Project 2: CSCI 6461 Machine Simulator

Team 17:

Abhi Bhardwaj, Esha Arun Angadi

Note: This Programs was built and tested on OpenJDK 22.0.2 version

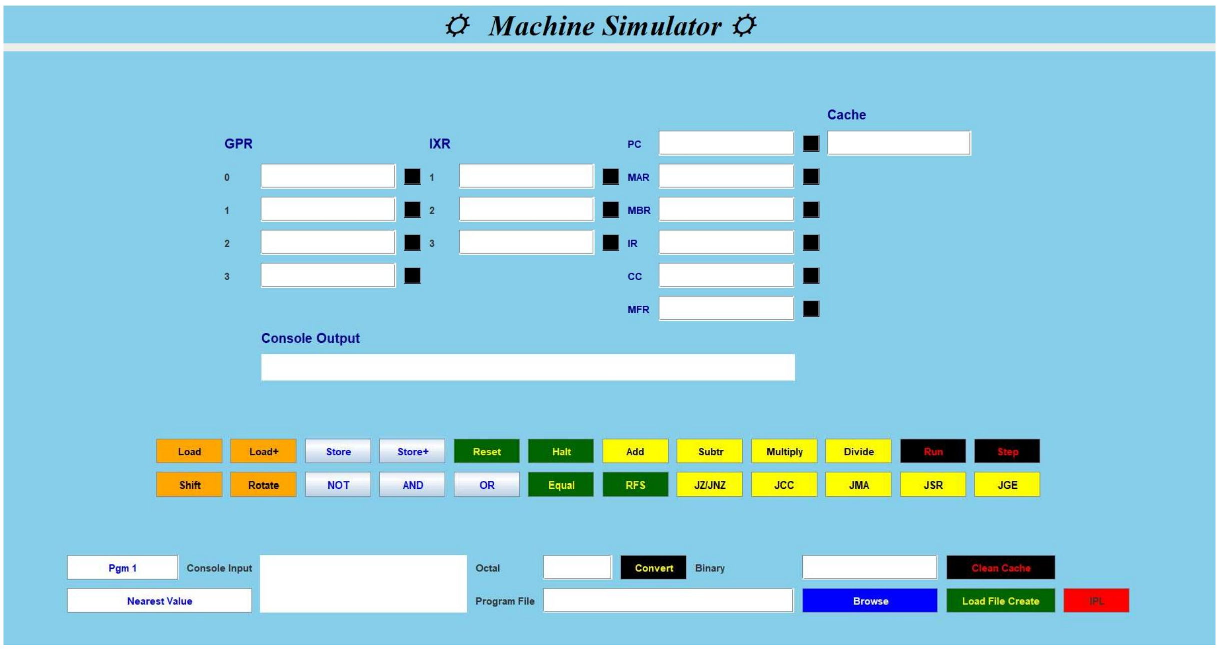
1. Introduction

The basic elements of computer system architecture are simulated by the CSA Simulator, a Java application. It mimics the CPU, memory, input/output devices, and number converters. The GUI of the system, which was developed using Java Swing, offers an easy way of handling memory and interacting with the CPU, hence making it easily accessible to the users.

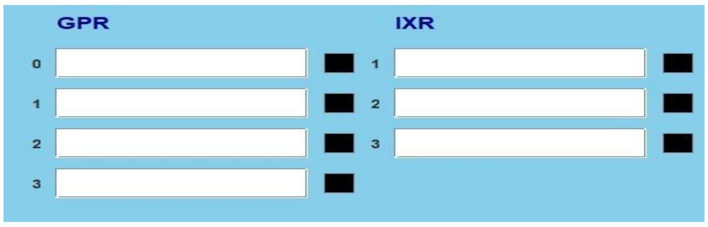
This documentation provides an overview of the project structure, functionality, and how to use Machine Simulator and how to run program2: Word search in given input sentences.

2. System Components

GUI (Graphical User Interface)



- Swing-based Java interface: The GUI makes use of Swing components, i.e., JLabel and JButton, to display the system memory, control buttons, and registers.
- Key Components-
  - **GPR and IXRs**: Display important registers such as General-Purpose Registers (GPRs), Index Registers (IXRs) both are 16 bits. On hitting black button loads the value in register.



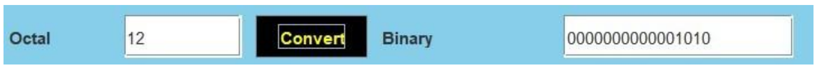
- Load, Load+, Store, Store+:

  - **Load**: Gets the value from the memory address that the MAR indicates and puts it in the MBR. The load operation must be started by the user entering a binary value into the MAR.
  - **Load+**: Increases the MAR by 1 after performing a load operation.
  - **Store**: Moves the value from the MBR to the memory address that the MAR specifies.
  - **Store+**: Increases the MAR by 1 after performing a store operation.

- Reset and Halt: **Reset** will reset the Simulator and initialize all values with **0**. On pressing **Halt** will stop the execution.

- Run and Step: **Run** will execute the input Load file and **Step** will execute the instruction one by one.

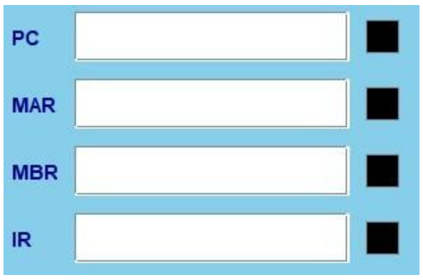
- Octal and Binary: Given an any Octal number on pressing convert button will convert it into 16 bits binary number.



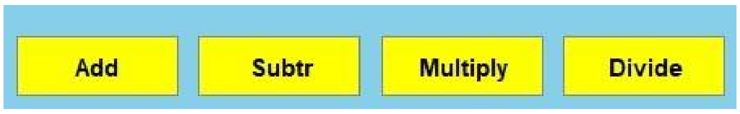
- Program File: This will take input file(.txt) from system using **Browse** button.

- **IPL**: It will initialize the memory with content according to given input load file.

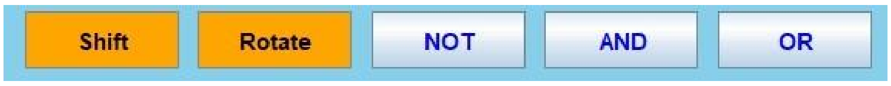
- **PC** (12 bits): Contains the address of the next instruction.
- **MAR** (12 bits): Holds the memory location.
- **MBR** (16 bits): Stores the content at the memory location specified by the MAR.
- **IR** (16 bits): Displays the instruction pointed to by the PC.



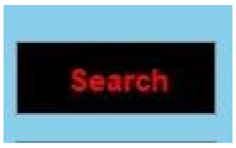
- **Add, Subtract, Multiply, Divide:** This will perform targeted operation as per input given in Console Input.



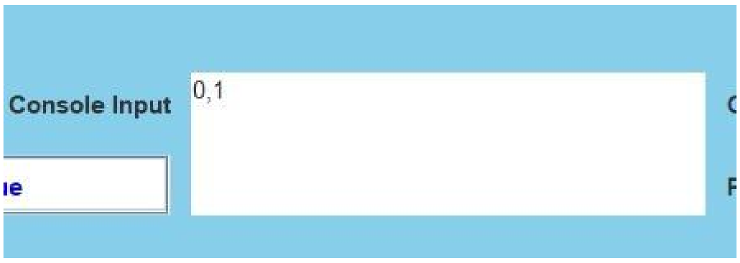
- **Shift, Rotate, NOT, AND, OR:** This will perform resultant operation as per input given register provided on "Console Input".



- **Search:** This button helps in searching the word in six sentence user enters.

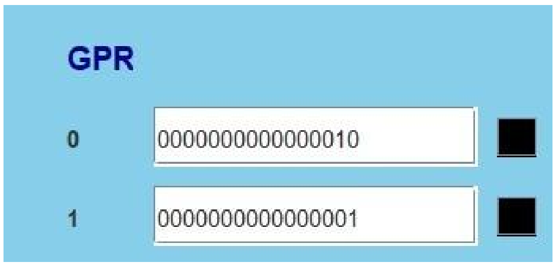


E.g. Working of OR operation:
1. We provide register number in console Input: **Register 0, 1**



0,1

2. Load the following register number which we provided in "**Console Input Section"**



0 0000000000000010
1 0000000000000001

3. On hitting ***OR button*** the OR operation will be performed on GPR0, GPR1 content and result gets stored in GPR0. Which is 3 in our case (11 - Binary)

3. Memory Management:

- 2KB Memory Array: The memory system included with the simulator is represented by a brief array of 2048 cells (2KB), all of which are initially set to zero. A cell can hold 16 bits of information.
- Fixed Memory Size: Even though the size is now restricted to 2KB, future versions may provide more flexibility by enabling dynamic memory allocation.

4. Number Conversion Utility:

- The Converter Class: This utility offers crucial functions for converting between binary, decimal, and hexadecimal formats to ease CPU operations.
  - Binary data can be converted to its decimal representation using the BinaryToDecimal() function.
  - The DecimalToBinary() function can be used to translate a decimal value into binary representation.
  - These conversion methods are frequently used by the system, especially when handling memory locations and CPU instructions.

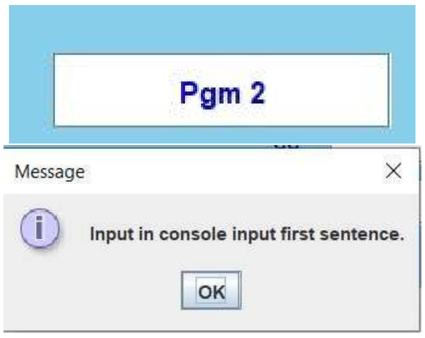
5. Run the panel:

- To begin the process, press the "Run" button. The program counter (PC) will advance to the following address, the memory address register (MAR) will save the current address, and the memory buffer register (MBR) will display the pertinent data.
- The machine will continue to execute each instruction and load data into the Instruction Register (IR) until it encounters an operation with opcode 0000 (the HLT instruction).

Note: The load.txt file is used to load all addresses and values with help of clicking IPL button. 0000 is the default address if it is not present in the load.txt. The machine is able to access addresses between 0 and 2048.

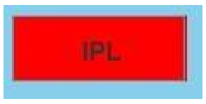
6. Program Execution Step:

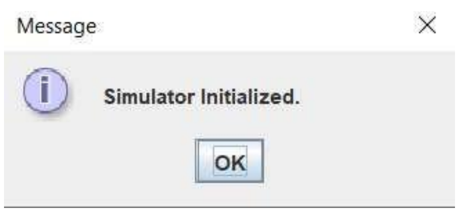
1. Click the PGM2 button, and it will ask you enter sentence in "***Console Input"*** *section*.



Input in console input first sentence.

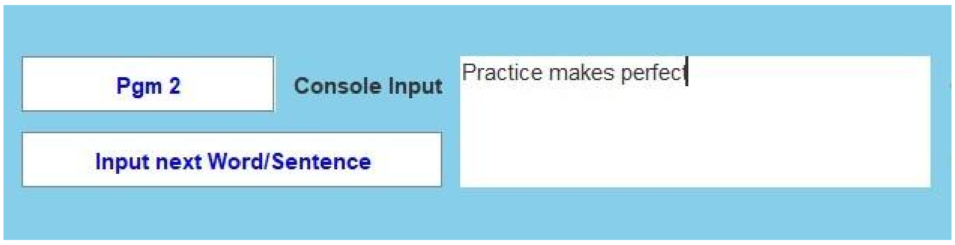
2. Hit ***IPL*** for initializing Simulator





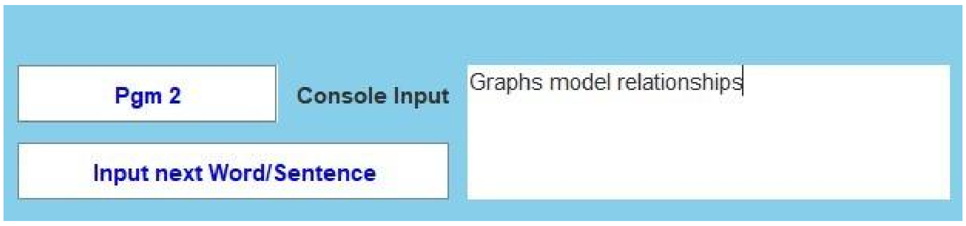
Simulator Initialized.

3. Now one by one enter all six sentences in ***Console Input***. After writing each sentence in console hit ***Input next Word/Sentence***.
1st sentence: Practice makes perfect



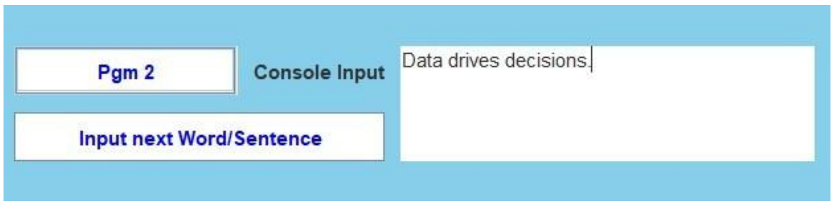
Practice makes perfect

2nd sentence: Graphs model relationships.



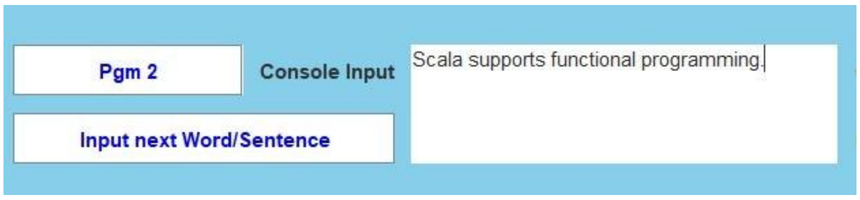
Graphs model relationships

3rd sentence: Data drives decisions.



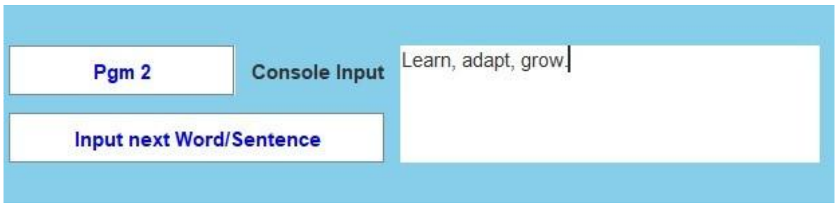
Data drives decisions.

4th sentence: Scala supports functional programming.



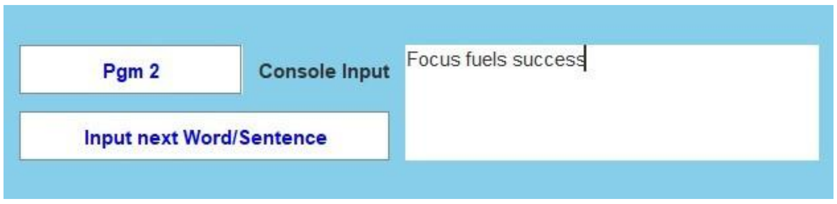
Scala supports functional programming.

5th sentence: Learn, adapt, grow.



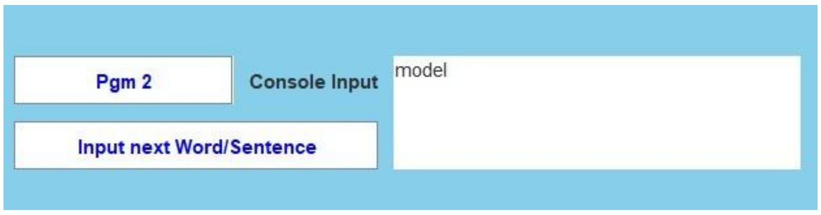
Learn, adapt, grow.

6th sentence: Focus fuels success.



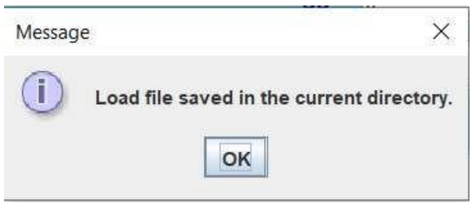
Focus fuels success

4. Once all six sentences have been entered, the user must enter the word in the input console and then hit Input next word/sentence one last time, as indicated below, in order to locate the word in all six sentences.



model

5. Then click on ***Search*** button in order to search the word provided by user.


Message

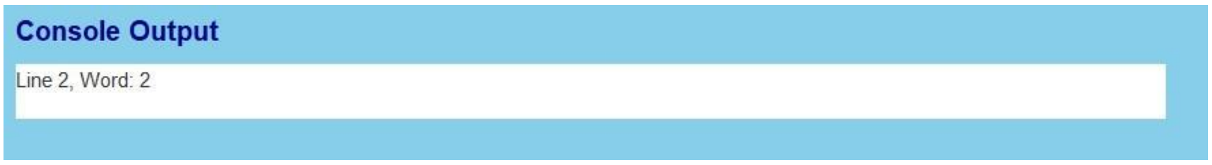
Load file saved in the current directory.

6. If a word is present in a sentence, it will be displayed in the ***Console Output*** along with the line number and its position within that line.

For e.g.: Given sentences in our case:

1. Practice makes perfect.
2. Graphs ***model*** relationships.
3. Data drives decisions.
4. Scala supports functional programming.
5. Learn, adapt, grow.
6. Focus fuels success.

We searched for the word **"*model*"** which is present in **line 2 at position 2**


Console Output

Line 2, Word: 2

7. Conclusion

In summary, we have effectively created the front panel of the Machine Simulator with all the components and operations needed. These include CPU control and core memory functioning together with an easy-to-use graphical user interface. This front panel, being the primary user interface, makes the simulator readily accessible and greatly enriches the user experience. It also contains a new feature of the tool "Search" which searches for the word in user sentences.