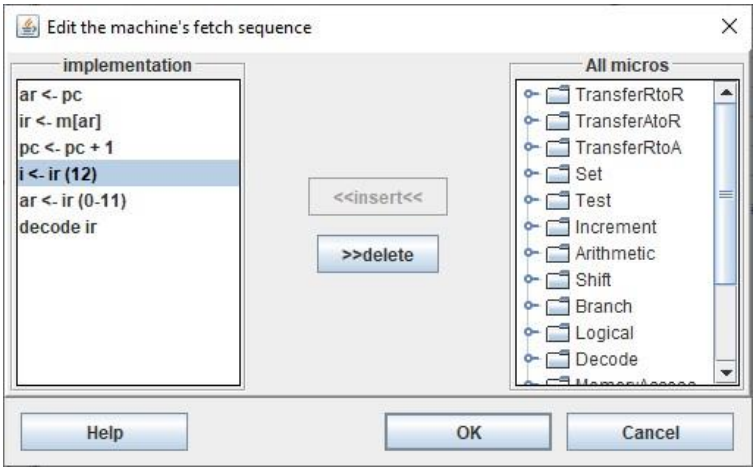


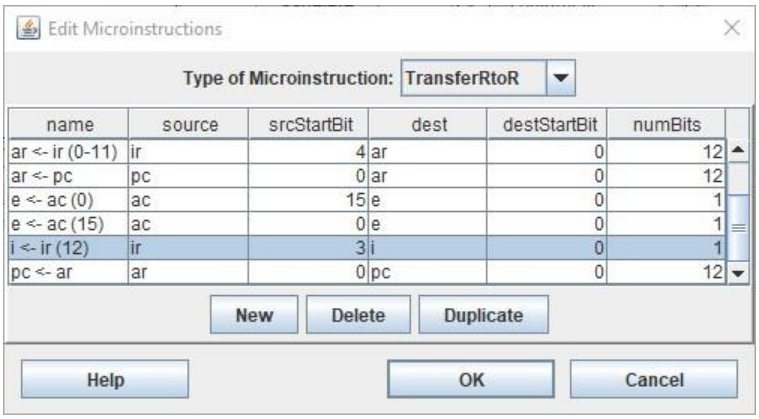
Instruction Format

isz	6	opcode mode address
bsa	5	opcode mode address
bun	4	opcode mode address
sta	3	opcode mode address
lda	2	opcode mode address
add	1	opcode mode address
and	0	opcode mode address

Registers

Fetch Sequence

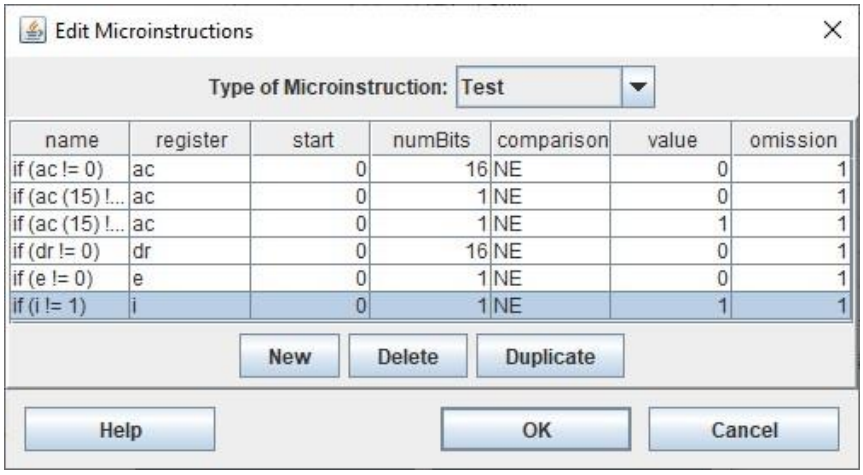




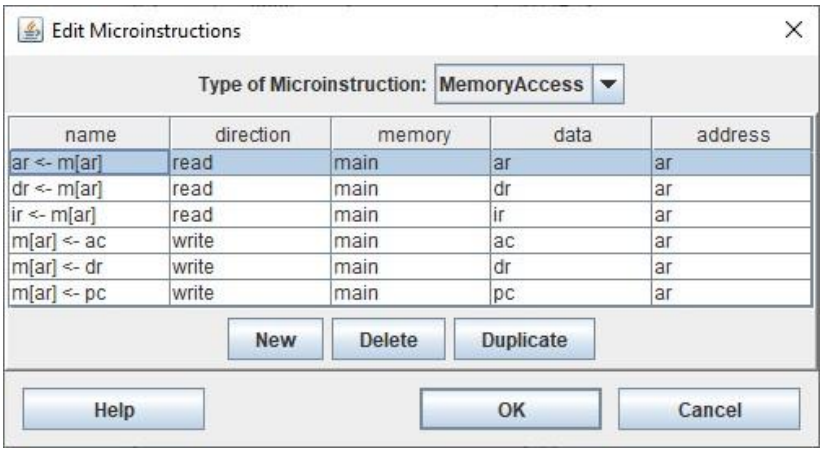
Extracting Mode Bit from IR

Checking Addressing Mode

- **<u>If (i != 1)</u>** (CPUSim skips a micro-operation when this is true)



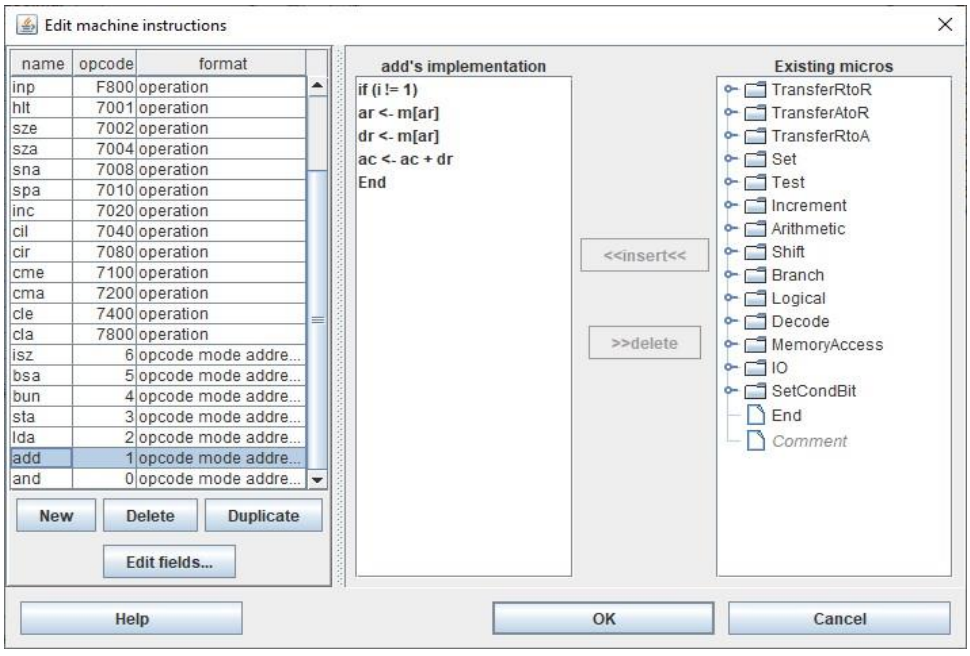
- **ar <- m[ar]**



- **other microinstructions follow**

Example

Implementation of ADD



Assembly Code

```
add 1 0x025
add 0 0x020
```

Memory After Loading

RAM main

Address: Hexadecimal Data: Hexadecimal Row size: 1

Address	Data	Comments
020	0001	
021	0000	
022	3025	add 1 0x025
023	2020	add 0 0x020
024	0000	
025	0020	

Registers After Execution of 0x022

Registers

Base: Hexadecimal

name	wid...	value
ac	16	A938
dr	16	0001
ir	16	3025
tr	16	0000
ar	12	020
pc	12	023
inpr	8	00
outr	8	00
e	1	1
fgi	1	0
fgo	1	0
i	1	1
s	1	0

Registers After Execution of 0x023

Registers

Base: Hexadecimal

name	wid...	value
ac	16	A939
dr	16	0001
ir	16	2020
tr	16	0000
ar	12	020
pc	12	024
inpr	8	00
outr	8	00
e	1	1
fgi	1	0
fgo	1	0
i	1	0
s	1	0