

The School of Engineering and Applied Science

of The George Washington University

CSCI 6461

Fall 2023

Project 1, Team 12

| Anirban Khara | G40191570 |
| --- | --- |
| Harish Kannan | G41152289 |
| Sri Sasmi Polu | G40795997 |

**Table of Content:**1. Project Design

* Process Flowchart
* Front Panel Design
* GUI Component Registers

2. Usage Instructions

3. Operation

1. **Design**

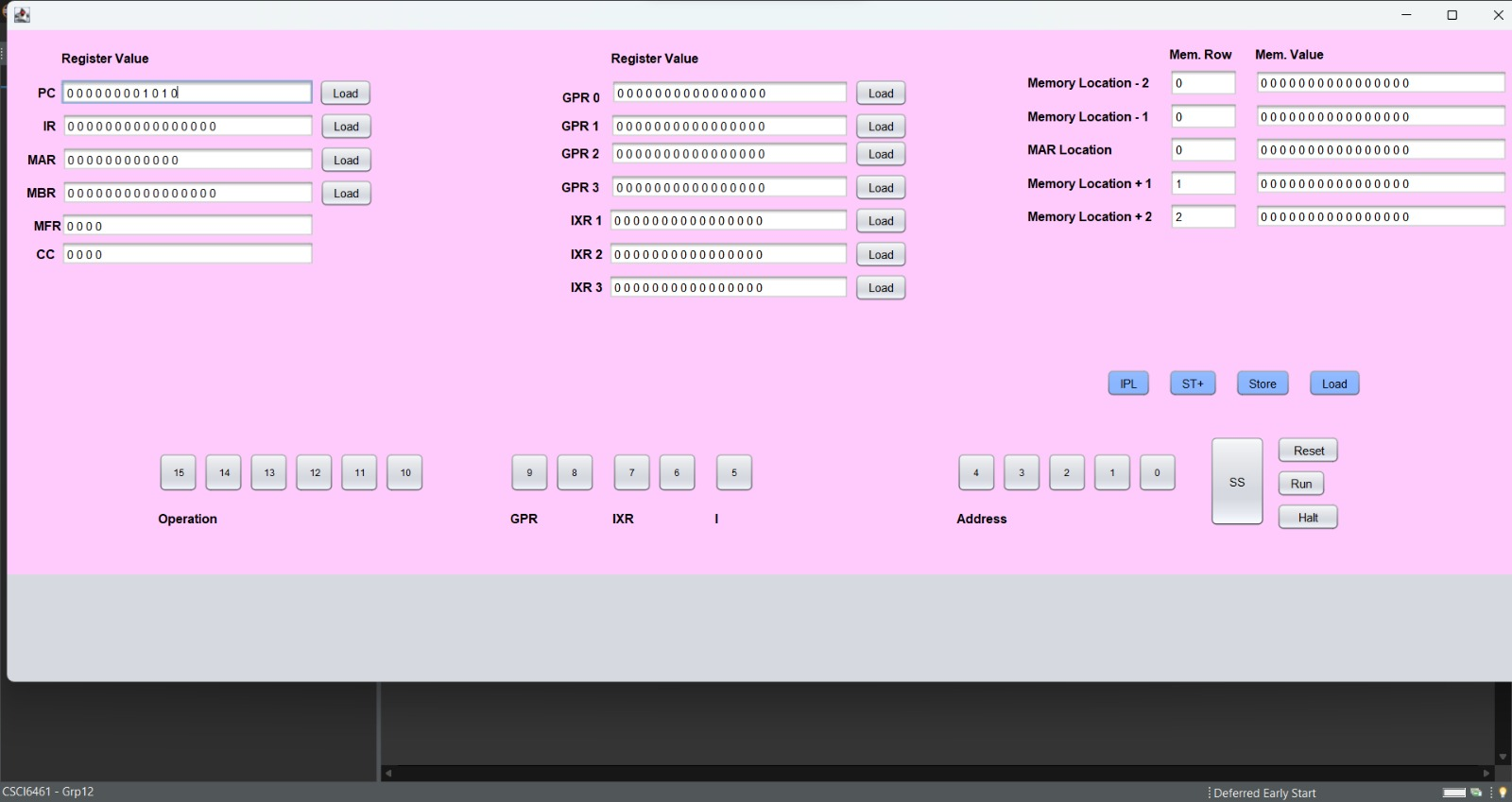
**1.1 Process Flowchart**

The Basic Machine CISC Simulator initializes and loads a machine code program into memory, executing instructions sequentially. Each instruction is loaded, decoded, and executed, with registers and flags updated accordingly. The process iteratively continues until all instructions are executed, marking the end of the simulation.

* Initialize the simulator, registers, memory, and flags.
* Load a machine code program into memory.
* Set the Program Counter (PC) to the program's starting address.
* Begin the execution cycle.
* Load the instruction at the PC into the Instruction Register (IR).
* Increment the PC to the next instruction.
* Decode the instruction in the IR.
* If it’s a Load/Store instruction, calculate the effective address using base and offset (if index registers are used), access memory using the Memory Address Register (MAR), and execute the operation, loading data from the Memory Buffer Register (MBR) to a General Purpose Register (GPR) or vice versa.
* For other instructions, execute as per their operation codes and operands.
* Update relevant registers and flags.
* If there are more instructions, return to step 5; otherwise, end execution.

**1.2 Front Panel Design**

| **Machine Register** | **Function** |
| --- | --- |
| PC | Display and Change |
| MAR | Display and Change |
| MBR | Display and Change |
| GPR Registers 0-3 | Display and Change |
| Index Registers 1-3 | Display and Change |
| CC | Display |
| MFR | Display |
| IR | Display and Change |

****

**1.3 GUI Component Registers**

The user interface allows for input in hexadecimal or binary format, displaying the corresponding conversion. The internal registers, including General Purpose Registers, Index Registers, and others, play crucial roles in instruction execution, data storage, and program running processes.

The UI Display features an input panel where users can input numbers in either hexadecimal or binary format. Upon entering a binary number, its hexadecimal equivalent will be displayed and vice versa. This feature facilitates easy conversion and display of numbers in both formats.

**GPR0-3 (General Purpose Registers 0-3):** These are four 16-bit registers capable of storing both data and memory addresses. They are versatile and essential for various operations.

**IXR1-0 (Index Registers 1-0):** Comprising three 16-bit registers, the Index Registers store offsets for memory locations. They are crucial for pointing to operand addresses during program execution.

**PC (Program Counter):** A 12-bit register, the Program Counter holds the address of the next instruction to be executed, facilitating efficient task execution and tracking of the current operation point.

**MAR (Memory Address Register):** This 12-bit register is vital for accessing data and instructions from memory during the execution phase. It holds the address of the required data, which is then transferred to the MBR.

**MFR (Machine Fault Register):** The 4-bit Machine Fault Register is designed to capture and indicate machine faults.

**MBR (Memory Buffer Register):** A 16-bit register, the MBR stores data being transferred to and from the immediate access store, acting as a buffer between the processor and memory units.

**IR (Instruction Register):** The 16-bit Instruction Register holds the instruction currently being executed or decoded, playing a pivotal role in the instruction execution process.

**CC (Condition Code Register):** The CC register displays flags for specific conditions like overflow and division by zero.

**SS (Single Step Button):** The SS button allows users to execute instructions one step at a time, providing a granular view of the operation.

**Run Button:** The Run button executes all instructions inputted by the user, producing the final output.

**INIT Button:** The INIT button initializes the Program Counter, Instruction Register, and memory, preparing the system for operation.

**IPL Button:** The IPL button is used to initiate the running of the program, setting the operation in motion.

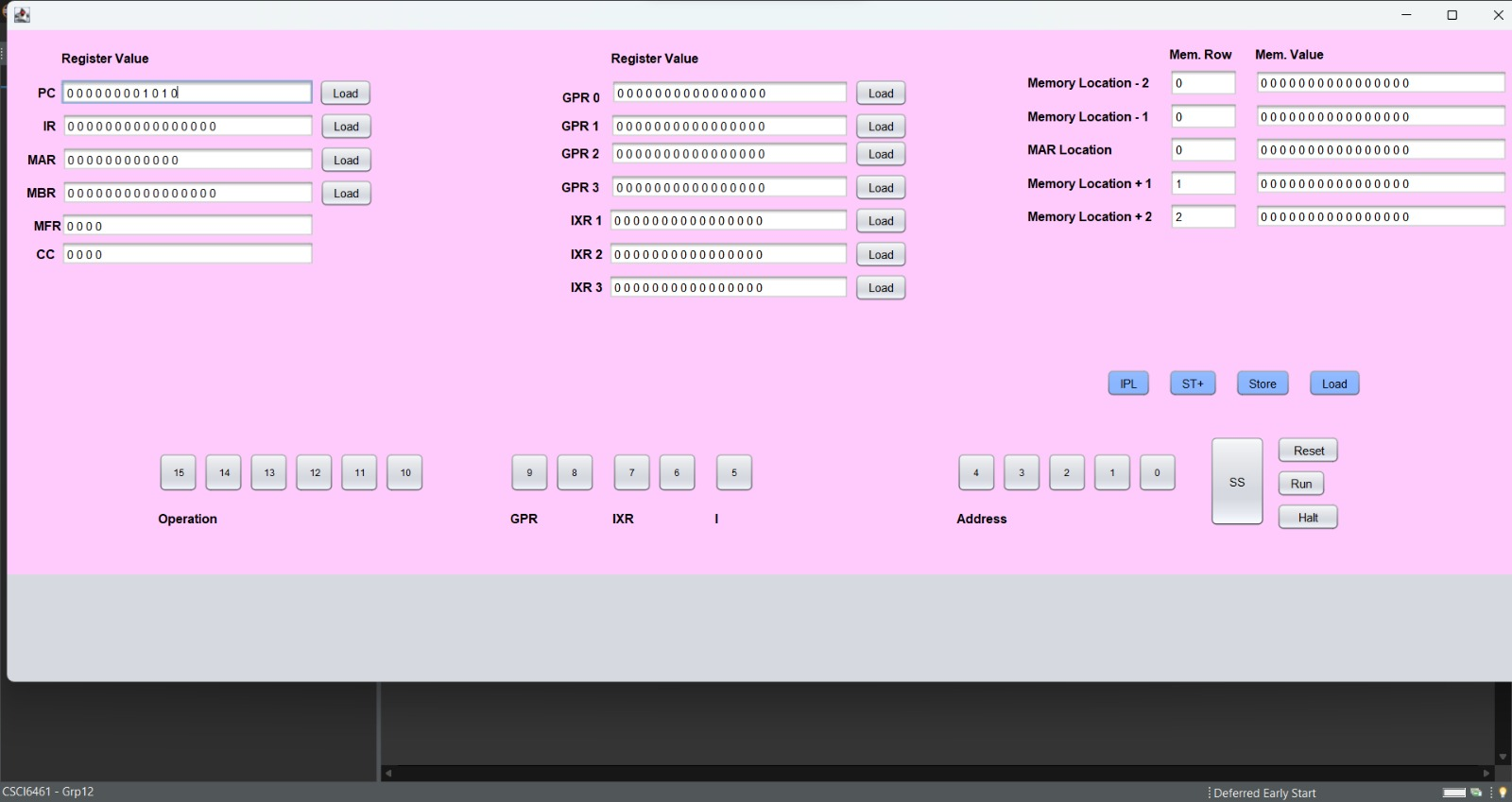
1. **Usage Instructions**

This brief guide provides essential instructions for setting up and operating the CSCI6461\_Project\_Part1 simulator. The process involves downloading the source code, preparing the software, and understanding the simulator's operation. The simulator requires the latest Java version and specific file naming conventions for smooth operation.

* Download the source code by acquiring the file **CSCI6461\_Team12\_Part1.zip** from the designated board.
* Unzip the downloaded file and navigate to the**/CSAproject.1/dist** folder.
* Ensure you have the latest Java version installed to avoid compatibility issues experienced with older versions.
* Within the dist folder, you'll find **CSCI 6461 - Grp 12.jar**.
* To run this from command line, open your terminal, go to the folder and run the command **java -jar “CSCI 6461 - Grp 12.jar”**
* Execution begins at memory address 0x0030.
* You may replace the IPL file located under /dst. Load it by clicking the IPL button. A demonstration program is included for your reference.
* Execute the program step-by-step using the ‘SS’ button or run it entirely with the ‘run’ button. The program will halt upon encountering a HLT instruction, making a bad memory call, pressing the Halt button, or pressing the run button again.
* The display panel allows for memory reading, showing two memory addresses adjacent to the MAR.
* Note: Although the front panel buttons are functional, they may not visually indicate their active or inactive states.

1. **Operations**

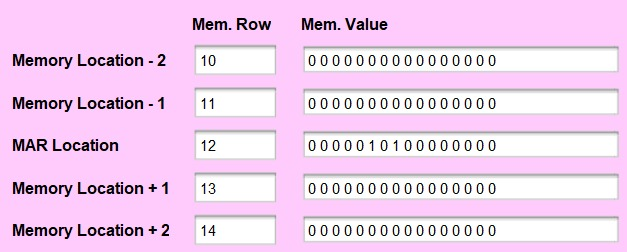
**1. Open the simulator (Steps in Section 2 - Usage Instructions)**

****

**2. Click on the IPL button**

****

**3. Check the memory locations and their respective value that is loaded from IPL.txt**

****