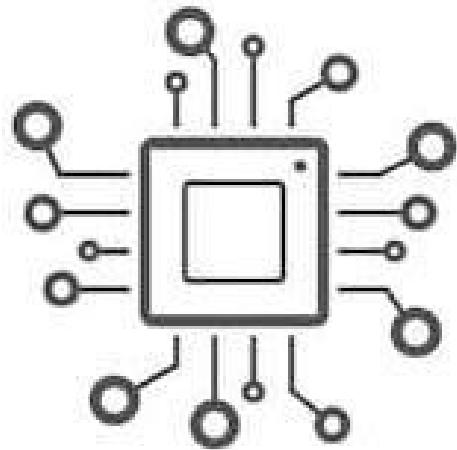




Atma Ram Sanatan Dharma College University of Delhi



Computer System Architecture Practical File



Submitted By

KHUSHAL SACHDEVA

88044

Bsc. (Hons.) Computer Science (SEM - 1)

Submitted To

Ms Uma Ojha

Department of Computer Science

Create a machine based on the following architecture :

REGISTERS

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

Memory and Instruction Format

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

Basic Computer Instructions

Symbol	Hexadecimal code		Description
	<i>I</i> = 0	<i>I</i> = 1	
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

Edit Modules

Type of Module: Register

name	width	initial value	read-only
AC	16	0	<input type="checkbox"/>
AR	12	0	<input type="checkbox"/>
DR	16	0	<input type="checkbox"/>
E	1	0	<input type="checkbox"/>
I	1	0	<input type="checkbox"/>
INPR	8	0	<input type="checkbox"/>
IR	16	0	<input type="checkbox"/>
OUTR	8	0	<input type="checkbox"/>
PC	12	0	<input type="checkbox"/>
S	1	0	<input type="checkbox"/>
TR	16	0	<input type="checkbox"/>

New Delete Duplicate Properties...

?

OK Cancel

Memory

Edit Modules

Type of Module: RAM

name	length	cellSize
main	4096	16

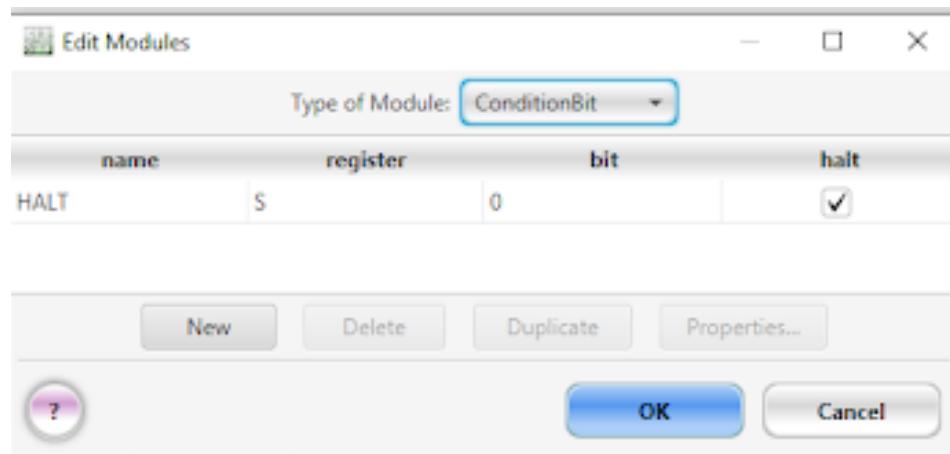
New Delete Duplicate Properties...

?

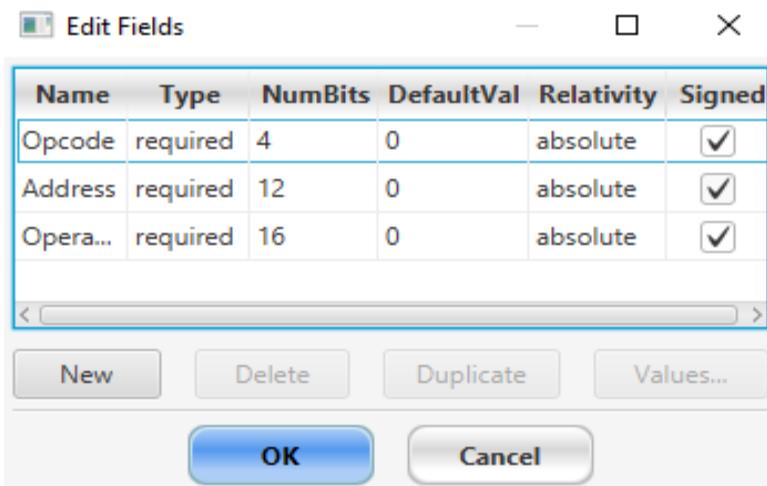
OK Cancel



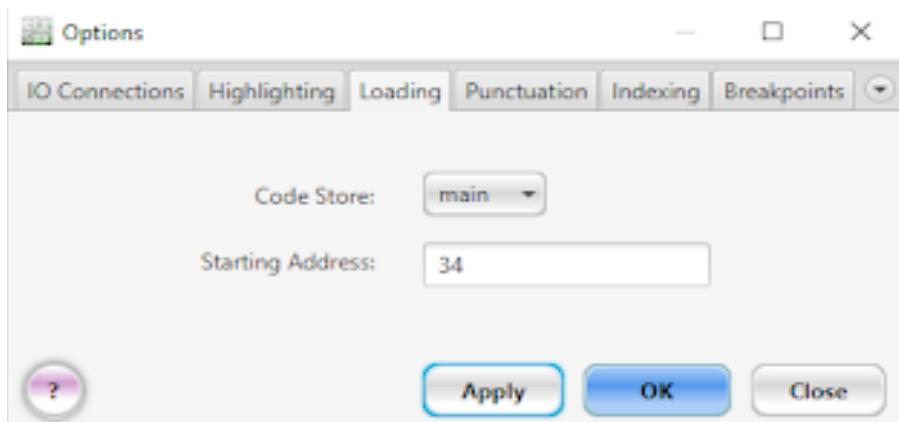
Condition Bits



Instruction Formats



Loading Point



Create the micro-operations and associate with instructions as given in the list of basic computer instructions:

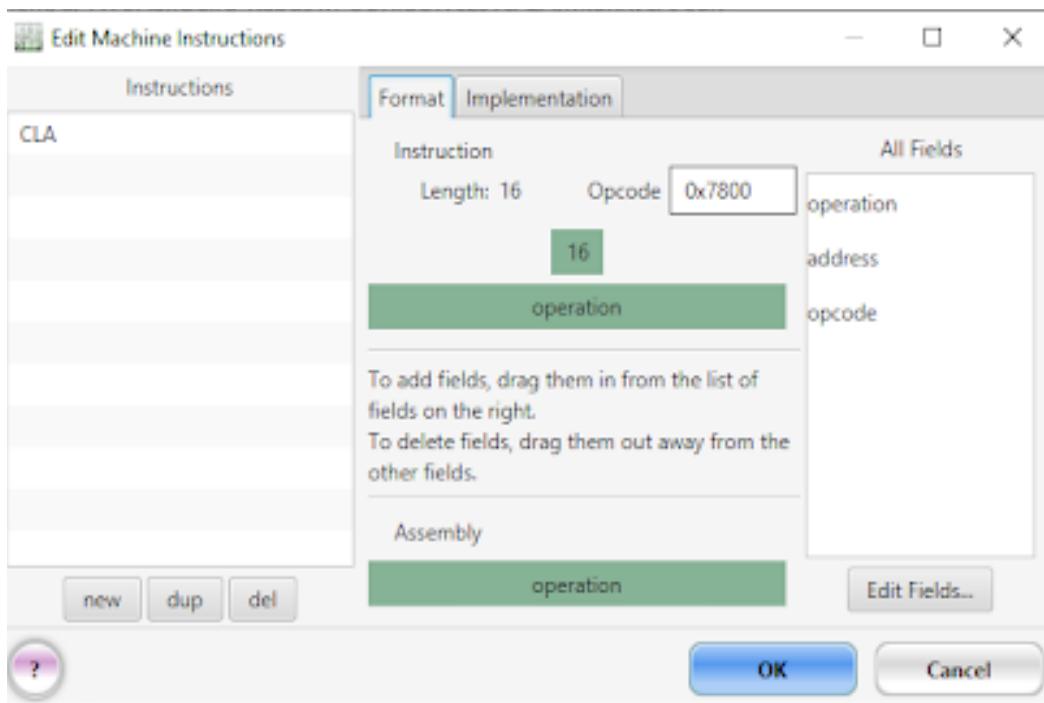
Register Reference Instructions

$D_2IT_3 = r$ (common to all register-reference instructions)
 $IR(i) = B_i$ [bit in $IR(0-11)$ that specifies the operation]

CLA	$rB_{15}: SC \leftarrow 0$	Clear SC
CLE	$rB_{15}: AC \leftarrow 0$	Clear AC
CMA	$rB_7: E \leftarrow 0$	Clear E
CMB	$rB_7: AC \leftarrow \overline{AC}$	Complement AC
CIR	$rB_7: AC \leftarrow shr\ AC, AC(15) \leftarrow E, E \leftarrow AC(0)$	Circulate right
CIL	$rB_7: AC \leftarrow shl\ AC, AC(0) \leftarrow E, E \leftarrow AC(15)$	Circulate left
INC	$rB_7: AC \leftarrow AC + 1$	Increment AC
SPA	$rB_7: If (AC(15) = 0) then (PC \leftarrow PC + 1)$	Skip if positive
SNA	$rB_7: If (AC(15) = 1) then (PC \leftarrow PC + 1)$	Skip if negative
SZA	$rB_7: If (AC = 0) then PC \leftarrow PC + 1$	Skip if AC zero
SZE	$rB_7: If (E = 0) then (PC \leftarrow PC + 1)$	Skip if E zero
HLT	$rB_7: S \leftarrow 0$ (S is a start-stop flip-flop)	Halt computer

CLA – Clear AC

Implementation:



- AC \leftarrow 0

Edit Microinstructions

Type of Microinstruction: Set

name	register	start	numBits	value
AC<-0	AC	0	16	0
E<-0	E	0	1	0

New Delete Duplicate

OK Cancel

Edit Machine Instructions

Instructions Format Implementation

CLA

Execute sequence

AC<-0
End

MicrInstructions

- MicroInstructions
 - arithmetic
 - branch
 - decode
 - end
 - comment
 - increment
 - io
 - logical
 - memoryAccess
 - set
 - setCondBit
 - shift
 - test
 - transferRtoR
 - transferRtoA
 - transferAtoR

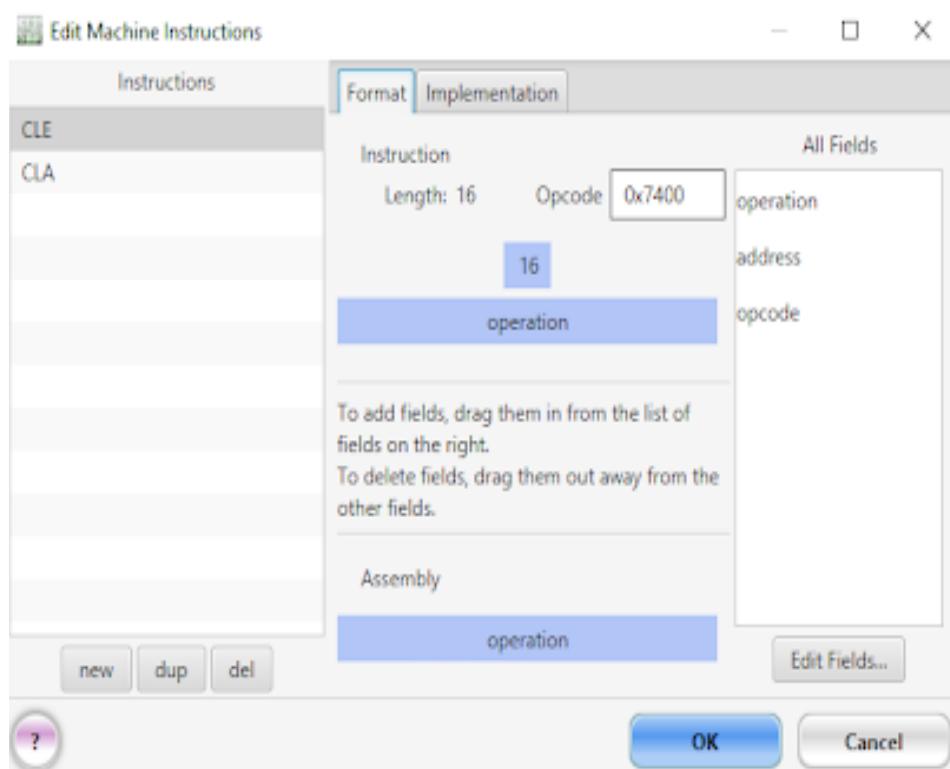
new dup del

OK Cancel

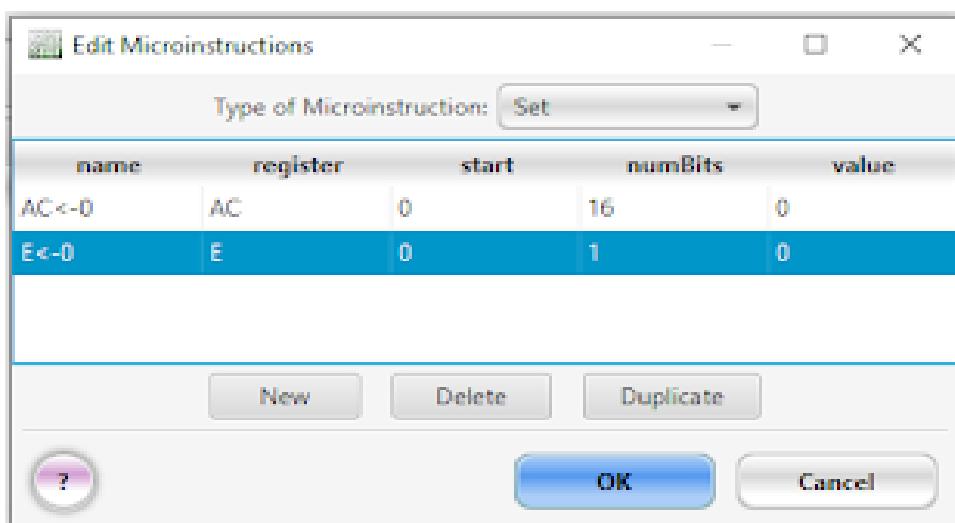


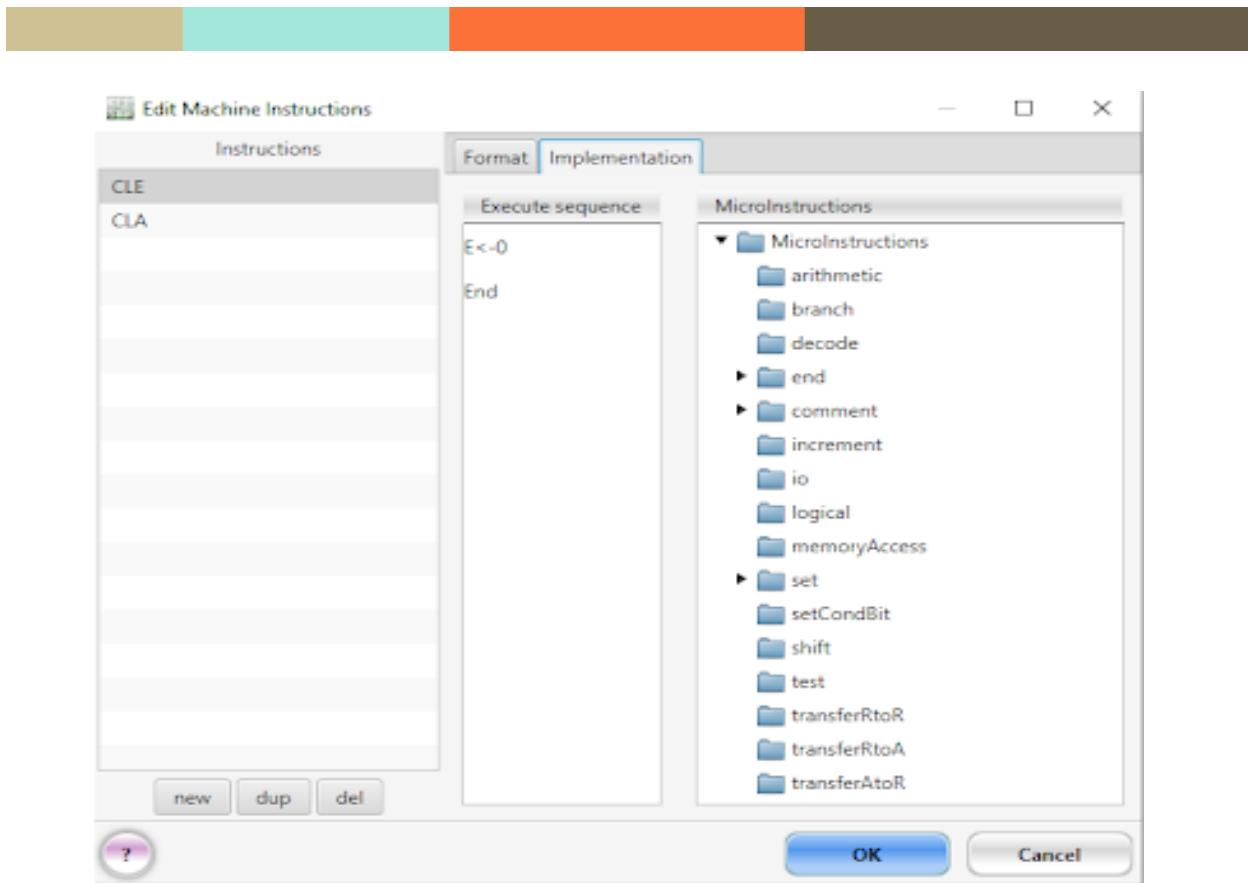
CLE – Clear E

Implementation:



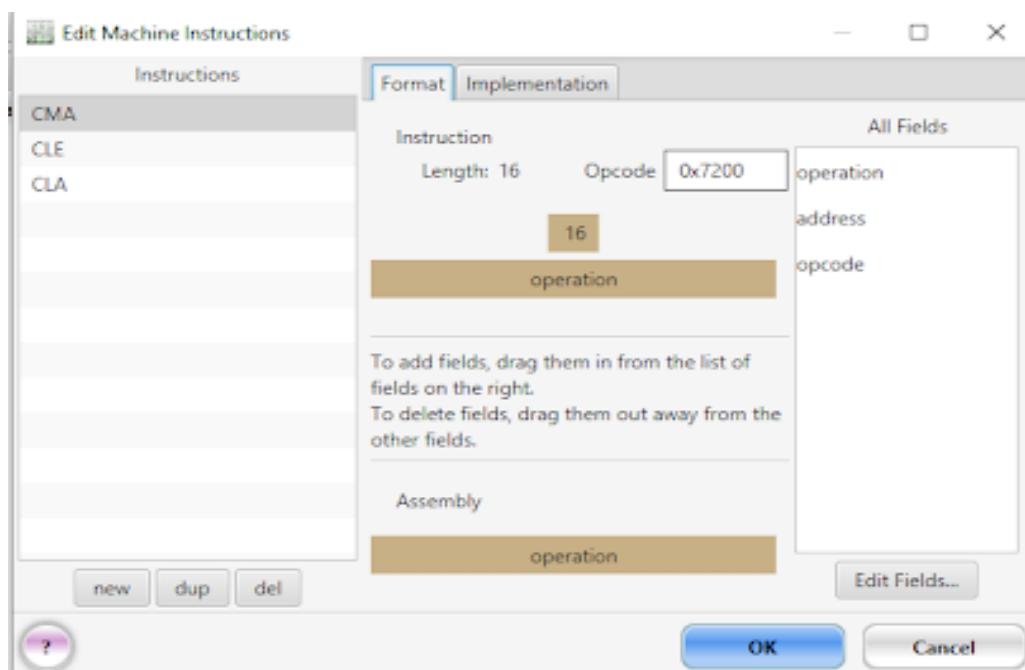
- E \leftarrow 0



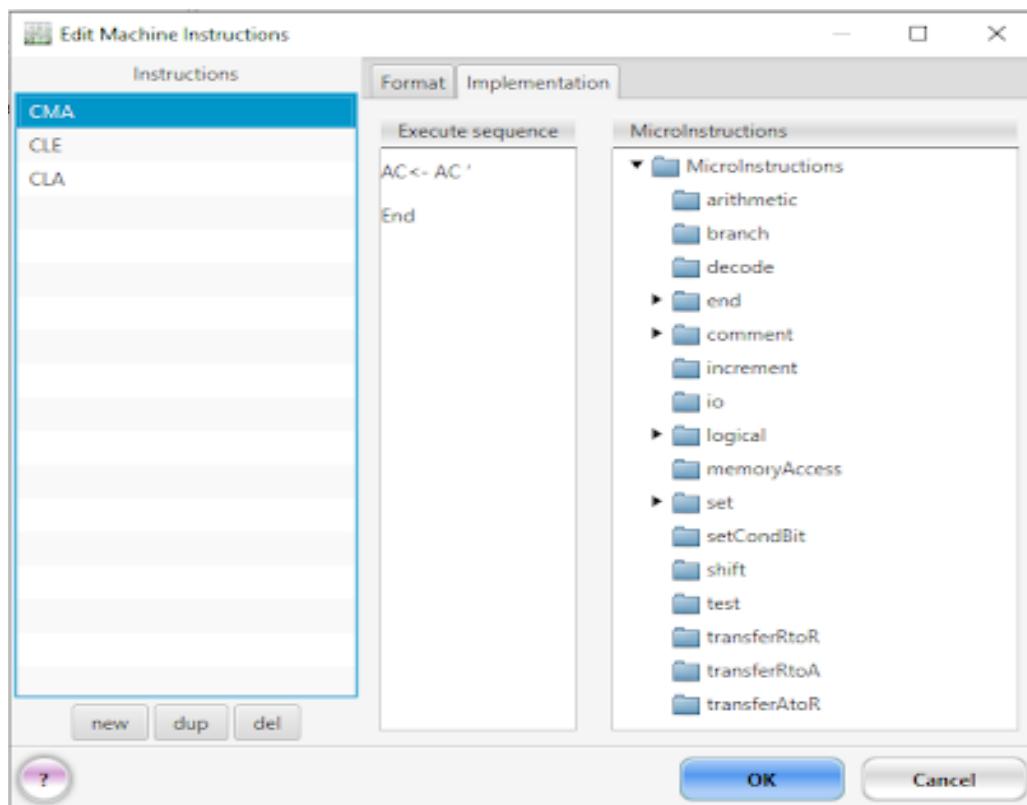
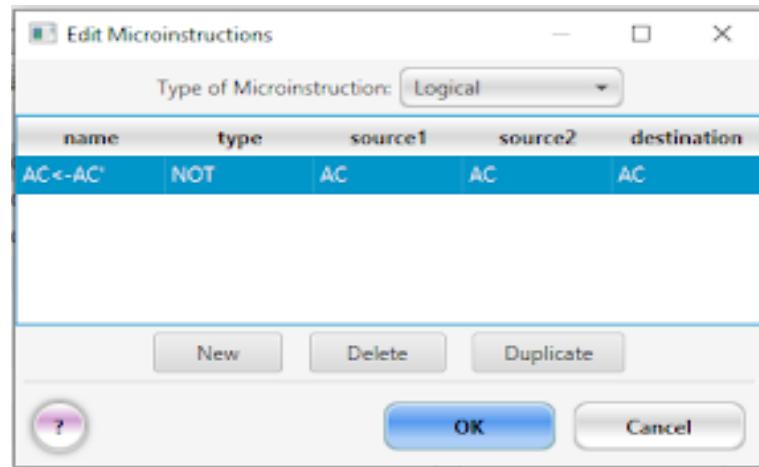


CMA – Complement AC

Implementation:



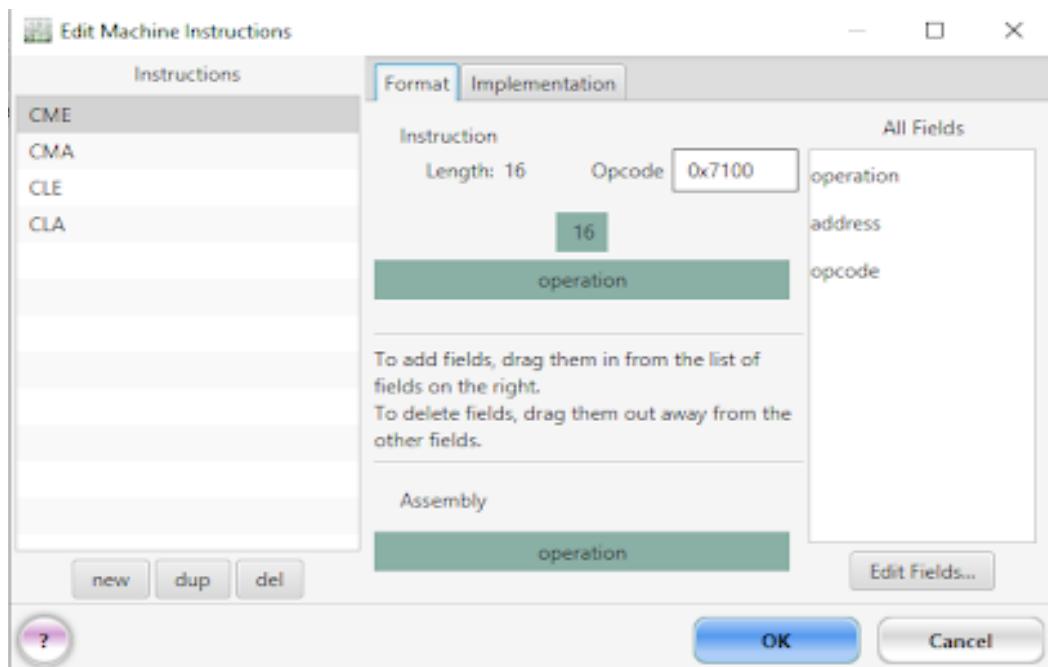
- $AC \leftarrow AC'$





CME – Complement E

Implementation:

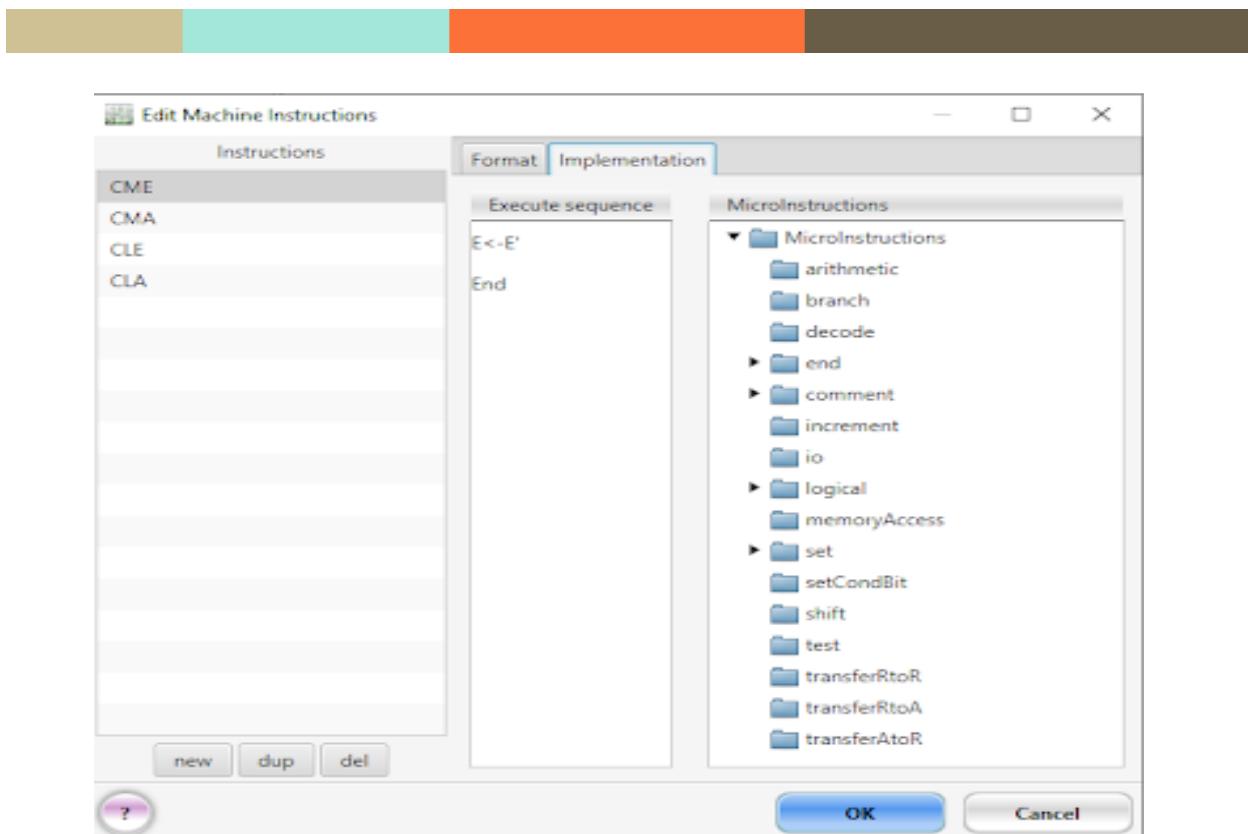


- $E \leftarrow E'$

Screenshot of the 'Edit Microinstructions' dialog box. The 'Logical' type is selected. The table shows two microinstructions:

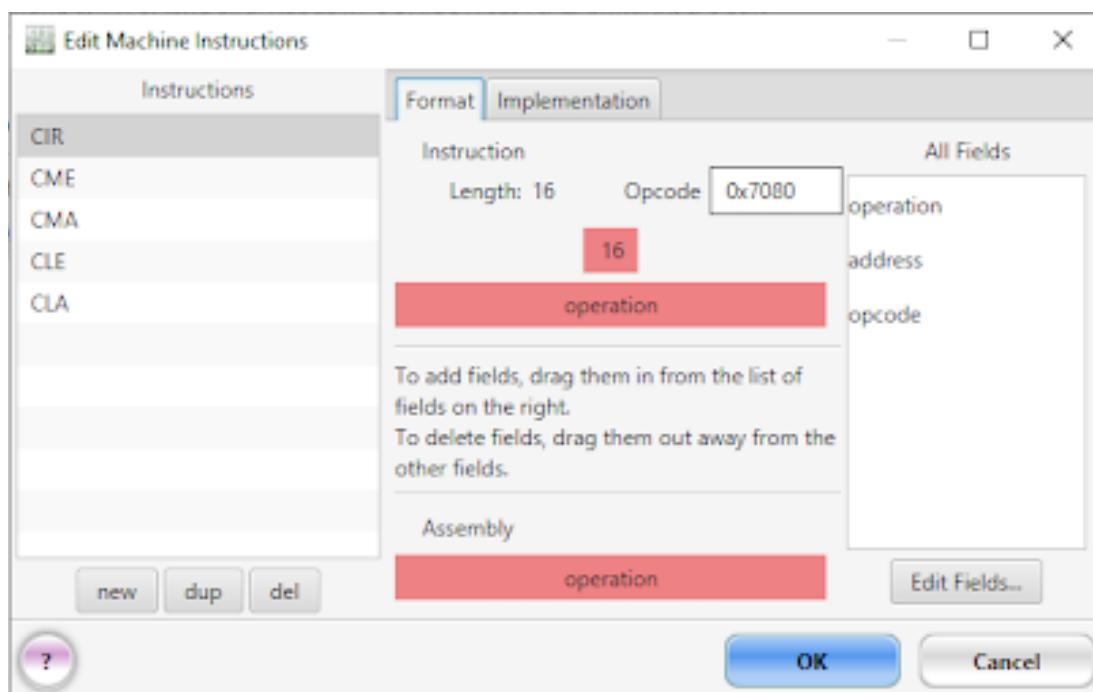
name	type	source1	source2	destination
$E \leftarrow E'$	NOT	E	E	E
AC->AC'	NOT	AC	AC	AC

Buttons at the bottom include New, Delete, Duplicate, ?, OK, and Cancel.



CIR - Circulate Right

Implementation:



- $E \leftarrow AC(0)$

Edit Microinstructions

Type of Microinstruction: TransferRtoR

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

New Delete Duplicate

?

OK Cancel

- $AC \leftarrow shr AC$

Edit Microinstructions

Type of Microinstruction: Shift

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

New Delete Duplicate

?

OK Cancel

- AC(15) <- E

Edit Microinstructions

Type of Microinstruction: TransferRtoR

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

New Delete Duplicate

?

OK Cancel

Edit Machine Instructions

Instructions: CIR

Format Implementation

CL	E<-AC(0)	MicrolInstructions
CIR	AC<-shr AC	arithmetic
CME	AC(15)<-E	branch
CMA	End	decode
CLE		end
CLA		comment

new dup del

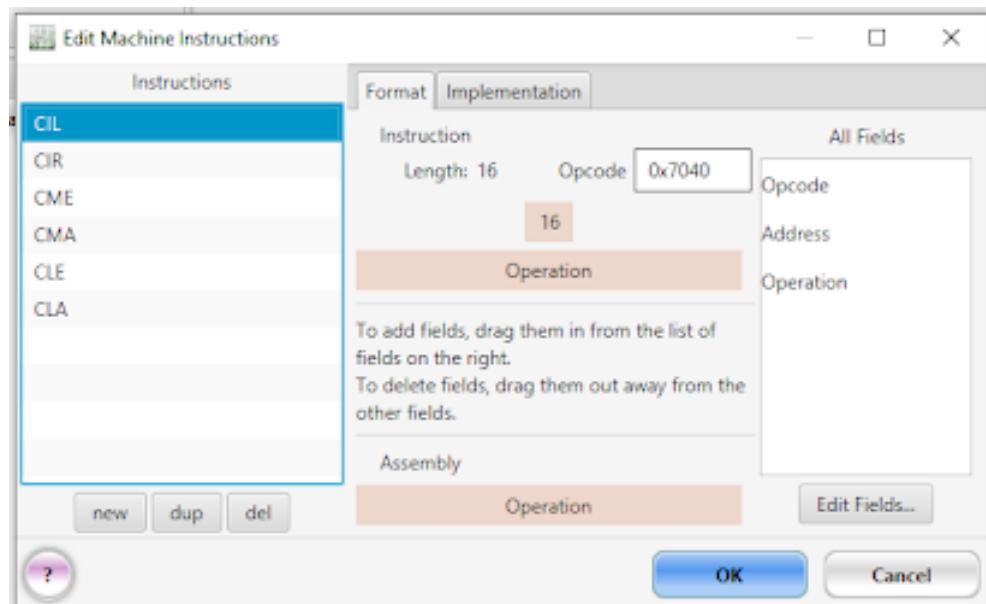
?

OK Cancel



CIL - Circulate Left

Implementation:



- E \leftarrow AC(15)

Edit Microinstructions

Type of Microinstruction: TransferRtoR

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

New Delete Duplicate

OK Cancel

- $AC \leftarrow shl\ AC$

Edit Microinstructions

Type of Microinstruction: Shift

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

New Delete Duplicate

OK Cancel

- $AC(0) \leftarrow E$

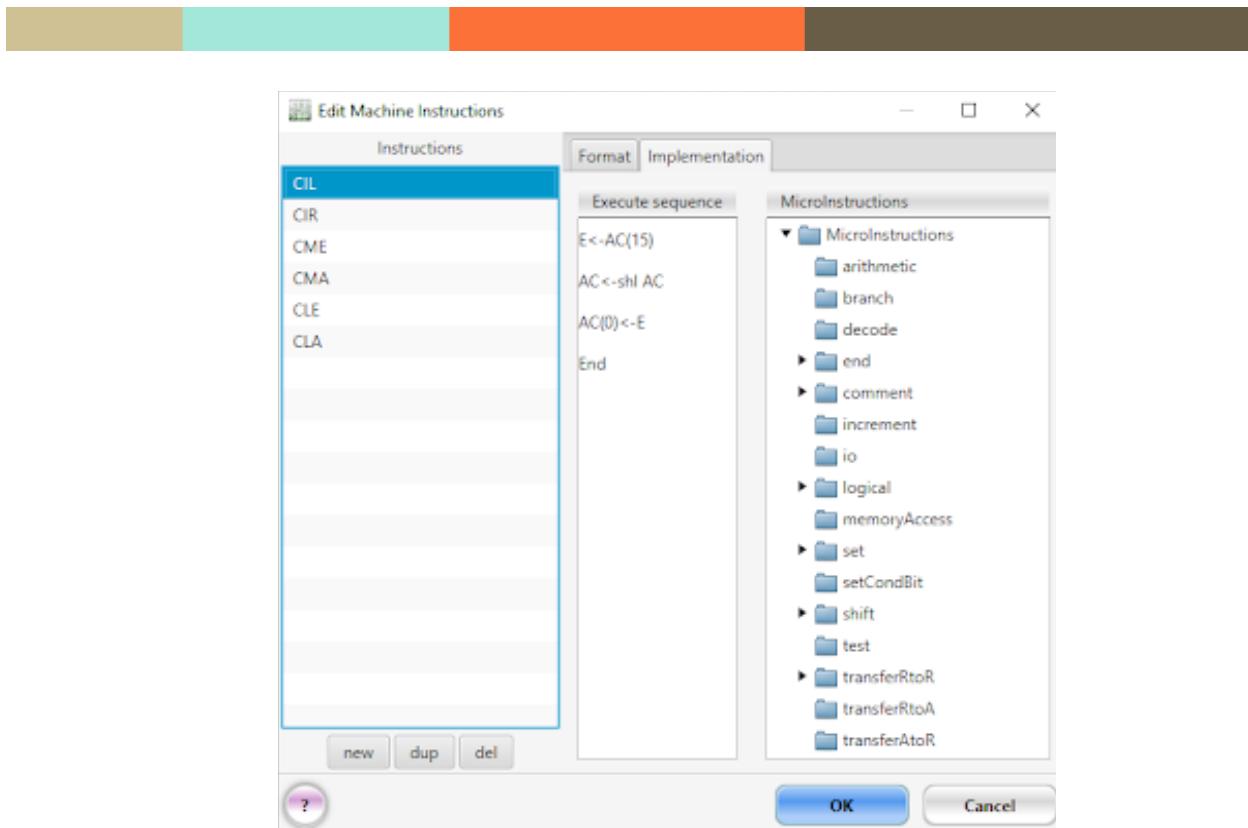
Edit Microinstructions

Type of Microinstruction: TransferRtoR

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

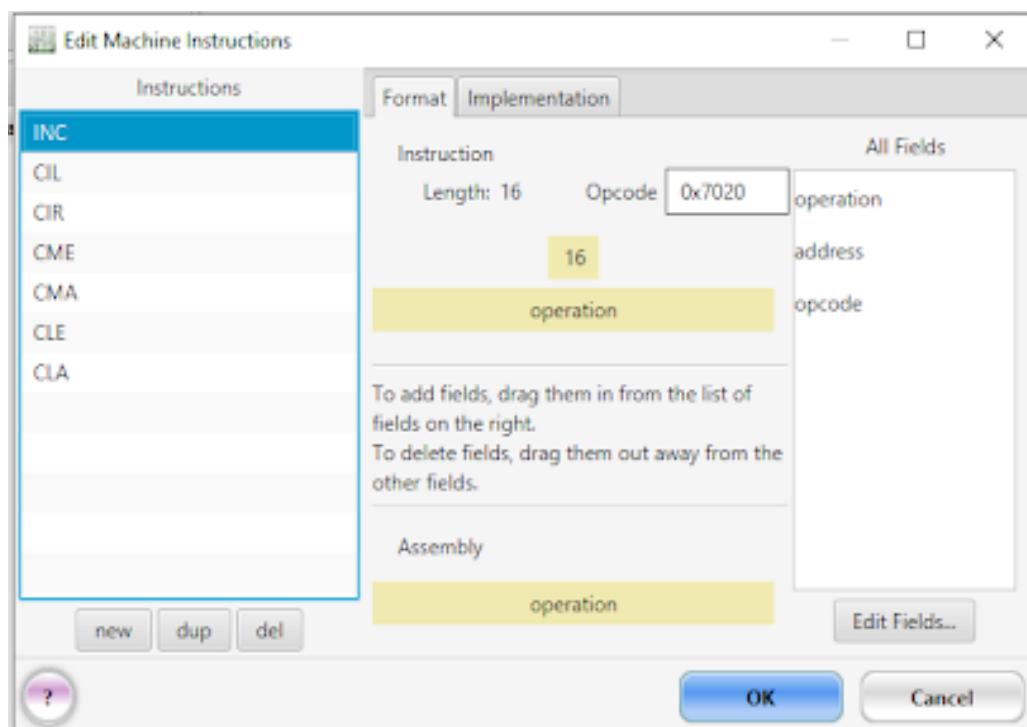
New Delete Duplicate

OK Cancel

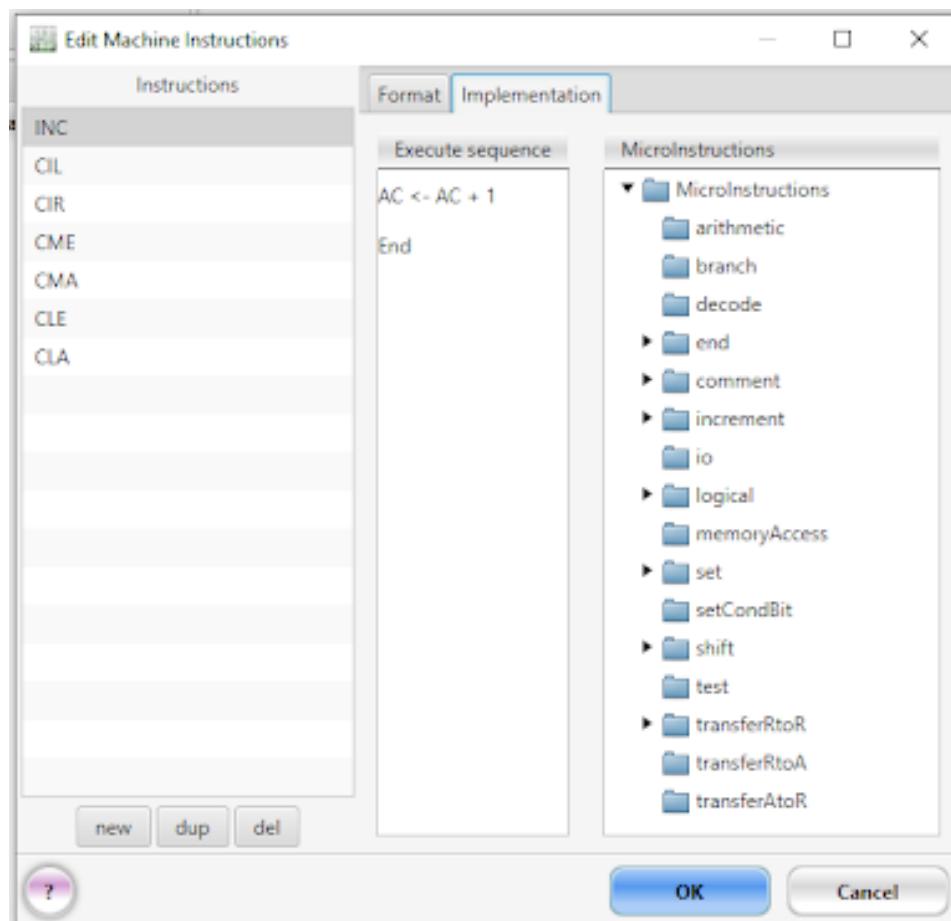
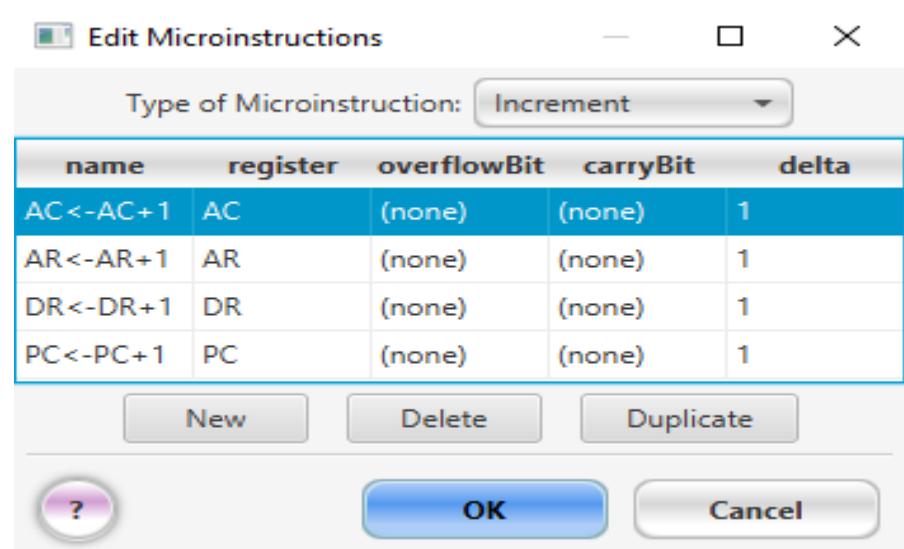


INC – Increment AC

Implementation:



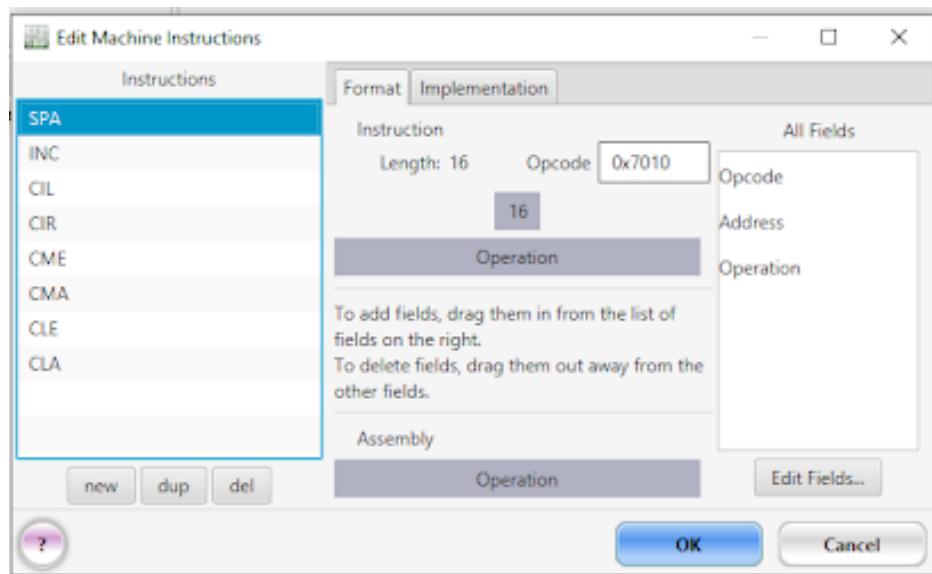
- $AC \leftarrow AC + 1$



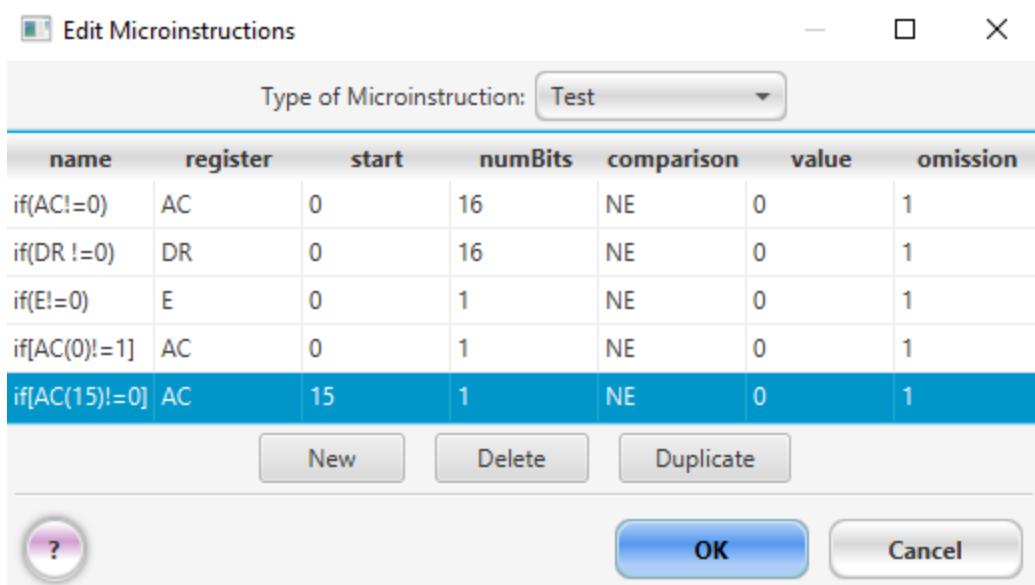


SPA – Skip If Positive

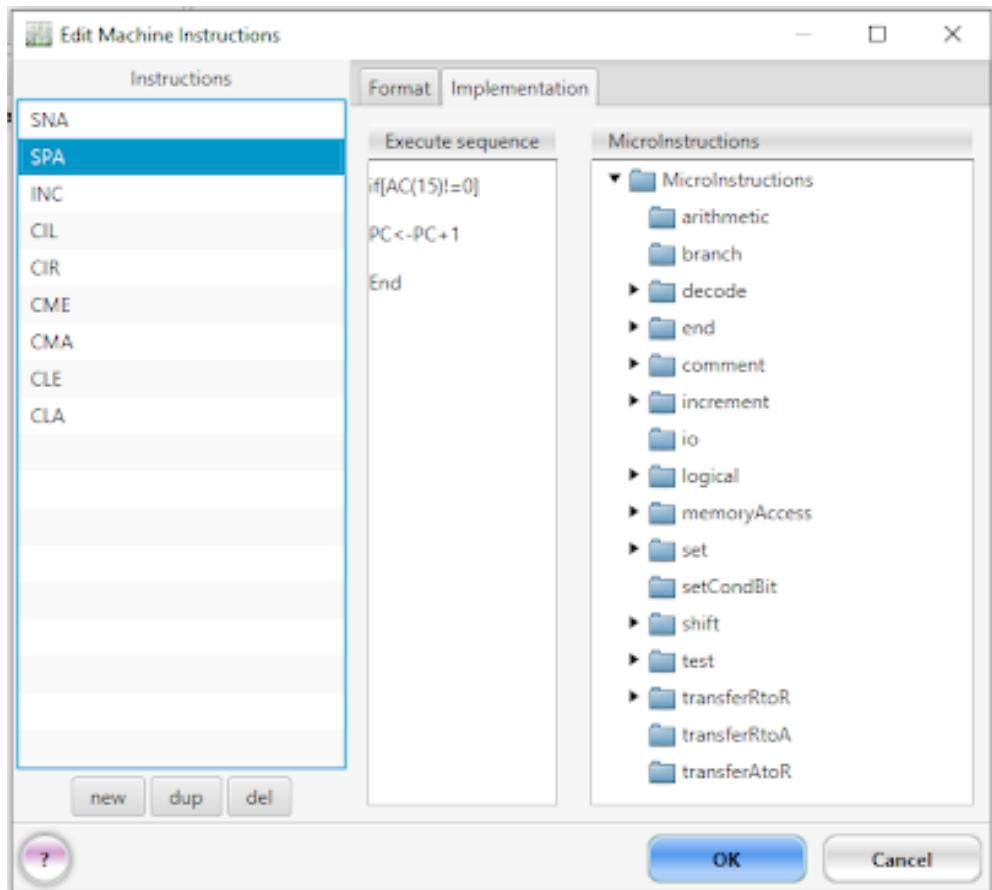
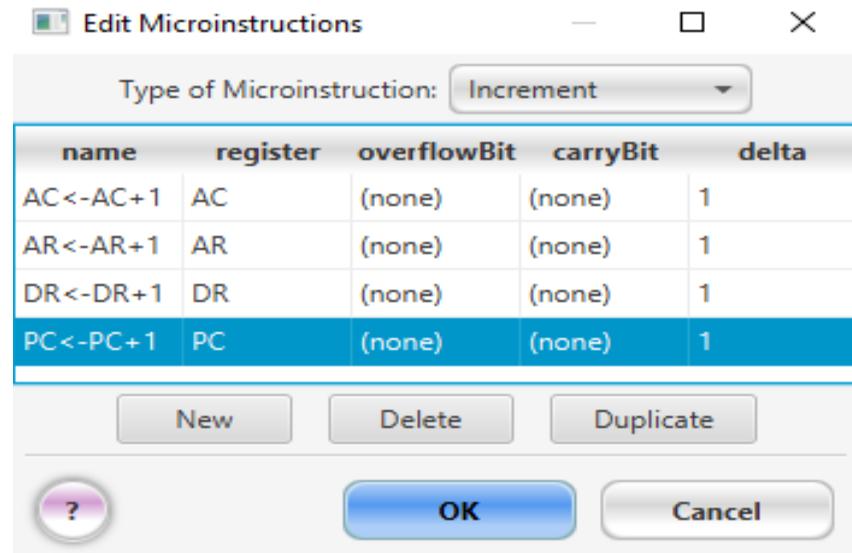
Implementation:



- if [AC(15) != 0] (**CPUSim skips a micro-operation when this is true**)



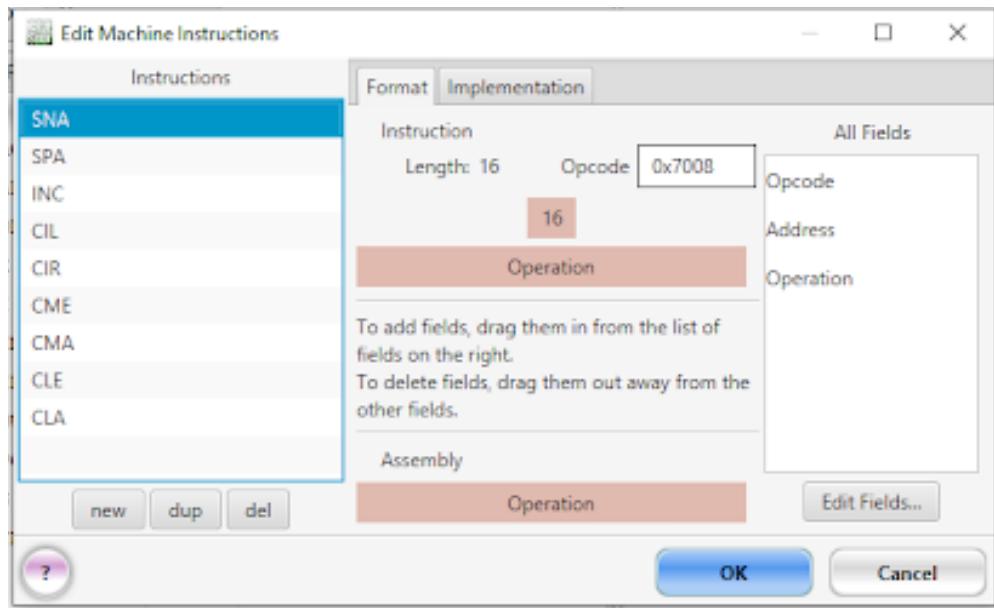
- $PC \leftarrow PC + 1$





SNA – Skip If Negative

Implementation:



- if [AC(15) !=1]

Edit Microinstructions

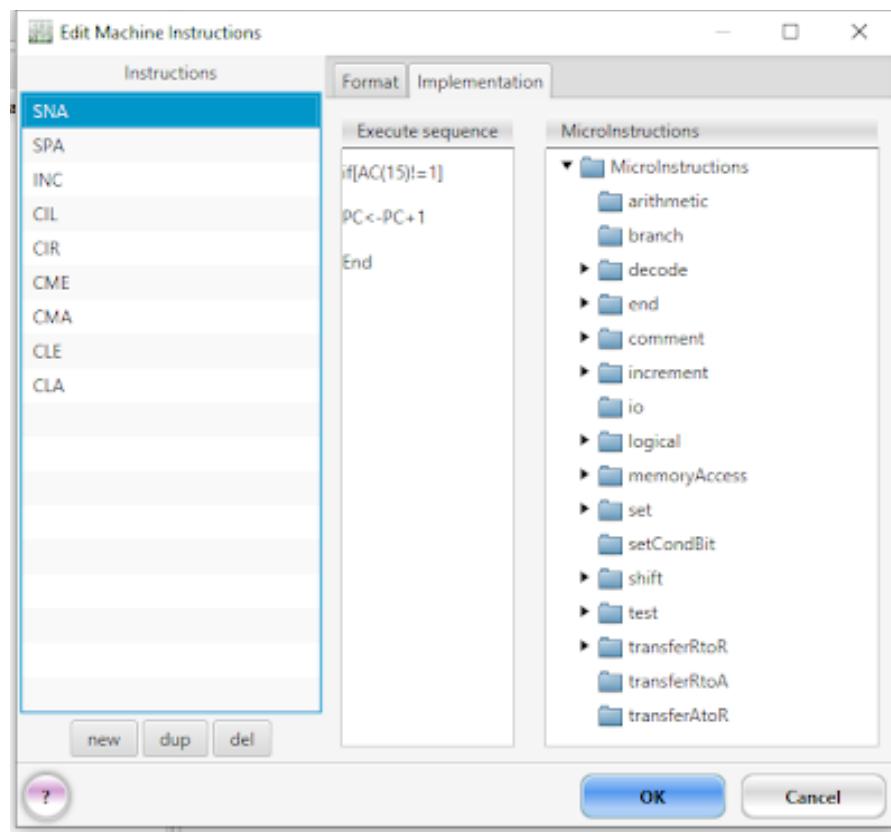
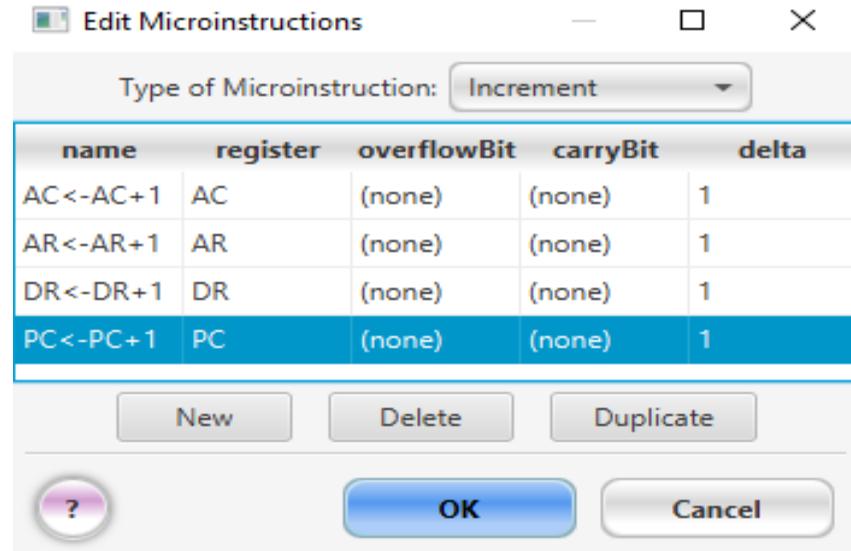
Type of Microinstruction: Test

name	register	start	numBits	comparison	value	omission
if(AC!=0)	AC	0	16	NE	0	1
if(DR !=0)	DR	0	16	NE	0	1
if(E!=0)	E	0	1	NE	0	1
if[AC(15)!=1]	AC	15	1	NE	0	1
if[AC(15)!=0]	AC	15	1	NE	0	1

New Delete Duplicate

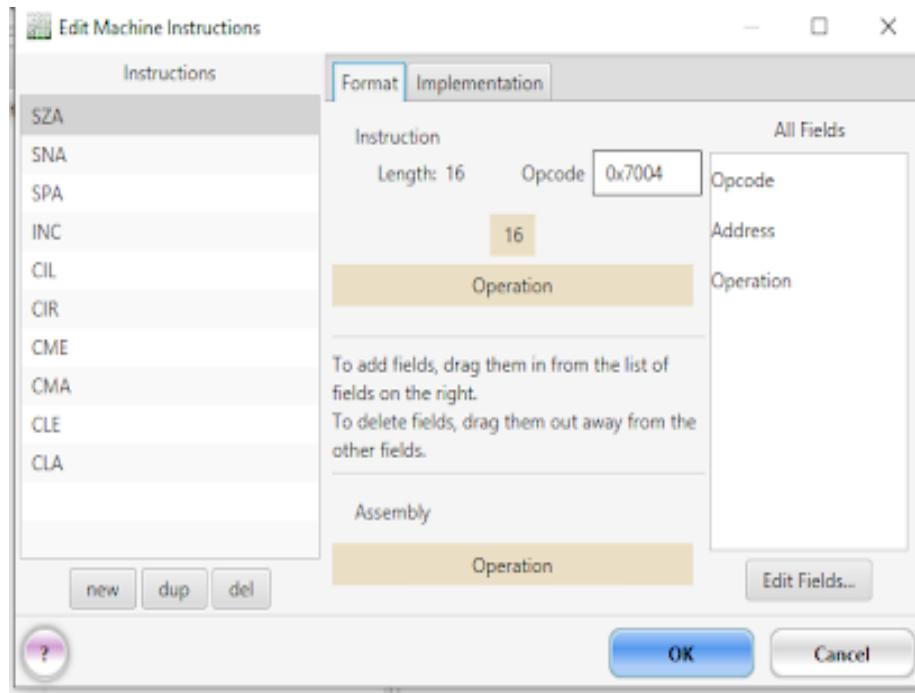
OK Cancel

- $PC \leftarrow PC + 1$



SZA – Skip If AC Zero

Implementation:



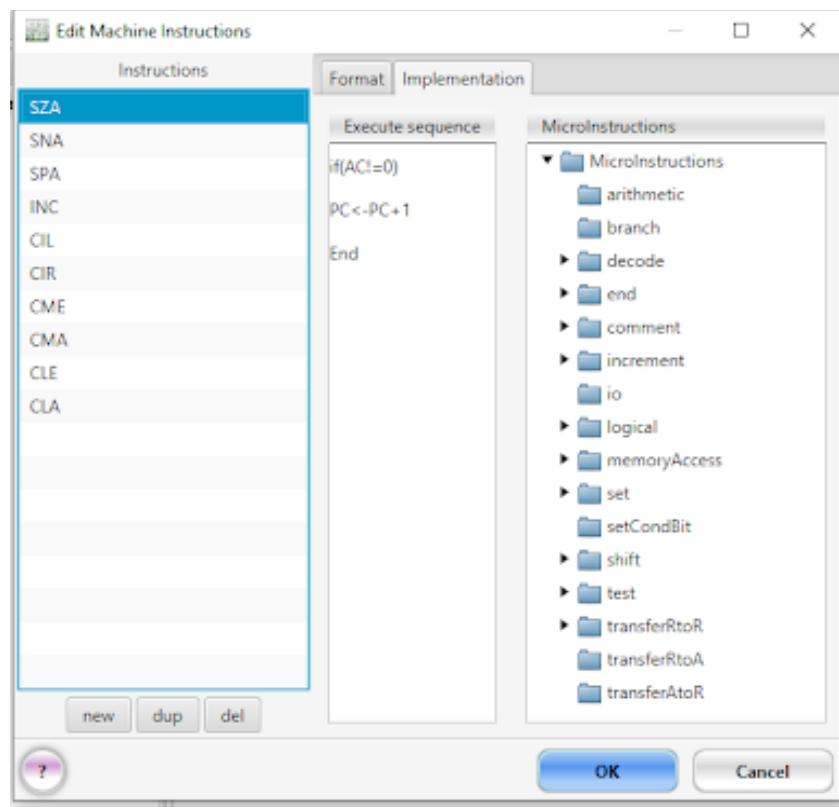
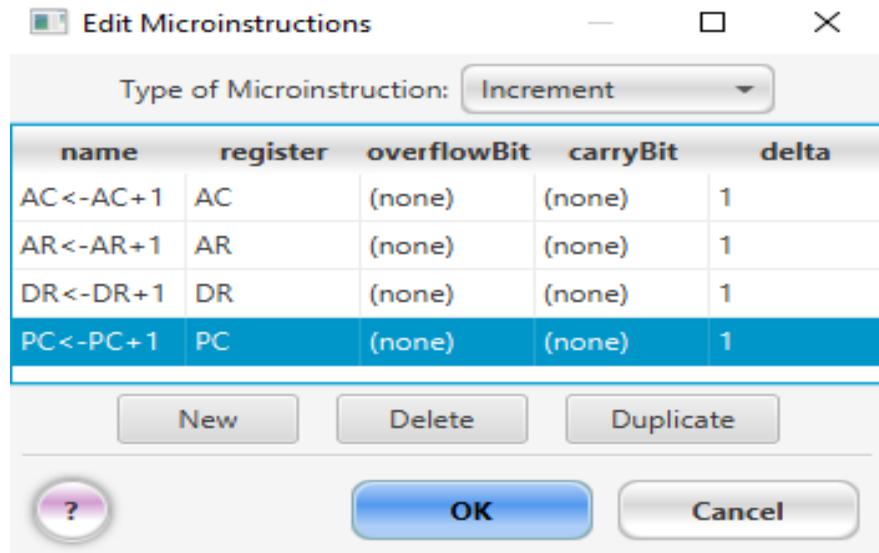
- If $AC \neq 0$ (CPUSim skips a micro-operation when this is true)

The screenshot shows the 'Edit Microinstructions' dialog box. The 'Type of Microinstruction' dropdown is set to 'Test'. Below is a table of microinstructions:

name	register	start	numBits	comparison	value	omission
if($AC \neq 0$)	AC	0	16	NE	0	1
if($DR \neq 0$)	DR	0	16	NE	0	1
if($E \neq 0$)	E	0	1	NE	0	1
if($AC(15) \neq 0$)	AC	15	1	NE	0	1
if($AC(15) \neq 1$)	AC	15	1	NE	0	1

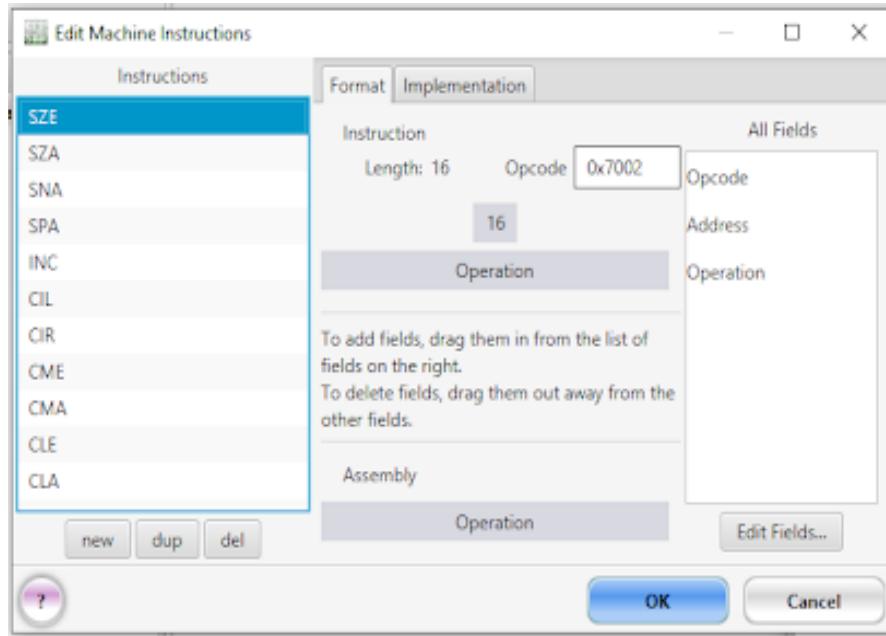
At the bottom, there are 'New', 'Delete', and 'Duplicate' buttons, followed by 'OK' and 'Cancel' buttons.

- $PC \leftarrow PC + 1$



SZE – Skip If E Zero

Implementation:



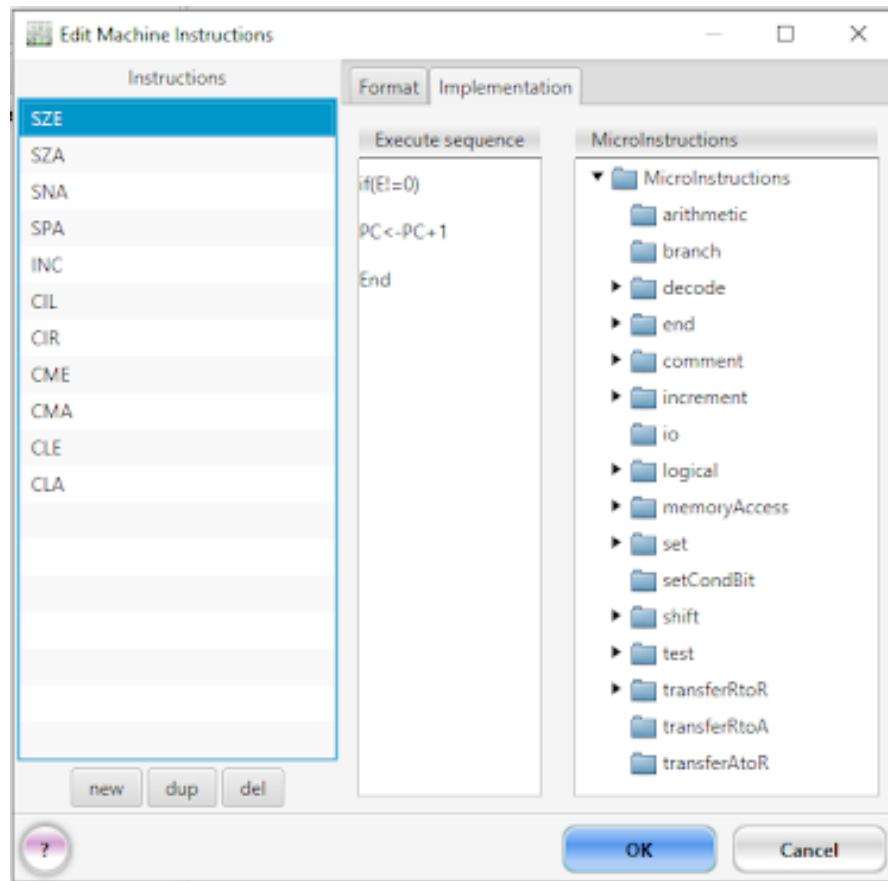
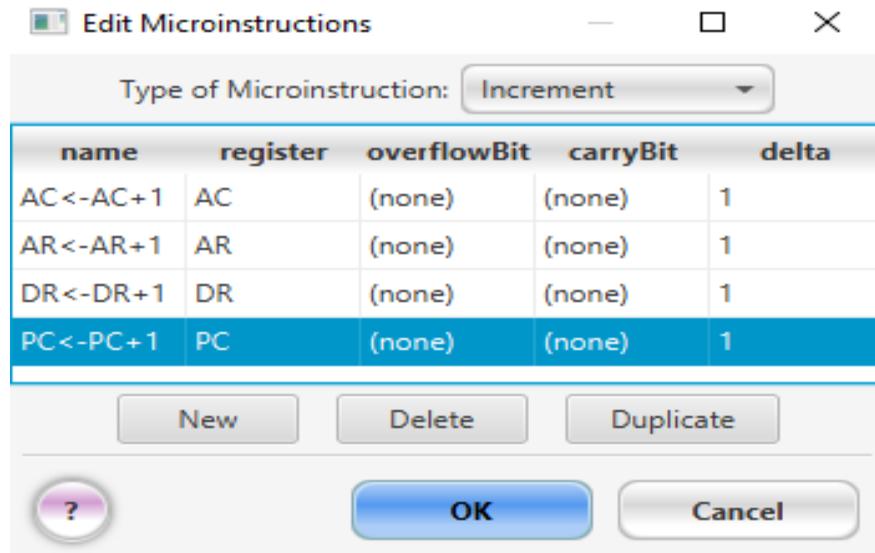
- If $E \neq 0$ (CPUSim skips a micro-operation when this is true)

The screenshot shows the 'Edit Microinstructions' dialog box. The 'Type of Microinstruction' dropdown is set to 'Test'. The main area is a table with the following columns: name, register, start, numBits, comparison, value, and omission. The rows are:

name	register	start	numBits	comparison	value	omission
if(AC!=0)	AC	0	16	NE	0	1
if(DR !=0)	DR	0	16	NE	0	1
if(E!=0)	E	0	1	NE	0	1
if[AC(15)!=0]	AC	15	1	NE	0	1
if[AC(15)!=1]	AC	15	1	NE	0	1

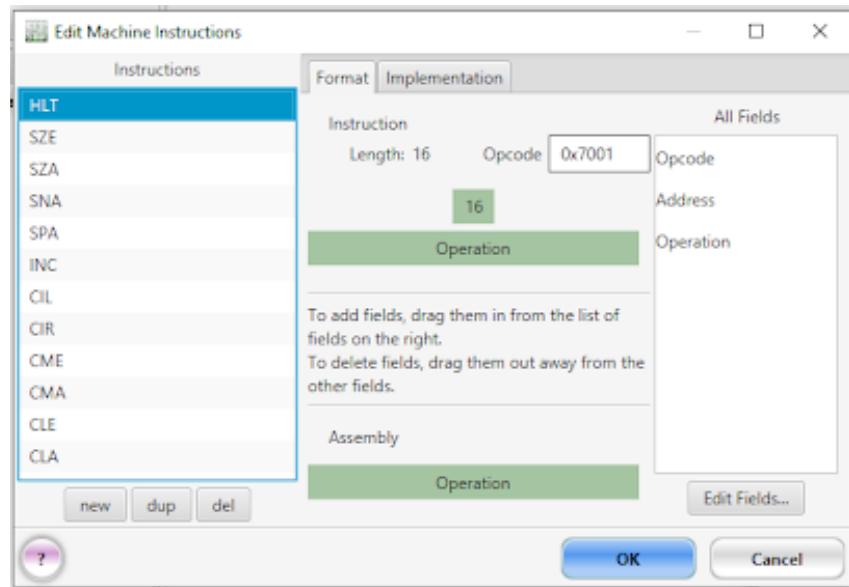
At the bottom are 'New', 'Delete', and 'Duplicate' buttons, followed by 'OK' and 'Cancel' buttons.

- $PC \leftarrow PC + 1$

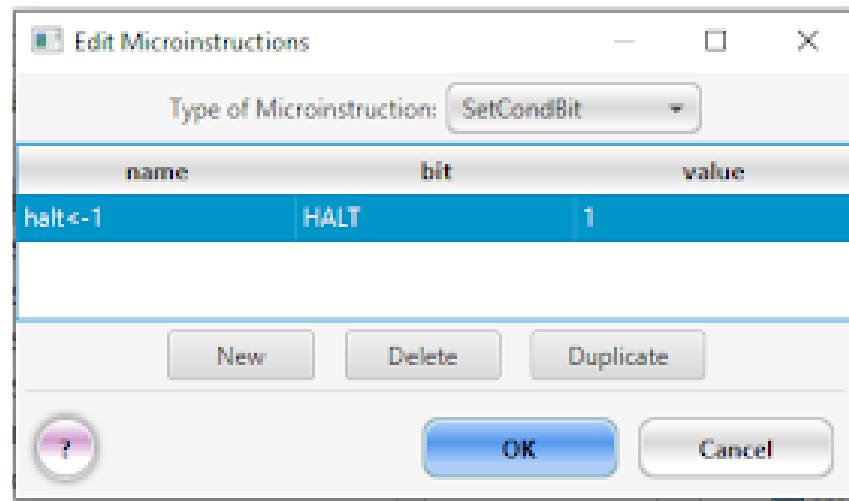


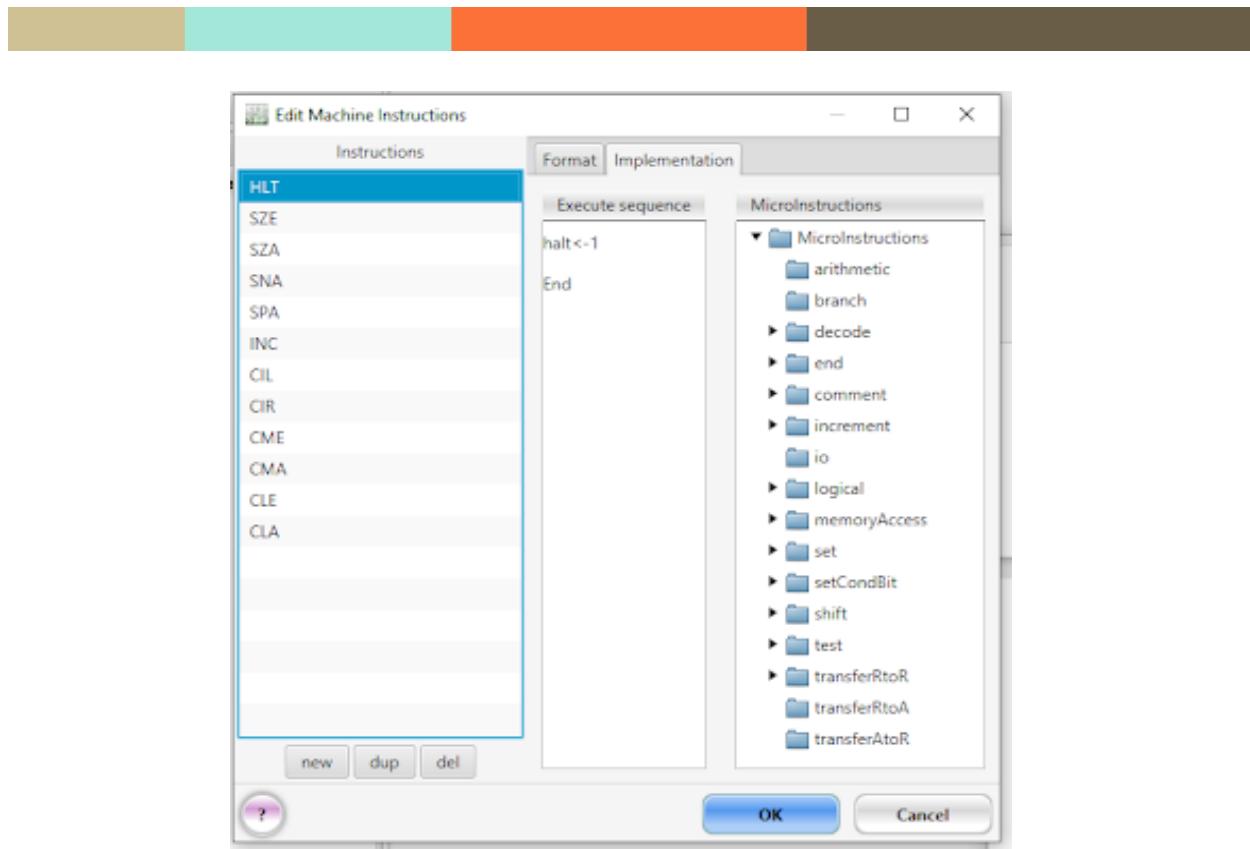
HLT – Halt Computer

Implementation:



- HALT <- 1



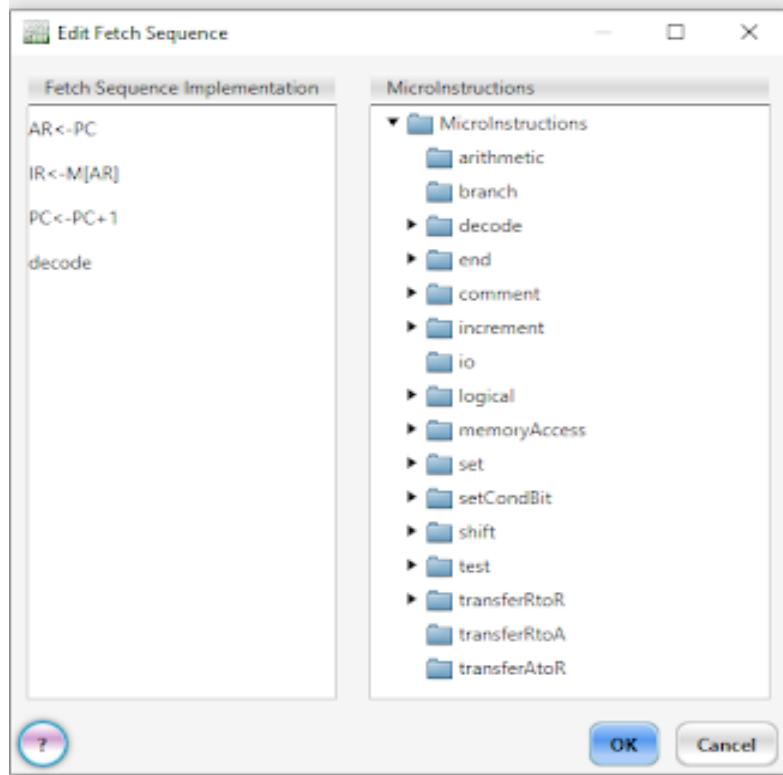




Create the fetch routine for the basic computer as designed in the previous practical.

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

Fetch Sequence



Implementation of the Fetch Sequence:

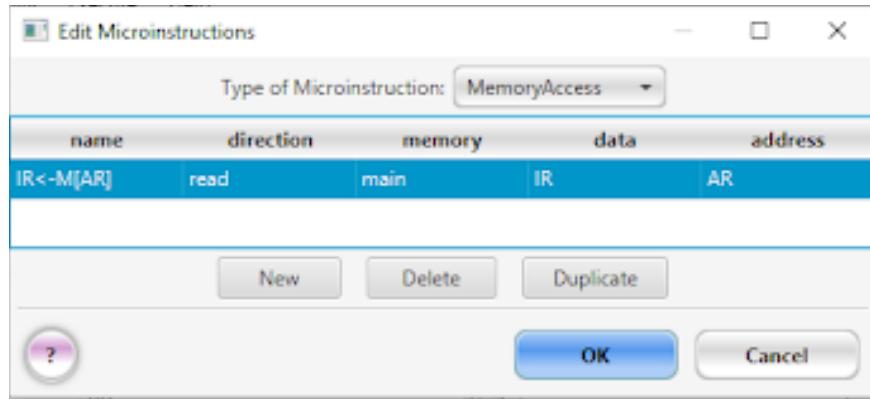
- AR <- PC

Screenshot of the 'Edit Microinstructions' dialog box. The table lists microinstructions of type 'TransferRtoR':

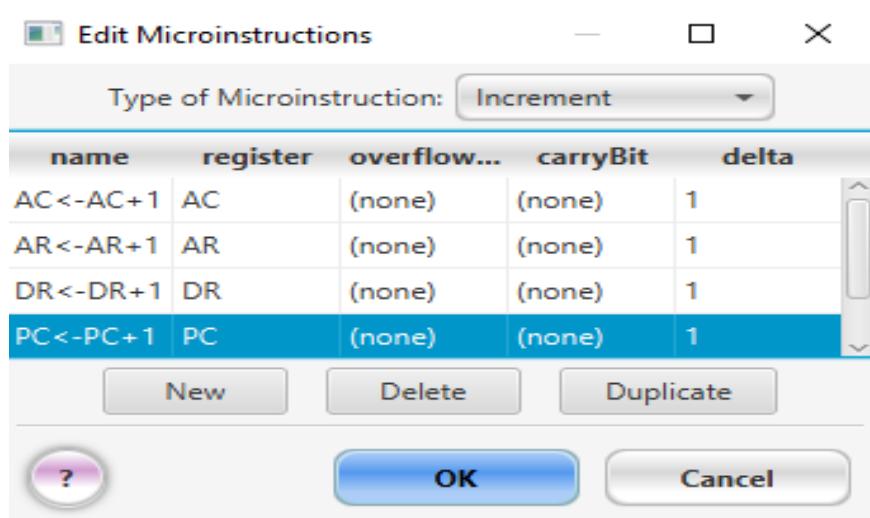
name	source	srcStartBit	dest	destStartBit	numBits
AC(0)<-E	AC	15	AC	0	1
AC(15)<-E	E	0	AC	0	1
AR<-PC	PC	0	AR	0	12
E<-AC(0)	AC	15	E	0	1
E<-AC(15)	AC	0	E	0	1

Buttons at the bottom include New, Delete, Duplicate, OK, and Cancel.

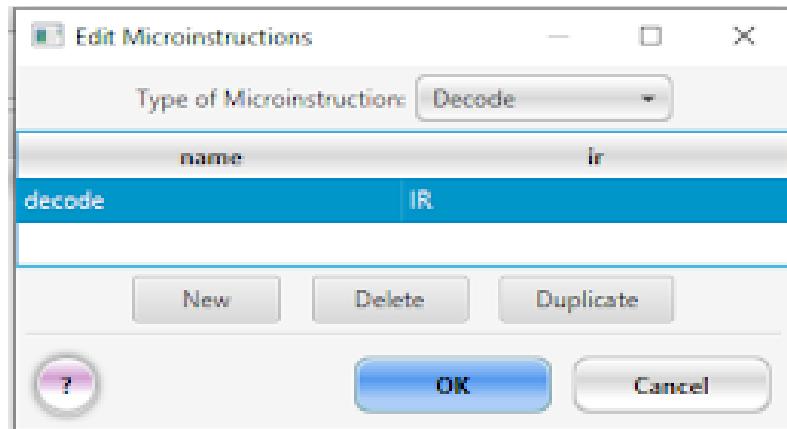
- $IR \leftarrow M[AR]$



- $PC \leftarrow PC + 1$



- Decode



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	Data
AC	16	A937
AR	12	000
DR	16	0000
E	1	1
I	1	0
INPR	8	00
IR	16	0000
OUTR	8	00
PC	12	022
S	1	0
TR	16	0000

Memory

Addr	Hex	Data	Hex
main			
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	
032		0000	



CLA

Assembly Program **CLA**

Memory After Loading

main	
Addr	Data
020	0000
021	0000
022	7800
023	0000
024	0000
025	0000
026	0000
027	0000
028	0000
029	0000
02A	0000

Registers After Execution

Name	Width	Data
AC	16	0000
AR	12	780
DR	16	0000
E	1	1
I	1	0
INPR	8	00
IR	16	7800
OUTR	8	00
PC	12	023
S	1	0
TR	16	0000

CLE

Assembly Program CLE

Memory After Loading

Addr	Hex	Data	Hex
main			
Addr		Data	
020		0000	
021		0000	
022		7400	
023		0000	
024		0000	
025		0000	
026		0000	
027		0000	
028		0000	
029		0000	
02A		0000	
02B		0000	
02C		0000	

Registers After Execution

Name	Width	Data
AC	16	A937
AR	12	022
DR	16	0000
E	1	0
I	1	0
INPR	8	00
IR	16	7400
OUTR	8	00
PC	12	023
S	1	0
TR	16	0000

CMA

Assembly Program CMA

Memory After Loading

Addr	Hex	Data
main		
020	0000	
021	0000	
022	7200	
023	0000	
024	0000	
025	0000	
026	0000	
027	0000	
028	0000	
029	0000	
02A	0000	

Registers After Execution

Name	Width	Data
AC	16	56C8
AR	12	022
DR	16	0000
E	1	1
I	1	0
INPR	8	00
IR	16	7200
OUTR	8	00
PC	12	023
S	1	0
TR	16	0000



CME

Assembly Program | [CME](#)

Memory After Loading

Addr	Hex	Data
main		
020	0000	
021	0000	
022	7100	
023	0000	
024	0000	
025	0000	
026	0000	
027	0000	
028	0000	
029	0000	
02A	0000	
02B	0000	
02C	0000	

Registers After Execution

Name	Width	Data
AC	16	A937
AR	12	022
DR	16	0000
E	1	0
I	1	0
INPR	8	00
IR	16	7100
OUTR	8	00
PC	12	023
S	1	0
TR	16	0000

CIR

Assembly Program 

Memory After Loading

Addr	Hex	Data
main		
020	0000	
021	0000	
022	7080	
023	0000	
024	0000	
025	0000	
026	0000	
027	0000	
028	0000	
029	0000	
02A	0000	
02B	0000	
02C	0000	

Registers After Execution

Name	Width	Data
AC	16	D49B
AR	12	708
DR	16	0000
E	1	1
I	1	0
INPR	8	00
IR	16	7080
OUTR	8	00
PC	12	023
S	1	0
TR	16	0000



CIL

Assembly Program [CIL](#)

Memory After Loading

Addr	Hex	Data
main		
020	0000	
021	0000	
022	7040	
023	0000	
024	0000	
025	0000	
026	0000	
027	0000	
028	0000	
029	0000	
02A	0000	
02B	0000	
02C	0000	

Registers After Execution

Registers		
Name	Width	Data
AC	16	526F
AR	12	704
DR	16	0000
E	1	1
I	1	0
INPR	8	00
IR	16	7040
OUTR	8	00
PC	12	023
S	1	0
TR	16	0000



INC

Assembly Program INC

Memory After Loading

Addr	Hex	Data
main		
020	0000	
021	0000	
022	7020	
023	0000	
024	0000	
025	0000	
026	0000	
027	0000	
028	0000	
029	0000	
02A	0000	
02B	0000	
02C	0000	

Registers After Execution

Name	Width	Data
AC	16	A938
AR	12	022
DR	16	0000
E	1	1
I	1	0
INPR	8	00
IR	16	7020
OUTR	8	00
PC	12	023
S	1	0
TR	16	0000

SPA

Assembly Program **SPA**

Memory After Loading

Addr	Hex	Data	Hex
main			
Addr		Data	
020		0000	
021		0000	
022		7010	
023		0000	
024		0000	
025		0000	
026		0000	
027		0000	
028		0000	
029		0000	
02A		0000	
02B		0000	
02C		0000	

Registers After Execution

Name	Width	Data
AC	16	A937
AR	12	022
DR	16	0000
E	1	1
I	1	0
INPR	8	00
IR	16	7010
OUTR	8	00
PC	12	023
S	1	0
TR	16	0000

SNA

Assembly Program **SNA**

Memory After Loading

Addr	Hex	Data
main		
020	0000	
021	0000	
022	7008	
023	0000	
024	0000	
025	0000	
026	0000	
027	0000	
028	0000	
029	0000	
02A	0000	
02B	0000	
02C	0000	

Registers After Execution

Name	Width	Data
AC	16	A937
AR	12	022
DR	16	0000
E	1	1
I	1	0
INFR	8	00
IR	16	7008
OUTR	8	00
PC	12	024
S	1	0
TR	16	0000

SZA

Assembly Program [SZA](#)

Memory After Loading

Addr	Hex	Data
main		
020	0000	
021	0000	
022	7004	
023	0000	
024	0000	
025	0000	
026	0000	
027	0000	
028	0000	
029	0000	
02A	0000	
02B	0000	
02C	0000	

Registers After Execution

Name	Width	Data
AC	16	A937
AR	12	022
DR	16	0000
E	1	1
I	1	0
INPR	8	00
IR	16	7004
OUTR	8	00
PC	12	023
S	1	0
TR	16	0000

SZE

Assembly Program **SZE**

Memory After Loading

Addr	Hex	Data
main		
020	0000	
021	0000	
022	7002	
023	0000	
024	0000	
025	0000	
026	0000	
027	0000	
028	0000	
029	0000	
02A	0000	
02B	0000	
02C	0000	

Registers After Execution

Name	Width	Data
AC	16	A937
AR	12	022
DR	16	0000
E	1	1
I	1	0
INPR	8	00
IR	16	7002
OUTR	8	00
PC	12	023
S	1	0
TR	16	0000

HLT

Assembly Program **HLT**

Memory After Loading

Addr	Data
main	
020	0000
021	0000
022	7001
023	0000
024	0000
025	0000
026	0000
027	0000
028	0000
029	0000
02A	0000
02B	0000
02C	0000

Registers After Execution

Name	Width	Data
AC	16	A937
AR	12	022
DR	16	0000
E	1	1
I	1	0
INPR	8	00
IR	16	7001
OUTR	8	00
PC	12	023
S	1	1
TR	16	0000

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

Memory Reference Instructions

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

For executing indirect instructions and direct instructions in one instruction we can add IF(I=0) -> Direct Exec and AR<-M[AR] implementations in direct instructions. (Also created indirect instructions separately)

- IF (I=0) -> Direct Exec

Edit Microinstructions

Type of Microinstruction: Test

name	register	start	numBits	comparison	value	omission
if(AC!=0)	AC	0	16	NE	0	1
if(DR !=0)	DR	0	16	NE	0	1
if(E!=0)	E	0	1	NE	0	1
if(I=0)->DIRECT EXEC	I	0	1	EQ	0	0
if[AC(15)!=0]	AC	15	1	NE	0	1
if[AC(15)!=1]	AC	15	1	NE	0	1

New Delete Duplicate

? OK Cancel

- 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

? OK Cancel

AND to AC

AND – Direct Addressing Mode

Implementation:

Edit Machine Instructions

Instructions: ISZ_I, ISZ, BSA_I, BSA, BUN_I, BUN, STA_I, STA, LDA_I, LDA, ADD_I, ADD, AND_I, AND

Format Implementation

Instruction Length: 16 Opcode 0x0

4 12

Opcodes 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.

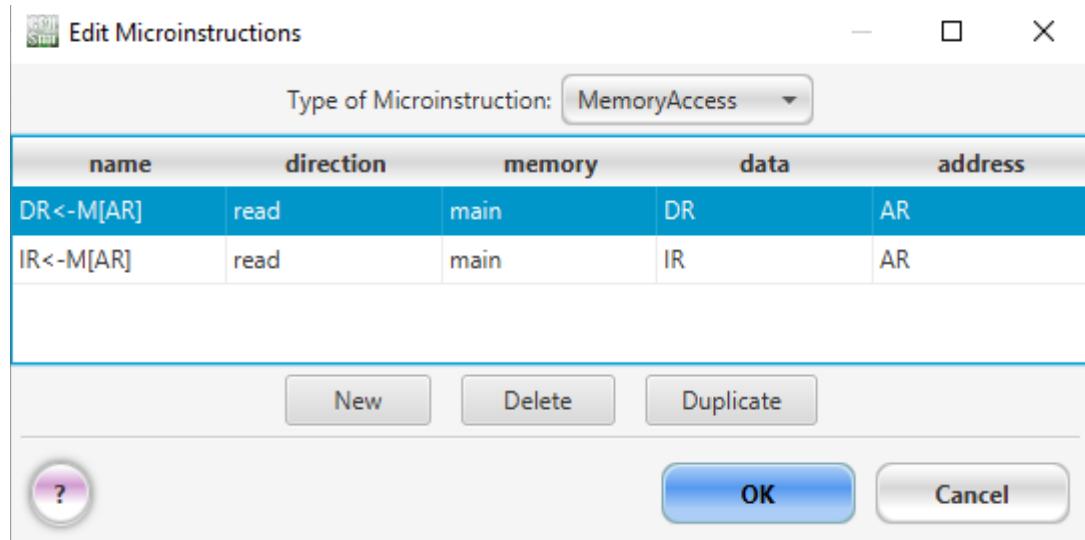
All Fields: Opcode, Address, Operation

Assembly: Opcode, Address

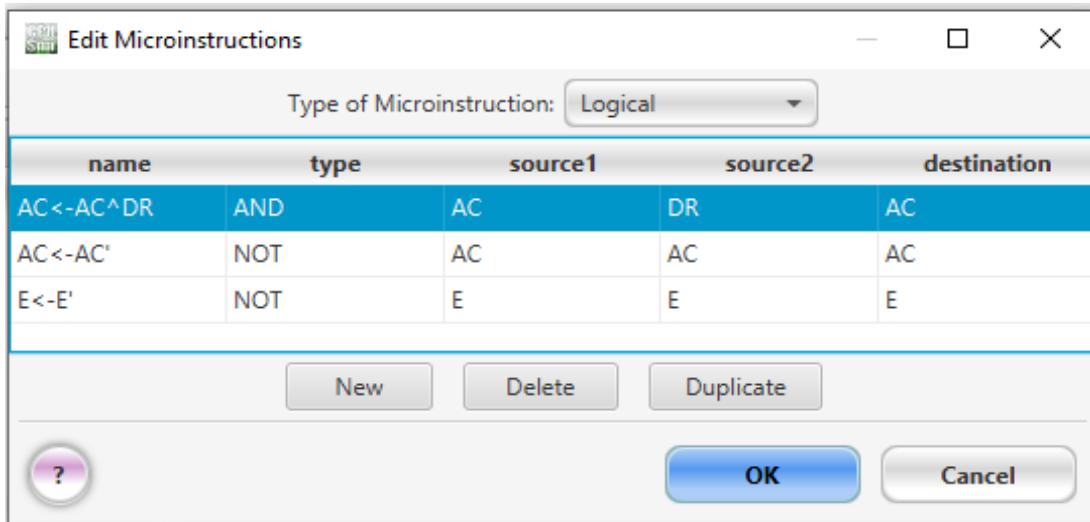
Edit Fields...

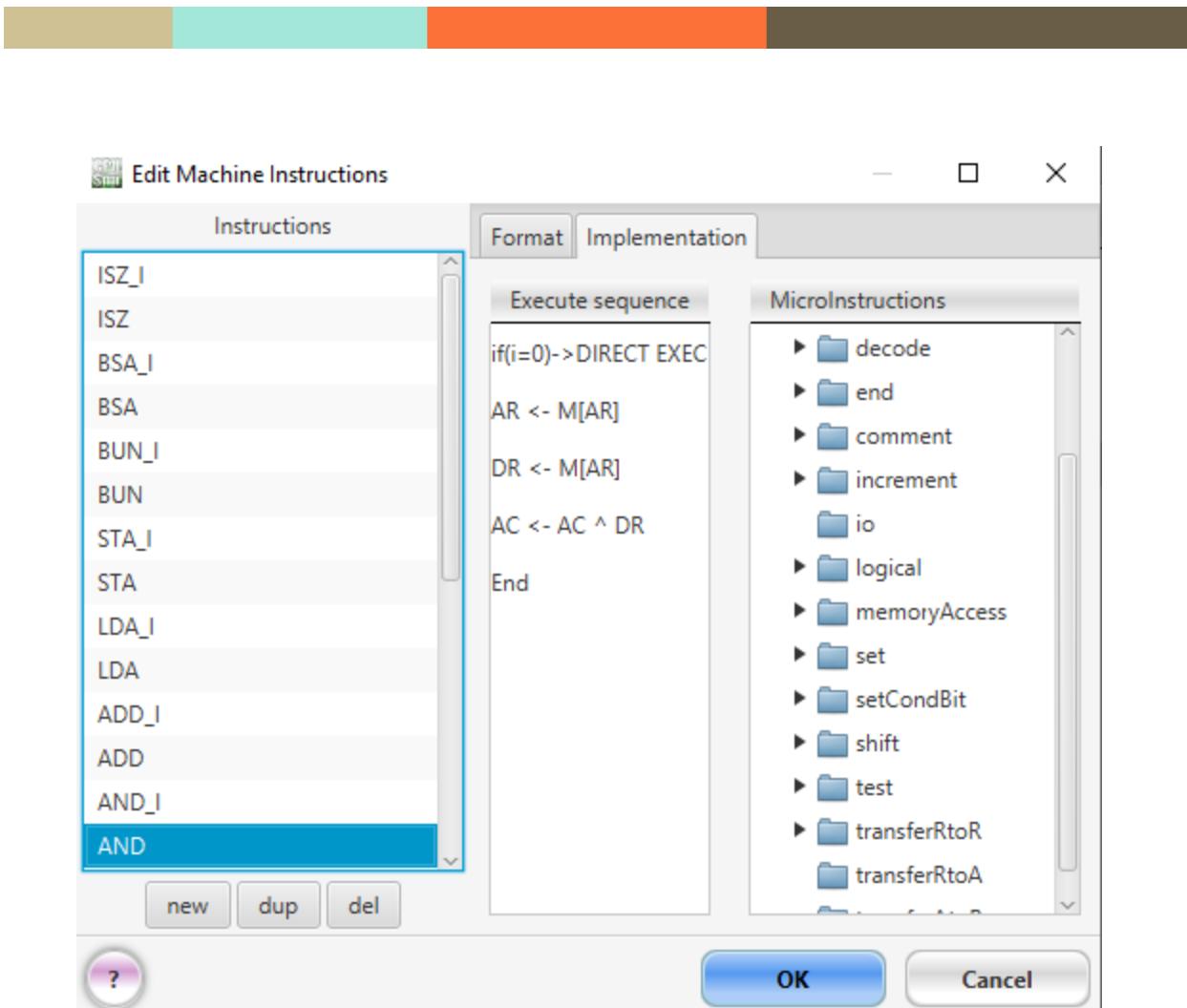
? OK Cancel

- DR<-M [AR]



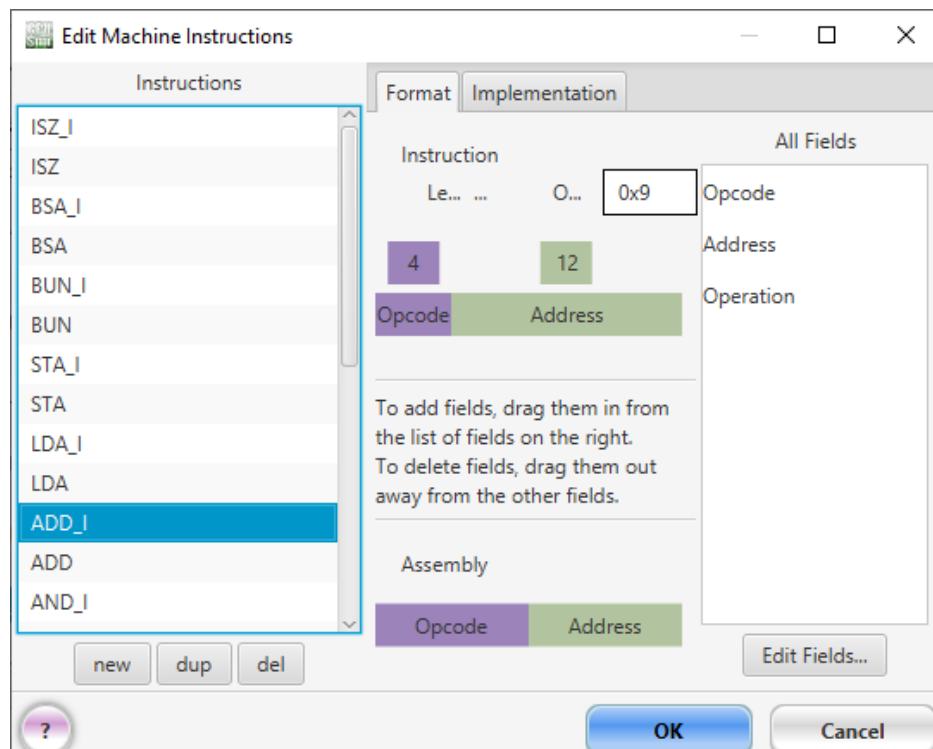
- AC <- AC^DR





AND_I - Indirect Addressing Mode

Implementation:



- AR $\leftarrow M[AR]$

Screenshot of the 'Edit Microinstructions' dialog box. The 'Type of Microinstruction' dropdown is set to 'MemoryAccess'. The table lists three microinstructions:

name	direction	memory	data	address
AR $\leftarrow M[AR]$	read	main	AR	AR
DR $\leftarrow M[AR]$	read	main	DR	AR
IR $\leftarrow M[AR]$	read	main	IR	AR

Buttons at the bottom include 'New', 'Delete', 'Duplicate', '?', 'OK', and 'Cancel'.

- DR \leftarrow M[AR]

Edit Microinstructions

Type of Microinstruction: MemoryAccess

name	direction	memory	data	address
DR <- M[AR]	read	main	DR	AR
AR <- M[AR]	read	main	AR	AR
DR<-M[AR]	read	main	DR	AR
IR<-M[AR]	read	main	IR	AR

New Delete Duplicate

?

OK Cancel

- AC \leftarrow AC \wedge DR

Edit Microinstructions

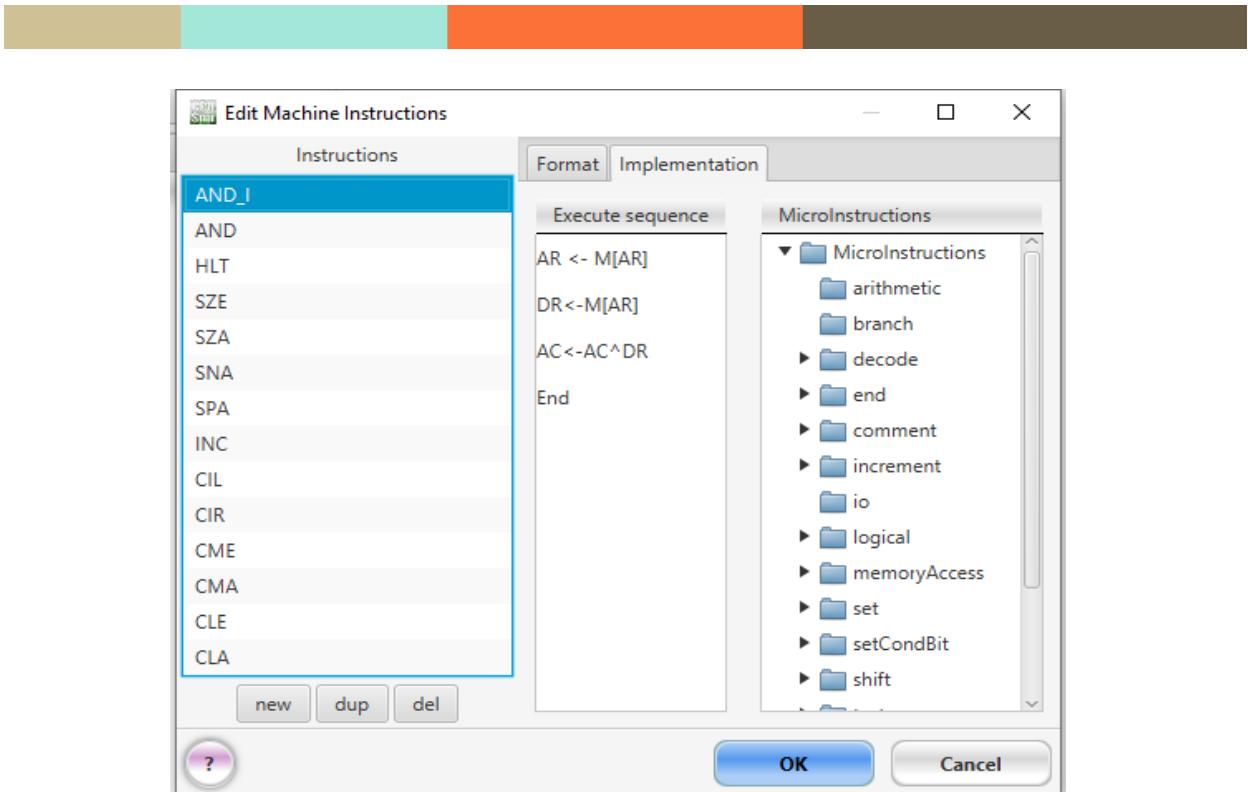
Type of Microinstruction: Logical

name	type	source1	source2	destination
AC<-AC \wedge DR	AND	AC	DR	AC
AC<-AC'	NOT	AC	AC	AC
E<-E'	NOT	E	E	E

New Delete Duplicate

?

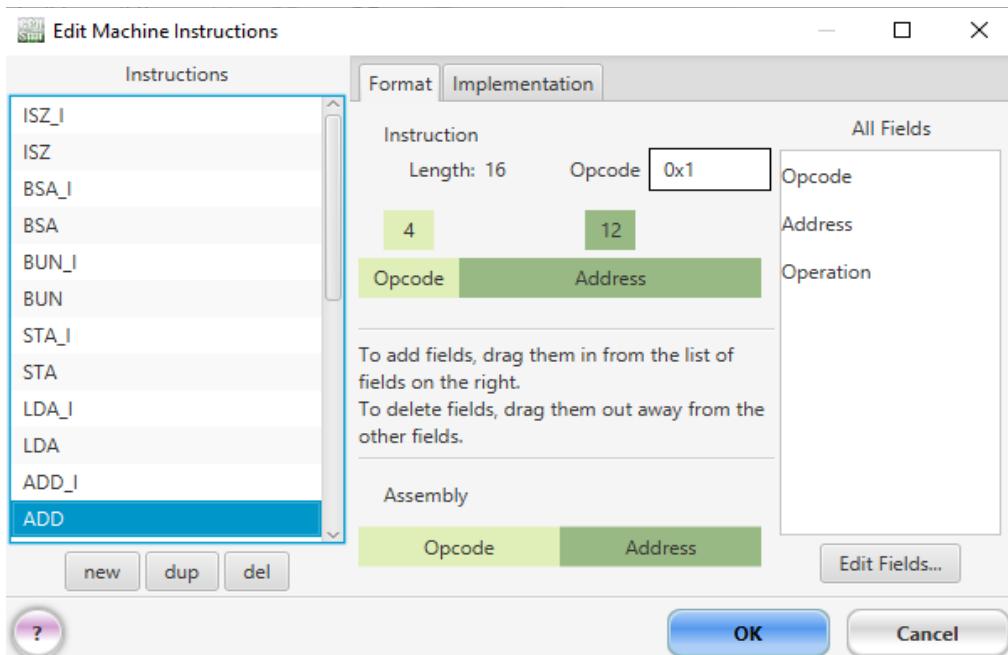
OK Cancel



ADD to AC

ADD – Direct Addressing Mode

Implementation:



- DR \leftarrow M[AR]

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

New Delete Duplicate

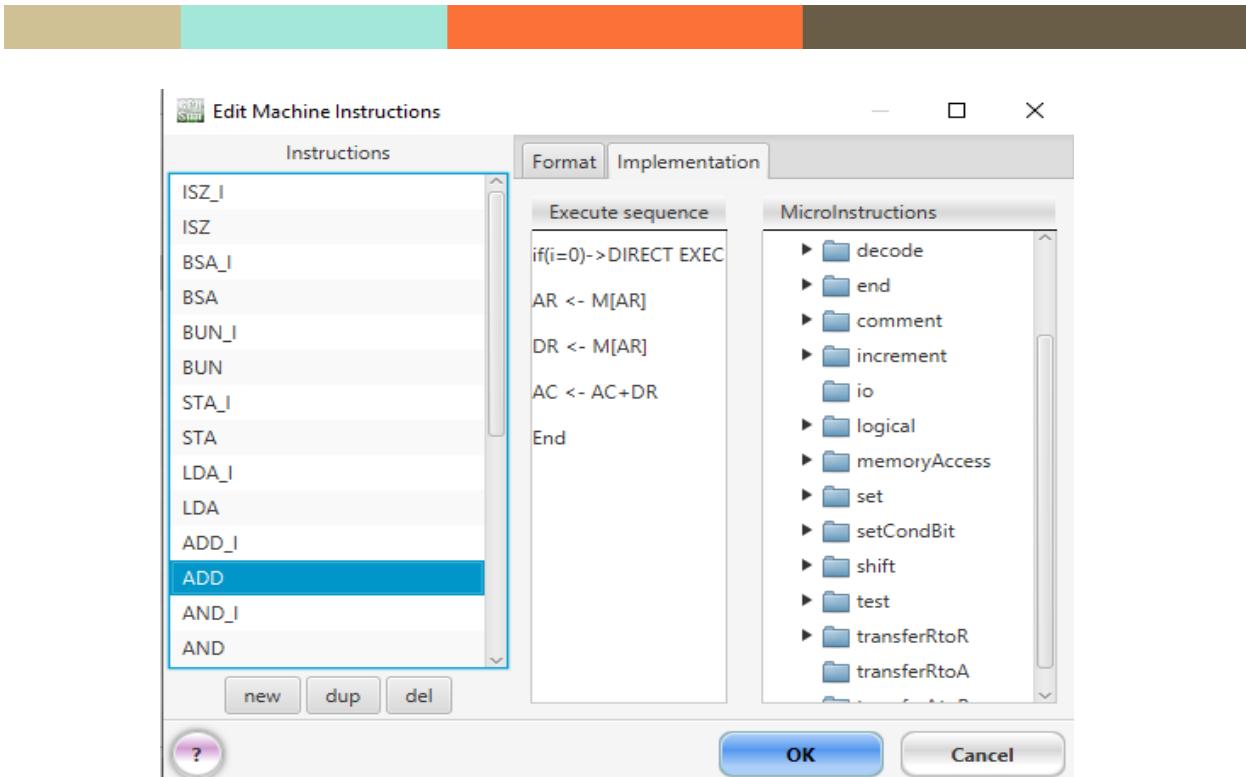
OK Cancel

- AC \leftarrow AC + DR

Edit Microinstructions						
Type of Microinstruction: Arithmetic						
name	type	source1	source2	destination	overflowBit	carryBit
AC <- AC + DR	ADD	AC	DR	AC	HALT	HALT

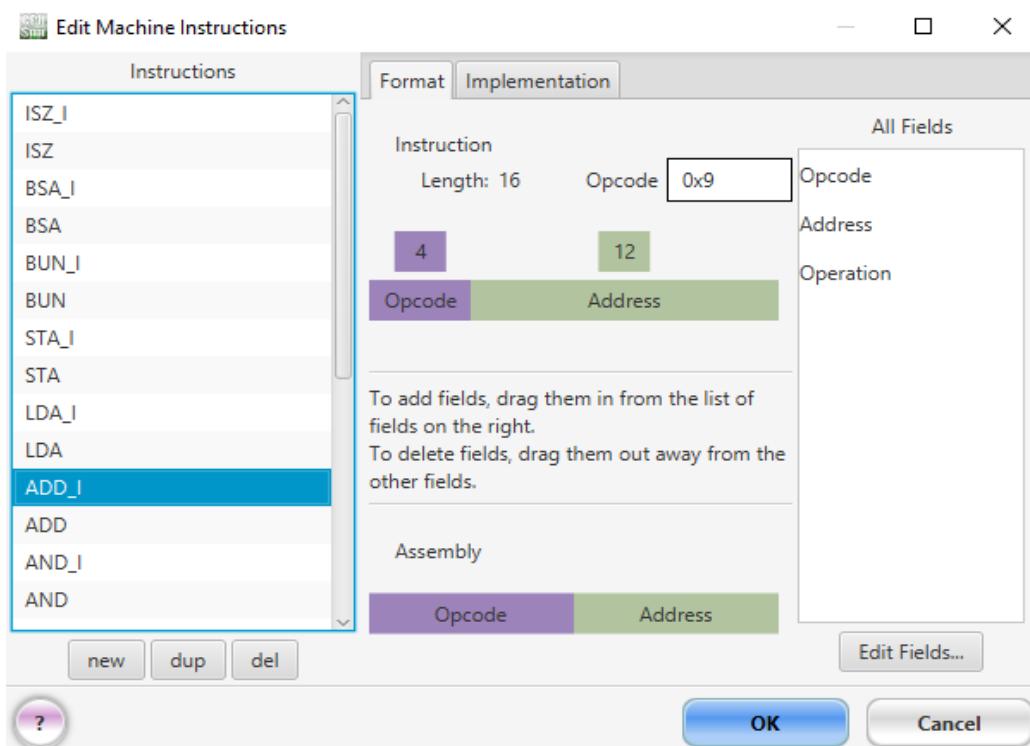
New Delete Duplicate

OK Cancel

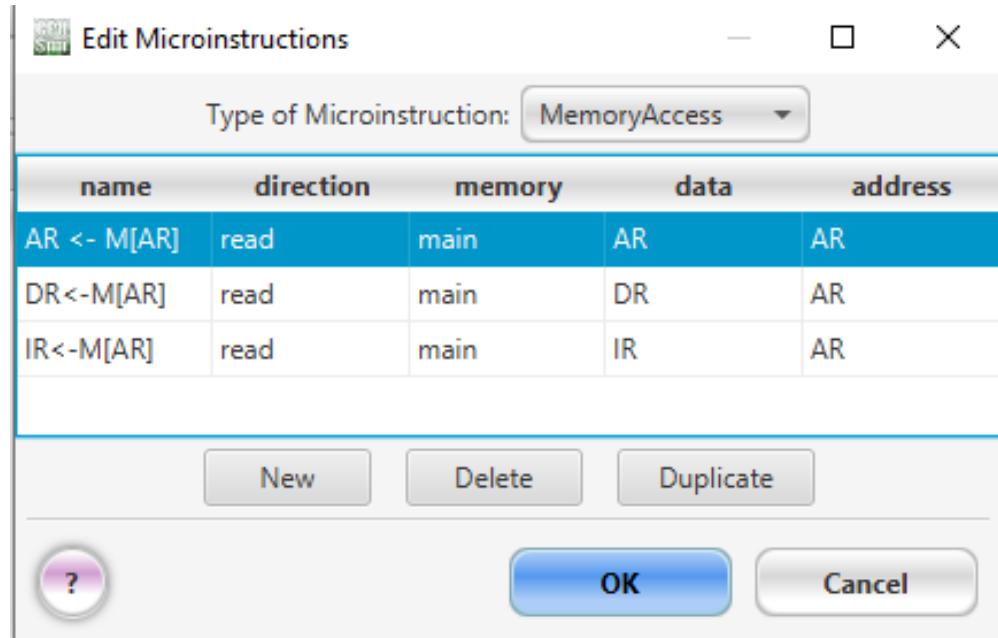


ADD_I - Indirect Addressing Mode

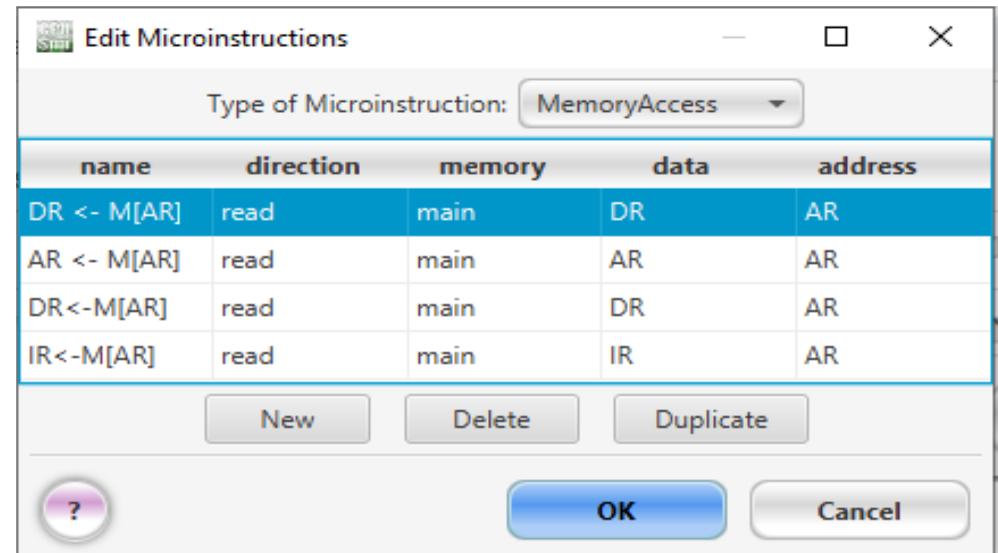
Implementation:



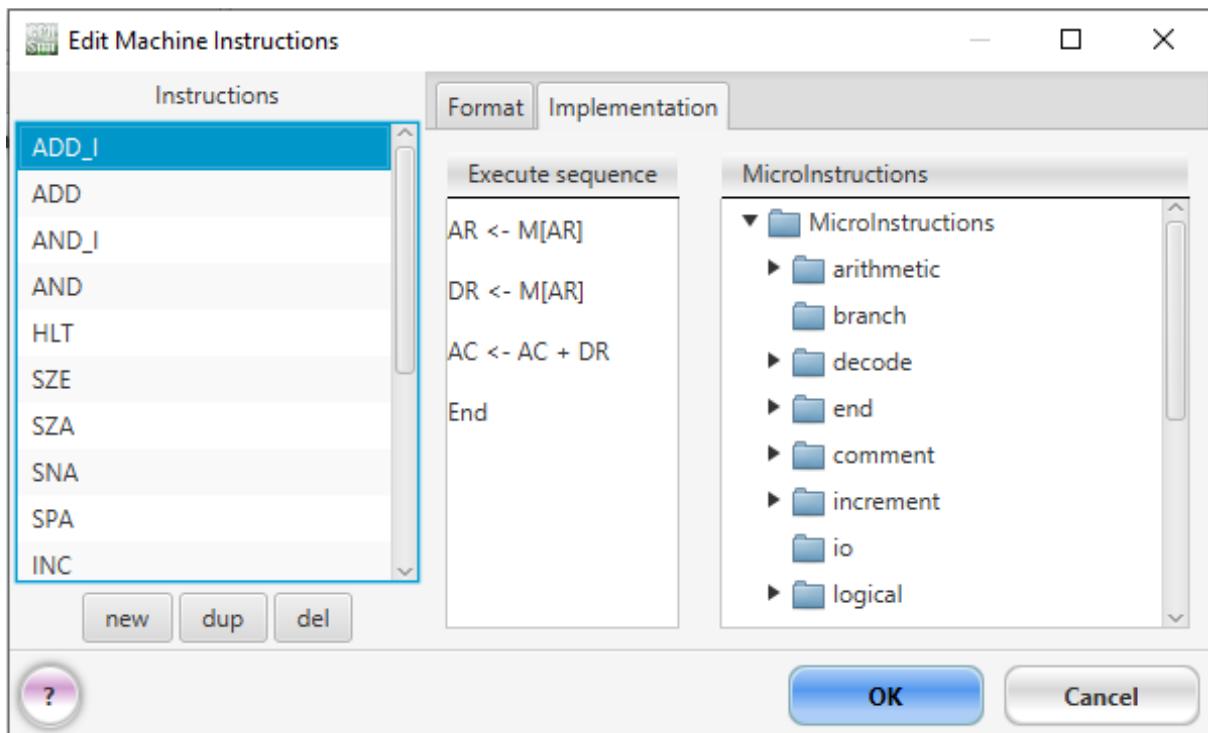
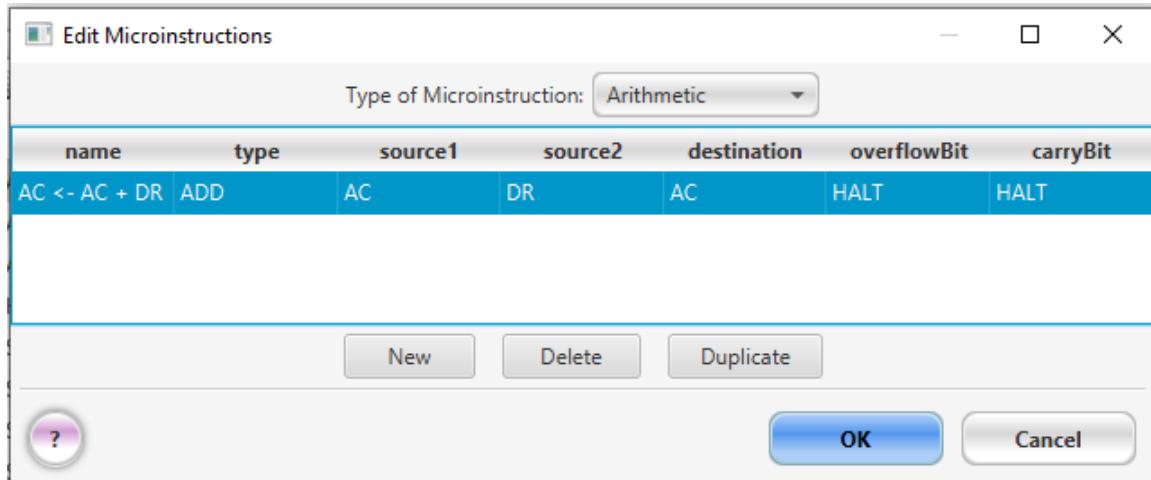
- $AR \leftarrow M[AR]$



- $DR \leftarrow M[AR]$



- $AC \leftarrow AC + DR$

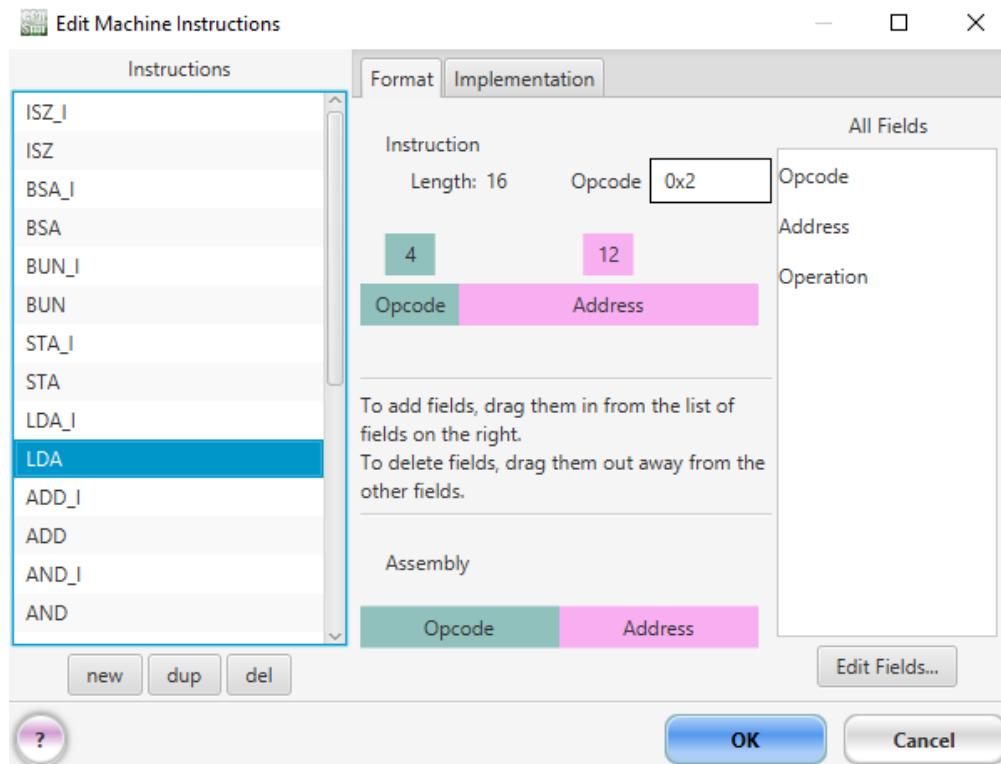




LOAD to AC

LDA – Direct Addressing Mode

Implementation:



- DR $\leftarrow M[AR]$

Edit Microinstructions				
Type of Microinstruction: MemoryAccess				
name	direction	memory	data	address
DR $\leftarrow M[AR]$	read	main	DR	AR
AR $\leftarrow M[AR]$	read	main	AR	AR
DR $\leftarrow M[AR]$	read	main	DR	AR
IR $\leftarrow M[AR]$	read	main	IR	AR

Buttons at the bottom: New, Delete, Duplicate, OK, Cancel.

- AC <- DR

Edit Microinstructions

Type of Microinstruction: TransferRtoR

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

New Delete Duplicate

?

OK Cancel

Edit Machine Instructions

Instructions: ISZ_I, ISZ, BSA_I, BSA, BUN_I, BUN, STA_I, STA, LDA_I, LDA, ADD_I, ADD, AND_I, AND

Format Implementation

Execute sequence: if(i=0)->DIRECT EXEC, AR <- M[AR], DR <- M[AR], AC<-DR, End

MicroInstructions: decode, end, comment, increment, io, logical, memoryAccess, set, setCondBit, shift, test, transferRtoR, transferRtoA

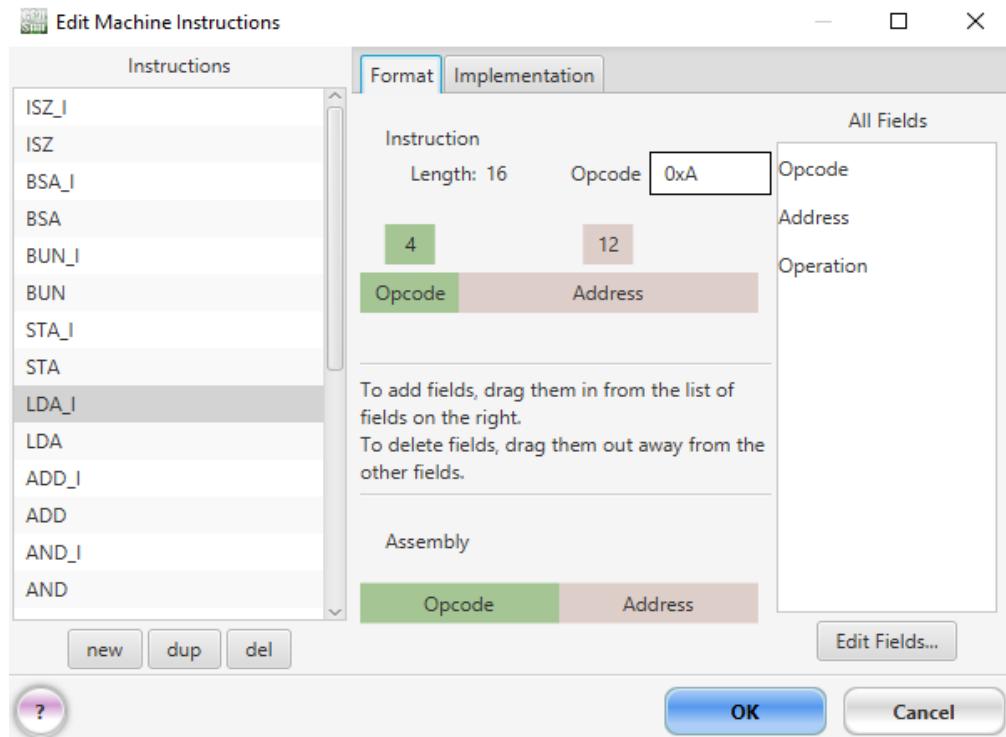
new dup del

?

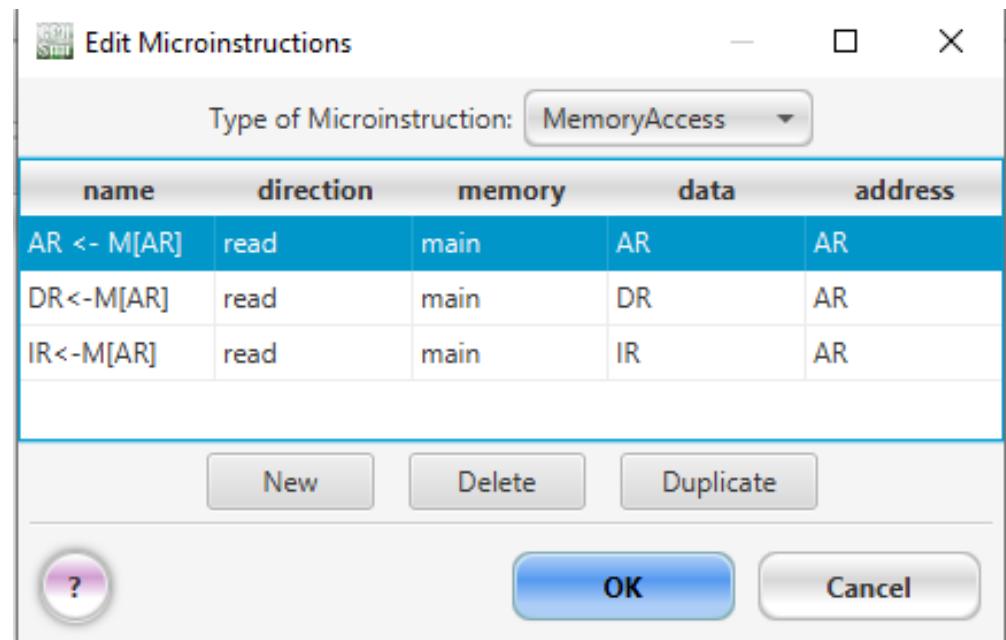
OK Cancel

LDA_I - Indirect Addressing Mode

Implementation:



- AR $\leftarrow M[AR]$



- DR \leftarrow M[AR]

Edit Microinstructions

Type of Microinstruction: MemoryAccess

name	direction	memory	data	address
DR <- M[AR]	read	main	DR	AR
AR <- M[AR]	read	main	AR	AR
DR<-M[AR]	read	main	DR	AR
IR<-M[AR]	read	main	IR	AR

New Delete Duplicate

?

OK Cancel

- AC \leftarrow DR

Edit Microinstructions

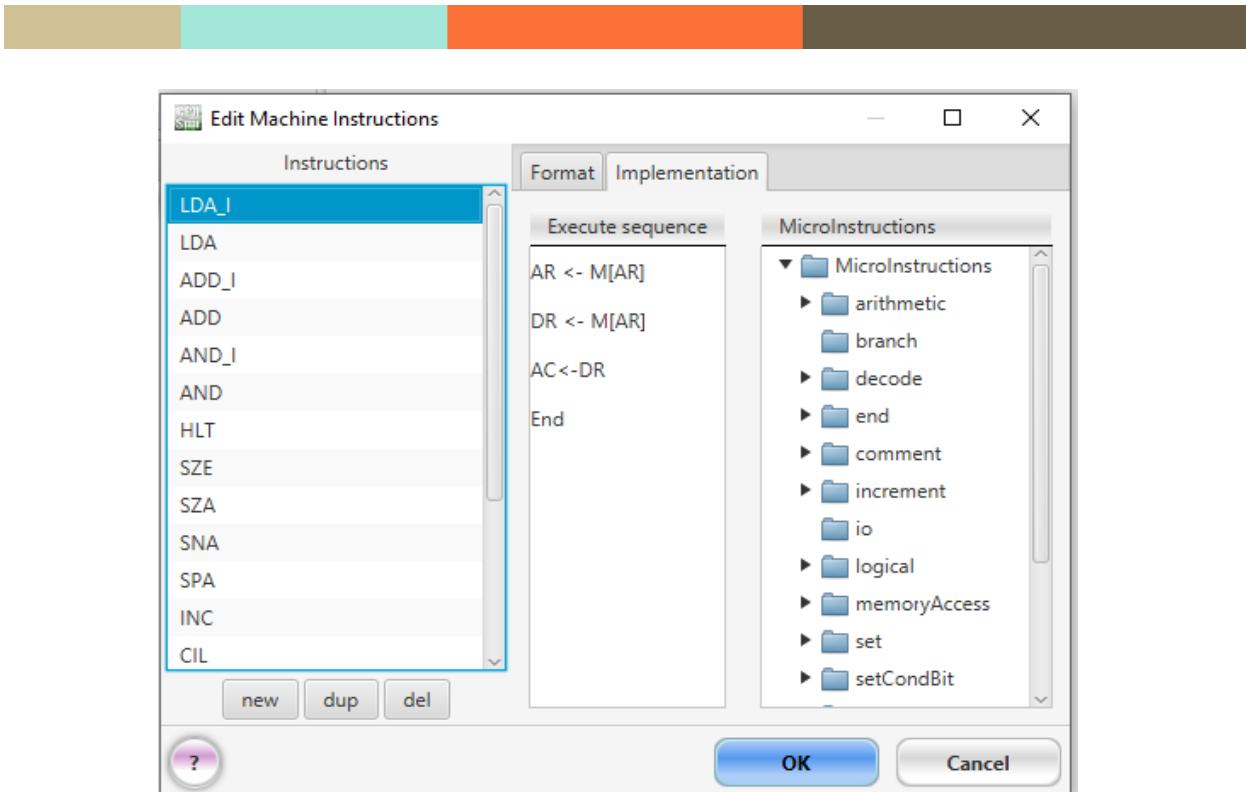
Type of Microinstruction: TransferRtoR

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

New Delete Duplicate

?

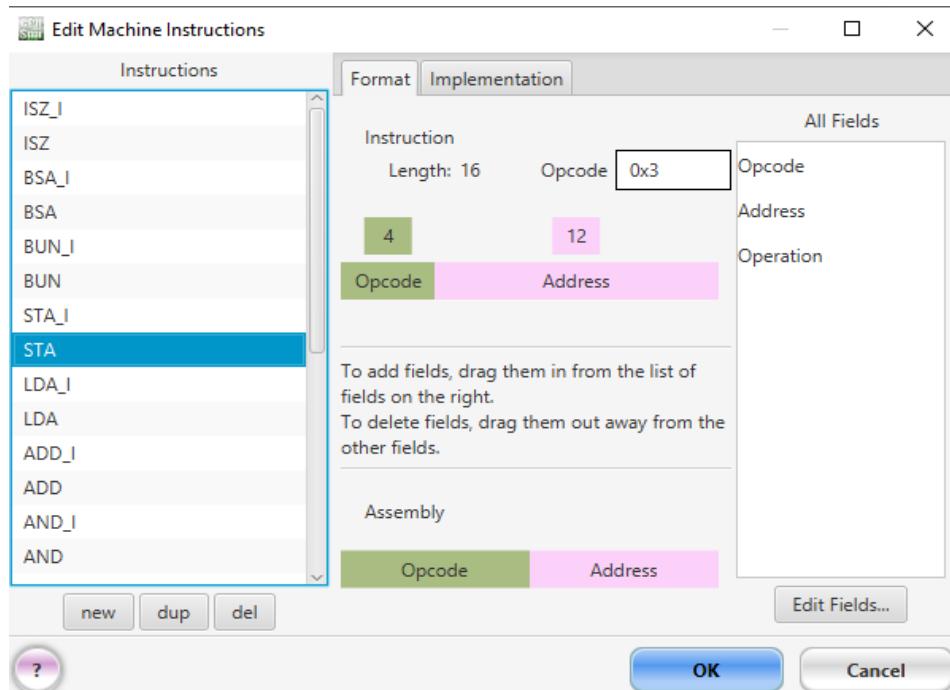
OK Cancel



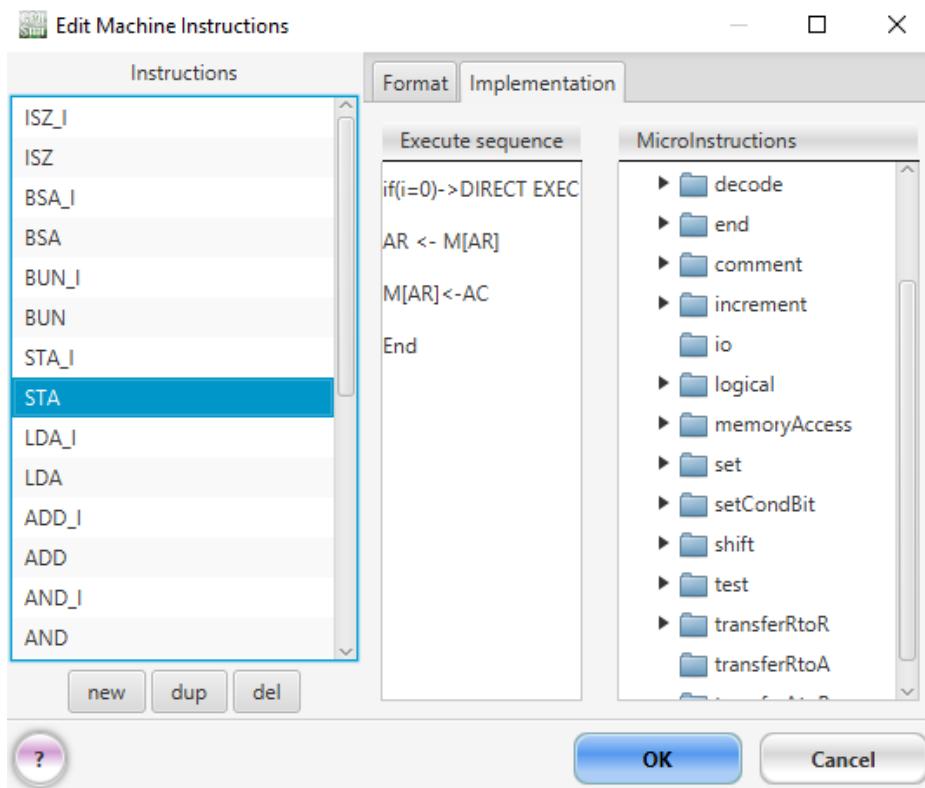
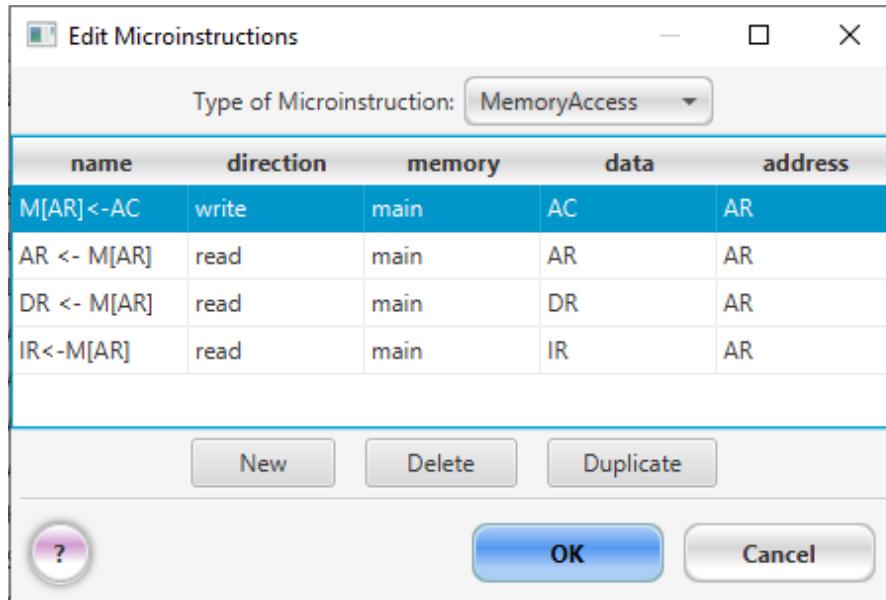
STORE AC

STA - Direct Addressing Mode

Implementation:



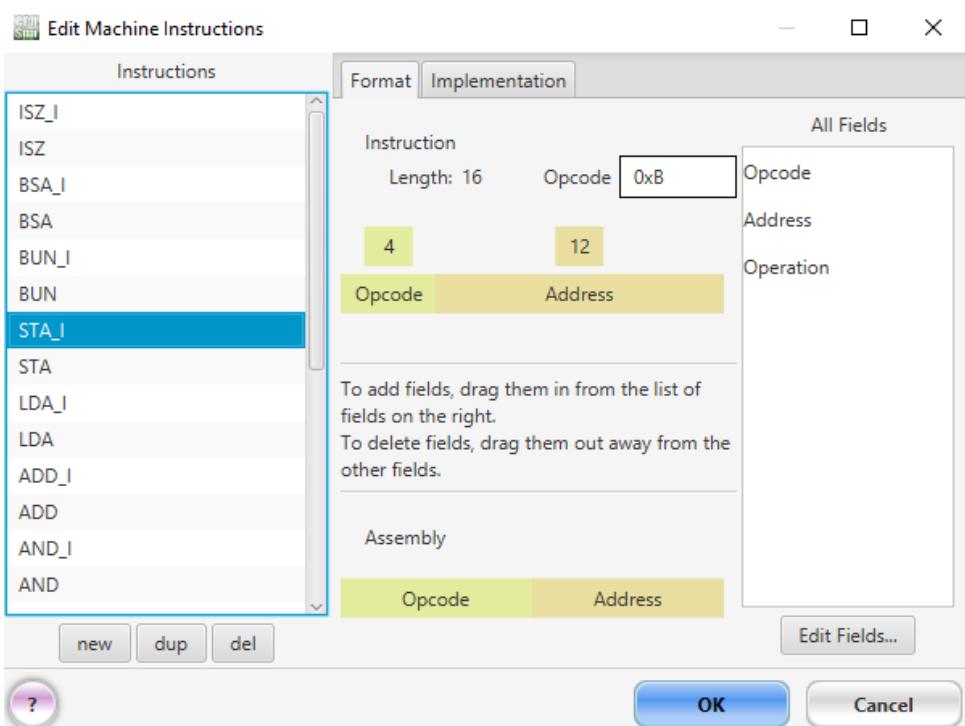
- $M[AR] \leftarrow AC$





STA_I - Indirect Addressing Mode

Implementation:



- $AR \leftarrow M[AR]$

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

New Delete Duplicate

? OK Cancel

- $M[AR] \leftarrow AC$

Edit Microinstructions

Type of Microinstruction: MemoryAccess

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

New Delete Duplicate

?

OK Cancel

Edit Machine Instructions

Instructions: STA_I, LDA_I, LDA, ADD_I, ADD, AND_I, AND, HLT, SZE

Format Implementation

Execute sequence:

```

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

```

MicroInstructions:

- MicroInstructions
 - arithmetic
 - branch
 - decode
 - end
 - comment
 - increment
 - io
 - logical
 - memoryAccess

new dup del

?

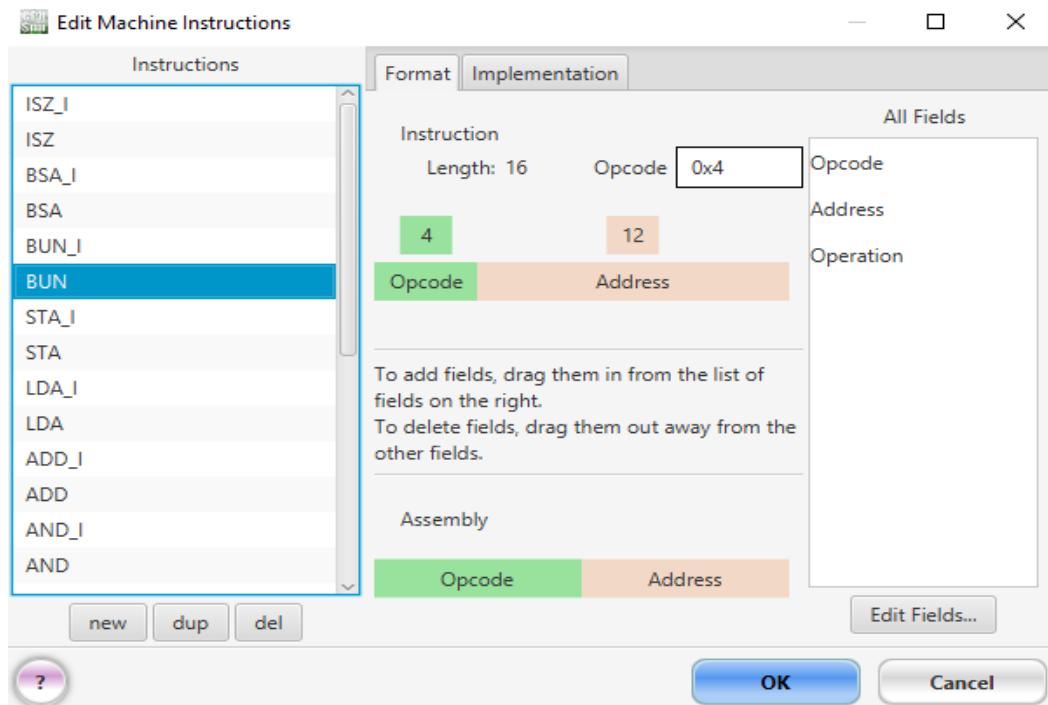
OK Cancel



Branch UNconditionality

BUN – Direct Addressing Mode

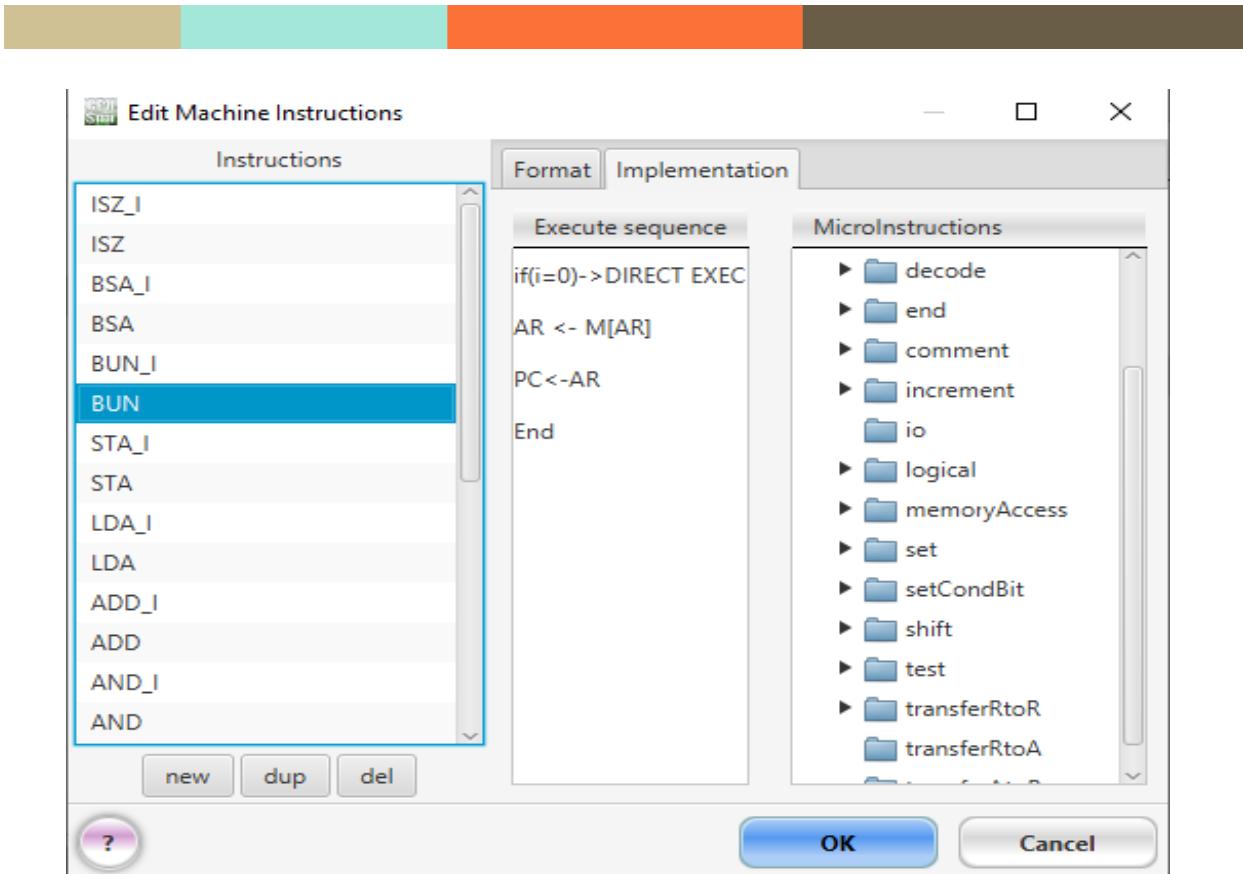
Implementation:



- PC ← AR

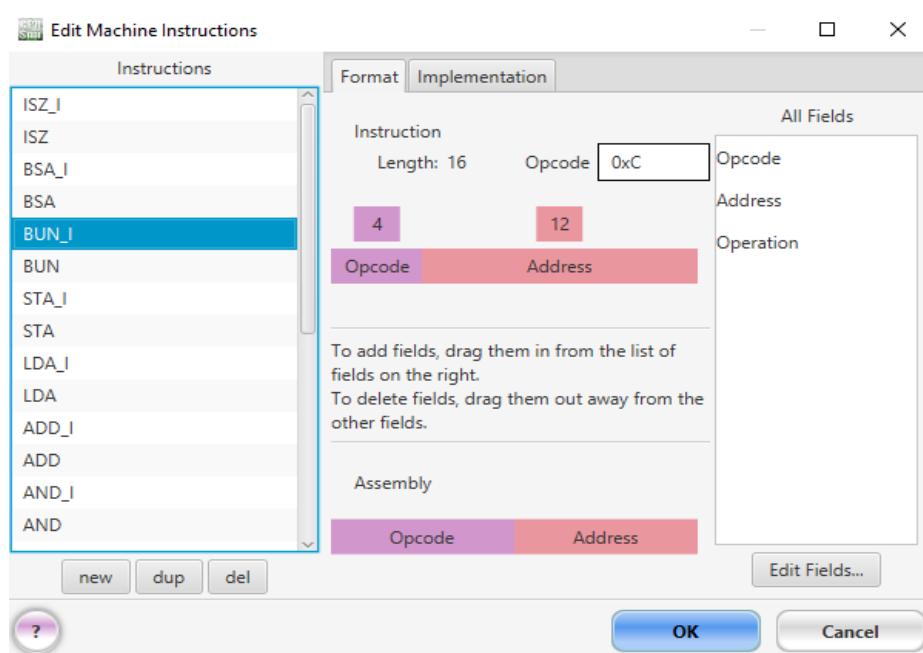
Edit Microinstructions					
Type of Microinstruction: TransferRtoR					
name	source	srcStartBit	dest	destStartBit	numBits
PC-<-AR	AR	0	PC	0	12
AC(0)-<E	AC	15	AC	0	1
AC(15)-<E	E	0	AC	0	1
AC-<-DR	DR	0	AC	0	16
AR-<-PC	PC	0	AR	0	12
E-<-AC(0)	AC	15	E	0	1
E-<-AC(15)	AC	0	E	0	1

Buttons at the bottom: New, Delete, Duplicate, ?, OK, Cancel.

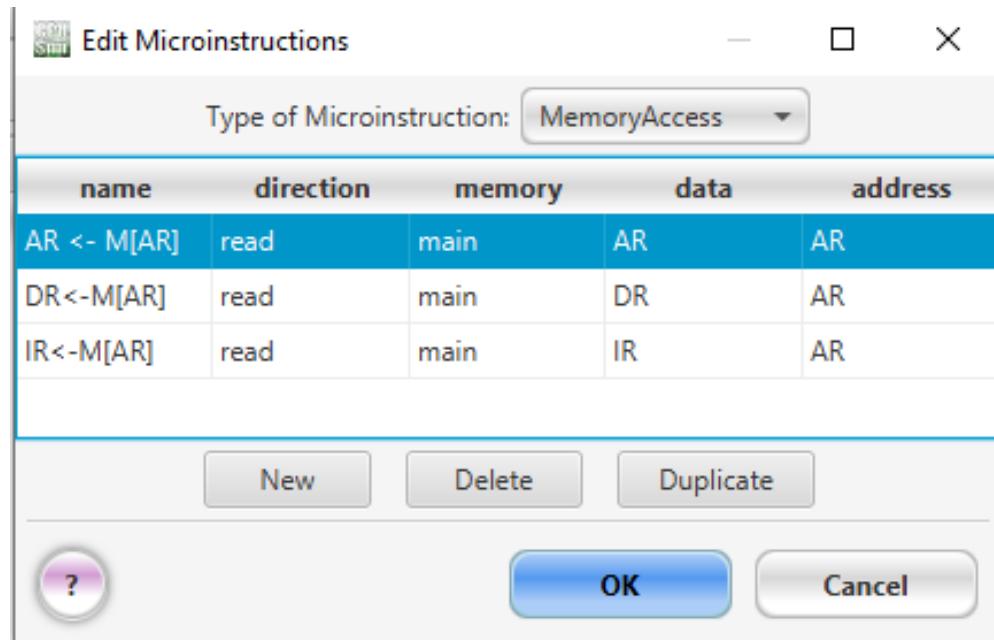


BUN_I - Indirect Addressing Mode

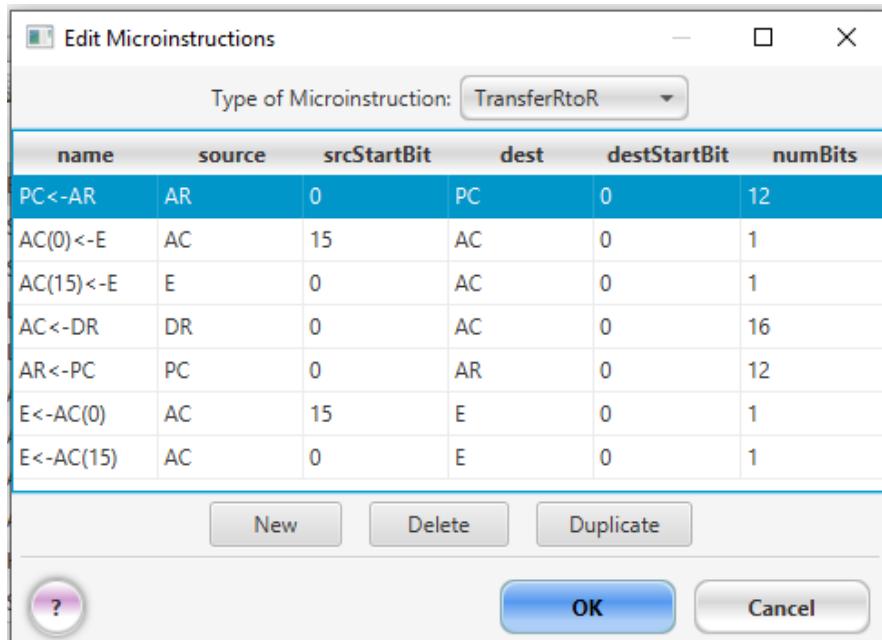
Implementation:

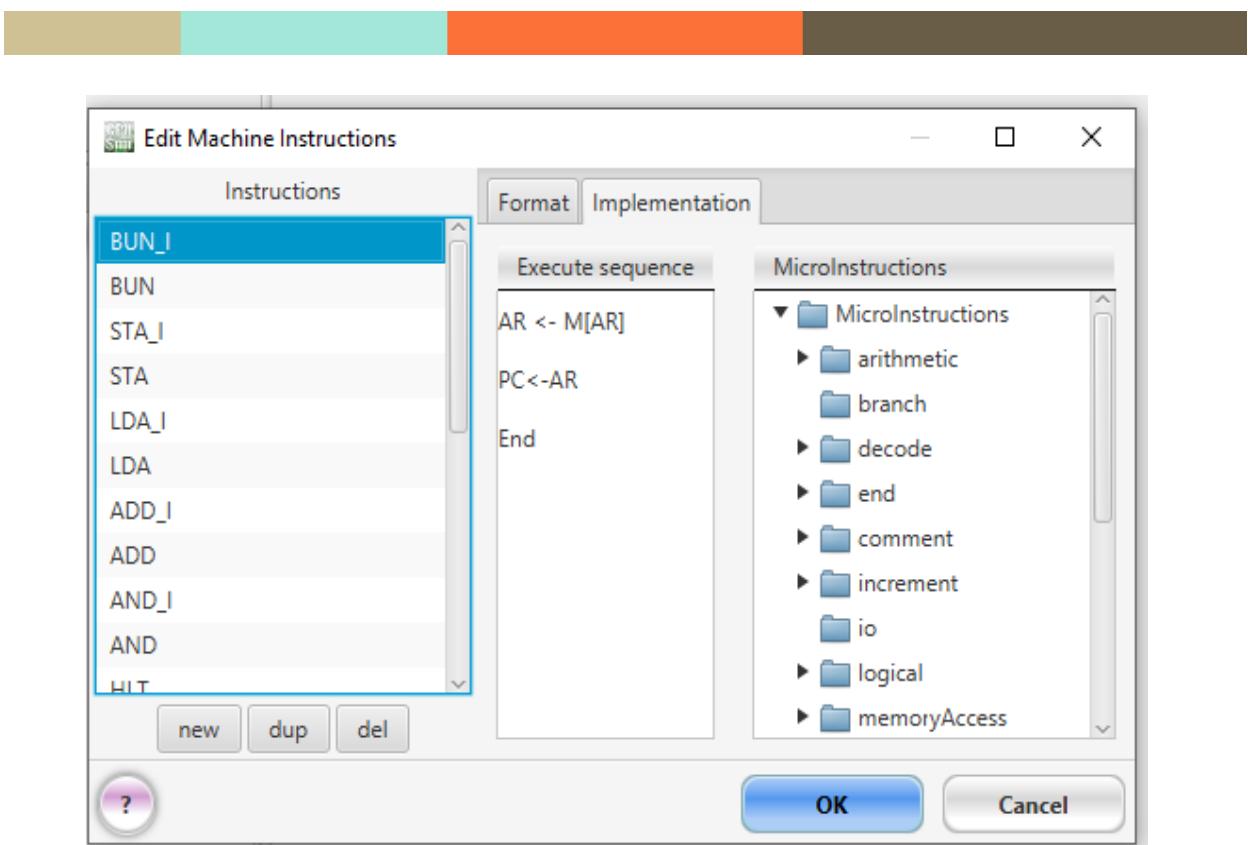


- $AR \leftarrow M[AR]$



- $PC \leftarrow AR$



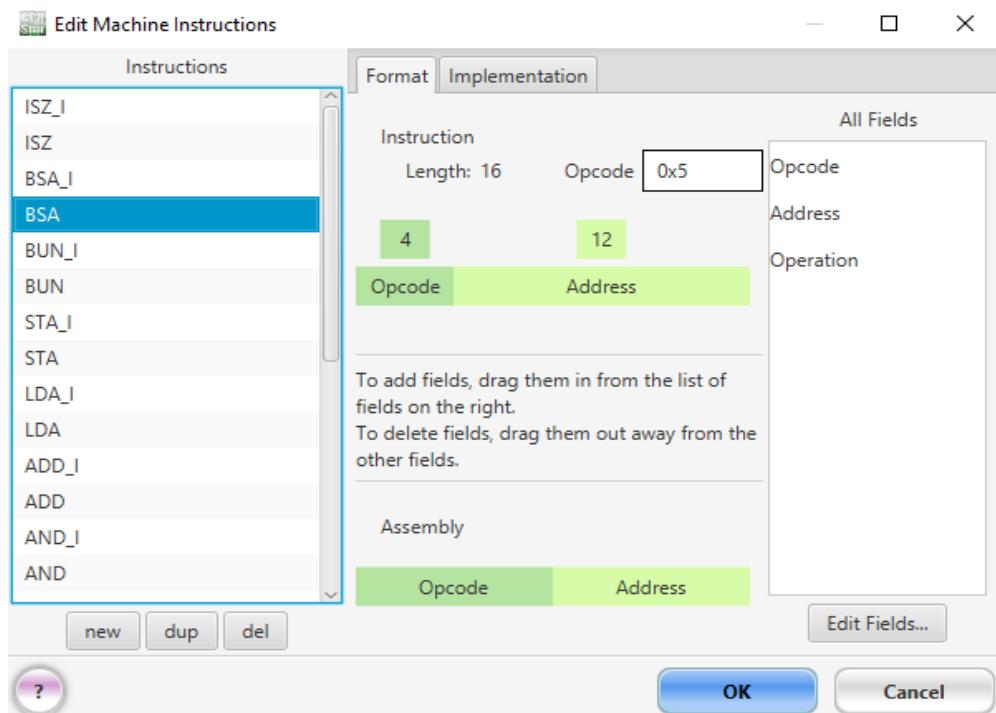




Branch and Save return Address

BSA – Direct Addressing Mode

Implementation:



- $M[AR] \leftarrow PC$

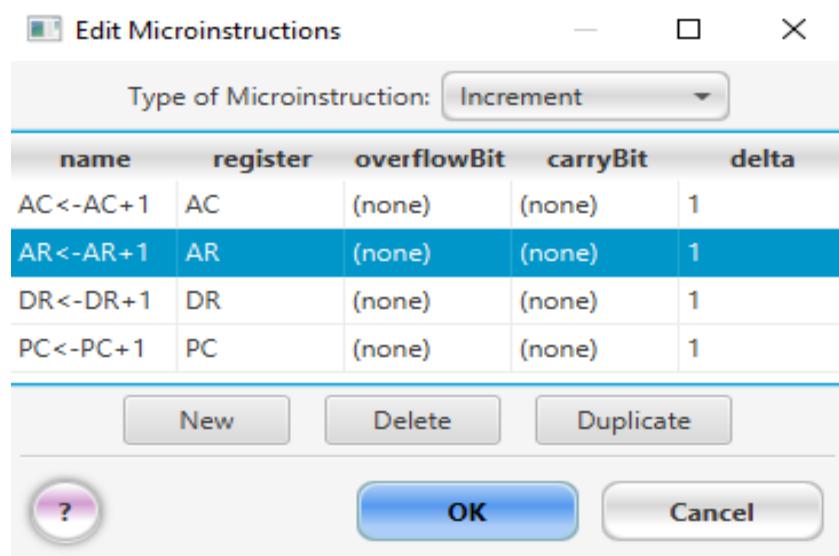
Edit Microinstructions

Type of Microinstruction: MemoryAccess

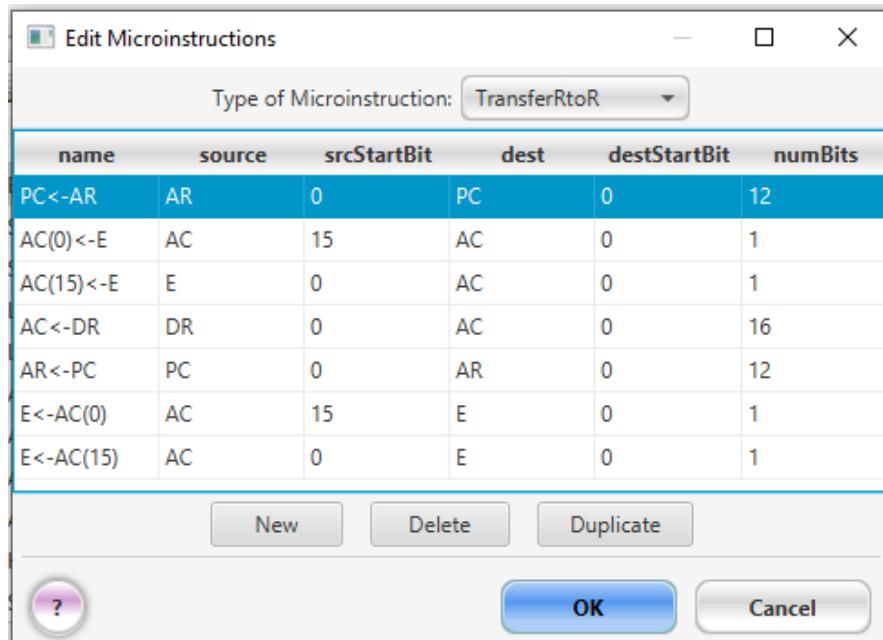
name	direction	memory	data	address
$M[AR] \leftarrow PC$	write	main	PC	AR
$AR \leftarrow M[AR]$	read	main	AR	AR
$DR \leftarrow M[AR]$	read	main	DR	AR
$IR \leftarrow M[AR]$	read	main	IR	AR
$M[AR] \leftarrow AC$	write	main	AC	AR

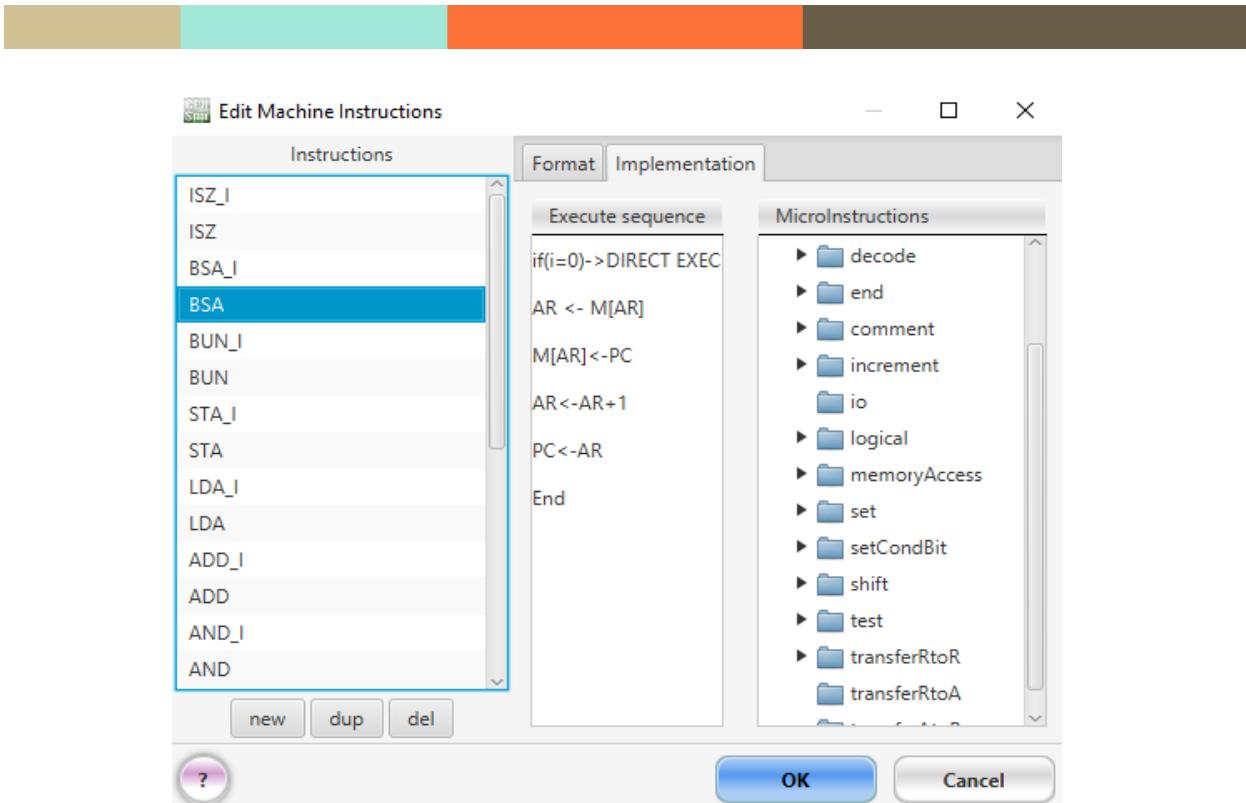
New Delete Duplicate ? OK Cancel

- $AR \leftarrow AR + 1$



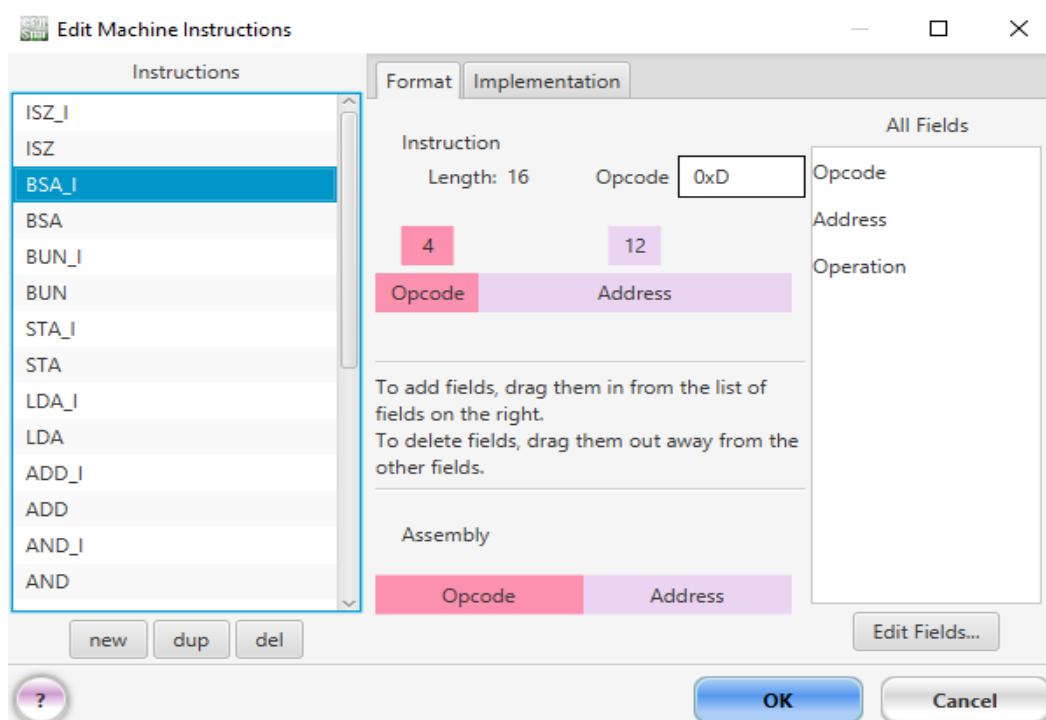
- $PC \leftarrow AR$





BSA_I - Indirect Addressing Mode

Implementation:



- $AR \leftarrow M[AR]$

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

New Delete Duplicate

?

OK Cancel

- $M[AR] \leftarrow PC$

Edit Microinstructions

Type of Microinstruction: MemoryAccess

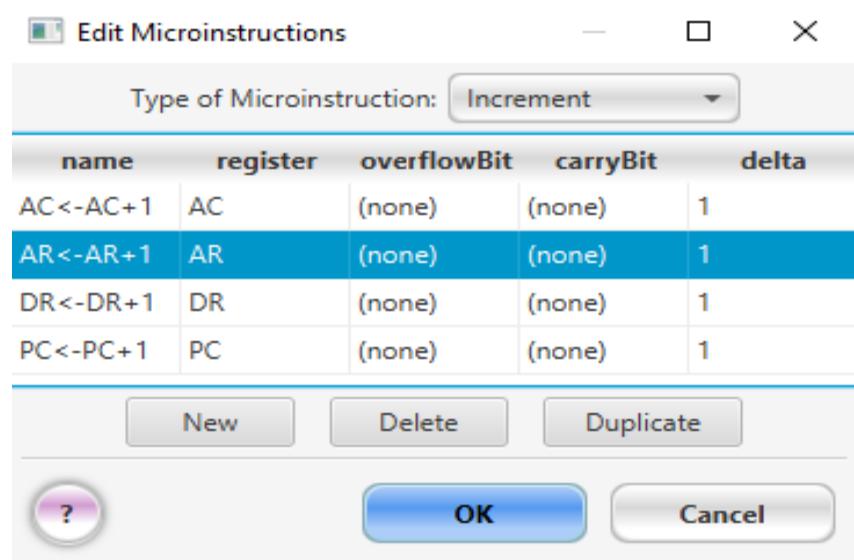
name	direction	memory	data	address
M[AR]<-PC	write	main	PC	AR
AR <- M[AR]	read	main	AR	AR
DR <- M[AR]	read	main	DR	AR
IR <- M[AR]	read	main	IR	AR
M[AR]<-AC	write	main	AC	AR

New Delete Duplicate

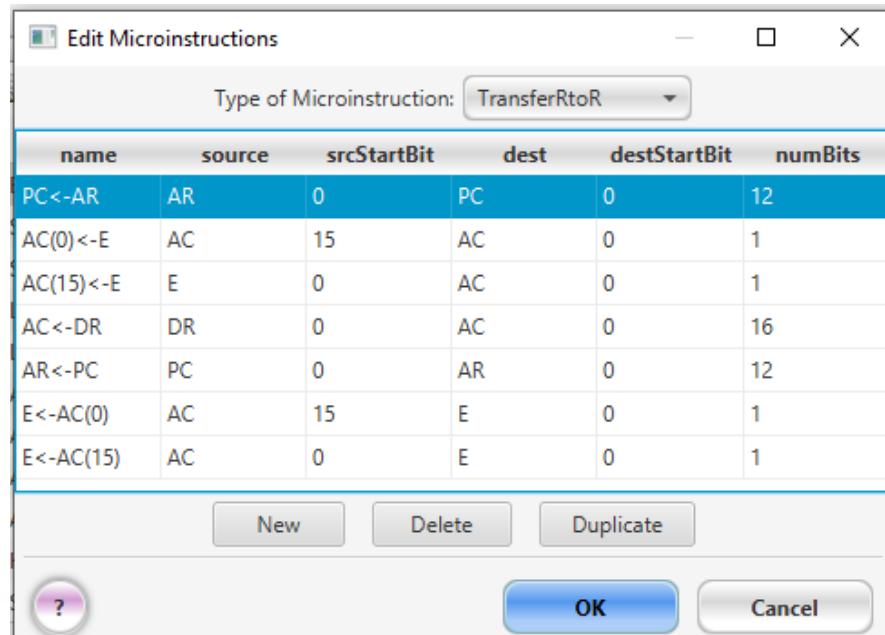
?

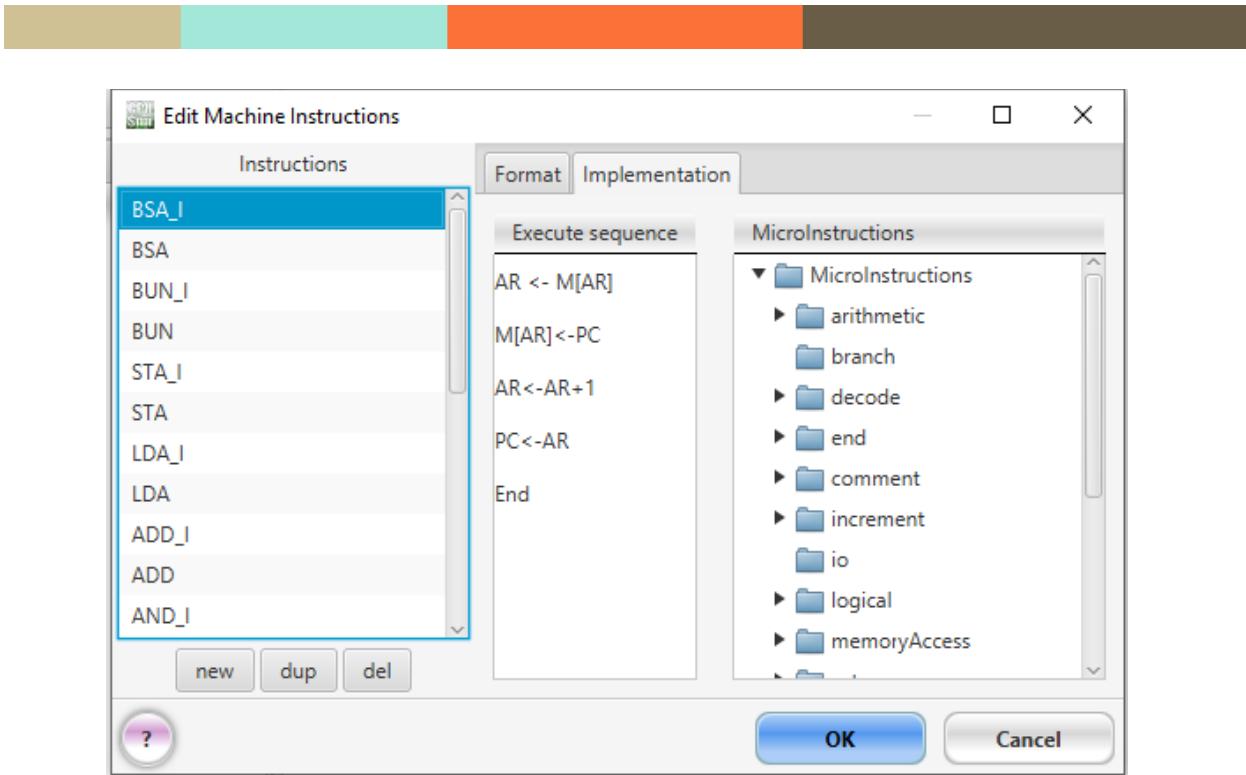
OK Cancel

- $AR \leftarrow AR + 1$



- $PC \leftarrow AR$

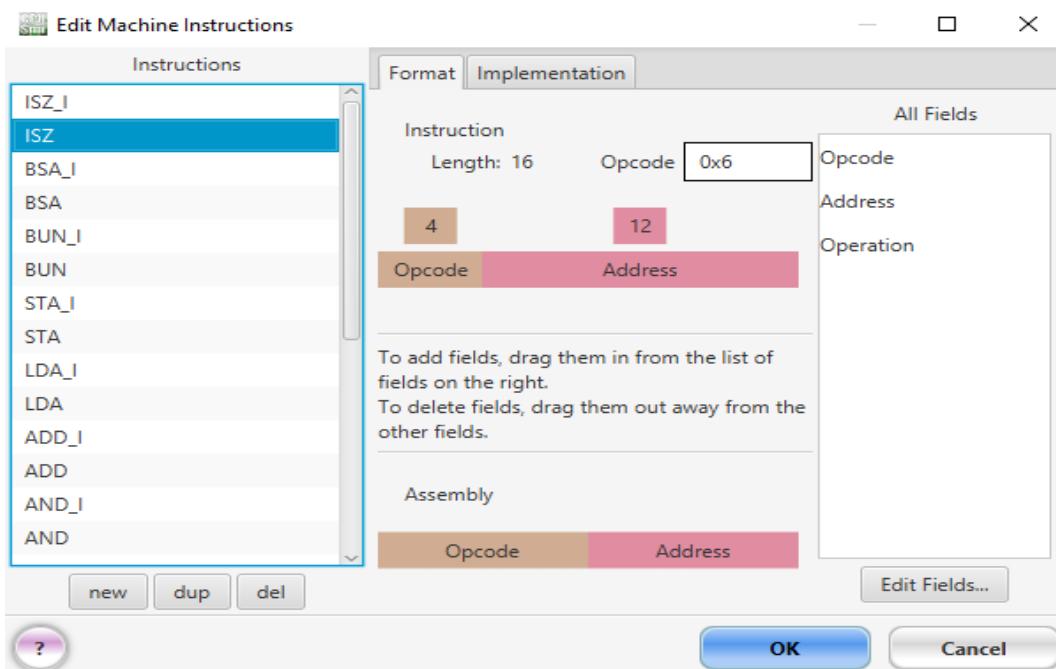




Increment and Skip If Zero

ISZ - Direct Addressing Mode

Implementation:



- DR \leftarrow 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

?

OK Cancel

- DR \leftarrow DR + 1

Edit Microinstructions

Type of Microinstruction: Increment

name	register	overflow...	carryBit	delta
AC<-AC+1	AC	(none)	(none)	1
AR<-AR+1	AR	(none)	(none)	1
DR<-DR+1	DR	(none)	(none)	1
PC<-PC+1	PC	(none)	(none)	1

New Delete Duplicate

?

OK Cancel

- $M[AR] \leftarrow DR$

Edit Microinstructions

Type of Microinstruction: MemoryAccess

name	direction	memory	data	address
M[AR]<-DR	write	main	DR	AR
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]<-PC	write	main	PC	AR

New Delete Duplicate

?

OK Cancel

- $\text{if } (DR \neq 0)$ (CPUSim skips a micro-operation when this is true)

Edit Microinstructions

Type of Microinstruction: Test

name	register	start	numBits	comparison	value	omission
if(DR!=0)	DR	0	16	NE	0	1
if(AC!=0)	AC	0	16	NE	0	1
if(E!=0)	E	0	1	NE	0	1
if[AC(15)!=0]	AC	0	1	NE	0	1
if[AC(15)!=1]	AC	0	1	NE	1	1

New Delete Duplicate

?

OK Cancel

- $PC \leftarrow PC + 1$

Edit Microinstructions

Type of Microinstruction: Increment

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

New Delete Duplicate

? OK Cancel

Edit Machine Instructions

Instructions

- ISZ_I
- ISZ
- BSA_I
- BSA
- BUN_I
- BUN
- STA_I
- STA
- LDA_I
- LDA
- ADD_I
- ADD
- AND_I
- AND

Format Implementation

Execute sequence

```

if(i=0)->DIRECT EXEC
AR <- M[AR]
DR <- M[AR]
DR<-DR+1
M[AR]<-DR
if(DR !=0)
PC<-PC+1
End

```

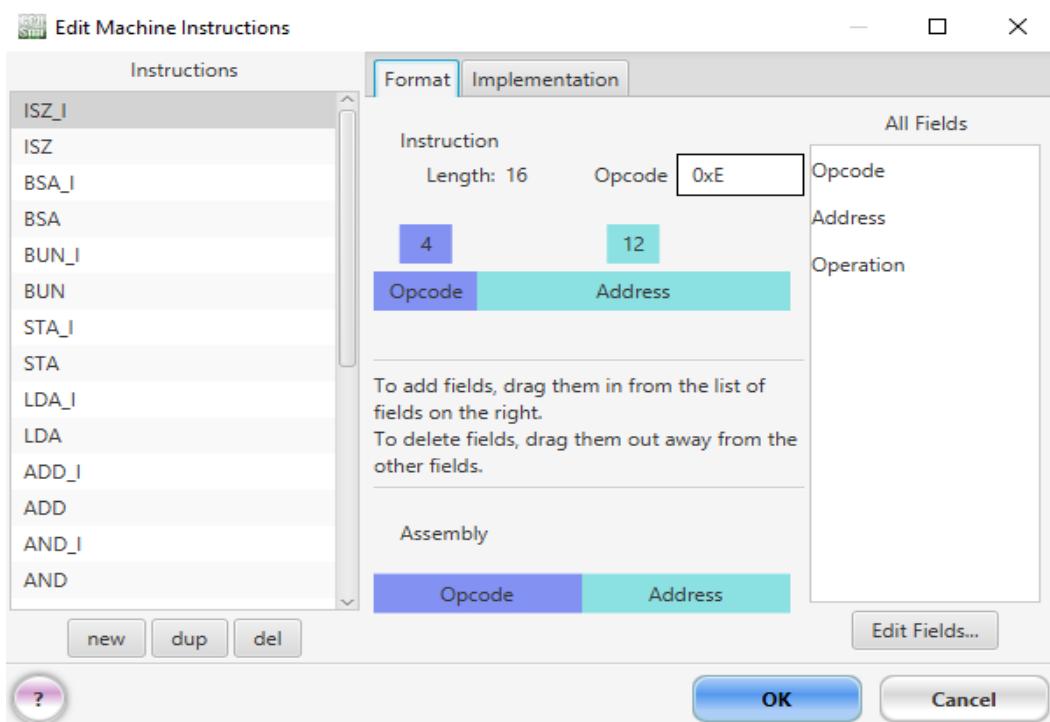
MicroInstructions

- decode
- end
- comment
- increment
- io
- logical
- memoryAccess
- set
- setCondBit
- shift
- test
- transferRtoR
- transferRtoA

? OK Cancel

ISZ_I - Indirect Addressing Mode

Implementation:



- AR $\leftarrow M[AR]$

The screenshot shows the 'Edit Microinstructions' dialog box. The 'Type of Microinstruction' dropdown is set to 'MemoryAccess'. The main area is a table with columns: name, direction, memory, data, and address. There are three rows:

name	direction	memory	data	address
AR $\leftarrow M[AR]$	read	main	AR	AR
DR<-M[AR]	read	main	DR	AR
IR<-M[AR]	read	main	IR	AR

 At the bottom, there are buttons for '?', 'New', 'Delete', 'Duplicate', 'OK', and 'Cancel'.

- DR \leftarrow 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

?

OK Cancel

- DR \leftarrow DR + 1

Edit Microinstructions

Type of Microinstruction: Increment

name	register	overflow...	carryBit	delta
AC<-AC+1	AC	(none)	(none)	1
AR<-AR+1	AR	(none)	(none)	1
DR<-DR+1	DR	(none)	(none)	1
PC<-PC+1	PC	(none)	(none)	1

New Delete Duplicate

?

OK Cancel



- $M[AR] \leftarrow DR$

Edit Microinstructions

Type of Microinstruction: MemoryAccess

name	direction	memory	data	address
M[AR]<-DR	write	main	DR	AR
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]<-PC	write	main	PC	AR

New Delete Duplicate

?

OK Cancel

- $\text{if } (DR \neq 0)$ (CPUSim skips a micro-operation when this is true)

Edit Microinstructions

Type of Microinstruction: Test

name	register	start	numBits	comparison	value	omission
if(DR!=0)	DR	0	16	NE	0	1
if(AC!=0)	AC	0	16	NE	0	1
if(E!=0)	E	0	1	NE	0	1
if[AC(15)!=0]	AC	0	1	NE	0	1
if[AC(15)!=1]	AC	0	1	NE	1	1

New Delete Duplicate

?

OK Cancel

- $PC \leftarrow PC + 1$

Edit Microinstructions

Type of Microinstruction: Increment

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

New Delete Duplicate

? OK Cancel

Edit Machine Instructions

Instructions

- ISZ_I
- ISZ
- BSA_I
- BSA
- BUN_I
- BUN
- STA_I
- STA
- LDA_I
- LDA
- ADD_I
- ADD
- AND_I

Format Implementation

Execute sequence

- AR <- M[AR]
- DR <- M[AR]
- DR<-DR+1
- M[AR]<-DR
- if(DR !=0)
- PC<-PC+1
- End

MicrolInstructions

- MicroInstructions
 - arithmetic
 - branch
 - decode
 - end
 - comment
 - increment
 - io
 - logical
 - memoryAccess
 - set
 - setCondBit

new dup del

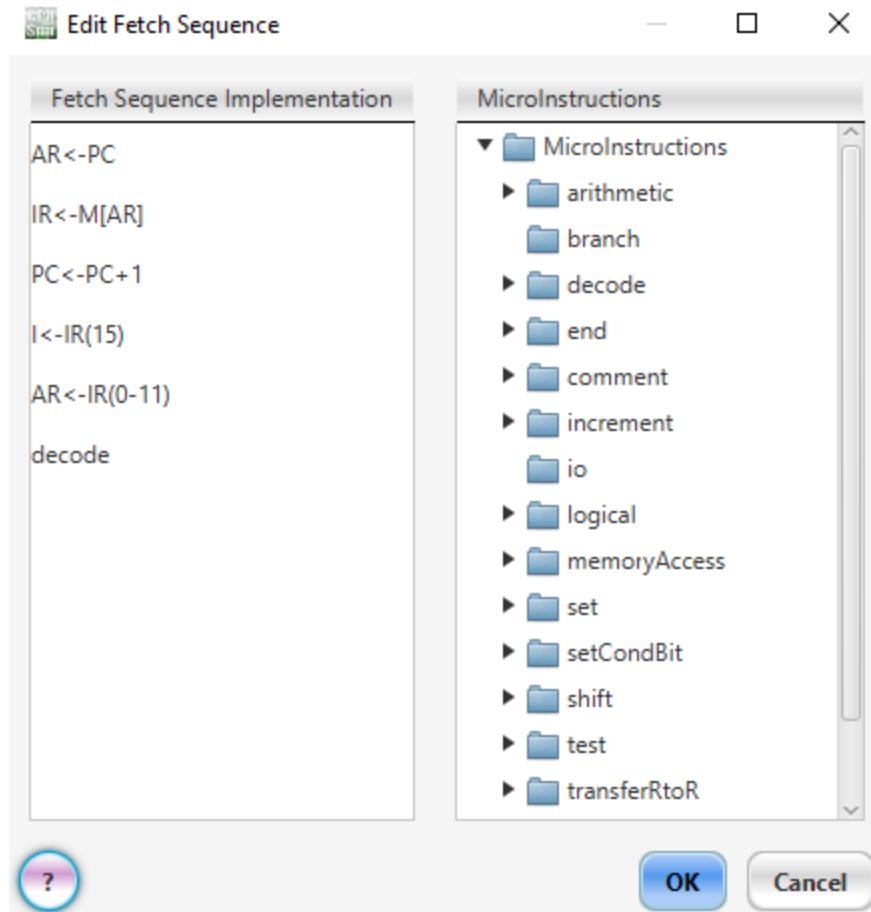
? OK Cancel



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

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

Fetch Sequence





Implementation of the Fetch Sequence:

- AR <- PC

Edit Microinstructions

Type of Microinstruction: TransferRtoR

name	source	srcStartBit	dest	destStartBit	numBits
AC(0)<-E	AC	15	AC	0	1
AC(15)<-E	E	0	AC	0	1
AR<-PC	PC	0	AR	0	12
E<-AC(0)	AC	15	E	0	1
E<-AC(15)	AC	0	E	0	1

New Delete Duplicate

OK Cancel

- IR <- M[AR]

Edit Microinstructions

Type of Microinstruction: MemoryAccess

name	direction	memory	data	address
IR<-M[AR]	read	main	IR	AR

New Delete Duplicate

OK Cancel

- $PC \leftarrow PC + 1$

Edit Microinstructions

Type of Microinstruction: Increment

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

New Delete Duplicate

? OK Cancel

- $I \leftarrow IR(15)$

Edit Microinstructions

Type of Microinstruction: TransferRtoR

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

New Delete Duplicate

? OK Cancel

- $AR \leftarrow IR(0-11)$

Edit Microinstructions

Type of Microinstruction: TransferRtoR

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

New Delete Duplicate

?

OK Cancel

- Decode IR

Edit Microinstructions

Type of Microinstruction: Decode

name	ir
decode	IR

New Delete Duplicate

?

OK Cancel

Objective

Q-1 Simulate the machine for the following memory-reference instructions with I = 0 (Direct) 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) AND | (b) ADD | (c) LDA | (d) STA |
| (e) BUN | (f) BSA | (g) ISZ | |



OBSERVATION

Before Loading

Registers

Name	Width	Data
AC	16	A937
AR	12	000
DR	16	0000
E	1	1
I	1	0
INPR	8	00
IR	16	0000
OUTR	8	00
PC	12	022
S	1	0
TR	16	0000

Memory

main	
Addr	Data
021	0000
022	0000
023	0000
024	0000
025	0000

main	
Addr	Data
081	0000
082	B8F2
083	0000
084	0000



AND

Assembly Program AND 0x082

Memory After Loading

main		
	Addr	Data
021	0000	
022	0082	
023	0000	
024	0000	

Registers After Execution

Registers		
Name	Width	Data
AC	16	A832
AR	12	082
DR	16	B8F2
E	1	1
I	1	0
INPR	8	00
IR	16	0082
OUTR	8	00
PC	12	023
S	1	0
TR	16	0000



ADD

Assembly Program ADD 0x082

Memory After Loading

main	
Addr	Data
021	0000
022	1082
023	0000
024	0000
025	0000

Registers After Execution

Registers		
Name	Width	Data
AC	16	6229
AR	12	082
DR	16	B8F2
E	1	1
I	1	0
INPR	8	00
IR	16	1082
OUTR	8	00
PC	12	023
S	1	1
TR	16	0000

| EXECUTION HALTED NORMALLY due to the setting of the bit(s) : [HALT]



LDA

Assembly Program | LDA 0x082

Memory After Loading

main		
	Addr	Data
	021	0000
	022	2082
	023	0000
	024	0000
	025	0000

Registers After Execution

Registers		
Name	Width	Data
AC	16	B8F2
AR	12	082
DR	16	B8F2
E	1	1
I	1	0
INPR	8	00
IR	16	2082
OUTR	8	00
PC	12	023
S	1	0
TR	16	0000



STA

Assembly Program `STA 0x082`

Memory After Loading

main		
Addr	Data	
021	0000	
022	3082	
023	0000	
024	0000	
025	0000	

Registers After Execution

Registers		
Name	Width	Data
AC	16	A937
AR	12	082
DR	16	0000
E	1	1
I	1	0
INPR	8	00
IR	16	3082
OUTR	8	00
PC	12	023
S	1	0
TR	16	0000

Memory After Execution

main		
Addr	Data	
081	0000	
082	A937	
083	0000	
084	0000	
085	0000	



BUN

Assembly Program ADD 0x082

Memory After Loading

main		
	Addr	Data
021	0000	
022	4082	
023	0000	
024	0000	
025	0000	

Registers After Execution

Registers		
Name	Width	Data
AC	16	A937
AR	12	082
DR	16	0000
E	1	1
I	1	0
INPR	8	00
IR	16	4082
OUTR	8	00
PC	12	082
S	1	0
TR	16	0000



BSA

Assembly Program BSA 0x082

Memory After Loading

main		
Addr	Data	
021	0000	
022	5082	
023	0000	
024	0000	
025	0000	

Registers After Execution

Registers		
Name	Width	Data
AC	16	A937
AR	12	083
DR	16	0000
E	1	1
I	1	0
INPR	8	00
IR	16	5082
OUTR	8	00
PC	12	083
S	1	0
TR	16	0000

Memory After Execution

main		
Addr	Data	
081	0000	
082	0023	
083	0000	
084	0000	
085	0000	



ISZ

Assembly Program ISZ 0x0B2

Memory After Loading

main		
Addr	Data	
021	0000	
022	6082	
023	0000	
024	0000	
025	0000	

Registers After Execution

Registers		
Name	Width	Data
AC	16	A937
AR	12	082
DR	16	B8F3
E	1	1
I	1	0
INPR	8	00
IR	16	6082
OUTR	8	00
PC	12	023
S	1	0
TR	16	0000

Memory After Execution

main		
Addr	Data	
081	0000	
082	B8F3	
083	0000	
084	0000	
085	0000	



Objective

Q-2 Simulate the machine for the memory-reference instructions referred in above question with I = 1 (Indirect) 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	Data
AC	16	A937
AR	12	000
DR	16	0000
E	1	1
I	1	0
INPR	8	00
IR	16	0000
OUTR	8	00
PC	12	026
S	1	0
TR	16	0000

Memory

main	
Addr	Data
021	0000
022	0000
023	0000
024	0000

main	
Addr	Data
296	0000
297	0000
298	B8F2
299	0000

main	
Addr	Data
081	0000
082	0298
083	0000
084	0000

AND

Assembly Program `AND_I 0x082`

Memory After Loading

main	
Addr	Data
024	0000
025	0000
026	8082
027	0000

Registers After Execution

Registers		
Name	Width	Data
AC	16	A832
AR	12	298
DR	16	B8F2
E	1	1
I	1	1
INPR	8	00
IR	16	8082
OUTR	8	00
PC	12	027
S	1	0
TR	16	0000

ADD

Assembly Program ADD_I 0x082

Memory After Loading

main	
Addr	Data
025	0000
026	9082
027	0000
028	0000

Registers After Execution

Registers		
Name	Width	Data
AC	16	6229
AR	12	298
DR	16	B8F2
E	1	1
I	1	1
INPR	8	00
IR	16	9082
OUTR	8	00
PC	12	027
S	1	1
TR	16	0000

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



LDA

Assembly Program LDA_I 0x082

Memory After Loading

main	
Addr	Data
024	0000
025	0000
026	A082
027	0000

Registers After Execution

Registers		
Name	Width	Data
AC	16	B8F2
AR	12	298
DR	16	B8F2
E	1	1
I	1	1
INPR	8	00
IR	16	A082
OUTR	8	00
PC	12	027
S	1	0
TR	16	0000



STA

Assembly Program `STA_I 0x082`

Memory After Loading

main		
Addr	Data	
024	0000	
025	0000	
026	B082	
027	0000	

Registers After Execution

Registers		
Name	Width	Data
AC	16	A937
AR	12	298
DR	16	0000
E	1	1
I	1	1
INPR	8	00
IR	16	B082
OUTR	8	00
PC	12	027
S	1	0
TR	16	0000

Memory After Execution

main		
Addr	Data	
297	0000	
298	A937	
299	0000	
29A	0000	



BUN

Assembly Program `BUN_I 0x0B2`

Memory After Loading

main		
	Addr	Data
	024	0000
	025	0000
	026	C082
	027	0000

Registers After Execution

Registers		
Name	Width	Data
AC	16	A937
AR	12	298
DR	16	0000
E	1	1
I	1	1
INPR	8	00
IR	16	C082
OUTR	8	00
PC	12	298
S	1	0
TR	16	0000



BSA

Assembly Program BSA_I 0x082

Memory After Loading

main		
Addr	Data	
025	0000	
026	D082	
027	0000	
028	0000	

Registers After Execution

Registers		
Name	Width	Data
AC	16	A937
AR	12	299
DR	16	0000
E	1	1
I	1	1
INPR	8	00
IR	16	D082
OUTR	8	00
PC	12	299
S	1	0
TR	16	0000

Memory After Execution

main		
Addr	Data	
297	0000	
298	0027	
299	0000	
29A	0000	



ISZ

Assembly Program ISZ_I 0x082

Memory After Loading

main		
Addr	Data	
024	0000	
025	0000	
026	E082	
027	0000	

Registers After Execution

Registers		
Name	Width	Data
AC	16	A937
AR	12	298
DR	16	B8F3
E	1	1
I	1	1
INPR	8	00
IR	16	E082
OUTR	8	00
PC	12	027
S	1	0
TR	16	0000

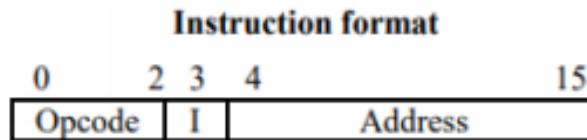
Memory After Execution

main		
Addr	Data	
297	0000	
298	B8F3	
299	0000	
29A	0000	



Objective

Q-3 Modify the machine created in Practical 1 according to the following 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

Edit Fields

Name	Type	NumBits	DefaultVal	Relativity	Signed
Mode	required	1	0	absolute	<input type="checkbox"/>
Opcode	required	3	0	absolute	<input checked="" type="checkbox"/>
Address	required	12	0	absolute	<input checked="" type="checkbox"/>
Operation	required	16	0	absolute	<input checked="" type="checkbox"/>

New Delete Duplicate Values... OK Cancel

a) Instruction Format

Edit Machine Instructions

Instructions		Format	Implementation
ADD_m		Instruction	All Fields
ISZ_I		Length: 16	Opcode_m
ISZ		3	Mode
BSA_I		1	Opcode
BSA		12	Address
BUN_I		Opcode_m M...	Operation
BUN		Address	
STA_I			
STA			
LDA_I			
LDA			
ADD_I			
		Assembly	
		Opcode_m Mode Address	
			Edit Fields...
?		OK	Cancel

b) Registers

Edit Modules

Type of Module: Register

name	width	initial value	read-only
AC	16	0	<input type="checkbox"/>
AR	12	0	<input type="checkbox"/>
DR	16	0	<input type="checkbox"/>
E	1	0	<input type="checkbox"/>
I	1	0	<input type="checkbox"/>
INPR	8	0	<input type="checkbox"/>
IR	16	0	<input type="checkbox"/>

New Delete Duplicate Properties... ? OK Cancel

Fetch Sequence

Edit Fetch Sequence

Fetch Sequence Implementation

- AR<-PC
- IR<-M[AR]
- PC<-PC+1
- I<-IR(12)
- AR<-IR(0-11)
- decode

MicrolInstructions

- ▼ MicrolInstructions
 - arithmetic
 - branch
 - decode
 - end
 - comment
 - increment
 - io
 - logical
 - memoryAccess
 - set
 - setCondBit
 - shift
 - test

? OK Cancel

The screenshot shows a software interface titled "Edit Microinstructions". At the top, there is a dropdown menu labeled "Type of Microinstruction: TransferRtoR". Below the dropdown is a table with columns: name, source, srcStartBit, dest, destStartBit, and numBits. The table contains nine rows of microinstruction details. The last row, "I<-IR(12)", is highlighted with a blue background. At the bottom of the dialog are buttons for "New", "Delete", "Duplicate", "?", "OK", and "Cancel".

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

Checking Addressing Mode (As done before in all Memory Reference Instructions)

1. If (i = 0)->DIRECT EXE (CPUSim skips a micro-operation when this is true)

The screenshot shows a software interface titled "Edit Microinstructions". At the top, there is a dropdown menu labeled "Type of Microinstruction: Test". Below the dropdown is a table with columns: name, register, start, numBits, comparison, value, and omission. The table contains six rows of microinstruction details. The fourth row, "if(i=0)->DIREC...", is highlighted with a blue background. At the bottom of the dialog are buttons for "New", "Delete", "Duplicate", "?", "OK", and "Cancel".

name	register	start	numBits	comparison	value	omission
if(AC!=0)	AC	0	16	NE	0	1
if(DR !=0)	DR	0	16	NE	0	1
if(E!=0)	E	0	1	NE	0	1
if(i=0)->DIRECT...	I	0	1	EQ	0	1
if[AC(15)!=0]	AC	15	1	NE	0	1
if[AC(15)!=1]	AC	15	1	NE	0	1

2. 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

?

OK Cancel

3. Remaining Microinstructions same as before

Implementation Of ADD

Edit Machine Instructions

Instructions

- ADD_m
- ISZ_I
- ISZ
- BSA_I
- BSA
- BUN_I
- BUN
- STA_I
- STA
- LDA_I
- LDA

Format Implementation

Execute sequence

```
if(i=0)->DIRECT EXEC
AR <- M[AR]
DR <- M[AR]
AC <- AC+DR
End
```

MicroInstructions

- MicroInstructions
 - arithmetic
 - branch
 - decode
 - end
 - comment
 - increment
 - io
 - logical
 - memoryAccess

new dup del

?

OK Cancel



Assembly Code

ADD_m 1 0x082

Memory Before Loading

main	
Addr	Data
021	0000
022	0000
023	0000
024	0000

main	
Addr	Data
081	0000
082	0298
083	0000
084	0000

main	
Addr	Data
296	0000
297	0000
298	B8F2
299	0000

Memory After Loading

main	
Addr	Data
021	0000
022	3082
023	0000
024	0000



Register After Execution

Registers		
Name	Width	Data
AC	16	6229
AR	12	298
DR	16	B8F2
E	1	1
I	1	1
INPR	8	00
IR	16	3082
OUTR	8	00
PC	12	023
S	1	1
TR	16	0000

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