CSCI 6461 Machine Simulator - Part 1 Design Notes

Team - 5

- Jaiswal Nitish
- Jadhav Yash
- Jiang Xuechun

Problem Statement

The goal of this phase is to design and implement the fundamental machine architecture of a **16-bit processor simulator**. This includes developing a **basic memory system**, executing **Load (LDR)** and **Store (STR)** instructions, and creating an **initial user interface** to interact with the simulator. The simulator will process a predefined **Instruction Set Architecture (ISA)** that consists of 64 instructions, handling **machine faults** for undefined instructions.

Simulator Components

1. Memory System

- A **word-addressable memory** structure with a capacity to store **16-bit words**.
- **Memory Address Register (MAR)** and **Memory Buffer Register (MBR)** for addressing and data transfer.
- Support for **basic memory read/write operations**.

2. Register Set

- **General Purpose Registers (GPRs)**: R0 - R3
- **Index Registers (IXRs)**: IX1 - IX3
- **Program Counter (PC)**, **Instruction Register (IR)**, and **Condition Codes (CC)**.

3. Instruction Execution

- **Load (LDR)**: Transfers data from memory into a register.
- **Store (STR)**: Transfers data from a register into memory.

4. User Interface

- A **GUI-based simulator**, displaying:
  - Register contents
  - Memory address inputs
  - Execution controls (**Load, Store, Run, Step, Halt**)
  - Console output for debugging.

Implementation Details

- **Programming Language**: Java (as used in Part 0).

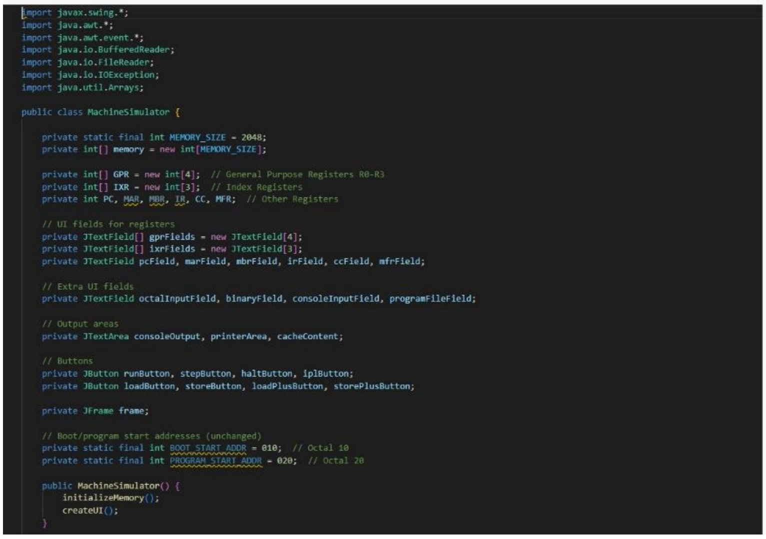
- **Data Structures**:
    - **Array-based memory model** (for simplicity).
    - **Object-oriented design** for registers and instruction handling.

Example Instruction Execution

- **LDR R3, 0, 15**
    - Fetches the value from memory address **15** into **Register 3**.
    - Updates relevant registers and displays changes in the GUI.
- **STR R3, 0, 20**
    - Stores the value from **Register 3** into memory address **20**.

Conclusion

This phase sets up the **core CPU simulation** with **basic memory access** and **register operations**. It also establishes an initial **graphical interface**, laying the foundation for more complex instruction executions in future phases.



```java
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.Arrays;

public class MachineSimulator {

    private static final int MEMORY_SIZE = 2048;
    private int[] memory = new int[MEMORY_SIZE];

    private int[] GPR = new int[4];  // General Purpose Registers R0-R3
    private int[] IXR = new int[3];  // Index Registers
    private int PC, MAR, MBR, IR, CC, MFR;  // Other Registers

    // UI fields for registers
    private JTextField[] gprFields = new JTextField[4];
    private JTextField[] ixrFields = new JTextField[3];
    private JTextField pcField, marField, mbrField, irField, ccField, mfrField;

    // Extra UI fields
    private JTextField octalInputField, binaryField, consoleInputField, programFileField;

    // Output areas
    private JTextArea consoleOutput, printerArea, cacheContent;

    // Buttons
    private JButton runButton, stepButton, haltButton, iplButton;
    private JButton loadButton, storeButton, loadPlusButton, storePlusButton;

    private JFrame frame;

    // Boot/program start addresses (unchanged)
    private static final int BOOT_START_ADDR = 010;  // Octal 10
    private static final int PROGRAM_START_ADDR = 020;  // Octal 20

    public MachineSimulator() {
        initializeMemory();
        createUI();
    }
```