CISC Simulator User Guide

Rev 2.0

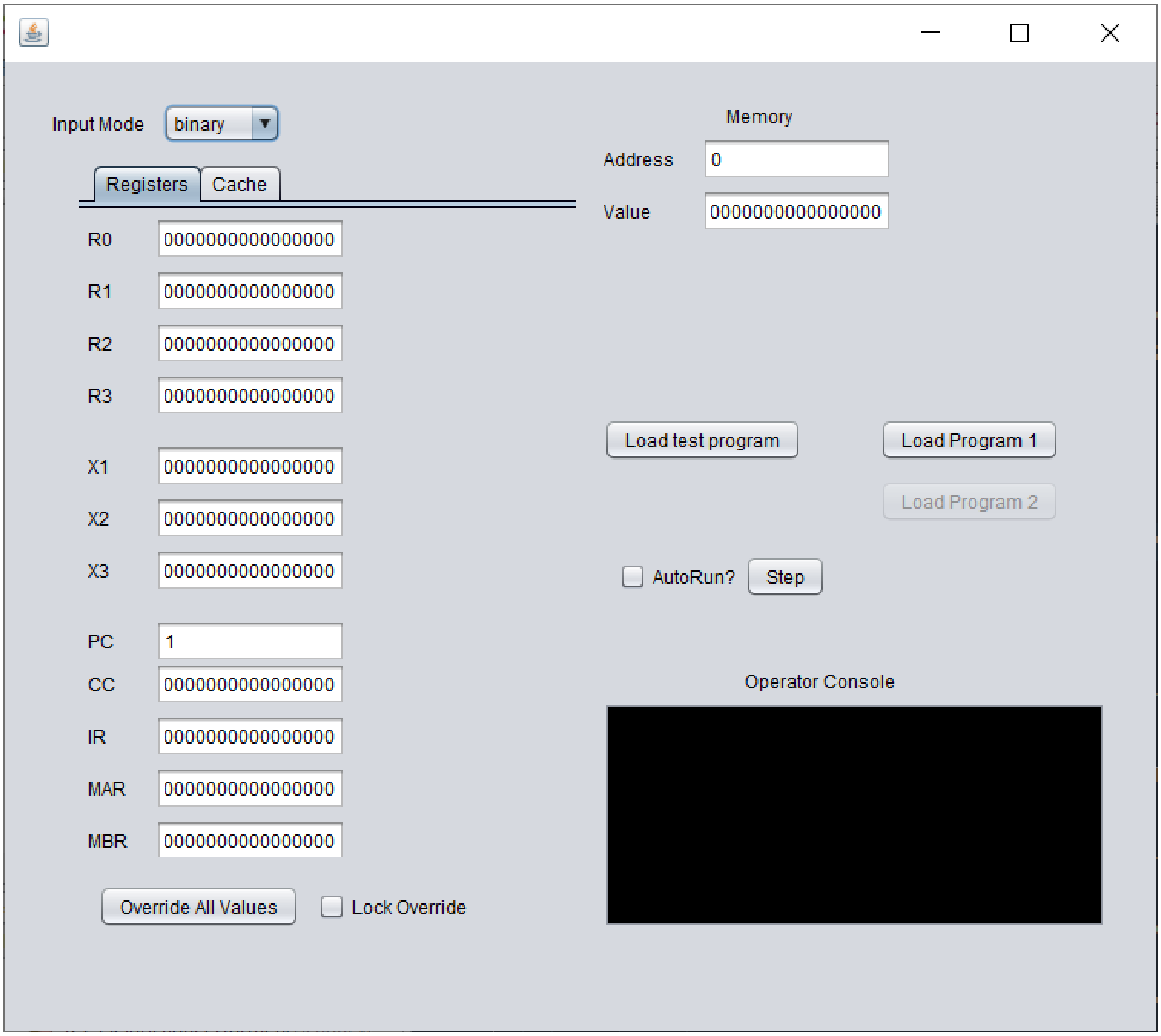
Group 2
Jonathan Pritchett
Ziwei Li
Haorong Xiao
Siyuan Zhang

Table Of Contents

Intro

This document provides a basic guide for a user of the Simulator. As additional features are added and improved in the simulator, this document will be updated as well.

UI Layout



Input Mode: binary

Address: 0

Value: 0000000000000000

R0: 0000000000000000

R1: 0000000000000000

R2: 0000000000000000

R3: 0000000000000000

X1: 0000000000000000

X2: 0000000000000000

X3: 0000000000000000

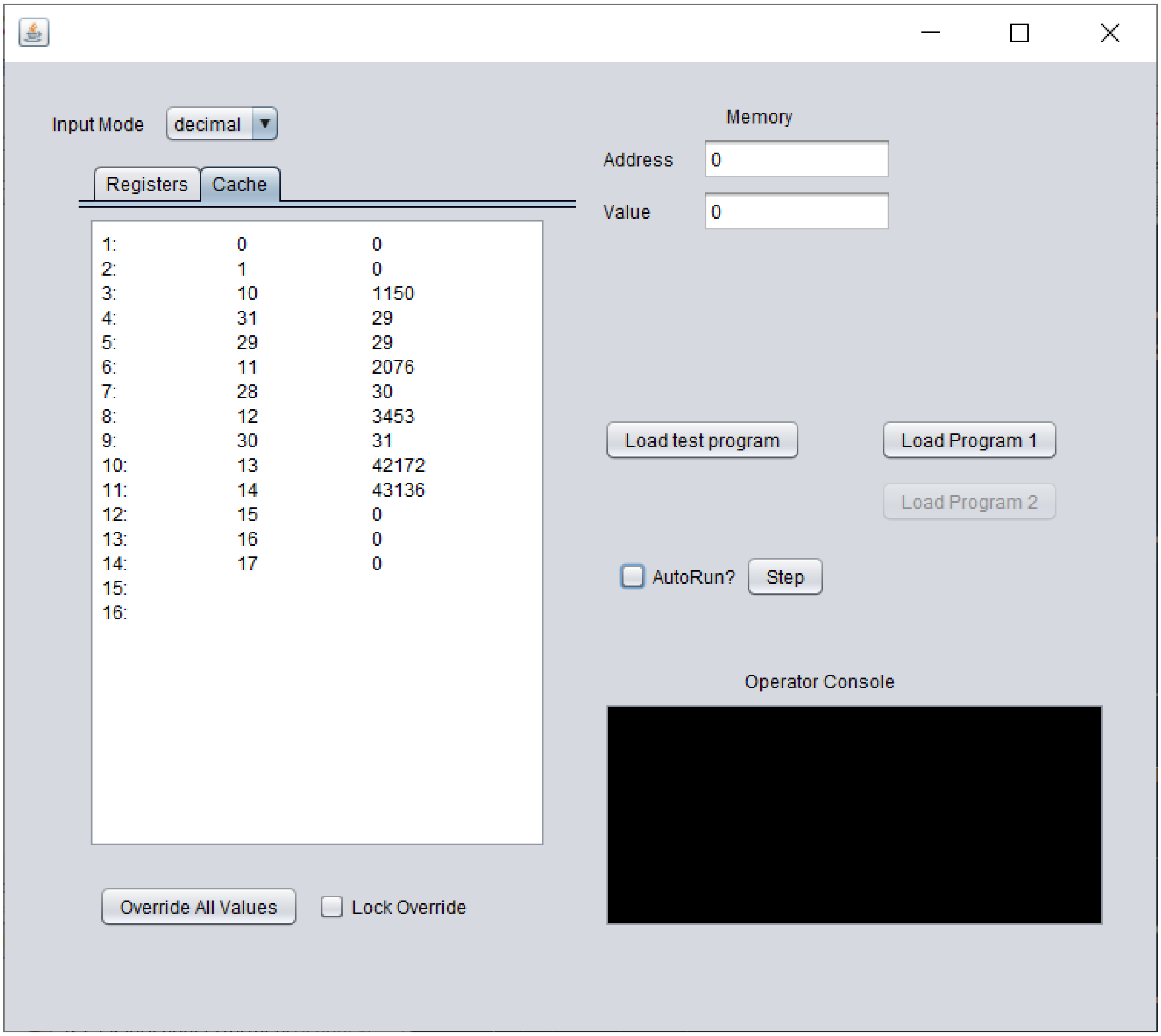
PC: 1

CC: 0000000000000000

IR: 0000000000000000

MAR: 0000000000000000

MBR: 0000000000000000



Input Mode: decimal

Address: 0

Value: 0

☐ AutoRun?

☐ Lock Override

1:	0	0
2:	1	0
3:	10	1150
4:	31	29
5:	29	29
6:	11	2076
7:	28	30
8:	12	3453
9:	30	31
10:	13	42172
11:	14	43136
12:	15	0
13:	16	0
14:	17	0
15:		
16:		

Overriding values

All of the displayed register and memory values may be freely edited by the user if the Lock override button is unchecked. Pressing the “Override All Values” buttons will force whatever values are currently in each of the fields into the appropriate register or memory address. Values entered into fields should be 16bit binary numbers.

*IMPORTANT NOTE: Currently, these fields are unformatted, and will accept any string as input. In the future, these will be restricted to only accept binary or decimal values, but this has not been implemented yet.

Changing Input Mode

The Input Mode dropdown allows the user to select between entering (and viewing) values in Binary or Decimal. Note that this does not affect the operator console (see below).

Operator Console

The operator console allows the user to provide input and view output for the program running in the simulator. Any text output from the program will be displayed here and any input the program requires should be typed here as well.

Memory

All 2048 memory addresses may be accessed by using the memory fields. Entering a 16-bit binary address into the “Address” field and pressing enter will populate the “Value” field with the current 16-bit data stored at that address. Manually entering a 16-binary number into the “Value” field and pressing the “Override” button will force that number into the current address indicated by the “Address” field.

Clicking on the “Cache” tab will bring up a view of the memory cache. The first column indicates the cache-lines position, the second indicates the cached memory address, and the third indicates the data stored in the cache/memory.

Registers

Each of the available registers for the simulator are displayed along the left side of the panel. Each one has a label indicating which register it is, and a text field indicating its current value. Entering a 16-bit binary number into any of these fields and pressing the “Override” button will force that number into the register.

Advancing the simulation

The “Step” button allows the user to run through the simulation one cycle at a time. If there are non-zero values in the PC or Instruction registers, clicking the “Step” button will cause the simulator to execute current instructions, advance the PC register, and fetch the next instruction. Checking the “Auto-run” box will cause the simulation to run continuously until there are no more instructions to execute. Currently, Auto-run is set to advance at a rate of 1 cycle per second (this will be user-configurable in the future).

Loading the Programs

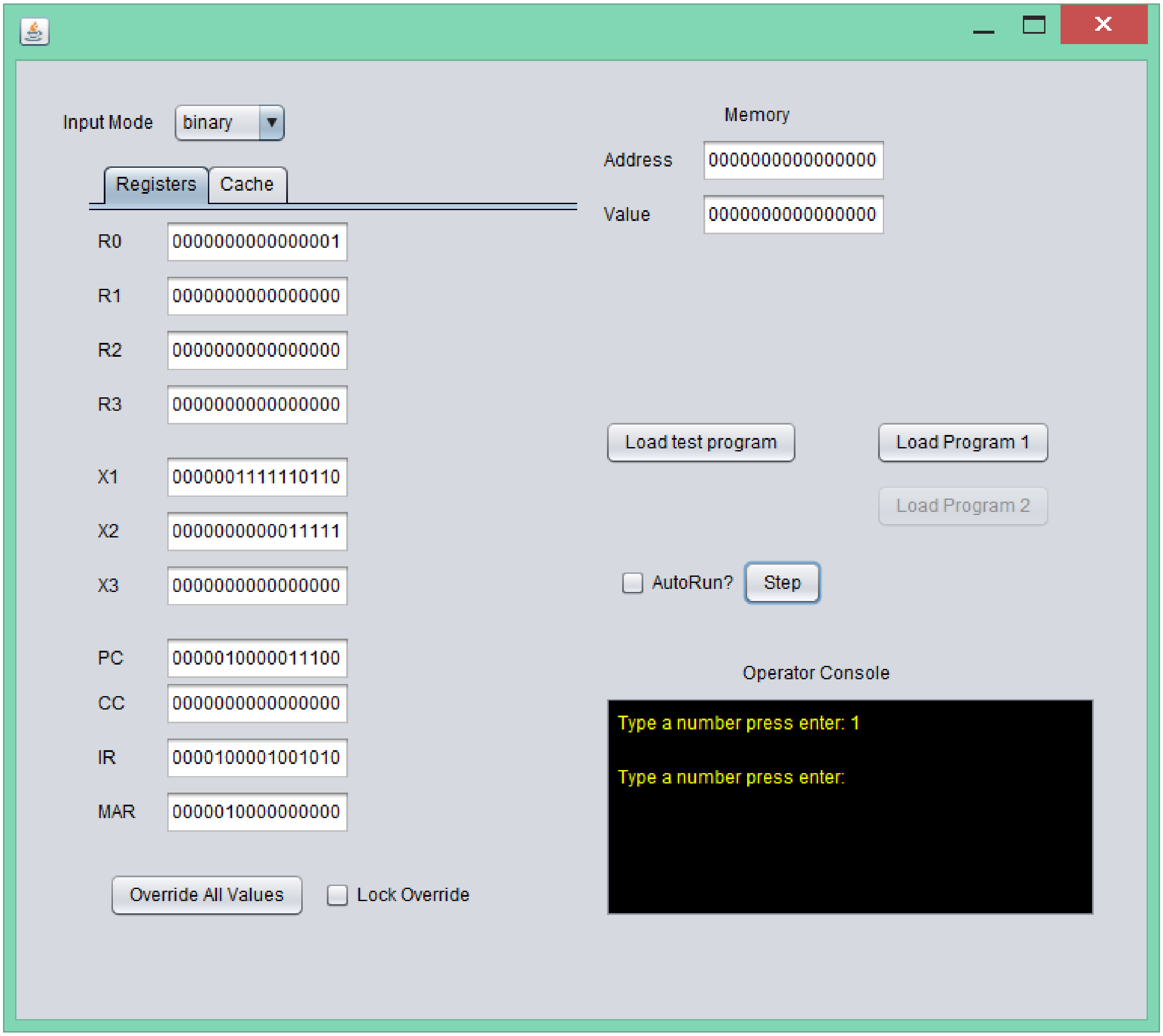
Pressing the “Load Test Program” will load a hard-coded program into the simulator’s memory. The “Step” and “Auto-run” buttons may then be used to run this test program. Likewise, clicking “Load Program 1” will load Program 1 into memory. Program 2 is currently unimplemented, and its button disabled.

Program 1 test result

This program takes 21 integer numbers from the user. Then, it will find the nearest number of those 20 numbers to the 21st number.

First of all, press the ‘Load Program 1’ button. The simulator will write the program into the memory.

To input numbers, user has to type the number and press Enter on the keyboard. Then, press the ‘Step’ twice to let the simulator run the instruction IN and store the number in the memory.



Input Mode: binary

Memory
Address: 0000000000000000
Value: 0000000000000000

R0: 0000000000000001
R1: 0000000000000000
R2: 0000000000000000
R3: 0000000000000000

X1: 0000001111110110
X2: 0000000000011111
X3: 0000000000000000

PC: 0000010000011100
CC: 0000000000000000
IR: 0000100001001010
MAR: 0000010000000000

□ AutoRun?
□ Lock Override

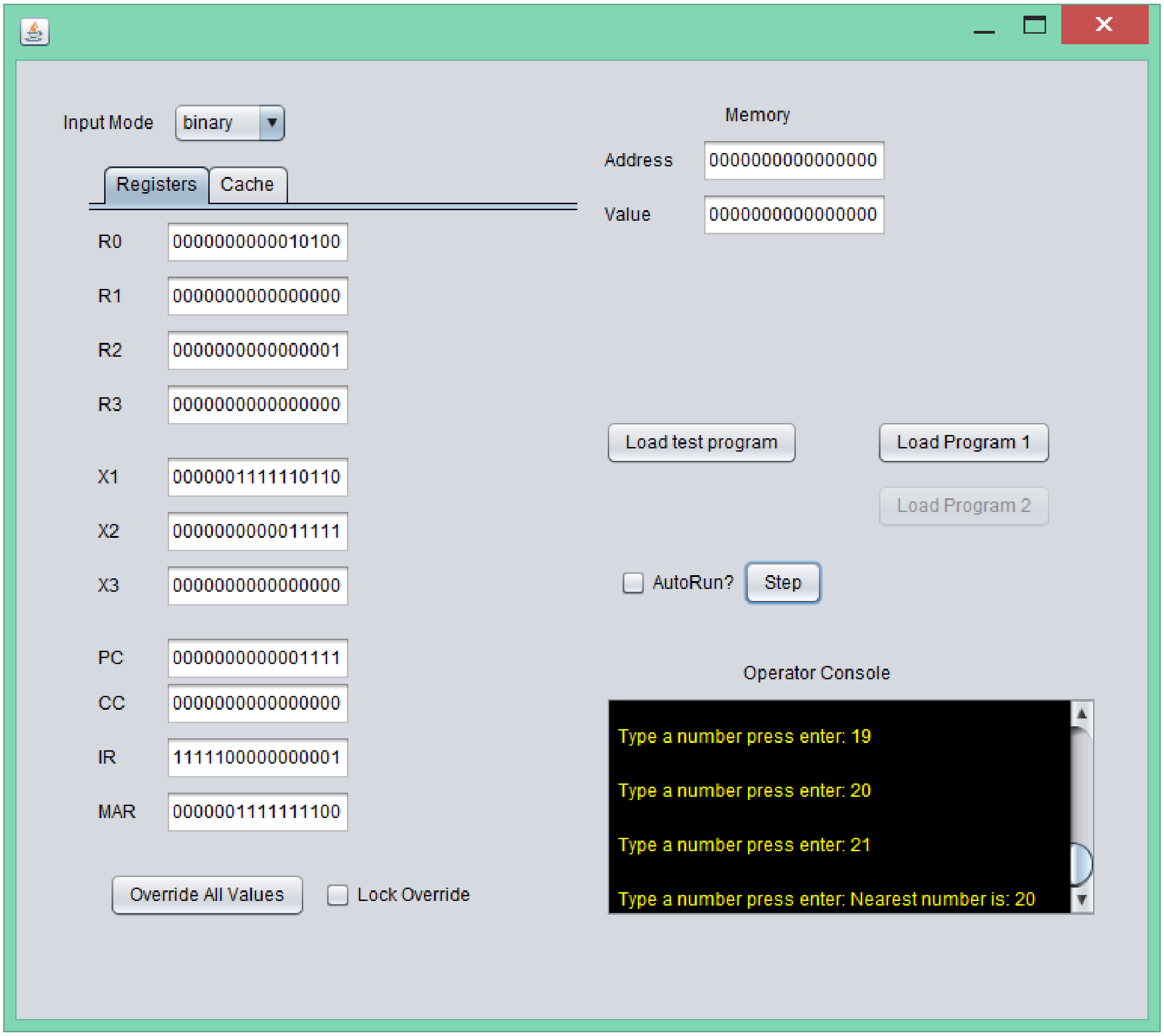
Operator Console

```
Type a number press enter: 1

Type a number press enter:
```

As we can see in the screenshot, the input number has been stored in R0 and it has been stored in memory. After having input 21 numbers, user can press ‘AutoRun’ checkbox to let simulator run automatically or press ‘Step’ button to check every step of this program.

This time, we input 1 to 20 in the simulator and the 21st number is 21. We do this is just in case of overflow and underflow situations. Here is the result of this program:



Input Mode: binary

R0	0000000000010100
R1	0000000000000000
R2	0000000000000001
R3	0000000000000000
X1	0000001111110110
X2	0000000000011111
X3	0000000000000000
PC	0000000000001111
CC	0000000000000000
IR	1111100000000001
MAR	0000001111111100

Memory

Address	0000000000000000
Value	0000000000000000

```
Type a number press enter: 19
Type a number press enter: 20
Type a number press enter: 21
Type a number press enter: Nearest number is: 20
```

Below is the code with instructions explained. For clarification, we use the ‘setMemory’ function called by the instance ‘memory’ from class ‘Memory’ to write every instruction into the memory with address and instruction specified. The former 40 functions is used for getting the input numbers from user.

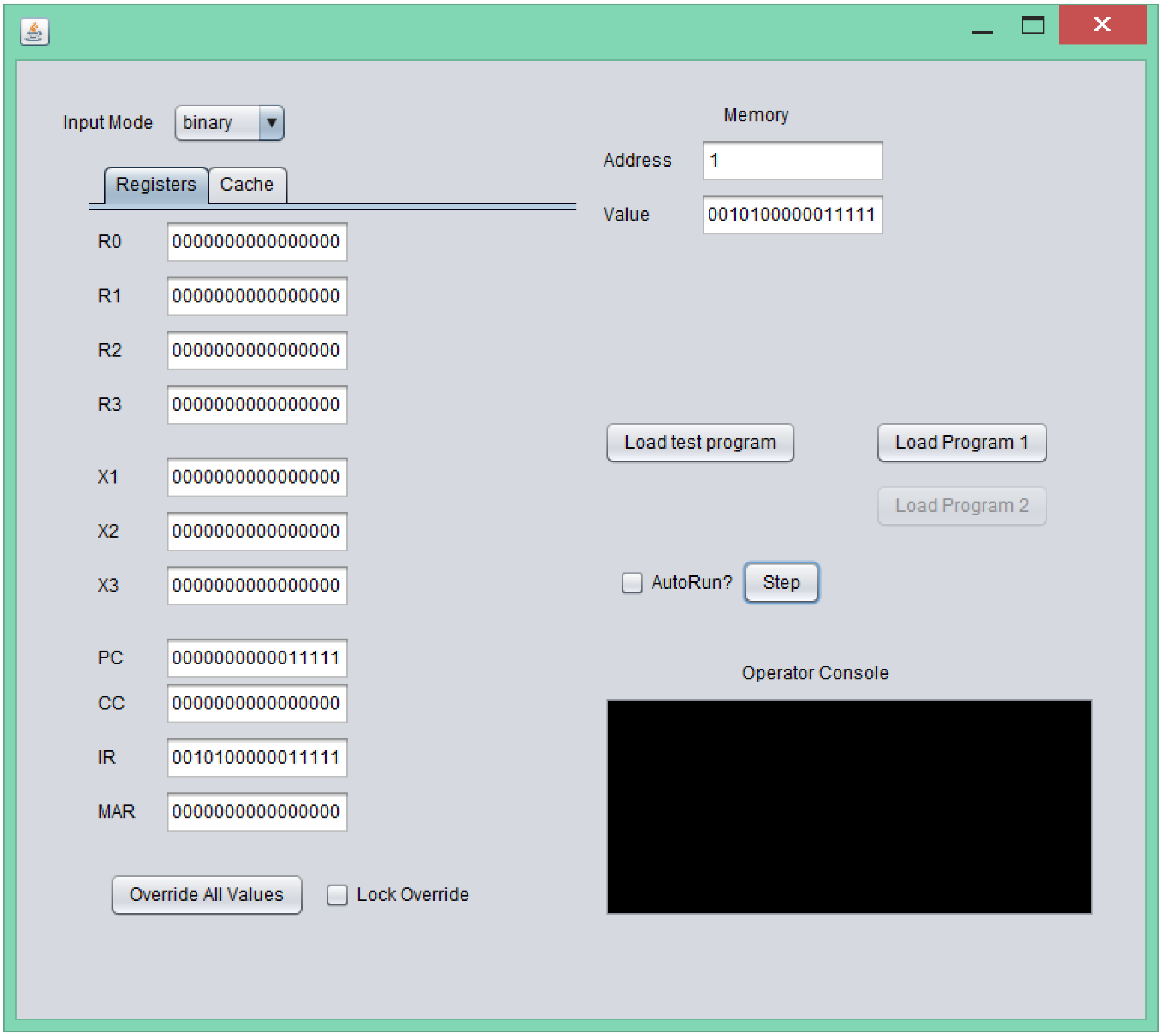
```
    memory.setMemory(1021, "0000001111101100");
    memory.setMemory(1023, "0111111111111111");
    memory.setMemory(1050, "1111010000000000");
    memory.setMemory(1051, "0000100001001010");
    memory.setMemory(1052, "1111010000000000");
    memory.setMemory(1053, "0000100001001011");
    memory.setMemory(1054, "1111010000000000");
    memory.setMemory(1055, "0000100001001100");
    memory.setMemory(1056, "1111010000000000");
    memory.setMemory(1057, "0000100001001101");
    memory.setMemory(1058, "1111010000000000");
    memory.setMemory(1059, "0000100001001110");
    memory.setMemory(1060, "1111010000000000");
    memory.setMemory(1061, "0000100001001111");
    memory.setMemory(1062, "1111010000000000");
    memory.setMemory(1063, "0000100001010000");
    memory.setMemory(1064, "1111010000000000");
    memory.setMemory(1065, "0000100001010001");
    memory.setMemory(1066, "1111010000000000");
    memory.setMemory(1067, "0000100001010010");
    memory.setMemory(1068, "1111010000000000");
    memory.setMemory(1069, "0000100001010011");
    memory.setMemory(1070, "1111010000000000");
    memory.setMemory(1071, "0000100001010100");
    memory.setMemory(1072, "1111010000000000");
    memory.setMemory(1073, "0000100001010101");
    memory.setMemory(1074, "1111010000000000");
    memory.setMemory(1075, "0000100001010110");
    memory.setMemory(1076, "1111010000000000");
    memory.setMemory(1077, "0000100001010111");
    memory.setMemory(1078, "1111010000000000");
    memory.setMemory(1079, "0000100001011000");
    memory.setMemory(1080, "1111010000000000");
    memory.setMemory(1081, "0000100001011001");
    memory.setMemory(1082, "1111010000000000");
    memory.setMemory(1083, "0000100001011010");
    memory.setMemory(1084, "1111010000000000");
    memory.setMemory(1085, "0000100001011011");
    memory.setMemory(1086, "1111010000000000");
    memory.setMemory(1087, "0000100001011100");
    memory.setMemory(1088, "1111010000000000");
    memory.setMemory(1089, "0000100001011101");
    memory.setMemory(1090, "1111010000000000");
    memory.setMemory(1091, "0000100001001000");
    memory.setMemory(1092, "0011010000001010");//JMA 10
    memory.setMemory(10, "0001101100010101");//AIR R3 21
    memory.setMemory(11, "0000101101000110");//STR R3 1020
    memory.setMemory(12, "0000011101000110");//LDR R3 1020
    memory.setMemory(13, "0100001100001111");//SOB R3 15
    memory.setMemory(14, "1111100000000001");//output the closest number
    memory.setMemory(15, "0000011001001000");//LDR R2 0 0 1022
    memory.setMemory(16, "0001001101000111");//AMR R3 1021
```

```
	memory.setMemory(17, "0000101101000100");//STR R3 1018
	memory.setMemory(18, "0001011001100100");//SMR R2 0 1 1018
	memory.setMemory(19, "0101001010000000");//MLT R2 R2
	memory.setMemory(20, "0000101101000010");//STR R3 1016
	memory.setMemory(21, "0000011001000010");//LDR R2 1016
	memory.setMemory(22, "0001011101001001");//SMR R3 0 0 1023
	memory.setMemory(23, "0100011110000010");//JGE R3 0 0 34
	memory.setMemory(24, "0000101001001001");//STR R2 0 0 1023
	memory.setMemory(25, "0000010001100100");//LDR R0 0 1 1018
	memory.setMemory(26, "0000100001000011");//STR R0 0 0 1017
	memory.setMemory(27, "0000011101000111");//LDR R3 1021
	memory.setMemory(28, "0001101100000010");//AIR R3 2
	memory.setMemory(29, "0000101101000111");//STR R3 1021
	memory.setMemory(30, "0000011101000110");//LDR R3 1020
	memory.setMemory(31, "0001111100000001");//SIR R3 1
	memory.setMemory(32, "0000101101000110");//STR R3 1020
	memory.setMemory(33, "0011010000001100");//JMA 12
	memory.setMemory(34, "0000011101000111");//LDR R3 1021
	memory.setMemory(35, "0001101100000010");//AIR R3 2
	memory.setMemory(36, "0000101101000111");//STR R3 1021
	memory.setMemory(37, "0000011101000110");//LDR R3 1020
	memory.setMemory(38, "0001111100000001");//SIR R3 1
	memory.setMemory(39, "0000101101000110");//STR R3 1020
	memory.setMemory(40, "0011010000001100");//JMA 12
```

Demonstration of other instructions

Some instructions we did not use in above program. Therefore, we have some screenshots here to show that they work.

Jump if Zero:
The register R0 equals to 0. Then, Program counter is set to ‘11111’.



Input Mode binary

Memory
Address 1
Value 0010100000011111

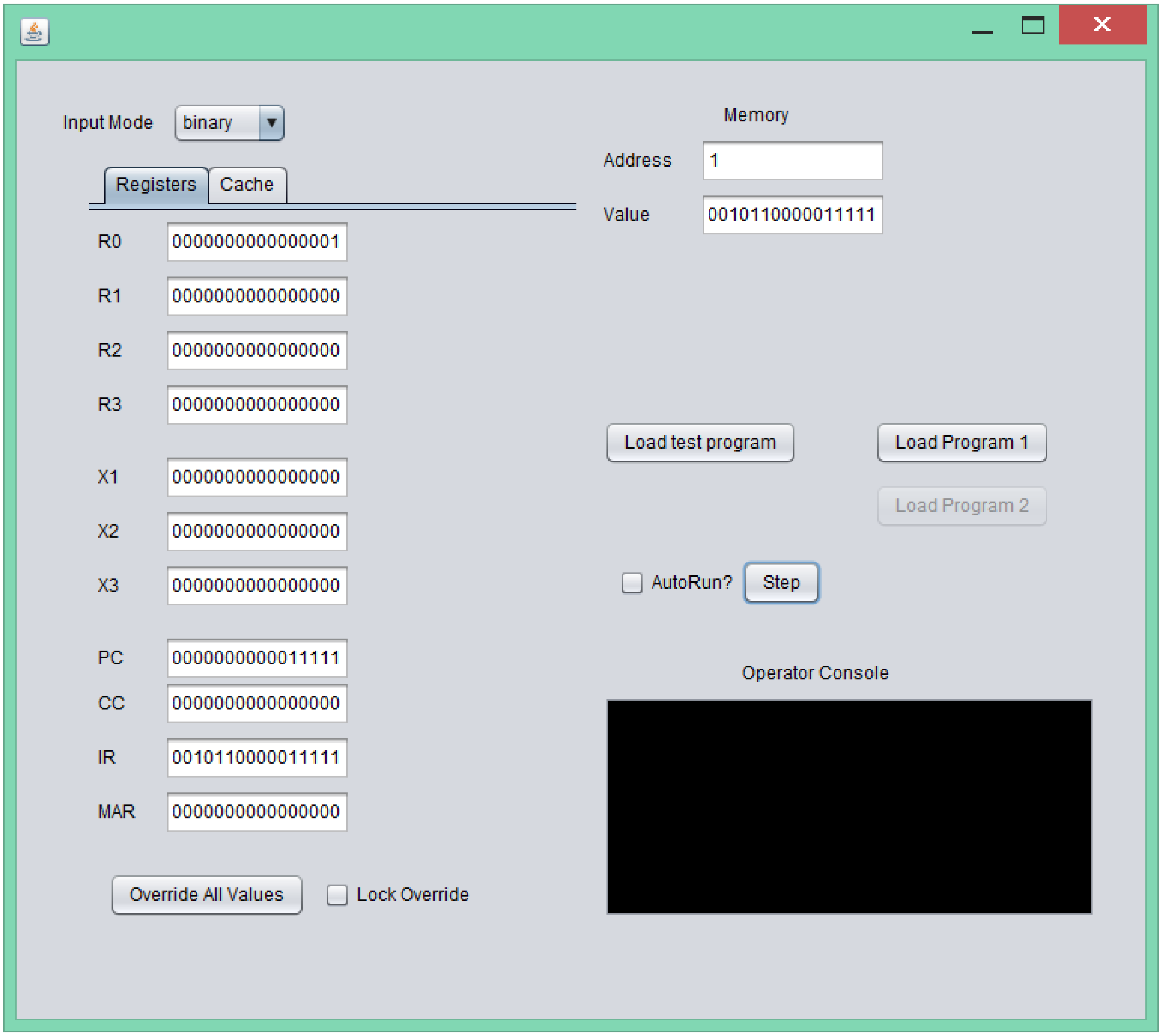
R0 0000000000000000
R1 0000000000000000
R2 0000000000000000
R3 0000000000000000

X1 0000000000000000
X2 0000000000000000
X3 0000000000000000

PC 0000000000011111
CC 0000000000000000
IR 0010100000011111
MAR 0000000000000000

□ AutoRun?
□ Lock Override

Jump if not equal:
The register R0 is set to 1. Then, Program counter is set to ‘11111’.

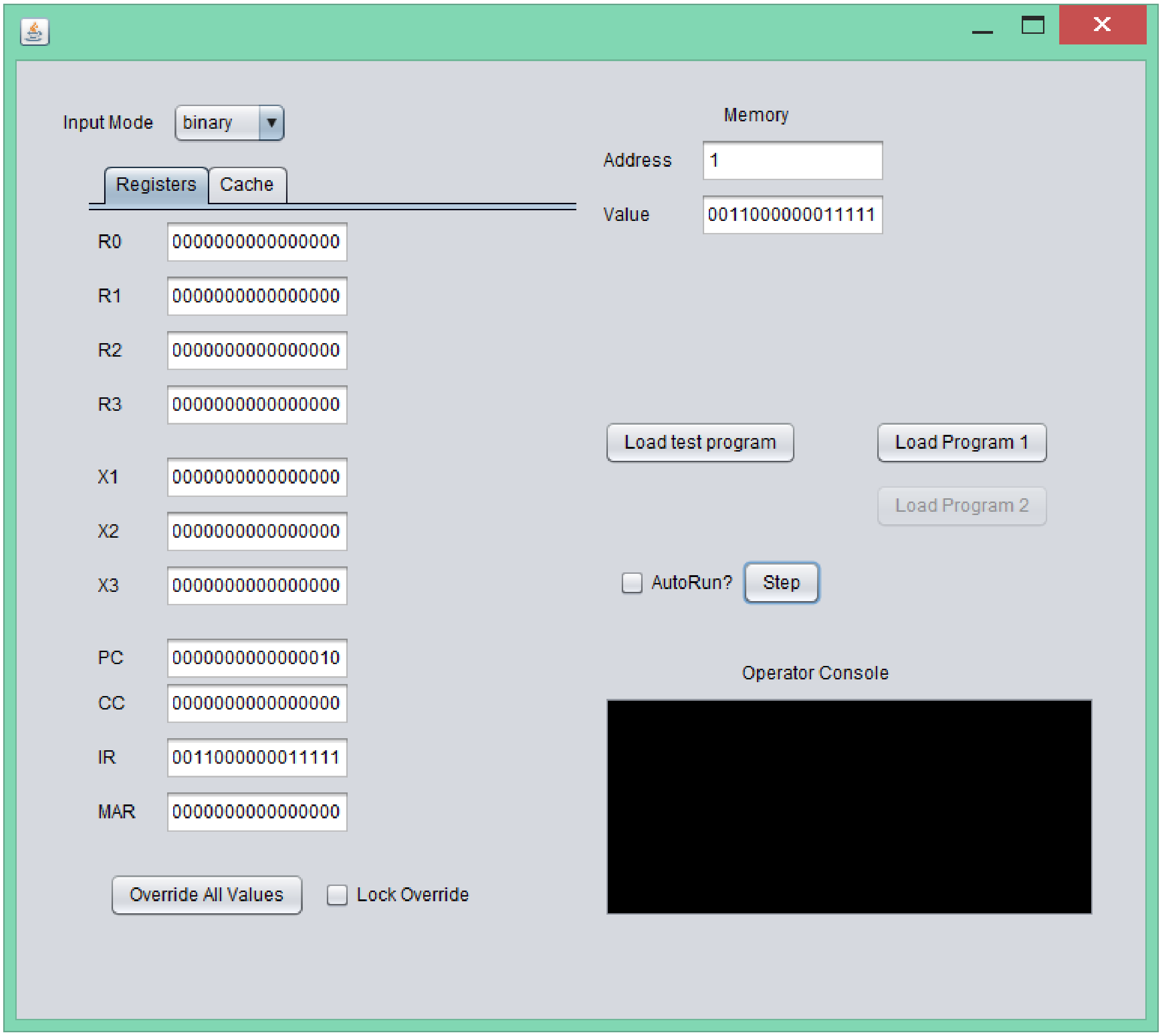


Input Mode: binary

Memory
Address: 1
Value: 0010110000011111

R0	0000000000000001
R1	0000000000000000
R2	0000000000000000
R3	0000000000000000
X1	0000000000000000
X2	0000000000000000
X3	0000000000000000
PC	0000000000011111
CC	0000000000000000
IR	0010110000011111
MAR	0000000000000000

Jump if condition code:
We set condition code here 0. Therefore, program counter adds 1 to itself.

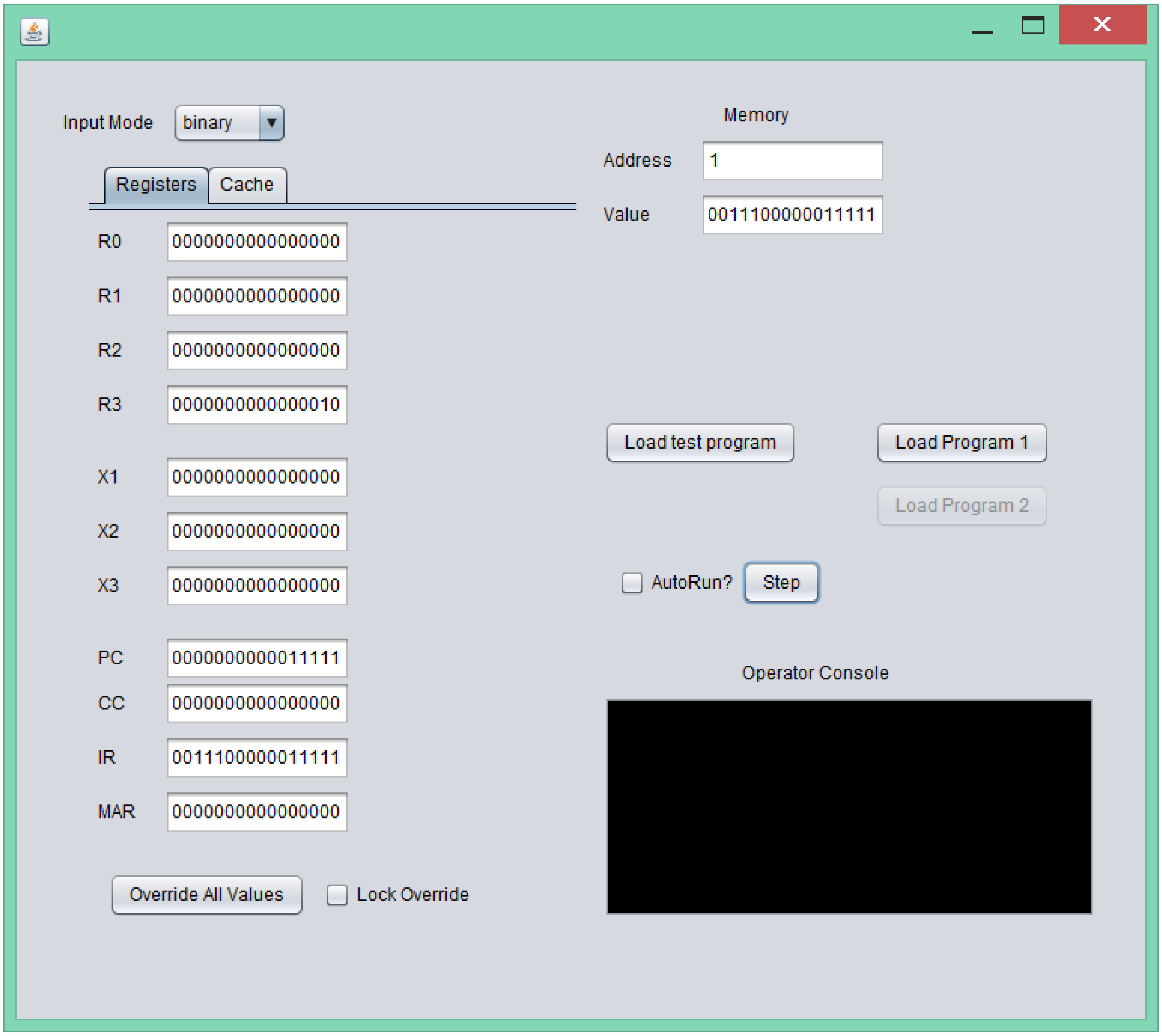


Input Mode: binary

Address: 1
Value: 0011000000011111

R0: 0000000000000000
R1: 0000000000000000
R2: 0000000000000000
R3: 0000000000000000
X1: 0000000000000000
X2: 0000000000000000
X3: 0000000000000000
PC: 0000000000000010
CC: 0000000000000000
IR: 0011000000011111
MAR: 0000000000000000

Jump and Save Return Address:
R3 = program counter + 1, program counter = effective address.



Input Mode: binary

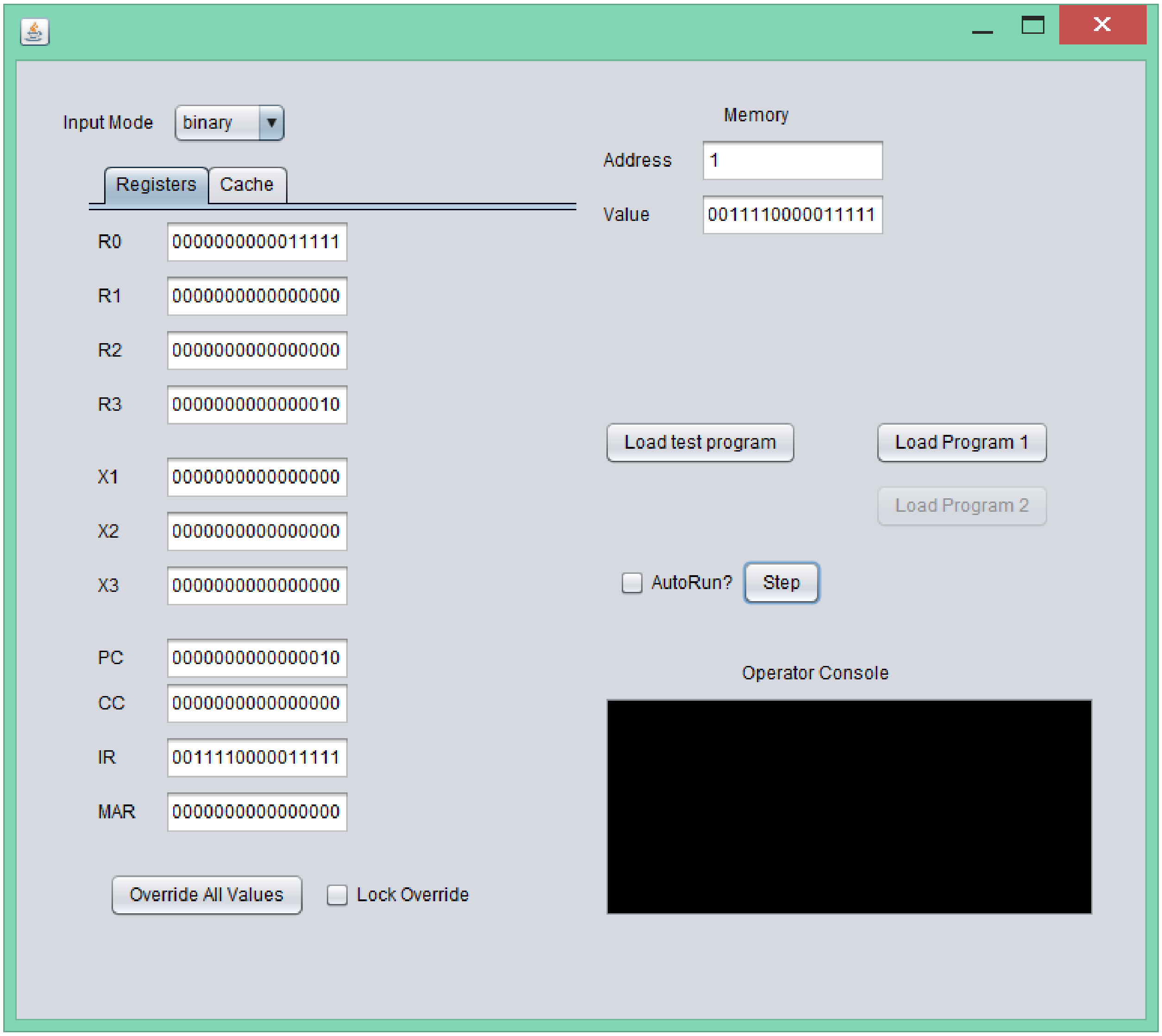
Memory
Address: 1
Value: 0011100000011111

R0: 0000000000000000
R1: 0000000000000000
R2: 0000000000000000
R3: 0000000000000010

X1: 0000000000000000
X2: 0000000000000000
X3: 0000000000000000

PC: 0000000000011111
CC: 0000000000000000
IR: 0011100000011111
MAR: 0000000000000000

Return From Subroutine with return code as Immed portion:
R0 = Immed, PC = c(R3)



Input Mode: binary

Memory
Address: 1
Value: 0011110000011111

R0: 0000000000011111
R1: 0000000000000000
R2: 0000000000000000
R3: 0000000000000010

X1: 0000000000000000
X2: 0000000000000000
X3: 0000000000000000

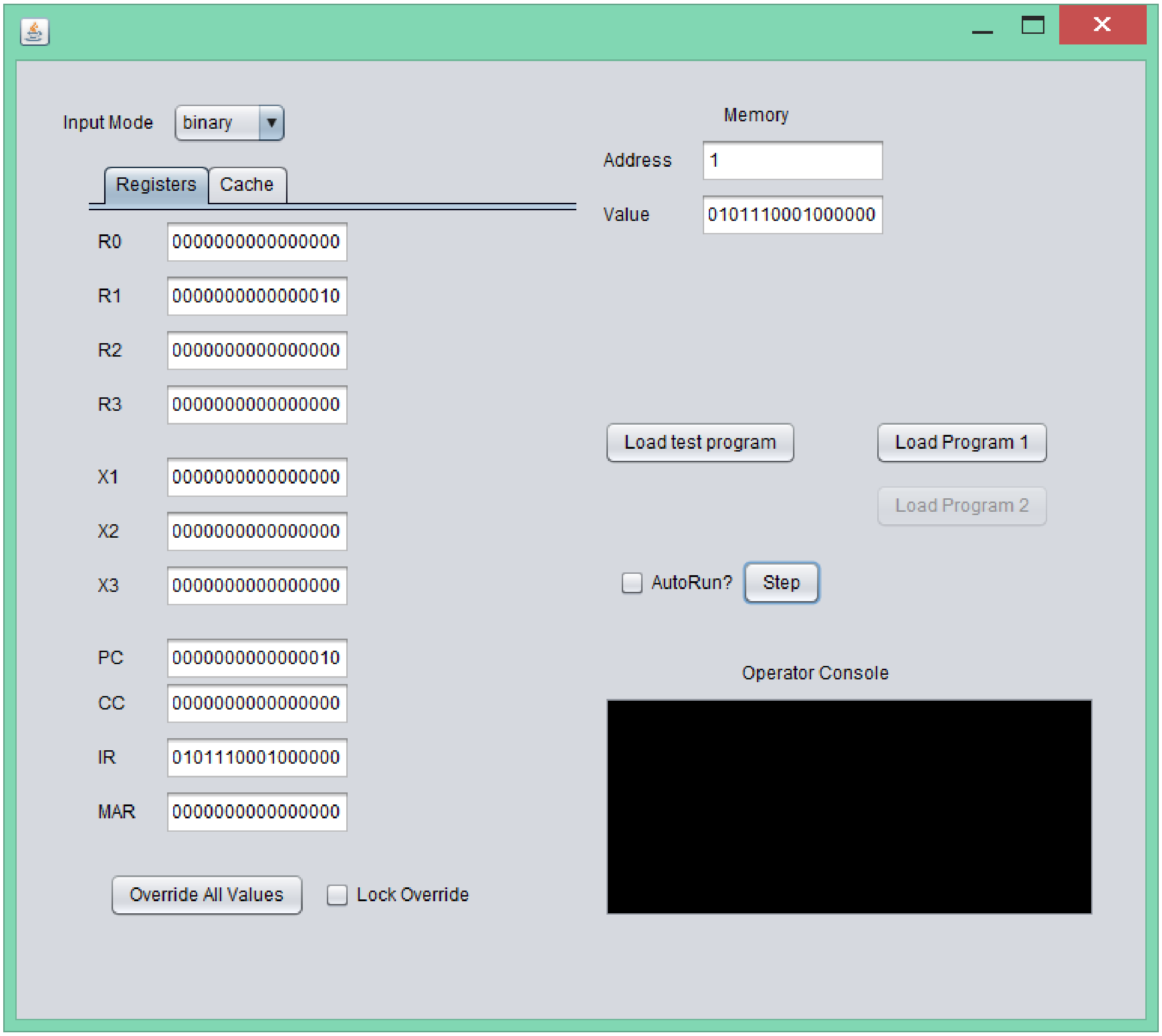
PC: 0000000000000010
CC: 0000000000000000
IR: 0011110000011111
MAR: 0000000000000000

□ AutoRun?

□ Lock Override

AND:

We set R0 = 01, R1 = 10. The result id stored in R0.



Input Mode: binary

Memory

Address: 1

Value: 0101110001000000

R0: 0000000000000000

R1: 0000000000000010

R2: 0000000000000000

R3: 0000000000000000

X1: 0000000000000000

X2: 0000000000000000

X3: 0000000000000000

PC: 0000000000000010

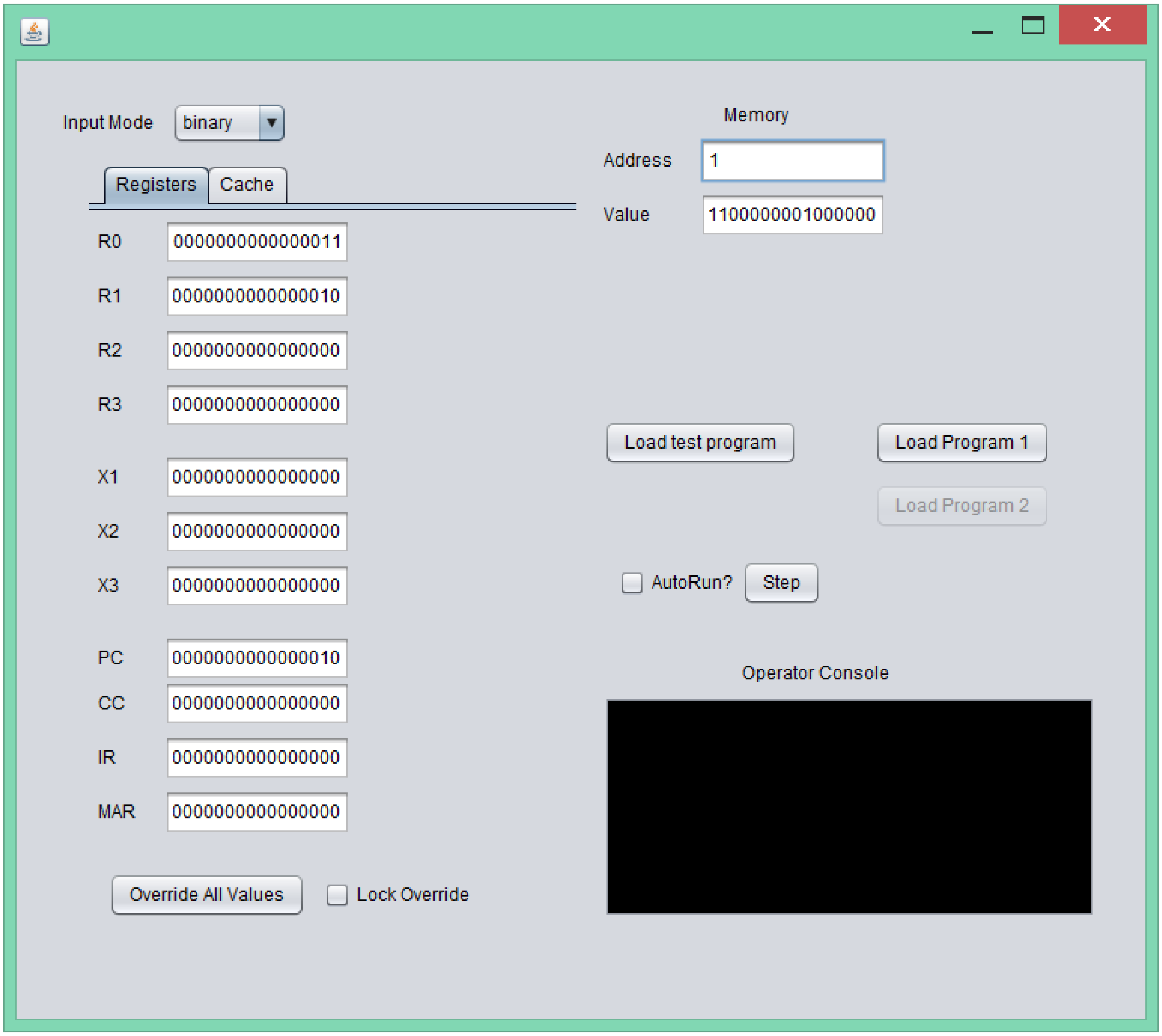
CC: 0000000000000000

IR: 0101110001000000

MAR: 0000000000000000

OR:

We set R0 = 01, R1 = 10. The result id stored in R0.



Input Mode: binary

Memory
Address: 1
Value: 1100000001000000

R0: 0000000000000011
R1: 0000000000000010
R2: 0000000000000000
R3: 0000000000000000

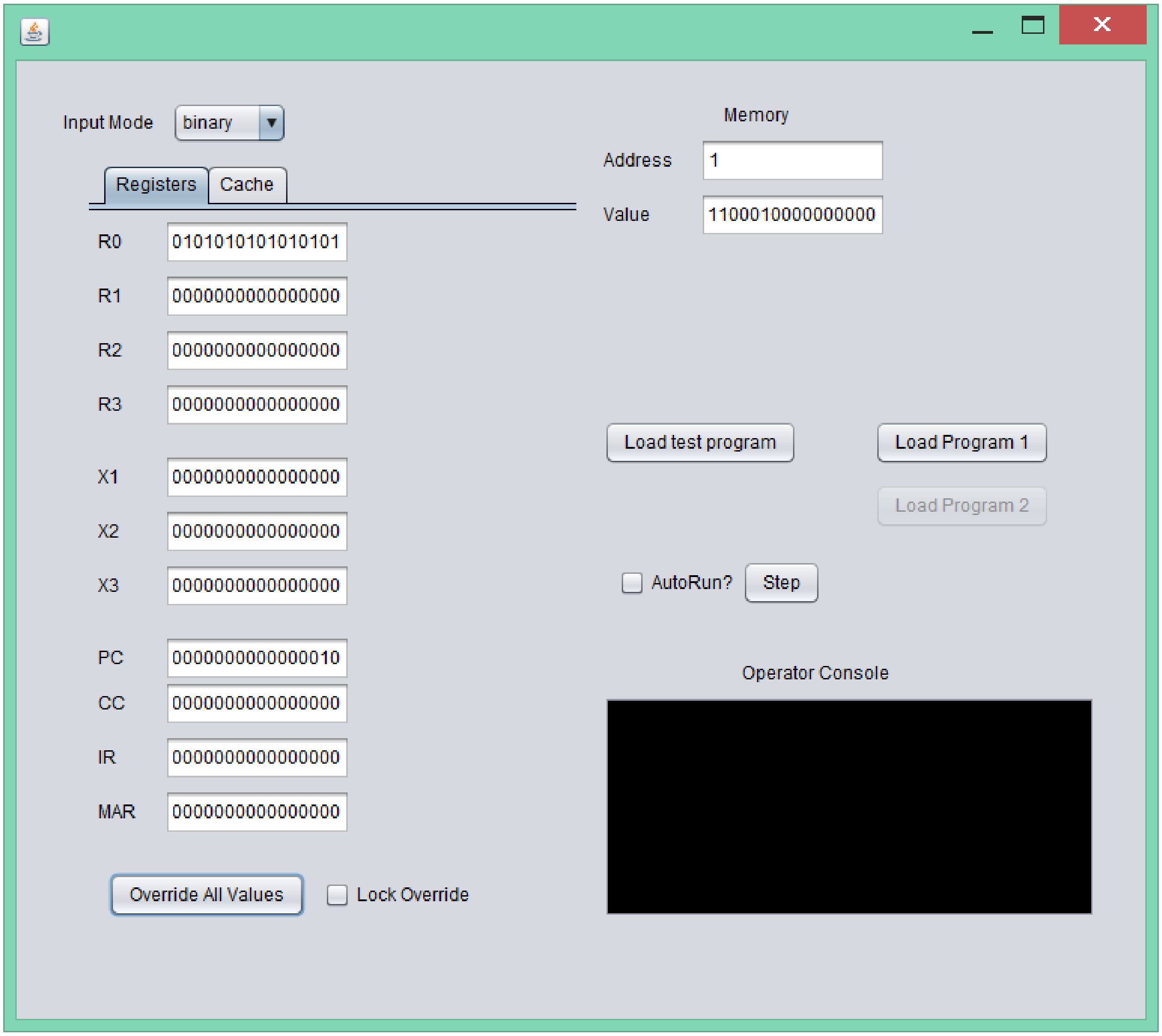
X1: 0000000000000000
X2: 0000000000000000
X3: 0000000000000000

PC: 0000000000000010
CC: 0000000000000000
IR: 0000000000000000
MAR: 0000000000000000

☐ AutoRun?

☐ Lock Override

NOT:
We R0 = 1010101010101010.



Input Mode: binary

Registers

R0: 0101010101010101
R1: 0000000000000000
R2: 0000000000000000
R3: 0000000000000000
X1: 0000000000000000
X2: 0000000000000000
X3: 0000000000000000
PC: 0000000000000010
CC: 0000000000000000
IR: 0000000000000000
MAR: 0000000000000000

Memory
Address: 1
Value: 1100010000000000

☐ AutoRun?

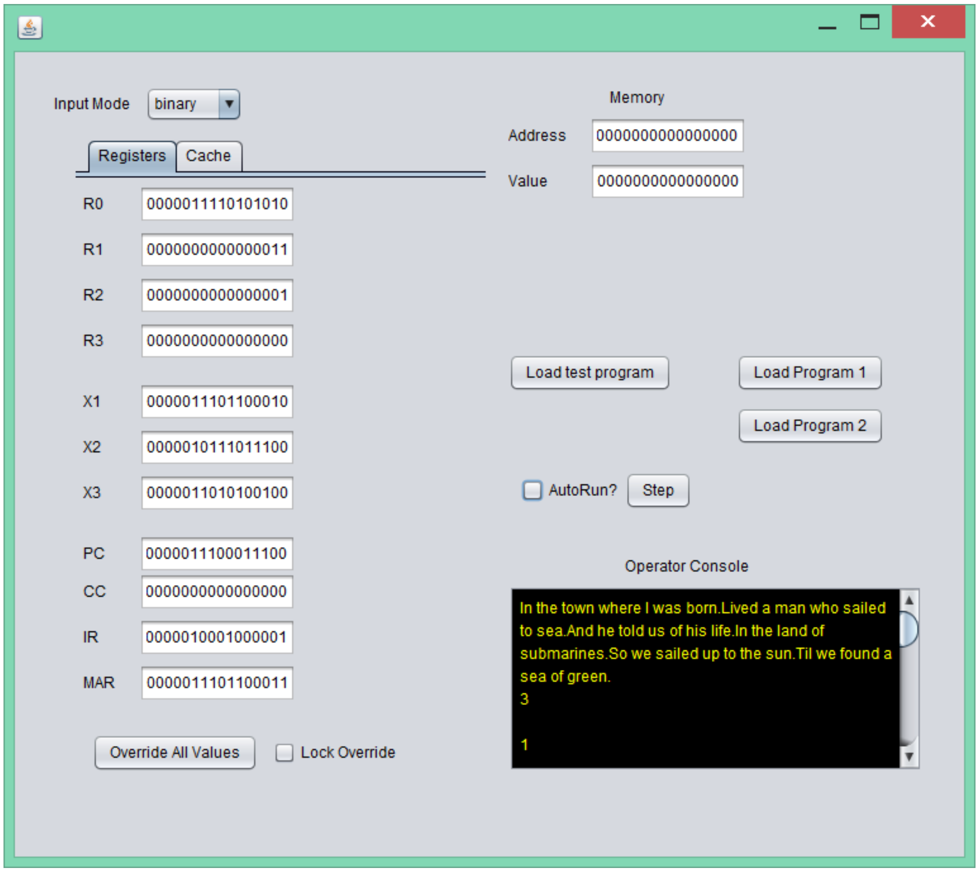
☐ Lock Override

Program 2 Test result

This program find the word required by the user in a paragraph with 6 sentences. When it finds the word, it will display the word number in the sentence and the sentence number.

First, click the ‘Load Program 2’ button and a window will show up and asks user to load the file containing the paragraph. To run the program, user can choose autorun or step by step by clicking the checkbox ‘AutoRun’ or ‘Step’ button.

Below is a screenshot of the result. The word we are going to find is ‘town’. And the input paragraph is ‘In the town where I was born.Lived a man who sailed to sea.And he told us of his life.In the land of submarines.So we sailed up to the sun.Til we found a sea of green.’ The word ‘town’ is in the first sentence and the it is the 3rd word in the sentence.



Input Mode binary

R0 0000011110101010
R1 0000000000000011
R2 0000000000000001
R3 0000000000000000
X1 0000011101100010
X2 0000010111011100
X3 0000011010100100
PC 0000011100011100
CC 0000000000000000
IR 0000010001000001
MAR 0000011101100011

Address 0000000000000000
Value 0000000000000000

□ AutoRun?

□ Lock Override

```
In the town where I was born.Lived a man who sailed
to sea.And he told us of his life.In the land of
submarines.So we sailed up to the sun.Til we found a
sea of green.
3

1
```

The R1 stores the word number and the R2 stores the line number.