CISC SIMULATOR

Manual

V 1.0

**Group 8**

Md Shahjalal

Tianyou Bao

Xuzheng Lu

Contents

[1 Introduction 1](#_Toc19995876)

[1.1 Debugging Panel 1](#_Toc19995877)

[1.1.1 Register Indicators Area 1](#_Toc19995878)

[1.1.2 Memory Area 2](#_Toc19995879)

[1.1.3 Controller Area 2](#_Toc19995880)

[2.2 Classic Panel 3](#_Toc19995881)

[2 Operation 3](#_Toc19995882)

[2.1 Writing Values to Registers 3](#_Toc19995883)

[2.2 Writing Values to Memory 3](#_Toc19995884)

[2.2.1 Using Memory Address Register and Memory Buffer Register 3](#_Toc19995885)

[2.2.2 Modifying the Memory Area 4](#_Toc19995886)

[2.3 Executing Instructions 5](#_Toc19995887)

[2.3.1 Executing Instructions Step-by-Step 5](#_Toc19995888)

[2.3.1 Executing Instructions Automatically 5](#_Toc19995889)

[3 Instructions Reference 6](#_Toc19995890)

[3.1 Load/Store Instructions 6](#_Toc19995891)

[3.1.1 LDR Instruction 6](#_Toc19995892)

[3.1.2 STR Instruction 6](#_Toc19995893)

[3.1.3 LDA Instruction 7](#_Toc19995894)

[3.1.4 LDX Instruction 7](#_Toc19995895)

[3.1.5 STX Instruction 7](#_Toc19995896)

[3.2 Other Instructions 7](#_Toc19995897)

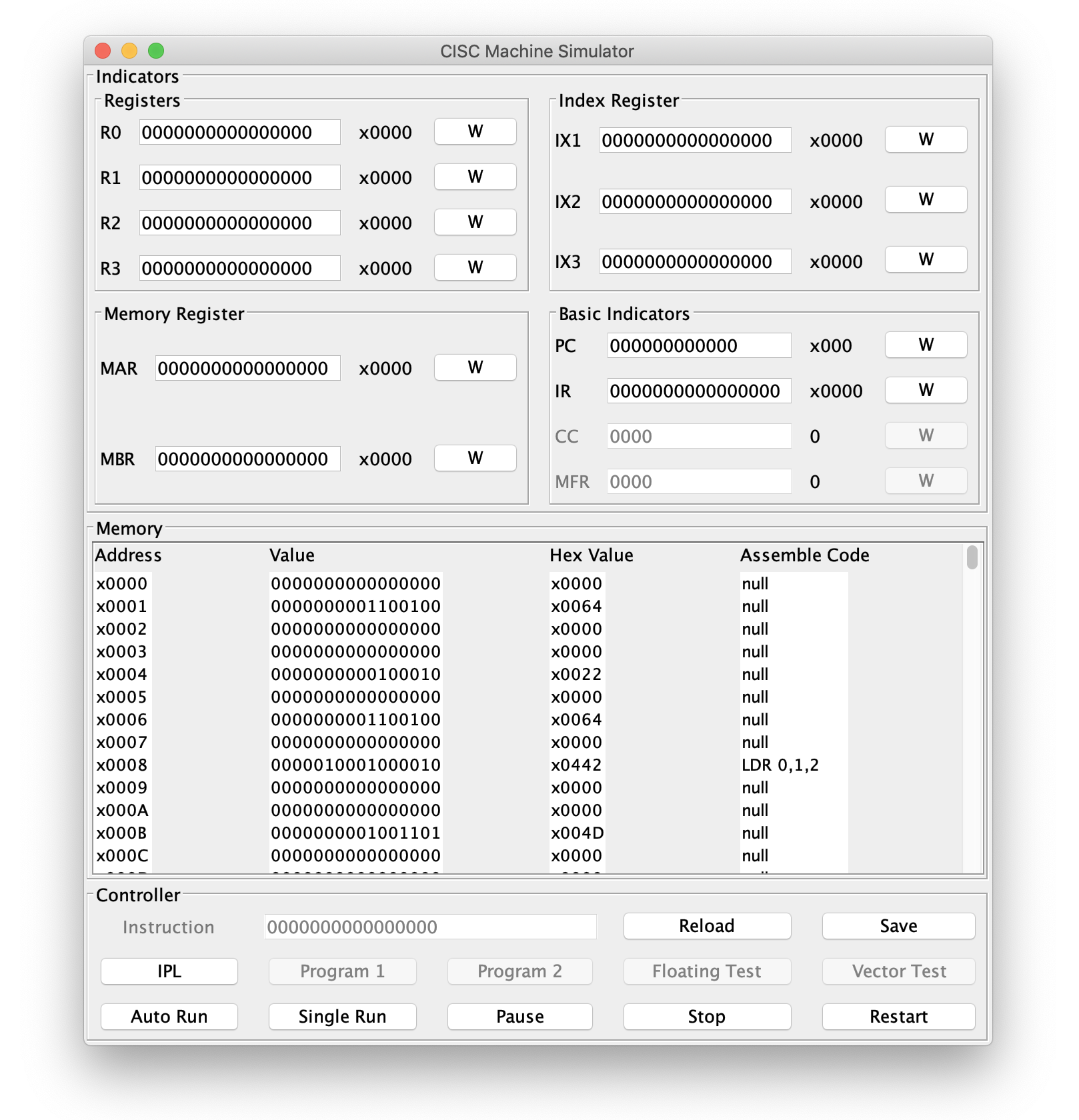
[3.2.1 Halt Instruction 7](#_Toc19995898)

# 1 Introduction

This simulator is a simulation of a Complex Instruction Set Computer (CISC). Two panels are designed for the simulator.

## 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

#### 

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.

|  |  |
| --- | --- |
| **Button** | **Description** |
| Reload | Initialize the values |
| Save | Save inputs |
| IPL | Pre-load a program |
| Auto Run | Run instructions automatically |
| Single Run | Run instructions step by step |
| Pause | Pause the machine |
| Stop | Stop the machine |
| Restart | Restart the machine |

## 2.2 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.

# 2 Operation

## 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

#### 

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

#### 

**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

#### 

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

#### 

**Step 3**: Input a value into the MBR box

#### 

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

#### 

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

#### 

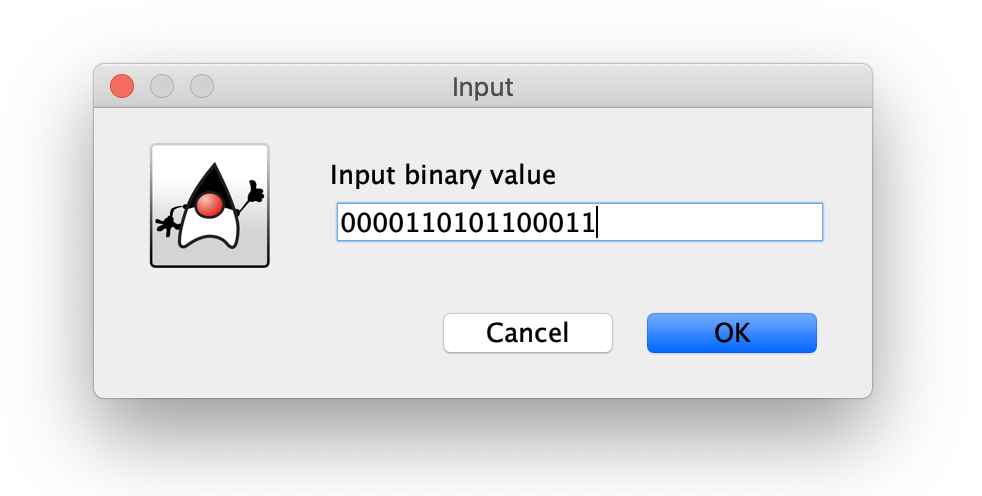
#### 

### 2.2.2 Modifying the Memory Area

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

#### 

**Step 2**: An input window as the following will pop up. Input the value that needs to write to the memory



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

#### 

## 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

#### 

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

#### 

**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.

#### 

### 2.3.1 Executing Instructions Automatically

**Step 1**: Store instructions to the Memory

#### 

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

#### 

**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.

#### 

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

# 3 Instructions Reference

## 3.1 Load/Store Instructions

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

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

|  |  |  |
| --- | --- | --- |
| **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 |

### 3.1.1 LDR Instruction

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

### 3.1.2 STR Instruction

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

### 3.1.3 LDA Instruction

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

### 3.1.4 LDX Instruction

|  |  |
| --- | --- |
| Instruction: | LDX x, address[, I] |
| Octal-Opcode: | 41 |
| Binary-Opcode: | 100001 |
| Function: | Loads Index Register from Memory |

### 3.1.5 STX Instruction

|  |  |
| --- | --- |
| Instruction: | STX x, address[, I] |
| Octal-Opcode: | 42 |
| Binary-Opcode: | 100010 |
| Function: | Stores Index Register to Memory |

## 3.2 Other Instructions

### 3.2.1 Halt Instruction

|  |  |  |
| --- | --- | --- |
| 000000 | 0000 | 00000000 |
| 0 5 | 6 9 | 1 1  0 5 |

|  |  |
| --- | --- |
| Instruction: | Halt |
| Octal-Opcode: | 00 |
| Binary-Opcode: | 000000 |
| Function: | Stops the machine |