CISC Simulator Design Document

Rev 1.0

Group 2
Jonathan Pritchett
Ziwei Li
Haorong Xiao
Siyuan Zhang

Table of Contents

Description

The CISC Simulator is a program that creates a simple simulator of a rudimentary computer. In its current iteration, this simulated computer consists of a set of registers, a simple memory, a small set of instructions, and a graphical user interface to serve as the computer front panel.

Data Representation

Data in the computer shall be stored in 16-bit words. To represent this, a class “Word” will be created which stores a 16-bit word as a 16 char array. This class will have accessor methods allowing for easy translation of this data into ints and strings and will allow for accessing of individual and subsets of bits.

Registers

In this first iteration of the CISC simulator, the following 11 16-bit registers are used.

R0...R3	4 General-purpose registers
PC	Program counter
IR	Instruction Register
MAR	Memory Address Register
MBR	Memory Buffer Register
X1...X3	3 Index Registers

Each register consists of a single Word object. All of the registers are collectively stored within a single class, “Registers”, which may be instantiated once and shared among the components of the simulator.

Memory

The Memory system for this version consists of a single class that contains an array of 2048 “Word” objects. The memory is instantiated once and shared throughout the simulator.

Instructions

Five instructions are currently implemented within the simulator.

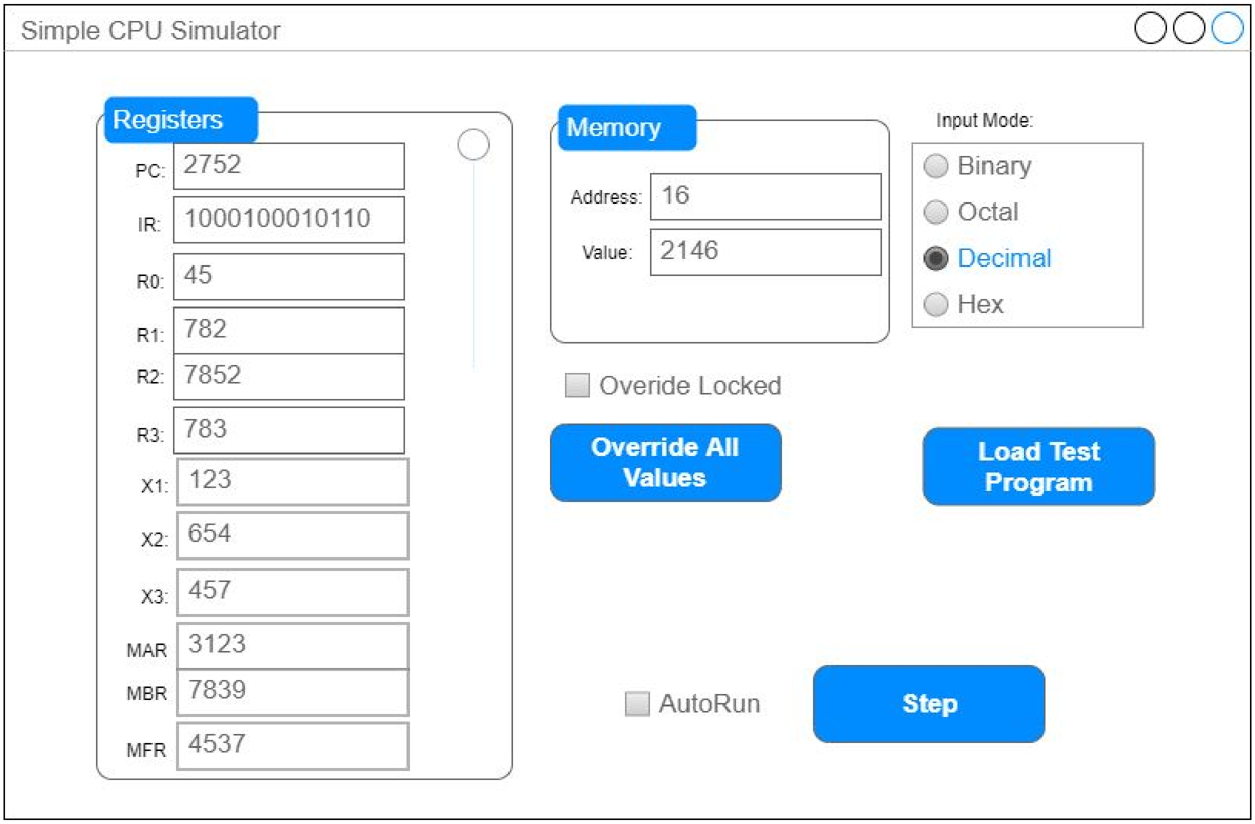
Instruction	OpCode	Description
LDR	01	Load register from memory
STR	02	Store register to memory
LDA	03	Load register with address
LDX	41	Load index register from memory
STX	42	Store index register to memory

Front Panel GUI

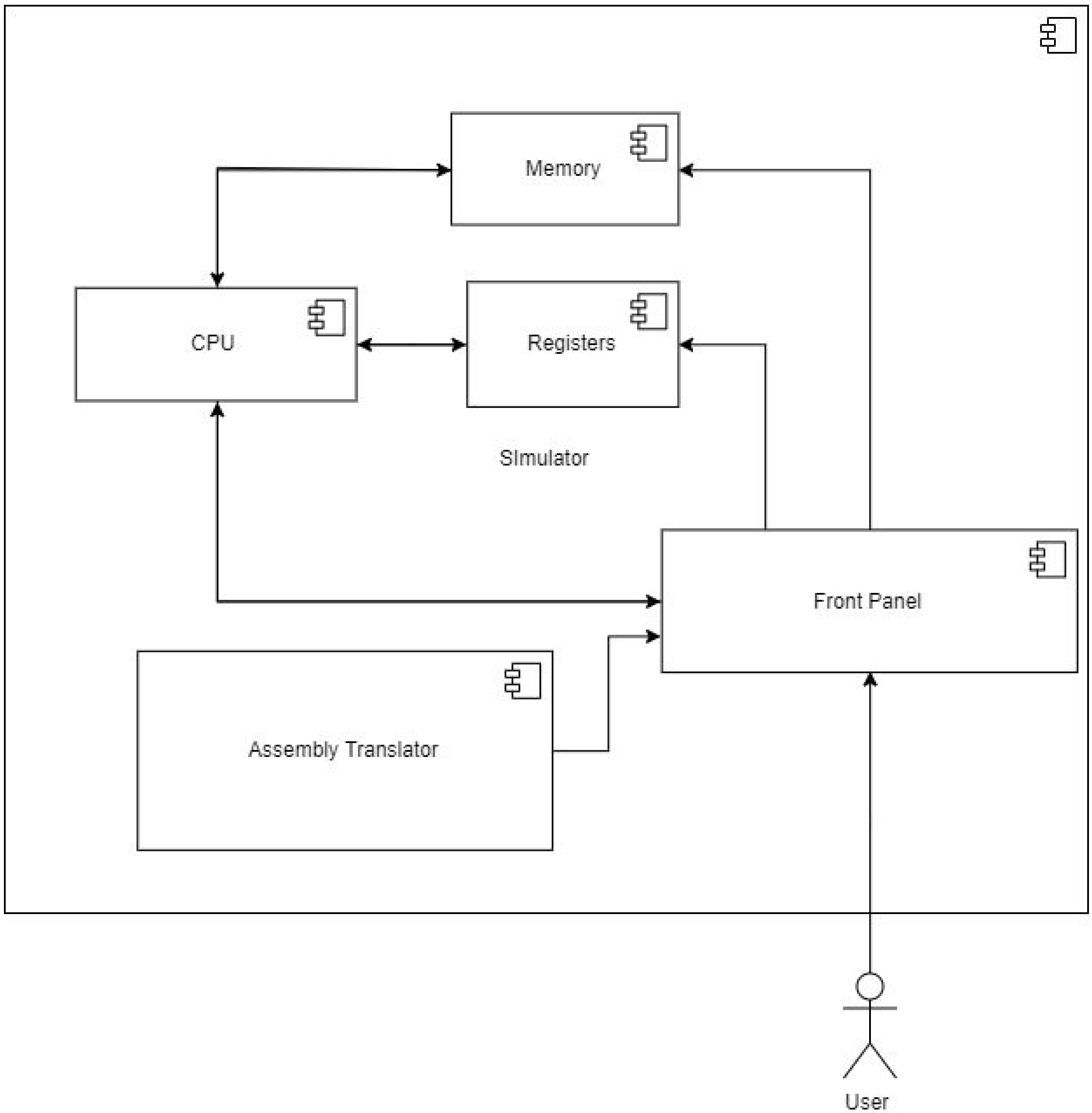
The front panel GUI provides the user with a way to view the contents of all of the registers and memory locations used by the simulator. It also allows the user to override any of these values manually and to run the simulator from its current state.

Figures

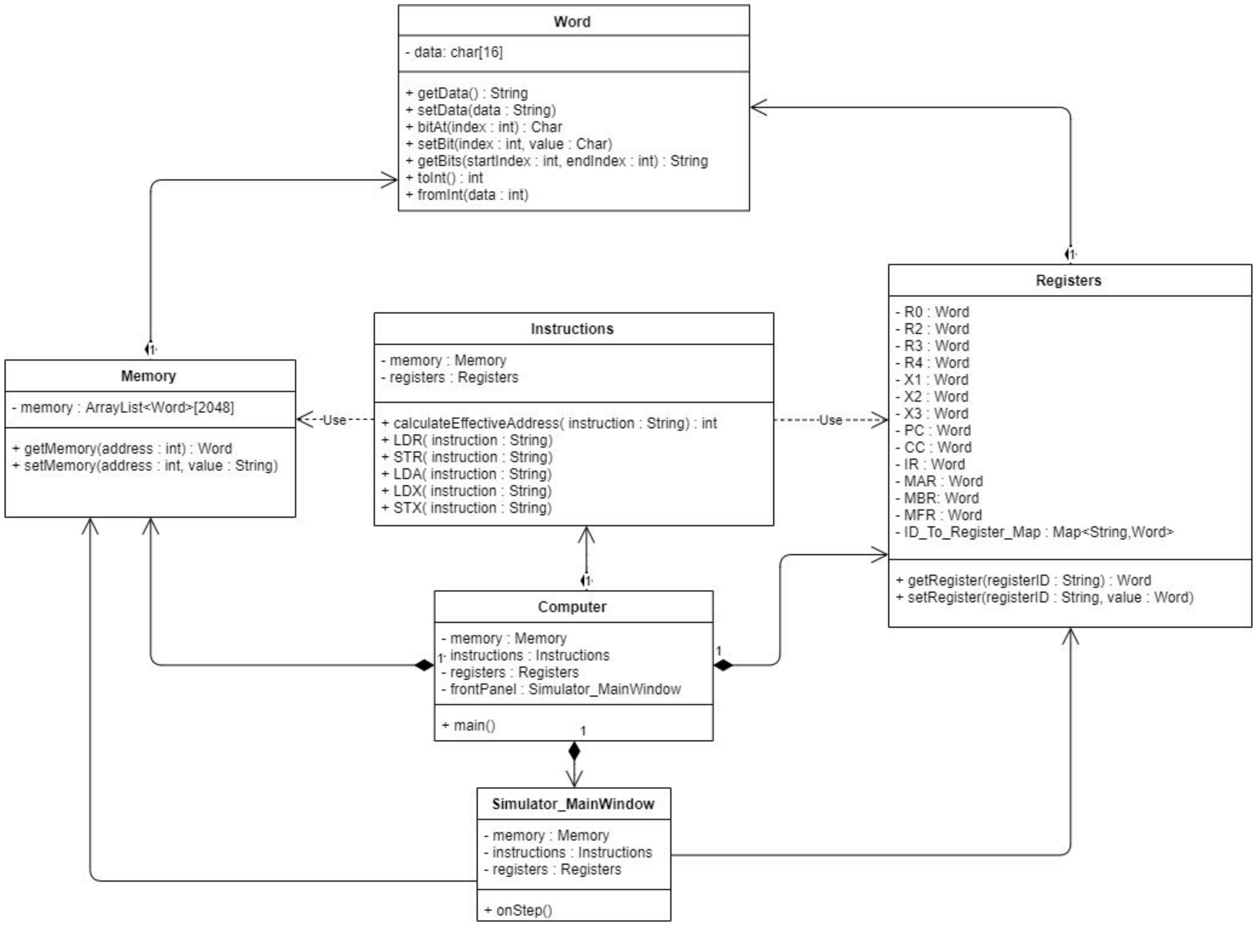
UI Mockup



Component Diagram



Class Diagram



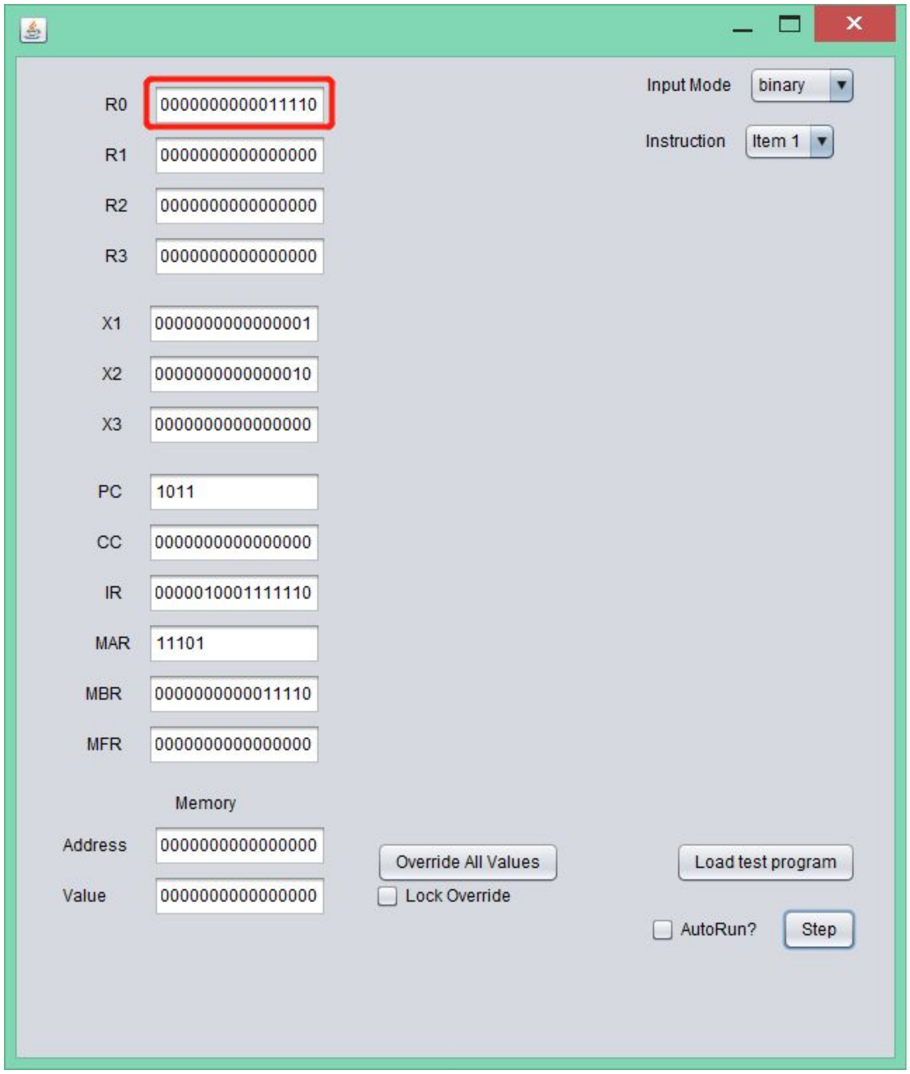
Test Program

We initialize the test program by pressing the “load test program” button in the lower right corner.
We set up 4 memory addresses and 2 index registers:

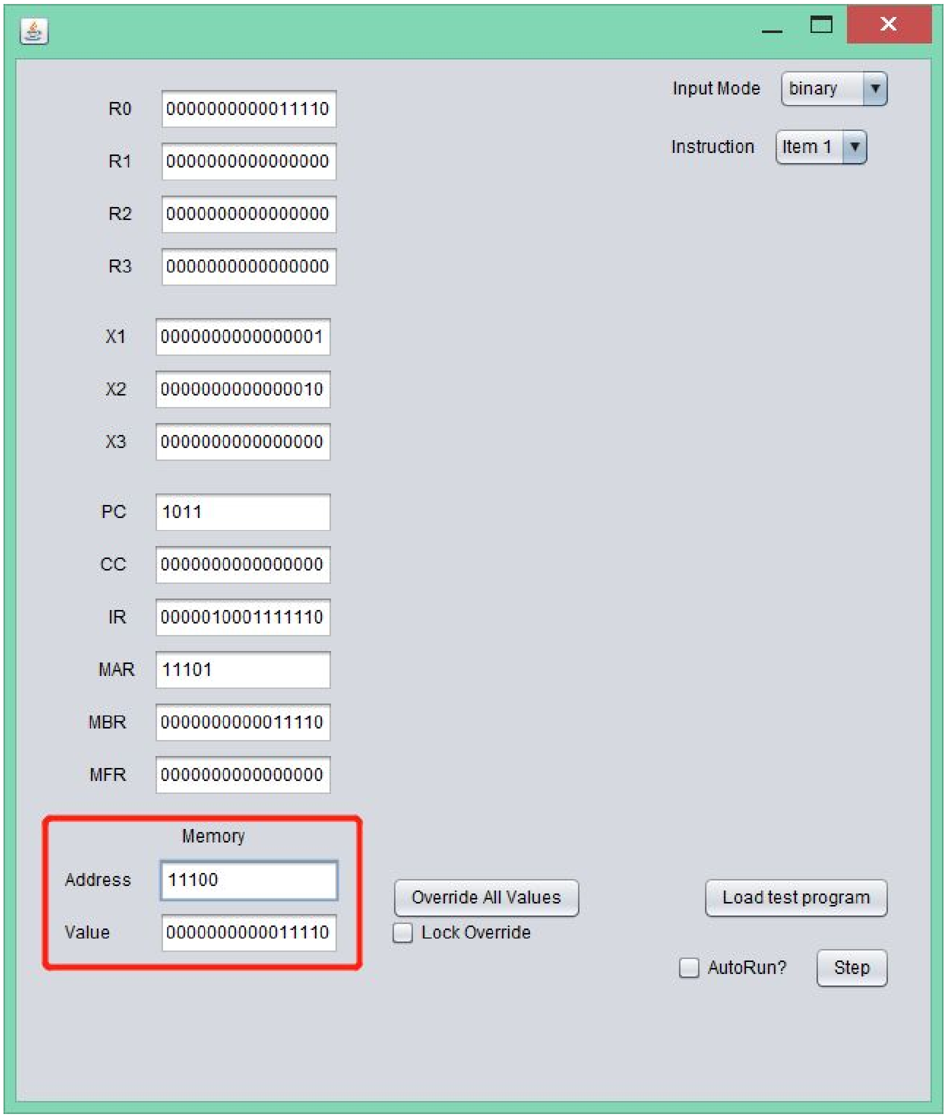
- memory address 31 stores integer value 29
- memory address 30 stores integer value 31
- memory address 29 stores integer value 30
- index register X1 stores integer value 1 and index register X2 stores integer value 2.

Then, we have stored 5 instructions in memory:

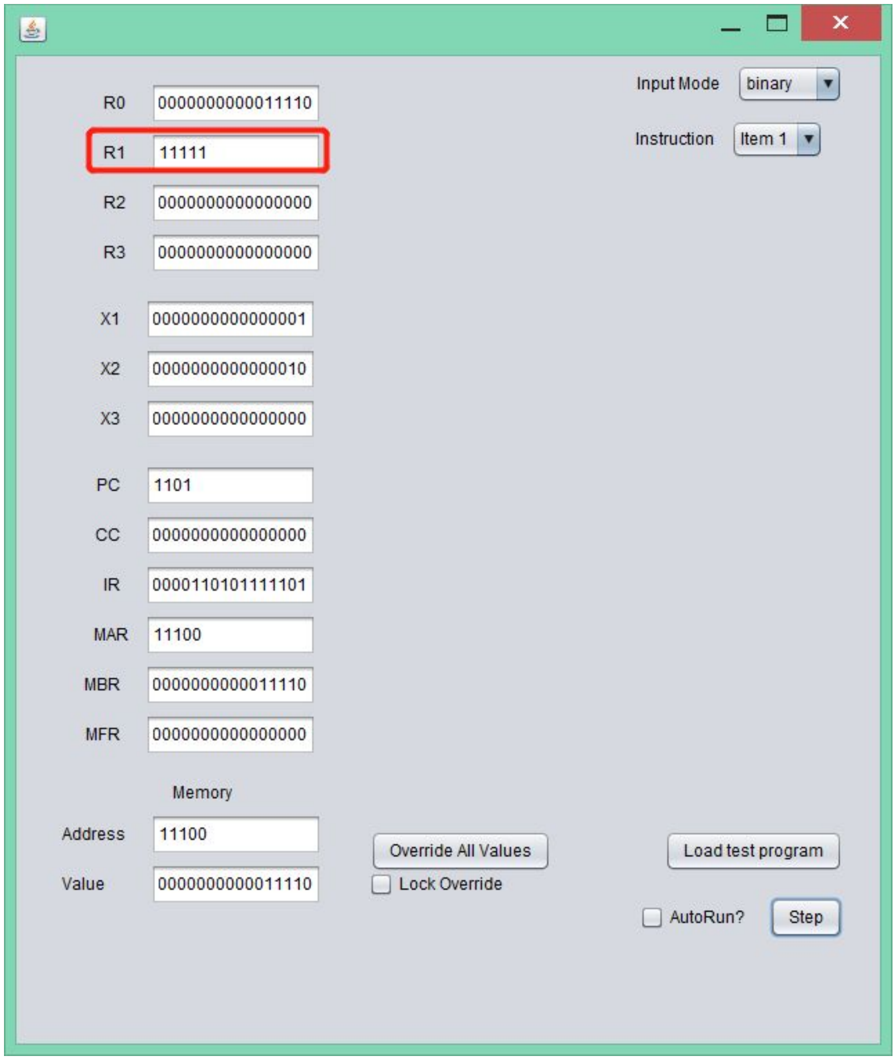
(1) Load register from memory: 000001 00 01 1 11110. (EA = memory[31] = 29)
This instruction loads effective address 29 by indexing and indirecting to register R0, which stores 30 representing by ‘11110’.’



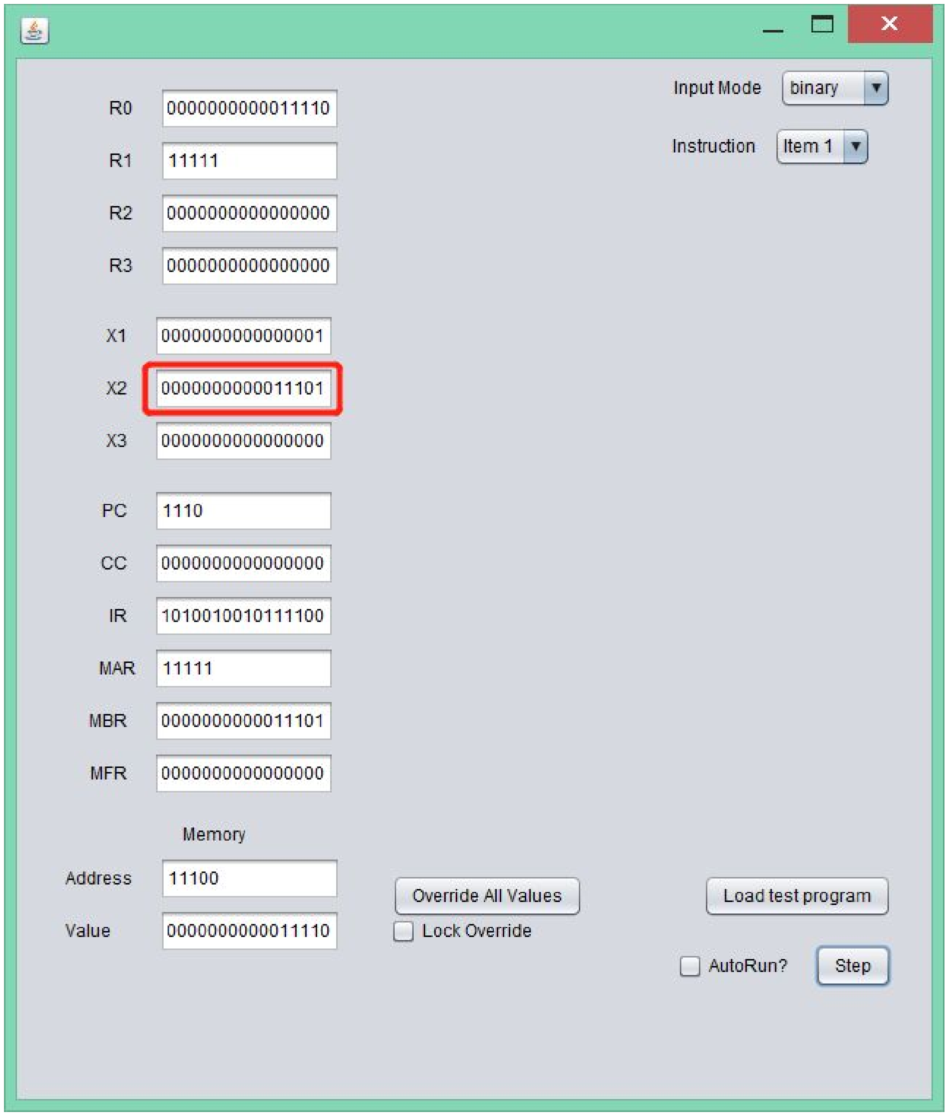
(2) Store register to memory: 000010 00 00 0 11100. (EA = 28)
This instruction stores the value of R0 to the effective address 28. Then, the content of memory address 28 become 30.



(3) Load register with address : 000011 01 01 1 11101. (EA = memory[30] = 31)
This instruction loads the content in effective address to register R1 by indexing and indirecting. The effective address here is 31. Therefore, 31 is stored in R1.

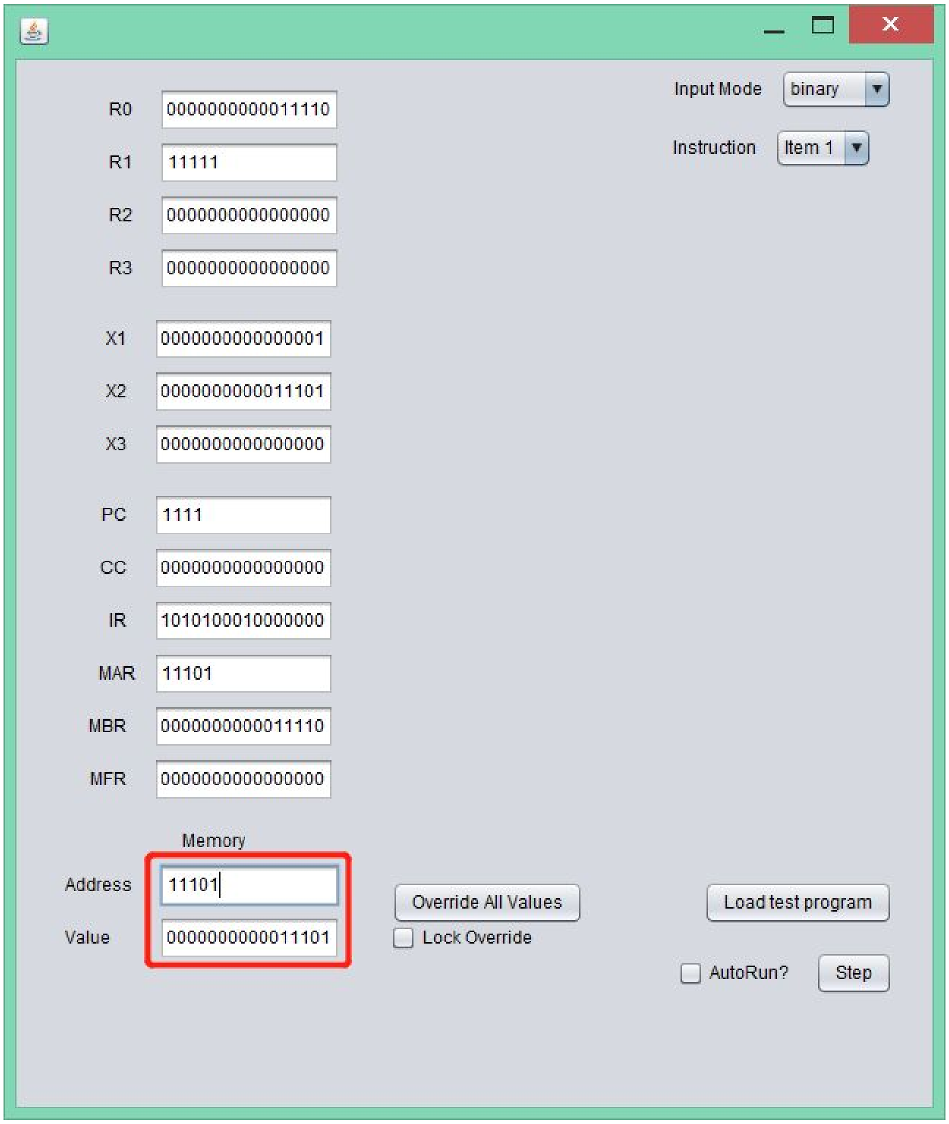


(4) Load index register from memory: 101001 00 10 1 11100. (EA = memory[30] = 31)
This instruction loads the content in effective address to index register X2 by indexing and indirecting. Consequently, the content of Index Register 2 becomes 29.



(5) Store index register to memory: 101010 00 10 0 00000. (EA = 29)

This instruction stores the value of index register 2 to memory[29].


R0
0000000000011110
R1
11111
R2
0000000000000000
R3
0000000000000000
X1
0000000000000001
X2
0000000000011101
X3
0000000000000000
PC
1111
CC
0000000000000000
IR
1010100010000000
MAR
11101
MBR
0000000000011110
MFR
0000000000000000
Input Mode
binary
Instruction
Item 1
Memory
Address
11101
Value
0000000000011101
Override All Values
Lock Override
Load test program
AutoRun?
Step