CISC SIMULATOR

Manual

V 3.0

**Group 8**

Md Shahjalal

Tianyou Bao

Xuzheng Lu

Contents

[1 Introduction 1](#_Toc23439469)

[1.1 Debugging Panel 1](#_Toc23439470)

[1.1.1 Register Indicators Area 1](#_Toc23439471)

[1.1.2 Memory Area 2](#_Toc23439472)

[1.1.3 Controller Area 2](#_Toc23439473)

[1.2 Operation Panel (Console) 3](#_Toc23439474)

[1.3 Classic Panel 4](#_Toc23439475)

[1.4 Themes 4](#_Toc23439476)

[2 Basic Operations 5](#_Toc23439477)

[2.1 Writing Values to Registers 5](#_Toc23439478)

[2.2 Writing Values to Memory 5](#_Toc23439479)

[2.2.1 Using Memory Address Register and Memory Buffer Register 6](#_Toc23439480)

[2.2.2 Modifying the Memory Area 6](#_Toc23439481)

[2.3 Executing Instructions 7](#_Toc23439482)

[2.3.1 Executing Instructions Step-by-Step 7](#_Toc23439483)

[2.3.1 Executing Instructions Automatically 8](#_Toc23439484)

[3 Executing Programs 9](#_Toc23439485)

[3.1 Executing Program 1 10](#_Toc23439486)

[3.1.1 Program 1 Description 10](#_Toc23439487)

[3.1.2 Running Program 1 10](#_Toc23439488)

[3.2 Executing Program 2 11](#_Toc23439489)

[3.1.1 Program 2 Description 11](#_Toc23439490)

[3.1.2 Running Program 2 11](#_Toc23439491)

[3.3 Executing a Custom Program Using IPL 12](#_Toc23439492)

[3.3.1 Using Operation Panel (Console) 12](#_Toc23439493)

[3.3.2 Using Debugging Panel 12](#_Toc23439494)

[4 Instructions Reference 12](#_Toc23439495)

[4.1 Load/Store Instructions 12](#_Toc23439496)

[4.1.1 (01) LDR 12](#_Toc23439497)

[4.1.2 (02) STR 13](#_Toc23439498)

[4.1.3 (03) LDA 13](#_Toc23439499)

[4.1.4 (41) LDX 13](#_Toc23439500)

[4.1.5 (42) STX 14](#_Toc23439501)

[4.2 Arithmetic and Logical Instructions 14](#_Toc23439502)

[4.2.1 (04) AMR 14](#_Toc23439503)

[4.2.2 (05) SMR 15](#_Toc23439504)

[4.2.3 (06) AIR 15](#_Toc23439505)

[4.2.4 (07) SIR 15](#_Toc23439506)

[4.2.5 (20) MLT 16](#_Toc23439507)

[4.2.6 (21) DVD 16](#_Toc23439508)

[4.2.7 (22) TRR 16](#_Toc23439509)

[4.2.8 (23) AND 17](#_Toc23439510)

[4.2.9 (24) ORR 17](#_Toc23439511)

[4.2.10 (25) NOT 17](#_Toc23439512)

[4.3 Transfer Instructions 17](#_Toc23439513)

[4.3.1 (10) JZ 18](#_Toc23439514)

[4.3.2 (11) JNE 18](#_Toc23439515)

[4.3.3 (12) JCC 18](#_Toc23439516)

[4.3.4 (13) JMA 19](#_Toc23439517)

[4.3.5 (14) JSR 19](#_Toc23439518)

[4.3.6 (15) RFS 19](#_Toc23439519)

[4.3.7 (16) SOB 20](#_Toc23439520)

[4.3.8 (17) JGE 20](#_Toc23439521)

[4.4 Shift/Rotate Instructions 20](#_Toc23439522)

[4.4.1 (31) SRC 21](#_Toc23439523)

[4.4.2 (32) RRC 21](#_Toc23439524)

[4.5 Floating Point and Vector Instructions 21](#_Toc23439525)

[4.5.1 (33) FADD 22](#_Toc23439526)

[4.5.2 (34) FSUB 22](#_Toc23439527)

[4.5.3 (35) VADD 23](#_Toc23439528)

[4.5.4 (36) VSUB 23](#_Toc23439529)

[4.5.5 (37) CNVRT 23](#_Toc23439530)

[4.5.6 (50) LDFR 24](#_Toc23439531)

[4.5.7 (51) STFR 24](#_Toc23439532)

[4.6 I/O Instructions 24](#_Toc23439533)

[4.6.1 (61) IN 24](#_Toc23439534)

[4.6.2 (62) OUT 25](#_Toc23439535)

[4.6.3 (63) CHK 25](#_Toc23439536)

[4.7 Other Instructions 25](#_Toc23439537)

[4.7.1 (00) HALT 25](#_Toc23439538)

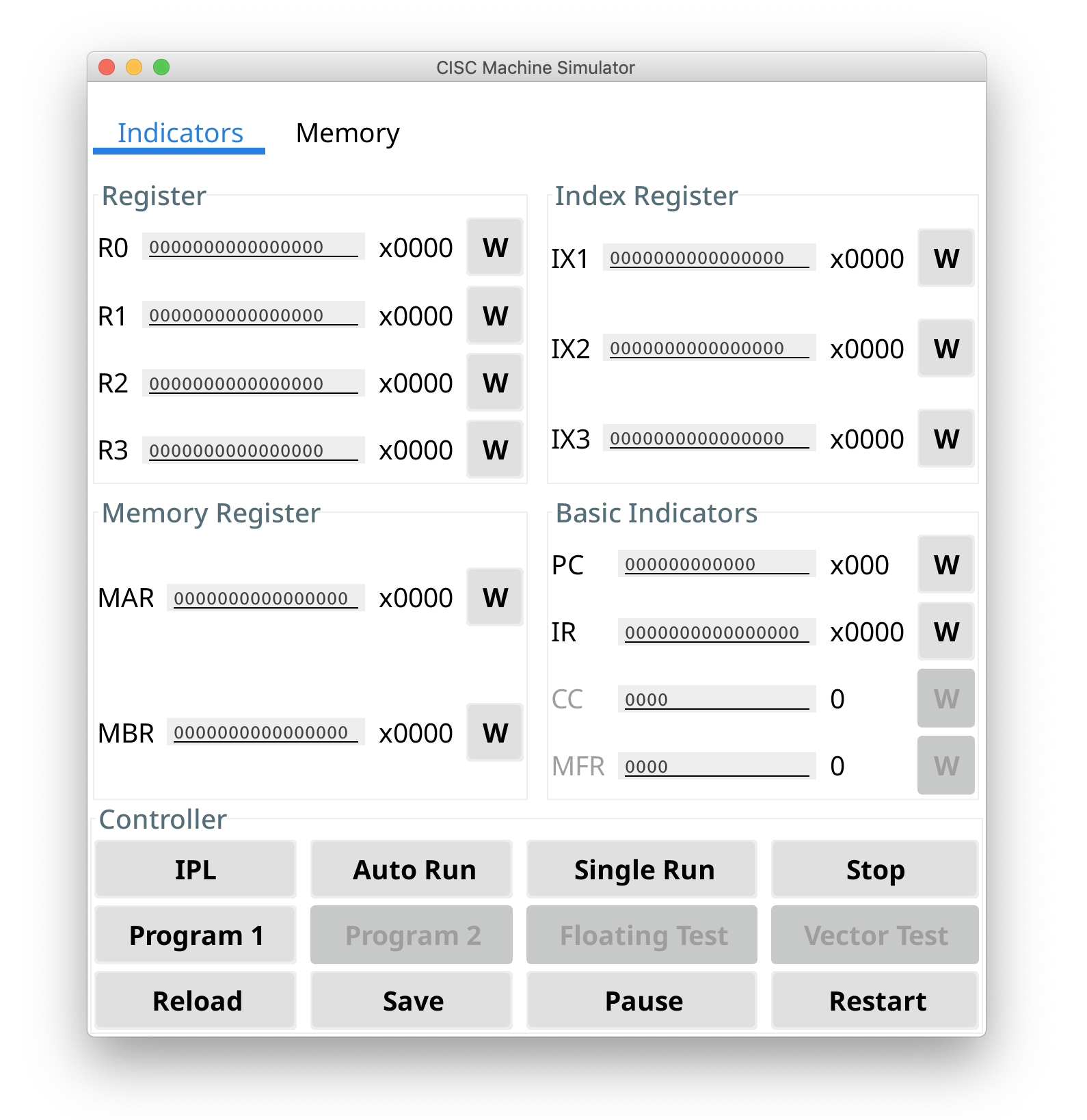
[4.7.2 (30) TRAP 26](#_Toc23439539)

# 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.

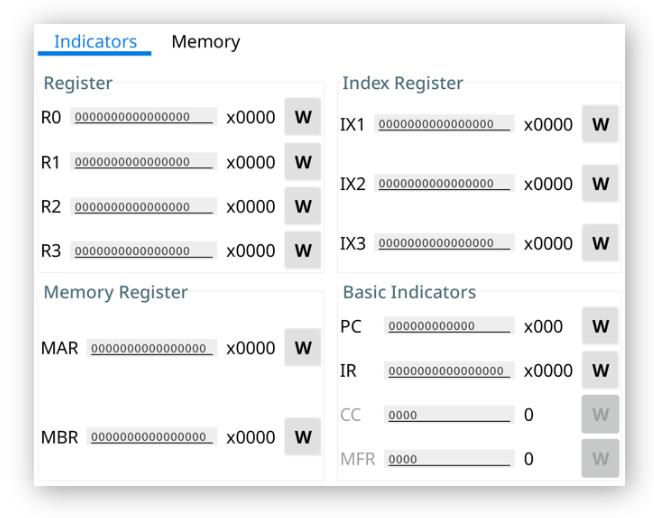
## 1.1 Debugging Panel

**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

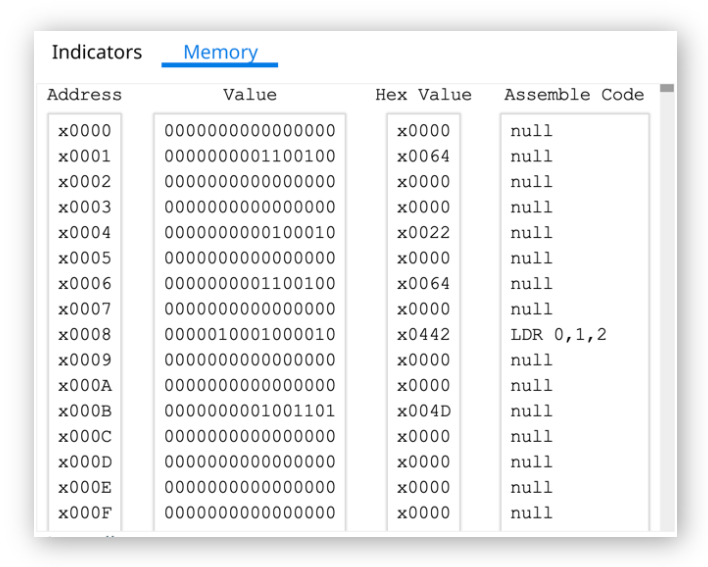


The Register Indicators display the values of all kinds of registers.

* 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



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.

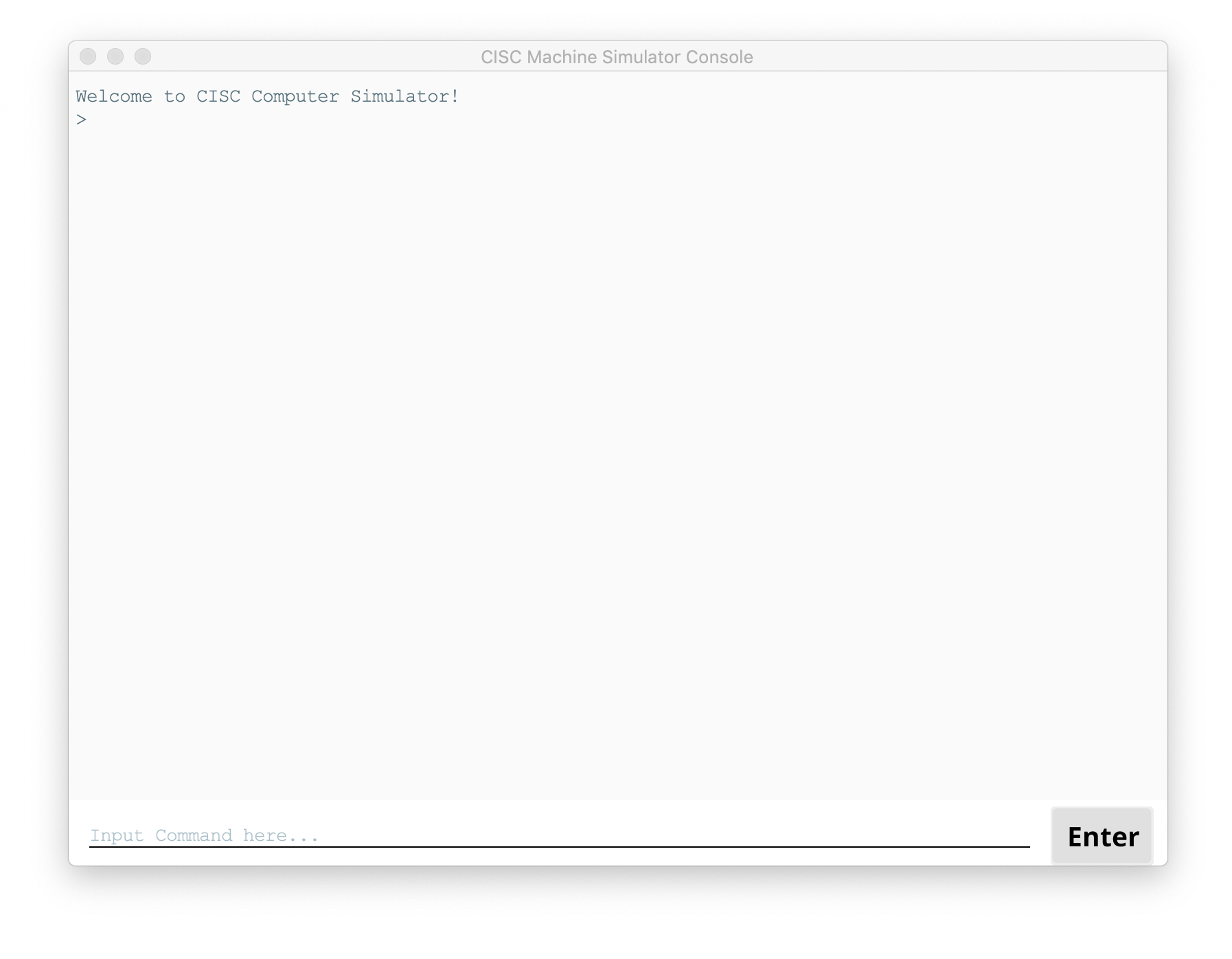


Functions of the buttons in Controller Area:

|  |  |
| --- | --- |
| **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 |
| Program 2 | Run Program 2 |
| Floating Test | Test Floating Point Operations |
| Vector Test | Test Vector Operations |
| 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 the theme of UI  Format: switch theme {$THEME\_NAME}  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.

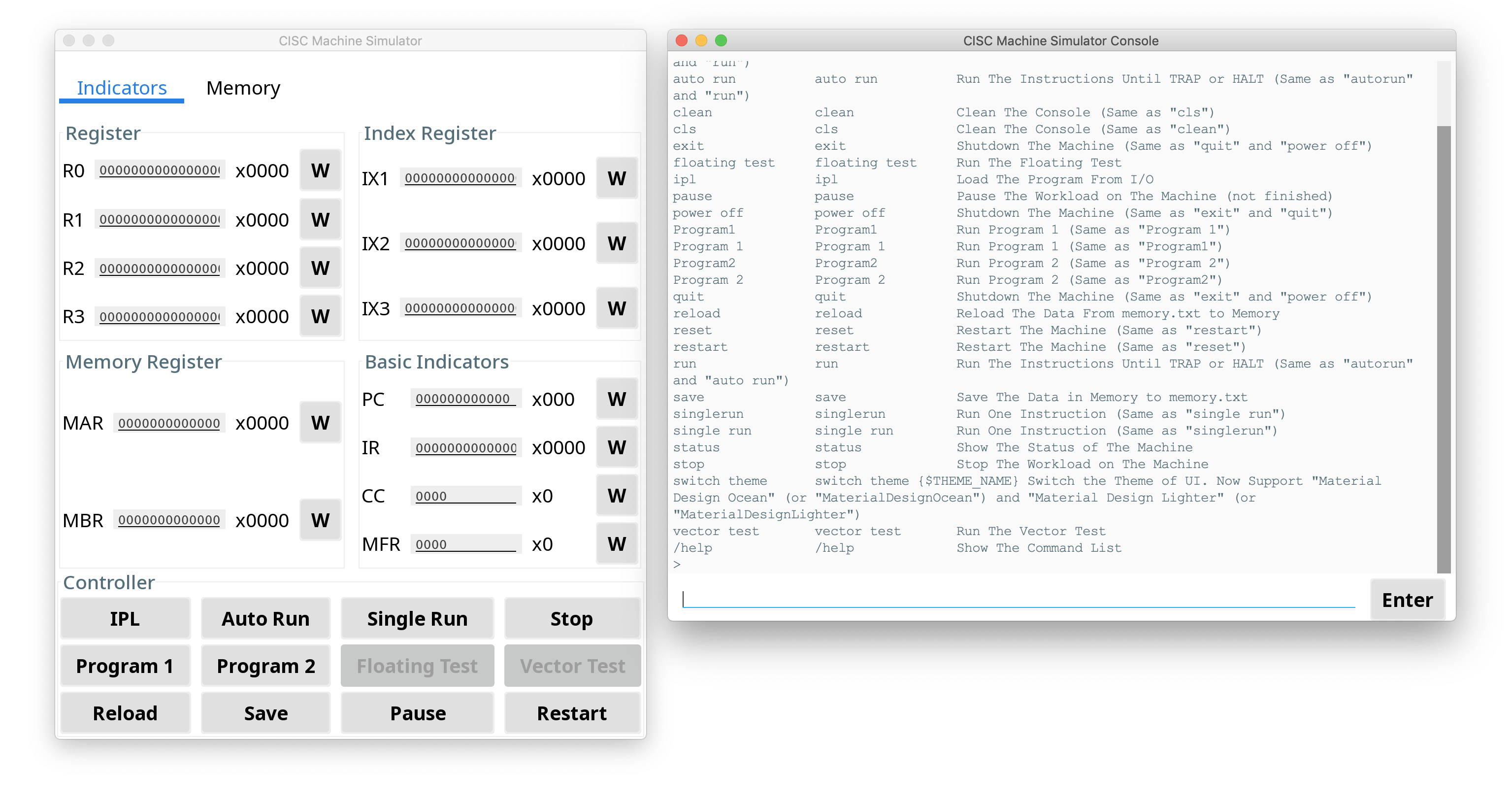
The **Classic Panel** has not been finished yet and will be released in the next version.

## 1.4 Themes

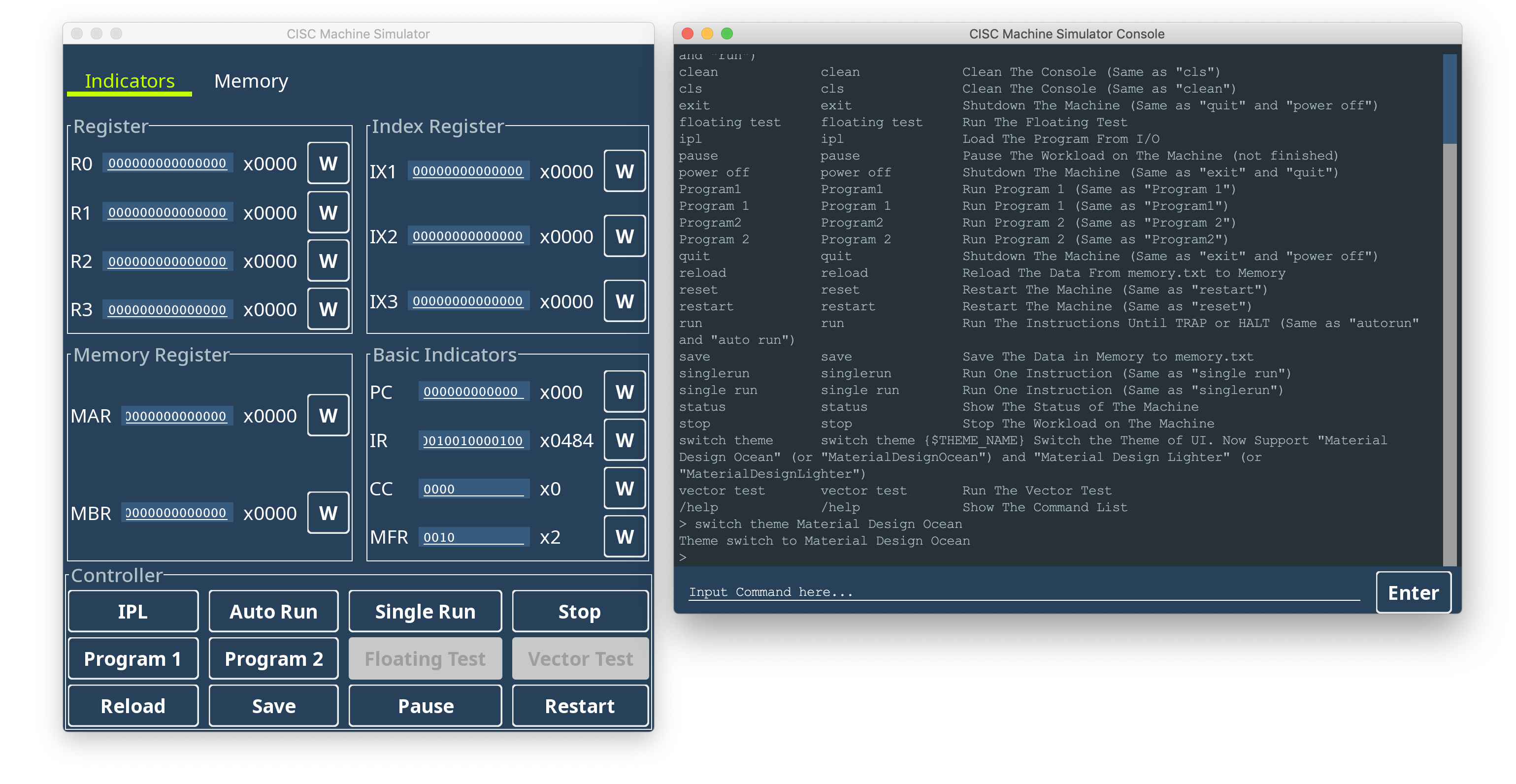
Two themes, Material Design Ocean and Material Design Lighter, are supported now.

To change the theme, input ‘switch theme {$THEME\_NAME}’ in **Operation Panel**.

**Material Design Lighter Theme (default)**



**Material Design Ocean Theme**

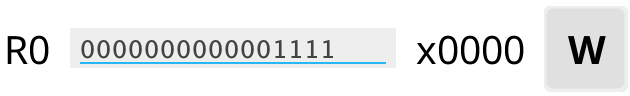


# 2 Basic Operations

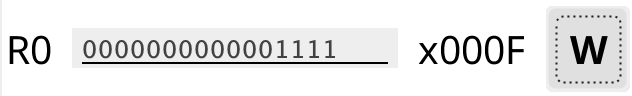
## 2.1 Writing Values to Registers

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

**Step 1**: Input a value into the box.



**Step 2**: Click the ‘W’ button at right to write the value to the register.



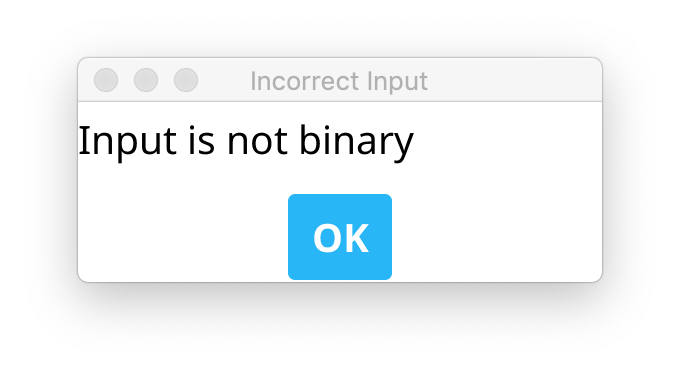
**Step 3**: Done! The value will be written to the Register.

**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

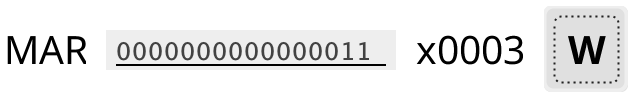
Two methods are acceptable to write a value to the Memory.

### 2.2.1 Using Memory Address Register and Memory Buffer Register

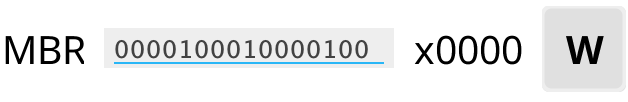
**Step 1**: Input a value into the MAR box.



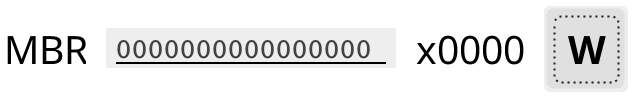
**Step 2**: Click the ‘W’ button of MAR.



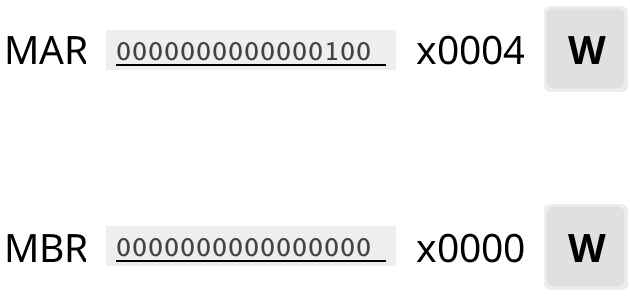
**Step 3**: Input a value into the MBR box.

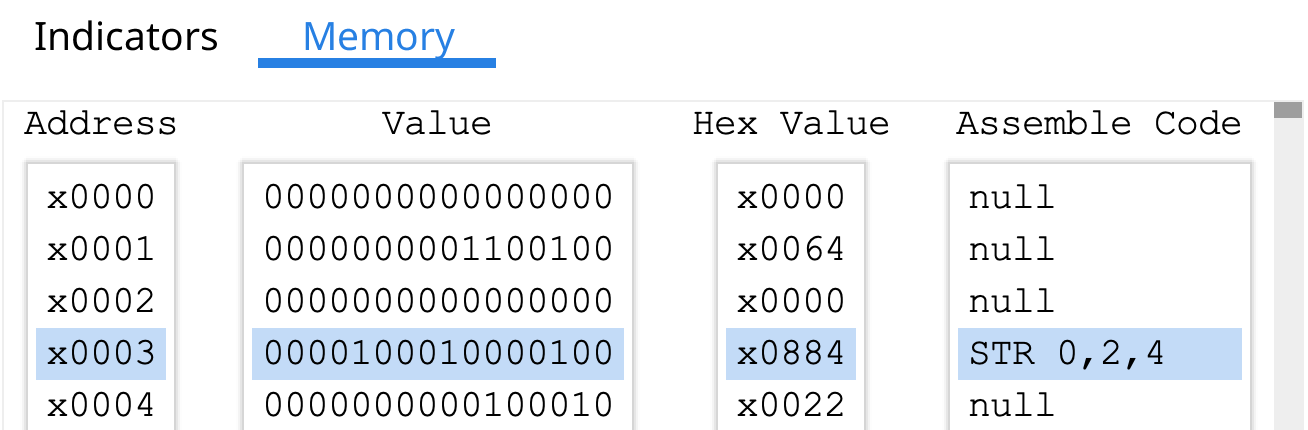


**Step 4**: Click the ‘W’ button of MBR.



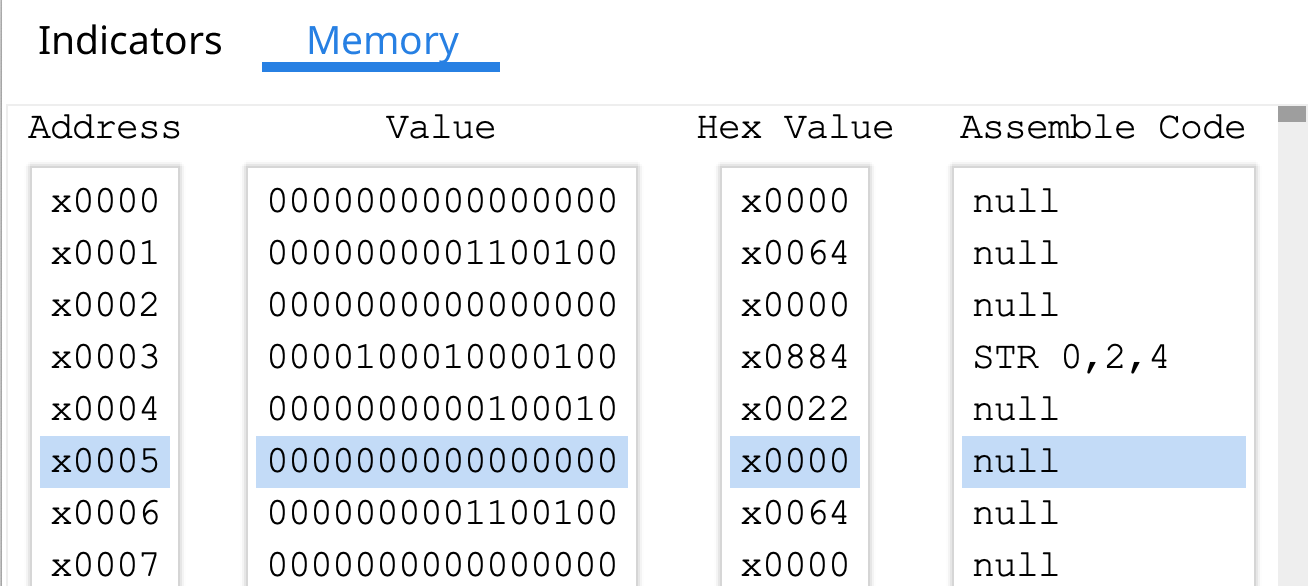
**Step 5**: Done! The value of MAR will be written to the Memory, and the MAR will automatically change to the next address.



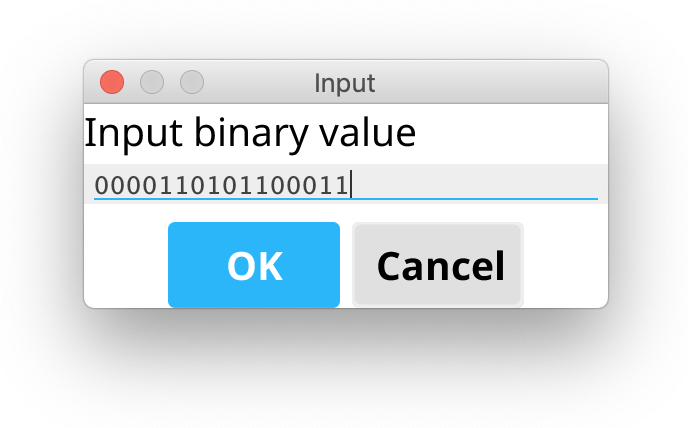


### 2.2.2 Modifying the Memory Area

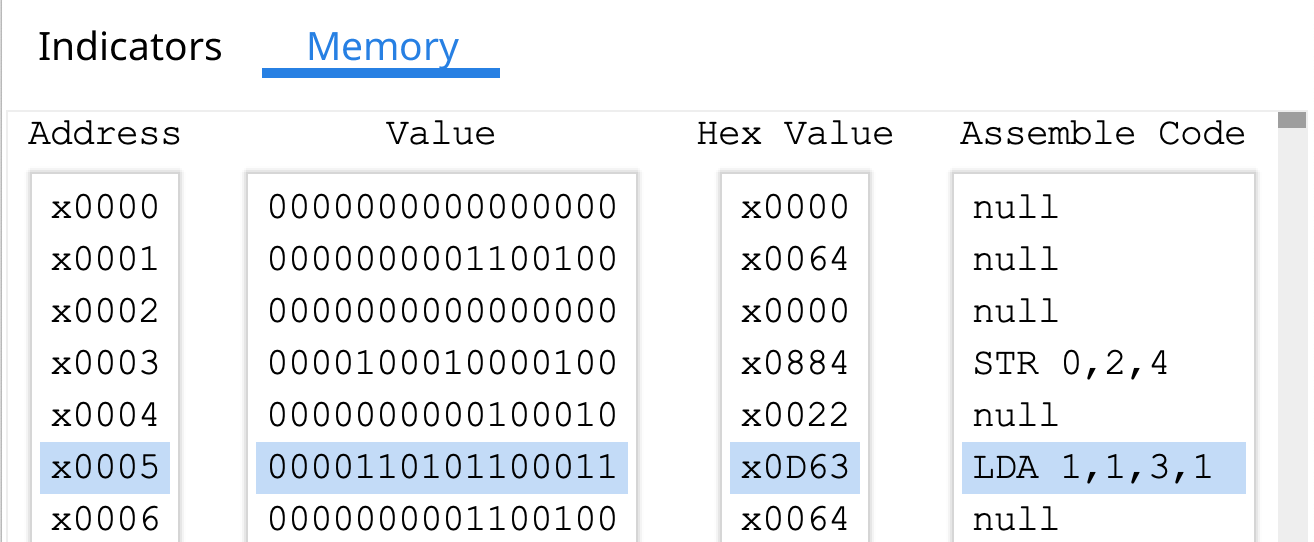
**Step 1**: Double click the memory row that needs to modify.



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



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

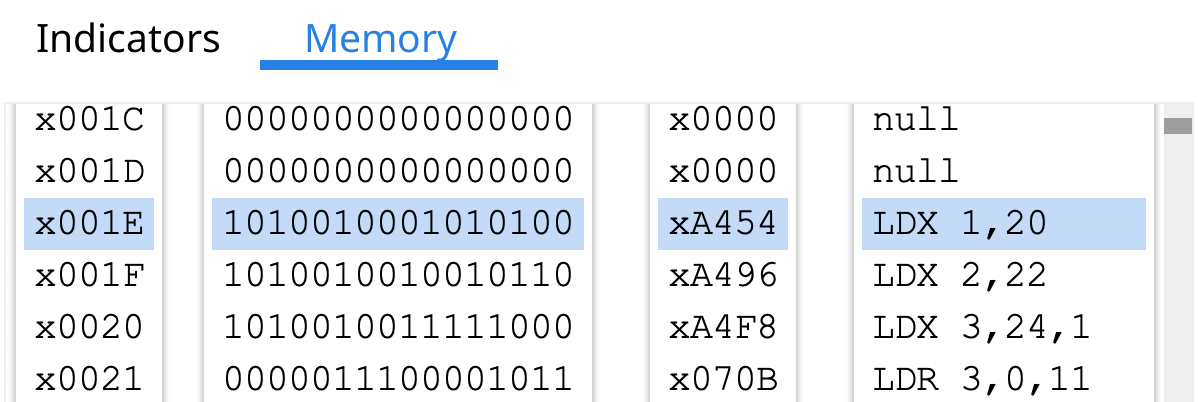


## 2.3 Executing Instructions

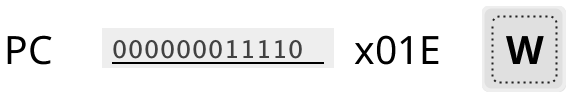
Instruction can be executed step-by-step or automatically.

### 2.3.1 Executing Instructions Step-by-Step

**Step 1**: Store an instruction to the Memory.

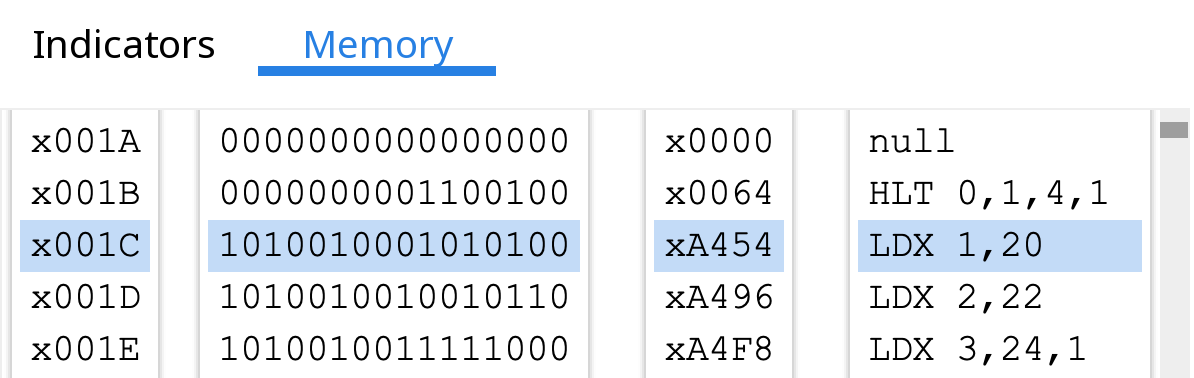


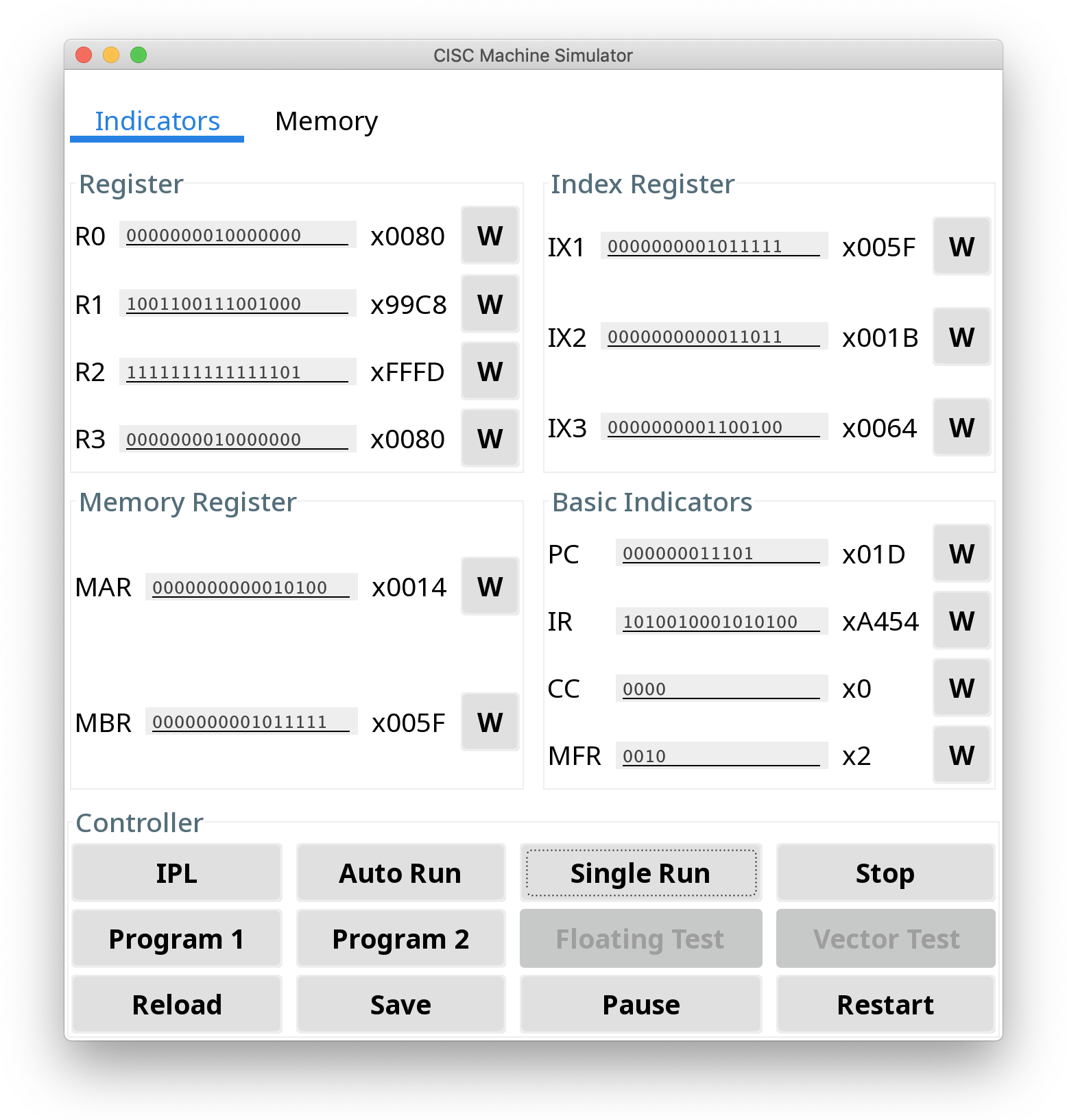
**Step 2**: Write the address of the instruction to the Program Counter (PC).



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

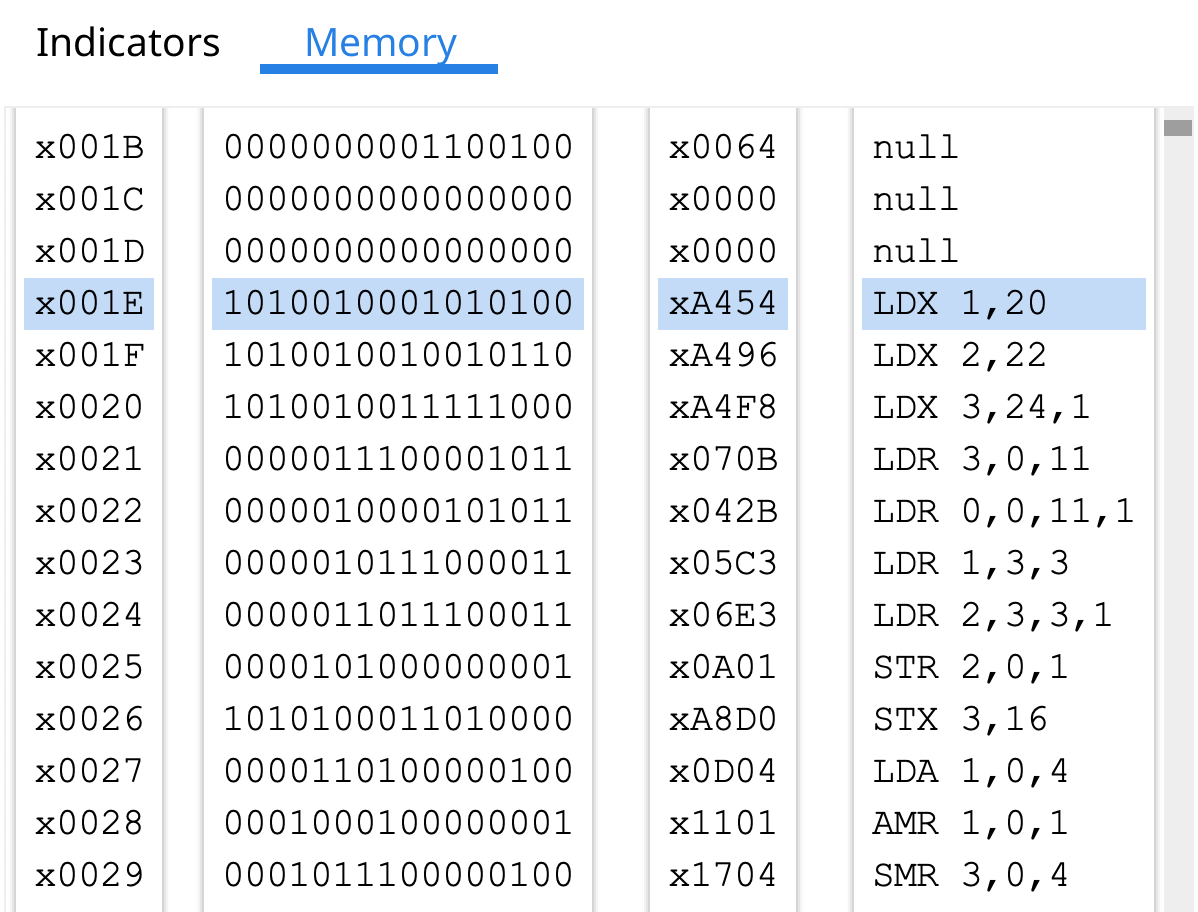
* The Program Counter will automatically point to the next address of Memory.
* The Instruction Register will store the last executed instruction.



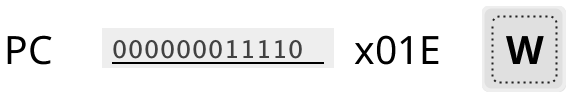


### 2.3.1 Executing Instructions Automatically

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

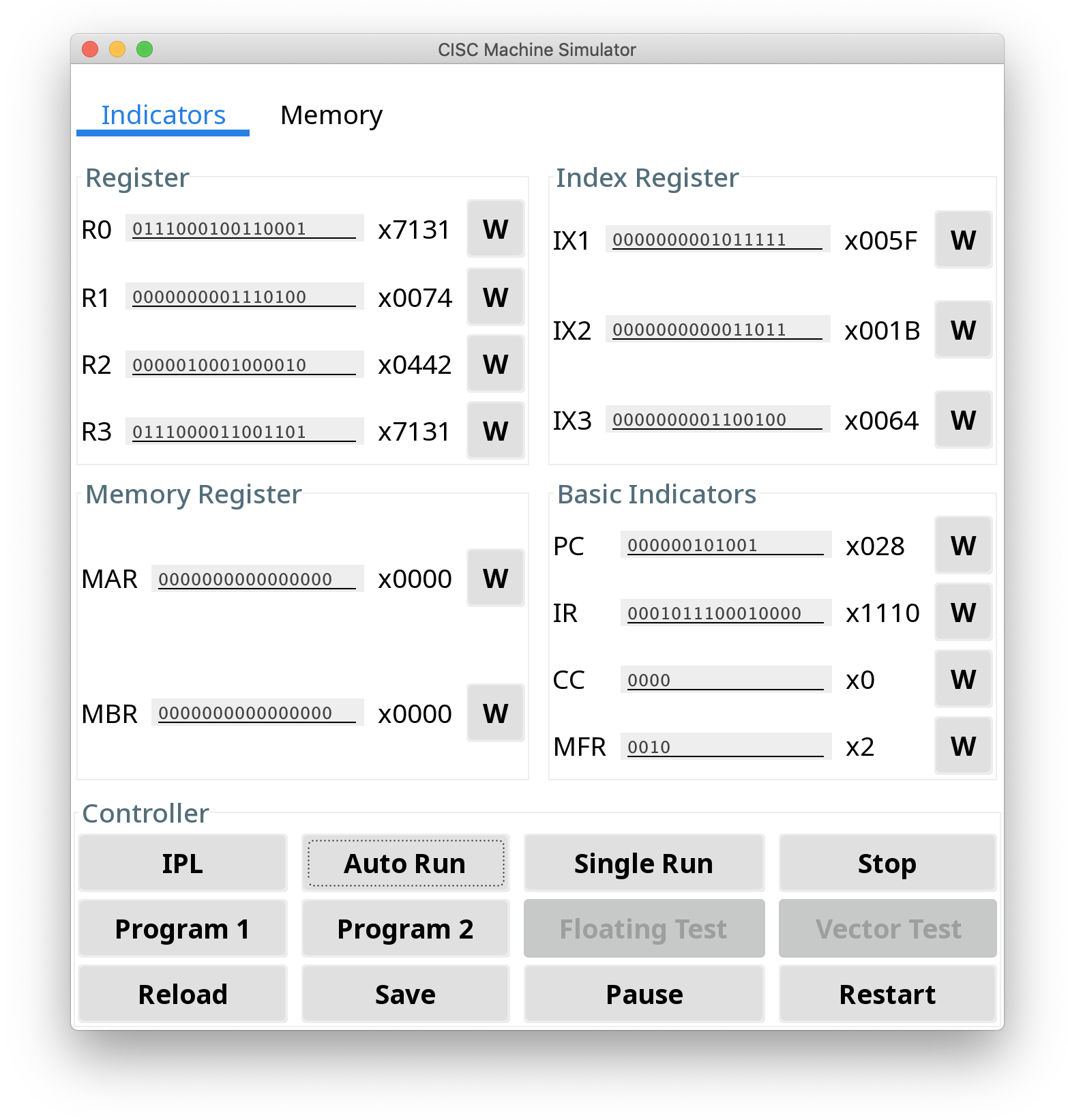


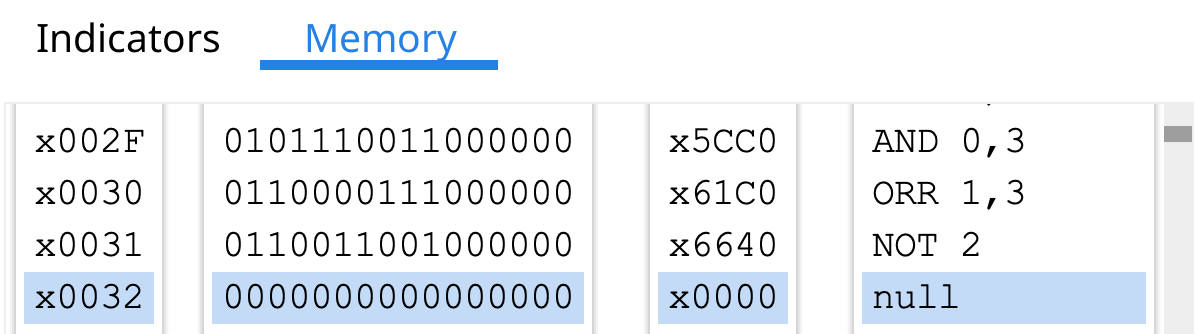
**Step 2**: Write the address of the **starting** instruction to the Program Counter (PC).

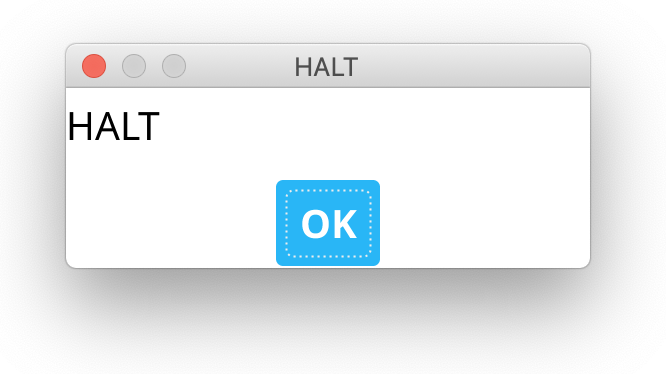


**Step 3**: Click the ‘Auto Run’ button, and then the instructions will be executed automatically.

* 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 to pause the program or the ‘Stop’ button to stop the program.

# 3 Executing Programs

## 3.1 Executing Program 1

### 3.1.1 Program 1 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.

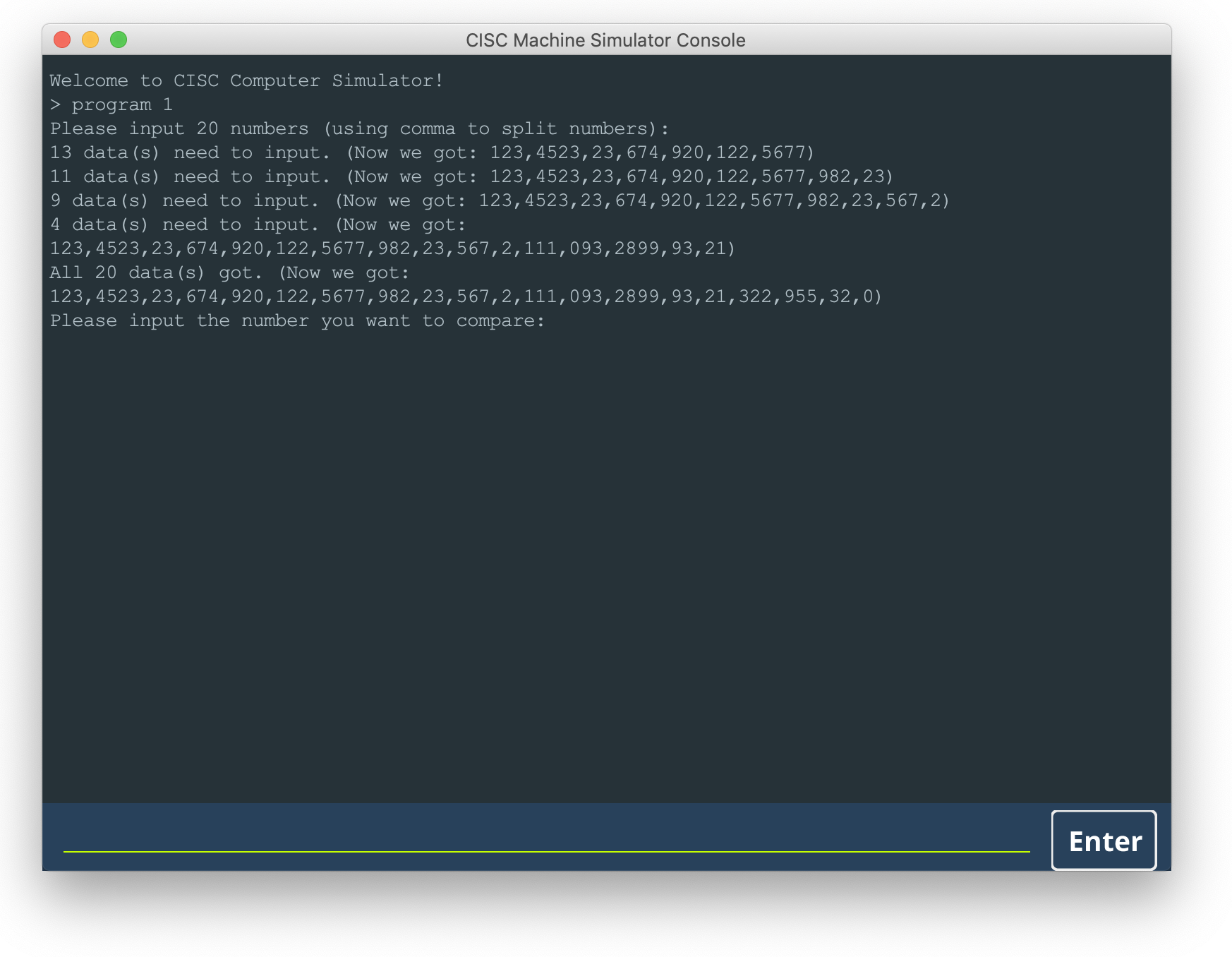
### 3.1.2 Running Program 1

**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**.

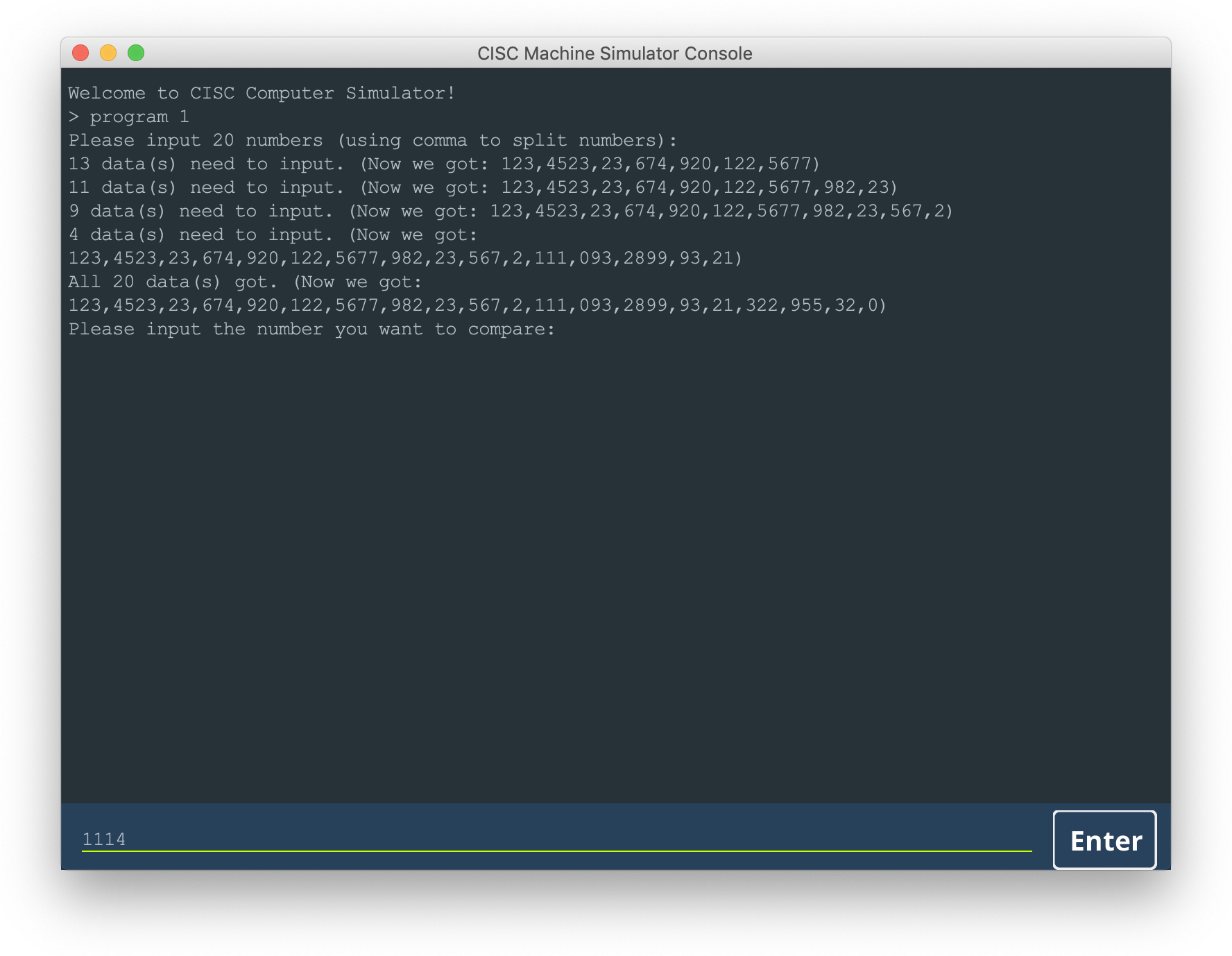




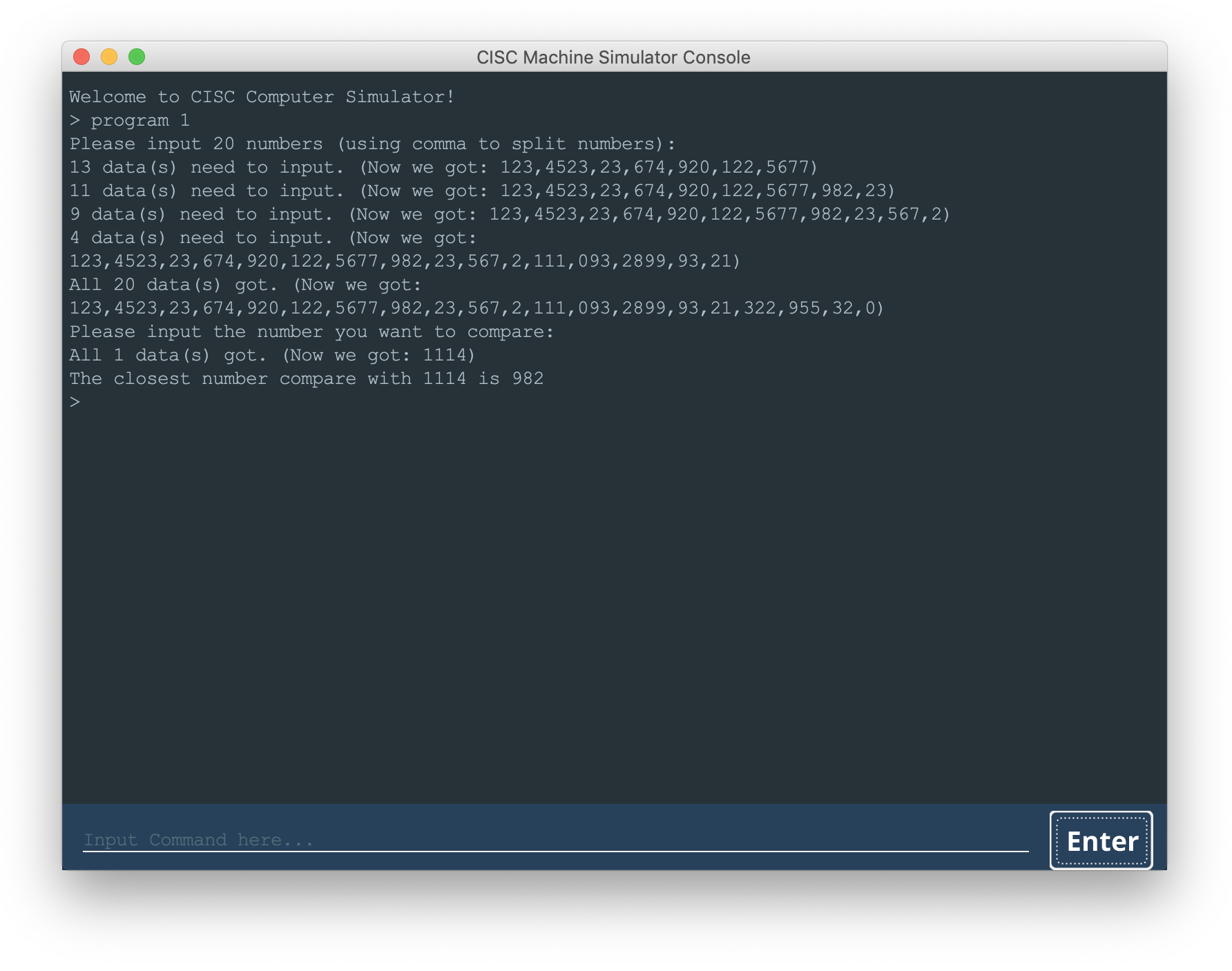
**Step 2**: Input numbers (use comma to split numbers) and then click the ‘Enter’ button. You can fill the numbers several times to input the required 20 numbers.



**Step 3**: Input the number for comparing.



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



### 3.1.3 Program 1 Files

Two

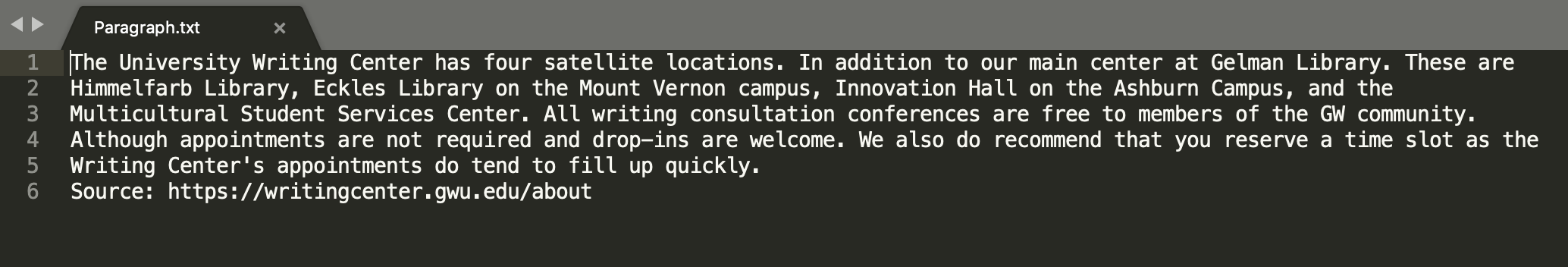
## 3.2 Executing Program 2

### 3.1.1 Program 2 Description

Program 2 is a program that reads a set of a paragraph of 6 sentences from a file into memory. It prints the sentences on the console printer. It then asks the user for a word. It searches the paragraph to see if it contains the word. If so, it prints out the word, the sentence number, and the word number in the sentence.

### 3.1.2 Running Program 2

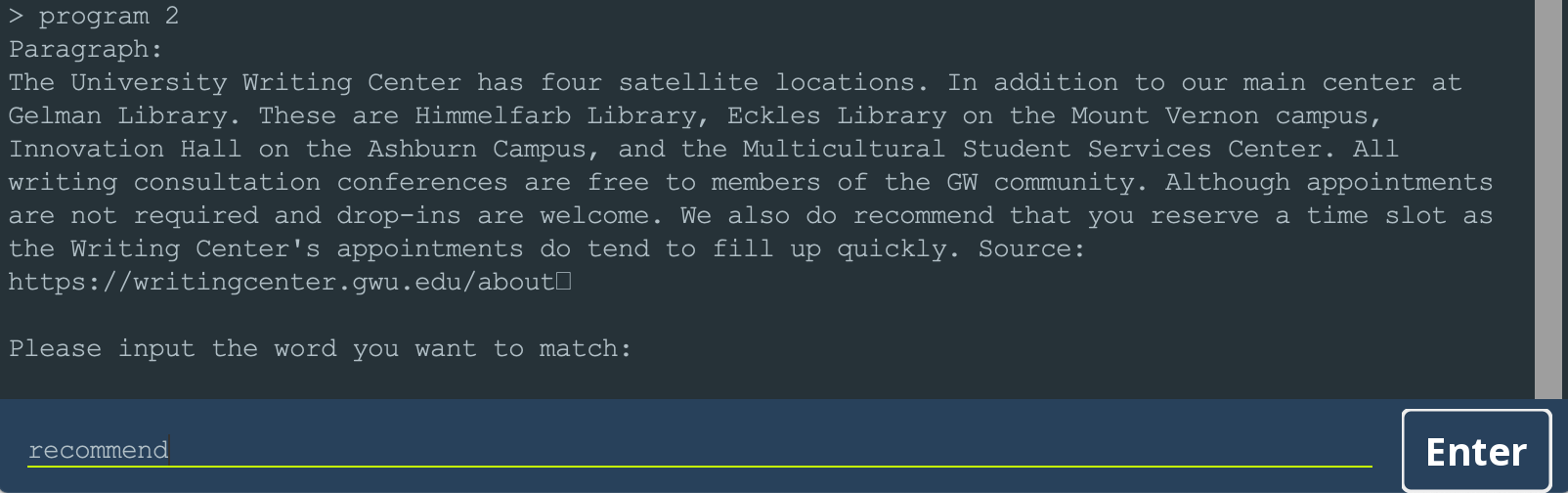
**Step 1**: Write a paragraph of 6 sentences into the text file ' Paragraph.txt'.



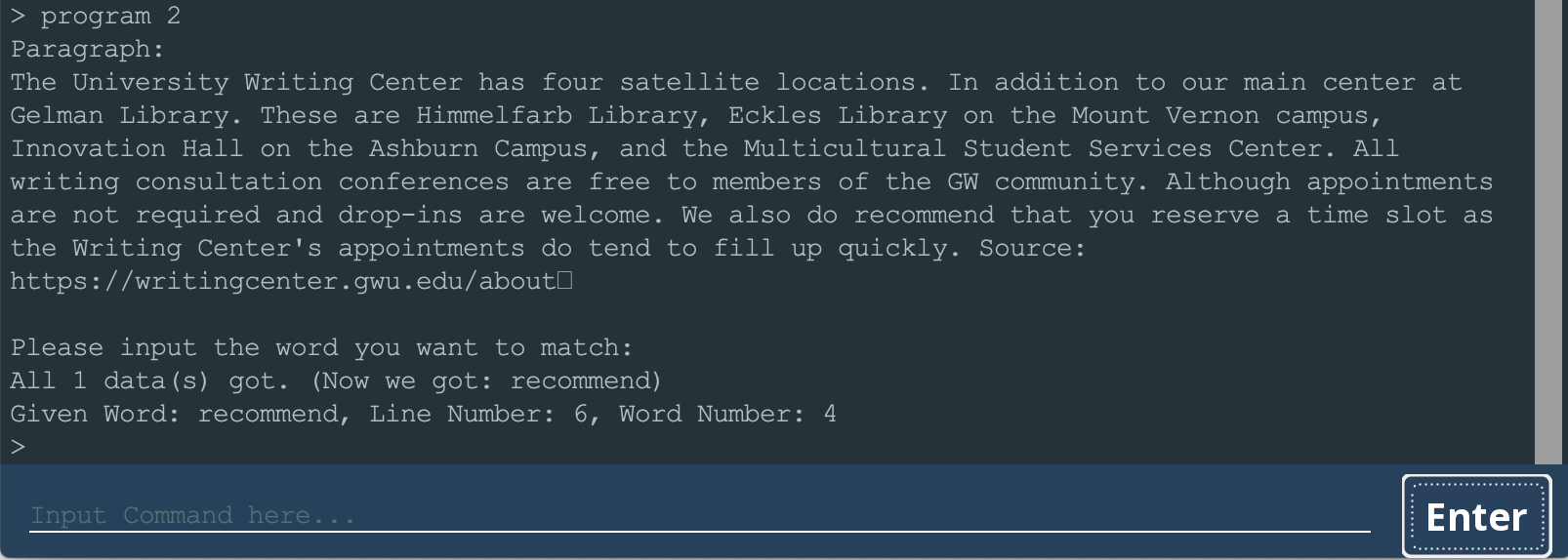
**Step 2**: Input ‘program 2’ or ‘program2’ in the **Operation Panel (Console)** and then click the ‘Enter’ button. Or click the ‘Program 2’ button in the **Debugging Panel**.



**Step 3**: The paragraph will be loaded from the text file to Memory and will be displayed in the **Operation Panel (Console)**. Then, input the word to be matched to the paragraph.



**Step 4**: Done! The **Operation Panel (Console)** will print out the word, the sentence number in the paragraph, and the word number in the sentence.



## 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  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 |

### 4.1.1 (01) LDR

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

|  |  |
| --- | --- |
| Instruction: | LDR r, x, address[, I] |
| Octal-Opcode: | 01 |
| Binary-Opcode: | 000001 |
| Function: | Loads Register from Memory |
| Notes: | r = 0...3  r <- c(EA)  r <- c(c(EA)), if I bit set |

### 4.1.2 (02) STR

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

|  |  |
| --- | --- |
| Instruction: | STR r, x, address[, I] |
| Octal-Opcode: | 02 |
| Binary-Opcode: | 000010 |
| Function: | Stores Register to Memory |
| Notes: | r = 0...3  Memory(EA) <- c(r) |

### 4.1.3 (03) LDA

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

|  |  |
| --- | --- |
| Instruction: | LDA r, x, address[, I] |
| Octal-Opcode: | 03 |
| Binary-Opcode: | 000011 |
| Function: | Loads Register with Address |
| Notes: | r = 0...3  r <- EA |

### 4.1.4 (41) LDX

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 101001 |  | IX | I | Address |
| 0 5 | 6 7 | 8 9 | 1  0 | 1 1  1 5 |

|  |  |
| --- | --- |
| Instruction: | LDX x, address[, I] |
| Octal-Opcode: | 41 |
| Binary-Opcode: | 101001 |
| Function: | Loads Index Register from Memory |
| Notes: | Xx <- c(EA) |

### 4.1.5 (42) STX

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 101010 |  | IX | I | Address |
| 0 5 | 6 7 | 8 9 | 1  0 | 1 1  1 5 |

|  |  |
| --- | --- |
| Instruction: | STX x, address[, I] |
| Octal-Opcode: | 42 |
| Binary-Opcode: | 101010 |
| Function: | Stores Index Register to Memory |
| Notes: | Memory(EA) <- c(Xx) |

## 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  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 |

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

|  |  |  |  |
| --- | --- | --- | --- |
| Opcode | Rx | Ry |  |
| 0 5 | 6 7 | 8 9 | 1 1  0 5 |

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 000100 | R | IX | I | Address |
| 0 5 | 6 7 | 8 9 | 1  0 | 1 1  1 5 |

|  |  |
| --- | --- |
| Instruction: | AMR r, x, address[, I] |
| Octal-Opcode: | 04 |
| Binary-Opcode: | 000100 |
| Function: | Add Memory to Register |
| Notes: | r = 0...3  r <- c(r) + c(EA) |

### 4.2.2 (05) SMR

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 000101 | R | IX | I | Address |
| 0 5 | 6 7 | 8 9 | 1  0 | 1 1  1 5 |

|  |  |
| --- | --- |
| Instruction: | SMR r, x, address[, I] |
| Octal-Opcode: | 05 |
| Binary-Opcode: | 000101 |
| Function: | Subtract Memory from Register |
| Notes: | r = 0...3  r <- c(r) - c(EA) |

### 4.2.3 (06) AIR

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 000110 | R |  |  | Immed |
| 0 5 | 6 7 | 8 9 | 1  0 | 1 1  1 5 |

|  |  |
| --- | --- |
| Instruction: | AIR r, immed |
| Octal-Opcode: | 06 |
| Binary-Opcode: | 000110 |
| Function: | Add Immediate to Register |
| Notes: | r = 0...3  r <- c(r) + Immed  if Immed = 0, does nothing  if c(r) = 0, loads r with Immed |

### 4.2.4 (07) SIR

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 000111 | R |  |  | Immed |
| 0 5 | 6 7 | 8 9 | 1  0 | 1 1  1 5 |

|  |  |
| --- | --- |
| Instruction: | SIR r, immed |
| Octal-Opcode: | 07 |
| Binary-Opcode: | 000111 |
| Function: | Subtract Immediate from Register |
| Notes: | r = 0...3  r <- c(r) - Immed  if Immed = 0, does nothing  if c(r) = 0, loads R1 with -(Immed) |

### 4.2.5 (20) MLT

|  |  |  |  |
| --- | --- | --- | --- |
| 010100 | Rx | Ry |  |
| 0 5 | 6 7 | 8 9 | 1 1  0 5 |

|  |  |
| --- | --- |
| Instruction: | MLT rx, ry |
| Octal-Opcode: | 20 |
| Binary-Opcode: | 010100 |
| Function: | Multiply Register by Register |
| Notes: | rx, rx+1 <- c(rx) \* c(ry)  rx, ry must be 0 or 2  rx contains the high order bits  rx+1 contains the low order bits of the result  Set OVERFLOW flag, if overflow |

### 4.2.6 (21) DVD

|  |  |  |  |
| --- | --- | --- | --- |
| 010101 | Rx | Ry |  |
| 0 5 | 6 7 | 8 9 | 1 1  0 5 |

|  |  |
| --- | --- |
| Instruction: | DVD rx, ry |
| Octal-Opcode: | 21 |
| Binary-Opcode: | 010101 |
| Function: | Divide Register by Register |
| Notes: | rx, rx+1 <- c(rx) / c(ry)  rx, ry must be 0 or 2  rx contains the quotient; rx+1 contains the remainder  If c(ry) = 0, set cc(3) to 1 (set DIVZERO flag) |

### 4.2.7 (22) TRR

|  |  |  |  |
| --- | --- | --- | --- |
| 010110 | Rx | Ry |  |
| 0 5 | 6 7 | 8 9 | 1 1  0 5 |

|  |  |
| --- | --- |
| Instruction: | TRR rx, ry |
| Octal-Opcode: | 22 |
| Binary-Opcode: | 010110 |
| Function: | Test the Equality of Register and Register |
| Notes: | If c(rx) = c(ry), set cc(4) <- 1;  Else, cc(4) <- 0 |

### 4.2.8 (23) AND

|  |  |  |  |
| --- | --- | --- | --- |
| 010111 | Rx | Ry |  |
| 0 5 | 6 7 | 8 9 | 1 1  0 5 |

|  |  |
| --- | --- |
| Instruction: | AND rx, ry |
| Octal-Opcode: | 23 |
| Binary-Opcode: | 010111 |
| Function: | Logical AND of Register and Register |
| Notes: | c(rx) <- c(rx) AND c(ry) |

### 4.2.9 (24) ORR

|  |  |  |  |
| --- | --- | --- | --- |
| 011000 | Rx | Ry |  |
| 0 5 | 6 7 | 8 9 | 1 1  0 5 |

|  |  |
| --- | --- |
| Instruction: | ORR rx, ry |
| Octal-Opcode: | 24 |
| Binary-Opcode: | 011000 |
| Function: | Logical OR of Register and Register |
| Notes: | c(rx) <- c(rx) OR c(ry) |

### 4.2.10 (25) NOT

|  |  |  |
| --- | --- | --- |
| 011001 | Rx |  |
| 0 5 | 6 7 | 8 1  5 |

|  |  |
| --- | --- |
| Instruction: | NOT rx |
| Octal-Opcode: | 25 |
| Binary-Opcode: | 011001 |
| Function: | Logical NOT of Register to Register |
| Notes: | C(rx) <- NOT c(rx) |

## 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 | 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 |

### 4.3.1 (10) JZ

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

|  |  |
| --- | --- |
| Instruction: | JZ r, x, address[, I] |
| Octal-Opcode: | 10 |
| Binary-Opcode: | 001010 |
| Function: | Jump if Zero |
| Notes: | r = 0...3  If c(r) = 0, then PC <- EA; Else PC <- PC+1 |

### 4.3.2 (11) JNE

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

|  |  |
| --- | --- |
| Instruction: | JNE r, x, address[, I] |
| Octal-Opcode: | 11 |
| Binary-Opcode: | 001011 |
| Function: | Jump if Not Equal |
| Notes: | r = 0...3  If c(r) != 0, then PC <- EA;  Else PC <- PC + 1 |

### 4.3.3 (12) JCC

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 001100 | CC | IX | I | Address |
| 0 5 | 6 7 | 8 9 | 1  0 | 1 1  1 5 |

|  |  |
| --- | --- |
| Instruction: | JCC cc, x, address[, I] |
| Octal-Opcode: | 12 |
| Binary-Opcode: | 001100 |
| Function: | Jump if Condition Code |
| Notes: | cc = 0...3, specifies the bit in the Condition Code Register to check  If cc = 1, PC <- EA; Else PC <- PC + 1 |

### 4.3.4 (13) JMA

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 001101 |  | IX | I | Address |
| 0 5 | 6 7 | 8 9 | 1  0 | 1 1  1 5 |

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

### 4.3.5 (14) JSR

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 001110 |  | IX | I | Address |
| 0 5 | 6 7 | 8 9 | 1  0 | 1 1  1 5 |

|  |  |
| --- | --- |
| Instruction: | JSR x, address[, I] |
| Octal-Opcode: | 14 |
| Binary-Opcode: | 001110 |
| Function: | Jump and Save Return Address |
| Notes: | R3 <- PC+1  PC <- EA  R0 should contain pointer to arguments  Argument list should end with -1 (all 1s) value |

### 4.3.6 (15) RFS

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 001111 |  |  |  | Immed |
| 0 5 | 6 7 | 8 9 | 1  0 | 1 1  1 5 |

|  |  |
| --- | --- |
| 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 |
| Notes: | R0 <- Immed  PC <- c(R3) |

### 4.3.7 (16) SOB

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 010000 | R | IX | I | Address |
| 0 5 | 6 7 | 8 9 | 1  0 | 1 1  1 5 |

|  |  |
| --- | --- |
| Instruction: | SOB r, x, address[, I] |
| Octal-Opcode: | 16 |
| Binary-Opcode: | 010000 |
| Function: | Subtract One and Branch |
| Notes: | r = 0...3  r <- c(r) - 1  If c(r) > 0, PC <- EA;  Else PC <- PC + 1 |

### 4.3.8 (17) JGE

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 010001 | R | IX | I | Address |
| 0 5 | 6 7 | 8 9 | 1  0 | 1 1  1 5 |

|  |  |
| --- | --- |
| Instruction: | JGE r, x, address[, I] |
| Octal-Opcode: | 17 |
| Binary-Opcode: | 010001 |
| Function: | Jump Greater than or Equal to |
| Notes: | If c(r) >= 0, then PC <- EA; Else PC <- PC + 1 |

## 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 | R | A/L | L/R |  | Count |
| 0 5 | 6 7 | 8 | 9 | 1 1  0 1 | 1 1  2 5 |

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Opcode | R | A/L | L/R |  | Count |
| 0 5 | 6 7 | 8 | 9 | 1 1  0 1 | 1 1  2 5 |

|  |  |
| --- | --- |
| Instruction: | SRC r, count, L/R, A/L |
| Octal-Opcode: | 31 |
| Binary-Opcode: | 011111 |
| Function: | Shift Register by Count |
| Notes: | c(r) is shifted left (L/R =1) or right (L/R = 0) either logically (A/L = 1) or arithmetically (A/L = 0)  Count = 0...15; If Count = 0, no shift occurs |

### 4.4.2 (32) RRC

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Opcode | R | A/L | L/R |  | Count |
| 0 5 | 6 7 | 8 | 9 | 1 1  0 1 | 1 1  2 5 |

|  |  |
| --- | --- |
| Instruction: | RRC r, count, L/R, A/L |
| Octal-Opcode: | 32 |
| Binary-Opcode: | 100000 |
| Function: | Rotate Register by Count |
| Notes: | c(r) is rotated left (L/R = 1) or right (L/R =0) either logically (A/L =1)  Count = 0...15 |

## 4.5 Floating Point and Vector Instructions

The instructions for calculation of floating points and vectors.

The binary instruction code format of Floating Point and Vector Instructions is as follows:

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

|  |  |  |
| --- | --- | --- |
| **Opcode:** | 6 bits | Specifies the instruction |
| **FR:** | 2 bis | Specifies the Floating Point 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 |

Floating Point numbers are 16 bits in length, and the format of the Floating Point number is as follows:

|  |  |  |
| --- | --- | --- |
| S | Exponent | Mantissa |
| 0 | 17 | 8 1  5 |

|  |  |  |
| --- | --- | --- |
| **S:** | 1 bit | The sign of the Floating Point number |
| **Exponent:** | 7 bits | Specifies the Exponent |
| **Mantissa:** | 8 bits | Specifies the Mantissa |

### 4.5.1 (33) FADD

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 100001 | FR | IX | I | Address |
| 0 5 | 6 7 | 8 9 | 1  0 | 1 1  1 5 |

|  |  |
| --- | --- |
| Instruction: | FADD fr, x, address[, I] |
| Octal-Opcode: | 33 |
| Binary-Opcode: | 100001 |
| Function: | Floating Point Add Memory to Floating Point Register |
| Notes: | c(fr) <- c(fr) + c(EA)  c(fr) <- c(fr) + c(c(EA)), if I bit set  fr must be 0 or 1. |

### 4.5.2 (34) FSUB

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 100010 | FR | IX | I | Address |
| 0 5 | 6 7 | 8 9 | 1  0 | 1 1  1 5 |

|  |  |
| --- | --- |
| Instruction: | FSUB fr, x, address[, I] |
| Octal-Opcode: | 34 |
| Binary-Opcode: | 100010 |
| Function: | Floating Point Subtract Memory from Floating Point Register |
| Notes: | c(fr) <- c(fr) - c(EA)  c(fr) <- c(fr) - c(c(EA)), if I bit set  fr must be 0 or 1  UNDERFLOW may be set |

### 4.5.3 (35) VADD

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 100011 | FR | IX | I | Address |
| 0 5 | 6 7 | 8 9 | 1  0 | 1 1  1 5 |

|  |  |
| --- | --- |
| Instruction: | VADD fr, x, address[, I] |
| Octal-Opcode: | 35 |
| Binary-Opcode: | 100011 |
| Function: | Vector Add Memory to Floating Point Register |
| Notes: | fr contains the length of the vectors  c(EA) or c(c(EA)), if I bit set, is address of first vector  c(EA+1) or c(c(EA+1)), if I bit set, is address of the second vector  Let V1 be vector at address; Let V2 be vector at address+1  Then, V1[i] = V1[i]+ V2[i], i = 1, c(fr). |

### 4.5.4 (36) VSUB

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 100100 | FR | IX | I | Address |
| 0 5 | 6 7 | 8 9 | 1  0 | 1 1  1 5 |

|  |  |
| --- | --- |
| Instruction: | VSUB fr, x, address[, I] |
| Octal-Opcode: | 36 |
| Binary-Opcode: | 100100 |
| Function: | Vector Subtract Memory to Floating Point Register |
| Notes: | fr contains the length of the vectors  c(EA) or c(c(EA)), if I bit set is address of first vector  c(EA+1) or c(c(EA+1)), if I bit set is address of the second vector  Let V1 be vector at address; Let V2 be vector at address+1  Then, V1[i] = V1[i] - V2[i], i = 1, c(fr) |

### 4.5.5 (37) CNVRT

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 100101 | R | IX | I | Address |
| 0 5 | 6 7 | 8 9 | 1  0 | 1 1  1 5 |

|  |  |
| --- | --- |
| Instruction: | CNVRT r, x, address[, I] |
| Octal-Opcode: | 37 |
| Binary-Opcode: | 100101 |
| Function: | Convert to Fixed/Floating Point |
| Notes: | If F = 0, convert c(EA) to a Fixed Point number and store in r  If F = 1, convert c(EA) to a Floating Point number and store in FR0  The r register contains the value of F before the instruction is executed |

### 4.5.6 (50) LDFR

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 110010 | FR | IX | I | Address |
| 0 5 | 6 7 | 8 9 | 1  0 | 1 1  1 5 |

|  |  |
| --- | --- |
| Instruction: | LDFR fr, x, address [, I] |
| Octal-Opcode: | 50 |
| Binary-Opcode: | 110010 |
| Function: | Load Floating Point Register from Memory |
| Notes: | fr <- c(EA), c(EA+1)  fr <- c(c(EA), c(EA)+1), if I bit set |

### 4.5.7 (51) STFR

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 110011 | FR | IX | I | Address |
| 0 5 | 6 7 | 8 9 | 1  0 | 1 1  1 5 |

|  |  |
| --- | --- |
| Instruction: | STFR fr, x, address [, I] |
| Octal-Opcode: | 51 |
| Binary-Opcode: | 110011 |
| Function: | Store Floating Point Register to Memory |
| Notes: | EA, EA+1 <- c(fr)  c(EA), c(EA)+1 <- c(fr), if I-bit set |

## 4.6 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 | R |  | DevID |
| 0 5 | 6 7 | 81  0 | 1 1  1 5 |

|  |  |  |
| --- | --- | --- |
| **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. |

### 4.6.1 (61) IN

|  |  |  |  |
| --- | --- | --- | --- |
| 111101 | R |  | DevID |
| 0 5 | 6 7 | 81  0 | 1 1  1 5 |

|  |  |
| --- | --- |
| Instruction: | IN r, devid |
| Octal-Opcode: | 61 |
| Binary-Opcode: | 111101 |
| Function: | Input Character to Register from Device |
| Notes: | r = 0...3 |

### 4.6.2 (62) OUT

|  |  |  |  |
| --- | --- | --- | --- |
| 111110 | R |  | DevID |
| 0 5 | 6 7 | 81  0 | 1 1  1 5 |

|  |  |
| --- | --- |
| Instruction: | OUT r, devid |
| Octal-Opcode: | 62 |
| Binary-Opcode: | 111110 |
| Function: | Output Character to Device from Register |
| Notes: | r = 0...3 |

### 4.6.3 (63) CHK

|  |  |  |  |
| --- | --- | --- | --- |
| 111111 | R |  | DevID |
| 0 5 | 6 7 | 81  0 | 1 1  1 5 |

|  |  |
| --- | --- |
| Instruction: | CHK r, devid |
| Octal-Opcode: | 63 |
| Binary-Opcode: | 111111 |
| Function: | Check Device Status to Register |
| Notes: | r = 0...3  c(r) <- device status |

## 4.7 Other Instructions

### 4.7.1 (00) HALT

|  |  |
| --- | --- |
| 000000 |  |
| 0 5 | 6 1  5 |

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

### 4.7.2 (30) TRAP

|  |  |  |
| --- | --- | --- |
| 011110 |  | Trap Code |
| 0 5 | 6 1  1 | 1 1  2 5 |

|  |  |
| --- | --- |
| Instruction: | TRAP code |
| Octal-Opcode: | 30 |
| Binary-Opcode: | 011110 |
| Function: | Traps to memory address 0, which contains the address of a table in memory. Stores the PC+1 in memory location 2. |
| Notes: | The table can have a maximum of 16 entries representing 16 routines for user-specified instructions stored elsewhere in memory. Trap code contains an index into the table, e.g. it takes values 0 – 15.  When a TRAP instruction is executed, it goes to the routine whose address is in memory location 0, executes those instructions, and returns to the instruction stored in memory location 2. The PC+1 of the TRAP instruction is stored in memory location 2. |