

CISCO SIMULATOR

Manual

V 2.1

Group 8

Md Shahjalal
Tianyou Bao
Xuzheng Lu

Contents

1 Introduction	1
1.1 Debugging Panel	1
1.1.1 Register Indicators Area	1
1.1.2 Memory Area	2
1.1.3 Controller Area	2
1.2 Operation Panel (Console)	3
1.3 Classic Panel	4
1.4 Themes	4
2 Basic Operations	5
2.1 Writing Values to Registers	5
2.2 Writing Values to Memory	5
2.2.1 Using Memory Address Register and Memory Buffer Register	6
2.2.2 Modifying the Memory Area	6
2.3 Executing Instructions	7
2.3.1 Executing Instructions Step-by-Step	7
2.3.1 Executing Instructions Automatically	8
3 Executing Programs	10
3.1 Executing Program 1	10
3.1.1 Program Description	10
3.1.2 Running the Program	10
3.2 Executing Program 2	11
3.3 Executing a Custom Program Using IPL	11
3.3.1 Using Operation Panel (Console)	11
3.3.2 Using Debugging Panel	12
4 Instructions Reference	12
4.1 Load/Store Instructions	12
4.1.1 (01) LDR	12
4.1.2 (02) STR	12
4.1.3 (03) LDA	13
4.1.4 (41) LDX	13
4.1.5 (42) STX	13
4.2 Arithmetic and Logical Instructions	13
4.2.1 (04) AMR	14
4.2.2 (05) SMR	14
4.2.3 (06) AIR	14
4.2.4 (07) SIR	15
4.2.5 (20) MLT	15
4.2.6 (21) DVD	15
4.2.7 (22) TRR	15

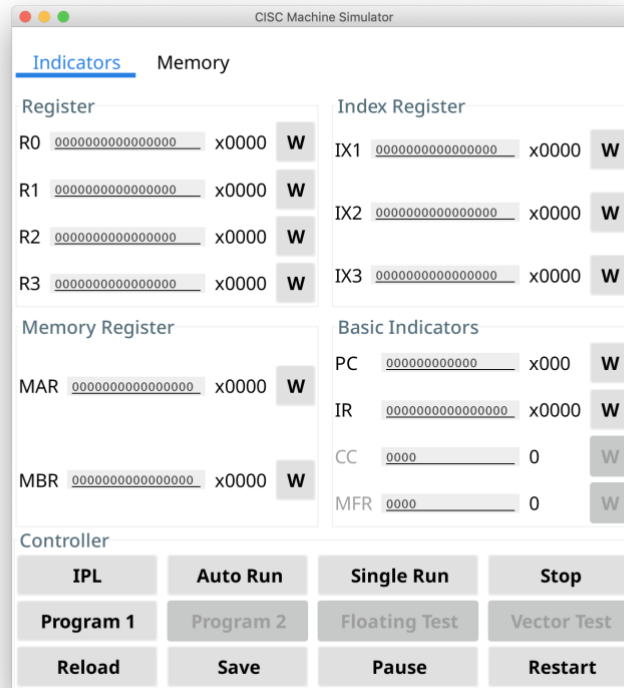
4.2.8 (23) AND	16
4.2.9 (24) ORR.....	16
4.2.10 (25) NOT.....	16
4.3 Transfer Instructions.....	16
4.3.1 (10) JZ.....	17
4.3.2 (11) JNE.....	17
4.3.3 (12) JCC	17
4.3.4 (13) JMA.....	17
4.3.5 (14) JSR	18
4.3.6 (15) RFS.....	18
4.3.7 (16) SOB	18
4.3.8 (17) JGE.....	18
4.4 Shift/Rotate Instructions	19
4.4.1 (31) SRC	19
4.4.2 (32) RRC.....	19
4.5 I/O Instructions	19
4.5.1 (61) IN.....	20
4.5.2 (62) OUT.....	20
4.5.3 (63) CHK	20
4.6 Other Instructions.....	21
4.6.1 (00) HALT	21

1 Introduction

This simulator is a simulation of a Complex Instruction Set Computer (CISC). Three panels are designed for the simulator, and two themes are supported.

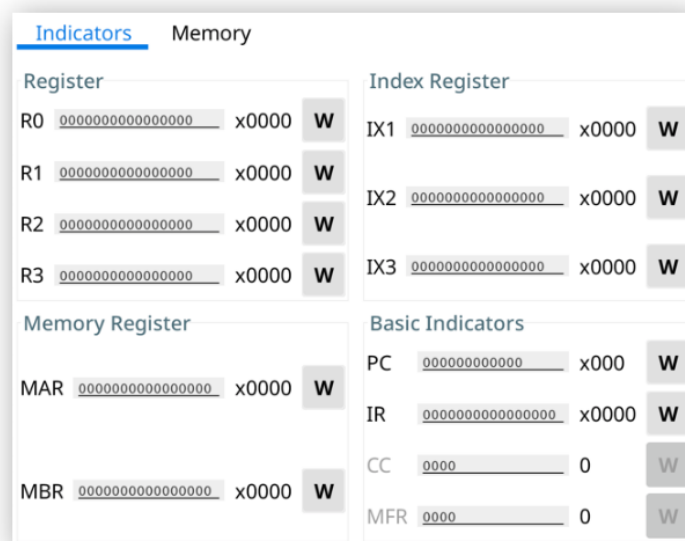
1.1 Debugging Panel

Debugging Panel displays all the information about the Registers, Indicators, and Memory in the computer and can be written manually.



The panel is divided into three parts:

1.1.1 Register Indicators Area

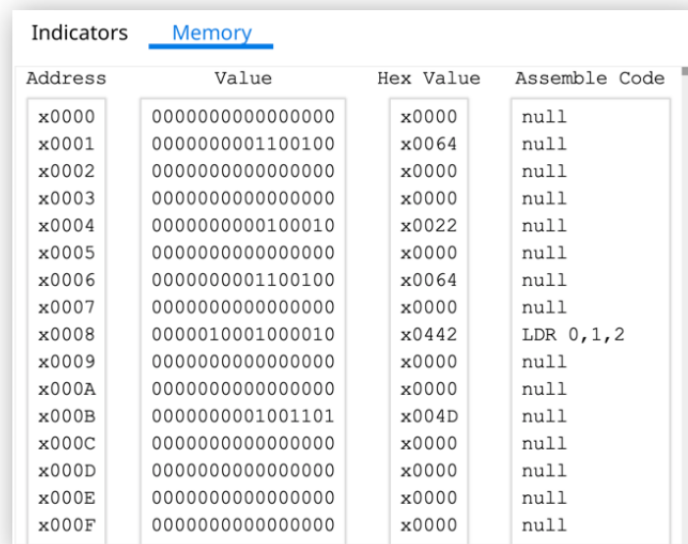


The Register Indicators display the values of all kinds of registers.

- Click the 'W' button to manually modify the value of a register.
- Hexadecimal values are shown on the right.

Type	Size(bits)	Number	Description
R0...R3	16	4	General-Purpose Register
IX1...IX3	16	3	Index Register
MAR	16	1	Memory Address Register
MBR	16	1	Memory Buffer Register
PC	12	1	Program Counter
IR	16	1	Instruction Register
CC	4	1	Condition Code
MFR	4	1	Machine Fault Register

1.1.2 Memory Area




Address	Value	Hex Value	Assemble Code
x0000	0000000000000000	x0000	null
x0001	0000000001100100	x0064	null
x0002	0000000000000000	x0000	null
x0003	0000000000000000	x0000	null
x0004	0000000000100010	x0022	null
x0005	0000000000000000	x0000	null
x0006	0000000001100100	x0064	null
x0007	0000000000000000	x0000	null
x0008	0000010001000010	x0442	LDR 0,1,2
x0009	0000000000000000	x0000	null
x000A	0000000000000000	x0000	null
x000B	0000000001001101	x004D	null
x000C	0000000000000000	x0000	null
x000D	0000000000000000	x0000	null
x000E	0000000000000000	x0000	null
x000F	0000000000000000	x0000	null

The Memory Area shows the address, the value, the Hexadecimal value, and the Assemble Code of each line on memory.

- The memory address pointed by the Program Counter will be highlighted.
- Double click to manually modify the binary value of a memory row.

1.1.3 Controller Area

The Controller Area integrates all function buttons and the instruction input box.



IPL	Auto Run	Single Run	Stop
Program 1	Program 2	Floating Test	Vector Test
Reload	Save	Pause	Restart

Functions of the buttons in Controller Area:

Button	Function
IPL	Pre-load a program from I/O
Auto Run	Run the instructions until TRAP or HALT
Single Run	Run one instruction
Stop	Stop the workload on the machine
Program 1	Run Program 1
Reload	Reload the data from memory.txt to memory
Save	Save the data in memory to memory.txt
Pause	Pause the workload on the machine
Restart	Restart the machine

1.2 Operation Panel (Console)

Operation Panel is a console used for system operation through the command line.



Commands supported:

Command	Description
autorun	Run the instructions until TRAP or HALT (Same as 'auto run' and 'run')
auto run	Run the instructions Until TRAP or HALT (Same as 'autorun' and 'run')
clean	Clean the console (same as 'cls')
cls	Clean the console (same as 'clean')
exit	Shutdown the machine (same as 'quit' and 'power off')
floating test	Run the Floating Test
ipl	Load the program from I/O
pause	Pause the workload on the machine (not finished)
power off	Shutdown the machine (same as 'exit' and 'quit')
Program1	Run Program 1 (same as 'Program 1')
Program 1	Run Program 1 (same as 'Program1')
Program2	Run Program 1 (same as 'Program 2')

Program 2	Run Program 1 (same as 'Program2')
quit	Shutdown the machine (same as 'exit' and 'power off')
reload	Reload the data from memory.txt to memory
reset	Restart the machine (Same as 'restart')
restart	Restart the machine (Same as 'reset')
run	Run the instructions until TRAP or HALT (same as 'autorun' and 'auto run')
save	Save the data in memory to memory.txt
singlerun	Run one instruction (Same as 'single run')
single run	Run one instruction (Same as 'singlerun')
status	Show the status of the machine
stop	Stop the workload on the machine
switch theme	switch theme {\$THEME_NAME} Switch the theme of UI. Now support 'Material Design Ocean' (or 'MaterialDesignOcean') and 'Material Design Lighter' (or 'MaterialDesignLighter')
vector test	Run the Vector Test
/help	Show the command list

1.3 Classic Panel

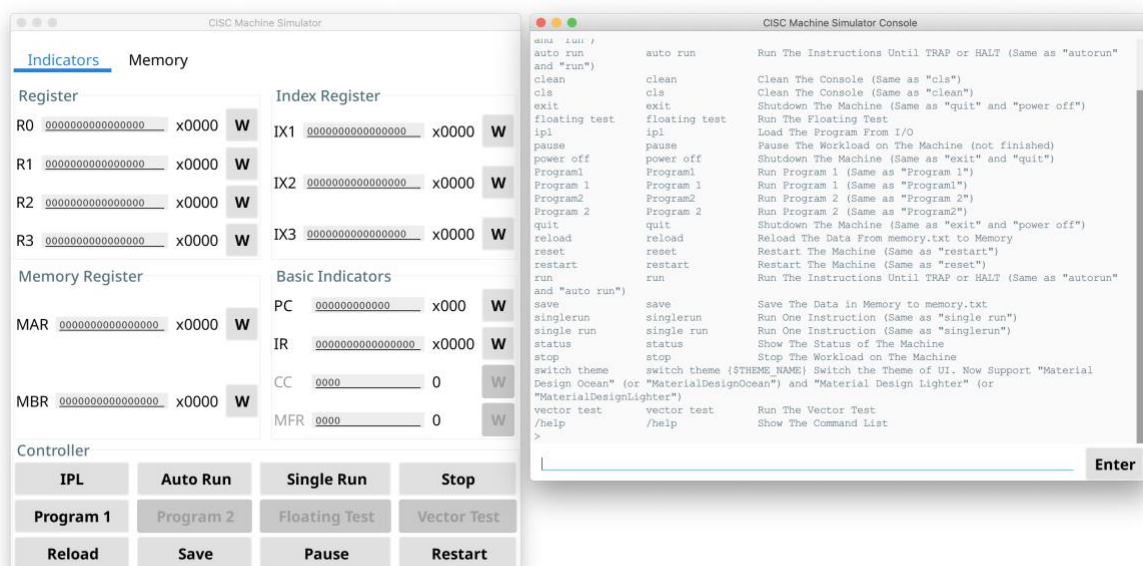
The appearance and operational logic of the **Classic Panel** emulate the PDP-8 computer. Users will use switches to input and lights for indication.

The **Classic Panel** has not been finished yet and will be released in the next version.

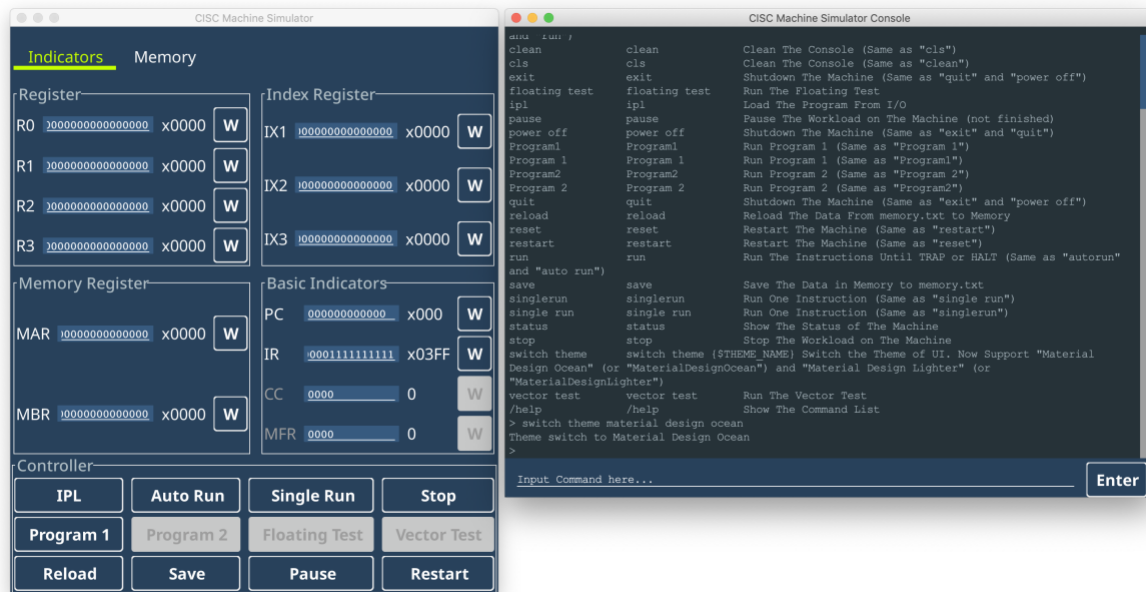
1.4 Themes

Two themes, Material Design Ocean and Material Design Lighter, are supported now. To change the theme, input 'switch theme {\$THEME_NAME}' in **Operation Panel**.

Material Design Lighter Theme (default)



Material Design Ocean Theme



2 Basic Operations

2.1 Writing Values to Registers

Following the steps below to write a value to a register.

Step 1: Input a value into the box.

R0 0000000000001111 x0000 W

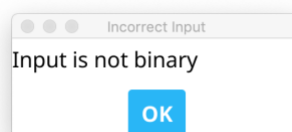
Step 2: Click the 'W' button at right to write the value to the register.

R0 0000000000001111 x000F W

Step 3: Done! The value will be written to the Register.

Error handling:

- Input too long: Remove the excess bits from the left
- Input too short: Add zeros from the left
- Input is not binary: Pop up an Error window



2.2 Writing Values to Memory

Two methods are acceptable to write a value to the Memory.

2.2.1 Using Memory Address Register and Memory Buffer Register

Step 1: Input a value into the MAR box.

MAR 0000000000000011 x0003 W

Step 2: Click the 'W' button of MAR.

MAR 0000000000000011 x0003 W

Step 3: Input a value into the MBR box.

MBR 0000100010000100 x0000 W

Step 4: Click the 'W' button of MBR.

MBR 0000000000000000 x0000 W

Step 5: Done! The value of MAR will be written to the Memory, and the MAR will automatically change to the next address.

MAR 0000000000000100 x0004 W

MBR 0000000000000000 x0000 W

Indicators <u>Memory</u>			
Address	Value	Hex Value	Assemble Code
x0000	0000000000000000	x0000	null
x0001	0000000001100100	x0064	null
x0002	0000000000000000	x0000	null
x0003	0000100010000100	x0884	STR 0,2,4
x0004	0000000000100010	x0022	null

2.2.2 Modifying the Memory Area

Step 1: Double click the memory row that needs to modify.

Indicators <u>Memory</u>			
Address	Value	Hex Value	Assemble Code
x0000	0000000000000000	x0000	null
x0001	0000000001100100	x0064	null
x0002	0000000000000000	x0000	null
x0003	0000100010000100	x0884	STR 0,2,4
x0004	0000000000100010	x0022	null
x0005	0000000000000000	x0000	null
x0006	0000000001100100	x0064	null
x0007	0000000000000000	x0000	null

Step 2: An window as the following will pop up. Input the value to be written to the memory.

Input binary value
 0000110101100011

OK Cancel

Step 3: Click the 'OK' button, and then the value will be written to the Memory.

Indicators <u>Memory</u>			
Address	Value	Hex Value	Assemble Code
x0000	0000000000000000	x0000	null
x0001	0000000001100100	x0064	null
x0002	0000000000000000	x0000	null
x0003	0000100010000100	x0884	STR 0,2,4
x0004	0000000000100010	x0022	null
x0005	0000110101100011	x0D63	LDA 1,1,3,1
x0006	0000000001100100	x0064	null

2.3 Executing Instructions

Instruction can be executed step-by-step or automatically.

2.3.1 Executing Instructions Step-by-Step

Step 1: Store an instruction to the Memory.

Indicators <u>Memory</u>			
x001C	0000000000000000	x0000	null
x001D	0000000000000000	x0000	null
x001E	1010010001010100	xA454	LDX 1,20
x001F	1010010010010110	xA496	LDX 2,22
x0020	1010010011111000	xA4F8	LDX 3,24,1
x0021	0000011100001011	x070B	LDR 3,0,11

Step 2: Write the address of the instruction to the Program Counter (PC).

PC 000000011110 x01E

W

Step 3: Click the ‘Single Run’ button, and then the instruction will be executed.

- The Program Counter will automatically point to the next address of Memory.
- The Instruction Register will store the last executed instruction.

Indicators		Memory	
x001C	0000000000000000	x0000	null
x001D	0000000000000000	x0000	null
x001E	1010010001010100	xA454	LDX 1,20
x001F	1010010010010110	xA496	LDX 2,22
x0020	1010010011111000	xA4F8	LDX 3,24,1
x0021	0000011100001011	x070B	LDR 3,0,11

Indicators

Memory

Register

R0

0000000000000000

x0000

W

R1

0000000000000000

x0000

W

R2

0000000000000000

x0000

W

R3

0000000000000000

x0000

W

Memory Register

MAR

0000000010100

x0014

W

MBR

0000001010000

x0050

W

Index Register

IX1

000000001010000

x0050

W

IX2

0000000000000000

x0000

W

IX3

0000000000000000

x0000

W

Basic Indicators

PC

000000011111

x01F

W

IR

0010001010100

xA454

W

CC

0000

0

W

MFR

0000

0

W

2.3.1 Executing Instructions Automatically

Step 1: Store instructions to the Memory.

Indicators		Memory	
x001B	0000000001100100	x0064	null
x001C	0000000000000000	x0000	null
x001D	0000000000000000	x0000	null
x001E	1010010001010100	xA454	LDX 1,20
x001F	1010010010010110	xA496	LDX 2,22
x0020	1010010011111000	xA4F8	LDX 3,24,1
x0021	0000011100001011	x070B	LDR 3,0,11
x0022	0000010000101011	x042B	LDR 0,0,11,1
x0023	0000010111000011	x05C3	LDR 1,3,3
x0024	0000011011100011	x06E3	LDR 2,3,3,1
x0025	0000101000000001	x0A01	STR 2,0,1
x0026	1010100011010000	xA8D0	STX 3,16
x0027	0000110100000100	x0D04	LDA 1,0,4
x0028	0001000100000001	x1101	AMR 1,0,1

Step 2: Write the address of the **starting** instruction to the Program Counter (PC).

PC x01E

Step 3: Click the 'Auto Run' button, and then the instructions will be executed automatically.

- The Program Counter will automatically point to the next address of Memory after an instruction being executed.
- All the indicators will be continuously updated while the program is running.

CISC Machine Simulator

Indicators Memory

Register

R0	<input type="text" value="0000000000000000"/>	x0000	<input type="button" value="W"/>
R1	<input type="text" value="000000000010101"/>	x002B	<input type="button" value="W"/>
R2	<input type="text" value="11111111001010"/>	xFF95	<input type="button" value="W"/>
R3	<input type="text" value="000000000010101"/>	x002B	<input type="button" value="W"/>

Index Register

IX1	<input type="text" value="00000000010100"/>	x0050	<input type="button" value="W"/>
IX2	<input type="text" value="00000000010111"/>	x005F	<input type="button" value="W"/>
IX3	<input type="text" value="00000000011001"/>	x0064	<input type="button" value="W"/>

Memory Register

MAR	<input type="text" value="0000000000000000"/>	x0000	<input type="button" value="W"/>
MBR	<input type="text" value="0000000000000000"/>	x0000	<input type="button" value="W"/>

Basic Indicators

PC	<input type="text" value="000000110010"/>	x032	<input type="button" value="W"/>
IR	<input type="text" value="0110011001000"/>	x6640	<input type="button" value="W"/>
CC	<input type="text" value="0000"/>	0	<input type="button" value="W"/>
MFR	<input type="text" value="0000"/>	0	<input type="button" value="W"/>

Controller

<input type="button" value="IPL"/>	<input type="button" value="Auto Run"/>	<input type="button" value="Single Run"/>	<input type="button" value="Stop"/>
<input type="button" value="Program 1"/>	<input type="button" value="Program 2"/>	<input type="button" value="Floating Test"/>	<input type="button" value="Vector Test"/>
<input type="button" value="Reload"/>	<input type="button" value="Save"/>	<input type="button" value="Pause"/>	<input type="button" value="Restart"/>

Indicators **Memory**

x002F	0101110011000000	x5CC0	AND 0,3
x0030	0110000111000000	x61C0	ORR 1,3
x0031	0110011001000000	x6640	NOT 2
x0032	0000000000000000	x0000	null

HALT

Step 4: Click the 'Pause' button to pause the program or the 'Stop' button to stop the program.

3 Executing Programs

3.1 Executing Program 1

3.1.1 Program Description

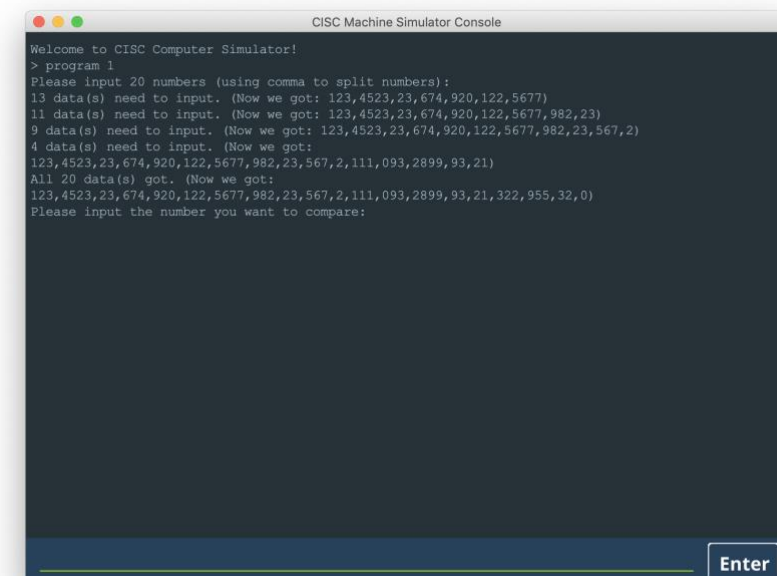
Program 1 is a program that reads 20 numbers (integers) from the keyboard, prints the numbers to the console printer, requests a number from the user, and searches the 20 numbers read in for the number closest to the number entered by the user. The numbers distributed over the range of 0 ... 65535. Print the number entered by the user and the number closest to that number.

3.1.2 Running the Program

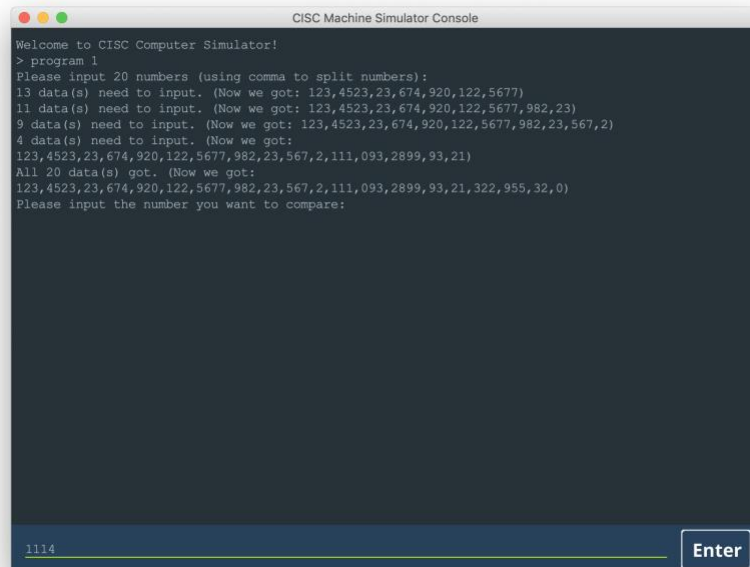
Step 1: Input 'program 1' or 'program1' in the **Operation Panel (Console)** and then click the 'Enter' button. Or click the 'Program 1' button in the **Debugging Panel**.



Step 2: Input numbers (use comma to split numbers) and then click the 'Enter' button. You can fill the numbers several times to input the required 20 numbers.



Step 3: Input the number for comparing.

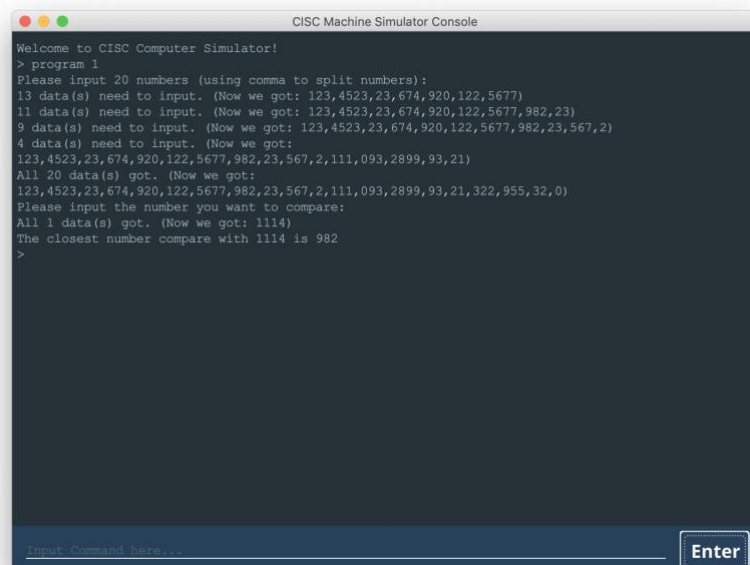


```

Welcome to CISC Computer Simulator!
> program 1
Please input 20 numbers (using comma to split numbers):
13 data(s) need to input. (Now we got: 123,4523,23,674,920,122,5677)
11 data(s) need to input. (Now we got: 123,4523,23,674,920,122,5677,982,23)
9 data(s) need to input. (Now we got: 123,4523,23,674,920,122,5677,982,23,567,2)
4 data(s) need to input. (Now we got:
123,4523,23,674,920,122,5677,982,23,567,2,111,093,2899,93,21)
All 20 data(s) got. (Now we got:
123,4523,23,674,920,122,5677,982,23,567,2,111,093,2899,93,21,322,955,32,0)
Please input the number you want to compare:
1114
Enter

```

Step 4: Done! The result of the calculation will be output to the **Console**.



```

Welcome to CISC Computer Simulator!
> program 1
Please input 20 numbers (using comma to split numbers):
13 data(s) need to input. (Now we got: 123,4523,23,674,920,122,5677)
11 data(s) need to input. (Now we got: 123,4523,23,674,920,122,5677,982,23)
9 data(s) need to input. (Now we got: 123,4523,23,674,920,122,5677,982,23,567,2)
4 data(s) need to input. (Now we got:
123,4523,23,674,920,122,5677,982,23,567,2,111,093,2899,93,21)
All 20 data(s) got. (Now we got:
123,4523,23,674,920,122,5677,982,23,567,2,111,093,2899,93,21,322,955,32,0)
Please input the number you want to compare:
All 1 data(s) got. (Now we got: 1114)
The closest number compare with 1114 is 982
>
Input Command here...
Enter

```

3.2 Executing Program 2

Not implemented yet.

3.3 Executing a Custom Program Using IPL

3.3.1 Using Operation Panel (Console)

Step 1: Write the custom program in a text file.

Step 2: Input 'ipl' command to the console to import the program to the memory.

Step 4: Input 'auto run' or 'autorun' command to the console, and then the program will be executed. Or input 'single run' or 'singlerun' command to run the program step-by-step.

3.3.2 Using Debugging Panel

Step 1: Write the custom program in a text file.

Step 2: Click the 'IPL' button to import the program to the memory.

Step 3: Click the 'Auto Run' button, and then the program will be executed. Or click the 'Single Run' button to run the program step-by-step.

4 Instructions Reference

4.1 Load/Store Instructions

The instructions to load/store values from/to Registers or Memory. The binary instruction code format of Load/Store Instructions is as follows:

Opcode	R	IX	I	Address
0 5	6 7	8 9	1 1	1 5
		0 1		

Opcode: 6 bits Specifies the instruction
R: 2 bits Specifies the General-Purpose Register
IX: 2 bits Specifies the Index Register
I: 1 bit Specifies Indirect Addressing
 If I = 1, indirect addressing; otherwise, no indirect addressing.
Address: 5 bits Specifies the location

4.1.1 (01) LDR

000001	R	IX	I	Address
0 5	6 7	8 9	1 1	1 5
		0 1		

Instruction: LDR r, x, address[, I]
 Octal-Opcode: 01
 Binary-Opcode: 000001
 Function: Loads Register from Memory

4.1.2 (02) STR

000010	R	IX	I	Address
0 5	6 7	8 9	1 1	1 5
		0 1		

Instruction: STR r, x, address[, I]
 Octal-Opcode: 02
 Binary-Opcode: 000010
 Function: Stores Register to Memory

4.1.3 (03) LDA

000011	R	IX	I	Address
0 5	6 7	8 9	1 1	1 5
		0 1		

Instruction: LDA r, x, address[, I]
Octal-Opcode: 03
Binary-Opcode: 000011
Function: Loads Register with Address

4.1.4 (41) LDX

101001	R	IX	I	Address
0 5	6 7	8 9	1 1	1 5
		0 1		

Instruction: LDX x, address[, I]
Octal-Opcode: 41
Binary-Opcode: 101001
Function: Loads Index Register from Memory

4.1.5 (42) STX

101010	R	IX	I	Address
0 5	6 7	8 9	1 1	1 5
		0 1		

Instruction: STX x, address[, I]
Octal-Opcode: 42
Binary-Opcode: 101010
Function: Stores Index Register to Memory

4.2 Arithmetic and Logical Instructions

The instructions to perform most of the computational works in the machine.

The binary instruction code format of basic Arithmetic and Logical Instructions is as follows:

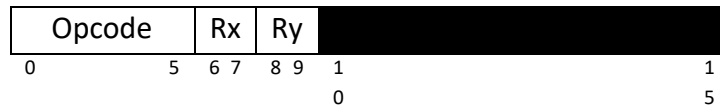
Opcode	R	IX	I	Address
0 5	6 7	8 9	1 1	1 5
		0 1		

Opcode: 6 bits Specifies the instruction
R: 2 bits Specifies the General-Purpose Register
IX: 2 bits Specifies the Index Register

I: 1 bit Specifies Indirect Addressing
If I =1, indirect addressing; otherwise, no indirect addressing.

Address: 5 bits Specifies the location

The binary instruction code format of register-to-register Arithmetic and Logical Instructions is as follows:

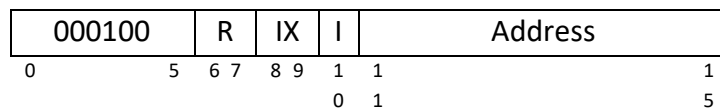


Opcode: 6 bits Specifies the instruction

Rx: 2 bits Specifies the General-Purpose Register x

Ry: 2 bits Specifies the General-Purpose Register y

4.2.1 (04) AMR



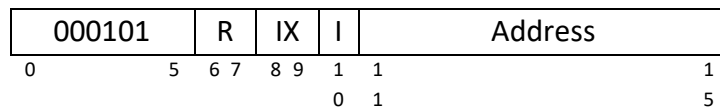
Instruction: AMR r, x, address[, I]

Octal-Opcode: 04

Binary-Opcode: 000100

Function: Add Memory to Register

4.2.2 (05) SMR



Instruction: SMR r, x, address[, I]

Octal-Opcode: 05

Binary-Opcode: 000101

Function: Subtract Memory from Register

4.2.3 (06) AIR

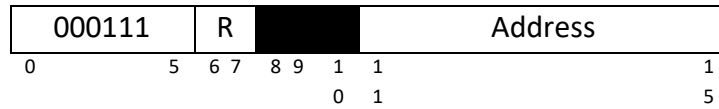


Instruction: AIR r, immed

Octal-Opcode: 06

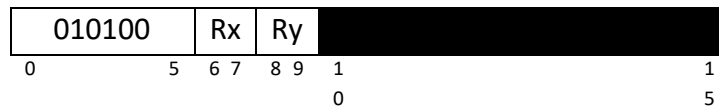
Binary-Opcode: 000110
Function: Add Immediate to Register

4.2.4 (07) SIR



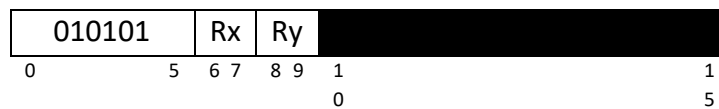
Instruction: SIR r, immed
Octal-Opcode: 07
Binary-Opcode: 000111
Function: Subtract Immediate from Register

4.2.5 (20) MLT



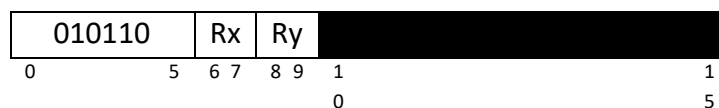
Instruction: MLT rx, ry
Octal-Opcode: 20
Binary-Opcode: 010100
Function: Multiply Register by Register

4.2.6 (21) DVD



Instruction: DVD rx, ry
Octal-Opcode: 21
Binary-Opcode: 010101
Function: Divide Register by Register

4.2.7 (22) TRR



Instruction: TRR rx, ry
Octal-Opcode: 22
Binary-Opcode: 010110

Function: Test the Equality of Register and Register

4.2.8 (23) AND



Instruction: AND rx, ry

Octal-Opcode: 23

Binary-Opcode: 010111

Function: Logical AND of Register and Register

4.2.9 (24) ORR



Instruction: ORR rx, ry

Octal-Opcode: 24

Binary-Opcode: 011000

Function: Logical OR of Register and Register

4.2.10 (25) NOT



Instruction: NOT rx

Octal-Opcode: 25

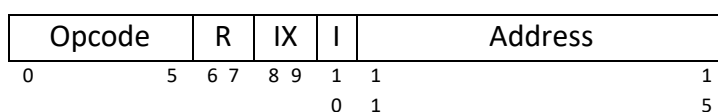
Binary-Opcode: 011001

Function: Logical NOT of Register to Register

4.3 Transfer Instructions

The instructions to check the value of a register and then change the control of program execution.

The binary instruction code format of Transfer Instructions is as follows:



Opcode: 6 bits Specifies the instruction
R: 2 bits Specifies the General-Purpose Register
IX: 2 bits Specifies the Index Register
I: 1 bit Specifies Indirect Addressing
 If I =1, indirect addressing; otherwise, no indirect addressing.
Address: 5 bits Specifies the location

4.3.1 (10) JZ

001010	R	IX	I	Address
0 5	6 7	8 9	1 1	1
			0 1	5

Instruction: JZ r, x, address[, I]
 Octal-Opcode: 10
 Binary-Opcode: 001010
 Function: Jump if Zero

4.3.2 (11) JNE

001011	R	IX	I	Address
0 5	6 7	8 9	1 1	1
			0 1	5

Instruction: JNE r, x, address[, I]
 Octal-Opcode: 11
 Binary-Opcode: 001011
 Function: Jump if Not Equal

4.3.3 (12) JCC

001100	R	IX	I	Address
0 5	6 7	8 9	1 1	1
			0 1	5

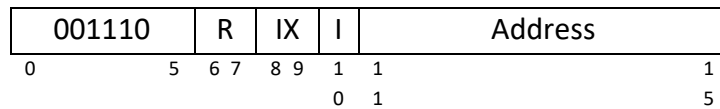
Instruction: JCC cc, x, address[, I]
 Octal-Opcode: 12
 Binary-Opcode: 001100
 Function: Jump if Condition Code

4.3.4 (13) JMA

001101	R	IX	I	Address
0 5	6 7	8 9	1 1	1
			0 1	5

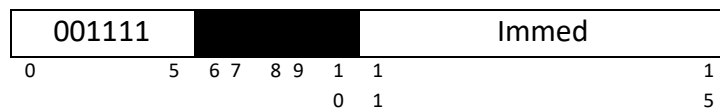
Instruction: JMA x, address[, I]
 Octal-Opcode: 13
 Binary-Opcode: 001101
 Function: Unconditional Jump to Address

4.3.5 (14) JSR



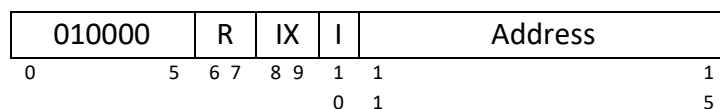
Instruction: JSR x, address[, I]
 Octal-Opcode: 14
 Binary-Opcode: 001110
 Function: Jump and Save Return Address

4.3.6 (15) RFS



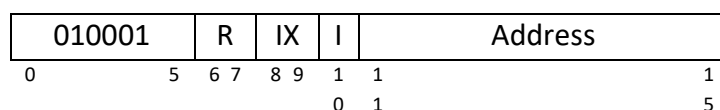
Instruction: RFS immed
 Octal-Opcode: 15
 Binary-Opcode: 001111
 Function: Return from Subroutine with Return Code as Immediate Portion (optional) Stored in the Instruction's Address Field

4.3.7 (16) SOB



Instruction: SOB r, x, address[, I]
 Octal-Opcode: 16
 Binary-Opcode: 010000
 Function: Subtract One and Branch

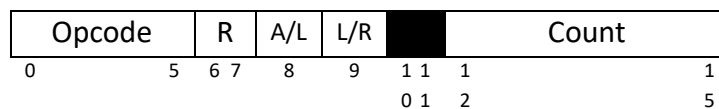
4.3.8 (17) JGE



Instruction: JGE r, x, address[, I]
 Octal-Opcode: 17
 Binary-Opcode: 010001
 Function: Jump Greater than or Equal to

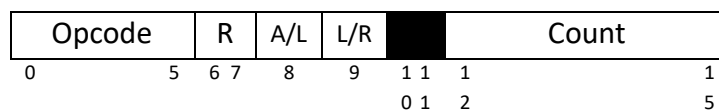
4.4 Shift/Rotate Instructions

The instructions to manipulate a datum in a register. The binary instruction code format of Shift and Rotate Instructions is as follows:



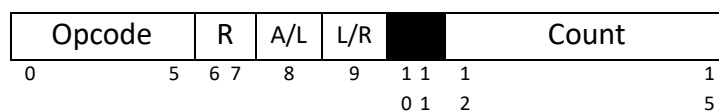
Opcode: 6 bits Specifies the instruction
R: 2 bits Specifies the General-Purpose Register
A/L: 2 bits Arithmetic Shift (A/L = 0); Logical Shift (A/L = 1)
L/R: 2 bits Logical Rotate (L/R = 1)
Count: 4 bits Specifies the Count for Operation

4.4.1 (31) SRC



Instruction: SRC r, count, L/R, A/L
 Octal-Opcode: 31
 Binary-Opcode: 011111
 Function: Shift Register by Count

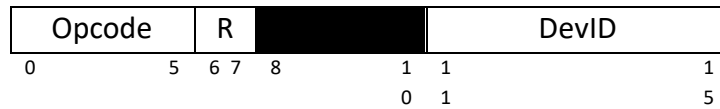
4.4.2 (32) RRC



Instruction: RRC r, count, L/R, A/L
 Octal-Opcode: 32
 Binary-Opcode: 100000
 Function: Rotate Register by Count

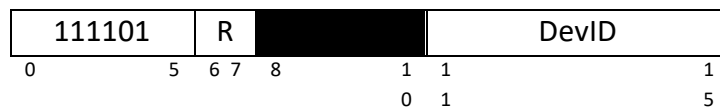
4.5 I/O Instructions

The instructions to communicate with the peripherals attached to the computer system. The binary instruction code format of I/O Instructions is as follows:



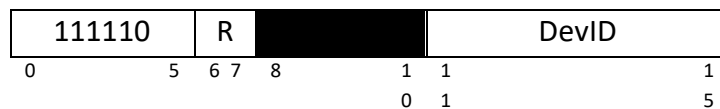
Opcode: 6 bits Specifies the instruction
R: 2 bits Specifies the General-Purpose Register
DevID: 5 bits Device ID:
 0 Console Keyboard
 1 Console Printer
 2 Card Reader
 3-31 Console Registers, Switches, etc.

4.5.1 (61) IN



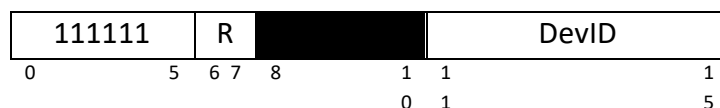
Instruction: IN r, devid
 Octal-Opcode: 61
 Binary-Opcode: 111101
 Function: Input Character to Register from Device

4.5.2 (62) OUT



Instruction: OUT r, devid
 Octal-Opcode: 62
 Binary-Opcode: 111110
 Function: Output Character to Device from Register

4.5.3 (63) CHK



Instruction: CHK r, devid
 Octal-Opcode: 63
 Binary-Opcode: 111111

Function: Check Device Status to Register

4.6 Other Instructions

4.6.1 (00) HALT



Instruction: HALT
Octal-Opcode: 00
Binary-Opcode: 000000
Function: Stop the machine