CSCI 6461 COMPUTER SYSTEM ARCHITECTURE PROJECT
PART 1: DESIGN NOTES
TEAM NO: 9

1. Sumairah Rahman
2. Sourabh Rajgole
3. Chiranjib Samantaray

1. Function and Purpose

The main goal of this simulator for a machine is to mimic a simple computer system able to perform low-level commands. It includes a GUI for easy use in accessing registers, memory, and control commands. It also supports functionality for loading ROM files, running programs, and general and index register related operations.

This simulator is created exclusively for research and educational use, allowing users to:

- Import external programs in files to system memory.
- Step through program execution cycle-by-cycle.
- Execute programs and observe register/memory modifications.
- Perform data manipulation using memory load/store operations.
- Execute control flow related operations, including subroutine calls, jumps, and branches.
- Perform simple arithmetic and logical operations within the system architecture framework.

2. Components and Structure

Main Program: MachineSimulator (Java Class)

- Initializes memory and registers.
- Loads and execute machine instructions from a ROM file.
- Provides GUI elements for user interaction.

Graphical User Interface (Swing-based UI)

The simulator features an intuitive GUI that allows users to monitor and interact with the machine state dynamically. The UI consists of:

- **Register Panel:** Displays and allows input for General Purpose Registers (GPRs), Index Registers (IXRs), and control registers (PC, MAR, MBR, IR, CC, MFR).
- **Memory Management Panel:** Displays memory contents and allows storing/loading values.
- **Control Buttons:** Includes functions such as:
    - IPL (Initialize Program Load): Resets memory and loads a program.
    - Run: Executes the entire program until completion or a HALT instruction.
    - Step: Executes one instruction at a time.
    - Halt: Manually stops execution.
    - Load/Store: Facilitates memory interaction.
- **I/O Operations:**
    - Console input for user interactions.
    - Printer area that logs execution steps.
    - Cache content displays for recently accessed memory locations.

Memory and Register Structure
The simulator implements a basic memory model and register system:

- **Memory:** 2048 locations, each storing a 16-bit word.
- **Registers:**
    - **General Purpose Registers (GPRs):** R0 - R3, used for computation and data storage.
    - **Index Registers (IXRs):** IX1 - IX3, used for address modification.
    - **Control Registers:**
        - **Program Counter (PC):** Holds the address of the next instruction.
        - **Memory Address Register (MAR):** Holds the address being accessed in memory.
        - **Memory Buffer Register (MBR):** Stores data being read from or written to memory.
        - **Instruction Register (IR):** Stores the currently executed instruction.
        - **Condition Code Register (CC):** Holds flags for arithmetic/logical conditions.
        - **Machine Fault Register (MFR):** Stores error codes in case of invalid operations.

3. Logic Flow

Program Execution Process

1. **Initialization (IPL Button Pressed)**
    - Memory is reset to zero.
    - Registers are initialized to zero.
    - A ROM file is loaded if specified.

2. **Instruction Fetch-Decode-Execute Cycle**
    - **Fetch:** The instruction at the Program Counter (PC) is loaded into the Instruction Register (IR).
    - **Decode:** The instruction is broken down into opcode, register, and memory address.
    - **Execute:** The corresponding operation (Load, Store, Jump, Arithmetic, Logical) is performed.
    - **PC Update:** The PC is incremented or modified based on control flow instructions.

3. **Step Execution**
    - Executes a single instruction and updates registers/memory.
    - Displays the updated state in the UI.

4. **Run Execution**
    - Iterates through the fetch-decode-execute cycle until a HALT instruction is encountered or memory limit is reached.

4. Instruction Processing
Instruction Format (16-bit)

Bits	**Description**
0-5	Opcode
6-7	GPR Index
8-9	IXR Index
10	Indirect Bit (I)
11-15	Address Field

Supported Instructions

- **Load/Store Instructions:** LDR (Load Register), STR (Store Register), LDA (Load Address), LDX (Load Index Register), STX (Store Index Register).
- **Control Instructions:** JZ (Jump if Zero), JNE (Jump if Not Equal), JCC (Jump on Condition), JMA (Jump Unconditionally), JSR (Jump to Subroutine), RFS (Return from Subroutine), SOB (Subtract One and Branch), JGE (Jump if Greater or Equal).

5. Handling Errors
The simulator incorporates robust error handling to prevent invalid operations:

- **Invalid Instructions:** An invalid opcode will be encountered in cases of incorrect usage.
- **Memory Access Errors:** It prevents accessing or writing in memory locations beyond the limits allowed.
- **Management of invalid input:** Ensures values entered in registers or in-memory space follow appropriate formatting guidelines.
- **Machine Fault Register (MFR)**: It is a store for faults and may be checked for faults while running.

6. Flexibility and Extensibility
The machine simulator is designed to be modular and extensible:

- **Modular Design:** Instruction processing, memory handling, and UI interactions are handled in separate functions to allow for easy modifications.
- **Extensible Instruction Set:** Additional instructions can be added by modifying the execution logic.
- **Dynamic Memory Handling:** The memory model can be expanded to accommodate more complex simulations.
- **Debugging Features:** The printer and console areas provide real-time feedback for debugging and analysis.

7. Output Handling

- **Console Output:** Displays program execution logs, including register updates and control flow changes.
- **Printer Area:** Logs instructions executed, memory updates, and errors encountered.
- **Cache Display:** Shows recently accessed memory locations to simulate cache behavior.

8. Summary

The machine simulator is a learning tool aimed at explaining processes involved in instruction execution, memory handling, and register manipulation. By offering interactive graphic user interface and extensive feedback about execution, it assists in learning about programming at a machine level in a controlled environment. Its expandability and presence of adjustable elements also make it a flexible tool for studying computer structure and instruction set simulation.