Manuel Pérez González, Mashaal Al Mazro’ei, Pedro Rios

[Email address]

Abstract

The CISC Simulator is a project to build a basic Computer Simulator that demonstrates the structure of a computer system, executes instructions and operations.

CSCI 6461 Semester project

CISC Computer Simulator - System Documentation

# Overview

The CISC Simulator is a project to build a basic Computer Simulator that demonstrates the structure of a computer system, executes instructions and operations. The system emphasizes the Instruction Set Architecture (ISA). The project consists of four fundamental components which are the CPU, Memory, Operators Console and the Engineering Console.

The below documentation is the deliverable of Phase I.

# Scope

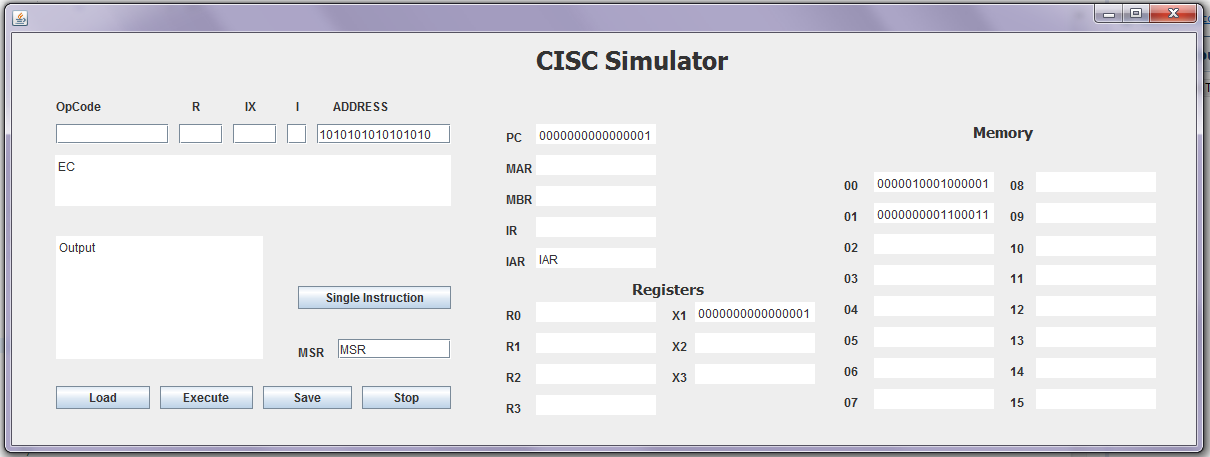
The project is going to be gradually built on four phases starting with the Basic Machine where the Front Panel of the Simulator is designed and implemented along with the Load/Store instructions. Furthermore, in the next phase, the Memory and Cache Operations are going to be implemented. On the following phase, all of the executable instructions are going to be implemented.

# Tools

The Front Panel was initially prototyped using Inkscape (a free and open-source editor to create illustrations and diagrams). The prototype was then developed on Eclipse IDE using Java Swing (an API for providing a graphical user interface (GUI) for Java programs). The LDR/STR instructions were also implemented using Java.

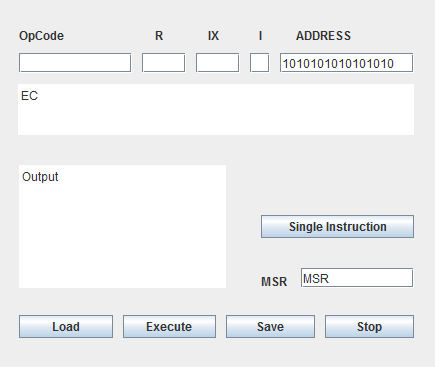
# A Tour of the User Interface

The user interface consists of four main areas; Operator’s Console CPU and Memory. Below is an overview of the Front Panel of the Simulator.



# Operators Console

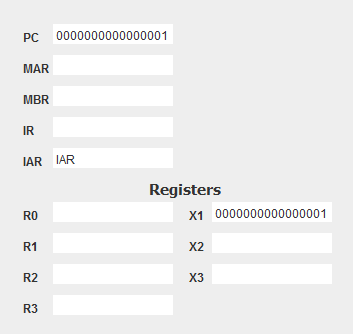
The operators console is the control area for the user to input data, run the programs and display the output. It includes the following components:



* **OPCode Area:** to enter the OPCode to be executed and will display the output.
* **Engineering Console:** o display the status of the simulator during execution.
* **Output Area:** will display the output of the executed program.
* **Single Instruction:** to execute the instruction at the PC address.
* **MSR:** will display the status of the health of the machine.
* **Load Switch:** will load the registers with instructions from the memory using LDR.
* **Execute Switch:** to execute the programs to be implemented. The program is first selected and loaded using the **Load** button, the **Execute** button will run the program.
* **Save Switch:**  store the state of the simulation on a file.
* **Stop Switch**:will Terminate/Halt the program execution.

# Central Processor

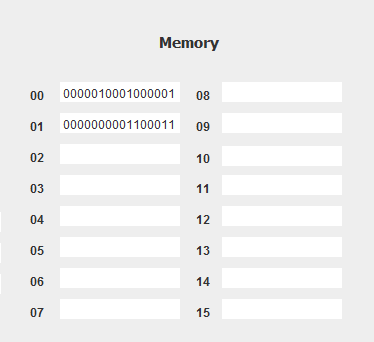
The CPU area contains the Registers and Indexes.



* PC**:** displays the current instruction that is being executed.
* MAR**:** displays the memory address of the data to be fetched.
* MBR**:** displays the data fetched from and to the memory.
* IR**:** displays the currently executing register.
* IAR**:** displays the address of the current executing register.
* Registers R0-R2**:** will be used as General Purpose Registers
* Index Registers X1-X3**:** Registers

# Memory

The memory area contains 16 memory locations.



## Simulator Operation Part I

1. When the simulator runs the simulator will load a hard coded into the PC and one register (memory). (*This was because there are many parts are still not implemented and because of the limited scope of this part*).
2. The **Load** instruction will be executed by clicking the ‘***Single Instruction***’ button.
3. The **Store** instruction will be executed by clicking the ‘***Single Instruction***’ button.

Below is the series of steps that takes place while the simulation is running.

When the Source code is executed, the following operation will run:

1. The data from Memory 01 and 02 will be fetched and loaded into the MBR and IR
2. The data from the PC will also be fetched and loaded into the MAR.
3. The data from MBR and IR will be stored into the Memory.

These steps (for this phase) occur when the button ‘Single instruction’ is used to execute the instructions. They also reflect a sequence of steps that can be considered a typical case.

These steps are executed depending on the instruction being processed as in many cases the data in the MBR or IR register does not go directly to Memory but it is used for intermediate steps and thus is saves in one of the general registers (R1-R3) for example.