CISC SIMULATOR

Manual

V 2.1

**Group 8**

Md Shahjalal

Tianyou Bao

Xuzheng Lu

Contents

[1 Introduction 1](#_Toc21811519)

[1.1 Debugging Panel 1](#_Toc21811520)

[1.1.1 Register Indicators Area 1](#_Toc21811521)

[1.1.2 Memory Area 2](#_Toc21811522)

[1.1.3 Controller Area 2](#_Toc21811523)

[1.2 Operation Panel (Console) 3](#_Toc21811524)

[1.3 Classic Panel 4](#_Toc21811525)

[1.4 Themes 4](#_Toc21811526)

[2 Basic Operations 5](#_Toc21811527)

[2.1 Writing Values to Registers 5](#_Toc21811528)

[2.2 Writing Values to Memory 5](#_Toc21811529)

[2.2.1 Using Memory Address Register and Memory Buffer Register 6](#_Toc21811530)

[2.2.2 Modifying the Memory Area 6](#_Toc21811531)

[2.3 Executing Instructions 7](#_Toc21811532)

[2.3.1 Executing Instructions Step-by-Step 7](#_Toc21811533)

[2.3.1 Executing Instructions Automatically 8](#_Toc21811534)

[3 Executing Programs 10](#_Toc21811535)

[3.1 Executing Program 1 10](#_Toc21811536)

[3.1.1 Program Description 10](#_Toc21811537)

[3.1.2 Running the Program 10](#_Toc21811538)

[3.2 Executing Program 2 11](#_Toc21811539)

[3.3 Executing a Custom Program Using IPL 11](#_Toc21811540)

[3.3.1 Using Operation Panel (Console) 11](#_Toc21811541)

[3.3.2 Using Debugging Panel 12](#_Toc21811542)

[4 Instructions Reference 12](#_Toc21811543)

[4.1 Load/Store Instructions 12](#_Toc21811544)

[4.1.1 (01) LDR 12](#_Toc21811545)

[4.1.2 (02) STR 12](#_Toc21811546)

[4.1.3 (03) LDA 13](#_Toc21811547)

[4.1.4 (41) LDX 13](#_Toc21811548)

[4.1.5 (42) STX 13](#_Toc21811549)

[4.2 Arithmetic and Logical Instructions 13](#_Toc21811550)

[4.2.1 (04) AMR 14](#_Toc21811551)

[4.2.2 (05) SMR 14](#_Toc21811552)

[4.2.3 (06) AIR 14](#_Toc21811553)

[4.2.4 (07) SIR 15](#_Toc21811554)

[4.2.5 (20) MLT 15](#_Toc21811555)

[4.2.6 (21) DVD 15](#_Toc21811556)

[4.2.7 (22) TRR 15](#_Toc21811557)

[4.2.8 (23) AND 16](#_Toc21811558)

[4.2.9 (24) ORR 16](#_Toc21811559)

[4.2.10 (25) NOT 16](#_Toc21811560)

[4.3 Transfer Instructions 16](#_Toc21811561)

[4.3.1 (10) JZ 17](#_Toc21811562)

[4.3.2 (11) JNE 17](#_Toc21811563)

[4.3.3 (12) JCC 17](#_Toc21811564)

[4.3.4 (13) JMA 17](#_Toc21811565)

[4.3.5 (14) JSR 18](#_Toc21811566)

[4.3.6 (15) RFS 18](#_Toc21811567)

[4.3.7 (16) SOB 18](#_Toc21811568)

[4.3.8 (17) JGE 18](#_Toc21811569)

[4.4 Shift/Rotate Instructions 19](#_Toc21811570)

[4.4.1 (31) SRC 19](#_Toc21811571)

[4.4.2 (32) RRC 19](#_Toc21811572)

[4.5 I/O Instructions 19](#_Toc21811573)

[4.5.1 (61) IN 20](#_Toc21811574)

[4.5.2 (62) OUT 20](#_Toc21811575)

[4.5.3 (63) CHK 20](#_Toc21811576)

[4.6 Other Instructions 21](#_Toc21811577)

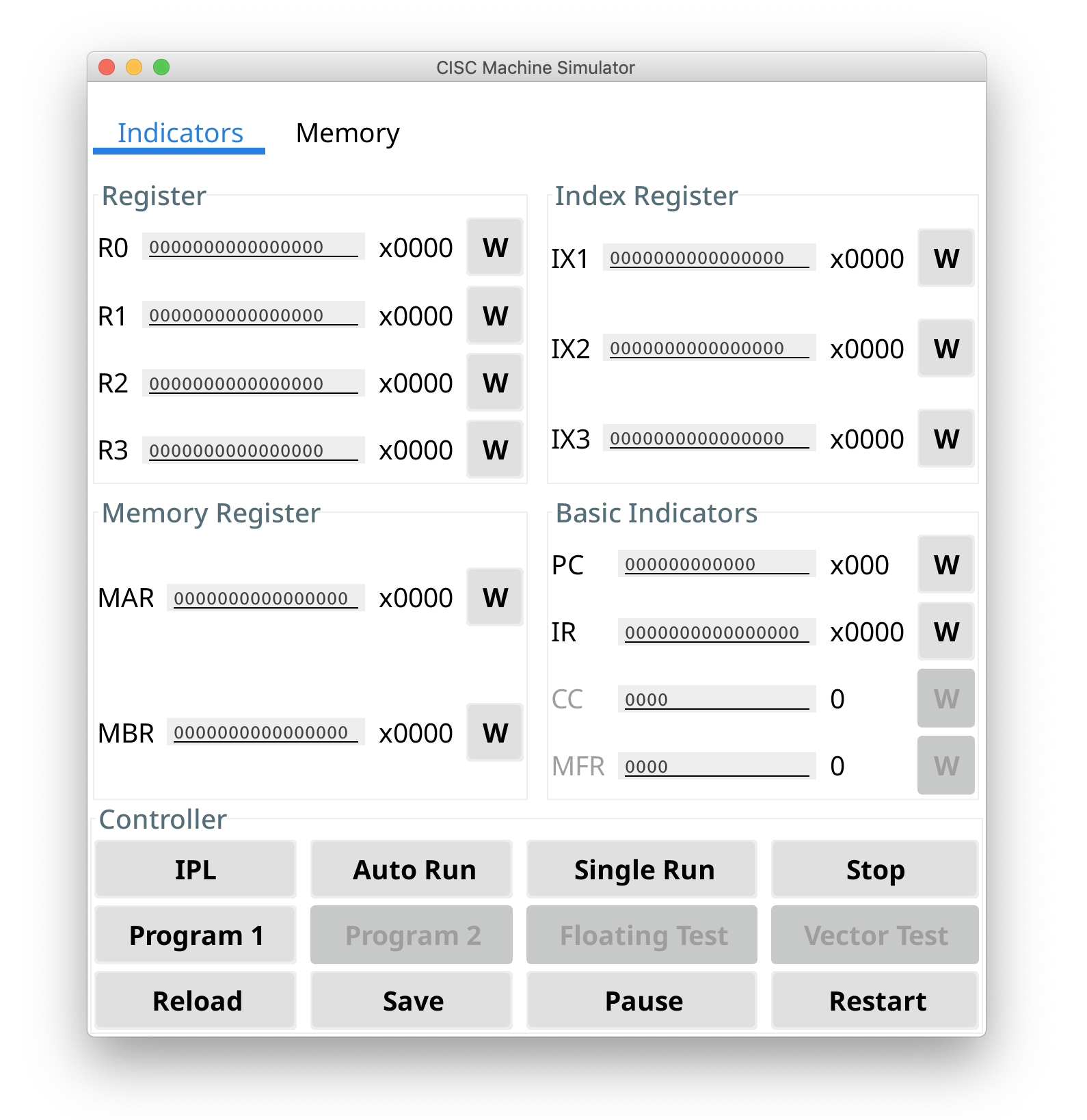
[4.6.1 (00) HALT 21](#_Toc21811578)

# 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 |
| 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.

#### 

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

#### 

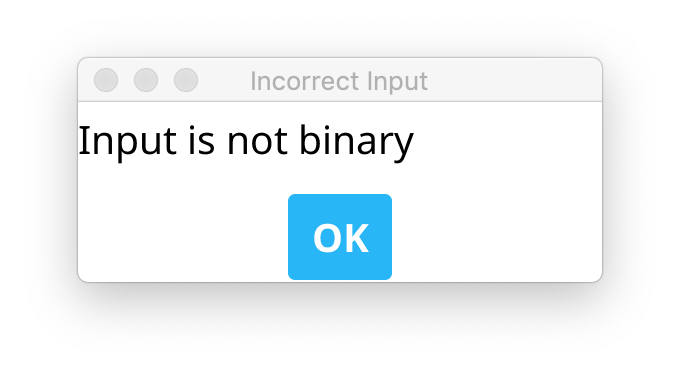
**Step 3**: Done! The value will be written to the Register.

**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.

#### 

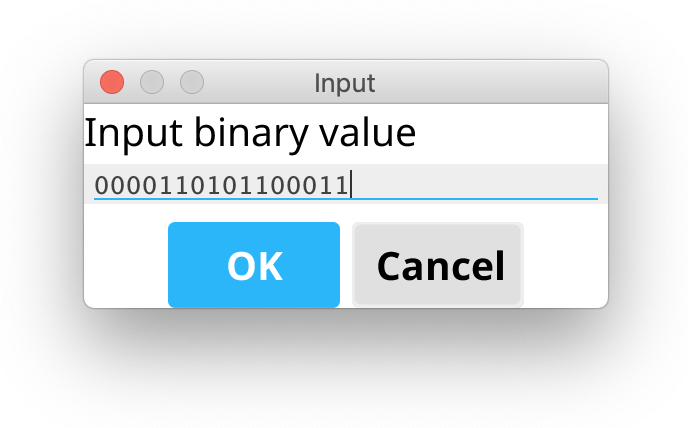
#### 

### 2.2.2 Modifying the Memory Area

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

#### 

**Step 2**: An 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 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.

#### 

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

#### 

## 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  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 |

### 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 |

### 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 |

### 4.1.4 (41) LDX

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 101001 | R | 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 |

### 4.1.5 (42) STX

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 101010 | R | 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 |

## 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 |

### 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 |

### 4.2.3 (06) AIR

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 000110 | R |  |  | Address |
| 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 |

### 4.2.4 (07) SIR

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 000111 | R |  |  | Address |
| 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 |

### 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 |

### 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 |

### 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 |

### 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 |

### 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 |

### 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 |

## 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 |

### 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 |

### 4.3.3 (12) JCC

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 001100 | R | 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 |

### 4.3.4 (13) JMA

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 001101 | R | 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 |

### 4.3.5 (14) JSR

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 001110 | R | 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 |

### 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 |

### 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 |

### 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 |

## 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 |

### 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 |

## 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 | 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. |

### 4.5.1 (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 |

### 4.5.2 (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 |

### 4.5.3 (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 |

## 4.6 Other Instructions

### 4.6.1 (00) HALT

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

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