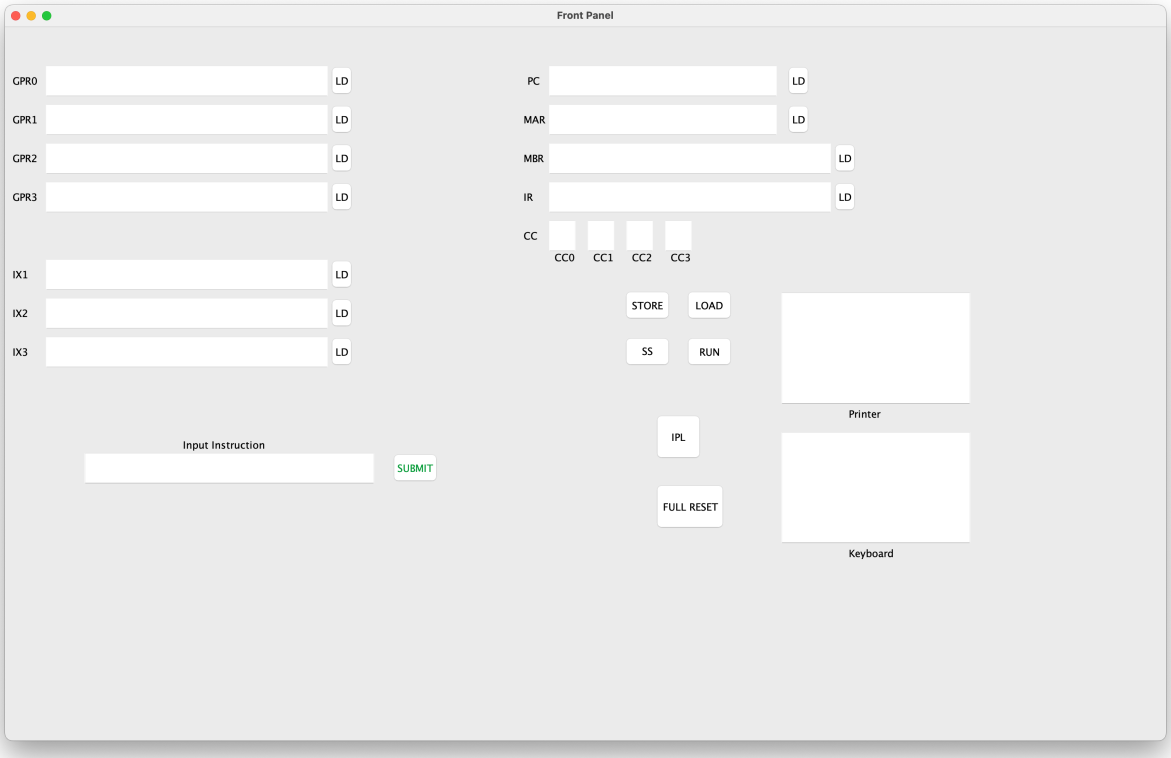
CISC SIMULATOR DESIGN DOCUMENT

Nikhil Reddy Kodumuru
Poojasri Mothukuri
Prudhvi Lakshmi Sesha Sai Cheedella
Sai Nikhil Varada

Design Layout:



The Assembler:
An assembler is a program that translates assembly language code into machine code or an intermediate code.

- The code for the assembler is inside the class named “Assembler”.
- The class has two ArrayList objects called “inputRows” and “outputData” which are used to store the data read from the input file and the resulting output data respectively.
- The “readInit” method is responsible for executing the entire assembly process. It calls readInit to read input data and then calls encodeInstructions to process the instructions and generate the machine code. If the input is valid (isValid), it writes the output to a file named "instructions.txt."
- “generateInstruction” method is responsible for generating the machine instruction for a given assembly instruction (row1 and row2). It interprets different assembly instructions and converts them into machine instructions.

- “encodeInstructions” method traverses he inputRows and processes each instruction. For each instruction, it generates the corresponding machine instruction using the generateInstruction method. The resulting machine instructions are stored in the outputData list.
- There are several utility methods like “octalBinary”,” binaryToHex”, “genLeftPaddedString” are defined to perform tasks such as converting numbers between different bases and generating left-padded strings.
- Instruction set code includes assembly instructions (**LDX**, **STX**, **LDR**, **STR**, etc.), and it generates the corresponding machine instruction for each instruction.
- The program uses Java's file I/O classes (**File**, **FileReader**, **BufferedReader**, **FileWriter**) to read from and write to files.

UI:

- Import Statements:
  - There are many important import statements that include necessary Java Swing and AWT classes for creating GUI components.
  - “ActionEvent”,” ActionListener” and “keyEvent” are imported for handling user actions and keyboard events.
  - Hashmaps and maps are imported for working with key-value pairs.
  - Pattern is imported for working with regular expressions.
  - “JTextField”, “PlainDocument”, and “AttributeSet” are imported for handling text fields and documents in Swing.
- JTextFieldLimit Class:
  - This is a custom class that extends “PlainDocument”. It limits the number of characters that can be entered into a “JTextField”.
- ActionListeners
  - These are attached to various buttons like submit, ldPC, fullReset, etc.
  - These listeners respond to user actions such as button clicks and execute specific actions like processing input instructions or resetting the UI.
- Keylistener for Keyboard Input:
  - A keylistener is attached to the keyboard JTextField to handle keyboard input events.
- Full Reset:
  - This method is responsible for resetting both the UI and presumably the CPU values to their initial states.
- CleareUI:
  - The clearUI method clears the text fields in the UI.
- UpdateUI:
  - This method updates the Ui based on the values obtained from the simulator.
- Reset Input Instruction method:
  - This method resets the input instruction and validation message in the UI.

File Data:

- It is a custom format to handle file data.

Instructions –

- These are the instructions that are given to the simulator as input.

Simulator:

- UI Integration:
    - The simulator integrates with the user interface (UI) component for displaying and interacting with the simulated system.
- Register and Memory Simulation:
    - It simulates various registers (General Purpose Registers - GPR0 to GPR3, Index Registers - IX1 to IX3, and others like PC, MAR, MBR, IR) and a memory array (2048 length) to mimic a computer's memory management.
- Initial Program. Load (IPL) and Assembler Integration:
    - The loadIPL method indicates an Initial Program Load functionality, reading instructions from a file and loading them into memory. This suggests an integration with an assembler component.
- Instruction Execution:
    - The program includes an executeInstruction method for fetching, decoding, and executing instructions from memory, simulating a CPU's instruction cycle.
- Load and Store Operations:
    - Methods like loadGPR0, loadIX1, store, etc., simulate the loading and storing of data to and from registers and memory.
- Utility Methods:
    - Includes utility methods like reverseArray for reversing an array and genBinaryPaddedString for generating binary strings with padding.
- Keyboard Input Handling:
    - KeyBoardKeyPressed and KeyBoardKeyReleased handle keyboard events, suggesting the simulator accepts input from the keyboard.
- Keyboard Input Handling:
    - Methods KeyBoardKeyPressed and KeyBoardKeyReleased handle keyboard events, suggesting the simulator accepts input from the keyboard.
- Condition Code Registers:
    - It simulates condition code registers (CC0 to CC3) for various states like overflow, underflow, etc.
- Error Checking:
    - Includes methods for checking memory bounds and handling exceptions.
- Main Method:
    - The entry point of the program, where it initializes the UI.
- Data and Control Flow:
    - The simulator handles data and control flow according to the instructions, simulating a computer's operational cycle.