CSCI-6461

Computer System Architecture

PROJECT 1 - Simulator

Designed by – TEAM 5

Avish Kaushik

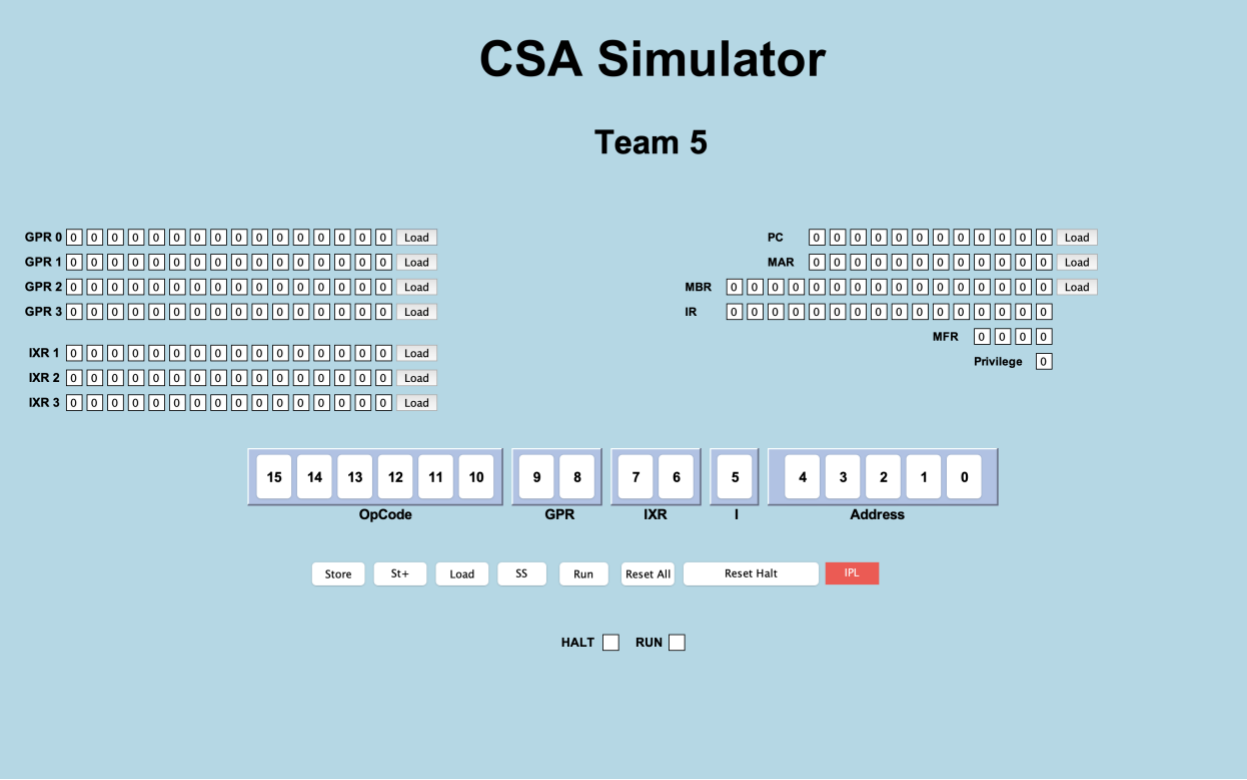
Harichandana Samudrala

GITHUB URL - https://github.com/AvishKaushik/Simulator.git

Design Notes for the Simulator

1. Overview of the Project

The CSA Simulator is a tool that emulates the key components of a computer system architecture, including the CPU, memory, input/output devices, and conversion utilities. The system is implemented in Java with a graphical user interface (GUI) using Swing to provide users with interactive controls for simulating CPU operations, memory management, and device interaction.



---

2. System Components

Graphical User Interface (GUI)

- **Java Swing-based GUI**: The GUI is created using Swing components such as JLabel and JButton to visually represent registers, memory, and control buttons.

- **Key GUI Components**:
    - **Registers and Labels**: General Purpose Registers (GPRs), Index Registers (IXRs), and the Machine Fault Register (MFR) are displayed using JLabel components.
    - **Control Buttons**: Buttons like "Store," "Load," "Reset," and "Run" allow users to interact with the system, controlling data storage, code execution, and system resets.
    - **Event Handling**: Each button has an associated event listener, which calls methods like Store(), LoadValue(), and RunProg() to execute corresponding actions.
- **Threading for Responsiveness**: The "Run" button uses a background SwingWorker thread to ensure the GUI remains responsive during long-running processes, such as executing a program.
- **Fixed Layout**: The GUI layout is manually set using setBounds() to position elements, resulting in a static layout. This could pose challenges for resizing or adapting to different screen resolutions.

Memory Management (Memory.java)

- **2KB Memory Array**: The memory is simulated using a short[] array with a fixed size of 2048 cells (2KB). Each cell is initialized to zero at the start.
- **Fixed Memory Size**: The memory size is currently hardcoded, which may limit flexibility for future expansion. A constructor could be added to make the memory size configurable.

Devices Interface (Devices.java)

- **Console Input/Output Simulation**: The device interface mimics a console using JTextArea components:
    - **ConsoleOut**: A simulated console output (printer), styled with a monospaced font (Courier New) and a green-on-black color scheme.
    - **ConsoleIn**: A single-line input area representing the console keyboard.
    - **Panels**: Panels are used to organize the components, each with a titled border for clarity.

- **Manual Layout**: The device panels use setBounds() for manual positioning, making the GUI non-responsive to window resizing.

Number Conversion Utility (Converter.java)

- **Binary, Decimal, and Hex Conversions**: The Converter class provides utility functions to convert numbers between binary, decimal, and hexadecimal formats, a key feature for simulating CPU operations.
    - **BinaryToDecimal()**: Converts a binary array to its decimal equivalent.
    - **DecimalToBinary()**: Converts a decimal number to its binary representation.
    - **HexToDecimal()**: Converts a hexadecimal string to a decimal number.
- **Reusability**: These conversion functions are designed to be reusable across different parts of the system, particularly in the CPU and memory handling processes.

CPU Simulation (CPU.java)

- **Register Management**: The CPU class manages a set of registers, including General Purpose Registers (GPRs), the Program Counter (PC), Instruction Register (IR), and more.
- **Instruction Execution**: The Execute(Memory m) method processes instructions by decoding binary opcodes and executing corresponding operations, such as loading and storing data, halting the program, and addressing operations.
    - **Opcode Handling**: A switch-case structure is used to handle various opcodes like LDR (load register), STR (store register), and HLT (halt).
- **Memory Fault Handling**: The Memory Fault Register (MFR) detects and manages memory faults. If a fault occurs, the system resets the PC and stores the fault address in memory.
- **Inheritance**: The CPU class extends the Converter class, allowing direct access to number conversion methods, which are integral for interpreting opcodes and addressing information.

---

3. Key Design Considerations

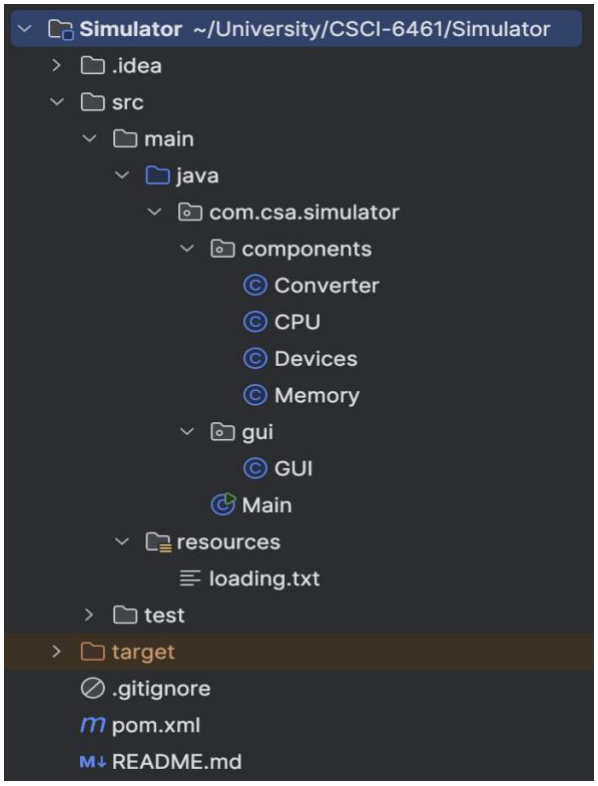
- **Manual Layout**: The reliance on setBounds() for GUI positioning results in a static layout that may not scale well with different screen sizes. Refactoring to use layout managers like GridLayout or BorderLayout would enhance the flexibility of the interface.
- **Threading for Long Operations**: The use of SwingWorker to handle long-running operations in the background ensures that the GUI remains interactive while the system executes processes like the "Run" functionality.
- **Memory Size Flexibility**: The memory is currently fixed at 2KB, which could limit future scalability. Implementing a dynamic memory allocation system or configurable memory size could provide greater flexibility in the simulator.
- **Reusability of Conversion Utilities**: The Converter class methods for binary, decimal, and hex conversions are decoupled from the specific CPU or memory logic, making them highly reusable across the system.

---

4. Project Structure

- **src/main/java**: Contains the main source code for the project.
    - **com.csa.simulator.components**: Contains various components of the simulator, including:
        - **Converter**: Likely handles conversion operations or data types.
        - **CPU**: Manages CPU-related functionality.
        - **Devices**: Manages connected devices or I/O operations.
        - **Memory**: Handles memory management for the simulator.
    - **com.csa.simulator.gui**: Contains classes related to the graphical user interface (GUI).

    - **GUI**: The main class for the user interface components.
    - **Main**: Likely the main entry point of the application.



- **src/main/resources**: Contains non-code resources like configuration files.
    - **loading.txt**: Likely used during application loading or initialization.
- **src/test**: (Currently empty in the image) Reserved for test cases and unit tests.
- **target**: (Generated by Maven) Holds the compiled bytecode and packaged outputs (e.g., JAR files).
- **pom.xml**: The Maven Project Object Model file, used for managing project dependencies and build configurations.
- **README.md**: Documentation file providing an overview of the project and setup instructions.
- **.gitignore**: Specifies files to be ignored by Git version control (e.g., compiled binaries or temporary files).
