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.

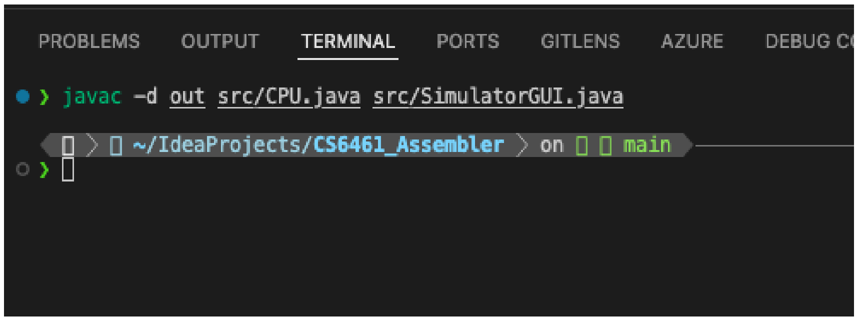
3. User Guide: How to Operate the Simulator

Step 1: Compiling the Simulator

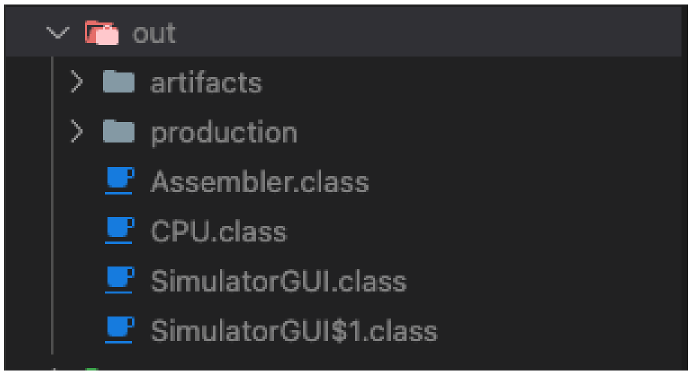
Ensure both SimulatorGUI.java and CPU.java are in the src directory. Open a terminal in the project's root folder and run:

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

This will place the compiled .class files into an out directory.



```
❯ javac -d out src/CPU.java src/SimulatorGUI.java
  ~/IdeaProjects/CS6461_Assembler  on   main
❯
```

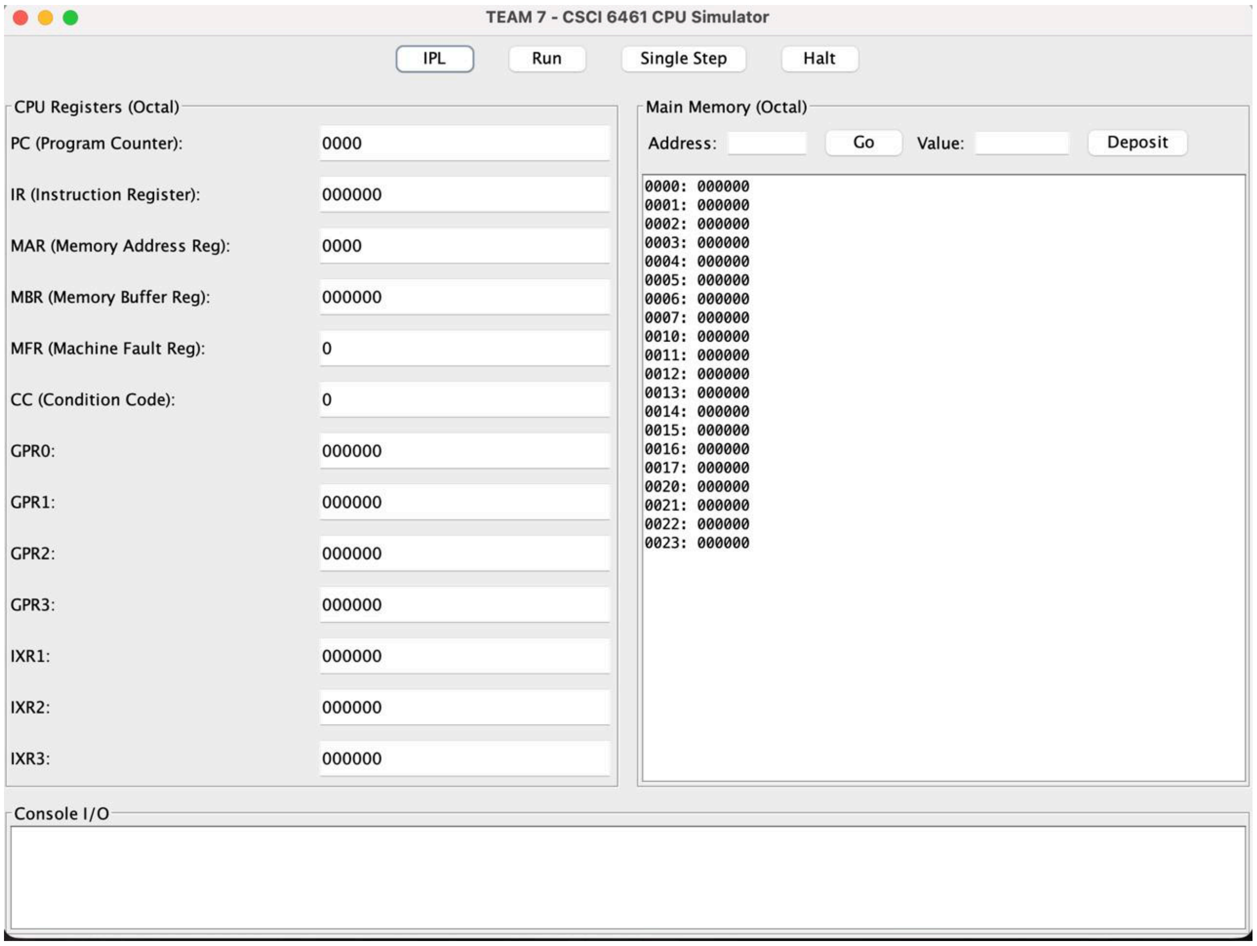


```
out
  artifacts
  production
  Assembler.class
  CPU.class
  SimulatorGUI.class
  SimulatorGUI$1.class
```

Step 2: Running the Simulator

From the same terminal, run the following command to start the GUI:

```
java -cp out SimulatorGUI
```



CPU Registers (Octal)

PC (Program Counter):	0000
IR (Instruction Register):	000000
MAR (Memory Address Reg):	0000
MBR (Memory Buffer Reg):	000000
MFR (Machine Fault Reg):	0
CC (Condition Code):	0
GPR0:	000000
GPR1:	000000
GPR2:	000000
GPR3:	000000
IXR1:	000000
IXR2:	000000
IXR3:	000000

Main Memory (Octal)

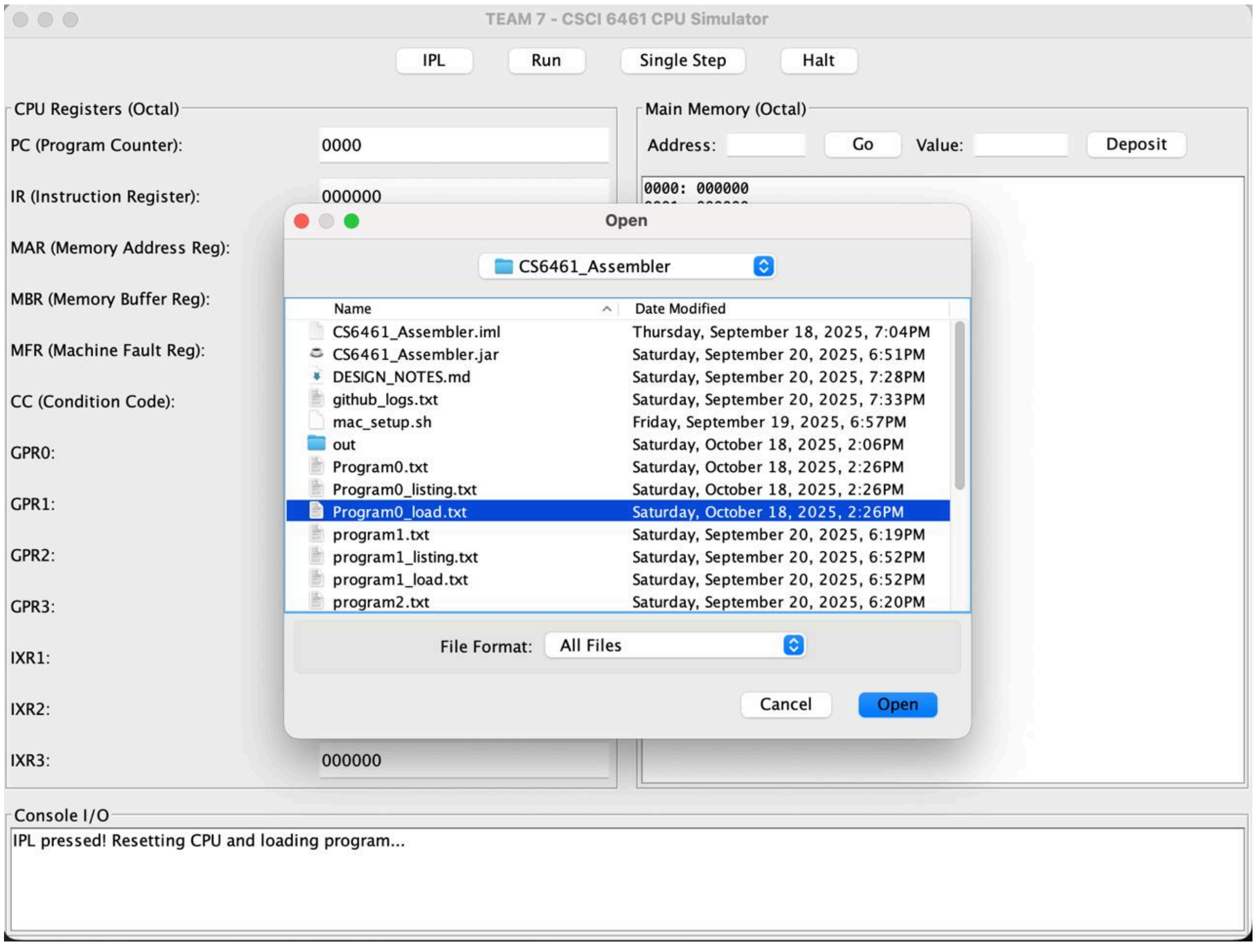
Address: Value:

```
0000: 000000
0001: 000000
0002: 000000
0003: 000000
0004: 000000
0005: 000000
0006: 000000
0007: 000000
0010: 000000
0011: 000000
0012: 000000
0013: 000000
0014: 000000
0015: 000000
0016: 000000
0017: 000000
0020: 000000
0021: 000000
0022: 000000
0023: 000000
```

Console I/O

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 (which should have been generated by your Part 0 assembler).
3. Upon successful loading, the "Console I/O" will display a confirmation message, and the machine code will be loaded into the simulator's memory. The PC will be set to the starting address of the program.



PC (Program Counter): 0000
IR (Instruction Register): 000000
MAR (Memory Address Reg):
MBR (Memory Buffer Reg):
MFR (Machine Fault Reg):
CC (Condition Code):
GPR0:
GPR1:
GPR2:
GPR3:
IXR1:
IXR2:
IXR3: 000000

```
0000: 000000
```

Name	Date Modified
CS6461_Assembler.iml	Thursday, September 18, 2025, 7:04PM
CS6461_Assembler.jar	Saturday, September 20, 2025, 6:51PM
DESIGN_NOTES.md	Saturday, September 20, 2025, 7:28PM
github_logs.txt	Saturday, September 20, 2025, 7:33PM
mac_setup.sh	Friday, September 19, 2025, 6:57PM
out	Saturday, October 18, 2025, 2:06PM
Program0.txt	Saturday, October 18, 2025, 2:26PM
Program0_listing.txt	Saturday, October 18, 2025, 2:26PM
Program0_load.txt	Saturday, October 18, 2025, 2:26PM
program1.txt	Saturday, September 20, 2025, 6:19PM
program1_listing.txt	Saturday, September 20, 2025, 6:52PM
program1_load.txt	Saturday, September 20, 2025, 6:52PM
program2.txt	Saturday, September 20, 2025, 6:20PM

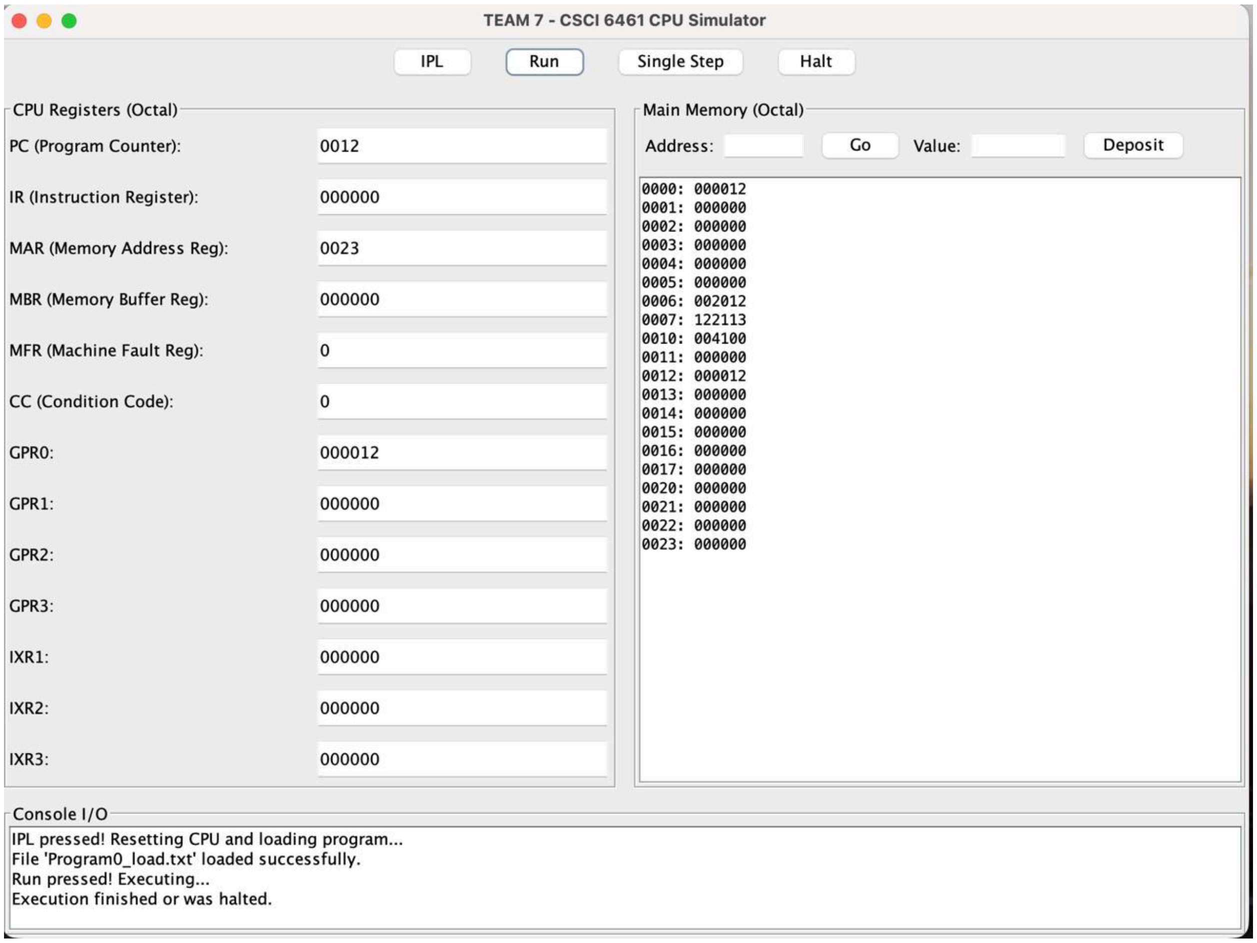
File Format: All Files

```
IPL pressed! Resetting CPU and loading program...
```

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.
  - After each click, observe the CPU registers (PC, IR, GPRs, IXRs) and the memory view to see how the state of the machine changes. This is the best way to verify that each instruction is working correctly.
- **Run:**
  - Click the **Run** button to execute instructions continuously until a HLT instruction is reached or a machine fault occurs.
  - The GUI will display the final state of the machine after the program has finished.



CPU Registers (Octal)

PC (Program Counter):	0012
IR (Instruction Register):	000000
MAR (Memory Address Reg):	0023
MBR (Memory Buffer Reg):	000000
MFR (Machine Fault Reg):	0
CC (Condition Code):	0
GPR0:	000012
GPR1:	000000
GPR2:	000000
GPR3:	000000
IXR1:	000000
IXR2:	000000
IXR3:	000000

Main Memory (Octal)

Address: Value:

```
0000: 000012
0001: 000000
0002: 000000
0003: 000000
0004: 000000
0005: 000000
0006: 002012
0007: 122113
0010: 004100
0011: 000000
0012: 000012
0013: 000000
0014: 000000
0015: 000000
0016: 000000
0017: 000000
0020: 000000
0021: 000000
0022: 000000
0023: 000000
```

Console I/O

```
IPL pressed! Resetting CPU and loading program...
File 'Program0_load.txt' loaded successfully.
Run pressed! Executing...
Execution finished or was halted.
```

Step 5: Interacting with Memory

You can manually inspect and modify memory at any time:

- **To View:** Enter an octal address in the "Address" field and click **Go**. The value at that address will be displayed in the "Value" field.
- **To Modify:** Enter an octal address and an octal value in their respective fields and click **Deposit**. The new value will be written to memory.

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).