GEORGE WASHINGTON UNIVERSITY

Fall 2023

CSCI 6461 Computer System Architecture

Project Part 3 - Simulator for Executing All Instructions

Design Notes

Team 12 -

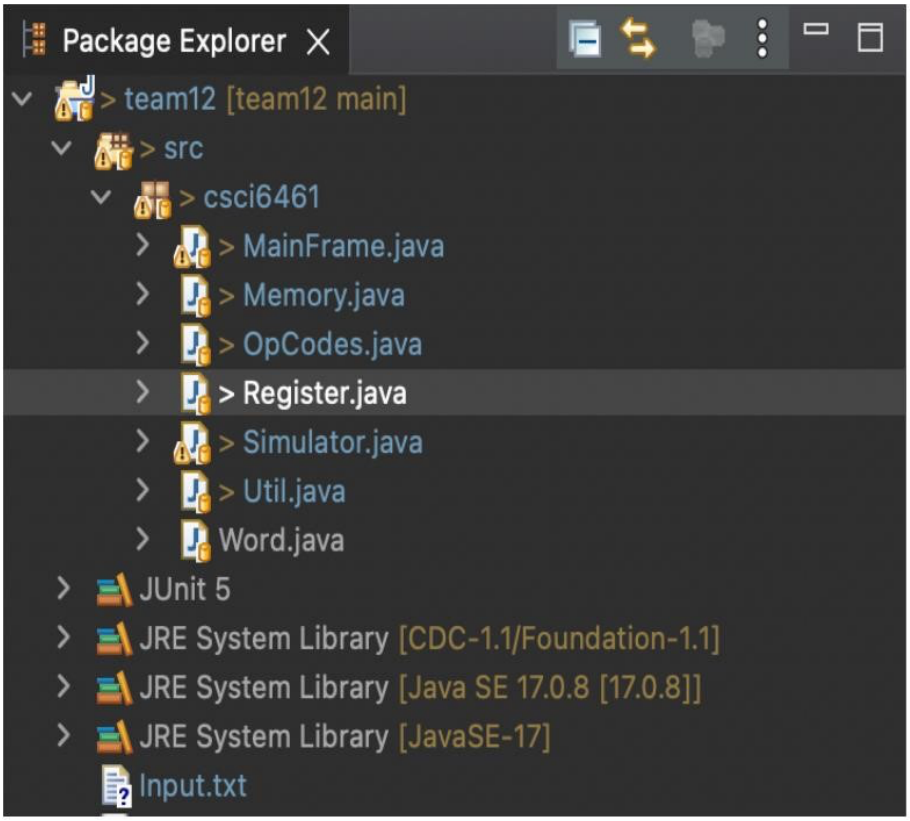
Sai Bharath Reddy Lattupalli - G41128949

Namana Y Tarikere - G21372717

Reshma Rajkumar - G36576199

Vaishnavi Goyal - G47669343

Classes



1. **MainFrame.java:** This class contains all methods to update the UI. It has "main()" which is the starting point for the program to start its execution. This Java code implements a graphical user interface (GUI) for a simulated computer system. The MainFrame class extends JFrame and includes text fields for various registers (GPR, IXR, PC, MAR, MBR, IR, MFR, Privileged, CC) along with buttons to load values into these registers, control the simulation, initialize the system, and load input from a file. The GUI enables users to interact with the simulated system, to load programs, execute instructions, and visualize the real time state of the system.

2. **Simulator.java:** This class has all the methods to perform operations such as Load, Store, Fetch, Decode and moves data between registers and memory. In this phase we have implemented code for executing all the Instructions. This class has methods for initializing the registers and simulator, to set all the UI components with default, to run the instructions one at a time, load instructions to given address, decode the instruction based on opcode, calculate effective address, to map string to register, store register to memory, to convert the Instructions into hexadecimal string.

3. **Memory.java:** This is a class responsible for representing the memory of a CISC (Complex Instruction Set Computer) simulator. The memory in this simulator contains 4096 words. It provides methods that the "Simulator" class utilizes for reading and writing the contents of this memory.

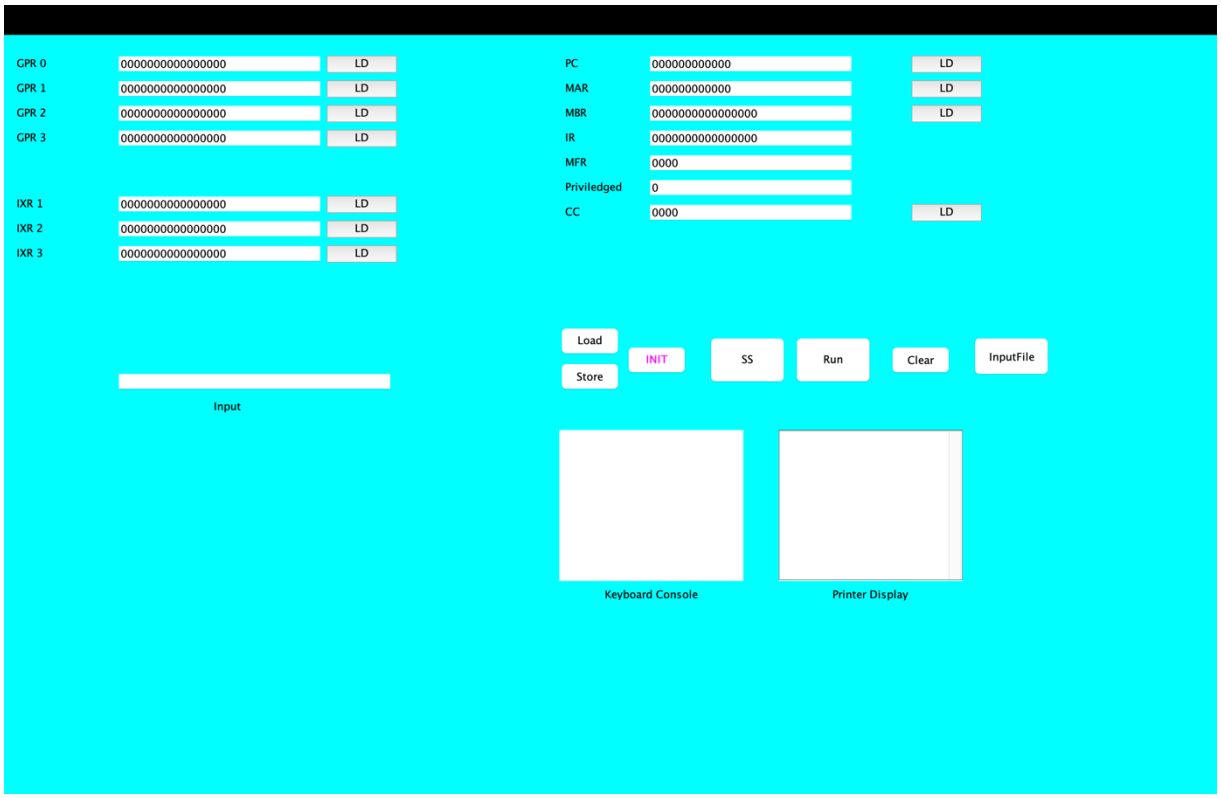
4. **Util.java:** This is a flexible utility class that offers ways to translate data between various forms, such as converting integers to Bit Set format or their hexadecimal representation. The conversion logic is kept distinct and organized from other sections of the code by this class, which also encourages code reuse. When such conversions are required, it can be applied in a variety of situations.

5. **Word.java:** This is a class designed to represent a word in a simulator. Within this framework, a word consists of a string of 16 binary digits, denoted as 0s and 1s. Working with these phrases in the simulator is made simpler by this class, which encapsulates their behaviour and characteristics. It might provide procedures for changing the 16-bit binary value and doing other operations on words.

6. **Register.java:** This class represents the register in a computer system that can hold a binary value and it returns the size of the register. The size of the register determines the maximum amount of data that can be stored in it. The Register.java class contains methods to set and retrieve the binary value stored in the register, as well as a method to return the size of the register.

7. **OpCodes.java:** The opcodes for the simulator have been defined in this class. An opcode is the portion of a machine language instruction that specifies the operation to be performed. The opcode is followed by zero or more operands, which provide additional information about the operation to be performed. The OpCodes.java class contains a list of opcodes and their corresponding octal values. These opcodes are used by the simulator to execute instructions. Each opcode corresponds to a specific operation that the simulator can perform.

Design Layout:


GPR 0
0000000000000000
LD
GPR 1
0000000000000000
LD
GPR 2
0000000000000000
LD
GPR 3
0000000000000000
LD
IXR 1
0000000000000000
LD
IXR 2
0000000000000000
LD
IXR 3
0000000000000000
LD
PC
000000000000
LD
MAR
000000000000
LD
MBR
0000000000000000
LD
IR
0000000000000000
MFR
0000
Priviledged
0
CC
0000
LD
Load
Store
INIT
SS
Run
Clear
InputFile
Input
Keyboard Console
Printer Display