CISC SIMULATOR

Manual

V 2.0

**Group 8**

Md Shahjalal

Tianyou Bao

Xuzheng Lu

Contents

[1 Introduction 1](#_Toc21533756)

[1.1 Debugging Panel 1](#_Toc21533757)

[1.1.1 Register Indicators Area 1](#_Toc21533758)

[1.1.2 Memory Area 2](#_Toc21533759)

[1.1.3 Controller Area 2](#_Toc21533760)

[2.2 Classic Panel 3](#_Toc21533761)

[2 Operation 3](#_Toc21533762)

[2.1 Writing Values to Registers 3](#_Toc21533763)

[2.2 Writing Values to Memory 3](#_Toc21533764)

[2.2.1 Using Memory Address Register and Memory Buffer Register 3](#_Toc21533765)

[2.2.2 Modifying the Memory Area 4](#_Toc21533766)

[2.3 Executing Instructions 4](#_Toc21533767)

[2.3.1 Executing Instructions Step-by-Step 4](#_Toc21533768)

[2.3.1 Executing Instructions Automatically 5](#_Toc21533769)

[3 Instructions Reference 6](#_Toc21533770)

[3.1 Load/Store Instructions 6](#_Toc21533771)

[3.1.1 LDR 6](#_Toc21533772)

[3.1.2 STR 7](#_Toc21533773)

[3.1.3 LDA 7](#_Toc21533774)

[3.1.4 LDX 7](#_Toc21533775)

[3.1.5 STX 7](#_Toc21533776)

[3.2 Arithmetic and Logical Instructions 8](#_Toc21533777)

[3.2.1 AMR 8](#_Toc21533778)

[3.2.2 SMR 8](#_Toc21533779)

[3.2.3 AIR 9](#_Toc21533780)

[3.2.4 SIR 9](#_Toc21533781)

[3.2.5 MLT 9](#_Toc21533782)

[3.2.6 DVD 9](#_Toc21533783)

[3.2.7 TRR 10](#_Toc21533784)

[3.2.8 AND 10](#_Toc21533785)

[3.2.9 ORR 10](#_Toc21533786)

[3.2.10 NOT 10](#_Toc21533787)

[3.3 Transfer Instructions 11](#_Toc21533788)

[3.3.1 JZ 11](#_Toc21533789)

[3.3.2 JNE 11](#_Toc21533790)

[3.3.3 JCC 11](#_Toc21533791)

[3.3.4 JMA 12](#_Toc21533792)

[3.3.5 JSR 12](#_Toc21533793)

[3.3.6 RFS 12](#_Toc21533794)

[3.3.7 SOB 12](#_Toc21533795)

[3.3.8 JGE 13](#_Toc21533796)

[3.4 Shift/Rotate Instructions 13](#_Toc21533797)

[3.4.1 SRC 13](#_Toc21533798)

[3.4.2 RRC 14](#_Toc21533799)

[3.5 I/O Instructions 14](#_Toc21533800)

[3.5.1 IN 14](#_Toc21533801)

[3.5.2 OUT 14](#_Toc21533802)

[3.5.3 CHK 15](#_Toc21533803)

[3.6 Other Instructions 15](#_Toc21533804)

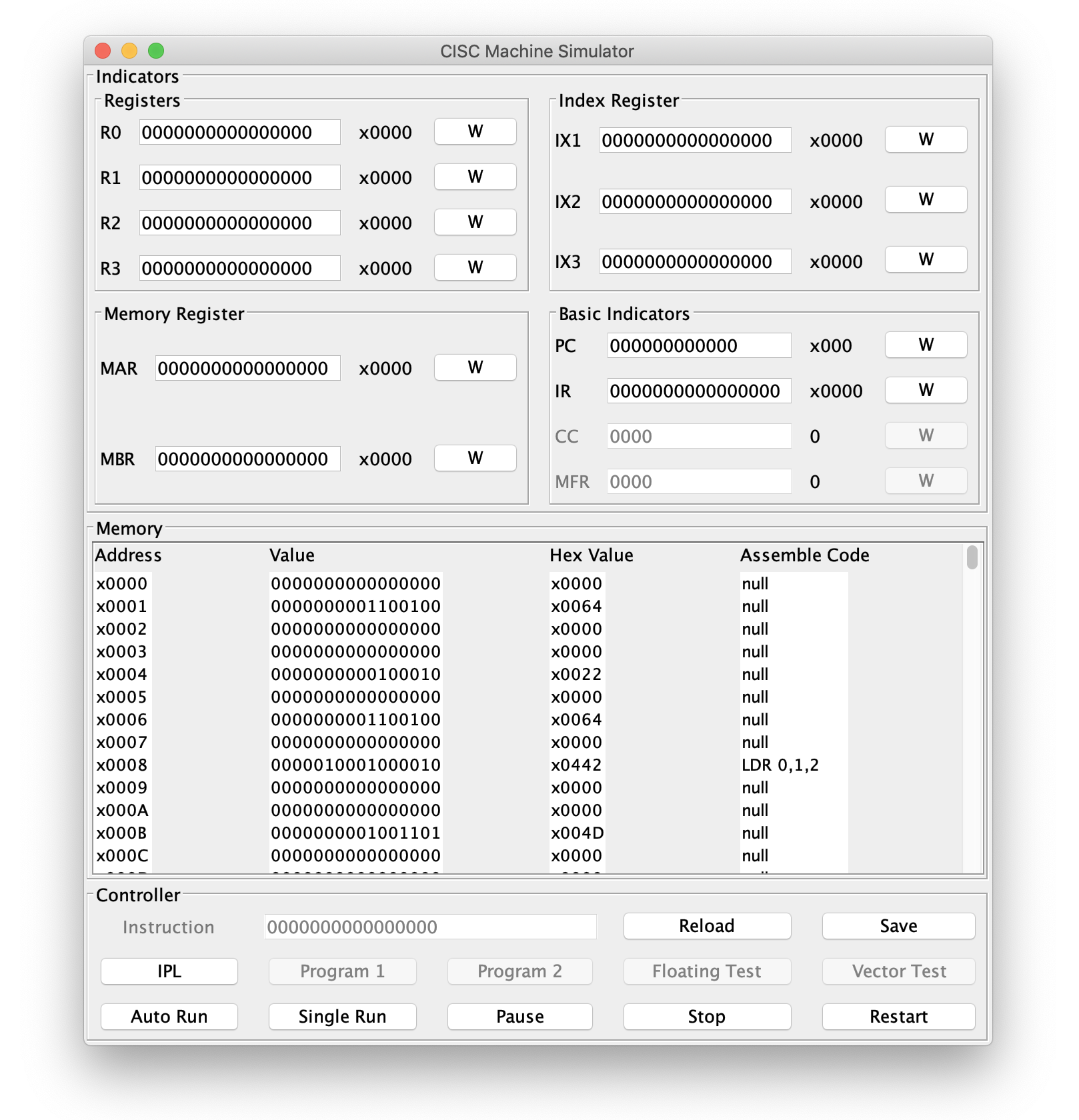
[3.6.1 HALT 15](#_Toc21533805)

# 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.

#### 

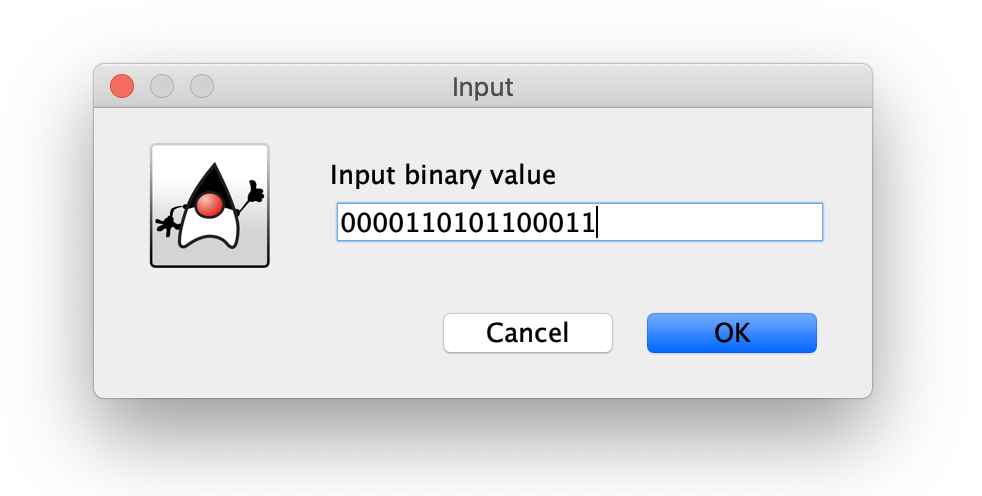
#### 

### 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 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 |

### 3.1.1 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 |

### 3.1.2 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 |

### 3.1.3 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 |

### 3.1.4 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 |

### 3.1.5 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 |

## 3.2 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 |

### 3.2.1 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 |

### 3.2.2 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 |

### 3.2.3 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 |

### 3.2.4 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 |

### 3.2.5 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 |

### 3.2.6 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 |

### 3.2.7 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 |

### 3.2.8 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 |

### 3.2.9 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 |

### 3.2.10 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 |

## 3.3 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 |

### 3.3.1 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 |

### 3.3.2 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 |

### 3.3.3 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 |

### 3.3.4 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 |

### 3.3.5 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 |

### 3.3.6 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 |

### 3.3.7 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 |

### 3.3.8 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 |

## 3.4 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 |

### 3.4.1 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 |

### 3.4.2 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 |

## 3.5 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. |

### 3.5.1 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 |

### 3.5.2 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 |

### 3.5.3 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 |

## 3.6 Other Instructions

### 3.6.1 HALT

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

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