CSCI 6461 Computer Architecture - Project Part 1

Design Notes and User Guide for the Basic Machine Simulator

Submitted By: *Sribalaji Annamalai Senthilkumar, Abenezer Golda, Razia Noor, Christabell Rego*

1. Overview

Part 1 of this project transitions from assembly to simulation by focusing on the creation of the **basic machine architecture**. The primary objective is to build a functional Graphical User Interface (GUI) that represents the front panel of the CS6461 computer. This includes implementing the core CPU components like registers and memory, and executing the first set of fundamental instructions: LDR, LDA, LDX, and STR.

The successful completion of Part 1 is demonstrated by loading a simple assembly program (assembled in Part 0) into the simulator's memory and stepping through its execution, observing the state of the CPU registers and memory changing in real-time via the GUI.

2. Design Justification

The simulator is designed with a clear separation of concerns, dividing the logic between the user interface and the underlying CPU core.

Core Component: SimulatorGUI.java

This class is responsible for all visual aspects and user interactions.

- **Technology:** We chose **Java Swing** for the GUI framework. It is a standard part of the Java Development Kit (JDK), requires no external dependencies, and is well-suited for creating the required desktop application with interactive components like buttons and text fields.
- **Layout:** The interface is organized logically with a BorderLayout containing distinct panels for controls, registers, and memory. This provides an intuitive user experience that mirrors the layout of a physical machine's front panel.
- **Event-Driven Logic:** All user actions (e.g., clicking "IPL", "Single Step") are handled by event listeners. To prevent the GUI from freezing during continuous execution (Run), a SwingWorker is used to run the CPU loop on a separate thread, ensuring the interface remains responsive.

Core Component: CPU.java

This class encapsulates the entire state and logic of the simulated machine.

- **Encapsulation:** All registers (PC, IR, GPRs, etc.) and the main memory are contained within the CPU class. This creates a single source of truth for the machine's state, making the system easier to manage and debug. The GUI interacts with the CPU through public methods like executeInstruction(), readMemory(), and reset().
- **Instruction Execution:** A central executeInstruction() method implements a classic **Fetch-Decode-Execute** cycle. It fetches the instruction at the address pointed to by the PC, decodes the opcode and other fields using bitwise operations (shifting >> and masking &), and uses a switch statement on the opcode to perform the correct action. This design is highly extensible, as adding new instructions in Part 2 will simply involve adding new case blocks.

3. User Guide: How to Operate the Simulator

Step 1: Generate the JAR file

You can build the executable `.jar` file using either an IDE like IntelliJ or directly from the command line.

Option 1: Using IntelliJ IDEA (Recommended)

1. **Open Project Structure:** In the menu bar, go to **File -> Project Structure...**.
2. **Go to Artifacts:** Select **Artifacts** from the left-hand panel.
3. **Create JAR Artifact:** Click the **+** button, hover over **JAR**, and select **From modules with dependencies...**.
4. **Configure Main Class:** In the dialog box, click the `...` button for **Main Class** and select `SimulatorGUI`.
5. **Build Artifact:** Click **OK**, then go to the main menu and select **Build -> Build Artifacts... -> Build**. The `CS6461_Part1.jar` will be created in the `out/artifacts/` directory.

Option 2: Using the Command Line

Compile Java Files: Open a terminal in your project's root directory and run:

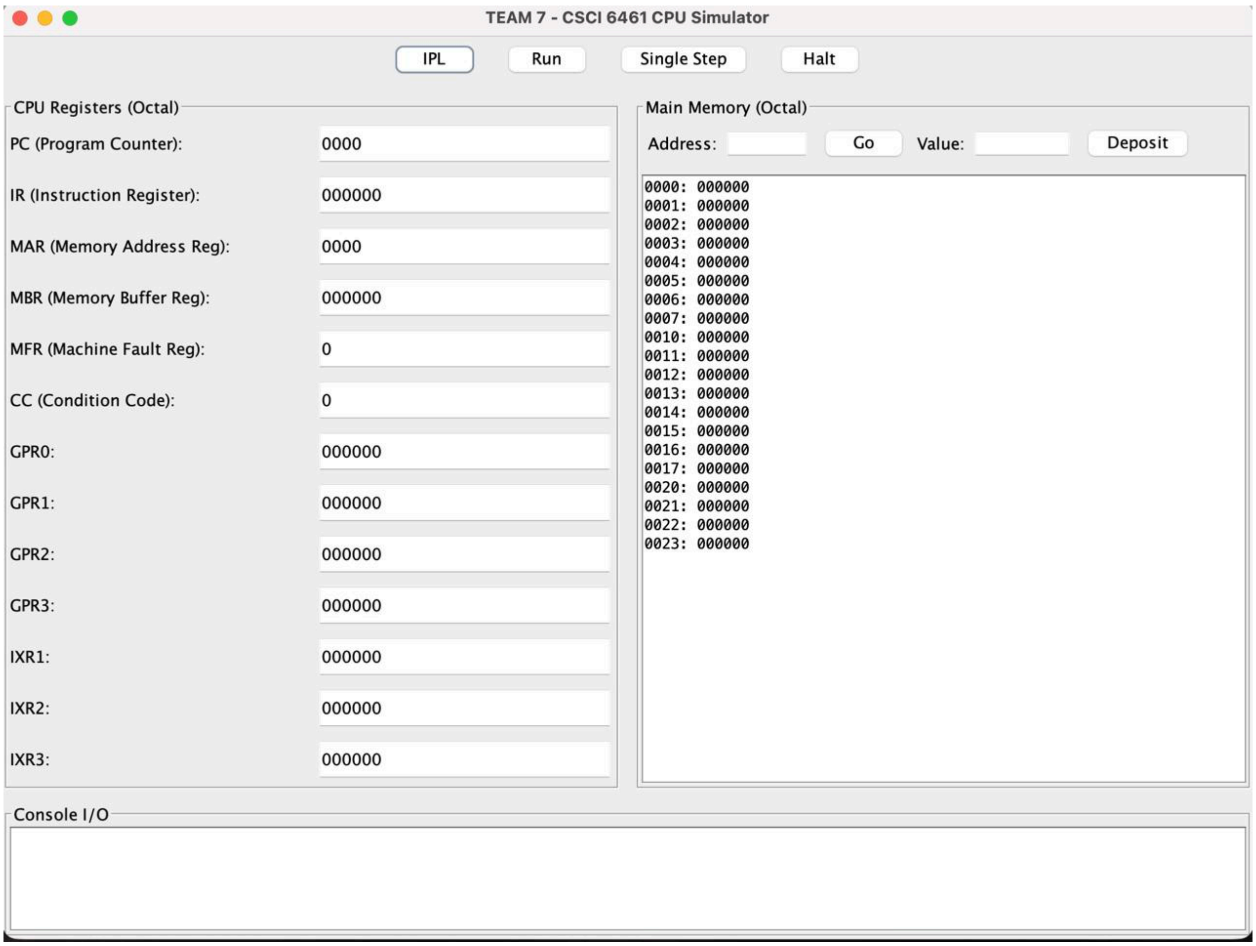
javac -d out src/CPU.java src/SimulatorGUI.java

Create the JAR: Run the following command to package the compiled files into an executable JAR:
jar cfe CS6461_Part1.jar SimulatorGUI -C out .

Step 2: Run the Simulator

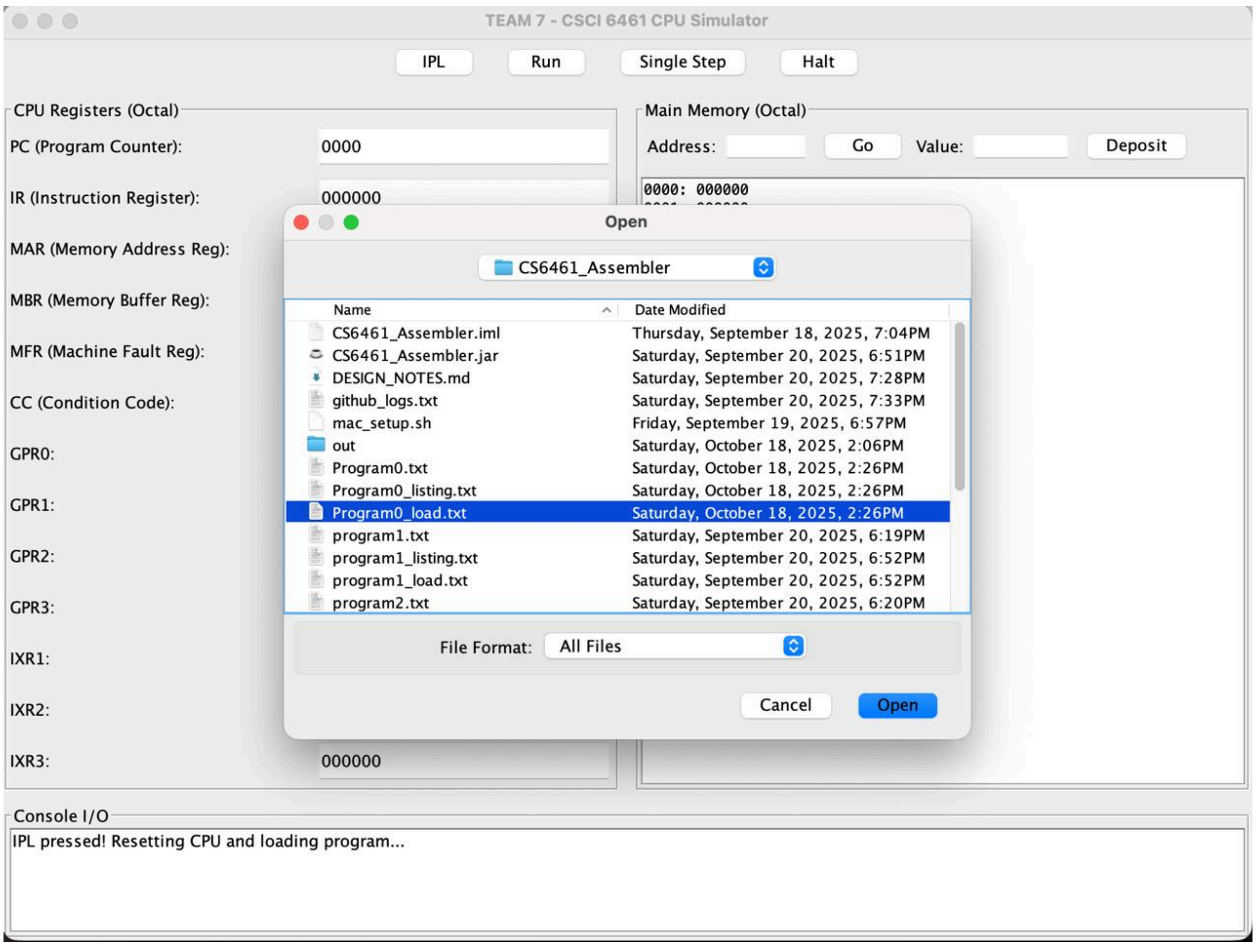
Open a terminal or command prompt, navigate to the directory containing the JAR file, and execute the following command:

java -jar CS6461_Part1.jar



Step 3: Loading a Program (IPL)

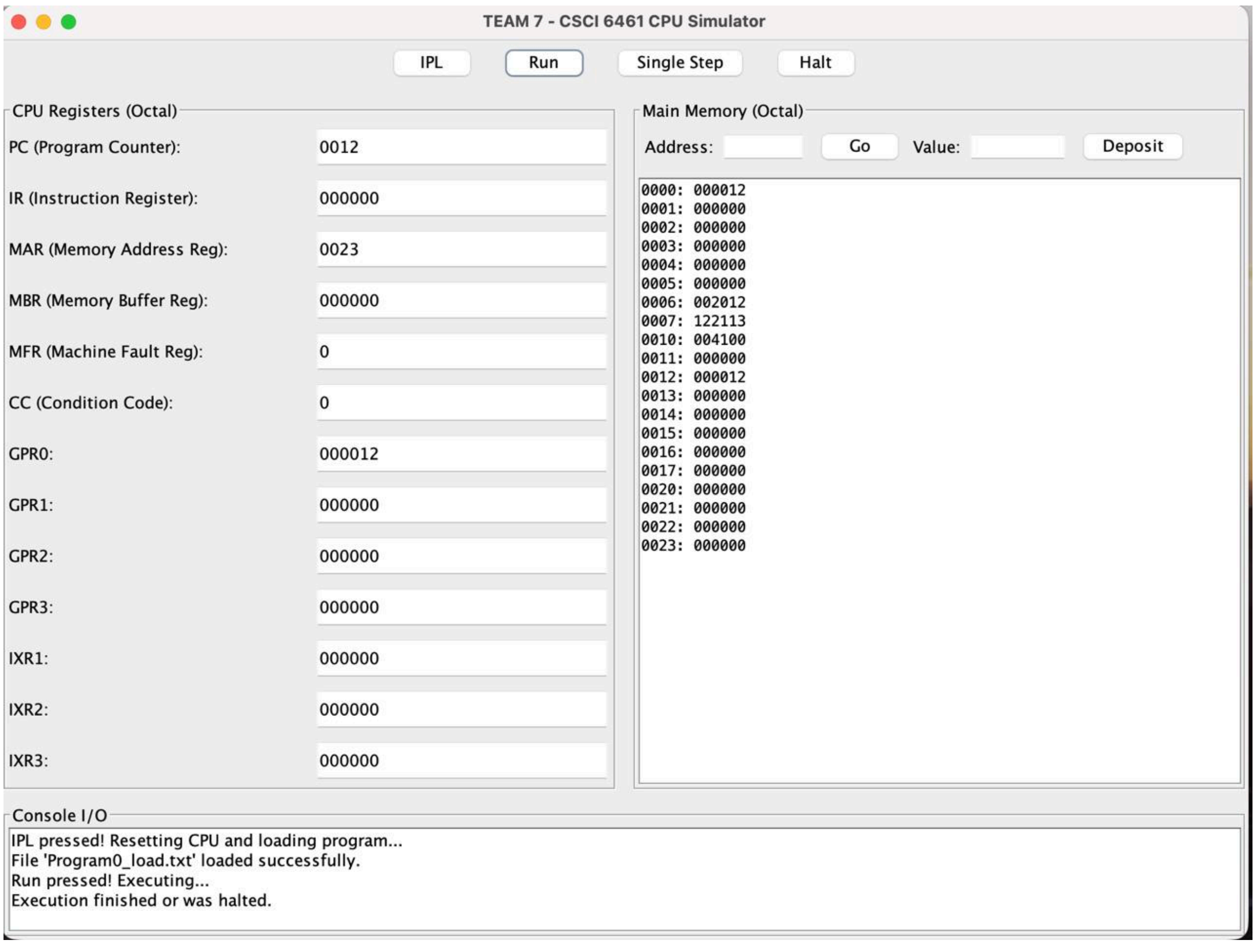
1. Click the **IPL (Initial Program Load)** button on the GUI.
2. A file chooser dialog will appear. Select your program0_load.txt file.
3. Upon successful loading, the "Console I/O" will display a confirmation message.



Step 4: Executing the Program

You have two options for execution:

- **Single Step (Recommended for Debugging):** Click the **Single Step** button to execute one instruction at a time.
- **Run:** Click the **Run** button to execute instructions continuously until a HLT instruction is reached.



Step 5: Interacting with Memory

- **To View:** Enter an octal address in the "Address" field and click **Go**.
- **To Modify:** Enter an octal address and an octal value and click **Deposit**.\

4. Test Case: program0.txt

This simple program is used to verify the functionality of the basic load and store instructions.

Source Code (program0.txt):

```
; CSCI 6461 - Program 0
; Demonstrates LDR, LDX, and STR instructions.

        LOC 6             ; Start code at address 6
START:
        LDR 0,0,VAL_A     ; Load GPR0 with the value at VAL_A (10)
        LDX 1,STORE_LOC   ; Load IXR1 with the address of STORE_LOC
        STR 0,1,0         ; Store GPR0's value into the address held by IXR1
        HLT               ; Halt the machine

; --- Data Section ---
VAL_A:      DATA 10
STORE_LOC:  DATA 0
```

Expected Outcome: After running, the value 12 (octal for 10) will be stored at memory location 13 (octal address of STORE_LOC).