# CSCI6461 Section 10 Group Project Documentation: Part 1

**Team Number:** 09

**Group Members:**

1. Jugal Gajjar
2. Kamalasankari Subramaniakuppusamy
3. Karan Patel

## Introduction:

Part 1 of the CS6461 Computer Architecture Project aims to create a simulator that modifies system registers based on user inputs. This simulator will read the load file generated by the assembler and simulate the machine's operations. To simulate the C6461's behavior, a graphical control panel will allow the user to interact with the system in real-time, managing the simulation process and adjusting the registers as needed.

## Objectives of Part 1:

The objectives of Part 1 of the CS6461 Computer Architecture Project Simulator can be summarized as follows:

1. Simulation of Basic Operations: To simulate a basic computing system capable of executing a limited set of instructions (LDR, STR, LDA, LDX, STX) to demonstrate CPU operations.
2. User Interaction: To develop an interactive control panel that allows users to easily load programs, modify register values, and step through or run code.
3. Memory Management: To support the loading and management of instructions and data in memory, requiring users to prepare load files in a specific format.
4. Debugging Support: To include features like single-step execution and real-time updates of registers and memory to help users trace the program's execution flow.
5. Visualization of Execution: To provide a visual representation of the system's state, including the status of registers, the program counter, and the instruction register, along with the effects of executed instructions.
6. Clear Documentation: To offer thorough documentation that guides users through installation, operation, and troubleshooting of the simulator.
7. Future Development: To set the foundation for future improvements, such as adding more instructions, enhancing error handling, and potentially integrating the assembler in later versions.

**Design Structure:**

1. Main Class (‘Main.java’):
   - The entry point of the assembler.
2. Simulator Class (‘Simulator.java’)
   - Manages the graphical user interface (GUI) for interacting with the simulator, handling file input, instruction execution, and display updates.
   - Key Functions:
     - setup GUI: Creates the GUI using JTextField, JButton, and JLabel for registers and control buttons.
     - File Operations: Allows the user to load a program file in .txt format via a file chooser, and the instructions are processed sequentially.
     - Execution Control: Provides buttons for Run, Step, Halt, Reset, Load, and Store operations.

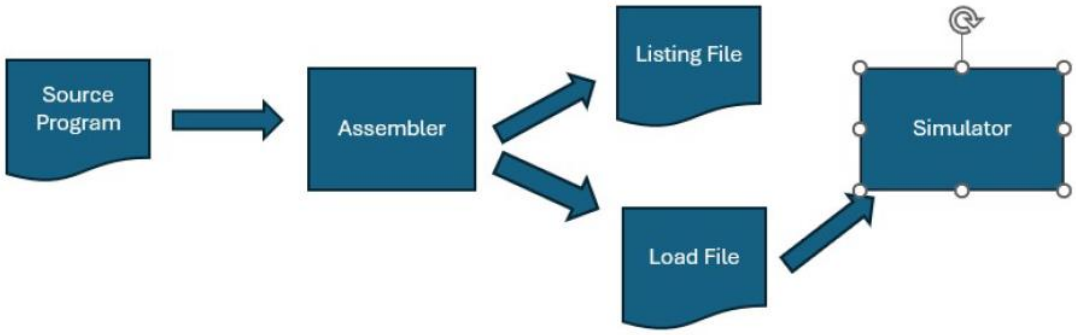



Fig. Overall Assembler Data Flow

**GitHub Link:**

https://github.com/JugalGajjar/CSCI6461-Computer-System-Architecture-Project/

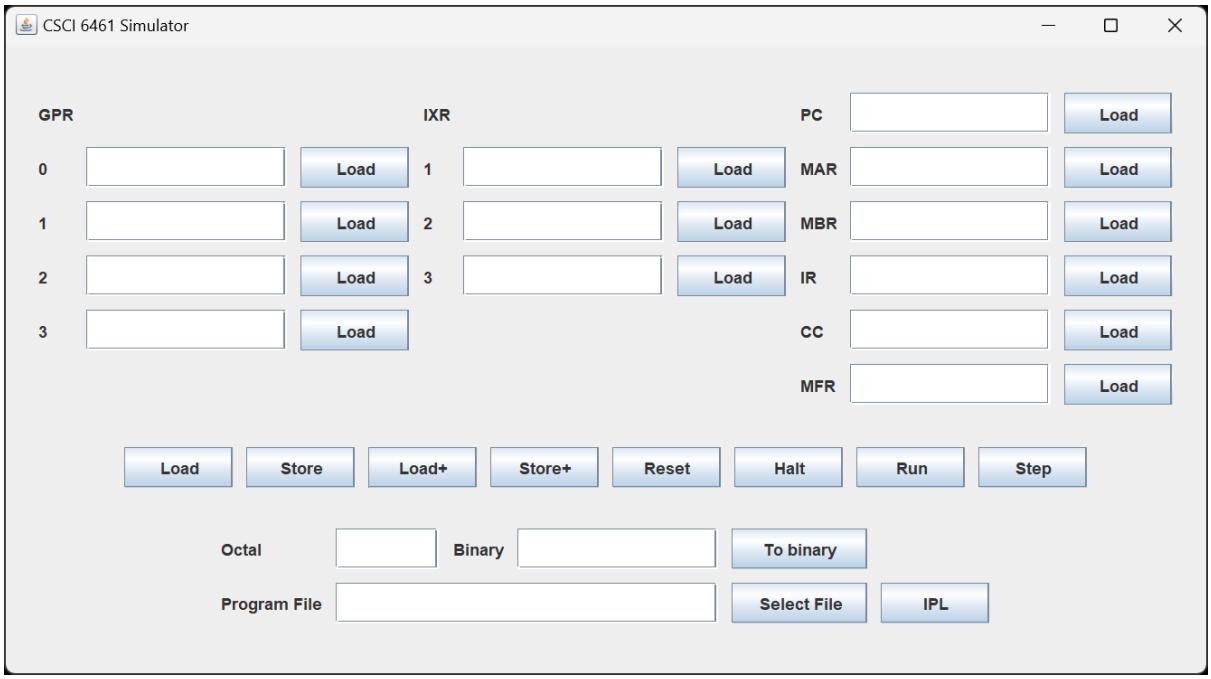
**Steps to Use:**

1. Install OpenJDK 17.0.7
2. Execute .jar file. (java -jar CSCI6461.jar)

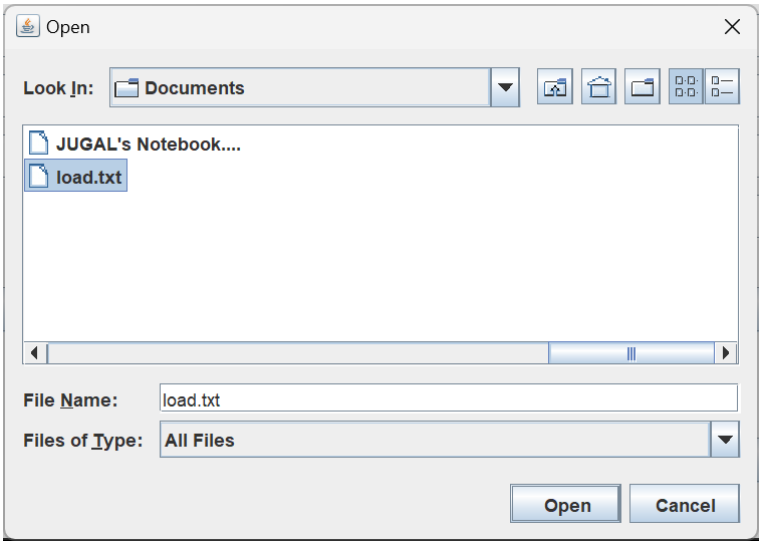
**Important Note:**

The key requirement to effectively execute the simulator is to create a load text file. This load text file should be created and stored in .txt format and must be stored in the same directory as the jar file for effective execution.

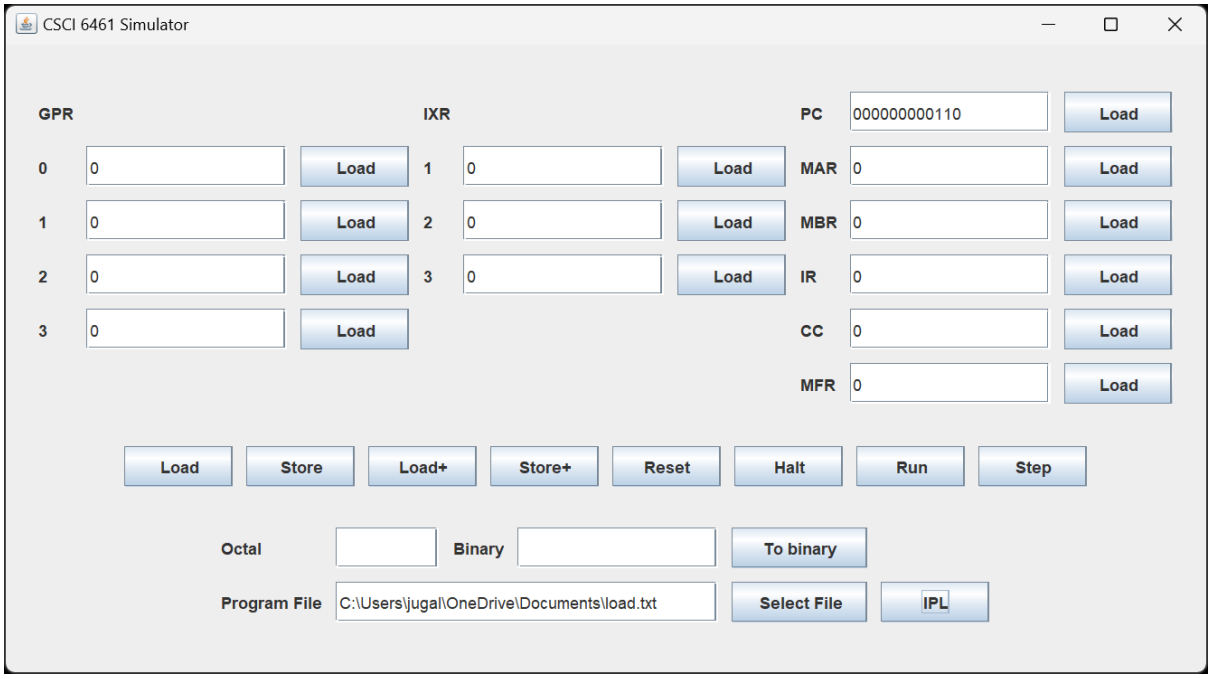
## Interface Components:



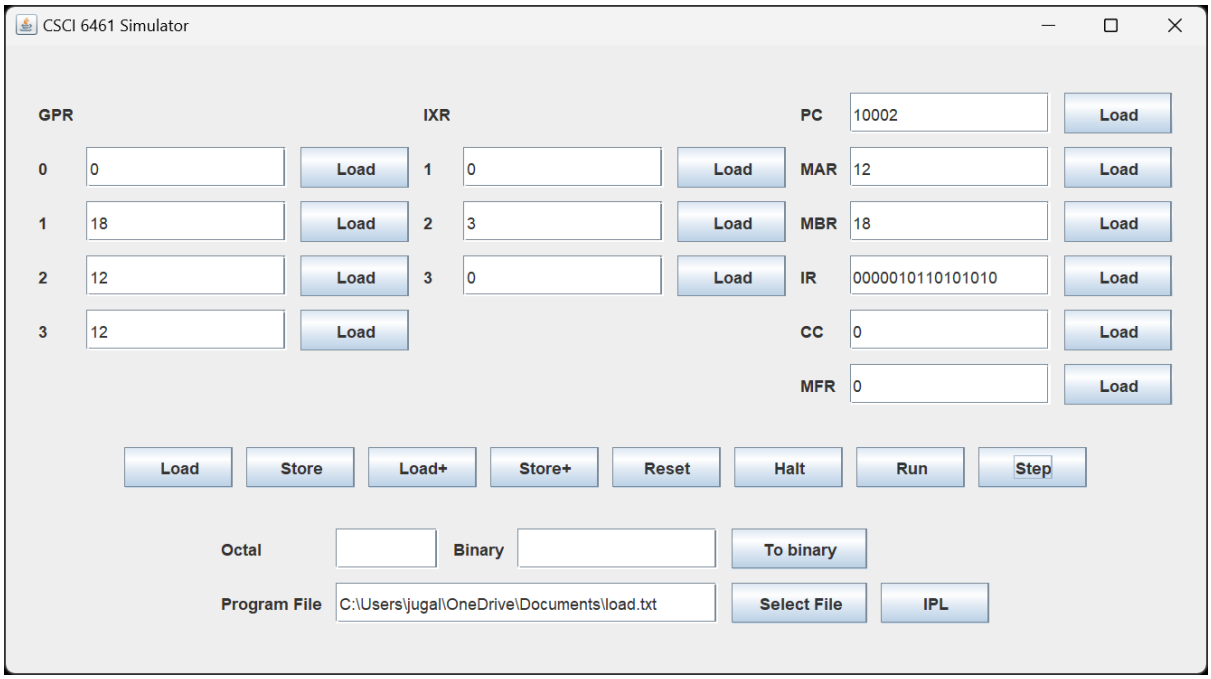

Console



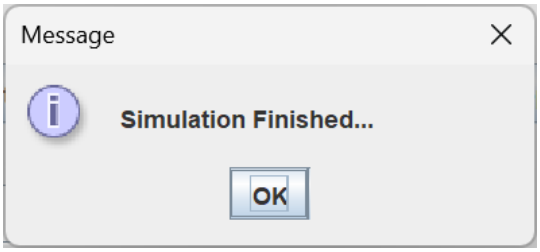

File Selector




Initialized Simulator




Some intermediate step with using ‘Step’




Message received upon successful 'Run'

1. **General Purpose Registers (GPR)**
    - There are four registers (labeled 0–3), each capable of holding a 16-bit binary value.
    - Each register includes a yellow button that allows users to load values directly from the Binary input field.
2. **Index Registers (IXR)**
    - Comprising three index registers (1–3), each also supports a 16-bit binary value.
    - Like the GPRs, each index register features a yellow button for value loading.
3. **Essential Registers**
    - **Program Counter (PC)**: This register points to the address of the next instruction to be executed. When updated, it retrieves the relevant instruction from memory and places it in the Instruction Register.
    - **Memory Address Register (MAR)**: This register contains the address of the word that will be accessed in memory.
    - **Memory Buffer Register (MBR)**: This stores the last word fetched from memory or the word that was just written back.
    - **Instruction Register (IR)**: This displays the instruction ready for execution. Each time the PC updates, the corresponding instruction appears in the IR and executes when either the Step or Run button is pressed.
    - **Condition Code (CC)**: This will be implemented with arithmetic and logical operations and includes four 1-bit flags: O (overflow), U (underflow), D (division by zero), and E (indicating equality).
    - **Machine Fault Register (MFR)**: This shows the machine fault code if an error occurs, with complete functionality expected in later updates.

**Buttons and Controls**

1. **Run**: Initiates the continuous execution of the loaded instructions.
2. **Step**: Executes one instruction at a time, which is particularly useful for debugging.

3. **Halt**: Stops the execution process of the simulator.
4. **IPL (Initial Program Load) and Select File**: Users must first select the appropriate program file (formatted in Octal) by clicking "Select File," followed by clicking IPL to load the program into memory.
5. **Load / Load+**: These buttons fetch the word from the memory address specified by the MAR into the MBR. The Load+ option also increments the MAR after loading.
6. **Store / Store+**: These buttons save the word in the MBR to the memory address indicated by the MAR. Store+ will increment the MAR after the operation.
7. **Clear**: Resets all registers and memory to zero, which is recommended before loading a new program.
8. **Octal Input/Conversion**: This field allows users to enter octal values, with an accompanying button (->Bin) to convert them into binary format.
9. **Binary Input**: Accepts binary input for loading into registers when the respective buttons are pressed. The input must fit within the register's limits; otherwise, it will not be accepted.
10. **Arrow Button (->A)**: Positioned next to the IR, this button reveals the assembly language interpretation of the instruction as understood by the machine.
11. **Contents of Memory Address Register (c(MAR))**: Clicking this will display a pop-up showing the contents of memory at the address currently stored in the MAR

**Memory and Registers Management:**

a. When the Step button is activated, it refreshes all registers and updates the console display, allowing users to see the impact of each instruction on the panel. Utilizing the Step function is recommended for those who wish to observe the effects of individual instructions.

b. By pressing the Run button, the program will execute until it finishes or reaches a Halt instruction. This rapid execution leads to frequent updates of the registers, making it challenging for users to monitor changes in real-time. While the CPU updates registers after each instruction, we intentionally chose not to refresh the console display with every update during the Run process. This decision was made because constant display refreshes would not significantly enhance the user experience.

**Conclusion:**

The CSCI 6461 simulator is a tool that replicates the functioning of the C6461 computer architecture by executing machine-level instructions. It processes program files, decodes opcodes, and performs operations on registers and memory, providing a real-time view of how each instruction is executed. The simulator not only supports various instruction types but also facilitates debugging through its step-by-step execution feature. By offering an interactive interface, this simulator enables users to gain a deeper understanding of how the C6461 system

manages memory, processes instructions, and handles control flow. Overall, this project serves as a foundation for experimenting with low-level operations and understanding the internal workings of computer architecture.